System i

Programmazione di IBM Developer Kit per Java

Versione 6 Release 1

System i

Programmazione di IBM Developer Kit per Java

Versione 6 Release 1

Nota

Prima di utilizzare queste informazioni ed il prodotto da esse supportato, leggere le informazioni contenute in “Informazioni particolari”, a pagina 581.

Questa edizione si applica alla versione 6, release 1, livello di modifica 0 di IBM Developer Kit per Java (numero prodotto 5761-JV1) e a tutti i successivi release e livelli di modifica, a meno che non diversamente indicato nelle nuove edizioni. Questa versione non viene eseguita su tutti i modelli RISC (reduced instruction set computer) né sui modelli CISC.

Indice

IBM Developer Kit per Java

IBM Developer Kit per Java è ottimizzato per l'utilizzo nell'ambiente System i5. Esso utilizza la compatibilità della programmazione Java e delle interfacce utente consentendo così di sviluppare applicazioni System i5.

IBM Developer Kit per Java consente di creare ed eseguire programmi Java su System i5. IBM Developer Kit per Java è un'implementazione compatibile di Sun Microsystems, Inc. Java Technology, si presume quindi che l'utente conosca la documentazione JDK (Java Development Kit). Per facilitare la gestione delle loro e delle nostre informazioni, forniamo dei collegamenti alle informazioni diSun Microsystems, Inc.

Se per qualsiasi motivo i nostri collegamenti alla documentazione di JDK (Java Development Kit) di Sun Microsystems, Inc. non dovessero funzionare, consultare la relativa documentazione di riferimento HTML per le informazioni necessarie. È possibile reperire tali informazioni sul World Wide Web all'indirizzo The Source for Java Technology java.sun.com.

Nota: per importanti informazioni legali, leggere la sezione "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

Novità nella V6R1

Leggere le informazioni sulle informazioni nuove o notevolmente modificate per la raccolta di argomenti di IBM Developer Kit per Java.

Rimozione del supporto Esecuzione diretta

Nella V6R1, l'elaborazione diretta non è più supportata. Questo significa che il livello di ottimizzazione per i programmi Java è ignorato e OPTIMIZE(*INTERPRET) viene utilizzato quando si crea un programma Java su V6R1. Per ulteriori dettagli, consultare i seguenti argomenti:

- "Compatibilità tra release" a pagina 33
- "Considerazioni sulle prestazioni Java" a pagina 412
- "Compilazione statica di Java" a pagina 415
- "Considerazioni sulle prestazioni della compilazione statica di Java" a pagina 415

Modifiche al supporto dell'autorizzazione adottata

Il supporto per l'autorizzazione adottata del profilo utente tramite i programmi Java è stato ritirato nella V6R1. Consultare "Modifiche all'autorizzazione adottata nella V6R1" a pagina 257 per determinare se le applicazioni stanno utilizzando l'autorizzazione adottata e per informazioni su come modificare le applicazioni per accogliere questa modifica.

Hardware JCE (Java Cryptography Extension)

L'implementazione IBMJCECAI5OS estende JCE (Java) e JCA (Java Cryptography Architecture) per aggiungere le funzioni necessarie per utilizzare la crittografia hardware tramite le interfacce IBM CCA

(Common Cryptographic Architecture). Consultare "Utilizzo della crittografia hardware" a pagina 272.

JVMDI (Java Virtual Machine Tool Interface)

La JVMTI è un'interfaccia per analizzare la JVM (Java virtual machine). La JVMTI sostituisce la JVMPI (Java Virtual Machine Profiler Interface) e la JVMDI (Java Virtual Machine Debugger Interface). Consultare "Java Virtual Machine Tool Interface" a pagina 418 per ulteriori dettagli.

Potenziamenti JSSE (Java Secure Socket Extension)

JSSE 6 è disponibile alla V6R1. Consultare "Utilizzo di Java Secure Socket Extension 6" a pagina 321 per ulteriori informazioni.

Nuovi comandi CL Java

Sono stati aggiunti vari comandi CL correlati a Java per V6R1:

- Se si sta utilizzando IBM Technology for Java Virtual Machine, è possibile utilizzare il comando CL WRKJVMJOB (Gestione lavori JVM) per raccogliere dati sulle prestazioni. Consultare "Utilizzo del comando Gestione lavori JVM" a pagina 419.
- Se si sta utilizzando IBM Technology for Java Virtual Machine, è possibile utilizzare il comando CL GENJVMDMP (Generazione dump JVM) per generare i dump di heap, sistema e Java. Consultare "Utilizzo del comando Generazione dump JVM" a pagina 443.
- Il comando PRTJVMJOB (Stampa lavoro JVM) consente di stampare le JVM (Java Virtual Machines) in esecuzione nei lavori attivi. Consultare "Comandi CL supportati da Java" a pagina 428.

JDBC 4.0

JDBC 4.0 è conforme a J2SE 6. Per le modifiche all'interfaccia MetaData in JDBC 4.0, consultare "Modifiche in JDBC 4.0" a pagina 67.

Aggiornamento alla valuta

I seguenti argomenti sono stati aggiornati per riflettere le opzioni supportate di 5761-JV1 alla V6R1:

- "Supporto per più opzioni 5761-JV1 LP" a pagina 6
- "Installazione di IBM Technology for Java Virtual Machine" a pagina 4
- "Elenco delle proprietà di sistema Java" a pagina 16

Modifiche varie

- IPv6 (Internet Protocol versione 6) è ora pienamente supportato sia dalla JVM IBM Developer Kit per Java Classic che da quella IBM Technology for Java. Per informazioni generali su IPv6, consultare l'argomento relativo a Internet Protocol versione 6 nell'argomento relativo alla configurazione di TCP/IP dell'Information Center.
- La configurazione del fuso orario è stata modificata in V6R1. Per ulteriori informazioni, consultare "Configurazione del fuso orario" a pagina 25.
- Sono state apportate varie modifiche alle informazioni "NAWT (Native Abstract Windowing Toolkit)" a pagina 247.
- Delle nuove informazioni di debug sono state aggiunte per la JVM IBM Technology for Java. Consultare "Debug di sistema per IBM Technology for Java" a pagina 431.

Come esaminare le novità o le modifiche

Per fornire assistenza all'utente nell'esaminare le modifiche tecniche effettuate, queste informazioni utilizzano:

- L'immagine  contrassegna dove iniziano le informazioni nuove o modificate.
- L'immagine  per contrassegnare la fine di informazioni nuove o modificate.

Per individuare ulteriori informazioni sulle novità o le modifiche in questo release, consultare Memorandum per gli utenti.

File PDF per IBM Developer Kit per Java

È possibile visualizzare e stampare un file PDF che contiene le presenti informazioni.

Per visualizzare o scaricare la versione PDF di questo documento, selezionare IBM Developer Kit per Java (circa 4585 KB).

Salvataggio dei file PDF

Per salvare un PDF sulla stazione di lavoro per la visualizzazione o la stampa:

1. Fare clic con il tasto destro del mouse sul collegamento PDF nel proprio browser.
2. Fare clic sull'opzione che consente di salvare il PDF localmente.
3. Portarsi sull'indirizzario in cui si desidera salvare il PDF.
4. Fare clic su **Salva**.

Scaricamento di Adobe Reader

Per visualizzare o stampare tali PDF, è necessario che sul sistema sia installato Adobe Reader. È possibile scaricare una copia gratuita dal sito Web Adobe (www.adobe.com/products/acrobat/readstep.html)  .

Installazione e configurazione del IBM Developer Kit per Java

Se non si è ancora utilizzato il IBM Developer Kit per Java, attenersi alla seguente procedura per installarlo, configurarlo ed esercitarsi eseguendo un semplice programma Java "Hello World".

"Novità nella V6R1" a pagina 1
Leggere le informazioni sulle informazioni nuove o notevolmente modificate per la raccolta di argomenti di IBM Developer Kit per Java.

"Personalizzazione di System i5 per il IBM Developer Kit per Java" a pagina 13
Dopo aver installato il IBM Developer Kit per Java sul server è possibile personalizzare il server.

"Scaricamento ed installazione dei pacchetti Java" a pagina 8
Utilizzare queste informazioni per scaricare, installare ed utilizzare i pacchetti Java in modo più efficace sulla piattaforma System i.

"Compatibilità tra release" a pagina 33
Nella V6R1, l'elaborazione diretta non è più supportata. Questa modifica ha delle implicazioni sia sul parametro OPTIMIZE che sul parametro ENBPFRCOL.

Installazione di IBM Developer Kit per Java

L'installazione di IBM Developer Kit per Java consente di creare ed eseguire i programmi Java sul sistema.

Il programma su licenza 5761-JV1 è fornito con i CD di sistema e quindi JV1 è preinstallato. Immettere il comando GO LICPGM (Gestione programmi su licenza) e selezionare l'opzione 10 (Visualizzazione). Se il programma su licenza non è contenuto nell'elenco, procedere come segue:

1. Immettere il comando GO LICPGM sulla riga comandi.
2. Selezionare l'opzione 11 (Installazione programma su licenza).

3. Selezionare l'opzione 1 (Installazione) per il programma su licenza (LP - licensed program) 5761-JV1 *BASE e selezionare l'opzione corrispondente al JDK (Java Development Kit) che si desidera installare. Se l'opzione che si desidera installare non è visualizzata nell'elenco, è possibile aggiungerla immettendo l'opzione 1 (Installazione) nel campo Opzione. Immettere 5761JV1 nel campo del programma su licenza e il proprio numero di opzione nel campo dell'opzione del prodotto.

   Nota: è possibile installare più di un'opzione alla volta.

Dopo avere installato IBM Developer Kit per Java sul server, è possibile scegliere di personalizzare il sistema.

Concetti correlati

"Personalizzazione di System i5 per il IBM Developer Kit per Java" a pagina 13
Dopo aver installato il IBM Developer Kit per Java sul server è possibile personalizzare il server.

Attività correlate

"Esecuzione del primo programma Java Hello World" a pagina 9
Quest'argomento sarà di ausilio nell'esecuzione del primo programma Java.

Installazione di IBM Technology for Java Virtual Machine

IBM Technology for Java Virtual Machine è disponibile sia nella versione a 32 bit che in quella a 64 bit. Utilizzare queste istruzioni per installare IBM Technology for Java Virtual Machine.

IBM Technology for Java Virtual Machine è incluso nel programma su licenza 5761-JV1. Il programma su licenza 5761-JV1 è fornito con i CD di sistema. Per accedere all'opzione IBM Technology for Java, attenersi alla seguente procedura:

1. Immettere il comando GO LICPGM (Gestione programmi su licenza) e selezionare l'opzione 10 (Visualizzazione)
2. Se il programma su licenza non è contenuto nell'elenco, procedere come segue:
   a. Immettere il comando GO LICPGM sulla riga comandi.
   b. Selezionare l'opzione 11 (Installazione programma su licenza).
   c. Selezionare l'opzione 1 (Installazione) per il programma su licenza (LP - licensed program) 5761-JV1 *BASE e selezionare l'opzione che si desidera installare.
3. Aggiungere l'appropriata variabile di ambiente. Su una riga comandi, immettere uno dei seguenti comandi:
   a. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit')`
   b. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk50/64bit')`
   c. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit')`
   d. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit')`

Se non si è certi di quale JVM si sta attualmente utilizzando, è possibile eseguire una verifica utilizzando i seguenti metodi. Se nel risultato viene visualizzato `IBM J9 VM`, si sta utilizzando IBM Technology for Java.

- Cercare nella registrazione lavori il lavoro che contiene la JVM. Ci sarà un messaggio che indica quale JVM si sta utilizzando.
- Come parte del comando Java che si sta utilizzando per eseguire l'applicazione, aggiungere `-showversion`. Verrà visualizzata una riga aggiuntiva che mostra la JVM che si sta utilizzando.
- Da qsh o qp2term, eseguire `java -version`.

Informazioni correlate

Dimensioni e release dei programmi su licenza

Considerazioni per l'utilizzo di IBM Technology for Java Virtual Machine:

Tenere presente le seguenti considerazioni prima di utilizzare IBM Technology for Java Virtual Machine.

Utilizzo dei metodi nativi con IBM Technology for Java

Se si desidera utilizzare IBM Technology for Java e si hanno dei programmi che utilizzano i metodi nativi, è necessario compilare questi programmi con la memoria teraspace abilitata. Poiché la memoria teraspace non è abilitata per impostazione predefinita, è probabile che occorrerà rieseguire la compilazione. Questo è necessario perché l'oggetto Java si trova nella memoria PASE i5/OS, che è associata sopra la memoria teraspace, e viene restituito un puntatore della memoria teraspace. Inoltre, le funzioni JNI, come ad esempio GetxxxArrayRegion, hanno un parametro ad un buffer dove vengono memorizzati i dati. Questo puntatore deve puntare alla memoria teraspace per abilitare la funzione JNI in i5/OS PASE per copiare i dati in questa memoria. Se non si è eseguita la compilazione del programma con la memoria teraspace abilitata e si prova ad eseguire il metodo nativo con IBM Technology for Java, si riceverà il messaggio di uscita `MCH4443 (Modello di memoria non valido per il programma di destinazione LOADLIB)`.

Autorizzazione adottata

L'autorizzazione adottata per i programmi Java non è supportata da IBM Technology for Java Virtual Machine.

File e messaggi di diagnostica

Quando i metodi nativi ILE rilevano dei problemi, nella registrazione lavori saranno presenti dei messaggi. Quando IBM Technology for Java Virtual Machine o i metodi nativi PASE rilevano dei problemi, eseguiranno il dump dei file di diagnostica nell'IFS. Esistono vari tipi di questi "file principali", compresi `core.*.dmp`, `javacore.*.txt`, `Snap*.trc` e `heapdump.*.phd`. Questi file hanno una dimensione che può andare da qualche decina di KB a centinaia di MB. Nella maggior parte dei casi, dei problemi più gravi producono dei file più grandi. I file più grandi possono consumare in modo rapido e non immediatamente evidente grosse quantità di spazio IFS. Nonostante lo spazio che consumano, questi file sono utili per eseguire il debug. Quando è possibile, conservare questi file finché il problema cui fanno riferimento non sarà stato risolto.

Per ulteriori informazioni, consultare Advanced control of dump agents nel manuale Java Diagnostics Guide.

Considerazioni sulla migrazione

Prima di eseguire la migrazione dalla JVM Classic, che è una macchina virtuale a 64 bit, alla versione a 32 bit di IBM Technology for Java, considerare che ci possono essere delle limitazioni quando si utilizza l'ambiente a 32 bit. La quantità di memoria indirizzabile, ad esempio, è molto più piccola. Nella modalità a 32 bit, l'heap di oggetti Java non può crescere ad una dimensione molto superiore ai 3 gigabyte. Sarà anche possibile eseguire un massimo di circa 1000 sottoprocessi. Per ulteriori informazioni, consultare "Supporto per più opzioni 5761-JV1 LP" a pagina 6.

Concetti correlati

"Modifiche all'autorizzazione adottata nella V6R1" a pagina 257
Il supporto per l'autorizzazione adottata del profilo utente tramite i programmi Java è stato ritirato nella V6R1. Quest'argomento descrive come determinare se le applicazioni stanno utilizzando l'autorizzazione adottata e come modificare le applicazioni per accogliere questa modifica.

Installazione di un programma su licenza con il comando Ripristino programma su licenza

I programmi elencati nel pannello *Installazione programmi su licenza* sono quelli supportati dall'installazione LICPGM quando il proprio server era nuovo. A volte, diventano disponibili nuovi programmi che non sono elencati come programmi su licenza sul proprio server. Se questo avviene con il programma che si desidera installare, è necessario utilizzare il comando RSTLICPGM (Ripristino programma su licenza) per installarlo.

Per installare un programma su licenza con il comando RSTLICPGM (Ripristino programma su licenza), seguire queste fasi:

1. Inserire il nastro o il CD-ROM contenente il programma su licenza nell'unità appropriata.
2. Sulla riga comandi i5/OS, immettere:

   RSTLICPGM

   e premere il tasto Invio.

   Viene visualizzato il pannello *RSTLICPGM (Ripristino programma su licenza)*.
3. Nel campo *Prodotto*, immettere il numero dell'ID del programma su licenza che si desidera installare.
4. Nel campo *Unità*, specificare la propria unità di installazione.

   Nota: se si sta effettuando l'installazione da un'unità nastro, l'ID dell'unità è in genere in formato**TAP*xx***, dove***xx*** è un numero, ad esempio**01**.
5. Conservare le impostazioni predefinite per gli altri parametri nel pannello *Ripristino programma su licenza*. Premere il tasto Invio.
6. Vengono visualizzati ulteriori parametri. Conservare anche queste impostazioni predefinite. Premere il tasto Invio. Il programma inizia l'installazione.

Quando il programma su licenza ha terminato l'installazione, viene nuovamente visualizzato il pannello *Ripristino programma su licenza*.

Supporto per più opzioni 5761-JV1 LP

La piattaforma System i5 supporta più versioni dei JDK (Java Development Kit) e di J2SE (Java 2 Platform, Standard Edition).

Nota: In questa documentazione, a seconda del contesto, il termine JDK fa riferimento a qualsiasi versione supportata del JDK o del J2SE (Java 2 Platform, Standard Edition). Generalmente, il contesto in cui opera JDK include un riferimento alla versione e al numero di release specifici.

System i5 supporta l'utilizzo di più JDK simultaneamente ma solo tramite più JVM (Java virtual machine). Una singola JVM (Java virtual machine) esegue un solo JDK specificato. È possibile eseguire una singola JVM (Java virtual machine) per lavoro.

Individuare il JDK che si sta utilizzando o che si desidera utilizzare e selezionare l'opzione corrispondente da installare. Consultare "Installazione di IBM Developer Kit per Java" a pagina 3 per installare più di un JDK alla volta.

Con JVM Classic, la proprietà di sistema java.version determina quale JDK eseguire. Quando una JVM (Java virtual machine) è attiva e in esecuzione, la modifica della proprietà di sistema java.version non ha alcun effetto. Se si sta utilizzando IBM Technology for Java, selezionare quale opzione 5761-JV1 eseguire (e pertanto quale JDK/modo bit) impostando la variabile di ambiente JAVA_HOME. Questo è diverso da JVM Classic, che può utilizzare la proprietà di sistema java.version come immissione.

La seguente tabella elenca le opzioni supportate per questo release.

Opzioni 5761-JV1	JAVA_HOME	java.version
Opzione 6 - Classic 1.4	/QIBM/ProdData/Java400/jdk14/	1.4
Opzione 7 - Classic 5.0	/QIBM/ProdData/Java400/jdk15/	1.5
Opzione 8 - IBM Technology for Java 5.0 32-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit	1.5
Opzione 9 - IBM Technology for Java 5.0 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/64bit	1.5
Opzione 10 - Classic 6	/QIBM/ProdData/Java400/jdk6	1.6

Opzioni 5761-JV1	JAVA_HOME	java.version
Opzione 11 - IBM Technology for Java 6 32-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit	1.6
Opzione 12 - IBM Technology for Java 6 64-bit	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit	1.6

Il JDK predefinito scelto in questo ambiente a più JDK dipende da quali opzioni 5761-JV1 sono installate. La tabella che segue fornisce alcuni esempi. Le seguenti informazioni sono valide solo per i JDK Classic. È possibile accedere ai JDK IBM Technology for Java solo impostando la variabile di ambiente JAVA_HOME oppure tramite una chiamata diretta di un binario completo.

Installare	Immettere	Risultato
Opzione 6 (1.4)	`java Hello`	Viene eseguita J2SDK, Standard Edition, versione 1.4.
Opzione 7 (1.5) e Opzione 6 (1.4)	`java Hello`	Viene eseguito J2SDK, Standard Edition, versione 1.5.

Nota: se si installa un solo JDK, esso sarà quello predefinito. Se si installano più JDK, l'ordine di precedenza di seguito riportato determinerà quello predefinito:

1. Opzione 7 - Classic 5.0
2. Opzione 10 - Classic 6
3. Opzione 6 - Classic 1.4
4. Opzione 8 - IBM Technology for Java 5.0 32-bit
5. Opzione 9 - IBM Technology for Java 5.0 64-bit
6. Opzione 11- IBM Technology for Java 6 a 32 bit
7. Opzione 12 - IBM Technology for Java 6 64-bit

Installazione di estensioni per IBM Developer Kit per Java

Le estensioni sono pacchetti di classi Java che è possibile utilizzare per estendere la funzionalità della piattaforma principale. Le estensioni vengono compresse in uno o più file ZIP o JAR e vengono caricate nella JVM (Java virtual machine) da un programma di caricamento classi di estensioni.

Il meccanismo di estensione consente alla JVM (Java virtual machine) di utilizzare le classi di estensioni nello stesso modo in cui la JVM utilizza le classi di sistema. Il meccanismo di estensione fornisce anche un modo per richiamare le estensioni da specifici URL (Uniform Resource Locator) quando non sono già installati in J2SE (Java 2 Platform, Standard Edition).

Alcuni file JAR per le estensioni sono forniti con i5/OS. Se si desidera installare una di queste estensioni immettere il seguente comando:

```
ADDLNK OBJ('/QIBM/ProdData/Java400/ext/extensionToInstall.jar')
     NEWLNK('/QIBM/UserData/Java400/ext/extensionToInstall.jar')
     LNKTYPE(*SYMBOLIC)
```

Dove

`extensionToInstall.jar`

è il nome del file ZIP o JAR che contiene l'estensione che si desidera installare.

Nota: è possibile inserire i file JAR di estensione non forniti dall'IBM nell'indirizzario /QIBM/UserData/Java400/ext.

Quando si crea un collegamento o si aggiunge un file ad un'estensione nell'indirizzario /QIBM/UserData/Java400/ext, l'elenco di file in cui il programma caricamento classi di estensioni effettua la ricerca viene modificato per *ogni JVM (Java virtual machine) in esecuzione sul server*. Se non si desidera influenzare i programmi caricamento classi di estensione per altre JVM (Java virtual machine) sul server, ma si desidera ancora creare un collegamento ad un'estensione o installare un'estensione non fornita da IBM con il server, attenersi alla seguente procedura:

1. Creare un indirizzario per installare le estensioni. Utilizzare il comando MKDIR (Creazione indirizzario) dalla riga comandi i5/OS o il comando `mkdir` da Qshell Interpreter.
2. Inserire il file JAR di estensione nell'indirizzario creato.
3. Aggiungere il nuovo indirizzario alla proprietà java.ext.dirs. È possibile aggiungere il nuovo indirizzario alla proprietà java.ext.dirs utilizzando il campo PROP del comando JAVA dalla riga comandi i5/OS.

Se il nome del nuovo indirizzario è /home/username/ext, il nome del file dell'estensione è extensionToInstall.jar e il nome del programma Java è Hello, pertanto i comandi immessi dovrebbero assumere la seguente forma:

```
MKDIR DIR('/home/username/ext')

CPY OBJ('/productA/extensionToInstall.jar') TODIR('/home/username/ext') o
copiare il file in /home/username/ext utilizzando FTP (file transfer protocol).

JAVA Hello PROP((java.ext.dirs '/home/username/ext'))
```

Scaricamento ed installazione dei pacchetti Java

Utilizzare queste informazioni per scaricare, installare ed utilizzare i pacchetti Java in modo più efficace sulla piattaforma System i.

Pacchetti con GUI (Graphical User Interface)

I programmi Java utilizzati con GUI (graphical user interface) richiedono l'utilizzo di un'unità di presentazione con capacità di visualizzazione grafica. Ad esempio, è possibile utilizzare un PC, una stazione di lavoro tecnica o un computer di rete. È possibile utilizzare NAWT (Native Abstract Windowing Toolkit) per fornire alle applicazioni e ai servlet Java la piena funzionalità grafica di AWT (Abstract Windowing Toolkit) J2SE (Java 2 Platform, Standard Edition). Per ulteriori informazioni, consultare NAWT (Native Abstract Windowing Toolkit).

Sensibilità al maiuscolo e minuscolo e IFS (Integrated File System)

L'IFS (Integrated File System) fornisce i file system, che sono sensibili al maiuscolo e al minuscolo e che non riguardano i nomi file. QOpenSys è un esempio di file system sensibile al maiuscolo e al minuscolo all'interno dell'IFS. Root, '/', è un esempio di file system non sensibile al maiuscolo e al minuscolo. Per ulteriori informazioni, consultare l'argomento IFS (Integrated file system).

Anche se è possibile localizzare un JAR o una classe in un file system non sensibile al maiuscolo e al minuscolo, Java è comunque un linguaggio sensibile al maiuscolo e al minuscolo. Mentre `wrklnk '/home/Hello.class'` e `wrklnk '/home/hello.class'` producono gli stessi risultati, `JAVA CLASS(Hello)` e `JAVA CLASS(hello)` richiamano classi diverse.

Gestione file ZIP e gestione file JAR

I file ZIP e i file JAR contengono una serie di classi Java. Quando si utilizza il comando CRTJVAPGM (Creazione programma Java) su uno di questi file, le classi vengono verificate e convertite in un formato macchina interno. È possibile trattare i file ZIP e i file JAR come ogni altro file di classe individuale. Quando un formato macchina interno è associato ad uno di questi file, esso rimane associato al file. Il formato macchina interno viene utilizzato in applicazioni successive al posto dei file di classe per

migliorare le prestazioni. Se non si è sicuri se un programma Java corretto è associato al file di classe o al file JAR, utilizzare il comando DSPJVAPGM (Visualizzazione programma Java) per visualizzare le informazioni sul programma Java sul server.

Nei precedenti release di IBM Developer Kit per Java, era necessario ricreare un programma Java se l'utente aveva modificato il file JAR o il file ZIP in qualsiasi modo, in quanto il programma Java collegato sarebbe diventato inutilizzabile. Questo non si verifica più. In molti casi, se si modifica un file JAR o un file ZIP, il programma Java è ancora valido e non è necessario ricrearlo. Se vengono effettuate modifiche parziali, come l'aggiornamento di un singolo file di classe all'interno di un file JAR, è necessario soltanto ricreare i file di classe interessati che si trovano all'interno del file JAR.

I programmi Java rimangono collegati al file JAR dopo la maggior parte delle modifiche più comuni a tale file. Ad esempio, tali programmi Java rimangono collegati al file JAR quando:

- Si modifica o si ricrea un file JAR utilizzando lo strumento jar.
- Si sostituisce un file JAR utilizzando il comando COPY 0S/400 o il programma di utilità `cp` Qshell.

Se si accede ad un file JAR nell'IFS (integrated file system) tramite System i Access per Windows o da un'unità associata su un PC (personal computer), questi programmi Java rimangono collegati al file JAR quando:

- Si trascina e si rilascia un altro file JAR nel file JAR dell'IFS esistente.
- Si modifica o si ricrea il file JAR dell'IFS utilizzando lo strumento jar.
- Si sostituisce il file JAR dell'IFS utilizzando il comando di copia PC.

Quando si modifica o si sostituisce un file JAR, il programma Java collegato ad esso non è più corrente.

Esiste un'eccezione in cui i programmi Java non rimangono collegati al file JAR. I programmi Java collegati vengono eliminati se si utilizza FTP (file transfer protocol) per sostituire il file JAR. Ad esempio, ciò si verifica se si utilizza il comando `put` di FTP per sostituire il file JAR.

Per ulteriori informazioni sulle caratteristiche delle prestazioni dei file JAR, consultare "Considerazioni sulle prestazioni Java" a pagina 412.

Framework delle estensioni Java

In J2SE, le estensioni sono pacchetti di classi Java che è possibile utilizzare per estendere la funzionalità della piattaforma principale. Un'estensione o applicazione viene compressa in uno o più file JAR. Il meccanismo di estensione consente alla JVM (Java virtual machine) di utilizzare le classi di estensioni nello stesso modo in cui la JVM utilizza le classi di sistema. Il meccanismo di estensione fornisce inoltre un modo per richiamare estensioni da specifici URL quando non siano già installate nel J2SE o nel Java 2 Runtime Environment, Standard Edition.

Per ulteriori informazioni sull'installazione di estensioni, consultare "Installazione di estensioni per IBM Developer Kit per Java" a pagina 7.

Esecuzione del primo programma Java Hello World

Quest'argomento sarà di ausilio nell'esecuzione del primo programma Java.

È possibile ottenere il programma Hello World Java e procedere all'esecuzione in uno dei seguenti modi:

1. È possibile eseguire in modo semplice il programma Hello World Java inviato con IBM Developer Kit per Java.

   Per eseguire il programma incluso, operare le seguenti fasi:

a. Controllare che IBM Developer Kit per Java sia installato immettendo il comando GO LICPGM (Gestione programmi su licenza). Successivamente selezionare l'opzione 10 (Programmi su licenza installati visualizzati). Verificare che il programma su licenza 5761-JV1 *BASE e almeno una delle opzioni siano elencati come installati.
b. Immettere `java Hello` sulla riga comandi del Menu principale i5/OS. Premere Invio per eseguire il programma Hello World Java.
c. Se IBM Developer Kit per Java è stato installato correttamente, `Hello World` viene visualizzato nel pannello Shell di Java. Premere F3 (Fine) o F12 (Fine) per ritornare al pannello di immissione comandi.
d. Se la classe Hello World non viene eseguita, effettuare un controllo per assicurarsi che l'installazione sia stata completata correttamente oppure consultare "Come ottenere il supporto per IBM Developer Kit per Java" a pagina 577 per ottenere informazioni sul servizio.

2. È possibile inoltre eseguire il proprio programma Hello di Java. Per ulteriori informazioni su come creare il proprio programma Hello Java, consultare "Creazione, compilazione ed esecuzione del programma Java Hello World" a pagina 11.

Associazione di un'unità di rete al server

Per associare un'unità di rete, attenersi alla seguente procedura.

1. Accertarsi che System i Access per Windows sia installato sul server e sulla stazione di lavoro. Per ulteriori informazioni su come installare e configurare System i Access per Windows, consultare l'argomento relativo all'installazione di System i Access per Windows. È necessario disporre di un collegamento configurato per il server prima di poter associare un'unità di rete.
2. Aprire Windows Explorer:
   a. Fare clic con il tasto destro del mouse sul pulsante **Avvia** sul pannello delle attività Windows.
   b. Fare clic su **Esplora** nel menu.
3. Selezionare **Connetti unità di rete** dal menu **Strumenti**.
4. Selezionare l'unità che si intende utilizzare per collegarsi al proprio server.
5. Immettere il nome percorso sul proprio server. Ad esempio, `\\`*`MYSERVER`* dove *MYSERVER* è il nome del server.
6. Controllare la casella **Riconnetti all'accesso** se è vuota.
7. Fare clic su **OK** per terminare.

L'unità connessa viene visualizzata nella sezione **Tutte le cartelle** di Windows Explorer.

Creazione di un indirizzario sul server

È necessario creare un indirizzario sul server dove è possibile salvare le applicazioni Java.

Informazioni correlate

Introduzione a System i Navigator

Creazione di un indirizzario utilizzando System i Navigator

Scegliere questa opzione se è installato System i Access per Windows. Se si intende utilizzare System i Navigator per compilare ed eseguire il programma Java, è necessario selezionare questa opzione per assicurare che il programma sia salvato nell'ubicazione corretta per eseguire queste operazioni.

per creare un indirizzario sul System i, attenersi alla seguente procedura.

1. Aprire System i Navigator.
2. Fare doppio clic sul nome del proprio server nella finestra **Connessioni** per collegarsi. Se il proprio server non è elencato nella finestra **Connessioni**, effettuare quanto segue per aggiungerlo:
   a. Fare clic su **File** → **Aggiungi connessione...**
   b. Immettere il nome del proprio server nel campo **Sistema**.

c. Fare clic su **Avanti**.
d. Se non è già stato immesso, immettere il proprio ID utente nel campo **Utilizzare l'ID utente predefinito, richiedere se necessario**.
e. Fare clic su **Avanti**.
f. Fare clic su **Verifica connessione**. Ciò conferma che è possibile collegarsi al server.
g. Fare clic su **Fine**.

3. Espandere la cartella sotto il collegamento che si desidera utilizzare. Localizzare una cartella denominata **File System**. Se non si trova questa cartella, non è stata selezionata l'opzione per installare i File System durante l'installazione di System i Navigator. È necessario installare l'opzione File Systems di System i Navigator selezionando **Start → Programmi → System i Access per Windows → Installazione selettiva**.
4. Espandere la cartella **File System** e localizzare la cartella **IFS (Integrated File System)**.
5. Espandere la cartella **IFS (Integrated File System)**, quindi espandere la cartella **Root**. Espandendo la cartella **Root** viene visualizzata la stessa struttura dell'esecuzione del comando WRKLNK ('/') sulla riga comandi i5/OS.
6. Fare clic con il tasto destro del mouse sulla cartella dove si desidera aggiungere un sottoindirizzario. Selezionare **Nuova cartella** e immettere il nome del sottoindirizzario che si intende creare.

Creazione di un indirizzario utilizzando la riga di immissione comandi

Utilizzare queste istruzioni per creare un indirizzario se sul proprio sistema non è installato System i Access per Windows.

Per creare un indirizzario sul server, attenersi alla seguente procedura.

1. Collegarsi al server.
2. Sulla riga comandi, immettere:

   ```
   CRTDIR DIR('/mydir')
   ```

   dove *mydir* è il nome dell'indirizzario che si sta creando.

   Premere il tasto **Invio**.

   Viene visualizzato un messaggio in basso al proprio schermo, che dichiara "**Indirizzario creato.**"

Creazione, compilazione ed esecuzione del programma Java Hello World

La creazione del semplice programma Java Hello World è un ottimo punto di partenza per iniziare ad acquisire dimestichezza con IBM Developer Kit per Java.

Per creare, compilare ed eseguire il proprio programma Hello World Java, effettuare quanto segue:

1. Mappare un'unità di rete al sistema.
2. Creare un indirizzario sul server per le applicazioni Java.
3. Creare il file di origine come file di testo ASCII (American Standard Code Information Interchange) nell'IFS (Integrated File System). È possibile utilizzare un prodotto IDE (Integrated development environment) o un editor di testo come Notepad Windows per scrivere il codice dell'applicazione Java.
   a. Denominare il proprio file di testo `HelloWorld.java`.
   b. Assicurarsi che il proprio file contenga il seguente codice sorgente:

      ```
      class HelloWorld {
           public static void main (String args[]) {
                System.out.println("Hello World");
           }
      }
      ```

4. Compilare il file di origine.
   a. Immettere il comando STRQSH (Avvio Qshell) per avviare Qshell Interpreter.

b. Utilizzare il comando cd (Modifica indirizzario) per modificare l'indirizzario corrente nell'indirizzario dell'IFS (Integrated File System) che contiene il file `HelloWorld.java`.
c. Immettere `javac` seguito dal nome del file così come è stato salvato sul proprio disco. Ad esempio, immettere `javac HelloWorld.java`.

5. Impostare le autorizzazioni al file sul file di classe nell'IFS (integrated file system).
6. Eseguire il file di classe.
   a. Assicurarsi che il classpath Java sia impostato correttamente.
   b. Sulla riga comandi Qshell, immettere `java` seguito da `HelloWorld` per eseguire il proprio `HelloWorld.class` con JVM (Java virtual machine). Ad esempio, immettere `java HelloWorld`. È anche possibile utilizzare il comando RUNJVA (Esecuzione Java) sul sistema per eseguire `HelloWorld.class`: RUNJVA CLASS(HelloWorld)
   c. `"Hello World"` viene visualizzato sullo schermo se i comandi sono stati immessi correttamente. In caso di esecuzione nell'ambiente Qshell, viene visualizzata la richiesta shell (per impostazione predefinita, un $) indicando che la Qshell è pronta per un altro comando.
   d. Premere F3 (Fine) o F12 (Scollegamento) per ritornare al pannello di immissione comandi.

È anche possibile compilare ed eseguire facilmente l'applicazione Java utilizzando System i Navigator, una GUI (graphical user interface) per eseguire le attività sul sistema.

"Associazione di un'unità di rete al server" a pagina 10
Per associare un'unità di rete, attenersi alla seguente procedura.

"Creazione di un indirizzario sul server" a pagina 10
È necessario creare un indirizzario sul server dove è possibile salvare le applicazioni Java.

"Creazione e modifica dei file di origine Java"
È possibile creare e modificare i file di origine Java in vari modi: utilizzando System i Access per Windows, su una stazione di lavoro, con EDTF e con SEU.

"Classpath Java" a pagina 13
La JVM (Java virtual machine) utilizza il classpath Java per trovare le classi durante il tempo di esecuzione. I comandi e gli strumenti Java utilizzano inoltre il classpath per localizzare le classi. Il classpath di sistema predefinito, la variabile di ambiente CLASSPATH e il parametro del comando classpath determinano in quali indirizzari viene ricercata una specifica classe.

"Autorizzazioni file Java nell'IFS (integrated file system)." a pagina 246
Per eseguire o effettuare il debug di un programma Java è necessario che il file di classe, il file JAR o il file ZIP dispongano dell'autorizzazione alla lettura (*R). Gli indirizzari devono disporre delle autorizzazioni alla lettura e all'esecuzione (*RX).

Comando CRTJVAPGM (Creazione programma Java)

Comando RUNJVA (Esecuzione Java)

"Comandi System i Navigator supportati da Java" a pagina 429
System i Navigator è un'interfaccia grafica per il desktop Windows. È parte di System i Access per Windows e copre molte funzioni di i5/OS di cui i responsabili o gli utenti hanno bisogno per svolgere il loro lavoro giornaliero. È possibile utilizzare i comandi System i Navigator per creare ed eseguire i programmi Java.

Introduzione a System i Navigator

Creazione e modifica dei file di origine Java

È possibile creare e modificare i file di origine Java in vari modi: utilizzando System i Access per Windows, su una stazione di lavoro, con EDTF e con SEU.

Con System i Access per Windows

I file di origine Java sono file di testo ASCII (American Standard Code for Information Interchange) nell'IFS (integrated file system).

È possibile creare e modificare un file di origine Java con System i Access per Windows e un editor basato sulla stazione di lavoro.

Su una stazione di lavoro

È possibile creare un file di origine Java su una stazione di lavoro. Quindi, trasferire il file nell'IFS (Integrated File System) utilizzando FTP (file transfer protocol).

Per creare e modificare i file di origine Java su una stazione di lavoro:

1. Creare il file ASCII su una stazione di lavoro utilizzando l'editor scelto.
2. Collegarsi al server con FTP.
3. Trasferire il file di origine al proprio indirizzario nell'IFS (Integrated File System) come file binario, in modo che il file rimanga in formato ASCII.

Con EDTF

È possibile modificare i file da qualsiasi file system utilizzando il comando EDTF (Modifica file). Questo è un editor simile al SEU (Source Entry Utility) per modificare i file del flusso o i file del database. Per ulteriori informazioni, consultare il comando CL EDTF (Modifica file).

Con SEU (Source Entry Utility)

È possibile creare un file di origine Java come file di testo utilizzando SEU (source entry utility).

Per creare un file di origine Java come file di testo utilizzando SEU, effettuare quanto segue:

1. Creare un membro del file di origine utilizzando SEU.
2. Utilizzare il comando CPYTOSTMF (Copia nel file di flusso) per copiare il membro del file di origine in un file di flusso dell'IFS (Integrated file system), durante la conversione dei dati in ASCII.

Se è necessario effettuare delle modifiche al codice sorgente, modificare il membro del database utilizzando SEU e copiare nuovamente il file.

Per ulteriori informazioni sulla memorizzazione dei file, consultare "File correlati a Java nell'IFS" a pagina 246.

Personalizzazione di System i5 per il IBM Developer Kit per Java

Dopo aver installato il IBM Developer Kit per Java sul server è possibile personalizzare il server.

Classpath Java

La JVM (Java virtual machine) utilizza il classpath Java per trovare le classi durante il tempo di esecuzione. I comandi e gli strumenti Java utilizzano inoltre il classpath per localizzare le classi. Il classpath di sistema predefinito, la variabile di ambiente CLASSPATH e il parametro del comando classpath determinano in quali indirizzari viene ricercata una specifica classe.

In J2SE (Java 2 Platform, Standard Edition), la proprietà java.ext.dirs determina il classpath per le estensioni caricate. Consultare "Installazione di estensioni per IBM Developer Kit per Java" a pagina 7 per ulteriori informazioni.

Il classpath del bootstrap predefinito è definito dal sistema e non deve essere modificato. Sul server dell'utente il classpath del bootstrap predefinito specifica dove trovare le classi che fanno parte di IBM Developer Kit per Java, NAWT (Native Abstract Window Toolkit) e altre classi di sistema.

Per trovare una qualsiasi altra classe sul sistema, è necessario specificare il classpath da ricercare utilizzando la variabile di ambiente CLASSPATH o il parametro classpath. Il parametro classpath che viene utilizzato su uno strumento o comando sostituisce il valore specificato nella variabile di ambiente CLASSPATH.

È possibile gestire la variabile di ambiente CLASSPATH utilizzando il comando WRKENVVAR (Gestione variabile di ambiente). Dal pannello WRKENVVAR è possibile aggiungere o modificare la variabile di ambiente CLASSPATH. Il comando ADDENVVAR (Aggiunta variabile di ambiente) e il comando CHGENVVAR (Modifica variabile di ambiente) aggiungono o modificano la variabile di ambiente CLASSPATH.

Il valore della variabile di ambiente CLASSPATH è un elenco di nomi di percorso separati da due punti (:), nei quali viene effettuata la ricerca di una classe specifica. Un nome percorso è una sequenza di zero o più nomi indirizzario. Tali nomi sono seguiti dal nome dell'indirizzario, dal file ZIP o dal file JAR da ricercare nell'IFS (Integrated File System). I componenti del nome percorso sono separati dal carattere barra (/). Utilizzare un punto (.) per indicare l'indirizzario di lavoro corrente.

È possibile impostare la variabile CLASSPATH nell'ambiente Qshell utilizzando il programma di utilità di esportazione disponibile tramite Qshell Interpreter.

Questi comandi aggiungono la variabile CLASSPATH al proprio ambiente Qshell e la impostano sul valore `".:/myclasses.zip:/Product/classes"`

- Questo comando imposta la variabile CLASSPATH nell'ambiente Qshell:

  ```
  export -s CLASSPATH=.:/myclasses.zip:/Product/classes
  ```

- Questo comando imposta la variabile CLASSPATH dalla riga comandi:

  ```
  ADDENVVAR ENVVAR(CLASSPATH) VALUE(".:/myclasses.zip:/Product/classes")
  ```

J2SE ricerca prima il classpath del bootstrap, quindi gli indirizzari dell'estensione e infine il classpath. Sulla base dell'esempio precedentemente riportato, l'ordine di ricerca per J2SE è il seguente:

1. Il classpath del bootstrap ubicato nella proprietà sun.boot.class.path,
2. Gli indirizzari dell'estensione ubicati nella proprietà java.ext.dirs,
3. L'indirizzario di lavoro corrente,
4. Il file myclasses.zip ubicato nel file system ″root″ (/),
5. L'indirizzario delle classi nell'indirizzario Prodotto nel file system ″root″ (/).

Quando si accede all'ambiente Qshell, la variabile CLASSPATH è impostata sulla variabile di ambiente. Il parametro classpath specifica un elenco di nomi di percorso. Esso ha la stessa sintassi della variabile di ambiente CLASSPATH. Un parametro del classpath è disponibile su questi strumenti e comandi:

- comando `java` in Qshell
- strumento `javac`
- strumento `javah`
- strumento `javap`
- strumento `javadoc`
- strumento `rmic`
- Eseguire il comando RUNJVA (Esecuzione Java)

Per ulteriori informazioni su questi comandi, consultare "Comandi e strumenti per il IBM Developer Kit per Java" a pagina 420. Se si utilizza il parametro del classpath con uno qualsiasi di questi comandi o strumenti, esso ignora la variabile di ambiente CLASSPATH.

È possibile sostituire la variabile di ambiente CLASSPATH utilizzando la proprietà java.class.path. È possibile modificare sia la proprietà java.class.path sia le altre proprietà, utilizzando il file

SystemDefault.properties. I valori nei file SystemDefault.properties sostituiscono la variabile di ambiente CLASSPATH. Per ulteriori informazioni sul file SystemDefault.properties, consultare "File SystemDefault.properties".

In J2SE l'opzione -Xbootclasspath influisce anche sugli indirizzari nei quali viene eseguita la ricerca quando il sistema ricerca le classi. Utilizzando -Xbootclasspath/a:*path* viene accodato *path* al classpath del bootstrap predefinito, /p:*path* viene anteposto *path* al classpath del bootstrap e :*path* viene sostituito il classpath del bootstrap con *path*.

Nota: nello specificare -Xbootclasspath occorre prestare attenzione in quanto se è impossibile trovare una classe di sistema o se questa viene sostituita in modo non corretto da una classe definita dall'utente si verificano risultati imprevisti. Pertanto è opportuno fare in modo che la ricerca venga eseguita prima nel classpath predefinito di sistema e poi in un classpath definito dall'utente.

Consultare "Proprietà di sistema Java" per informazioni su come determinare l'ambiente in cui vengono eseguiti i programmi Java.

Per ulteriori informazioni, consultare le API del comando CL e del programma o l'IFS (Integrated file system).

Proprietà di sistema Java

Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

L'avvio di un'istanza di una JVM (Java virtual machine) imposta i valori per le proprietà di sistema che influenzano tale JVM.

È possibile scegliere di utilizzare i valori predefiniti per le proprietà di sistema Java o è possibile specificare dei valori utilizzando i seguenti metodi:

- aggiungendo i parametri alla riga comandi o all'API di richiamo JNI (Java Native Interface) quando si avvia il programma Java
- utilizzando la variabile di ambiente al livello lavoro QIBM_JAVA_PROPERTIES_FILE per puntare a uno specifico file delle proprietà. Ad esempio:

```
ADDENVVAR ENVVAR(QIBM_JAVA_PROPERTIES_FILE)
  VALUE(/QIBM/userdata/java400/mySystem.properties)
```

- Creando un file SystemDefault.properties nell'indirizzario user.home
- Utilizzo dei file /QIBM/userdata/java400/SystemDefault.properties

i5/OS e la JVM determinano i valori per le proprietà di sistema Java con il seguente ordine di precedenza:

1. Riga comandi o API di richiamo JNI
2. Variabile di ambiente QIBM_JAVA_PROPERTIES_FILE
3. File user.home SystemDefault.properties
4. /QIBM/UserData/Java400/SystemDefault.properties
5. Valori proprietà di sistema predefiniti

File SystemDefault.properties

Il file SystemDefault.properties è un file delle proprietà Java standard che consente di specificare le proprietà predefinite dell'ambiente Java.

Il file SystemDefault.properties che si trova nell'indirizzario principale ha priorità sul file SystemDefault.properties che si trova nell'indirizzario /QIBM/UserData/Java400.

Le proprietà impostate nel file /YourUserHome/SystemDefault.properties interessano solo le seguenti JVM (Java virtual machine) specifiche:

- Le JVM avviate senza specificare una proprietà user.home differente
- Le JVM avviate da altri utenti specificando la proprietà `user.home = /YourUserHome/`

Esempio: file SystemDefault.properties

L'esempio che segue imposta diverse proprietà Java:

```
#I commenti sono contrassegnati dal simbolo cancelletto
#Utilizzare J2SE 1.5
java.version=1.5
#Ciò imposta una proprietà speciale
myown.propname=6
```

Elenco delle proprietà di sistema Java

Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

L'avvio di un'istanza di una JVM (Java virtual machine) imposta le proprietà di sistema per tale istanza della JVM. Per ulteriori informazioni su come specificare i valori per le proprietà di sistema Java, consultare le seguenti pagine:

- "Proprietà di sistema Java" a pagina 15
- "File SystemDefault.properties" a pagina 15

Per ulteriori informazioni sulle proprietà di sistema Java, consultare "Proprietà di sistema Java JSSE 1.4" a pagina 286 e "Proprietà di sistema Java JSSE 1.5" a pagina 303.

La seguente tabella elenca le proprietà di sistema Java per le opzioni 5761-JV1 Classic supportate.

La tabella elenca, per ciascuna proprietà, il nome e i valori predefiniti che vengono applicati o una breve descrizione. La tabella indica quali proprietà di sistema hanno dei valori differenti nelle versioni differenti di J2SE (Java 2 Platform, Standard Edition). Quando la colonna che elenca i valori predefiniti non indica versioni differenti di J2SE, tutte le versioni supportate di J2SE utilizzano il valore predefinito.

awt.toolkit	sun.awt.motif.MToolkit l'impostazione di awt.toolkit verrà annullata a meno che os400.awt.native=true o java.awt.headless=true
file.encoding	ISO8859_1 (valore predefinito) Mette in corrispondenza il CCSID (coded character set identifier) con il corrispondente CCSID ASCII ISO. Inoltre, imposta il valore file.encoding sul valore Java che rappresenta il CCSID ASCII ISO. Consultare "Valori file.encoding e CCSID System i5" a pagina 26 per una tabella che mostra la relazione tra i possibili valori file.encoding e il CCSID più strettamente corrispondente.
file.encoding.pkg	sun.io
file.separator	/ (barra)
i5os.crypto.device	Specifica l'unità crittografica da utilizzare. Se questa proprietà non è impostata, viene utilizzata l'unità predefinita CRP01.
i5os.crypto.keystore	Specifica il file memorizzazione chiave CCA da utilizzare. Se questa proprietà non è impostata, viene utilizzato il file memorizzazione chiave indicato nella descrizione dell'unità crittografica.

java.awt.headless	• J2SE v1.4: false • J2SE 5.0: false (valore predefinito) • JDK 6: false Questa proprietà specifica se l'API AWT (Abstract Windowing Toolkit) opera in modalità headless (senza server) o meno. Il valore predefinito false rende disponibile la funzione AWT completa solo se è stato abilitato AWT impostando os400.awt.native su true. L'impostazione di questa proprietà su true supporta la modalità AWT headless (senza server) e forza esplicitamente os400.awt.native su true.
java.class.path	. (punto) (valore predefinito) Indica il percorso utilizzato da i5/OS per individuare le classi. Viene impostato sul valore predefinito CLASSPATH specificato dall'utente.
java.class.version	• J2SE v1.4: 48.0 • J2SE 5.0: 49.0 • JDK 6: 50.0
java.compiler	• jitc (default) - Specifica che si compilerà il codice utilizzando il compilatore JIT (Just-In-Time) (jitc). • jitc_de - Questo valore è conservato per ragioni di compatibilità con i release precedenti. L'elaborazione diretta non è supportata in V6R1, quindi questo valore darà come risultato lo stesso comportamento di tempo di esecuzione del valore di proprietà jitc. • NONE - Specifica che tutto il codice verrà eseguito utilizzando l'interpreter bytecode.
java.ext.dirs	J2SE v1.4: • /QIBM/ProdData/OS400/Java400/jdk/lib/ext: • /QIBM/ProdData/Java400/jdk14/lib/ext: • /QIBM/UserData/Java400/ext J2SE 5.0: (valore predefinito) • /QIBM/ProdData/Java400/jdk15/lib/ext: • /QIBM/UserData/Java400/ext JDK 6: • /QIBM/ProdData/Java400/jdk6/lib/ext • /QIBM/UserData/Java400/ext
java.home	J2SE v1.4: /QIBM/ProdData/Java400/jdk14 J2SE v1.5: /QIBM/ProdData/Java400/jdk15 (valore predefinito) JDK 6: /QIBM/ProdData/Java400/jdk6 Questa proprietà viene utilizzata solo per l'emissione. Consultare "Supporto per più opzioni 5761-JV1 LP" a pagina 6 per ulteriori dettagli.
java.library.path	elenco librerie i5/OS

java.net.preferIPv4Stack	• false (no) - valore predefinito • true Su macchine a doppio stack le proprietà di sistema vengono fornite per impostare lo stack protocollo preferito (IPv4 o IPv6) e i tipi di famiglia di indirizzi preferiti (inet4 o inet6). Lo stack IPv6 viene preferito per impostazione predefinita, dato che su una macchina a doppio stack il socket IPv6 può comunicare con entrambi i peer IPv4 e IPv6. Questa impostazione può essere modificata attraverso questa proprietà. Per ulteriori informazioni, consultare il manuale Networking IPv6 User Guide.
java.net.preferIPv6Addresses	• true • false (no) (valore predefinito) Sebbene IPv6 sia disponibile sul sistema operativo, la preferenza predefinita consiste nel preferire un indirizzo definito su IPv4 a un indirizzo IPv6. Questa proprietà controlla se vengono utilizzati gli indirizzi IPv6 (true) o IPv4 (false). Per ulteriori informazioni, consultare il manuale Networking IPv6 User Guide.
java.policy	J2SE v1.4: /QIBM/ProdData/OS400/ Java400/jdk/lib/security/java.policy J2SE v5.0: /QIBM/ProdData/ Java400/jdk15/lib/security/java.policy (default value) JDK 6: /QIBM/ProdData/Java400/jdk6/lib/security/java.policy
java.specification.name	Specifica API piattaforma Java (valore predefinito)
java.specification.vendor	Sun Microsystems, Inc.
java.specification.version	• J2SE v1.4: 1.4 • J2SE v5.0: 1.5 (valore predefinito) • JDK 6: 1.6
java.use.policy	true
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
java.version	• 1.4.2 • 1.5.0 (valore predefinito) • 1.6.0 Determina la versione di J2SE che si desidera eseguire. Questa proprietà è valida solo quando si utilizza la JVM Classic. Quando si utilizza IBM Technology for Java, la proprietà ″java.version″ viene ignorata; la versione di JDK viene determinata dal valore della variabile di ambiente JAVA_HOME. Se si installa una singola versione di J2SE, tale versione sarà quella predefinita. Se si specifica una versione che non è installata verrà visualizzato un messaggio di errore. Se la specifica della versione ha esito negativo viene utilizzata come predefinita la versione più recente di J2SE.
java.vm.name	VM classica
java.vm.specification.name	Specifica della JVM (Java Virtual Machine)
java.vm.specification.vendor	Sun Microsystems, Inc.
java.vm.specification.version	1.0
java.vm.vendor	IBM Corporation

java.vm.version	• J2SE v1.4: 1.4 • J2SE v5.0: 1.5 (valore predefinito) • JDK 6: 1.6
line.separator	\n
os.arch	PowerPC
os.name	OS/400
os.version	V6R1M0 (valore predefinito) Ottiene il livello di release i5/OS dall'API (application programm interface) di richiamo delle informazioni sul prodotto.
os400.awt.native	Controlla se l'API AWT (Abstract Windowing Toolkit) è supportata o meno. I valori validi sono true e false. Il valore predefinito è false a meno che non sia stato impostato java.awt.headless=true, in questo caso si presume che os400.awt.native sia true.
os400.certificateContainer	Indirizza il supporto SSL (secure socket layer) Java in modo tale che utilizzi la memorizzazione certificati specificata per il programma Java avviato e la proprietà specificata. Se si specifica la proprietà di sistema os400.secureApplication, questa proprietà di sistema viene ignorata. Ad esempio, immettere `-Dos400.certificateContainer=/home/username/mykeyfile.kdb` o qualsiasi altro keyfile nell'IFS (integrated file system).
os400.certificateLabel	È possibile specificare questa proprietà di sistema insieme alla proprietà di sistema os400.certificateContainer. Questa proprietà permette la selezione del certificato presente nel contenitore specificato che si desidera che SSL (secure socket layer) utilizzi. Per esempio, immettere `-Dos400.certificateLabel=myCert`, dove myCert rappresenta il nome dell'etichetta assegnata al certificato tramite il DCM (Digital Certificate Manager) quando il certificato viene creato o importato.
os400.child.stdio.convert	Controlla la conversione dati per stdin, stdout e stderr in Java. La conversione dei dati tra i dati ASCII e EBCDIC (Extended Binary Coded Decimal Interchange Code) si verifica per impostazione predefinita nella JVM (Java virtual machine). L'uso di questa proprietà per attivare e disattivare queste conversioni influenza solo i processi secondari avviati da questo processo tramite il metodo Runtime.exec(). Questo valore di proprietà diventa il valore predefinito per os400.stdio.convert nei processi secondari. Consultare "Valori per le proprietà di sistema os400.stdio.convert e os400.child.stdio.convert." a pagina 23.
os400.class.path.security.check	20 (valore predefinito) Valori validi: • 0 Nessun controllo di sicurezza • 10 Equivalente a RUNJVA CHKPATH(*IGNORE) • 20 Equivalente a RUNJVA CHKPATH(*WARN) • 30 Equivalente a RUNJVA CHKPATH(*SECURE)

os400.class.path.tools	0 (valore predefinito) Valori validi: • 0: nessuno strumento Sun nella proprietà java.class.path • 1: antepone il file degli strumenti specifici di J2SE alla proprietà java.class.path Per J2SE v1.4, il percorso a tools.jar è: /QIBM/ProdData/OS400/Java400/jdk/lib/ Per J2SE v5.0, il percorso a tools.jar è: /QIBM/ProdData/Java400/jdk15/lib/ JDK 6, il percorso a tools.jar è: /QIBM/ProdData/Java400/jdk6/lib/
os400.create.type	I valori validi sono: • interpret • direct Entrambi i valori sono ignorati. Se è necessario creare un programma Java, esso verrà creato senza il codice di esecuzione diretto.
os400.define.class.cache.file	valore predefinito è /QIBM/ProdData/Java400/QDefineClassCache.jar Specifica il nome di un file JAR o ZIP. Consultare ″Utilizzo della cache per i programmi di caricamento classe utente″ all'interno di Considerazioni sulle prestazioni Java.
os400.define.class.cache.hour	• valore predefinito = 768 • massimo valore decimale = 9999 Specifica un valore decimale. Consultare ″Utilizzo della cache per i programmi di caricamento classe utente″ all'interno di Considerazioni sulle prestazioni Java.
os400.define.class.cache.maxpgms	• valore predefinito = 20000 • massimo valore decimale = 40000 Specifica un valore decimale. Consultare ″Utilizzo della cache per i programmi di caricamento classe utente″ all'interno di Considerazioni sulle prestazioni Java.
os400.defineClass.optLevel	In V6R1, tutti i valori vengono ignorati. Se è necessario creare un programma Java, esso verrà creato senza il codice di esecuzione diretto.
os400.display.properties	Se questo valore è impostato su 'true', tutte le proprietà della JVM (Java Virtual Machine) vengono stampate in emissione standard. Non vengono riconosciuti altri valori.
os400.enbpfrcol	Questa proprietà indica al compilatore JIT (just-in-time) di generare gli hook di raccolta dei dati relativi alle prestazioni nel codice generato JIT. I valori validi sono: • 0 - valore predefinito. Nessun dato relativo alle prestazioni può essere raccolto dal codice compilato JIT. • 1 - il JIT genera gli eventi *JVAENTRY e *JVAEXIT. • 7 - equivalente ad un valore di 1.
os400.exception.trace	Questa proprietà viene utilizzata solo per l'esecuzione del debug. Specificando questa proprietà si verifica l'invio delle eccezioni più recenti all'emissione standard quando esiste la JVM.

os400.file.create.auth, os400.dir.create.auth	Queste proprietà specificano le autorizzazioni assegnate ai file e agli indirizzari. Specificando le proprietà senza alcun valore o con valori non supportati si determina un'autorizzazione pubblica di *NONE. È possibile specificare os400.file.create.auth=RWX o os400.dir.create.auth=RWX, dove R=lettura, W=scrittura e X=esecuzione. Qualsiasi combinazione di queste autorizzazioni è valida.
os400.file.io.mode	Converte il CCSID del file se questo risulta diverso rispetto al valore file.encoding quando si specifica TEXT, piuttosto che il valore predefinito, che è BINARY.
os400.gc.heap.size.init	Un'alternativa all'uso di -Xms (impostando la dimensione GC iniziale). Si consiglia di continuare ad utilizzare -Xms, a meno che non si è costretti a fare diversamente, poiché questa proprietà è specifica di i5/OS. Questa proprietà è stata introdotta principalmente per poter specificare la dimensione GC iniziale nel file SystemDefault.properties. **Nota:** utilizzare questa proprietà con attenzione; sovrascriverà -Xms se specificata. È necessario che il valore sia un numero intero in dimensione di kilobyte e senza virgole.
os400.gc.heap.size.max	Un'alternativa all'uso di -Xmx (impostando la dimensione GC massima). Si consiglia di continuare ad utilizzare -Xmx, a meno che non si è costretti a fare diversamente, poiché questa proprietà è specifica di i5/OS. Questa proprietà consente di specificare la dimensione massima GC nel file SystemDefault.properties. **Nota:** utilizzare questa proprietà con attenzione; sovrascriverà -Xmx se specificata. È necessario che il valore sia un numero intero in dimensione di kilobyte e senza virgole.
os400.interpret	• 0 (valore predefinito) equivalente a CRTJVAPGM INTERPRET(*NO) • 1 equivalente a CRTJVAPGM INTERPRET(*YES)
os400.jit.mmi.threshold	Imposta il numero di volte in cui viene eseguito un metodo utilizzando l'MMI (Mixed-Mode Interpreter), prima che i5/OS utilizzi il compilatore JIT per compilare il metodo in istruzioni native della macchina. Solitamente, non si deve modificare il valore predefinito, che è 2000. • Il valore zero disabilita l'MMI e compila i metodi quando vengono chiamati per la prima volta. • I valori inferiori a quello predefinito tendono ad allungare il tempo di avvio e a diminuire la qualità delle prestazioni finali. • I valori superiori a quello predefinito diminuiscono inizialmente la qualità delle prestazioni fino a raggiungere la soglia, ma, in seguito, migliorano le prestazioni finali al tempo di esecuzione.
os400.job.file.encoding	Questa proprietà viene utilizzata solo per l'emissione. Elenca la codifica file del lavoro i5/OS in cui si trova la JVM.
os400.optimization	I valori validi sono 0, 10, 20, 30 e 40. In V6R1, tutti i valori vengono ignorati. Se è necessario creare un programma Java, esso viene creato senza il codice di esecuzione diretto.
os400.pool.size	Definisce quanto spazio (in kilobyte) rendere disponibile per ogni lotto heap nell'heap locale del sottoprocesso.

os400.run.mode	• jitc (valore predefinito) Utilizzare il compilatore JIT per determinare in che modo viene generato ed eseguito il codice • jitc_de Uguale al valore predefinito di jitc: utilizzare il compilatore JIT per determinare in che modo viene generato ed eseguito il codice. • interpret Equivalente a RUNJVA OPTIMIZE(*INTERPRET) e INTERPRET(*OPTIMIZE) o INTERPRET(*YES) • tipo_creazione_programma Utilizzare il compilatore JIT per determinare in che modo viene generato ed eseguito il codice
os400.run.verbose	Se questo valore è impostato su 'true', come standard viene stampato un caricamento classe dettagliato. Non vengono riconosciuti altri valori. Viene raggiunto lo stesso risultato di quando si specifica -verbose in QSHELL o OPTION(*VERBOSE) nei comandi CL, tranne che questa proprietà opera nel file SystemDefault.properties.
os400.runtime.exec	• EXEC (valore predefinito) Richiama le funzioni attraverso runtime.exec() utilizzando l'interfaccia EXEC. • QSHELL Richiama le funzioni attraverso runtime.exec() utilizzando Qshell Interpreter. Per ulteriori informazioni, consultare la sezione "Utilizzo di java.lang.Runtime.exec()" a pagina 229.
os400.secureApplication	Associa il programma Java che si avvia quando si utilizza questa proprietà di sistema (os400.secureApplication) con il nome dell'applicazione protetta e registrata. È possibile visualizzare i nomi dell'applicazione protetta registrata utilizzando il DCM (Digital Certificate Manager).
os400.security.properties	Consente un controllo completo del file java.security utilizzato. Quando si specifica questa proprietà, il J2SE non utilizza altri file java.security, compreso quello predefinito java.security specifico per J2SE.
os400.stderr	Consente la correlazione di stderr a un file o un socket. Consultare "Valori della proprietà di sistema os400.stdin, os400.stdout e os400.stderr" a pagina 24.
os400.stdin	Consente la correlazione di stdin a un file o un socket. Consultare "Valori della proprietà di sistema os400.stdin, os400.stdout e os400.stderr" a pagina 24.
os400.stdin.allowed	1 (valore predefinito) Specifica se stdin è consentito (1) o se non è consentito (0). Se il chiamante sta eseguendo un lavoro batch, stdin non dovrebbe essere consentito.
os400.stdio.convert	Consente il controllo della conversione dati per stdin, stdout e stderr in Java. La conversione dei dati si verifica per impostazione predefinita nella JVM (Java virtual machine) per convertire i dati ASCII in o da EBCDIC. È possibile attivare o disattivare queste conversioni con questa proprietà, la quale influenza il programma Java corrente. Consultare "Valori per le proprietà di sistema os400.stdio.convert e os400.child.stdio.convert." a pagina 23. Per i programmi Java avviati utilizzando il metodo Runtime.exec(), consultare os400.child.stdio.convert.
os400.stdout	Consente la correlazione di stdout a un file o un socket. Consultare valori predefiniti.
os400.xrun.option	Questa proprietà di sistema permette di utilizzare l'opzione Qshell -Xrun specificando una proprietà. È possibile utilizzarla per specificare un programma agent da eseguire durante l'avvio della JVM.

os400.verify.checks.disable	65535 (valore predefinito) Questo valore della proprietà di sistema è una stringa che rappresenta la somma di uno o più valori numerici. Per un elenco di questi valori, consultare "Valori per la proprietà di sistema os400.verify.checks.disable" a pagina 24.
os400.vm.inputargs	Questa proprietà viene utilizzata solo per l'emissione. Visualizzerà gli argomenti che la JVM ha ricevuto come immissioni. Questa proprietà può essere utile per l'esecuzione del debug che è stato specificato all'avvio della JVM.
path.separator	: (due punti)
sun.boot.class.path	Elenca tutti i file necessari al programma di caricamento classe di avvio predefinito. Non modificare questo valore.
user.dir	L'indirizzario di lavoro corrente (CWD-Current working directory) che sta utilizzando l'API getcwd.
user.home	Richiama l'indirizzario di lavoro iniziale utilizzando l'API Get (getpwnam). È possibile posizionare un file SystemDefault.properties in un percorso user.home per sostituire le proprietà predefinite in /QIBM/UserData/Java400/SystemDefault.properties. È possibile personalizzare il sistema per specificare una propria serie di valori di proprietà predefiniti.
user.language	La JVM (Java virtual machine) utilizza questa proprietà di sistema per leggere il valore LANGID del lavoro e utilizza questo valore per rilevare il linguaggio corrispondente.
user.name	La JVM (Java virtual machine) utilizza questa proprietà di sistema per richiamare il nome del profilo utente valido dalla sezione di sicurezza (Security.UserName) di TBC (Trusted Computing Base).
user.region	La JVM (Java virtual machine) utilizza questa proprietà di sistema per leggere il valore CNTRYID del lavoro e utilizza questo valore per determinare la regione dell'utente.
user.timezone	• La JVM cerca prima l'oggetto QLOCALE di sistema. • Se non lo trova, la JVM cerca il valore di sistema QTIMZON. Il campo 'Nome alternativo' nell'oggetto QTIMZON viene utilizzato per assegnare la proprietà JVM Java user.timezone, Il valore nel campo 'Nome alternativo' deve avere una lunghezza di almeno 3 caratteri altrimenti non verrà utilizzato. • Se il campo 'Nome alternativo' nell'oggetto QTIMZON ha una lunghezza inferiore ai 3 caratteri, la JVM proverà a trovare un valore GMT corrispondente basato sull'offset di sistema corrente. Esempio: un oggetto QTIMZON con un campo Nome alternativo vuoto ed un offset di -5 dà come risultato l'impostazione `user.timezone=GMT-5`. • Se non è stato ancora trovato un valore, la JVM imposta automaticamente user.timezone sull'UTC (Universal Time Coordinate - tempo universale coordinato). Per ulteriori informazioni, consultare l'argomento relativo agli ID di fuso orario che è possibile specificare per la proprietà user.timezone nel WebSphere Software Information Center.

Concetti correlati

"Personalizzazione di System i5 per il IBM Developer Kit per Java" a pagina 13
Dopo aver installato il IBM Developer Kit per Java sul server è possibile personalizzare il server.

Valori per le proprietà di sistema os400.stdio.convert e os400.child.stdio.convert.:

Le seguenti tabelle mostrano i valori di sistema per le proprietà di sistema os400.stdio.convert e os400.child.stdio.convert.

Tabella 1. Valori di sistema per **os400.stdio.convert**

Valore	Descrizione
Y (valore predefinito)	Ogni stdio viene convertito nel e dal valore file.encoding al CCSID del lavoro durante la lettura o la scrittura.
N	Non viene eseguita alcuna conversione stdio durante la lettura o la scrittura.
1	Solo i dati stdin vengono convertiti dal CCSID del lavoro al file.encoding avviene la lettura.
2	Solo i dati stdout vengono convertiti dal file.encoding al CCSID del lavoro durante la scrittura.
3	Vengono eseguite sia conversioni stdin che stdout.
4	Solo i dati stderr vengono convertiti dal file.encoding al CCSID del lavoro durante la scrittura.
5	Vengono eseguite sia conversioni stdin che stderr.
6	Vengono eseguite sia le conversioni stdout che stderr.
7	Vengono eseguite tutte le conversioni stdio.

Tabella 2. Valori di sistema per **os400.child.stdio.convert**

Valore	Descrizione
N (valore predefinito)	Non viene eseguita alcuna conversione stdio durante la lettura o la scrittura.
Y	Ogni stdio viene convertito nel e dal valore file.encoding al CCSID del lavoro durante la lettura o la scrittura.
1	Solo i dati stdin vengono convertiti dal CCSID del lavoro al file.encoding avviene la lettura.
2	Solo i dati stdout vengono convertiti dal file.encoding al CCSID del lavoro durante la scrittura.
3	Vengono eseguite sia conversioni stdin che stdout.
4	Solo i dati stderr vengono convertiti dal file.encoding al CCSID del lavoro durante la scrittura.
5	Vengono eseguite sia conversioni stdin che stderr.
6	Vengono eseguite sia le conversioni stdout che stderr.
7	Vengono eseguite tutte le conversioni stdio.

Valori della proprietà di sistema os400.stdin, os400.stdout e os400.stderr:

La tabella seguente indica i valori di sistema per le proprietà di sistema os400.stdin, os400.stdout e os400.stderr.

Valore	Nome esempio	Descrizione	Esempio
File	SomeFileName	SomeFileName è un percorso assoluto o un percorso relativo per l'indirizzario corrente.	file:/QIBM/UserData/ Java400/Output.file
Porta	HostName	Indirizzo porta	port:myhost:2000
Porta	TCPAddress	Indirizzo porta	port:1.1.11.111:2000

Valori per la proprietà di sistema os400.verify.checks.disable:

Il valore della proprietà di sistema os400.verify.checks.disable è una stringa che rappresenta la somma di uno o più valori numerici dell'elenco che segue.

Valore	Descrizione
1	**Saltare i controlli di accesso per le classi locali:** indica che si desidera che la Java(TM) virtual machine salti i controlli di accesso di campi protetti e privati e metodi per classi caricate dal file system locale. Ciò è utile quando si trasferiscono le applicazioni che contengono classi interne che si riferiscono a campi e metodi protetti e privati delle relative classi allegate.
2	**Eliminare NoClassDefFoundError durante il caricamento iniziale:** indica che si desidera che la JVM (Java virtual machine) ignori NoClassDefFoundErrors, che si verifica durante i controlli di verifica iniziali per la personalizzazione e l'accesso al metodo o al campo.
4	**Consentire di saltare il controllo di LocalVariableTable:** indica che se si incontra un errore nella LocalVariableTable di una classe, la classe opera come se la LocalVariableTable non esistesse. Altrimenti gli errori nella LocaleVariableTable danno come risultato un ClassFormatError.
7	Valore utilizzato durante il tempo di esecuzione.

È possibile indicare il valore in formato decimale, esadecimale o ottale. Esso ignora i valori inferiori a zero. Ad esempio, per selezionare i primi due valori dall'elenco, utilizzare questa sintassi di comando:

```
JAVA CLASS(Hello) PROP((os400.verify.checks.disable 3))
```

Internazionalizzazione

È possibile personalizzare i programmi Java per una regione specifica del mondo creando un programma Java internazionalizzato. Utilizzando fusi orari, locale e codifiche caratteri è possibile ottenere che il programma Java rifletta l'ora, il luogo e la lingua corretta.

Internazionalizzazione di Sun Microsystems, Inc.

Globalizzazione di i5/OS

Configurazione del fuso orario

Quando si utilizzano programmi Java sensibili ai fusi orari, occorre configurare il fuso orario sul sistema affinché i programmi Java utilizzino l'orario corretto.

Il metodo più semplice per configurare il fuso orario consiste nell'impostare il valore di sistema QTIMZON su uno degli oggetti *TIMZON forniti da i5/OS. Per determinare correttamente l'ora locale, JVM (Java virtual machine) richiede che sia il valore di sistema QUTCOFFSET che la proprietà di sistema Java user.timezone siano impostati correttamente. L'impostazione del valore di sistema QTIMZON esegue entrambe le operazioni per conto dell'utente. Gli oggetti TIMZON contengono un nome lungo alternativo che specifica il valore Java user.timezone che verrà utilizzato; selezionare pertanto il valore QTIMZON che contiene il nome alternativo appropriato. Ad esempio, l'oggetto TIMZON QN0600CST2 contiene il nome alternativo America/Chicago e fornisce il corretto supporto orario per il fuso centrale degli Stati Uniti.

Nota: l'impostazione della proprietà di sistema user.timezone fornita dal valore di sistema QTIMZON può essere sostituita in due modi:

- Specificando il valore user.timezone in modo esplicito sulla riga comandi oppure nel file SystemDefault.properties

- Utilizzando LOCALE per impostare il valore user.timezone

  Il metodo LOCALE è stato utilizzato prima che fosse disponibile il supporto del valore di sistema QTIMZON su i5/OS, ma non è più consigliato.

Valore di sistema i5/OS: QTIMZON

Comando CL WRKTIMZON (Gestione descriz. fuso orario)

Sezione relativa alla Gestione delle locali nell'NLS (National Language Support)

Informazioni di riferimento Javadoc relative al fuso orario di Sun Microsystems, Inc.

Codifiche del carattere Java

I programmi Java possono convertire i dati in diversi formati, consentendo alle applicazioni di trasferire e utilizzare informazioni da diversi tipi di serie di caratteri internazionali.

Internamente, JVM ovvero Java virtual machine gestisce sempre i dati in Unicode. Tuttavia, tutti i dati trasferiti in JVM o fuori da essa sono nel formato corrispondente alla proprietà file.encoding. I dati letti in JVM vengono convertiti da file.encoding in Unicode e i dati inviati fuori alla JVM vengono convertiti da Unicode in file.encoding.

I file di dati per i programmi Java sono memorizzati nell'IFS (Integrated File System). I file nell'IFS vengono forniti di tag con un CCSID (Coded character set identifier) che identifica la codifica del carattere dei dati contenuti nel file.

Quando un programma Java legge i dati, ci si aspetta che siano nella codifica di carattere corrispondente a file.encoding. Quando un programma registra i dati in un file da un programma Java, ciò avviene in una codifica di carattere corrispondente a file.encoding. Ciò si applica anche ai file codice sorgente Java (file .java) elaborati dal comando javac e ai dati inviati e ricevuti tramite i socket TCP/IP (Transmission Control Protocol/Internet Protocol) utilizzando il pacchetto java.net.

I dati letti da o registrati in System.in, System.out e System.err sono gestiti in modo differente rispetto ai dati letti da o registrati in altre origini quando sono assegnati a stdin, stdout e stderr. Dato che stdin, stdout e stderr sono normalmente collegati alle unità EBCDIC su System i, JVM esegue una conversione sui dati per convertire la normale codifica di caratteri di file.encoding in un CCSID corrispondente al CCSID del lavoro System i. Quando si reindirizzano System.in, System.out o System.err su un file o socket e non si indirizzano su stdin, stdout o stderr, questa conversione di dati aggiuntiva non viene eseguita e i dati rimangono in una codifica di carattere corrispondente a file.encoding.

Quando è necessario leggere o scrivere i dati da un programma Java in una codifica di carattere diversa da file.encoding, il programma può utilizzare le classi IO Java java.io.InputStreamReader, java.io.FileReader, java.io.OutputStreamReader e java.io.FileWriter. Queste classi Java consentono di specificare un valore file.encoding che assume precedenza sulla proprietà file.encoding predefinita, correntemente utilizzata da JVM.

i dati verso o dal database DB2 vengono convertiti al o dal CCSID del database System i tramite le API JDBC.

I dati trasferiti in o da altri programmi tramite Java Native Interface non vengono convertiti.

Globalizzazione

Internazionalizzazione di Sun Microsystems, Inc.

Valori file.encoding e CCSID System i5:

La tabella riportata di seguito illustra la relazione tra i possibili valori file.encoding e il CCSID (coded character set identifier) System i5 più strettamente corrispondente.

Per ulteriori informazioni sul supporto file.encoding, consultare Codifiche supportate da Sun Microsystems, Inc.

file.encoding	CCSID	Descrizione
ASCII	367	ASCII (American Standard Code for Information Interchange)
Big5	950	T-Cinese BIG-5 ASCII a 8-bit
Big5_HKSCS	950	Big5_HKSCS
Big5_Solaris	950	Big5 con sette ulteriori definizioni di caratteri ideografici Hanzi per la locale Solaris zh_TW.BIG5 locale
CNS11643	964	Serie di caratteri nazionali cinesi per il cinese tradizionale
Cp037	037	IBM EBCDIC Stati Uniti, Canada, Paesi Bassi
Cp273	273	IBM EBCDIC Germania, Austria
Cp277	277	IBM EBCDIC Danimarca, Norvegia
Cp278	278	IBM EBCDIC Finlandia, Svezia
Cp280	280	IBM EBCDIC Italia
Cp284	284	IBM EBCDIC Spagna, America latina
Cp285	285	IBM EBCDIC Regno Unito
Cp297	297	IBM EBCDIC Francia
Cp420	420	IBM EBCDIC Arabo
Cp424	424	IBM EBCDIC Ebraico
Cp437	437	PC Stati Uniti ASCII a 8-bit
Cp500	500	IBM EBCDIC Internazionale
Cp737	737	MS-DOS Greco ASCII a 8-bit
Cp775	775	MS-DOS Baltico ASCII a 8-bit
Cp838	838	IBM EBCDIC Tailandia
Cp850	850	Latino-1 multinazionale ASCII a 8-bit
Cp852	852	Latino-2 ASCII a 8-bit
Cp855	855	Cirillico ASCII a 8-bit
Cp856	0	Ebraico ASCII a 8-bit
Cp857	857	Latino-5 ASCII a 8-bit
Cp860	860	Portogallo ASCII a 8-bit
Cp861	861	Islanda ASCII a 8-bit
Cp862	862	Ebraico ASCII a 8-bit
Cp863	863	Canada ASCII a 8-bit
Cp864	864	Arabo ASCII a 8-bit
Cp865	865	Danimarca, Norvegia ASCII a 8-bit
Cp866	866	Cirillico ASCII a 8-bit
Cp868	868	Urdu ASCII a 8-bit
Cp869	869	Greco ASCII a 8-bit
Cp870	870	IBM EBCDIC Latino-2
Cp871	871	IBM EBCDIC Islanda
Cp874	874	Tailandia ASCII a 8-bit
Cp875	875	IBM EBCDIC Greco

file.encoding	CCSID	Descrizione
Cp918	918	IBM EBCDIC Urdu
Cp921	921	Baltico ASCII a 8-bit
Cp922	922	Estonia ASCII a 8-bit
Cp930	930	IBM EBCDIC Giapponese esteso Katakana
Cp933	933	IBM EBCDIC Coreano
Cp935	935	IBM EBCDIC Cinese semplificato
Cp937	937	IBM EBCDIC Cinese tradizionale
Cp939	939	IBM EBCDIC Giapponese esteso Latino
Cp942	942	Giapponese ASCII a 8-bit
Cp942C	942	Variante di Cp942
Cp943	943	Dati PC misti in giapponese per open env
Cp943C	943	Dati PC misti in giapponese per open env
Cp948	948	8-bit ASCII IBM Cinese tradizionale
Cp949	944	Coreano ASCII a 8-bit KSC5601
Cp949C	949	Variante di Cp949
Cp950	950	T-Cinese BIG-5 ASCII a 8-bit
Cp964	964	EUC Cinese tradizionale
Cp970	970	EUC Coreano
Cp1006	1006	Urdu ISO a 8-bit
Cp1025	1025	IBM EBCDIC Cirillico
Cp1026	1026	IBM EBCDIC Turchia
Cp1046	1046	Arabo ASCII a 8-bit
Cp1097	1097	IBM EBCDIC Farsi
Cp1098	1098	Farsi ASCII a 8-bit
Cp1112	1112	IBM EBCDIC Baltico
Cp1122	1122	IBM EBCDIC Estonia
Cp1123	1123	IBM EBCDIC Ucraina
Cp1124	0	Ucraina ISO a 8-bit
Cp1140	1140	Variante di Cp037 con carattere dell'euro
Cp1141	1141	Variante di Cp273 con carattere dell'euro
Cp1142	1142	Variante di Cp277 con carattere dell'euro
Cp1143	1143	Variante di Cp278 con carattere dell'euro
Cp1144	1144	Variante di Cp280 con carattere dell'euro
Cp1145	1145	Variante di Cp284 con carattere dell'euro
Cp1146	1146	Variante di Cp285 con carattere dell'euro
Cp1147	1147	Variante di Cp297 con carattere dell'euro
Cp1148	1148	Variante di Cp500 con carattere dell'euro
Cp1149	1149	Variante di Cp871 con carattere dell'euro
Cp1250	1250	MS-Win Latino-2
Cp1251	1251	MS-Win Cirillico
Cp1252	1252	MS-Win Latino-1

file.encoding	CCSID	Descrizione
Cp1253	1253	MS-Win Greco
Cp1254	1254	MS-Win Turco
Cp1255	1255	MS-Win Ebraico
Cp1256	1256	MS-Win Arabo
Cp1257	1257	MS-Win Baltico
Cp1258	1251	MS-Win Russo
Cp1381	1381	S-Cinese GB ASCII a 8-bit
Cp1383	1383	EUC Cinese semplificato
Cp33722	33722	EUC Giapponese
EUC_CN	1383	EUC per Cinese semplificato
EUC_JP	5050	EUC per Giapponese
EUC_JP_LINUX	0	JISX 0201, 0208 , EUC giapponese codifica
EUC_KR	970	EUC per Coreano
EUC_TW	964	EUC per Cinese tradizionale
GB2312	1381	S-Cinese GB ASCII a 8-bit
GB18030	1392	Cinese semplificato, PRC standard
GBK	1386	Nuovo Cinese semplificato ASCII 9 a 8-bit
ISCII91	806	Codifica ISCII91 di script indiani
ISO2022CN	965	ISO 2022 CN, cinese (solo conversione in Unicode)
ISO2022_CN_CNS	965	CNS11643 in ISO 2022 CN, cinese tradizionale (conversione da Unicode soltanto)
ISO2022_CN_GB	1383	GB2312 in ISO 2022 CN, cinese semplificato (conversione da Unicode soltanto)
ISO2022CN_CNS	965	ASCII a 7-bit per Cinese tradizionale
ISO2022CN_GB	1383	ASCII a 7-bit per Cinese semplificato
ISO2022JP	5054	ASCII a 7-bit per Giapponese
ISO2022KR	25546	ASCII a 7-bit per Coreano
ISO8859_1	819	ISO 8859-1 Alfabeto latino n. 1
ISO8859_2	912	ISO 8859-2 ISO Latino-2
ISO8859_3	0	ISO 8859-3 ISO Latino-3
ISO8859_4	914	ISO 8859-4 ISO Latino-4
ISO8859_5	915	ISO 8859-5 ISO Latino-5
ISO8859_6	1089	ISO 8859-6 ISO Latino-6 (Arabo)
ISO8859_7	813	ISO 8859-7 ISO Latino-7 (Greco/Latino)
ISO8859_8	916	ISO 8859-8 ISO Latino-8 (Ebraico)
ISO8859_9	920	ISO 8859-9 ISO Latino-9 (ECMA-128, Turchia)
ISO8859_13	0	Alfabeto latino n. 7
ISO8859_15	923	ISO8859_15
ISO8859_15_FDIS	923	ISO 8859-15, Alfabeto latino n. 9
ISO-8859-15	923	ISO 8859-15, Alfabeto latino n. 9
JIS0201	897	Standard industriale Giapponese X0201
JIS0208	5052	Standard indutriale Giapponese X0208

file.encoding	CCSID	Descrizione
JIS0212	0	Standard industriale Giapponese X0212
JISAutoDetect	0	Trova ed esegue la conversione da Shift-JIS, EUC-JP, ISO 2022 JP (solo conversione in Unicode)
Johab	0	Codifica Hangul di composizione Coreana (completa)
K018_R	878	Cirillico
KSC5601	949	Coreano ASCII a 8-bit
MacArabic	1256	Macintosh Arabico
MacCentralEurope	1282	Macintosh latino-2
MacCroatian	1284	Macintosh Croato
MacCyrillic	1283	Macintosh Cirillico
MacDingbat	0	Macintosh Dingbat
MacGreek	1280	Macintosh Greco
MacHebrew	1255	Macintosh ebraico
MacIceland	1286	Macintosh Islanda
MacRoman	0	Macintosh Romeno
MacRomania	1285	Macintosh Romania
MacSymbol	0	Macintosh Simboli
MacThai	0	Macintosh tailandese
MacTurkish	1281	Macintosh turco
MacUkraine	1283	Macintosh ucraino
MS874	874	MS-Win Tailandia
MS932	943	Windows giapponese
MS936	936	Windows cinese semplificato
MS949	949	Windows coreano
MS950	950	Windows cinese tradizionale
MS950_HKSCS	NA	Windows cinese tradizionale con estensioni Hong Kong S.A.R. della Cina
SJIS	932	Giapponese ASCII a 8-bit
TIS620	874	Standard industriale tailandese 620
US-ASCII	367	ASCII (American Standard Code for Information Interchange)
UTF8	1208	UTF-8 (CCSID IBM 1208, che non è ancora disponibile sulla piattaforma System i5)
UTF-16	1200	Formato di conversione UCS a 16 bit, ordine dei byte identificato da un segno opzionale di ordine dei byte
UTF-16BE	1200	Formato di conversione Unicode a 16 bit, ordine byte big-endian
UTF-16LE	1200	Formato di conversione Unicode a 16 bit, ordine byte little-endian
UTF-8	1208	Formato di trasformazione UCS a 8 bit
Unicode	13488	UNICODE, UCS-2
UnicodeBig	13488	Lo stesso di Unicode
UnicodeBigUnmarked		Unicode senza contrassegno ordine di byte
UnicodeLittle		Unicode con ordine di byte little-endian
UnicodeLittleUnmarked		UnicodeLittle senza contrassegno di ordine di byte

Per i valori predefiniti, consultare Valori file.encoding predefiniti.

Valori file.encoding predefiniti:

Questa tabella mostra in che modo viene impostato il valore file.encoding in base al CCSID (System i coded character set identifier) all'avvio della JVM (Java Virtual Machine).

CCSID System i	File.encoding predefinito	Descrizione
37	ISO8859_1	Inglese per Stati Uniti, Canada, Nuova Zelanda e Australia; portoghese per Portogallo e Brasile; olandese per Paesi Bassi
256	ISO8859_1	#1 internazionale
273	ISO8859_1	Tedesco/Germania, Tedesco/Austria
277	ISO8859_1	Danese/Danimarca, Norvegese/Norvegia, Norvegese/Norvegia, NY
278	ISO8859_1	Finlandese/Finlandia
280	ISO8859_1	Italiano/Italia
284	ISO8859_1	Catalano/Spagna, Spagnolo/Spagna
285	ISO8859_1	Inglese/Gran Bretagna, Inglese/Irlanda
290	Cp943C	Parte SBCS del Giapponese EBCDIC misto (CCSID 5026)
297	ISO8859_1	Francese/Francia
420	Cp1046	Arabo/Egitto
423	ISO8859_7	Grecia
424	ISO8859_8	Ebraico/Israele
500	ISO8859_1	Tedesco/Svizzera, Francese/Belgio, Francese/Canada, Francese/Svizzera
833	Cp970	Parte SBCS del Coreano EBCDIC misto (CCSID 933)
836	Cp1383	Parte SBCS del Cinese-S EBCDIC misto (CCSID 935)
838	TIS620	Tailandese
870	ISO8859_2	Ceco/Repubblica Ceca, Croato/Croazia, Ungherese/ Ungheria, Polacco/Polonia
871	ISO8859_1	Islandese/Islanda
875	ISO8859_7	Greco/Grecia
880	ISO8859_5	Bulgaria (ISO 8859_5)
905	ISO8859_9	Turchia estesa
918	Cp868	Urdu
930	Cp943C	Giapponese EBCDIC misto (simile a CCSID 5026)
933	Cp970	Coreano/Corea
935	Cp1383	Cinese semplificato
937	Cp950	Cinese tradizionale

CCSID System i	File.encoding predefinito	Descrizione
939	Cp943C	Giapponese EBCDIC misto (simile a CCSID 5035)
1025	ISO8859_5	Bielorusso/Bielorussia, Bulgaro/Bulgaria, Macedone/Macedonia, Russo/Russia
1026	ISO8859_9	Turco/Turchia
1027	Cp943C	Parte SBCS del Giapponese EBCDIC misto (CCSID 5035)
1097	Cp1098	Farsi
1112	Cp921	Lituano/Lituania, Lettone/Lettonia, Baltico
1388	GBK	Cinese semplificato EBCDIC misto (GBK è incluso)
5026	Cp943C	Giapponese EBCDIC misto CCSID (Katakana esteso)
5035	Cp943C	Giapponese EBCDIC misto CCSID (Latino esteso)
8612	Cp1046	Arabo (solo caratteri di base) (o ASCII 420 e 8859_6)
9030	Cp874	Tailandese (host esteso SBCS)
13124	GBK	Parte SBCS del Cinese semplificato EBCDIC misto (GBK è incluso)
28709	Cp948	Parte SBCS del Cinese tradizionale EBCDIC misto (CCSID 937)

Esempi: creazione di un programma Java internazionalizzato

Se è necessario personalizzare un programma Java per una specifica regione del mondo, è possibile creare un programma Java internazionalizzato con le locali Java.

Locali Java.

La creazione di un programma Java internazionalizzato comporta diverse attività:

1. Isolamento di dati e codici sensibili alla locale. Ad esempio, stringhe, date e numeri contenuti nel programma.
2. Impostazione e acquisizione della locale utilizzando la classe Locale.
3. Formattazione di date o numeri per la specifica di una locale, se non si desidera utilizzare la locale predefinita.
4. Creazione di pacchetti di risorse per la gestione di stringhe e di altri dati sensibili alla locale.

Fare riferimento agli esempi di seguito riportati per effettuare le attività necessarie alla creazione di un programma Java internazionalizzato:

- "Esempio: internazionalizzazione delle date utilizzando la classe java.util.DateFormat" a pagina 446
- "Esempio: internazionalizzazione del pannello numerico utilizzando la classe java.util.NumberFormat" a pagina 447
- "Esempio: internazionalizzazione di dati specifici della locale utilizzando la classe java.util.ResourceBundle" a pagina 447

Globalizzazione

Internazionalizzazione di Sun Microsystems, Inc.

Compatibilità tra release

Nella V6R1, l'elaborazione diretta non è più supportata. Questa modifica ha delle implicazioni sia sul parametro OPTIMIZE che sul parametro ENBPFRCOL.

Nella V6R1, l'elaborazione diretta non è più supportata. Questo significa che il livello di ottimizzazione per i programmi Java è ignorato e OPTIMIZE(*INTERPRET) viene utilizzato quando si crea un programma Java su V6R1 o un release successivo.

Come nei release precedenti, un programma Java contiene una versione precedentemente verificata di uno o più file di classe Java. Tuttavia, in V6R1, un programma Java non contiene istruzioni macchina. Durante il tempo di esecuzione, il programma Java viene interpretato dai codici byte o altrimenti eseguito con il compilatore JIT (Just-In-Time).

Per i release di destinazione precedenti alla V6R1, il parametro OPTIMIZE specifica il livello di ottimizzazione del programma Java. Quando si crea un programma Java per un release di destinazione precedente alla V6R1, il valore del parametro OPTIMIZE è incapsulato nel programma Java, ma non viene generata alcuna istruzione macchina. Il valore incapsulato del parametro OPTIMIZE viene utilizzato durante la riconversione del programma Java sul release di destinazione del sistema operativo.

Per i programmi Java creati per il release di destinazione V6R1 e successive, il valore di parametro ENBPFRCOL viene ignorato e viene utilizzato ENDPFRCOL(*NONE). Per abilitare la raccolta delle prestazioni per le applicazioni Java in V6R1 e release successivi, è necessario utilizzare la proprietà di sistema Java os400.enbpfrcol. Per ulteriori informazioni, consultare l'argomento "Elenco delle proprietà di sistema Java" a pagina 16.

Per un release di destinazione precedente a V6R1, è possibile specificare dei valori diversi da *NONE, ma dei valori diversi da *NONE non saranno effettivi finché il programma Java non verrà riconvertito sul release di destinazione del sistema operativo.

Concetti correlati

"Novità nella V6R1" a pagina 1
Leggere le informazioni sulle informazioni nuove o notevolmente modificate per la raccolta di argomenti di IBM Developer Kit per Java.

"Considerazioni sulle prestazioni Java" a pagina 412
La conoscenza delle seguenti considerazioni può essere di ausilio nel migliorare le prestazioni delle applicazioni Java.

Accesso al database con il IBM Developer Kit per Java

Con il IBM Developer Kit per Java, i programmi Java possono accedere ai file database in vari modi.

Accesso al database System i5 con il programma di controllo JDBC IBM Developer Kit per Java

Il programma di controllo JDBC IBM Developer Kit per Java, noto anche come programma di controllo "nativo", fornisce un accesso programmatico ai file di database System i5. Utilizzando l'API JDBC (Java Database Connectivity), le applicazioni scritte nel linguaggio Java possono accedere alle funzioni del database JDBC con l'SQL (Structured Query Language) incorporato, eseguire le istruzioni SQL, richiamare i risultati e trasmettere le modifiche verso il database. L'API JDBC può inoltre essere utilizzato per interagire con più risorse dati in ambiente distribuito ed eterogeneo.

La CLI (Command Language Interface) SQL99, su cui si basa l'API JDBC, è la base per ODBC. JDBC fornisce una correlazione naturale e di semplice utilizzo dal linguaggio di programmazione Java alle idee e ai concetti definiti nello standard SQL.

Documentazione per JDBC di Sun Microsystems, Inc.

Domande frequenti sul programma di controllo JDBC nativo

Specifica API di JDBC 4.0

Introduzione a JDBC

Il programma di controllo JDBC (Java Database ConnectivityI fornito con IBM Developer Kit per Java è denominato programma di controllo JDBC di IBM Developer Kit per Java. Questa unità è inoltre comunemente conosciuta come programma di controllo JDBC nativo.

Per selezionare quale programma di controllo JDBC si adatti alle necessità dell'utente, considerare i seguenti suggerimenti:

- Sarebbe opportuno che i programmi eseguiti direttamente su un server dove si trova il database utilizzassero il programma di controllo JDBC nativo per le prestazioni. Questo include la maggior parte delle soluzioni servlet e JSP (JavaServer Page) e le applicazioni scritte per essere eseguite localmente su un sistema.
- I programmi che devono collegarsi ad un System i5 remoto utilizzano le classi JDBC IBM Toolbox per Java . Il programma di controllo JDBC IBM Toolbox per Java è una solida implementazione di JDBC e viene fornito come parte di IBM Toolbox per Java. Essendo Java, puro, il programma di controllo JDBC di IBM Toolbox per Java è facile da configurare per i client e richiede poche operazioni di configurazione del server.
- I programmi eseguiti su un System i5 e che devono collegarsi ad un database remoto non System i5 utilizzano il programma di controllo JDBC nativo e configurano un collegamento DRDA (Distributed Relational Database Architecture) a tale server remoto.

Tipi di programmi di controllo JDBC:

Questo argomento definisce i tipi di programma di controllo JDBC (Java Database Connectivity). Tali tipi vengono utilizzati per suddividere in categorie la tecnologia usata per collegarsi al database. Un fornitore di programmi di controllo JDBC utilizza tali tipi per descrivere il modo in cui opera il relativo prodotto. Alcuni tipi di programmi di controllo JDBC sono più adatti per alcune applicazioni rispetto ad altre.

Tipo 1

I programmi di controllo di tipo 1 sono unità "bridge". Esse utilizzano un'altra tecnologia come ODBC (Open Database Connectivity) per comunicare con un database. Questo è un vantaggio dato che i programmi di controllo ODBC esistono per molte piattaforme RDBMS (Relational Database Management System). JNI (Java Native Interface) viene utilizzata per richiamare funzioni ODBC dal programma di controllo JDBC.

Un programma di controllo di tipo 1 deve avere l'unità bridge installata e configurata prima che sia possibile utilizzare JDBC con esso. Ciò può costituire uno svantaggio considerevole per un'applicazione di produzione. Non è possibile utilizzare i programmi di controllo di tipo 1 in un'applet in quanto le applet non possono caricare un codice nativo.

Tipo 2

I programmidi controllo di tipo 2 utilizzano un'API nativa per comunicare con un sistema di database. I metodi nativi Java sono utilizzati per richiamare le funzioni API che eseguono le operazioni del database. I programmi di controllo di tipo 2 sono generalmente più rapidi rispetto a quelli di tipo 1.

Affinché i programmi di controllo di tipo 2 funzionino, è necessario che il codice binario sia installato e configurato. Anche un programma di controllo di tipo 2 utilizza la JNI. Non è possibile utilizzare un

programma di controllo di tipo 2 in un'applet in quanto le applet non possono caricare un codice nativo. È possibile che un programma di controllo JDBC di tipo 2 richieda l'installazione di un software di rete DBMS (Database Management System).

Il programma di controllo JDBC del Developer Kit per Java è un programma di controllo JDBC di tipo 2.

Tipo 3

Questi programmi di controllo utilizzano un protocollo di rete e un middleware per comunicare con un server. Il server converte il protocollo in chiamate di funzione DBMS specifiche per DBMS.

I programmi di controllo JDBC di tipo 3 sono la soluzione JDBC più flessibile in quanto non richiedono alcun codice binario nativo sul client. Un programma di controllo di tipo 3 non richiede alcuna installazione client.

Tipo 4

Un programma di controllo di tipo 4 utilizza Java per implementare un protocollo di rete del fornitore DBMS. Dato che i protocolli sono solitamente di proprietà, i fornitori DBMS sono generalmente le uniche società che forniscono un programma di controllo JDBC di tipo 4.

I programmi di controllo di tipo 4 sono tutte unità Java. Ciò significa che non esiste alcuna configurazione o installazione client. Tuttavia, un programma di controllo di tipo 4 potrebbe non essere adatto per alcune applicazioni se il protocollo sottostante non gestisce bene questioni come connettività di rete e sicurezza.

Il programma di controllo di IBM Toolbox per Java JDBC è un programma di controllo JDBC di tipo 4, che indica che la API è un programma di controllo del protocollo di rete Java puro.

Requisiti JDBC:

Quest'argomento indica i requisiti necessari per accedere a JDBC principale e a JTA (Java Transaction API).

Prima di scrivere e distribuire le applicazioni JDBC potrebbe essere necessario includere nel classpath dei file jar specifici.

JDBC principale

Per l'accesso di JDBC (Java Database Connectivity) principale al database locale, non esistono requisiti. Tutto il supporto è incorporato, preinstallato e configurato.

Compatibilità JDBC

Il programma di controllo JDBC nativo è compatibile con tutte le specifiche JDBC pertinenti. Il livello di compatibilità del programma di controllo JDBC non dipende dal release i5/OS, ma dal release JDK che si utilizza. Il livello di compatibilità del programma di controllo JDBC nativo per i vari JDK viene elencato come segue:

Versione	Livello di compatibilità del programma di controllo JDBC
JDK 1.4 e versioni successive	Queste versioni JDK sono conformi a JDBC 3.0.
J2SE 6 e versioni successive	JDBC 4.0

Supporto didattico JDBC:

Quello che segue è un supporto didattico sulla scrittura di un programma JDBC (Java Database Connectivity) e sulla sua esecuzione su un System i5 con il programma di controllo JDBC nativo. Esso è progettato per mostrare all'utente le fasi basilari necessarie affinché il proprio programma esegua JDBC.

L'esempio crea una tabella e la popola con alcuni dati. Esso elabora una query per acquisire i dati dal database e per visualizzarli sullo schermo.

Eseguire il programma di esempio

Per eseguire il programma di esempio, sono necessarie le seguenti fasi:

1. Copiare il programma nella propria stazione di lavoro.
   a. Copiare l'esempio ed incollarlo in un file sulla stazione di lavoro.
   b. Salvare il file con lo stesso nome classe public fornito e con l'estensione .java. In questo caso, è necessario denominare il file BasicJDBC.java sulla propria stazione di lavoro locale.
2. Trasferire il file dalla stazione di lavoro al server. Da una richiesta comandi, immettere i seguenti comandi:

   ```
   ftp <nome server>
   <Enter your user ID>
   <Enter your password>
   cd /home/cujo
   put BasicJDBC.java
   quit
   ```

   Affinché questi comandi funzionino, è indispensabile disporre di un indirizzario in cui inserire il file. Nell'esempio, /home/cujo è l'ubicazione, ma è possibile utilizzare qualsiasi ubicazione si desideri.

   Nota: è possibile che i comandi FTP appena riportati siano differenti in base all'impostazione del proprio server, ma dovrebbero essere simili. Non è importante il modo in cui si trasferisce il file al proprio server a condizione che lo si trasferisca nell'IFS (Integrated File System). Strumenti come VisualAge per Java possono automatizzare completamente questo processo per l'utente.
3. Assicurarsi di impostare il proprio classpath nell'indirizzario dove si inserisce il file in modo che i comandi Java trovino i file quando l'utente li esegue. Da una riga comandi CL, è possibile utilizzare WRKENVVAR per esaminare quali variabili di ambiente sono impostate per il proprio profilo utente.
   - Se si visualizza una variabile di ambiente denominata CLASSPATH, è necessario assicurarsi che l'ubicazione in cui si inserisce il file .java sia nella stringa di indirizzari elencati in quell'ambito o aggiungerla se non è stata specificata.
   - Se non esiste alcuna variabile di ambiente CLASSPATH, è necessario aggiungerne una. È possibile effettuare ciò con il seguente comando:

     ```
     ADDENVVAR ENVVAR(CLASSPATH)
     VALUE('/home/cujo:/QIBM/ProdData/Java400/jdk15/lib/tools.jar')
     ```

   Nota: per compilare il codice Java dal comando CL, è necessario includere il file tools.jar. Tale file JAR include il comando javac.
4. Compilare il file Java in un file di classe. Immettere il seguente comando dalla riga comandi CL:

   ```
   JAVA CLASS(com.sun.tools.javac.Main) PARM(My_Program_Name.java)
   java BasicJDBC
   ```

   È inoltre possibile compilare il file Java da QSH:

   ```
   cd /home/cujo
   javac BasicJDBC.java
   ```

   QSH assicura automaticamente la possibilità di rilevare il file tools.jar. Come risultato, non è necessario aggiungerlo al proprio classpath. Anche l'indirizzario corrente è nel classpath. Immettendo il comando cd (modifica indirizzario), viene trovato anche il file BasicJDBC.java.

Nota: È inoltre possibile compilare il file sulla propria stazione di lavoro e utilizzare FTP per inviare il file di classe al proprio server in modalità binaria. Questo è un esempio della capacità di Java di eseguire su qualsiasi piattaforma.

Eseguire il programma utilizzando il seguente comando dalla riga comandi CL o da QSH:

```
java BasicJDBC
```

L'emissione è riportata di seguito:

```
----------------------
| 1 | Frank Johnson  |
|                    |
| 2 | Neil Schwartz  |
|                    |
| 3 | Ben Rodman     |
|                    |
| 4 | Dan Gloore     |
----------------------
There were 4 rows returned.
Output is complete.
Java program completed.
```

Sito Web del programma di controllo JDBC di IBM Toolbox per Java

Pagina su JDBC di Sun Microsystems, Inc.

Esempio: JDBC:

Questo è un esempio della modalità di utilizzo del programma BasicJDBC. Questo programma utilizza il programma di controllo JDBC nativo per IBM Developer Kit per Java per creare una semplice tabella ed elaborare una query che visualizza i dati in tale tabella.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio BasicJDBC. Questo programma utilizza il programma di controllo JDBC nativo
// affinché Developer Kit per Java crei una tabella semplice ed elabori una query
// che visualizzi i dati in tale tabella.
//
// Sintassi del comando:
//    BasicJDBC
//
////////////////////////////////////////////////////////////////////////////
//
// Questo codice sorgente è un esempio del programma di controllo JDBC IBM Developer
Kit per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
```

```
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////////////////////////////////////////////

// Includere le classi Java da utilizzare. In questa applicazione, molte classi
// del pacchetto java.sql vengono utilizzate e anche la classe
// java.util.Properties viene utilizzata come parte per stabilire
// un collegamento al database.
import java.sql.*;
import java.util.Properties;

// Creare una classe public che includa il programma.
public class BasicJDBC {

    // Il collegamento è una variabile privata dell'oggetto.
    private Connection connection = null;

    // Qualsiasi classe che deve essere un 'punto di immissione' per eseguire
    // un programma deve disporre di un metodo principale. Quest'ultimo costituisce
    // il punto di partenza dell'elaborazione nel momento in cui viene chiamato il programma.
    public static void main(java.lang.String[] args) {

        // Creare un oggetto di tipo BasicJDBC. Esso si rivela
        // fondamentale nella programmazione orientata agli oggetti. Una volta
        // creato un oggetto, chiamare diversi metodi su tale
        // oggetto per eseguire il lavoro.
        // In questo caso, chiamando il programma di creazione dell'oggetto
        // viene creata un collegamento al database che viene utilizzato
        // dagli altri metodi per eseguire un lavoro sul database.
        BasicJDBC test = new BasicJDBC();

        // Chiamare il metodo rebuildTable. Questo metodo assicura che la
        // tabella utilizzata nel programma esiste e viene visualizzata
        // correttamente. Il valore di ritorno è di tipo booleano per
        // indicare se la nuova creazione della tabella è stata completata
        // con esito positivo o meno. In caso non lo sia, viene visualizzato
        // un messaggio e si esce dal programma.
        if (!test.rebuildTable()) {
            System.out.println("Failure occurred while setting up " +
                               " for running the test.");
            System.out.println("Test will not continue.");
            System.exit(0);
        }

        // Il metodo run query viene chiamato successivamente. Questo metodo
        // elabora un'istruzione di selezione SQL sulla tabella che è stata
        // creata nel metodo rebuildTable. L'emissione di tale query
        // viene inviato all'emissione standard affinché possa essere esaminata.
        test.runQuery();

        // Infine, viene chiamato il metodo cleanup. Questo metodo assicura
        // che il collegamento al database cui è collegato l'oggetto
        // viene chiuso.
        test.cleanup();
    }


    /**
    Questo è il programma di creazione per la verifica JDBC. Esso crea un collegamento
    database che viene memorizzato in una variabile di istanza da utilizzare in
    successive chiamate al metodo.
    **/
    public BasicJDBC() {

        // Un modo per creare un collegamento database consiste nel passare un URL
        // e un oggetto di proprietà java in DriverManager. Il codice riportato
```

```
    // di seguito crea un oggetto di proprietà che dispone di un proprio ID utente
    // e parola d'ordine. Queste parti di informazioni vengono utilizzate per
    // collegarsi al database.
    Properties properties = new Properties ();
    properties.put("user", "cujo");
    properties.put("user", "newtiger");

    // Utilizzare un blocco try/catch per acquisire tutte le eccezioni che
    // derivano dal seguente codice.
    try {
        // DriverManager deve sapere che esiste un programma di controllo JDBC disponibile
        // per gestire una richiesta di collegamento utente. La seguente riga
        // fa in modo che il programma di controllo JDBC venga caricato e che DriveManager venga registrato.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        // Creare l'oggetto DatabaseConnection utilizzato da questo programma in
        // tutte le altre chiamate del metodo effettuate. Il seguente codice
        // specifica che deve essere stabilito un collegamento al database locale
        // e che tale collegamento deve essere conforme alle proprietà impostate
        // precedentemente (cioè, deve utilizzare l'ID utente e la parola d'ordine specificati).
        connection = DriverManager.getConnection("jdbc:db2:*local", properties);

    } catch (Exception e) {
        // Se una qualsiasi delle righe del blocco try/catch ha esito negativo,
        // controllare i trasferimenti alla seguente riga di codice. Un'applicazione
        // affidabile tenta di gestire il problema o di fornire ulteriori dettagli.
        // In questo programma viene visualizzato il messaggio di errore dell'eccezione
        // e l'applicazione consente al programma di eseguire una restituzione.
        System.out.println("Caught exception: " + e.getMessage());
    }
}


/**
Assicurarsi che la tabella qgpl.basicjdbc venga visualizzata correttamente all'inizio
della verifica.

@returns boolean    Restituisce true se la tabella è stata ricreata con esito positivo;
                    restituisce false se si è verificato un errore.
**/
public boolean rebuildTable() {
    // Raggruppare tutte le funzionalità in un blocco try/catch in modo che si
    // tenti di gestire gli errori quando si verificano all'interno di questo metodo.
    try {

        // Gli oggetti Statement vengono usati per elaborare istruzioni SQL sul
        // database. L'oggetto Connection è utilizzato per creare uno Statement
        // Statement.
        Statement s = connection.createStatement();

        try {
            // Creare la tabella di verifica da zero. Elaborare un'istruzione di
            // aggiornamento che tenti di cancellare la tabella se attualmente esiste.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // Non eseguire nulla se si verifica un'eccezione. Si presuppone che
            // il problema consiste nel fatto che la tabella interrotta non
            // esiste e che può essere creata successivamente.
        }

        // Utilizzare l'oggetto statement per creare la tabella.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Utilizzare l'oggetto statement per popolare la tabella con alcuni dati.
        s.executeUpdate("insert into qgpl.basicjdbc values(1, 'Frank Johnson')");
        s.executeUpdate("insert into qgpl.basicjdbc values(2, 'Neil Schwartz')");
        s.executeUpdate("insert into qgpl.basicjdbc values(3, 'Ben Rodman')");
```

```
        s.executeUpdate("insert into qgpl.basicjdbc values(4, 'Dan Gloore')");

        // Chiudere l'istruzione SQL per indicare al database che non è più
        // necessario.
        s.close();

        // Se l'intero metodo viene elaborato con esito positivo, viene restituito true.
        // A questo punto, la tabella è stata creata o aggiornata correttamente.
        return true;

    } catch (SQLException sqle) {
        // Se una delle istruzioni SQL ha esito negativo (diverso dall'interruzione
        // della tabella gestita nel blocco try/catch interno), viene visualizzato il
        // messaggio di errore e viene restituito false al chiamante,
        // indicante che potrebbe non essere completa.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}



/**
Esegue una query sulla tabella dimostrativa e i risultati vengono visualizzati
nell'emissione standard.
**/
public void runQuery() {
    // Raggruppare tutte le funzionalità in un blocco try/catch in modo che si
    // tenti di gestire gli errori quando si verificano all'interno di questo
    // metodo.
    try {
        // Creare un oggetto Statement.
        Statement s = connection.createStatement();

        // Utilizzare l'oggetto statement per eseguire una query SQL. Le
        // query restituiscono oggetti ResultSet usati per consultare i
        // dati forniti dalla query.
        ResultSet rs = s.executeQuery("select * from qgpl.basicjdbc");

        // Visualizzare la parte superiore della 'tabella' e iniziare il conteggio
        // del numero di righe restituite.
        System.out.println("--------------------");
        int i = 0;

        // Il successivo metodo ResultSet viene usato per elaborare le righe di un
        // ResultSet. Il successivo metodo deve essere chiamato una volta prima che i
        // primi dati siano disponibili per la visualizzazione. Purché venga restituito
        // true, esiste un'altra riga di dati che può essere utilizzata.
        while (rs.next()) {

            // Ottenere entrambe le colonne nella tabella per ogni riga e scrivere una
            // riga nella tabella sul pannello con i dati. Quindi, aumentare il
            // conteggio delle righe elaborate.
            System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
            i++;
        }

        // Inserire un bordo alla fine della tabella e visualizzare il numero
        // delle righe come emissione.
        System.out.println("--------------------");
        System.out.println("There were " + i + " rows returned.");
        System.out.println("Output is complete.");

    } catch (SQLException e) {
        // Visualizzare le informazioni aggiuntive sulle eccezioni SQL
        // che vengono generate come emissione.
        System.out.println("SQLException exception: ");
```

```
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            e.printStackTrace();
        }
    }


    /**
    Il seguente metodo assicura che le risorse JDBC ancora assegnate
    sono libere.
    **/
    public void cleanup() {
        try {
            if (connection != null)
                connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Utilizzo della JNDI per gli esempi IBM Developer Kit per Java:

I DataSources lavorano di pari passo con JNDI (Java Naming and Directory Interface). JNDI è un livello di astrazione Java per i servizi dell'indirizzario così come JDBC (Java Database Connectivity) è un livello di astrazione per i database.

viene utilizzato più frequentemente con LDAP (Lightweight Directory Access Protocol), ma è possibile utilizzarlo anche con COS (CORBA Object Service), registro RMI (Remote Method Invocation) Java o file system sottostante. Questo utilizzo vario viene compiuto tramite diversi tecnici di manutenzione dell'indirizzario che convertono le comuni richieste JNDI in specifiche richieste di servizio dell'indirizzario.

Nota: tener presente che l'utilizzo di RMI può essere un'attività complessa. Prima di scegliere RMI come soluzione, assicurarsi di conoscere tutte le eventuali ramificazioni di questa operazione. Un buon punto di partenza per conoscere RMI è RMI Java (Remote Method Invocation).

Gli esempi DataSource sono stati progettati utilizzando il tecnico di manutenzione del file system JNDI. Se si intende eseguire gli esempi forniti, è necessario un tecnico di manutenzione JNDI.

Seguire queste indicazioni per impostare l'ambiente per il tecnico di manutenzione del file system:

1. Scaricare il supporto JNDI del file system dal sito JNDI di Sun Microsystems.
2. Trasferire (con FTP o altro meccanismo) fscontext.jar e providerutil.jar nel sistema e inserirli in /QIBM/UserData/Java400/ext. Questo è l'indirizzario delle estensioni e dei file JAR in essi collocati che vengono rilevati automaticamente quando si esegue l'applicazione (ovvero non è necessario che siano nel classpath).

Una volta che si dispone del supporto per un tecnico di manutenzione per JNDI, è necessario impostare le informazioni sul contesto per le proprie applicazioni. Ciò può essere compiuto inserendo le informazioni richieste in un file SystemDefault.properties. Esistono vari ambiti sul sistema dove è possibile specificare le proprietà predefinite, ma il modo migliore è creare un file di testo denominato SystemDefault.properties nel proprio indirizzario principale (cioè, in /home/).

Per creare un file, utilizzare le seguenti righe o aggiungerle al proprio file esistente:

```
# Needed env settings for JNDI.
java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
java.naming.provider.url=file:/DataSources/jdbc
```

Queste righe specificano che il tecnico di manutenzione del file system gestisce richieste JNDI e che /DataSources/jdbc è il root per attività che utilizzano JNDI. È possibile modificare questa ubicazione, ma è necessario che esista l'indirizzario che si specifica. Viene specificata l'ubicazione in cui i DataSource dell'esempio sono collegati e applicati.

Connessioni

L'oggetto Connection rappresenta un collegamento ad una origine dati in JDBC (Java Database Connectivity). Gli oggetti Statement vengono creati tramite gli oggetti Connection per l'elaborazione di istruzioni SQL sul database. Un programma applicativo può avere più collegamenti allo stesso tempo. Tutti questi oggetti Connection possono collegarsi allo stesso database o a database differenti.

È possibile ottenere un collegamento in JDBC in due modi:

- Tramite la classe DriverManager.
- Utilizzando DataSource.

L'utilizzo delle DataSource per ottenere un collegamento è preferibile in quanto aumenta la portabilità e manutenibilità dell'applicazione. Esso consente inoltre ad un'applicazione di utilizzare in modo chiaro il collegamento e il lotto dell'istruzione e le transazioni distribuite.

Concetti correlati

Creare vari tipi di oggetti Statement per interagire con il database
Si utilizza un oggetto Statement per l'elaborazione di un'istruzione SQL statica e l'ottenimento dei risultati prodotti da questa. È possibile aprire solo un ResultSet alla volta per ciascun oggetto Statement. Tutti i metodi dell'istruzione che elaborano un'istruzione SQL chiudono implicitamente un ResultSet corrente dell'istruzione se ne esiste uno aperto.

Controllare le transazioni sul database
Una transazione è un'unità logica di lavoro. Per completare un'unità logica di lavoro, potrebbe essere necessario eseguire varie azioni su un database.

Richiamare i metadati relativi al database
L'interfaccia DatabaseMetaData è implementata dal programma di controllo JDBC IBM Developer Kit per Java per fornire informazioni sulle sue origini di dati sottostanti. Essa viene utilizzata principalmente dagli strumenti e server dell'applicazione per stabilire come interagire con una specifica origine dati. Le applicazioni possono inoltre utilizzare i metodi DatabaseMetaData per ottenere informazioni su una origine dati, ma ciò avviene meno frequentemente.

Classe DriverManager Java:

DriverManager è una classe statica in J2SE (Java 2 Plaform, Standard Edition) e JDK (Java SE Development Kit). DriverManager gestisce la serie di programmi di controllo JDBC (Java Database Connectivity) disponibile per le applicazioni.

Le applicazioni possono utilizzare più programmi di controllo JDBC contemporaneamente se necessario. Ciascuna applicazione specifica un programma di controllo JDBC utilizzando un URL (Uniform Resource Locator). Inoltrando un URL per un programma di controllo JDBC specifica al DriverManager, l'applicazione informa il DriverManager del tipo di collegamento JDBC da restituire all'applicazione.

Prima di questa operazione, DriverManager deve conoscere i programmi di controllo JDBC disponibili in modo da poter distribuire i collegamenti. Effettuando una chiamata al metodo Class.forName, esso carica una classe nella JVM (Java virtual machine) in esecuzione in base al relativo nome stringa che viene passato nel metodo. Segue un esempio del metodo class.forName utilizzato per caricare il programma di controllo JDBC nativo:

Esempio: caricamento del programma di controllo JDBC nativo

```
// Caricare il programma di controllo JDBC nativo nel DriverManager per
// renderlo disponibile per le richieste getConnection.

Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

I programmi di controllo JDBC sono progettati per informare il DriverManager della loro esistenza automaticamente quando la relativa classe di implementazione dell'unità viene caricata. Una volta elaborata la riga del codice precedentemente menzionata, il programma di controllo JDBC nativo è disponibile per il DriverManager da gestire. La seguente riga del codice richiede un oggetto Connection che utilizza l'URL di JDBC nativo:

Esempio: richiesta di un oggetto Connection

```
// Ottenere un collegamento che utilizza il programma di controllo JDBC nativo.

Connection c = DriverManager.getConnection("jdbc:db2:*local");
```

Il formato più semplice dell'URL JDBC è un elenco di tre valori separati da due punti. Il primo valore nell'elenco rappresenta il protocollo che è sempre `jdbc` per gli URL JDBC. Il secondo valore è il sottoprotocollo e vengono utilizzati `db2` o `db2iSeries` per specificare il programma di controllo JDBC nativo. Il terzo valore è il nome del sistema per stabilire il collegamento ad un sistema specifico. È possibile utilizzare due valori specifici per collegarsi al database locale. Essi sono *LOCAL e localhost (entrambi non sono sensibili al maiuscolo e al minuscolo). È inoltre possibile fornire un nome sistema specifico nel seguente modo:

```
Connection c =
  DriverManager.getConnection("jdbc:db2:rchasmop");
```

In questo modo si crea un collegamento al sistema rchasmop. Se il sistema cui si sta tentando di collegarsi è un sistema remoto (ad esempio, tramite Distributed Relational Database Architecture)è necessario utilizzare il nome sistema dall'indirizzario del database relazionale.

Nota: se non specificato, l'ID utente e la parola d'ordine correntemente utilizzata per il collegamento vengono anche utilizzati per stabilire la connessione al database.

Nota: il programma di controllo universale JDBC di IBM DB2 utilizza anche il sottoprotocollo db2. Per assicurarsi che il programma di controllo JDBC nativo gestisca l'URL, le applicazioni devono utilizzare l'URL jdbc:db2iSeries:xxxx invece di jdbc:db2:xxxx. Se l'applicazione non utilizza il programma di controllo nativo per accettare gli URL con sottoprotocollo db2, l'applicazione deve caricare la classe com.ibm.db2.jdbc.app.DB2iSeriesDrive invece di com.ibm.db2.jdbc.app.DB2Driver. Quando questa classe viene caricata, il programma di controllo nativo cessa di gestire gli URL che contengono il sottoprotocollo db2.

Proprietà

Il metodo DriverManager.getConnection acquisisce un URL a stringa singola indicato in precedenza ed è uno dei metodi su DriverManager per ottenere un oggetto Connection. Esiste anche un'altra versione del metodo del DriverManager.getConnection che acquisisce un ID utente e una parola d'ordine. Segue un esempio di questa versione:

Esempio: metodo DriverManager.getConnection che acquisisce un ID utente e una parola d'ordine

```
// Ottenere un collegamento che utilizza il programma di controllo JDBC nativo.

Connection c = DriverManager.getConnection("jdbc:db2:*local", "cujo", "newtiger");
```

La riga del codice tenta di collegarsi al database locale come utente cujo con la parola d'ordine newtiger indipendentemente dall'utente che sta eseguendo l'applicazione. Esiste anche una versione del metodo DriverManager.getConnection che acquisisce un oggetto java.util.Properties per consentire un'ulteriore personalizzazione. Segue un esempio:

Esempio: metodo DriverManager.getConnection che acquisisce un oggetto java.util.Properties

```
// Ottenere un collegamento che utilizza il programma di controllo JDBC nativo.

Properties prop = new java.util.Properties();
prop.put("user", "cujo");
prop.put("password","newtiger");
Connection c = DriverManager.getConnection("jdbc:db2:*local", prop);
```

Il codice è funzionalmente equivalente alla versione menzionata in precedenza che ha passato ID utente e parola d'ordine come parametri.

Consultare Proprietà di collegamento per un elenco completo delle proprietà di collegamento per il programma di controllo JDBC nativo.

Proprietà URL

Un altro modo per specificare le proprietà è inserirle in una lista sull'oggetto URL stesso. Ogni proprietà nell'elenco è separata da un punto e virgola e l'elenco deve essere nel formato `property name=property value`. Questa è una scorciatoia e non modifica significativamente il modo in cui viene eseguita l'elaborazione come mostra il seguente esempio:

Esempio: specifica delle proprietà URL

```
// Ottenere un collegamento che utilizza il programma di controllo JDBC nativo.

Connection c = DriverManager.getConnection("jdbc:db2:*local;user=cujo;password=newtiger");
```

Il codice è di nuovo funzionalmente equivalente agli esempi menzionati in precedenza.

Se viene specificato un valore della proprietà sia nell'oggetto delle proprietà che nell'oggetto URL, la versione URL ha la precedenza sull'altra. Segue un esempio:

Esempio: proprietà URL

```
// Ottenere un collegamento che utilizza il programma di controllo JDBC nativo.
Properties prop = new java.util.Properties();
prop.put("user", "someone");
prop.put("password","something");
Connection c = DriverManager.getConnection("jdbc:db2:*local;user=cujo;password=newtiger",
prop);
```

L'esempio utilizza l'ID utente e la parola d'ordine dalla stringa URL invece della versione nell'oggetto Proprietà. Ciò finisce per essere funzionalmente equivalente al codice menzionato in precedenza.

Esempio: ID utente e parola d'ordine non validi:

Questo esempio illustra come utilizzare la proprietà Connection in modalità di denominazione SQL.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio InvalidConnect.
//
// Questo programma utilizza la proprietà Connection nella modalità di denominazione SQL.
//
////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di programma di controllo JDBC IBM Developer per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
```

```
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////////////////////////////////////////////
import java.sql.*;
import java.util.*;

public class InvalidConnect {

    public static void main(java.lang.String[] args)
    {
        // Registrare il programma di controllo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: JDBC driver did not load.");
            System.exit(0);
        }

        // Tentare di ottenere un collegamento senza specificare un utente o una
        // parola d'ordine. Il tentativo ha esito positivo e il collegamento usa
        // lo stesso profilo utente tramite il quale è in esecuzione il lavoro.
        try {
            Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
            c1.close();
        } catch (SQLException e) {
            System.out.println("This test should not get into this exception path.");
            e.printStackTrace();
            System.exit(1);
        }

        try {
            Connection c2 = DriverManager.getConnection("jdbc:db2:*local",
                                                        "notvalid", "notvalid");
        } catch (SQLException e) {
            System.out.println("This is an expected error.");
            System.out.println("Message is " + e.getMessage());
            System.out.println("SQLSTATE is " + e.getSQLState());
        }

    }
}
```

Proprietà di collegamento del programma di controllo JDBC:

La tabella di seguito riportata contiene le proprietà di collegamento del programma di controllo JDBC e i relativi valori e descrizioni.

Proprietà	Valori	Significato
accesso	all, read call, read only	Questo valore consente di limitare il tipo di operazioni eseguibili con uno specifico collegamento. Il valore predefinito è 'all' e indica che il collegamento ha pieno accesso all'API JDBC. Il valore 'read call' consente al collegamento solo di effettuare query e di chiamare procedure memorizzate. I tentativi di aggiornamento del database tramite un'istruzione SQL vengono arrestati. Il valore 'read only' consente di limitare un collegamento alle sole query. Le chiamate a procedure memorizzate e le istruzioni di aggiornamento vengono arrestate.
commit automatico	true, false	Questo valore consente di definire l'impostazione del commit automatico del collegamento. Il valore predefinito è 'true' a meno che la proprietà di isolamento della transazione sia stata impostata su un valore diverso da 'none'. In quel caso, il valore predefinito è "false".
stile batch	2.0, 2.1	La specifica 2.1 JDBC definisce un altro metodo per gestire le eccezioni in aggiornamento batch. L'unità può adattarsi a ciascuna di esse. È necessario che il valore predefinito si applichi come stabilito nella specifica 2.0 JDBC.

Proprietà	Valori	Significato
dimensione blocco	0, 8, 16, 32, 64, 128, 256, 512	Questo valore indica il numero di righe selezionate insieme per una serie di risultati. Per una tipica elaborazione di solo inoltro di una serie di risultati, si ottiene un blocco di questa dimensione. Quindi non si ha accesso al database in quanto ogni riga viene elaborata dall'applicazione. Solo una volta raggiunta la fine del blocco il database richiede un altro blocco di dati. Questo valore viene utilizzato solo se la proprietà di blocco abilitato è impostata ″true″. Impostare la proprietà di dimensione del blocco su 0 equivale a impostare la proprietà blocco abilitato su ″false″. Il valore predefinito prevede l'utilizzo del blocco con una dimensione di 32. Questa è una decisione arbitraria ed è possibile che il valore predefinito in futuro venga modificato. Il blocco non viene utilizzato sulle serie di risultati che è possibile scorrere.
blocco abilitato	true, false	Questo valore è utilizzato per determinare se il collegamento utilizza il blocco sul richiamo riga della serie di risultati. È possibile che il blocco migliori in maniera significativa le prestazioni delle serie dei risultati. L'impostazione predefinita di questa proprietà è ″true″.
congelamento commit	false, true	Questo valore specifica se viene utilizzato ″commit hold″ quando viene richiamato il metodo connection.commit(). Quando viene utilizzato ″commit hold″, i cursori e le altre risorse di database non sono chiuse o liberate quando viene richiamato il commit. Il valore predefinito è ″false″.

Proprietà	Valori	Significato
congelamento cursore	true, false	Questo valore specifica se le serie di risultati restano aperte quando viene eseguito il commit di una transazione. Un valore di ″true″ indica che un'applicazione può accedere alle relative serie dei risultati una volta chiamato il commit. Un valore di ″false″ indica che il commit chiude ogni cursore aperto durante il collegamento. L'impostazione predefinita di questa proprietà è ″true″. Questa proprietà del valore funziona come valore predefinito per tutte le serie di risultati effettuate per il collegamento. Con l'aggiunta del supporto di conservabilità del cursore in JDBC 3.0, tale valore predefinito viene semplicemente sostituito se un'applicazione specifica in seguito una diversa capacità di conservabilità. Se si sta effettuando una migrazione a JDBC 3.0 da una versione precedente, ricordare che il supporto di conservabilità del cursore non era disponibile prima di JDBC 3.0. Nelle versioni precedenti, il valore predefinito di ″true″ era inviato al momento del collegamento, ma non ancora riconosciuto da JVM. La proprietà di conservabilità del cursore, quindi, non influirà sulla funzionalità del database fino a JDBC 3.0.
sensibilità del cursore	asensitive, sensitive	Specifica la sensibilità del cursore utilizzata dai cursori ResultSet.TYPE_SCROLL_SENSITIVE. Per impostazione predefinita, il programma di controllo JDBC nativo crea dei cursori non sensibili (asensitive) per i cursori ResultSet.TYPE_SCROLL_SENSITIVE.
troncamento dati	true, false	Questo valore specifica se è opportuno che il troncamento dei dati carattere generi avvisi ed eccezioni (true) o se i dati devono essere troncati senza che venga emesso alcun messaggio (false). Se il valore predefinito è ″true″, il troncamento dei dati dei campi carattere viene rispettato.
formato dati	julian, mdy, dmy, ymd, usa, iso, eur, jis	Questa proprietà consente di modificare la formattazione delle date.

Proprietà	Valori	Significato
separatore data	/(barra), - (trattino), . (punto), , (virgola), b	Questa proprietà consente di modificare il separatore della data. Essa è valida solo in combinazione con alcuni valori dateFormat (secondo le regole del sistema).
arrotondamento virgola mobile decimale	round half even, round half up, round down, round ceiling, round floor, round half down, round up, round half even	Questa proprietà specifica la modalità di arrotondamento che deve essere utilizzata dalle operazioni a virgola mobile decimale. Il valore predefinito è round half even.
separatore decimale	.(punto), ,(virgola)	Questa proprietà consente di modificare il separatore decimale.
mappa diretta	true, false	Questa proprietà specifica se le ottimizzazioni di associazione diretta del databsse verranno utilizzate quando si richiamano le serie di risultati dal database. Il valore predefinito è true.

Proprietà	Valori	Significato
elaborazione dell'uscita	true, false	Questa proprietà imposta un indicatore se è necessario che le istruzioni durante il collegamento effettuino l'elaborazione di uscita. L'utilizzo dell'elaborazione di uscita è un metodo per codificare le proprie istruzioni SQL in modo che siano generiche e simili per tutte le piatteforme, quindi il database legge le clausole di uscita e sostituisce la versione specifica del sistema appropriata per l'utente. Ciò è positivo, ma obbliga il sistema a un lavoro supplementare. Nel caso in cui si ha la certezza di utilizzare solo istruzioni SQL che già contengono una sintassi SQL i5/OS valida, si consiglia di impostare questo valore su "false" per migliorare le prestazioni. Il valore predefinito per questa proprietà è "true", dal momento che deve essere compatibile con la specifica JDBC (cioè, l'elaborazione di uscita è attiva per impostazione predefinita). Tale valore viene aggiunto a causa dell'insufficienza della specifica JDBC. È possibile soltanto impostare l'elaborazione di uscita su "off" nella classe Statement. Ciò funziona se si utilizzano istruzioni semplici. Si crea la propria istruzione, si arresta l'elaborazione di uscita e si avvia l'elaborazione delle istruzioni. Tuttavia, nel caso delle "prepared statement" e "callable statement", questo schema non funziona. Si fornisce l'istruzione SQL al momento della creazione della "prepared statement" o "callable statement" e non si modifica più. Quindi l'istruzione viene preparata in anticipo e modificare l'elaborazione di uscita dopo ciò non ha alcun significato. Disporre di questa proprietà di collegamento consente di evitare un ulteriore sovraccarico.
errori	basic, full	Questa proprietà consente di restituire il testo dell'errore del secondo livello di sistema nei messaggi dell'oggetto SQLException. Il valore predefinito è 'basic' e restituisce solo il testo del messaggio standard.

Proprietà	Valori	Significato
metadati estesi	true, false	La proprietà specifica se il programma di controllo deve richiedere i metadati estesi dal database. L'impostazione di questa proprietà su true aumenta l'accuratezza delle informazioni restituite dai seguenti metodi ResultSetMetaData: • getColumnLabel(int) • getSchemaName(int) • getTableName(int) • isReadOnly(int) • isSearchable(int) • isWriteable(int) L'impostazione di questa proprietà su true può ridurre le prestazioni perché richiede il richiamo di una quantità maggiore di informazioni dal database.
ignora avvertenze	Un elenco separato da virgole di stati SQL che devono essere ignorati.	Per impostazione predefinita, il programma di controllo JDBC nativo creerà internamente un oggetto java.sql.SQLWarning per ciascun avviso restituito dal database. Questa proprietà specifica un elenco di stati SQL per cui il programma di controllo JDBC nativo non deve creare degli oggetti di avviso. Ad esempio, un avviso con SQLSTATE 0100C viene creato ogni volta che una serie di risultati è restituita da una procedura memorizzata. Quest'avviso può essere ignorato senza rischi per migliorare le prestazioni delle applicazioni che richiamano le procedure memorizzate.

Proprietà	Valori	Significato
librerie	Un elenco di librerie, separate da spazi. (Un elenco di librerie può anche essere separata da due punti o da virgole.)	Questa proprietà consente di inserire un elenco di librerie nell'elenco di librerie del lavoro del server o di impostare la libreria predefinita specifica. La proprietà di denominazione (naming) influenza il funzionamento di questa proprietà. Nel caso dell'impostazione predefinita, in cui la proprietà di denominazione è impostata su sql, JDBC funziona come ODBC. L'elenco delle librerie non incide sul modo in cui si elabora il collegamento. Esiste una libreria predefinita per tutte le tabelle non qualificate. Per impostazione predefinita, tale libreria ha lo stesso nome del profilo utente che è collegato. Se la proprietà delle librerie viene specificata, la prima libreria nell'elenco diventa la libreria predefinita. Se si specifica una libreria predefinita sull'URL di collegamento (come in jdbc:db2:*local/mylibrary), essa sostituisce ogni valore in questa proprietà. Nel caso in cui la proprietà di denominazione (naming) sia impostato su ″system″, ogni libreria specificata per questa proprietà viene aggiunta alla porzione dell'elenco delle librerie destinata all'utente e viene eseguita una ricerca nell'elenco di librerie per definire i riferimenti della tabella non qualificati.

Proprietà	Valori	Significato
soglia lob	Qualsiasi valore inferiore a 500000	Questa proprietà indica al programma di controllo di inserire i valori effettivi nella memoria della serie di risultati invece dei localizzatori per le colonne lob se la colonna lob è più piccola rispetto alla dimensione della soglia. Questa proprietà funziona rispetto alla dimensione della colonna, non alla dimensione stessa dei dati lob. Ad esempio, se la colonna lob viene definita in modo che contenga fino a 1 MB per ogni lob, ma tutti i valori della colonna sono inferiori ai 500 KB, vengono ancora utilizzati i localizzatori. Notare che il limite della dimensione viene impostato così com'è per consentire di selezionare i blocchi di dati senza pericolo di ottenere blocchi di dati che crescano sempre rispetto alla dimensione di assegnazione massima di 16 MB. Con ampie serie di risultati, è ancora facile superare questo limite, che causa l'esito negativo della selezione. È necessario prestare attenzione alle modalità in cui la proprietà di dimensione blocco e questa proprietà interagiscono con la dimensione di un blocco di dati. Il valore predefinito è 0. I localizzatori vengono sempre utilizzati per i dati lob.
precisione massima	31, 63	Questo valore specifica la precisione massima utilizzata per i tipi decimali e numerici. Il valore predefinito è 31.
scala massima	0-63	Questo valore specifica la scala massima (numero di posizioni decimali alla destra della virgola decimale) restituita che è utilizzata dai tipi decimali e numerici. Questo valore va da 0 a massima precisione. Il valore predefinito è 31.
scala di divisione minima	0-9	Il valore specifica la scala minima di divisione (numero di posizioni decimali alla destra della virgola decimale) restituita per tipi di dati intermedi e di risultato. Il valore va da 0 a 9 in quanto non è possibile superare la scala massima. Se si specifica 0, non viene utilizzata la scala minima di divisione. Il valore predefinito è 0.

Proprietà	Valori	Significato
denominazione	sql, system	Questa proprietà consente di utilizzare la tradizionale sintassi di denominazione di System i oppure la sintassi di denominazione SQL standard. Con la denominazione di sistema si utilizza il carattere /(barra) per separare i valori della raccolta e delle tabelle, con la denominazione SQL si utilizza il carattere .(punto). L'impostazione di questo valore ha ramificazioni anche per quanto riguarda la libreria predefinita. Consultare le proprietà delle librerie di cui sopra per ulteriori informazioni su questo argomento. Il valore predefinito prevede una denominazione di tipo SQL.
parola d'ordine	qualsiasi valore	Questa proprietà consente di specificare una parola d'ordine per il collegamento. Questa proprietà non funziona se non viene specificata anche la proprietà utente. Queste proprietà consentono di effettuare i collegamenti al database come un utente diverso da quello che esegue il lavoro System i. Specificare le proprietà utente e parola d'ordine equivale a utilizzare il metodo di collegamento con la firma getConnection (String url, String userId, String password).
prefetch	true, false	Questa proprietà specifica se l'unità di controllo richiama i primi dati per la serie di risultati subito dopo l'elaborazione o se attende fino alla richiesta dei dati. Se il valore predefinito è 'true', i dati vengono sottoposti a prefetch. Per applicazioni che utilizzano l'unità JDBC nativa, ciò costituisce raramente un problema. La proprietà è destinata in primo luogo a un utilizzo interno con procedure memorizzate Java e con funzioni definite dall'utente per le quali sia importante che il motore del database non richiami dati dalle serie di risultati per conto dell'utente prima che vengano richiesti.

Proprietà	Valori	Significato
qaqqinilib	nome libreria	Questa proprietà specifica la libreria che contiene il file qaqqini da utilizzare. Un file qaqqini contiene tutti gli attributi che possono potenzialmente influenzare negativamente le prestazioni del motore di database DB2 for i5/OS.
obiettivo ottimizzazione query	1, 2	Questa proprietà specifica l'obiettivo che il server deve utilizzare con l'ottimizzazione delle query. Questa impostazione corrisponde all'opzione QAQQINI del server denominata OPTIMIZATION_GOAL. Sono possibili i seguenti valori: **1** Ottimizzare la query per il primo blocco di dati (*FIRSTIO) **2** Ottimizzare la query per l'intera serie di risultati (*ALLIO) Il valore predefinito è 2.
riutilizzo oggetti	true, false	Questa proprietà specifica se il programma di controllo tenta di riutilizzare alcuni tipi di oggetti dopo che l'utente li ha chiusi. Questo è un aggiornamento delle prestazioni. Il valore predefinito è ″true″.
formato ora	hms, usa, iso, eur, jis	Questa proprietà consente di modificare la formattazione dei valori dell'ora.
separatore ora	:(due punti), .(punto), ,(virgola), b	Questa proprietà consente di modificare il separatore dell'ora. Essa è valida solo in combinazione con alcuni valori timeFormat (secondo le regole del sistema).
traccia	true, false	Questa proprietà consente di attivare la traccia del collegamento. È possibile utilizzarla come semplice assistente del debug. Il valore predefinito è 'false' e non utilizza la traccia.
isolamento transazione	none, read committed, read uncommitted, repeatable read, serializable	Questa proprietà consente di impostare il livello di isolamento transazione per il collegamento. Non esiste alcuna differenza tra impostare questa proprietà su un livello specifico e specificare un livello sul metodo setTransactionIsolation nell'interfaccia Connection. Il valore predefinito per questa proprietà è ″none″, in quanto l'impostazione predefinita di JDBC è la modalità di commit automatico.

Proprietà	Valori	Significato
conversione binaria	true, false	È possibile utilizzare questa proprietà per forzare il programma di controllo JDBC in modo che tratti i valori di dati ″binary″ e ″varbinary″ come se fossero ″char″ e ″varchar″. Il valore predefinito per questa proprietà è ″false″, dove i dati binari non vengono trattati come dati carattere.
conversione ES	binary, character	Questo valore consente di selezionare il tipo di dati utilizzato dalle costanti ES nell'espressione SQL. L'impostazione binaria indica che le costanti ES utilizzano il tipo di dati BINARY. L'impostazione carattere indica che le costanti ES utilizzano il tipo di dati CHARACTER FOR BIT DATA. Il valore predefinito è carattere.
utilizzo inserimento blocco	true, false	Questa proprietà consente al programma di controllo JDBC nativo di andare nella modalità di inserimento blocchi per inserire i blocchi di dati nel database. Questa è una versione ottimizzata dell'aggiornamento batch. Tale modalità ottimizzata può essere utilizzata solo in applicazioni che assicurano di non violare certi limiti del sistema o errori di inserimento dati e potenzialmente danneggiare i dati. Le applicazioni che attivano questa proprietà si collegano soltanto al sistema locale nel tentativo di eseguire aggiornamenti sottoposti a batch. Esse utilizzano DRDA per stabilire collegamenti remoti in quanto non è possibile gestire l'inserimento del blocco su DRDA. È inoltre necessario che le applicazioni assicurino che PreparedStatement con un'istruzione di inserimento SQL e una clausola di valori rendano tutti i valori di inserimento dei parametri. Non sono consentite le costanti nell'elenco di valori. Questo è un requisito del motore di inserimento bloccato del sistema. Il valore predefinito è ″false″.

Proprietà	Valori	Significato
utente	qualsiasi valore	Questa proprietà consente di specificare un ID utente per il collegamento. Questa proprietà non funziona correttamente se non viene specificata anche la proprietà parola d'ordine. Queste proprietà consentono di effettuare i collegamenti al database come un utente diverso da quello che esegue il lavoro System i. Specificare le proprietà utente e parola d'ordine equivale a utilizzare il metodo di collegamento con la firma getConnection(String url, String userId, String password).

Utilizzo di DataSource con UDBDataSource:

Le interfacce DataSource consentono una flessibilità aggiuntiva nell'utilizzo dei programmi di controllo JDBC (Java Database Connectivity).

È possibile suddividere l'utilizzo di DataSource in due fasi:

- **Disposizione**

  La disposizione è una fase di impostazione che si verifica prima dell'effettiva esecuzione di un'applicazione JDBC. La disposizione di solito implica l'impostazione di un DataSource in modo che abbia proprietà specifiche e il relativo collegamento in un servizio dell'indirizzario tramite l'utilizzo di JNDI (Java Naming and Directory Interface). Il servizio dell'indirizzario è più comunemente LDAP (Lightweight Directory Access Protocol), ma è possibile che sia CORBA (Common Object Request Broker Architecture) Object Service, RMI (Remote Method Invocation) Java o il file system sottostante.
- **Utilizzo**

  Separando la disposizione dall'utilizzo del tempo di esecuzione di DataSource, è possibile che molte applicazioni riutilizzino l'impostazione di DataSource. Modificando alcuni aspetti della disposizione, tutte le applicazioni che utilizzano DataSource acquisiscono automaticamente le modifiche.

Nota: tenere presente che l'utilizzo di RMI può essere un'attività complessa. Prima di scegliere RMI come soluzione, assicurarsi di conoscere tutte le eventuali ramificazioni di questa operazione.

Un vantaggio dei DataSource è che consentono ai programmi di controllo JDBC di funzionare a favore dell'applicazione senza avere un impatto direttamente sul processo di sviluppo dell'applicazione. Per ulteriori informazioni, consultare:

- "Utilizzo del supporto DataSource per la creazione di lotti di oggetti" a pagina 132
- "Creazione di lotti di istruzioni basati su DataSource" a pagina 136
- "Transazioni distribuite JDBC" a pagina 82

UDBDataSourceBind

Il programma "Esempio: creazione di UDBDataSource e suo collegamento alla JNDI" a pagina 59 è un esempio di creazione di un UDBDataSource e del suo collegamento alla JNDI. Questo programma completa tutte le attività di base necessarie. In pratica esso crea istanze per un oggetto UDBDataSource, imposta le proprietà su tale oggetto, richiama un contesto JNDI e collega l'oggetto ad un nome con il contesto JNDI.

Il codice del tempo di sviluppo è specifico del fornitore. È necessario che l'applicazione importi l'implementazione DataSource specifica che intende gestire. Nell'elenco di importazione, viene importata la classe UDBDataSource qualificata dal pacchetto. La parte meno nota dell'applicazione è il lavoro effettuato con JNDI (ad esempio, il richiamo dell'oggetto Context e la chiamata da collegare). Per informazioni aggiuntive, consultare JNDI di Sun Microsystems, Inc.

Una volta eseguito e completato con esito positivo questo programma, esiste una nuova voce nel servizio dell'indirizzario JNDI denominata SimpleDS. Tale voce si trova nell'ubicazione specificata dal contesto JNDI. L'implementazione DataSource viene ora disposta. Un programma applicativo può utilizzare questo DataSource per richiamare collegamenti al database e il lavoro correlato a JDBC.

UDBDataSourceUse

Il programma "Esempio: acquisizione di un contesto iniziale prima di collegare UDBDataSource" a pagina 60 è un esempio di un'applicazione JDBC che utilizza l'applicazione precedentemente distribuita.

L'applicazione JDBC ottiene un contesto iniziale come è accaduto prima di collegare UDBDataSource nel precedente esempio. Il metodo di ricerca viene quindi utilizzato su quel contesto per restituire un oggetto di tipo DataSource affinché l'applicazione lo utilizzi.

Nota: l'applicazione del tempo di esecuzione è interessata solo ai metodi dell'interfaccia DataSource, quindi non esiste alcun bisogno che sia consapevole della classe di implementazione. Ciò rende l'applicazione trasferibile.

Si supponga che UDBDataSourceUse è un'applicazione complessa che esegue un'operazione di ampia portata all'interno della propria organizzazione. Si dispone di una dozzina o più applicazioni simili di grande dimensione all'interno della propria applicazione. È necessario modificare il nome di uno dei sistemi nella propria rete. Eseguendo uno strumento di disposizione e modificando una singola proprietà UDBDataSource, si potrà ottenere questa nuova funzionalità in tutte le proprie applicazioni senza modificare il codice per esse. Uno dei benefici dei DataSource è che consentono di consolidare le informazioni di impostazione del sistema. Un altro vantaggio maggiore è che consentono ai programmi di controllo di implementare una funzionalità non visibile all'applicazione, come creazione di lotti di collegamenti, creazione di lotti di istruzioni e supporto per transazioni distribuite.

Una volta analizzato attentamente UDBDataSourceBind e UDBDataSourceUse, è possibile che ci si chieda in che modo l'oggetto DataSource conosca le loro attività. Non esiste alcun codice per specificare un sistema, un ID utente o una parola d'ordine in uno di questi programmi. La classe UDBDataSource ha dei valori predefiniti per tutte le proprietà: per impostazione predefinita, si collega al System i locale con il profilo utente e la parola d'ordine dell'applicazione in esecuzione. Se si intendeva essere certi che il collegamento fosse avvenuto, invece, con il profilo utente cujo, sarebbe stato possibile ottenere ciò in due modi:

- impostando l'ID utente e la parola d'ordine come proprietà DataSource. Per informazioni su come utilizzare questa tecnica, consultare "Esempio: creazione di UDBDataSourceBind ed impostazione delle proprietà DataSource" a pagina 59.
- utilizzando il metodo getConnection DataSource che acquisisce un ID utente e una parola d'ordine durante il tempo di esecuzione. Per informazioni su come utilizzare questa tecnica, consultare "Esempio: creazione di un UDBDataSource e acquisizione di un ID utente e una parola d'ordine" a pagina 61"Esempio: creazione di un UDBDataSource e acquisizione di un ID utente e una parola d'ordine" a pagina 61.

Esistono alcune proprietà che è possibile specificare per UDBDataSource così come esistono proprietà che è possibile specificare per i collegamenti creati con DriverManager. Per un elenco delle proprietà supportate per il programma di controllo JDBC nativo, consultare "Proprietà DataSource" a pagina 62.

Anche se questi elenchi sono simili, non è certo che siano simili nei release futuri. Si consiglia di iniziare la codifica nell'interfaccia DataSource.

Nota: il programma di controllo JDBC nativo ha anche altre due implementazioni DataSource: DB2DataSource e DB2StdDataSource. Queste implementazioni sono state dichiarate obsolete e non ne è consigliato l'utilizzo diretto. Queste implementazioni potrebbero essere rimosse in un release futuro.

Esempio: creazione di UDBDataSource e suo collegamento alla JNDI:

Questo è un esempio di come creare un UDBDataSource e collegarlo alla JNDI.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Importare i pacchetti richiesti. Al momento dello sviluppo,
// la classe specifica del programma di controllo JDBC che implementa
// DataSource deve essere importata.
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Creare un nuovo oggetto UDBDataSource e fornirgli
        // una descrizione.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource");

        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Collegare l'oggetto UDBDataSource appena creato al
        // servizio indirizzario JNDI, fornendogli un nome
        // utilizzabile per ricercare di nuovo questo oggetto
        // successivamente.
        ctx.rebind("SimpleDS", ds);
    }
}
```

Esempio: creazione di UDBDataSourceBind ed impostazione delle proprietà DataSource:

Questo è un esempio di come creare UDBDataSource e impostare l'ID utente e la parola d'ordine come proprietà DataSource.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Importare i pacchetti richiesti. Al momento dello sviluppo,
// la classe specifica del programma di controllo JDBC che implementa
// DataSource deve essere importata.
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind2
{
    public static void main(java.lang.String[] args)
    throws Exception
```

```
    {
        // Creare un nuovo oggetto UDBDataSource e fornirgli
        // una descrizione.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource "  +
                          "with cujo as the default " +
                          "profile to connect with.");

        // Fornire un ID utente e una parola d'ordine da utilizzare
        // per le richieste di collegamento.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Collegare l'oggetto UDBDataSource appena creato al
        // servizio indirizzario JNDI, fornendogli un nome
        // utilizzabile per ricercare di nuovo questo oggetto
        // successivamente.
        ctx.rebind("SimpleDS2", ds);
    }
}
```

Esempio: acquisizione di un contesto iniziale prima di collegare UDBDataSource:

Il seguente esempio ottiene un contesto iniziale prima di collegare UDBDataSource. Il metodo di ricerca viene quindi utilizzato su quel contesto per restituire un oggetto di tipo DataSource affinché l'applicazione lo utilizzi.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Importare i pacchetti richiesti. Non esistono codici
// specifici del programma di controllo necessari nelle applicazioni
// del tempo di esecuzione.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Richiamare l'oggetto UDBDataSource collegato usando il nome
        // con cui è stato precedentemente collegato. Nel tempo di esecuzione
        // viene usata solo l'interfaccia DataSource, quindi non c'è bisogno
        // di convertire l'oggetto nella classe di implementazione UDBDataSource.
        // (Non è necessario conoscere qual è la classe di
        // implementazione. Solo il nome JNDI logico è
        // necessario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una volta ottenuto il DataSource, può essere usato per stabilire
        // un collegamento. Questo oggetto Connection è dello stesso tipo di
        // quello restituito se viene utilizzato l'approccio DriverManager
        // per stabilire il collegamento. Quindi, quanto viene riportato da
```

```
        // questo punto in poi è esattamente uguale a qualsiasi altra
        // applicazione JDBC.
        Connection connection = ds.getConnection();

        // Il collegamento può essere usato per creare oggetti Statement e
        // aggiornare il database o elaborare le query come segue.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // Il collegamento viene chiuso prima del termine dell'applicazione.
        connection.close();
    }
}
```

Esempio: creazione di un UDBDataSource e acquisizione di un ID utente e una parola d'ordine:

Questo è un esempio di come creare UDBDataSource e utilizzare il metodo getConnection per ottenere un ID utente e una parola d'ordine al tempo di esecuzione.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/// Importare i pacchetti richiesti. Non esiste codice
// specifico del programma di controllo necessario nelle applicazioni
// del tempo di esecuzione.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Richiamare l'oggetto UDBDataSource collegato usando il nome
        // con cui è stato precedentemente collegato. Nel tempo di esecuzione
        // viene usata solo l'interfaccia DataSource, quindi non c'è bisogno
        // di convertire l'oggetto nella classe di implementazione UDBDataSource.
        // (Non è necessario conoscere qual è la classe di
        // implementazione. Solo il nome JNDI logico è
        // necessario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una volta ottenuto il DataSource, può essere usato per stabilire
        // un collegamento. Il profilo utente cujo e la parola d'ordine newtiger
        // vengono utilizzati per creare il collegamento al posto dell'ID utente e
        // della parola d'ordine predefiniti per il DataSource.
        Connection connection = ds.getConnection("cujo", "newtiger");

        // Il collegamento può essere usato per creare oggetti Statement e
        // aggiornare il database o elaborare le query come segue.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }
```

```
        // Il collegamento viene chiuso prima del termine dell'applicazione.
        connection.close();
    }
}
```

Proprietà DataSource:

Per ciascuna proprietà di collegamento del programma di controllo JDBC, esiste un corrispondente metodo di origine dati. Questa tabella contiene le proprietà di origine dati valide.

Per alcune proprietà, è possibile fare riferimento alla corrispondente proprietà di collegamento del programma di collegamento corrispondente per ulteriori informazioni.

Metodo Set (tipo di dati)	Valori	Descrizione
setAccess(Stringa)	″all″, ″read call″, ″read only″	Fare riferimento alla proprietà del collegamento di accesso.
setAutoCommit(booleano)	″true″, ″false″	Fare riferimento alla proprietà del collegamento di commit automatico.
setBatchStyle(Stringa)	″2.0″, ″2.1″	Fare riferimento alla proprietà di collegamento stile batch.
setBlockSize(int)	″0″, ″8″, ″16″, ″32″, ″64″, ″128″, ″256″, ″512″	Fare riferimento alla proprietà di collegamento dimensione blocco.
setCommitHold(booleano)	″true″, ″false″	Fare riferimento alla proprietà di collegamento congelamento commit.
setCursorHold(booleano)	″true″, ″false″	Fare riferimento alla proprietà di collegamento congelamento cursore.
setCursorSensitivity(Stringa)	″sensitive″, ″asensitive″	Fare riferimento alla proprietà di collegamento sensibilità del cursore.
setDataTruncation(booleano)	″true″, ″false″	Fare riferimento alla proprietà di collegamento troncamento dati.
setDatabaseName(Stringa)	Qualsiasi nome	Questa proprietà specifica il database cui il DataSource tenta di collegarsi. Il valore predefinito è *LOCAL. È necessario che il nome database esista nell'indirizzario del database relazionale sul sistema che esegue l'applicazione o sia il valore speciale *LOCAL o localhost per specificare il sistema locale.
setDataSourceName(Stringa)	Qualsiasi nome	Questa proprietà consente di inoltrare un nome JNDI (Java Naming and Directory Interface) ConnectionPoolDataSource per supportare il lotto di collegamenti.
setDateFormat(Stringa)	″julian″, ″mdy″, ″dmy″, ″ymd″, ″usa″, ″iso″, ″eur″, ″jis″	Fare rierimento alla proprietà di collegamento formato data.
setDateSeparator(Stringa)	″/″, ″-″, ″.″, ″,″, ″b″	Fare riferimento alla proprietà di collegamento separatore data.
setDecimalSeparator(Stringa)	″.″, ″,″	Fare riferimento alla proprietà di collegamento separatore decimale.
setDecfloatRoundingMode(Stringa)	″round half even″, ″round half up″, ″round down″, ″round ceiling″, ″round floor″, ″round half down″, ″round up″, ″round half even″	Fare riferimento alla proprietà di collegamento modalità arrotondamento virgola mobile decimale.

Metodo Set (tipo di dati)	Valori	Descrizione
setDescription(Stringa)	Qualsiasi nome	Questa proprietà consente l'impostazione di questa descrizione di testo dell'oggetto DataSource.
setDirectMap(booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento mappa diretta.
setDoEscapeProcessing(booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento elaborazione dell'uscita.
setFullErrors(booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento errori.
setIgnoreWarnings(Stringa)	Elenco separato da virgole di SQLSTATE.	Fare riferimento alla proprietà di collegamento ignora avvertenze.
setLibraries(Stringa)	Una lista di librerie, separata dagli spazi	Fare riferimento alla proprietà di collegamento librerie.
setLobThreshold(int)	Qualsiasi valore inferiore a 500000	Fare riferimento alla proprietà di collegamento soglia lob.
setLoginTimeout(int)	Qualsiasi valore	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setMaximumPrecision(int)	31, 63	Fare riferimento alla proprietà di collegamento precisione massima.
setMaximumScale(int)	0-63	Fare riferimento alla proprietà di collegamento scala massima.
setMinimumDivideScale(int)	0-9	Fare riferimento alla proprietà di collegamento scala di divisione minima.
setNetworkProtocol(int)	Qualsiasi valore	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setPassword(Stringa)	Qualsiasi stringa	Fare riferimento alla proprietà di collegamento parola d'ordine.
setPortNumber(int)	Qualsiasi valore	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setPrefetch(booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento prefetch.
setQaqqinilib(String)	nome libreria	Fare riferimento alla proprietà di collegamento qaqqinilib
setQueryOptimizeGoal(Stringa)	1, 2	Fare riferimento alla proprietà di collegamento obiettivo ottimizzazione query.
setReuseObjects(booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento riutilizzo degli oggetti.
setServerName(Stringa)	Qualsiasi nome	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setSystemNaming(booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento denominazione.
setTimeFormat(Stringa)	"hms", "usa", "iso", "eur", "jis"	Fare riferimento alla proprietà di collegamento formato ora.

Metodo Set (tipo di dati)	Valori	Descrizione
setTimeSeparator(Stringa)	":", ".", ",", "b"	Fare riferimento alla proprietà di collegamento separatore ora.
setTransactionIsolationLevel(Stringa)	"none", "read committed", "read uncommitted", "repeatable read", "serializable"	Fare riferimento alla proprietà di collegamento isolamento transazione.
setTranslateBinary(Booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento conversione binaria.
setUseBlockInsert(booleano)	"true", "false"	Fare riferimento alla proprietà di collegamento utilizzo inserimento blocco.
setUser(Stringa)	qualsiasi valore	Fare riferimento alla proprietà di collegamento utente.

Proprietà JVM per JDBC

Alcune impostazioni utilizzate dal programma di controllo JDBC nativo non possono essere impostate utilizzando una proprietà di collegamento. È necessario definire queste impostazioni per la JVM in cui è in esecuzione il programma di controllo JDBC nativo. Queste impostazioni vengono utilizzate per tutti i collegamenti creati dal programma di controllo JDBC nativo.

Il programma di controllo nativo riconosce le seguenti proprietà JVM:

Proprietà	Valori	Significato
jdbc.db2.job.sort.sequence	valore predefinito = *HEX	Impostando questa proprietà su true il programma di controllo JDBC nativo utilizzerà la Sequenza di ordinamento lavori dell'utente che ha avviato il lavoro invece del valore predefinito *HEX. Impostando questa proprietà su un altro valore o non impostandola, JDBC continuerà ad utilizzare il valore predefinito *HEX. Tenere presente le conseguenze di ciò. Quando i collegamenti JDBC inoltrano differenti profili utenti sulle richieste di collegamento, viene utilizzata per tutti i collegamenti la sequenza di ordinamento del profilo utente che ha avviato il server. Questo è un attributo di ambiente che viene impostato all'avvio e non un attributo di collegamento dinamico.
jdbc.db2.trace	1 o error = Traccia informazioni errore 2 o info = Traccia informazioni e informazioni errore 3 o verbose = Traccia informazioni dettagliate e informazioni errore 4 o all o true = Traccia di tutte le informazioni possibili	Questa proprietà attiva la traccia per il programma di controllo JDBC. È opportuno utilizzarla per la documentazione di un problema.

Proprietà	Valori	Significato
jdbc.db2.trace.config	stdout = Informazioni traccia inviate a stdout (valore predefinito) usrtrc = Informazioni traccia inviate a una traccia utente. Per ottenere le informazioni sulla traccia è possibile utilizzare il comando CL Dump traccia utente (DMPUSRTRC). file://<percorsofile> = Informazioni traccia inviate a un file. Se il nome del file contiene ″%j″, ″%j″ verrà sostituito con il nome lavoro. Un esempio di <percorsofile> è /tmp/jdbc.%j.trace.txt.	Questa proprietà viene utilizzata per specificare la destinazione dell'emissione della traccia.

Interfaccia DatabaseMetaData per IBM Developer Kit per Java

L'interfaccia DatabaseMetaData è implementata dal programma di controllo JDBC IBM Developer Kit per Java per fornire informazioni sulle sue origini di dati sottostanti. Essa viene utilizzata principalmente dagli strumenti e server dell'applicazione per stabilire come interagire con una specifica origine dati. Le applicazioni possono inoltre utilizzare i metodi DatabaseMetaData per ottenere informazioni su una origine dati, ma ciò avviene meno frequentemente.

L'interfaccia DatabaseMetaData include oltre 150 metodi che è possibile suddividere in categorie in base ai tipi di informazioni che forniscono. Questi sono descritti di seguito. L'interfaccia DatabaseMetaData contiene inoltre oltre 40 campi che sono costanti utilizzate come valori di ritorno per vari metodi DatabaseMetaData.

Consultare di seguito le sezioni relative alle modifiche in JDBC 3.0 e in JDBC 4.0 per informazioni sulle modifiche apportate ai metodi nell'interfaccia DatabaseMetaData.

Creazione di un oggetto DatabaseMetaData

Un oggetto DatabaseMetaData viene creato con il metodo Connection getMetaData. Una volta creato l'oggetto, è possibile utilizzarlo per trovare dinamicamente informazioni sull'origine dati sottostante. Il seguente esempio crea un oggetto DatabaseMetaData e lo utilizza per determinare il numero massimo di caratteri consentiti per un nome di tabella:

Esempio: creazione di un oggetto DatabaseMetaData

```
// con è un oggetto Connection
DatabaseMetaData dbmd = con.getMetadata();
int maxLen = dbmd.getMaxTableNameLength();
```

Richiamo di informazioni generali

Alcuni metodi DatabaseMetaData vengono utilizzati sia per reperire dinamicamente informazioni generali su una origine dati per ottenere dettagli sulla relativa implementazione. Alcuni di questi metodi includono quanto segue:

- getURL
- getUserName
- getDatabaseProductVersion, getDriverMajorVersion e getDriverMinorVersion
- getSchemaTerm, getCatalogTerm e getProcedureTerm
- nullsAreSortedHigh e nullsAreSortedLow
- usesLocalFiles e usesLocalFilePerTable
- getSQLKeywords

Come determinare il supporto delle funzioni

È possibile utilizzare un ampio gruppo di metodi DatabaseMetaData per stabilire se una funzione o un gruppo di funzioni date sono supportate dall'unità o dall'origine dati sottostante. Oltre a questo, esistono metodi che descrivono quale livello del supporto viene fornito. Alcuni di questi metodi che descrivono il supporto per funzioni individuali includono quanto segue:

- supportsAlterTableWithDropColumn
- supportsBatchUpdates
- supportsTableCorrelationNames
- supportsPositionedDelete
- supportsFullOuterJoins
- supportsStoredProcedures
- supportsMixedCaseQuotedIdentifiers

I metodi per descrivere un livello di supporto della funzione includono quanto segue:

- supportsANSI92EntryLevelSQL
- supportsCoreSQLGrammar

Limiti dell'origine dati

Un altro gruppo di metodi fornisce i limiti imposti da una origine dati assegnata. Alcuni metodi di questa categoria includono quanto segue:

- getMaxRowSize
- getMaxStatementLength
- getMaxTablesInSelect
- getMaxConnections
- getMaxCharLiteralLength
- getMaxColumnsInTable

I metodi in questo gruppo restituiscono il valore del limite come numero intero. Un valore di ritorno di zero indica che non esiste alcun limite o che il limite è sconosciuto.

Oggetti SQL e relativi attributi

Alcuni metodi DatabaseMetaData forniscono informazioni sugli oggetti SQL che popolano una origine dati assegnata. Tali metodi possono determinare gli attributi degli oggetti SQL. Essi restituiscono, inoltre, gli oggetti ResultSet in cui ogni riga descrive un oggetto particolare. Ad esempio, il metodo getUDT restituisce un oggetto ResultSet in cui esiste una riga per ogni UDT (user-defined table - tabella definita dall'utente) che è stata definita nell'origine dati. Esempi di questa categoria includono quanto segue:

- getSchemas e getCatalogs
- getTables
- getPrimaryKeys
- getProcedures e getProcedureColumns
- getUDT

Supporto transazione

Un esiguo gruppo di metodi fornisce informazioni sulla semantica della transazione supportata dall'origine dati. Esempi di questa categoria includono quanto segue:

- supportsMultipleTransactions

- getDefaultTransactionIsolation

Consultare "Esempio: restituzione di un elenco di tabelle utilizzando l'interfaccia DatabaseMetaData IBM Developer Kit per Java" a pagina 70 per un esempio di come utilizzare l'interfaccia DatabaseMetaData.

Modifiche in JDBC 3.0

Esistono modifiche ai valori di ritorno per alcuni dei metodi in JDBC 3.0. I seguenti metodi sono stati aggiornati in JDBC 3.0 per aggiungere campi ai ResultSet che essi restituiscono.

- getTables
- getColumns
- getUDT
- getSchemas

Nota: se si sviluppa un'applicazione utilizzando JDK (Java Development Kit 1.4, è possibile riconoscere che esiste un certo numero di colonne restituite durante la verifica. Si scrive la propria applicazione e si attende di accedere a tutte queste colonne. Tuttavia, se l'applicazione viene progettata anche per l'esecuzione sui precedenti release JDK, essa riceve SQLException quando tenta di accedere a questi campi che non esistono in release JDK precedenti. "Esempio: utilizzo dei ResultSet di metadati che hanno più di una colonna" a pagina 70 è un esempio di come è possibile scrivere un'applicazione per gestire diversi release JDK.

Modifiche in JDBC 4.0

In V6R1, la CLI (command language interface) sta cambiando l'implementazione delle API MetaData per richiamare anche le procedure memorizzate SYSIBM. Per questa ragione, i metodi MetaData JDBC utilizzeranno le procedure SYSIBM direttamente su V6R1, indipendentemente dal livello JDK. A causa di questa modifica, si noteranno le seguenti differenze:

- Il programma di controllo JDBC nativo consentiva precedentemente l'utilizzo di `localhost` come nome di catalogo per la maggior parte dei metodi. In JDBC 4.0, il programma di controllo JDBC nativo non restituirà alcuna informazione, se viene specificato `localhost`.
- Il programma di controllo JDBC nativo ha sempre restituito una serie di risultati vuota quando il parametro nullable di getBestRowIdentifier era impostato su false. Questo verrà rettificato in modo da restituire il risultato corretto.
- I valori restituiti da getColumns per le colonne BUFFER_LENGTH, SQL_DATA_TYPE e SQL_DATETIME_SUB possono essere differenti. Questi valori non devono essere utilizzati in un'applicazione JDBC perché la specifica JDBC definisce queste colonne come ″unused″ (non utilizzate).
- Il programma di controllo JDBC nativo riconosce i parametri di tabella e di schema di getCrossReference, getExportedKeys, getImportedKeys e getPrimaryKeys come ″pattern″ (modello). Ora, i parametri di tabella e di schema devono corrispondere al nome così com'è memorizzato nel database.
- Le viste utilizzate per implementare le viste definite dal sistema erano precedentemente descritte da getTables() come SYSTEM TABLES. Per essere congruenti con la famiglia DB2, queste viste sono ora descritte come VIEWS.
- I nomi di colonna restituiti da getProcedures sono differenti. Questi nomi di colonna non sono definiti dalla specifica JDBC 4.0. Inoltre, la colonna remarks per getProcedures precedentemente restituiva "" se non erano disponibili delle informazioni. Ora restituisce invece un valore nullo.

Tabella 3. Nomi di colonna restituiti da getProcedures in JDBC 4.0

Numero colonna	Nome precedente	Nome sotto JDBC 4.0
4	RESERVED1	NUM_INPUT_PARAMS

Tabella 3. Nomi di colonna restituiti da getProcedures in JDBC 4.0 (Continua)

Numero colonna	Nome precedente	Nome sotto JDBC 4.0
5	RESERVED2	NUM_OUTPUT_PARAMS
6	RESERVED3	NUM_RESULT_SETS

- Alcuni valori restituiti da getProcedureColumns per vari tipi di dati sono cambiati, come qui di seguito indicato:

Tabella 4. Valori restituiti da getProcedureColumns in JDBC 4.0

Tipo di dati	Colonna	Valore precedente	Valore in JDBC 4.0
ALL	Remarks	""	null
INTEGER	Length	Null	4
SMALLINT	Length	Null	2
BIGINT	dataType	19 (non corretto)	-5
BIGINT	Length	Null	8
DECIMAL	Length	Null	precisione + scala
NUMERIC	Length	Null	precisione + scala
DOUBLE	TypeName	DOUBLE PRECISION	DOUBLE
DOUBLE	Length	Null	8
FLOAT	TypeName	DOUBLE PRECISION	DOUBLE
FLOAT	Length	Null	8
REAL	Length	Null	4
DATE	Precision	null	10
DATE	Length	10	6
TIME	Precision	null	8
TIME	Length	8	6
TIME	Scale	null	0
TIMESTAMP	Precision	null	26
TIMESTAMP	Length	26	16
TIMESTAMP	Scale	null	6
CHAR	typeName	CHARACTER	CHAR
CHAR	Precision	null	come la lunghezza
VARCHAR	typeName	CHARACTER VARYING	VARCHAR
VARCHAR	Precision	null	come la lunghezza
CLOB	dataType	null (non corretto)	2005
CLOB	typeName	CHARACTER LARGE OBJECT	CLOB
CLOB	Precision	null	come la lunghezza
CHAR FOR BIT DATA	dataType	1 (CHAR)	-2 (BINARY)
CHAR FOR BIT DATA	typeName	CHARACTER	CHAR () FOR BIT DATA
CHAR FOR BIT DATA	Precision	null	come la lunghezza
BLOB	dataType	null (non corretto)	2004
BLOB	typeName	BINARY LARGE OBJECT	BLOB
BLOB	Precision	null	come la lunghezza

Tabella 4. Valori restituiti da getProcedureColumns in JDBC 4.0 (Continua)

Tipo di dati	Colonna	Valore precedente	Valore in JDBC 4.0
DATALINK	dataType	null (non corretto)	70
DATALINK	Precision	null	come la lunghezza
VARCHAR FOR BIT DATA	dataType	12 (VARCHAR)	-3 (VARBINARY)
VARCHAR FOR BIT DATA	typeName	CHARACTER VARYING	VARCHAR () FOR BIT DATA
VARCHAR FOR BIT DATA	Precision	null	come la lunghezza

Limitazione sulle procedure memorizzate di SOLA LETTURA

Il JDBC nativo supporta la proprietà `access = read only`. Questa proprietà è applicata a livello di JDBC. Per questa ragione, le procedure MetaData dovrebbero continuare a funzionare, se viene impostata questa proprietà. È tuttavia possibile utilizzare il programma di controllo JDBC nativo da una procedura memorizzata del database definita come di SOLA LETTURA. In questo caso, le procedure MetaData non funzioneranno.

Nuovo metodo: getClientInfoProperties()

Il metodo getClientInfoProperties richiama un elenco delle proprietà di informazioni sul client supportate dal programma di controllo. Ciascuna proprietà di informazioni sul client è memorizzata in un registro speciale SQL. Il programma di controllo JDBC nativo restituirà una serie di risultati con le seguenti informazioni:

Tabella 5. Informazioni restituite dal metodo getClientInfoProperties

Nome	Lunghezza massima	Valore predefinito	Descrizione
ApplicationName	255	spazio	Il nome dell'applicazione che sta attualmente utilizzando il collegamento
ClientUser	255	spazio	Il nome dell'utente per cui l'applicazione che sta utilizzando il collegamento sta eseguendo il lavoro. Questo può non essere uguale al nome utente che è stato utilizzato per stabilire il collegamento
ClientHostname	255	spazio	Il nome host del computer su cui è in esecuzione l'applicazione che sta utilizzando il collegamento
ClientAccounting	255	spazio	Informazioni sull'account.

I registri speciali SQL corrispondenti alle proprietà di informazioni sul client sono i seguenti:

Tabella 6. Registri speciali SQL

Nome	Registro speciale SQL
ApplicationName	CURRENT CLIENT_APPLNAME
ClientUser	CURRENT CLIENT_USERID
ClientHostname	CURRENT CLIENT_WRKSTNNAME

| *Tabella 6. Registri speciali SQL (Continua)*

Nome	**Registro speciale SQL**
ClientAccounting	CURRENT CLIENT_ACCTNG

| Le clientInfoProperties possono essere impostate utilizzando il metodo setClientInfo dell'oggetto
| Connection.

Concetti correlati

"ResultSet" a pagina 114
L'interfaccia ResultSet fornisce l'accesso ai risultati generati eseguendo delle query. Concettualmente, è possibile concepire i dati di un ResultSet come una tabella con un numero specifico di colonne e un numero specifico di righe. Per impostazione predefinita, le righe della tabella vengono richiamate in sequenza. All'interno di una riga, è possibile accedere ai valori della colonna in qualsiasi ordine.

Esempio: restituzione di un elenco di tabelle utilizzando l'interfaccia DatabaseMetaData IBM Developer Kit per Java:

Questo esempio mostra come restituire una lista di tabelle.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Collegarsi al server.
Connection c = DriverManager.getConnection("jdbc:db2:mySystem");

// Richiamare i meta dati database dal collegamento.
DatabaseMetaData dbMeta = c.getMetaData();

// Richiamare un elenco di tabelle che corrispondono a questi criteri.
String catalog = "myCatalog";
String schema = "mySchema";
String table  = "myTable%"; // % indica il modello di ricerca
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"}:
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterare attraverso il ResultSet per richiamare i valori.

// Chiudere il collegamento.
c.close():
```

Esempio: utilizzo dei ResultSet di metadati che hanno più di una colonna:

Questo rappresenta un esempio del modo in cui utilizzare i Resultset di metadati che possiedono più di una colonna.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////////////////////////////////////////////
//
// Esempio SafeGetUDTs. Questo programma mostra un modo per gestire i
// ResultSet metadati che hanno più colonne in JDK 1.4 rispetto
// ai release precedenti.
//
// Sintassi del comando:
//    java SafeGetUDTs
//
//////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di programma di controllo JDBC IBM Developer per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
```

```
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class SafeGetUDTs {

    public static int jdbcLevel;

    // Nota: il blocco static viene eseguito prima che inizi main.
    // Quindi, è presente access in jdbcLevel in
    // main.
    {
        try {
            Class.forName("java.sql.Blob");

            try {
                Class.forName("java.sql.ParameterMetaData");
                // Rilevata un'interfaccia JDBC 3.0. Deve supportare JDBC 3.0.
                jdbcLevel = 3;
            } catch (ClassNotFoundException ez) {
                // Impossibile trovare la classe JDBC 3.0 ParameterMetaData.
                // Deve essere in esecuzione sotto una JVM con il solo
                // supporto di JDBC 2.0.
                jdbcLevel = 2;
            }

        } catch (ClassNotFoundException ex) {
            // Impossibile trovare la classe JDBC 2.0 Blob. Deve essere in
            // esecuzione sotto una JVM con il solo supporto di JDBC 1.0.
            jdbcLevel = 1;
        }
    }

    // Punto di immissione del programma.
    public static void main(java.lang.String[] args)
    {
        Connection c = null;

        try {
            // Richiamare l'unità registrata.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            c = DriverManager.getConnection("jdbc:db2:*local");
            DatabaseMetaData dmd = c.getMetaData();

            if (jdbcLevel == 1) {
                System.out.println("No support is provided for getUDTs.  Just return.");
```

```
                System.exit(1);
            }

            ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN%", null);
            while (rs.next()) {

                // Richiamare tutte le colonne disponibili dal release
                // JDBC 2.0.
                System.out.println("TYPE_CAT is " + rs.getString("TYPE_CAT"));
                System.out.println("TYPE_SCHEM is " + rs.getString("TYPE_SCHEM"));
                System.out.println("TYPE_NAME is " + rs.getString("TYPE_NAME"));
                System.out.println("CLASS_NAME is " + rs.getString("CLASS_NAME"));
                System.out.println("DATA_TYPE is " + rs.getString("DATA_TYPE"));
                System.out.println("REMARKS is " + rs.getString("REMARKS"));

                // Selezionare tutte le colonne aggiunte in JDBC 3.0.
                if (jdbcLevel > 2) {
                    System.out.println("BASE_TYPE is " + rs.getString("BASE_TYPE"));
                }
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            if (c != null) {
                try {
                    c.close();
                } catch (SQLException e) {
                    // Ignorare l'eccezione di chiusura.
                }
            }
        }
    }
}
```

Eccezioni Java

Il linguaggio Java utilizza delle eccezioni per fornire capacità di gestione errore per i relativi programmi. Un'eccezione è un evento che si verifica quando si esegue il proprio programma che interrompe il normale flusso di istruzioni.

Il sistema del tempo di esecuzione Java e molte classi dai pacchetti Java emettono delle eccezioni in alcune circostanze utilizzando l'istruzione "throw". È possibile utilizzare lo stesso meccanismo per emettere eccezioni nei propri programmi Java.

Classe SQLException Java:

La classe SQLException e i relativi sottotipi forniscono informazioni sugli errori e le avvertenze che si verificano durante l'accesso ad una origine dati.

A differenza della maggior parte di JDBC, definiti dalle interfacce, il supporto dell'eccezione viene fornito nelle classi. La classe base per le eccezioni che si verificano durante l'esecuzione delle applicazioni JDBC è SQLException. Ogni metodo dell'API JDBC viene dichiarato come in grado di emettere delle SQLException. SQLException è un'estensione di java.lang.Exception e fornisce informazioni aggiuntive correlate agli errori che si verificano in un contesto database. In modo specifico, le seguenti informazioni sono disponibili da SQLException:

- Descrizione testo
- SQLState
- Codice di errore
- Un riferimento a qualsiasi altra eccezione che si è verificata

ExceptionExample è un programma che gestisce in maniera adeguata il rilevamento (previsto in questo caso) di SQLException e il dump di tutte le informazioni che esso fornisce.

Nota: JDBC fornisce un meccanismo dove è possibile collegare tra di loro le eccezioni. Ciò consente al programma di controllo o al database di notificare più errori in una singola richiesta. Non esistono attualmente istanze in cui il programma di controllo JDBC nativo può eseguire questa operazione. Queste informazioni vengono fornite soltanto come riferimento e non come chiara indicazione che il programma di controllo non effettuerà mai la suddetta operazione in futuro.

Come è stato notato, gli oggetti SQLException vengono emessi quando si verificano degli errori. Ciò è corretto, ma non è un quadro completo. In pratica, il programma di controllo JDBC nativo emette raramente delle SQLException effettive. Esso emette istanze delle relative sottoclassi SQLException. Ciò consente all'utente di determinare maggiori informazioni su quanto ha effettivamente avuto esito negativo come riportato di seguito.

DB2Exception.java

Neanche gli oggetti DB2Exception vengono emessi direttamente. Questa classe di base viene utilizzata per conservare la funzionalità comune a tutte le eccezioni JDBC. Esistono due sottoclassi di questa classe che devono essere le eccezioni standard emesse da JDBC. Queste sottoclassi sono DB2DBException.java e DB2JDBCException.java. Le DB2DBException sono eccezioni riportate all'utente che provengono direttamente dal database. Le DB2JDBCException vengono emesse quando il programma di controllo JDBC incontra dei problemi per conto proprio. La suddivisione della gerarchia della classe dell'eccezione in questa maniera consente all'utente di gestire i due tipi di eccezioni in modo differente.

DB2DBException.java

Come è stato specificato, le DB2DBException sono eccezioni che provengono direttamente dal database. Esse vengono incontrate quando il programma di controllo JDBC effettua una chiamata alla CLI e riceve un codice di ritorno SQLERROR. La funzione CLI SQLError viene chiamata per acquisire il testo del messaggio, SQLState e il codice fornitore in questi casi. Il testo di sostituzione per SQLMessage viene inoltre richiamato e restituito all'utente. La classe DatabaseException provoca un errore che il database riconosce e notifica al programma di controllo JDBC per cui creare l'oggetto dell'eccezione.

DB2JDBCException.java

DB2JDBCException vengono generate per condizioni di errore che provengono dal programma di controllo JDBC stesso. La funzionalità di questa classe di eccezioni è fondamentalmente differente; il programma di controllo JDBC gestisce la conversione del linguaggio del messaggio dell'eccezione e altre questioni che il sistema operativo e il database gestiscono per eccezioni generate all'interno del database. Quando è possibile, il programma di controllo JDBC aderisce agli SQLState del database. Il codice fornitore per eccezioni emesse dal programma di controllo JDBC è sempre -99999. Anche le DB2DBException riconosciute e restituite dal livello CLI spesso dispongono del codice di errore -99999. La classe JDBCException provoca un errore che il programma di controllo JDBC riconosce e per cui genera l'eccezione. Quando è stata eseguita durante lo sviluppo del release, è stata creata la seguente emissione. Si noti che il vertice dello stack contiene DB2JDBCException. Questa è un'indicazione del fatto che l'errore viene riportato dal programma di controllo JDBC sempre prima di effettuare la richiesta al database.

Esempio: SQLException:

Questo esempio illustra come rilevare un'SQLException ed eseguire il dump di tutte le informazioni che esso fornisce.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class ExceptionExample {
```

```
    public static Connection connection = null;

    public static void main(java.lang.String[] args) {

        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            int count = s.executeUpdate("insert into cujofake.cujofake values(1, 2,3)");

            System.out.println("Did not expect that table to exist.");


        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("-----------------------------------------------------");
            e.printStackTrace();
        } catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
            ex.printStackTrace();
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                System.out.println("Exception caught attempting to shutdown...");
            }
        }
    }
}
```

SQLWarning:

In alcune interfacce i metodi creano un oggetto SQLWarning se provocano un'avvertenza di accesso al database.

I metodi nelle seguenti interfacce possono generare un SQLWarning:

- Connection
- Statement e relativi sottotipi, PreparedStatement e CallableStatement
- ResultSet

Quando un metodo crea un oggetto SQLWarning, il chiamante non viene informato del fatto che si è verificata un'avvertenza per l'accesso ai dati. È necessario che il metodo getWarnings sia chiamato sull'oggetto appropriato per richiamare l'oggetto SQLWarning. È possibile emettere la sottoclasse DataTruncation di SQLWarning, tuttavia, in alcune circostanze. È rilevante il fatto che il programma di controllo JDBC nativo scelga di ignorare alcune avvertenze create dal database per l'aumento dell'efficienza. Ad esempio, un'avvertenza viene creata dal sistema quando si tenta di richiamare i dati oltre la fine di un ResultSet tramite il metodo ResultSet.next. In questo caso, il metodo successivo viene definito in modo tale da restituire ″false″ invece di ″true″, notificando all'utente l'errore. Non è necessario creare un oggetto per riformulare ciò, così l'avvertenza viene semplicemente ignorata.

Se si verificano più avvertenze per l'accesso ai dati, queste sono legate alla prima e possono essere richiamate chiamando il metodo SQLWarning.getNextWarning. Se non esistono più avvertenze nel concatenamento, getNextWarning restituisce `null`.

Gli oggetti SQLWarning successivi continuano ad essere aggiunti al concatenamento finché l'istruzione successiva viene elaborata oppure, nel caso di un oggetto ResultSet, quando il cursore viene riposizionato. Ne consegue che tutti gli oggetti SQLWarning presenti nel concatenamento vengono eliminati.

È possibile che l'utilizzo degli oggetti Connection, Statement e ResultSet determini la creazione di SQLWarning. Le SQLWarning sono messaggi informativi che indicano come, sebbene un'operazione specifica sia stata completata con esito positivo, potrebbero esistere altre informazioni da tenere in considerazione. Le SQLWarning sono un'estensione della classe SQLException, ma non vengono emesse. Queste vengono, al contrario, collegate all'oggetto che determina la loro creazione. Quando una SQLWarning viene creata, niente indica all'applicazione che l'avvertenza è stata generata. È necessario che l'applicazione richieda attivamente le informazioni sull'avvertenza.

Come le SQLException, è possibile che le SQLWarning siano collegate l'una all'altra. È possibile chiamare il metodo clearWarnings su un oggetto Connection, Statement o ResultSet per eliminare le avvertenze relative a quell'oggetto.

Nota: la chiamata al metodo clearWarnings non elimina tutte le avvertenze. Questa elimina solo le avvertenze associate a un particolare oggetto.

Il programma di controllo JDBC elimina gli oggetti SQLWarning in momenti specifici se non vengono eliminati manualmente. Gli oggetti SQLWarning vengono eliminati quando vengono intraprese le azioni seguenti:

- Per l'interfaccia Connection, le avvertenze vengono eliminate nella creazione di un nuovo oggetto Statement, PreparedStatement o CallableStatement.
- Per l'interfaccia Statement, le avvertenze vengono eliminate quando si elabora la nuova istruzione (o quando si elabora nuovamente l'istruzione per PreparedStatement e CallableStatement).
- Per l'interfaccia ResultSet, le avvertenze vengono eliminate quando il cursore viene riposizionato.

DataTruncation e troncamento non presidiato:

DataTruncation è una sottoclasse di SQLWarning. Mentre le SQLWarning non vengono emesse, gli oggetti DataTruncation a volte vengono emessi e collegati come altri oggetti SQLWarning. Il troncamento non presidiato si verifica quando le dimensioni di una colonna eccedono le dimensioni specificate dal metodo dell'istruzione setMaxFieldSize, ma non viene notificata alcuna avvertenza o eccezione.

Gli oggetti DataTruncation forniscono informazioni aggiuntive oltre a quanto viene restituito da un'SQLWarning. Le informazioni disponibili includono quanto segue:

- Il numero di byte dei dati trasferiti.
- L'indice del parametro o della colonna che è stato troncato.
- Se l'indice è relativo ad un parametro o ad una colonna ResultSet.
- Se il troncamento si è verificato durante la lettura dal database o la scrittura in esso.
- La quantità di dati effettivamente trasferiti.

In alcune istanze, è possibile codificare le informazioni ma non sempre le situazioni che si presentano sono comprensibili. Ad esempio, se viene utilizzato il metodo setFloat di PreparedStatement per inserire un valore in una colonna che contiene valori interi, è possibile che risulti un DataTruncation in quanto il valore mobile potrebbe essere superiore al valore massimo che la colonna può contenere. In queste situazioni, il conteggio di byte per il troncamento non ha senso, ma è importante affinché il programma di controllo fornisca le informazioni su di esso.

Notifica tramite i metodi set() e update()

Esiste una sottile differenza tra i programmi di controllo JDBC. Alcuni programmi di controllo come quelli nativi e JDBC di IBM Toolbox per Java rilevano e notificano problemi relativi al troncamento di

dati al momento dell'impostazione del parametro. Queste operazioni vengono effettuate sul metodo set PreparedStatement o sul metodo update ResultSet. Altri programmi di controllo riportano il problema al momento dell'elaborazione dell'istruzione tramite i metodi execute, executeQuery o updateRow.

Notificare il problema nel momento in cui si forniscono dati non corretti invece che nel momento in cui non è possibile continuare oltre l'elaborazione offre i seguenti vantaggi:

- È possibile affrontare l'errore nella propria applicazione quando si ha un problema invece che al momento dell'elaborazione.
- Tramite un controllo durante l'impostazione dei parametri, il programma di controllo JDBC può assicurare che i valori assegnati al database durante l'elaborazione dell'istruzione siano validi. Ciò consente al database di ottimizzare il relativo lavoro ed è possibile che l'elaborazione venga completata più velocemente.

Metodi ResultSet.update() che generano eccezioni DataTruncation

In alcuni release precedenti, i metodi ResultSet.update() inviavano le avvertenze quando esistevano le condizioni del troncamento. Ciò si verifica quando il valore dei dati sta per essere inserito nel database. La specifica impone che i programmi di controllo JDBC emettano eccezioni in questi casi. Come risultato, il programma di controllo JDBC funziona in questo modo.

Non vi è una differenza significativa tra la gestione di una funzione di aggiornamento ResultSet che riceve un errore di troncamento dati e la gestione di un parametro di istruzione preparata impostato per un'istruzione di aggiornamento o inserimento che riceve un errore. In entrambi i casi, il problema è lo stesso; l'utente ha fornito dei dati che non si adattavano all'ambito desiderato.

NUMERIC e DECIMAL vengono troncati a destra della virgola decimale senza che venga emesso alcun messaggio. Questa è la modalità in cui funzionano sia JDBC per UDB NT che SQL interattivo sulla piattaforma System i.

Nota: non viene arrotondato alcun valore quando si verifica un troncamento dei dati. Ogni parte frazionaria di un parametro che non si adatti alla colonna NUMERIC o DECIMAL viene semplicemente persa senza alcuna avvertenza.

Seguono degli esempi, in cui si presume che il valore nella clausola "value" sia effettivamente un parametro impostato su una prepared statement:

```
create table cujosql.test (col1 numeric(4,2))
a) insert into cujosql.test values(22.22)  // riuscito - inserire 22.22
b) insert into cujosql.test values(22.223) // riuscito - inserire 22.22
c) insert into cujosql.test values(22.227) // riuscito - inserire 22.22
d) insert into cujosql.test values(322.22) // non riuscito - Errore di
   conversione di assegnazione alla colonna COL1.
```

Differenza tra un'avvertenza del troncamento dati e un'eccezione del troncamento dati

La specifica stabilisce che il troncamento dati su un valore da scrivere nel database emetta un'eccezione. Se il troncamento dati non viene eseguito sul valore scritto nel database, viene creata un'avvertenza. Ciò significa che nel momento in cui viene identificata una situazione di troncamento dati, è inoltre necessario conoscere il tipo di istruzione che tale troncamento sta elaborando. Stabilito questo come requisito, quanto segue elenca la funzionalità di molti tipi di istruzioni SQL:

- In un'istruzione SELECT, i parametri della query non danneggiano mai il contenuto del database. Quindi, le situazioni di troncamento dati vengono sempre gestite inviando delle avvertenze.
- Nelle istruzioni VALUES INTO e SET, i valori di immissione vengono utilizzati solo per generare valori di emissione. Come risultato, vengono emesse delle avvertenze.

- In un'istruzione CALL, il programma di controllo JDBC non può determinare l'attività di una procedura memorizzata con un parametro. Vengono sempre emesse delle eccezioni quando un parametro di una procedura memorizzata viene troncato.
- Tutti gli altri tipi di istruzione emettono eccezioni piuttosto che inviare avvertenze.

Proprietà del troncamento dati per Connection e DataSource

È stata disponibile una proprietà del troncamento dati per molti release. Il valore predefinito per questa proprietà è "true", ad indicare che le emissioni del troncamento dati vengono controllate e che vengono inviate avvertenze o emesse eccezioni. La proprietà viene fornita per convenienza ed esigenze di prestazioni nei casi in cui l'utente non è informato del fatto che un valore non si adatti alla colonna del database. Si desidera che l'unità inserisca il massimo valore possibile nella colonna.

La proprietà del troncamento dati interessa solo i tipi di dati a base binaria e di carattere

Nei due release precedenti, la proprietà del troncamento dati determinava se era opportuno emettere eccezioni del troncamento dati. Tale proprietà era stata inserita affinché le applicazioni JDBC non tenessero conto di un valore troncato quando il troncamento non era significativo. Esistevano pochi casi in cui si intendeva memorizzare il valore 00 o 10 nel database quando le applicazioni tentavano di inserire 100 in un DECIMAL(2,0). Quindi, la proprietà del troncamento dati del programma di controllo JDBC è stata modificata per rispettare soltanto quelle situazioni in cui si richiedeva il parametro per tipi a base di caratteri come CHAR, VARCHAR, CHAR FOR BIT DATA e VARCHAR FOR BIT DATA.

La proprietà del troncamento dati si applica solo ai parametri

La proprietà del troncamento dati è un'impostazione del programma di controllo JDBC e non del database. Come risultato, essa non ha alcun effetto sulle costanti letterali di un'istruzione. Ad esempio, le seguenti istruzioni elaborate per inserire un valore in una colonna CHAR(8) nel database hanno ancora esito negativo con l'indicatore del troncamento dati impostato su "false" (si presume che quel collegamento è un oggetto java.sql.Connection assegnato ad un altro ambito).

```
Statement stmt = connection.createStatement();
Stmt.executeUpdate("create table cujosql.test (col1 char(8))");
Stmt.executeUpdate("insert into cujosql.test values('dettinger')");
// Non riuscito poiché il valore non si adatta alla colonna database.
```

Il programma di controllo JDBC nativo emette eccezioni per un troncamento dati non significativo

Il programma di controllo JDBC nativo non considera i dati forniti per i parametri. Questa operazione rallenta soltanto l'elaborazione. Tuttavia, possono verificarsi situazioni in cui non è importante per l'utente che un valore venga troncato, ma non è stata impostata la proprietà di collegamento del troncamento dati su "false".

Ad esempio, 'dettinger ', un char(10) che viene passato, emette un'eccezione anche se si adatta ogni elemento importante relativo al valore. In questo modo funziona JDBC per UDB NT; tuttavia, questa non è la funzionalità che si otterrebbe se si passasse il valore come costante letterale in un'istruzione SQL. In questo caso, il motore del database emetterebbe gli spazi aggiuntivi senza restituire alcun messaggio.

I problemi se il programma di controllo JDBC non emette un'eccezione sono i seguenti:
- Il sovraccarico delle prestazioni si estende ad ogni metodo set, sia nel caso in cui fosse necessario che se non lo fosse. Nella maggioranza dei casi dove non ci sarebbe alcun vantaggio, esiste un sovraccarico delle prestazioni considerevole su una funzione comune come setString().
- Il proprio workaround è di minima entità, ad esempio, chiamando la funzione di ridimensionamento sul valore della stringa passato.
- Esistono emissioni con la colonna del database da prendere in considerazione. Uno spazio in CCSID 37 non è affatto uno spazio in CCSID 65535 o 13488.

Troncamento non presidiato

Il metodo dell'istruzione setMaxFieldSize consente di specificare la dimensione del campo massimo per ogni colonna. Se i dati vengono troncati in quanto la relativa dimensione ha superato il valore della dimensione campo massimo, non viene notificata alcuna avvertenza o eccezione. Questo metodo, al pari della proprietà di troncamento dati precedentemente menzionata, influenza solo tipi basati sui caratteri quali CHAR, VARCHAR, CHAR FOR BIT DATA e VARCHAR FOR BIT DATA.

Transazioni JDBC

Una transazione è un'unità logica di lavoro. Per completare un'unità logica di lavoro, potrebbe essere necessario eseguire varie azioni su un database.

Il supporto transazionale consente alle applicazioni di assicurare quanto segue:

- Vengono seguite tutte le fasi necessarie per completare un'unità logica di lavoro.
- Quando una di tali fasi relativa ai file dell'unità di lavoro ha esito negativo, è possibile annullare tutto il lavoro eseguito come parte di quella unità logica di lavoro e il database può ritornare allo stato in cui si trovava prima dell'inizio della transazione.

Le transazioni sono utilizzate per fornire integrità di dati, una semantica corretta dell'applicazione e una visualizzazione coerente di dati durante l'accesso simultaneo. Tutti i programmi di controllo compatibili con JDBC (Java Database Connectivity) devono supportare le transazioni.

Nota: questa sezione riguarda solo le transazioni locali e il concetto JDBC standard delle transazioni. Java e il programma di controllo JDBC nativo supportano JTA (Java Transaction API), le transazioni distribuite e 2PC (two-phase commit protocol/protocollo di commit a due fasi).

Tutto il lavoro delle transazioni viene gestito a livello dell'oggetto Connection. Quando il lavoro relativo a una transazione viene completato, è possibile concluderlo chiamando il metodo commit. Se l'applicazione interrompe in modo imprevisto la transazione, viene chiamato il metodo rollback.

Tutti gli oggetti Statement sotto un collegamento sono una parte della transazione. Ciò significa che se un'applicazione crea tre oggetti Statement e utilizza ogni oggetto per apportare modifiche al database, quando si verifica una chiamata commit o rollback, il lavoro relativo alle tre istruzioni diventa permanente o viene eliminato.

Le istruzioni SQL commit e rollback vengono utilizzate per completare le transazioni quando si lavora solo con SQL. Non è possibile preparare dinamicamente queste istruzioni SQL e non bisogna tentare di utilizzarle nelle applicazioni JDBC al fine di completare le transazioni.

Modalità di commit automatico JDBC:

Per impostazione predefinita, JDBC utilizza una modalità operativa denominata commit automatico. Questo significa che ogni aggiornamento al database è reso immediatamente permanente.

Qualsiasi situazione nella quale un'unità logica di lavoro richiede più di un aggiornamento al database non può essere soddisfatta in modo sicuro in modalità di commit automatico. Se accade qualcosa all'applicazione o al sistema dopo che un aggiornamento è stato effettuato e prima che qualsiasi altro aggiornamento possa essere effettuato, la prima modifica non può essere annullata quando la modalità di commit automatico è in esecuzione.

Poiché le modifiche sono rese permanenti immediatamente in modalità di commit automatico, non esiste alcuna necessità che l'applicazione chiami il metodo commit o il metodo rollback. Ciò rende più semplice scrivere le applicazioni.

È possibile abilitare e disabilitare dinamicamente la modalità di commit automatico durante un collegamento. Il commit automatico viene abilitato nel modo seguente, presupponendo che l'origine dati esista già:

```
Connection connection = dataSource.getConnection();

Connection.setAutoCommit(false);        // Disabilita il commit automatico.
```

Se l'impostazione di commit automatico viene modificata a metà di una transazione, qualsiasi lavoro in sospeso viene automaticamente sottoposto a commit. Si genera una SQLException se il commit automatico viene abilitato per un collegamento che appartiene ad una transazione distribuita.

Livelli di isolamento della transazione:

I livelli di isolamento della transazione specificano quali dati sono visibili per le istruzioni all'interno di una transazione. Questi livelli influiscono direttamente sul livello di accesso simultaneo definendo quale interazione è possibile tra le transazioni rispetto alla stessa origine dati di destinazione.

Anomalie del database

Le anomalie del database sono risultati generati che sembrano errati se si considera l'ambito di una singola transazione, ma che sono corretti se si considera l'ambito di tutte le transazioni. I tipi differenti di anomalie del database sono descritti come segue:

- Letture **errate** si verificano quando:
  1. La Transazione A inserisce una riga in una tabella.
  2. La Transazione B legge la nuova riga.
  3. Viene eseguito il rollback della Transazione.

  È possibile che la Transazione B abbia effettuato del lavoro sul sistema in base a una riga inserita dalla transazione A, ma quella riga non è mai divenuta parte permanente del database.
- Le letture **non ripetibili** si verificano quando:
  1. La Transazione A legge una riga.
  2. La Transazione B modifica la riga.
  3. La Transazione A legge la stessa riga una seconda volta e ottiene i nuovi risultati.
- Le letture **fantasma** si verificano quando:
  1. La Transazione A legge tutte le righe che soddisfano una clausola WHERE su una query SQL.
  2. La Transazione B inserisce una riga aggiuntiva che soddisfa la clausola WHERE.
  3. La Transazione A rivaluta la condizione WHERE e raccoglie la riga aggiuntiva.

Nota: DB2 for i5/OS non sempre espone l'applicazione alle anomalie del database consentite ai livelli prescritti a causa delle sue strategie di vincolo.

Livelli di isolamento della transazione JDBC

Esistono cinque livelli di isolamento della transazione nell'API JDBC di IBM Developer Kit per Java. Segue un elenco di tali livelli dal meno restrittivo al più restrittivo:

JDBC_TRANSACTION_NONE
Si tratta di una costante speciale che indica che il programma di controllo JDBC non supporta le transazioni.

JDBC_TRANSACTION_READ_UNCOMMITTED
Questo livello consente alle transazioni di esaminare modifiche non convalidate ai dati. Tutte le anomalie del database sono possibili a questo livello.

JDBC_TRANSACTION_READ_COMMITTED
Questo livello indica che qualsiasi modifica apportata all'interno di una transazione non è visibile all'esterno di essa finché la transazione non viene sottoposta a commit. Questo evita la possibilità che si verifichino letture errate.

JDBC_TRANSACTION_REPEATABLE_READ
Questo livello indica che le righe lette conservano vincoli cosicché non è possibile che un'altra transazione le modifichi quando la transazione non è ancora completata. Questo impedisce letture errate e non ripetibili. Le letture fantasma sono ancora possibili.

JDBC_TRANSACTION_SERIALIZABLE
Le tabelle sono vincolate in relazione alla transazione cosicché non è possibile che le condizioni WHERE siano modificate da altre transazioni che aggiungono valori a o eliminano valori da una tabella. Questo evita tutti i tipi di anomalie del database.

È possibile utilizzare il metodo setTransactionIsolation per modificare il livello di isolamento della transazione relativo a un collegamento.

Considerazioni

Un errore comune di interpretazione è quello di ritenere che la specifica JDBC definisca i cinque livelli precedentemente menzionati. Si pensa comunemente che il valore TRANSACTION_NONE rappresenti il concetto di esecuzione senza controllo di commit. La specifica JDBC non definisce TRANSACTION_NONE nello stesso modo. TRANSACTION_NONE viene definito nella specifica JDBC come un livello nel quale il programma di controllo non supporta le transazioni e non è un programma di controllo compatibile con JDBC. Il livello NONE non è mai notificato quando si chiama il metodo getTransactionIsolation.

Il problema è in parte complicato dal fatto che un livello di isolamento della transazione predefinito di un programma di controllo JDBC viene stabilito dall'implementazione. Il livello predefinito di isolamento della transazione per il livello di isolamento della transazione predefinito del programma di controllo JDBC nativo è NONE. Ciò consente al programma di controllo di gestire i file che non possiedono giornali e non è necessario eseguire alcuna specifica come ad esempio i file nella libreria QGPL.

Il programma di controllo JDBC nativo consente di inoltrare JDBC_TRANSACTION_NONE al metodo setTransactionIsolation o specificare ″none″ come proprietà di un collegamento. Il metodo getTransactionIsolation, tuttavia, notifica sempre a JDBC_TRANSACTION_READ_UNCOMMITTED quando il valore è ″none″. È responsabilità dell'applicazione tenere traccia del livello in cui l'esecuzione si verifica se viene indicato come requisito nell'applicazione.

Nei release precedenti, il programma di controllo JDBC gestiva la selezione di ″true″ da parte dell'utente per il commit automatico modificando il livello di isolamento della transazione su none poiché il sistema non possedeva alcun concetto relativo a una modalità di commit automatico impostato su ″true″. Ciò restituiva un'approssimazione fedele della funzionalità, ma non forniva risultati corretti relativi a tutti gli scenari. Ciò non avviene più; il database separa il concetto di commit automatico dal concetto di livello di isolamento della transazione. Perciò è assolutamente giustificata l'esecuzione al livello JDBC_TRANSACTION_SERIALIZABLE con il commit automatico abilitato. L'unico scenario non valido riguarda l'esecuzione al livello JDBC_TRANSACTION_NONE senza la modalità di commit automatico. Non è possibile che l'applicazione ottenga il controllo sui limiti di commit quando il sistema non è in esecuzione con un livello di isolamento della transazione.

Livelli di isolamento della transazione tra la specifica JDBC e la piattaforma System i

La piattaforma System i possiede nomi comuni per i livelli di isolamento della transazione che non corrispondono ai nomi forniti dalla specifica JDBC. La seguente tabella mette in corrispondenza i nomi utilizzati dalla piattaforma System i ma non sono equivalenti a quelli utilizzati dalla specifica JDBC:

Livello JDBC*	Livello System i
JDBC_TRANSACTION_NONE	*NONE o *NC
JDBC_TRANSACTION_READ_UNCOMMITTED	*CHG o *UR
JDBC_TRANSACTION_READ_COMMITTED	*CS
JDBC_TRANSACTION_REPEATABLE_READ	*ALL o *RS
JDBC_TRANSACTION_SERIALIZABLE	*RR

* In questa tabella, il valore JDBC_TRANSACTION_NONE è allineato con i livelli System i *NONE e *NC per chiarezza. Non si tratta di una corrispondenza diretta da specifica a livello System i.

Punti di salvataggio:

I punti di salvataggio consentono l'impostazione di ″punti di passaggio″ in una transazione. I punti di salvataggio sono dei punti di controllo ai quali l'applicazione può ritornare senza sprecare l'intera transazione.

I punti di salvataggio sono nuovi in JDBC 3.0, nel senso che l'applicazione deve essere in esecuzione su JDK (Java Development Kit) 1.4 o release successivo per poterli utilizzare. Inoltre i punti di salvataggio sono nuovi per il Developer Kit per Java, nel senso che essi non sono supportati se JDK 1.4 o un release successivo non è utilizzato con release precedenti di Developer Kit per Java.

Nota: il sistema fornisce istruzioni SQL per la gestione dei punti di salvataggio. Si avvisa che le applicazioni JDBC non utilizzano queste istruzioni direttamente in un'applicazione. Tale operazione può funzionare, ma il programma di controllo JDBC perde la capacità di tenere traccia dei punti di salvataggio quando ciò viene effettuato. Quanto meno, bisogna evitare l'unione dei due modelli (cioè, l'utilizzo delle proprie istruzioni SQL relative ai punti di salvataggio e l'utilizzo dell'API JDBC).

Impostazione e ritorno ai punti di salvataggio

È possibile impostare i punti di salvataggio lungo l'intera transazione. È possibile che l'applicazione torni a uno qualsiasi di questi punti di salvataggio se qualcosa non funziona e che continui l'elaborazione da quel preciso punto. Nell'esempio seguente l'applicazione inserisce il valore FIRST nella tabella di database. Dopo aver fatto ciò, viene impostato un punto di salvataggio e un altro valore, SECOND, viene inserito nel database. Un'operazione di ritorno al punto di salvataggio viene emessa e annulla l'inserimento del valore SECOND, ma lascia il valore FIRST come parte della transazione in sospeso. Infine, viene inserito il valore THIRD e la transazione viene sottoposta a commit. La tabella di database contiene i valori FIRST e THIRD.

Esempio: impostazione e ritorno ai punti di salvataggio

```
Statement s = Connection.createStatement();
s.executeUpdate("insert into table1 values ('FIRST')");
Savepoint pt1 = connection.setSavepoint("FIRST SAVEPOINT");
s.executeUpdate("insert into table1 values ('SECOND')";);
connection.rollback(pt1);        // Annulla l'inserimento più recente.
s.executeUpdate("insert into table1 values ('THIRD')");
connection.commit();
```

Sebbene sia improbabile che l'impostazione di punti di salvataggio in modalità di commit automatico possa causare problemi, questi non possono essere sottoposti a rollback in quanto il loro periodo di attività termina con la fine di una transazione.

Rilascio di un punto di salvataggio

È possibile rilasciare i punti di salvataggio dall'applicazione con il metodo releaseSavepoint sull'oggetto Connection. Una volta rilasciato un punto di salvataggio, si verifica un'eccezione quando si tenta di ritornare su di esso. Quando si effettua il commit o il rollback di una transazione, tutti i punti di salvataggio vengono automaticamente rilasciati. Quando si ritorna a un punto di salvataggio, tutti i successivi punti di salvataggio vengono rilasciati.

Transazioni distribuite JDBC

In genere le transazioni in JDBC (Java Database Connectivity) sono locali. Ciò significa che un singolo collegamento esegue tutto il lavoro della transazione e che il collegamento può lavorare solo su una transazione alla volta.

Quando tutto il lavoro per quella transazione è stato completato o ha avuto esito negativo, viene chiamato il commit o il rollback per rendere il lavoro permanente ed è possibile iniziare una nuova transazione. Esiste, comunque, anche un supporto avanzato per transazioni disponibile in Java che fornisce una funzionalità oltre le transazioni locali. Tale supporto viene interamente specificato dalla specifica JTA (Java Transaction API).

JTA (Java Transaction API) dispone di un supporto per le transazioni complesse. Essa fornisce inoltre il supporto per separare le transazioni dagli oggetti Connection. Così come JDBC è modellato su specifiche ODBC (Object Database Connectivity) e CLI (Call Level Interface) X/Open, JTA è modellata sulla specifica XA (Extended Architecture) X/Open. JTA e JDBC lavorano insieme per separare le transazioni dagli oggetti Connection. Tale operazione di separazione consente all'utente di avere un singolo collegamento che lavora su più transazioni contemporaneamente. Viceversa, ciò consente all'utente di avere più collegamenti che lavorano su una singola transazione.

Nota: se si intende gestire JTA, consultare l'argomento Introduzione a JDBC per ulteriori informazioni sui file JAR (Java Archive) richiesti nel proprio classpath delle estensioni. Si desidera sia il pacchetto facoltativo JDBC 2.0 che i file JAR JTA (essi vengono trovati automaticamente da JDK se si sta eseguendo JDK 1.4 o una versione successiva). Non è possibile trovarli per impostazione predefinita.

Transazioni con JTA

Quando si utilizzano contemporaneamente JTA e JDBC, esistono una serie di fasi tra di essi per completare il lavoro di transazione. Il supporto per XA viene fornito tramite la classe XADataSource. Tale classe contiene il supporto per impostare un lotto di collegamenti esattamente nello stesso modo della classe principale ConnectionPoolDataSource.

Con un'istanza XADataSource, è possibile richiamare un oggetto XAConnection. Tale oggetto funziona come contenitore sia per l'oggetto Connection JDBC che per un oggetto XAResource. L'oggetto XAResource è progettato per gestire il supporto di transazione XA. XAResource gestisce le transazioni tramite oggetti denominati ID della transazione (XID).

L'XID è un'interfaccia che è necessario implementare. Essa rappresenta una definizione Java della struttura XID dell'identificativo della transazione X/Open. Questo oggetto contiene tre parti:

- Un ID formato della transazione globale
- Un ID della transazione globale
- Un qualificatore del salto

Esaminare la specifica JTA per dettagli completi sull'interfaccia.

Utilizzo del supporto UDBXADataSource per le transazioni distribuite e i lotti

Il supporto JTA (Java Transaction API) fornisce supporto diretto per i lotti di collegamenti. UDBXADataSource è un'estensione di un ConnectionPoolDataSource, che consente ad un'applicazione di accedere ad oggetti XAConnection inseriti in un lotto. Dal momento che UDBXADataSource è un ConnectionPoolDataSource, la configurazione e l'utilizzo di UDBXADataSource sono gli stessi descritti nell'argomento Utilizzare il supporto DataSource per il lotto di oggetti.

Proprietà XADataSource

In aggiunta alle proprietà fornite da ConnectionPoolDataSource, l'interfaccia XADataSource fornisce le seguenti proprietà:

Metodo Set (tipo di dati)	Valori	Descrizione
setLockTimeout (int)	0 o qualsiasi valore positivo	Qualsiasi valore positivo è un supero tempo di vincolo (in secondi) valido al livello della transazione. Un supero tempo di vincolo di 0 indica che non esiste alcun valore di tale supero tempo impostato al livello della transazione, sebbene sia possibile che ne esista uno ad altri livelli (il lavoro o la tabella). Il valore predefinito è 0.
setTransactionTimeout (int)	0 o qualsiasi valore positivo	Qualsiasi valore positivo è un supero tempo di transazione valido (in secondi). Un supero tempo di transazione di 0 indica che non esiste alcun valore del supero tempo di transazione imposto. Se la transazione è attiva per un periodo di tempo superiore al valore di supero tempo, viene contrassegnata come solo rollback e i successivi tentativi di eseguire il lavoro in essa provocano il verificarsi di un'eccezione. Il valore predefinito è 0.

ResultSet e transazioni

Oltre a delimitare l'inizio e la fine di una transazione come mostrato nel precedente esempio, è possibile sospendere le transazioni momentaneamente e riprenderle in seguito. Ciò fornisce una serie di scenari per le risorse ResultSet che vengono create durante una transazione.

Fine di una transazione semplice

Quando si termina una transazione, tutti i ResultSet creati sotto tale transazione vengono automaticamente chiusi. Si consiglia di chiudere esplicitamente i propri ResultSet quando si è terminato di utilizzarli per assicurare l'elaborazione parallela massima. Tuttavia, viene emessa un'eccezione se si accede a qualsiasi ResultSet aperto durante una transazione dopo che viene effettuata una chiamata XAResource.end.

Sospensione e ripristino

Mentre una transazione è sospesa, l'accesso ad un ResultSet creato mentre la transazione era attiva non è consentito e dà come risultato un'eccezione. Tuttavia, una volta ripristinata la transazione, il ResultSet è nuovamente disponibile e rimane nello stesso stato in cui si trovava prima che la transazione venisse sospesa.

Come effettuare dei ResultSet sospesi

Mentre la transazione è sospesa, non è possibile accedere al ResultSet. Tuttavia, è possibile elaborare nuovamente gli oggetti Statement sotto un'altra transazione per eseguire il lavoro. Dal momento che gli oggetti Statement JDBC possono avere solo un ResultSet alla volta (escluso il supporto JDBC 3.0 per più ResultSet simultanei da una chiamata alla procedura memorizzata), è necessario chiudere il ResultSet per la transazione sospesa per soddisfare la richiesta della nuova transazione. Questo è esattamente quanto si verifica.

Nota: sebbene JDBC 3.0 consenta ad uno Statement di disporre di più ResultSet aperti contemporaneamente per una chiamata ad una procedura memorizzata, essi vengono trattati come unità singola e chiudono tutti se l'oggetto Statement viene rielaborato sotto una nuova transazione. Non è possibile avere dei ResultSet da due transazioni attive contemporaneamente per un'istruzione singola.

Multiplexing

L'API JTA è progettata per separare le transazioni dai collegamenti JDBC. Questa API consente all'utente di avere più collegamenti che operano su una transazione singola o un singolo collegamento che lavora su più transazioni contemporaneamente. Ciò viene denominato**multiplexing** e permette di eseguire molte attività complesse che non è possibile realizzare soltanto con JDBC.

Per ulteriori informazioni sull'utilizzo di JTA, esaminare la specifica JTA. La specifica JDBC 3.0 contiene inoltre informazioni su come lavorano insieme queste due tecnologie per supportare le transazioni distribuite.

Commit a due fasi e registrazione della transazione

Le API JTA rendono pienamente esplicite le responsabilità del protocollo di commit a due fasi distribuito all'applicazione. Come hanno mostrato gli esempi, quando si utilizzano JTA e JDBC per accedere ad un database sotto una transazione JTA, l'applicazione utilizza i metodi XAResource.prepare() e XAResource.commit() o soltanto il metodo XAResource.commit() per convalidare le modifiche.

In aggiunta, quando si accede a più database distinti utilizzando una transazione singola, è responsabilità dell'applicazione assicurare che vengano eseguiti il protocollo di commit a due fasi e ogni registrazione associata richiesta per l'atomicità della transazione rispetto a quei database. Normalmente il commit a due fasi in elaborazione su più database (cioè, XAResource) e la relativa registrazione vengono eseguite sotto il controllo di un server dell'applicazione o di un monitor della transazione in modo che l'applicazione stessa non viene effettivamente coinvolta in tali problemi.

Ad esempio, è possibile che l'applicazione chiami un metodo commit() o effettui una restituzione dalla relativa elaborazione senza alcun errore. Il monitor della transazione o il server dell'applicazione sottostanti inizia quindi l'elaborazione per ogni database (XAResource) che ha partecipato alla singola transazione distribuita.

Il server dell'applicazione utilizza una registrazione estensiva durante l'elaborazione del commit a due fasi. Esso chiama prima il metodo XAResource.prepare() per ogni database che partecipa (XAResource), quindi il metodo XAResource.commit() per ogni database che partecipa (XAResource).

Se si verifica un errore durante questa elaborazione, le registrazioni del monitor della transazione del server dell'applicazione consentono a tale server di utilizzare successivamente le API JTA per correggere la transazione distribuita. Questa correzione, sotto il controllo del server dell'applicazione o il monitor della transazione, consente al server dell'applicazione di mettere la transazione in uno stato noto ad ogni database che partecipa (XAResource). Ciò assicura uno stato conosciuto dell'intera transazione distribuita su tutti i database che partecipano.

Concetti correlati

"Introduzione a JDBC" a pagina 34
Il programma di controllo JDBC (Java Database ConnectivityI fornito con IBM Developer Kit per Java è denominato programma di controllo JDBC di IBM Developer Kit per Java. Questa unità è inoltre comunemente conosciuta come programma di controllo JDBC nativo.

"Utilizzo del supporto DataSource per la creazione di lotti di oggetti" a pagina 132
È possibile utilizzare i DataSource per ottenere che più applicazioni condividano una configurazione comune per accedere ad un database. Ciò si realizza perché ogni applicazione fa riferimento allo stesso nome DataSource.

Riferimenti correlati

"Proprietà di ConnectionPoolDataSource" a pagina 134
È possibile configurare l'interfaccia ConnectionPoolDataSource utilizzando la serie di proprietà che essa fornisce.

Informazioni correlate

Specifica JTA (Java Transaction API) 1.0.1

Esempio: utilizzo di JTA per gestire una transazione:

Questo è un esempio del modo in cui utilizzare JTA (Java Transaction API) per gestire una transazione in una applicazione.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTACommit {

    public static void main(java.lang.String[] args) {
        JTACommit test = new JTACommit();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();
```

```
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
        } catch (SQLException e) {
            // Ignorare... non esiste
        }

        s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
        s.close();
    } finally {
        if (c != null) {
            c.close();
        }
    }
}


/**
 * Questa verifica usa il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Per transazioni XA, è necessario un identificativo transazione.
        // Un'implementazione dell'interfaccia XID non è inclusa all'unità di
        // di controllo JDBC. Vedere Transazioni con JTA per una descrizione di
        // questa interfaccia per creare una classe relativa ad essa.
        Xid xid = new XidImpl();

        // Il collegamento da XAResource può essere utilizzato come qualsiasi
        // collegamento JDBC.
        Statement     stmt   = c.createStatement();

        // La risorsa XA deve essere notificata prima di avviare qualsiasi
        // elaborazione di transazioni.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Viene eseguito un lavoro JDBC standard.
        int count =
          stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is pretty fun.')");

        // Quando viene completato un lavoro transazione, la risorsa XA deve
        // essere notificata di nuovo.
        xaRes.end(xid, XAResource.TMSUCCESS);

        // La transazione rappresentata dell'ID transazione viene preparata
        // per essere sottoposti a commit.
        int rc = xaRes.prepare(xid);

        // La transazione viene sottoposta a commit tramite XAResource.
        // L'oggetto JDBCConnection non viene usato per eseguire il commit
        // della transazione quando viene utilizzato JTA.
        xaRes.commit(xid, false);


    } catch (Exception e) {
```

```
            System.out.println("Something has gone wrong.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

Esempio: più collegamenti che operano su una transazione:

Questo è un esempio del modo in cui utilizzare più collegamenti che operano su una singola transazione.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;
public class JTAMultiConn {
   public static void main(java.lang.String[] args) {
      JTAMultiConn test = new JTAMultiConn();
      test.setup();
      test.run();
   }
/**
* Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
*/
   public void setup() {
      Connection c = null;
      Statement s = null;
      try {
         Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
         c = DriverManager.getConnection("jdbc:db2:*local");
         s = c.createStatement();
         try {
            s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
         }
         catch (SQLException e) {
         // Ignorare... non esiste
         }
         s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
                         (50))");
               s.close();
      }
      finally {
         if (c != null) {
            c.close();
         }
      }
   }
/**
* Questa verifica usa il supporto JTA per gestire le transazioni.
*/
   public void run() {
      Connection c1 = null;
      Connection c2 = null;
      Connection c3 = null;
```

```
try {
   Context ctx = new InitialContext();
   // Presupporre che il sorgente dati venga riportato da un UDBXADataSource.
   UDBXADataSource ds = (UDBXADataSource)
                        ctx.lookup("XADataSource");
   // Dal DataSource, ottenere un oggetto XAConnection che
   // contenga un XAResource e un oggetto Connection.
   XAConnection xaConn1 = ds.getXAConnection();
   XAConnection xaConn2 = ds.getXAConnection();
   XAConnection xaConn3 = ds.getXAConnection();
   XAResource xaRes1 = xaConn1.getXAResource();
   XAResource xaRes2 = xaConn2.getXAResource();
   XAResource xaRes3 = xaConn3.getXAResource();
   c1 = xaConn1.getConnection();
   c2 = xaConn2.getConnection();
   c3 = xaConn3.getConnection();
   Statement stmt1 = c1.createStatement();
   Statement stmt2 = c2.createStatement();
   Statement stmt3 = c3.createStatement();
   // Per transazioni XA, è necessario un identificativo transazione.
   // Il supporto per la creazione di XID viene lasciato di nuovo al
   // programma applicativo.
   Xid xid = JDXATest.xidFactory();
   // Eseguire l'elaborazione di alcune transazioni tramite cui sono stati
   // creati i tre collegamenti.
   xaRes1.start(xid, XAResource.TMNOFLAGS);
   int count1 = stmt1.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-A')");
   xaRes1.end(xid, XAResource.TMNOFLAGS);

   xaRes2.start(xid, XAResource.TMJOIN);
   int count2 = stmt2.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-B')");
   xaRes2.end(xid, XAResource.TMNOFLAGS);

   xaRes3.start(xid, XAResource.TMJOIN);
   int count3 = stmt3.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-C')");
   xaRes3.end(xid, XAResource.TMSUCCESS);
   // Una volta terminato, eseguire il commit della transazione come una singola unità
   // È richiesto un prepare() e commit() o un commit() a 1 fase per ogni
   // database separato (XAResource) che ha preso parte alla
   // transazione. Poiché tutte le risorse cui si ha avuto accesso (xaRes1, xaRes2
   // e xaRes3) si riferiscono allo stesso database, è necessario solo un prepare o commit.
   int rc = xaRes.prepare(xid);
   xaRes.commit(xid, false);
}
catch (Exception e) {
   System.out.println("Something has gone wrong.");
   e.printStackTrace();
}
finally {
   try {
      if (c1 != null) {
         c1.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Note: Cleaup exception " +
                         e.getMessage());
   }
   try {
      if (c2 != null) {
         c2.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Note: Cleaup exception " +
                         e.getMessage());
   }
```

```
         try {
            if (c3 != null) {
               c3.close();
            }
         }
         catch (SQLException e) {
            System.out.println("Note: Cleaup exception " +
                               e.getMessage());
         }
      }
   }
}
```

Esempio: utilizzo di un collegamento con più transazioni:

Questo è un esempio del modo in cui utilizzare un collegamento singolo con più transazioni.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTAMultiTx {

    public static void main(java.lang.String[] args) {
        JTAMultiTx test = new JTAMultiTx();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }
```

```
/**
 * Questa verifica usa il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();
        Statement     stmt   = c.createStatement();

        // Per transazioni XA, è necessario un identificativo transazione.
        // Ciò non intende implicare che tutti gli XID siano uguali.
        // Ogni XID deve essere univoco per distinguere le diverse transazioni
        // che si verificano.
        // Il supporto per la creazione di XID viene lasciato di nuovo al
        // programma applicativo.
        Xid xid1 = JDXATest.xidFactory();
        Xid xid2 = JDXATest.xidFactory();
        Xid xid3 = JDXATest.xidFactory();

        // Effettuare l'elaborazione tramite tre transazioni per questo collegamento.
        xaRes.start(xid1, XAResource.TMNOFLAGS);
        int count1 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-A')");
        xaRes.end(xid1, XAResource.TMNOFLAGS);

        xaRes.start(xid2, XAResource.TMNOFLAGS);
        int count2 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-B')");
        xaRes.end(xid2, XAResource.TMNOFLAGS);

        xaRes.start(xid3, XAResource.TMNOFLAGS);
        int count3 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-C')");
        xaRes.end(xid3, XAResource.TMNOFLAGS);

        // Preparare tutte le transazioni.
        int rc1 = xaRes.prepare(xid1);
        int rc2 = xaRes.prepare(xid2);
        int rc3 = xaRes.prepare(xid3);

        // Due transazioni sono sottoposte a commit e una a rollback.
        // Il tentativo di inserire il secondo valore nella tabella non
        // è stato sottoposto a commit
        xaRes.commit(xid1, false);
        xaRes.rollback(xid2);
        xaRes.commit(xid3, false);


    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    } finally {
        try {
            if (c != null)
                c.close();
        } catch (SQLException e) {
            System.out.println("Note:  Cleaup exception.");
            e.printStackTrace();
```

```
            }
        }
    }
}
```

Esempio: ResultSet sospesi:

Questo è un esempio del modo in cui un oggetto Statement viene rielaborato sotto un'altra transazione per eseguire il lavoro.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEffect {

    public static void main(java.lang.String[] args) {
        JTATxEffect test = new JTATxEffect();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.)");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }


    /**
     * Questa verifica usa il supporto JTA per gestire le transazioni.
     */
    public void run() {
        Connection c = null;
```

```
try {
    Context ctx = new InitialContext();

    // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
    UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

    // Dal DataSource, ottenere un oggetto XAConnection che
    // contenga un XAResource e un oggetto Connection.
    XAConnection  xaConn = ds.getXAConnection();
    XAResource    xaRes  = xaConn.getXAResource();
    Connection    c      = xaConn.getConnection();

    // Per transazioni XA, è necessario un identificativo transazione.
    // Un'implementazione dell'interfaccia XID non è inclusa all'unità di
    // controllo JDBC. Consultare Transazioni con JTA
    // per una descrizione di questa interfaccia per creare
    // la classe relativa ad essa.
    Xid xid = new XidImpl();

    // Il collegamento da XAResource può essere utilizzato come qualsiasi
    // collegamento JDBC.
    Statement     stmt   = c.createStatement();

    // La risorsa XA deve essere notificata prima di avviare qualsiasi
    // elaborazione di transazioni.
    xaRes.start(xid, XAResource.TMNOFLAGS);

    // Creare un ResultSet durante l'elaborazione JDBC e selezionare una riga.
    ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
    rs.next();

    // Il metodo end viene chiamato con l'opzione suspend. I
    // ResultSets associato alla transazione corrente sono 'on hold'.
    // In questo stato non sono né accessibili e né inviabili.
    xaRes.end(xid, XAResource.TMSUSPEND);


    // Nel frattempo, può essere eseguita un'altra elaborazione all'esterno della transazione
    // I ResultSets sotto la transazione possono essere chiusi se l'oggetto
    // Statement utilizzato per crearli viene riutilizzato.
    ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
    while (nonXARS.next()) {
        // Elaborazione...
    }


    // Tentare di ritornare alla transazione sospesa. Il ResultSet della
    // transazione sospesa è scomparso in quanto l'istruzione è stata
    // elaborata nuovamente.
    xaRes.start(newXid, XAResource.TMRESUME);
    try {
        rs.next();
    } catch (SQLException ex) {
        System.out.println("This exception is expected. " +
                           "The ResultSet closed due to another process.");
    }


    // Quando la transazione è stata completata, chiuderla
    // e eseguire il commit di qualsiasi elaborazione sia presente in essa.
    xaRes.end(xid, XAResource.TMNOFLAGS);
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);


} catch (Exception e) {
```

```
            System.out.println("Something has gone wrong.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

Esempio: fine di una transazione:

Questo è un esempio di chiusura di una transazione nell'applicazione.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEnd {

    public static void main(java.lang.String[] args) {
        JTATxEnd test = new JTATxEnd();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.)");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
```

```
}


/**
 * Questa verifica utilizza il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Per transazioni XA, è necessario l'identificativo transazione.
        // Un'implementazione dell'interfaccia XID non è inclusa all'unità
        // di controllo JDBC. Vedere Transazioni con JTA per una
        // descrizione di questa interfaccia per creare la classe relativa ad essa.
        Xid xid = new XidImpl();

        // Il collegamento da XAResource può essere utilizzato come qualsiasi
        // collegamento JDBC.
        Statement     stmt   = c.createStatement();

        // La risorsa XA deve essere notificata prima di avviare qualsiasi
        // elaborazione di transazioni.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Creare un ResultSet durante l'elaborazione JDBC e selezionare una riga.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Quando viene chiamato il metodo end, tutti i cursori ResultSet vengono chiusi.
        // L'accesso al ResultSet dopo questo punto fa in modo che
        // venga emessa un'eccezione.
        xaRes.end(xid, XAResource.TMNOFLAGS);

        try {
            String value = rs.getString(1);
            System.out.println("Something failed if you receive this message.");
        } catch (SQLException e) {
            System.out.println("The expected exception was thrown.");
        }

        // Eseguire il commit della transazione per assicurarsi che tutti i vincoli vengano
        // rilasciati.
        int rc = xaRes.prepare(xid);
        xaRes.commit(xid, false);


    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    } finally {
        try {
            if (c != null)
                c.close();
        } catch (SQLException e) {
            System.out.println("Note:  Cleaup exception.");
            e.printStackTrace();
```

```
            }
        }
    }
}
```

"Transazioni distribuite JDBC" a pagina 82
In genere le transazioni in JDBC (Java Database Connectivity) sono locali. Ciò significa che un singolo collegamento esegue tutto il lavoro della transazione e che il collegamento può lavorare solo su una transazione alla volta.

Esempio: sospensione e ripristino di una transazione:

Questo è un esempio di transazione sospesa e successivamente ripristinata.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxSuspend {

    public static void main(java.lang.String[] args) {
        JTATxSuspend test = new JTATxSuspend();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.)");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }


    /**
```

```
 * Questa verifica usa il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Per transazioni XA, è necessario un identificativo transazione.
        // Un'implementazione dell'interfaccia XID non è inclusa all'unità di
        // di controllo JDBC. Vedere Transazioni con JTA per una
        // descrizione di questa interfaccia per creare la classe relativa ad essa.
        Xid xid = new XidImpl();

        // Il collegamento da XAResource può essere utilizzato come qualsiasi
        // collegamento JDBC.
        Statement     stmt   = c.createStatement();

        // La risorsa XA deve essere notificata prima di avviare qualsiasi
        // elaborazione di transazioni.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Creare un ResultSet durante l'elaborazione JDBC e selezionare una riga.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Il metodo end viene chiamato con l'opzione suspend. I
        // ResultSets associato alla transazione corrente sono 'on hold'.
        // In questo stato non sono né accessibili e né inviabili.
        xaRes.end(xid, XAResource.TMSUSPEND);


        // Con la transazione può essere eseguita un'altra elaborazione. Ad esempio
        // è possibile creare un'istruzione ed elaborare una query. Questa
        // elaborazione e altre elaborazioni di transazioni eseguibili dalla
        // transazione sono separate sa quella effettuata precedentemente con XID.
        Statement nonXAStmt = conn.createStatement();
        ResultSet nonXARS = nonXAStmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
        while (nonXARS.next()) {
            // Elaborazione...
        }
        nonXARS.close();
        nonXAStmt.close();


        // Se si tenta di utilizzare risorse di transazioni sospese,
        // si otterrà un'eccezione.
        try {
            rs.getString(1);
            System.out.println("Value of the first row is " + rs.getString(1));
        } catch (SQLException e) {
            System.out.println("This was an expected exception - " +
                               "suspended ResultSet was used.");
        }


        // Riprendere la transazione sospesa e completare l'elaborazione.
        // Il ResultSet è rimasto immutato dall'inizio della sospensione.
```

```
            xaRes.start(newXid, XAResource.TMRESUME);
            rs.next();
            System.out.println("Value of the second row is " + rs.getString(1));


            // Quando la transazione è stata completata, chiuderla
            // e eseguire il commit di qualsiasi elaborazione sia presente in essa.
            xaRes.end(xid, XAResource.TMNOFLAGS);
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Something has gone wrong.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

Tipi Statement

L'interfaccia Statement e le relative sottoclassi PreparedStatement e CallableStatement vengono utilizzate per elaborare i comandi SQL (structured query language) sul database. Le istruzioni SQL determinano la creazione di oggetti ResultSet.

Le sottoclassi dell'interfaccia Statement vengono create con un certo numero di metodi sull'interfaccia Connection. Un oggetto Connection singolo può avere molti oggetti Statement creati contemporaneamente sotto di esso. Nei release precedenti, era possibile fornire il numero esatto di oggetti Statement che potevano essere creati. È impossibile che si verifichi ciò in questo release perché tipi differenti di oggetti Statement ottengono numeri diversi di ″handle″ all'interno del motore del database. Perciò i tipi di oggetti Statement utilizzati influenzano il numero di istruzioni che possono essere attive sotto un collegamento contemporaneamente.

Un'applicazione chiama il metodo Statement.close per indicare che l'applicazione ha terminato l'elaborazione di un'istruzione. Tutti gli oggetti Statement vengono chiusi quando il collegamento che li ha creati viene chiuso. Non bisogna, tuttavia, fare un affidamento completo su questa funzionalità per chiudere gli oggetti Statement. Ad esempio, se l'applicazione si modifica in modo tale che viene utilizzato un lotto di collegamenti invece di chiudere in modo esplicito i collegamenti, l'applicazione ″perde″ gli handle dell'istruzione poiché i collegamenti non si chiudono mai. La chiusura degli oggetti Statement quando non sono più necessari permette che le risorse database esterne utilizzate dall'istruzione vengano rilasciate immediatamente.

Il programma di controllo JDBC nativo tenta di individuare le perdite dell'istruzione e le gestisce per conto dell'utente. Affidarsi a quel supporto, tuttavia, determina prestazioni più scarse.

A causa della gerarchia di eredità in base alla quale CallableStatement estende PreparedStatement che a sua volta estende Statement, le caratteristiche di ogni interfaccia sono disponibili nella classe che estende l'interfaccia. Ad esempio, le caratteristiche della classe Statement sono anche supportate nelle classi PreparedStatement e CallableStatement. L'eccezione principale è rappresentata dai metodi executeQuery, executeUpdate e execute sulla classe Statement. Questi metodi includono un'istruzione SQL da elaborare dinamicamente e causano eccezioni se si tenta di utilizzarli con gli oggetti PreparedStatement o CallableStatement.

Oggetti Statement:

Si utilizza un oggetto Statement per l'elaborazione di un'istruzione SQL statica e l'ottenimento dei risultati prodotti da questa. È possibile aprire solo un ResultSet alla volta per ciascun oggetto Statement. Tutti i metodi dell'istruzione che elaborano un'istruzione SQL chiudono implicitamente un ResultSet corrente dell'istruzione se ne esiste uno aperto.

Creazione di Statement

Gli oggetti Statement sono creati dagli oggetti Connection con il metodo createStatement. Ad esempio, supponendo che un oggetto Connection denominato conn esista già, la riga seguente di codice crea un oggetto Statement per il passaggio delle istruzioni SQL al database:

```
Statement stmt = conn.createStatement();
```

Specifica delle caratteristiche di ResultSet

Le caratteristiche dei ResultSet sono associate all'istruzione che alla fine li crea. Il metodo Connection.createStatement consente di specificare queste caratteristiche del ResultSet. I seguenti sono alcuni esempi di chiamate valide per il metodo createStatement:

Esempio: il metodo createStatement

```
// Quanto riportato di seguito è nuovo in JDBC 2.0

Statement stmt2 = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATEABLE);

// Quanto segue è nuovo in JDBC 3.0

Statement stmt3 = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY, ResultSet.HOLD_CURSOR_OVER_COMMIT);
```

Per ulteriori informazioni su queste caratteristiche, consultare ResultSet.

Elaborazione di istruzioni

L'elaborazione delle istruzioni SQL con un oggetto Statement si realizza con i metodi executeQuery(), executeUpdate() ed execute().

Restituzione dei risultati delle query SQL

Se deve essere elaborata un'istruzione della query SQL che restituisce un oggetto ResultSet, si dovrebbe utilizzare il metodo executeQuery(). È possibile fare riferimento al programma di esempio che utilizza un metodo executeQuery dell'oggetto Statement per ottenere un ResultSet.

Nota: se un'istruzione SQL elaborata con executeQuery non restituisce un ResultSet, viene emessa una SQLException.

Restituzione di conteggi di aggiornamento per le istruzioni SQL

Se la SQL è nota per essere un'istruzione DDL (Data Definition Language) o un'istruzione DML (Data Manipulation Language) che restituisce un conteggio di aggiornamento, sarebbe opportuno utilizzare il metodo executeUpdate(). Il programma StatementExample utilizza un metodo executeUpdate dell'oggetto Statement.

Elaborazione delle istruzioni SQL nelle quali il ritorno previsto è sconosciuto.

Se il tipo di istruzione SQL non è conosciuto, si dovrebbe utilizzare il metodo execute. Una volta che questo metodo è stato elaborato, il programma di controllo JDBC è in grado di indicare all'applicazione i tipi di risultati che l'istruzione SQL ha creato tramite le chiamate API. Il metodo execute restituisce il valore ″true″ se il risultato contiene almeno un ResultSet e ″false″ se il valore restituito è un conteggio di aggiornamento. Una volta fornite queste informazioni, le applicazioni possono utilizzare getUpdateCount o getResultSet del metodo dell'istruzione per richiamare il valore di ritorno dall'elaborazione dell'istruzione SQL. Il programma StatementExecute utilizza il metodo execute su un oggetto Statement. Questo programma prevede un parametro da inoltrare rappresentato da un'istruzione SQL. Senza considerare il testo della SQL fornita, il programma elabora l'istruzione e determina le informazioni su ciò che è stato elaborato.

Nota: la chiamata al metodo getUpdateCount quando il risultato è un ResultSet restituisce -1. La chiamata al metodo getResultSet quando il risultato è un conteggio di aggiornamento restituisce un valore null.

Metodo cancel

I metodi del programma di controllo JDBC nativo sono sincronizzati per evitare che due sottoprocessi in esecuzione rispetto allo stesso oggetto danneggino l'oggetto. Un'eccezione è rappresentata dal metodo cancel. Il metodo cancel può essere utilizzato da un sottoprocesso per arrestare un'istruzione SQL ad esecuzione prolungata su un altro sottoprocesso in relazione allo stesso oggetto. Non è possibile che il programma di controllo JDBC nativa forzi il sottoprocesso ad arrestare il lavoro in esecuzione; questo può solamente richiedere che vengano arrestate le attività che si stanno effettuando. Per questa ragione, l'arresto di un'istruzione annullata richiede tempo ulteriore. È possibile utilizzare il metodo cancel per arrestare le query SQL in esecuzione sul sistema.

Esempio: utilizzo del metodo executeUpdate dell'oggetto Statement:

Questo è un esempio sul modo in cui utilizzare il metodo executeUpdate dell'oggetto Statement.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import java.util.Properties;

public class StatementExample {

    public static void main(java.lang.String[] args)
    {

        // Suggerimento: caricare quanto segue dall'oggetto proprietà.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Registrare il programma di controllo JDBC nativo. Se il programma di controllo non può
        // essere registrato, la verifica non può continuare.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        try {
```

```
            // Creare le proprietà del collegamento.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Collegarsi al database System i5 locale.
            c = DriverManager.getConnection(URL, properties);

            // Creare un oggetto Statement.
            s = c.createStatement();
            // Cancellare la tabella verifiche se presente. Nota: questo
            // esempio presuppone che MYLIBRARY della raccolta
            // esista sul sistema.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Continuare... probabilmente la tabella non esiste.
            }

            // Eseguire un'istruzione SQL che crea una tabella nel database.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

            // Eseguire alcune istruzioni SQL che inseriscono record nella tabella.
            s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('RICH', 123)");
            s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('FRED', 456)");
            s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('MARK', 789)");

            // Eseguire una query SQL sulla tabella.
            ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

            // Visualizzare tutti i dati nella tabella.
            while (rs.next()) {
                System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("Database processing has failed.");
            System.out.println("Reason: " + sqle.getMessage());
        } finally {
            // Chiudere le risorse del database
            try {
                if (s != null) {
                    s.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Statement.");
            }
             }

            try {
                if (c != null) {
                    c.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Connection.");
            }
        }
    }
}
```

PreparedStatement:

Le PreparedStatement estendono l'interfaccia Statement e forniscono un supporto per l'aggiunta di parametri alle istruzioni SQL.

Le istruzioni SQL inoltrate al database attraversano un processo a due fasi nella restituzione di risultati all'utente. Queste vengono in primo luogo preparate e successivamente elaborate. Con gli oggetti Statement, queste due fasi diventano una fase nelle applicazioni. Le PreparedStatement consentono la separazione di queste due fasi. La fase della preparazione si verifica quando si crea l'oggetto mentre la fase dell'elaborazione si verifica quando si richiama il metodo executeQuery, executeUpdate o execute sull'oggetto PreparedStatement.

La suddivisione dell'elaborazione SQL in fasi separate risulta inutile senza l'aggiunta di contrassegni di parametro. I contrassegni di parametro vengono inseriti in un'applicazione cosicché questa può informare il database che non ha un valore specifico al momento della preparazione, ma che ne fornisce uno prima del periodo di elaborazione. I contrassegni di parametro vengono rappresentati nelle istruzioni SQL da punti interrogativi.

I contrassegni di parametro rendono possibile la creazione di istruzioni SQL generali utilizzate per le richieste specifiche. Ad esempio, si consideri la seguente istruzione di query SQL:

```
SELECT * FROM EMPLOYEE_TABLE WHERE LASTNAME = 'DETTINGER'
```

Questa è un'istruzione SQL specifica che restituisce solo un valore; cioè informazioni su un impiegato di nome Dettinger. Aggiungendo un contrassegno di parametro, è possibile che l'istruzione diventi più flessibile:

```
SELECT * FROM EMPLOYEE_TABLE WHERE LASTNAME = ?
```

Impostando semplicemente il contrassegno di parametro su un valore, è possibile ottenere informazioni su qualsiasi impiegato presente nella tabella.

Le PreparedStatement forniscono miglioramenti significativi delle prestazioni rispetto a Statement perché l'esempio Statement precedente è in grado di attraversare la fase di preparazione solo una volta e successivamente viene elaborato con valori diversi per il parametro.

Nota: l'utilizzo delle PreparedStatement è un requisito per il supporto del lotto di istruzioni del programma di controllo JDBC.

Per ulteriori informazioni sull'utilizzo delle istruzioni preparate, incluso la loro creazione, sulla specifica delle caratteristiche della serie di risultati, sulla gestione delle chiavi generate automaticamente e sull'impostazione dei contrassegni, consultare le seguenti pagine:

Creazione e utilizzo di PreparedStatement:

Il metodo prepareStatement si utilizza per creare nuovi oggetti PreparedStatement. A differenza del metodo createStatement, è necessario che venga fornita l'istruzione SQL quando si crea l'oggetto PreparedStatement. In quella circostanza, l'istruzione SQL viene precompilata per l'utilizzo.

Ad esempio, presupponendo che un oggetto Connection denominato conn esista già, l'esempio seguente crea un oggetto PreparedStatement e prepara l'istruzione SQL per l'elaborazione all'interno del database.

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM EMPLOYEE_TABLE
                                              WHERE LASTNAME = ?");
```

Specifica delle caratteristiche ResultSet e del supporto chiavi generate automaticamente

Così come per il metodo createStatement, il metodo prepareStatement viene sovraccaricato per fornire un supporto per specificare le caratteristiche del ResultSet. Il metodo prepareStatement possiede anche variazioni per la gestione di chiavi create automaticamente. Quelli che seguono sono esempi di chiamate valide al metodo prepareStatement:

Esempio: il metodo prepareStatement

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Nuovo in JDBC 2.0

PreparedStatement ps2 = conn.prepareStatement("SELECT * FROM
    EMPLOYEE_TABLE WHERE LASTNAME = ?",

ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATEABLE);

// Nuovo in JDBC 3.0

PreparedStatement ps3 = conn.prepareStatement("SELECT * FROM
    EMPLOYEE_TABLE WHERE LASTNAME = ?",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,
    ResultSet.HOLD_CURSOR_OVER_COMMIT);

PreparedStatement ps4 = conn.prepareStatement("SELECT * FROM
    EMPLOYEE_TABLE WHERE LASTNAME = ?", Statement.RETURN_GENERATED_KEYS);
```

Gestione dei parametri

Prima che un oggetto PreparedStatement sia elaborato, è necessario impostare ognuno dei contrassegni di parametro su qualche valore. L'oggetto PreparedStatement fornisce un certo numero di metodi per l'impostazione dei parametri. Tutti i metodi sono del formato set<Type>, dove <Type> è un tipo di datiJava. Alcuni esempi di questi metodi includono setInt, setLong, setString, setTimestamp, setNull e setBlob. Quasi tutti questi metodi dispongono di due parametri:

- Il primo parametro è l'indice del parametro all'interno dell'istruzione. I contrassegni di parametro sono numerati, a partire da 1.
- Il secondo parametro rappresenta il valore su cui impostare il parametro. Esistono un paio di metodi set<Type> che possiedono parametri ulteriori come il parametro di lunghezza su setBinaryStream.

Consultare il Javadoc del pacchetto java.sql per ulteriori informazioni. Data l'istruzione SQL preparata nell'esempio precedente per l'oggetto ps, il codice seguente illustra il modo in cui il valore di parametro viene specificato prima dell'elaborazione:

```
ps.setString(1,'Dettinger');
```

Se viene fatto un tentativo di elaborazione di una PreparedStatement con i contrassegni di parametro non impostati, viene emessa una SQLException.

Nota: una volta immessi, i contrassegni di parametro mantengono lo stesso valore tra i processi a meno che non si verifichi la seguente situazione:

- Il valore viene modificato da un'altra chiamata a un metodo set.
- Il valore viene eliminato quando si chiama il metodo clearParameters.

Il metodo clearParameters indica tutti i parametri come non impostati. Dopo che è stata effettuata la chiamata a clearParameters, è necessario che il parametro set venga richiamato per tutti i parametri prima dell'elaborazione successiva.

Supporto ParameterMetaData

Una nuova interfaccia ParameterMetaData consente di richiamare le informazioni su un parametro. Questo supporto è il complemento a ResultSetMetaData ed è simile. Vengono fornite tutte le informazioni come la precisione, la scala, il tipo di dati, il nome del tipo di dati e se il parametro consente valori null.

Esempio: ParameterMetaData:

Questo è un esempio della modalità di utilizzo dell'interfaccia ParameterMetaData per richiamare le informazioni sui parametri.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////////////////////////////////////////////
//
// Esempio ParameterMetaData. Questo programma illustra
// il nuovo supporto JDBC 3.0 per l'acquisizione delle informazioni
// sui parametri in una PreparedStatement.
//
// Sintassi del comando:
//    java PMD
//
//////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di programma di controllo JDBC IBM Developer per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class PMD {

    // Punto di immissione del programma.
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Ottenere l'installazione.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.MYTABLE VALUES(?, ?, ?)");
        ParameterMetaData pmd = ps.getParameterMetaData();

        for (int i = 1; i < pmd.getParameterCount(); i++) {
            System.out.println("Parameter number " + i);
            System.out.println("  Class name is " + pmd.getParameterClassName(i));
            // Nota: modalità relativa a input, output o inout
            System.out.println("  Mode is " + pmd.getParameterClassName(i));
            System.out.println("  Type is " + pmd.getParameterType(i));
            System.out.println("  Type name is " + pmd.getParameterTypeName(i));
            System.out.println("  Precision is " + pmd.getPrecision(i));
            System.out.println("  Scale is " + pmd.getScale(i));
            System.out.println("  Nullable? is " + pmd.isNullable(i));
```

```
            System.out.println("  Signed? is " + pmd.isSigned(i));
        }
    }
}
```

Elaborazione di PreparedStatement:

L'elaborazione di istruzioni SQL con un oggetto PreparedStatement si realizza con i metodi executeQuery, executeUpdate ed execute nello stesso modo in cui vengono elaborati gli oggetti Statement. A differenza delle versioni Statement, nessun parametro viene inoltrato a questi metodi poiché l'istruzione SQL è stata già fornita quando l'oggetto è stato creato. Dal momento che PreparedStatement estende Statement, le applicazioni possono tentare di chiamare le versioni dei metodi executeQuery, executeUpdate ed execute che possiedono un'istruzione SQL. Ne consegue che una SQLException viene emessa.

Restituzione di risultati dalle query SQL

Se un'istruzione di query SQL che restituisce un oggetto ResultSet deve essere elaborata, sarebbe opportuno utilizzare il metodo executeQuery. Il programma PreparedStatementExample utilizza un metodo executeQuery dell'oggetto PreparedStatement per ottenere un ResultSet.

Nota: se un'istruzione SQL elaborata con il metodo executeQuery non restituisce un ResultSet, viene emessa una SQLException.

Restituzione dei conteggi di aggiornamento per le istruzioni SQL

Se la SQL è nota per essere un'istruzione DDL (Data Definition Language) o un'istruzione DML (Data Manipulation Language) che restituisce un conteggio di aggiornamento, sarebbe opportuno utilizzare il metodo executeUpdate. Il programma di esempio PreparedStatementExample utilizza un metodo executeUpdate dell'oggetto PreparedStatement.

Elaborazione delle istruzioni SQL nelle quali il ritorno previsto è sconosciuto

Se il tipo di istruzione SQL non è conosciuto, si dovrebbe utilizzare il metodo execute. Una volta che questo metodo è stato elaborato, il programma di controllo JDBC è in grado di indicare all'applicazione i tipi di risultati che l'istruzione SQL ha creato tramite le chiamate API. Il metodo execute restituisce il valore ″true″ se il risultato contiene almeno un ResultSet e ″false″ se il valore restituito è un conteggio di aggiornamento. Una volta fornite queste informazioni, le applicazioni possono utilizzare i metodi di istruzione getUpdateCount o getResultSet per richiamare il valore di ritorno dall'elaborazione dell'istruzione SQL.

Nota: la chiamata al metodo getUpdateCount quando il risultato è un ResultSet restituisce -1. La chiamata al metodo getResultSet quando il risultato è un conteggio di aggiornamento restituisce un valore null.

Esempio: utilizzo di PreparedStatement per ottenere un ResultSet:

Questo rappresenta un esempio dell'utilizzo del metodo executeQuery di un oggetto PreparedStatement per ottenere un ResultSet.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import java.util.Properties;

public class PreparedStatementExample {

    public static void main(java.lang.String[] args)
    {
```

```
// Caricare quanto segue da un oggetto proprietà.
String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
String URL    = "jdbc:db2://*local";

// Registrare il programma di controllo JDBC nativo. Se il programma di controllo non può
// essere registrato, la verifica non può continuare.
try {
    Class.forName(DRIVER);
} catch (Exception e) {
    System.out.println("Driver failed to register.");
    System.out.println(e.getMessage());
    System.exit(1);
}

Connection c = null;
Statement s = null;

//    Questo programma crea una tabella utilizzata
//    successivamente dalle istruzioni preparate.
try {
    // Creare le proprietà del collegamento.
    Properties properties = new Properties ();
    properties.put ("user", "userid");
    properties.put ("password", "password");

    // Collegarsi al database locale.
    c = DriverManager.getConnection(URL, properties);

    // Creare un oggetto Statement.
    s = c.createStatement();
    // Cancellare la tabella di verifica se esistente. Notare che
    // questo esempio presuppone che l'intera MYLIBRARY della
    // raccolta esista sul sistema.
    try {
        s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
    } catch (SQLException e) {
        // Continuare... probabilmente la tabella non esiste.
    }

    // Eseguire un'istruzione SQL che crea una tabella nel database.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Chiudere le risorse del database
    try {
        if (s != null) {
            s.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}


//    Questo programma utilizza poi un'istruzione preparata per inserire
//    molte righe nel database.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Creare un oggetto PreparedStatement utilizzato per inserire dati
    // nella tabella.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");
```

```
            for (int i = 0; i < nameArray.length; i++) {
                ps.setString(1, nameArray[i]);      // Impostare il nome dalla propria schiera.
                ps.setInt(2, i+1);                  // Impostare l'ID.
                ps.executeUpdate();
            }

        } catch (SQLException sqle) {
            System.out.println("Database processing has failed.");
            System.out.println("Reason: " + sqle.getMessage());
        } finally {
            // Chiudere le risorse del database
            try {
                if (ps != null) {
                    ps.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Statement.");
            }
        }


        //    Utilizzare un'istruzione preparata per interrogare la tabella
        //    database creata e restituire i dati che derivano da essa. In
        //    questo esempio, il parametro usato viene impostato arbitrariamente
        //    su 5, ciò indica che vengono restituite tutte le righe in cui il campo ID
        //    sia minore o uguale a 5.
        try {
            ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
                                    "WHERE ID <= ?");

            ps.setInt(1, 5);

            // Eseguire una query SQL sulla tabella.
            ResultSet rs = ps.executeQuery();
            // Visualizzare tutti i dati nella tabella.
            while (rs.next()) {
                System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("Database processing has failed.");
            System.out.println("Reason: " + sqle.getMessage());
        } finally {
            // Chiudere le risorse del database
            try {
                if (ps != null) {
                    ps.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Statement.");
            }

            try {
                if (c != null) {
                    c.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Connection.");
            }

        }
    }
}
```

CallableStatement:

L'interfaccia CallableStatement JDBC estende PreparedStatement e fornisce il supporto per i parametri di immissione/emissione e di emissione. L'interfaccia CallableStatement dispone inoltre del supporto per i parametri di immissione fornito dall'interfaccia PreparedStatement.

L'interfaccia CallableStatement consente l'utilizzo delle istruzioni SQL per chiamare le procedure memorizzate. Le procedure memorizzate sono programmi che hanno un'interfaccia database. Tali programmi dispongono di quanto segue:

- Essi possono disporre di parametri di immissione ed emissione o di parametri che sono sia di immissione che di emissione.
- Essi possono disporre di un valore di ritorno.
- Essi dispongono della capacità di restituire più ResultSet.

Concettualmente in JDBC, una chiamata ad una procedura memorizzata è una singola chiamata al database, ma il programma associato alla procedura memorizzata può elaborare centinaia di richieste database. Il programma di procedura memorizzata può inoltre eseguire una quantità di altre attività programmatiche non effettuate, di solito, con le istruzioni SQL.

Dato che le CallableStatement seguono il modello PreparedStatement di separazione delle fasi di preparazione ed elaborazione, esse hanno il potenziale per un nuovo utilizzo ottimizzato (consultare "PreparedStatement" a pagina 100 per ulteriori dettagli).Dato che le istruzioni SQL di una procedura memorizzata sono reciprocamente collegate in un programma, vengono elaborate come SQL statico ed è possibile ottenere ulteriori vantaggi delle prestazioni in quel modo. La compressione di una gran quantità di lavoro database in una singola chiamata riutilizzabile al database è un esempio di utilizzo delle procedure memorizzate in modo ottimale. Solo questa chiamata giunge sulla rete all'altro sistema, ma la richiesta può compiere una quantità considerevole di lavoro sul sistema remoto.

Creazione di CallableStatement

Il metodo prepareCall viene utilizzato per creare nuovi oggetti CallableStatement. Come avviene con il metodo prepareStatement, è necessario fornire l'istruzione SQL quando viene creato l'oggetto CallableStatement. In quel momento, l'istruzione SQL è precompilata. Ad esempio, presumendo che un oggetto Connection chiamato conn già esiste, quanto segue crea un oggetto CallableStatement e completa la fase di preparazione per rendere pronta l'istruzione SQL per l'elaborazione nel database:

```
PreparedStatement ps = conn.prepareStatement("? = CALL ADDEMPLOYEE(?, ?, ?");
```

La procedura memorizzata ADDEMPLOYEE acquisisce parametri di immissione per un nuovo nome impiegato, il relativo numero di previdenza sociale e l'ID utente del dirigente. Da queste informazioni, è possibile aggiornare più tabelle di database della società con informazioni sull'impiegato come ad esempio la data di assunzione, divisione, reparto e via di seguito. Inoltre, una procedura memorizzata è un programma che può generare indirizzi e-mail e ID utente standard per quell'impiegato. La procedura memorizzata può inoltre inviare un'e-mail al dirigente responsabile dell'assunzione con parole d'ordine e nomi utente iniziali; il dirigente responsabile dell'assunzione può quindi fornire le informazioni all'impiegato.

La procedura memorizzata ADDEMPLOYEE è impostata in modo da avere un valore di ritorno. Il codice di ritorno può essere un codice di errore o di esito positivo che il programma chiamante può utilizzare quando si verifica un errore. È inoltre possibile definire il valore di ritorno come numero ID della società del nuovo impiegato. Infine, è possibile che il programma della procedura memorizzata abbia elaborato delle query internamente e abbia lasciato i ResultSet derivati da quelle query, aperti e disponibili per il programma chiamante. È logico interrogare tutte le informazioni sul nuovo impiegato e renderle disponibili al chiamante tramite un ResultSet restituito.

Nelle seguenti sezioni viene illustrato come compiere ognuno di questi tipi di attività.

Specifica delle caratteristiche ResultSet e del supporto chiavi generate automaticamente

Come avviene con createStatement e prepareStatement, esistono più versioni di prepareCall che forniscono supporto per specificare le caratteristiche di ResultSet. A differenza di prepareStatement, il metodo prepareCall non fornisce variazioni per gestire le chiavi generate automaticamente da CallableStatement (JDBC 3.0 non supporta questo concetto.) Seguono esempi di chiamate valide al metodo prepareCall:

Esempio: il metodo prepareCall

```
// Quanto riportato di seguito è nuovo in JDBC 2.0

CallableStatement cs2 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE);

// Nuovo in JDBC 3.0

CallableStatement cs3 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,
    ResultSet.HOLD_CURSOR_OVER_COMMIT);
```

Gestione dei parametri

Come stabilito, gli oggetti CallableStatement possono assumere tre tipi di parametri:

- **IN**

  I parametri IN vengono gestiti nello stesso modo delle PreparedStatement. Si utilizzano i vari metodi set della classe PreparedStatement ereditata per impostare i parametri.
- **OUT**

  I parametri OUT vengono gestiti con il metodo registerOutParameter. Il formato più comune di registerOutParameter acquisisce un parametro dell'indice come primo parametro e un tipo SQL come secondo parametro. Ciò indica al programma di controllo JDBC quali dati aspettarsi dal parametro quando l'istruzione viene elaborata. Esistono altre due variazioni sul metodo registerOutParameter che è possibile trovare nel pacchetto Javadoc java.sql.
- **INOUT**

  I parametri INOUT richiedono che venga eseguito il lavoro sia per i parametri IN che per i parametri OUT. Per ogni parametro INOUT, è necessario chiamare un metodo set e il metodo registerOutParameter prima che sia possibile elaborare l'istruzione. La mancata impostazione o registrazione di un qualsiasi parametro dà come risultato l'emissione di una SQLException quando l'istruzione viene elaborata.

Per ulteriori informazioni, fare riferimento a "Esempio: creazione di una procedura con i parametri di immissione ed emissione" a pagina 112.

Come avviene con PreparedStatement, i valori di parametro CallableStatement rimangono gli stessi tra le elaborazioni a meno che non venga chiamato nuovamente un metodo set. Il metodo clearParameters non influenza i parametri registrati per l'emissione. Dopo aver eseguito la chiamata di clearParameters, è necessario impostare nuovamente un valore per tutti i parametri IN, tutti i parametri OUT invece, non necessitano di una nuova impostazione.

Nota: il concetto di parametro non va confuso con l'indice del contrassegno del parametro. Una chiamata della procedura memorizzata prevede l'inoltro di un certo numero di parametri. Un'istruzione SQL specifica contiene caratteri ? (contrassegni di parametro) che rappresentano i valori forniti nel tempo di esecuzione. Per comprendere la differenza dei due concetti, considerare il seguente esempio:

```
CallableStatement cs = con.prepareCall("CALL PROC(?, "SECOND", ?)");

cs.setString(1, "First");     //Parameter marker 1, Stored procedure parm 1

cs.setString(2, "Third");     //Parameter marker 2, Stored procedure parm 3
```

Accesso ai parametri di procedura memorizzata per nome

I parametri nelle procedure memorizzate hanno nomi associati ad essi come mostra la seguente dichiarazione della procedura memorizzata:

Esempio: parametri della procedura memorizzata

```
CREATE
PROCEDURE MYLIBRARY.APROC
    (IN PARM1 INTEGER)
LANGUAGE SQL SPECIFIC MYLIBRARY.APROC
BODY: BEGIN
    <Perform a task here...>
END BODY
```

Esiste un singolo parametro del numero intero con il nome PARM1. In JDBC 3.0, esiste un supporto per specificare i parametri della procedura memorizzata sia per nome che per indice. Il codice per impostare CallableStatement per questa procedura è il seguente:

```
CallableStatement cs = con.prepareCall("CALL APROC(?)");

cs.setString("PARM1", 6);     //Imposta il parametro immissione nell'indice 1 (PARM1) su 6.
```

Elaborazione delle CallableStatement:

L'elaborazione di chiamate ad una procedura memorizzata SQL con un oggetto CallableStatement JDBC viene compiuta con gli stessi metodi utilizzati con un oggetto PreparedStatement.

Restituzione dei risultati per le procedure memorizzate

Se un'istruzione della query SQL viene elaborata all'interno di una procedura memorizzata, è possibile rendere disponibili i risultati della query per il programma che chiama la procedura memorizzata. È inoltre possibile chiamare più query nell'ambito di una procedura memorizzata e che il programma chiamante elabori tutti i ResultSet disponibili.

Per ulteriori informazioni, consultare "Esempio: creazione di una procedura con più ResultSet" a pagina 110.

Nota: se una procedura memorizzata viene elaborata con executeQuery e non restituisce ResultSet, viene emessa una SQLException.

Accesso ai ResultSet simultaneamente

La restituzione dei risultati per le procedure memorizzate tratta i ResultSets e le procedure memorizzate e fornisce un esempio che si applica a tutti i release JDK (Java Development Kit). Nell'esempio, i ResultSet vengono elaborati in ordine dal primo ResultSet aperto dalla procedura memorizzata all'ultimo ResultSet aperto. Ogni ResultSet viene chiuso prima di aprire il successivo.

In JDK 1.4 e le versioni successive esiste un supporto per gestire i ResultSet dalle procedure memorizzate simultaneamente.

Nota: questa funzione è stata aggiunta al supporto del sistema sottostante tramite la CLI (Command Line Interface) in V5R2. Come risultato, l'esecuzione di JDK 1.4 o di una versione successiva su un sistema precedente la V5R2 non dispone di questo supporto.

Restituzione dei conteggi di aggiornamento per le procedure memorizzate

La restituzione dei conteggi di aggiornamento per le procedure memorizzate è una funzione discussa nella specifica JDBC, ma non è attualmente supportata sulla piattaforma System i. Non esiste alcun modo per restituire più conteggi di aggiornamento da una chiamata alla procedura memorizzata. Se è necessario un conteggio di aggiornamento da un'istruzione SQL elaborata all'interno di una procedura memorizzata, esistono due modi per restituire il valore:

- Restituire il valore come parametro di emissione.
- Passare nuovamente il valore come valore di ritorno dal parametro. Questo è un caso speciale di un parametro di emissione. Consultare Elaborazione delle procedure memorizzate che dispongono di un valore di ritorno per ulteriori informazioni.

Elaborazione delle procedure memorizzate dove il valore previsto è sconosciuto

Se non si conoscono i risultati da una chiamata alla procedura memorizzata, sarebbe opportuno che venga utilizzato il metodo execute. Una volta elaborato tale metodo, il programma di controllo JDBC può comunicare all'applicazione quali tipi di risultati sono stati generati dalla procedura memorizzata tramite le chiamate API. Il metodo execute restituisce ″true″ se il risultato è uno o più ResultSet. I conteggi di aggiornamento non provengono dalle chiamate della procedura memorizzata.

Elaborazione delle procedure memorizzate che dispongono di un valore di ritorno

La piattaforma System i supporta procedure memorizzate che dispongono di un valore di ritorno simile ad un valore di ritorno di una funzione. Tale valore da una procedura memorizzata è identificato come altri contrassegni di parametri e come se fosse assegnato dalla chiamata alla procedura memorizzata. Segue un esempio di quanto detto:

```
? = CALL MYPROC(?, ?, ?)
```

Il valore di ritorno da una chiamata alla procedura memorizzata è sempre di tipo intero e deve essere registrato come ogni altro parametro di emissione.

Per ulteriori informazioni, consultare "Esempio: creazione di una procedura con i valori di ritorno" a pagina 113.

Esempio: creazione di una procedura con più ResultSet:

Questo esempio mostra come accedere a un database e creare quindi una procedura con più ResultSet utilizzando JDBC.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import java.util.Properties;

public class CallableStatementExample1 {

    public static void main(java.lang.String[] args) {

        // Registrare il programma di controllo JDBC nativa. Se non è possibile
        // registrare il programma di controllo, la verifica non può continuare.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Creare le proprietà di collegamento
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Collegarsi al database server locale
```

```
Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

Statement s = c.createStatement();

// Creare una procedura con più ResultSets.
String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX1 " +
             "RESULT SET 2 LANGUAGE SQL READS SQL DATA SPECIFIC MYLIBRARY.SQLSPEX1 " +
             "EX1: BEGIN " +
             "   DECLARE C1 CURSOR FOR SELECT * FROM QSYS2.SYSPROCS " +
             "               WHERE SPECIFIC_SCHEMA = 'MYLIBRARY';  " +
             "   DECLARE C2 CURSOR FOR SELECT * FROM QSYS2.SYSPARMS  " +
             "               WHERE SPECIFIC_SCHEMA = 'MYLIBRARY';  " +
             "   OPEN C1; " +
             "   OPEN C2; " +
             "   SET RESULT SETS CURSOR C1, CURSOR C2; " +
             "END EX1 ";

try {
    s.executeUpdate(sql);
    } catch (SQLException e) {
    // NOTA:  in questo momento stiamo ignorando l'errore. Presupponiamo
    //        che l'unico motivo per cui ha avuto esito negativo consista
    //        nel fatto che la procedura esiste già. Altri motivi di questo
    //        errore possono essere dati dal fatto che il compilatore C
    //        non è stato rilevato per compilare la procedura oppure la
    //        raccolta MYLIBRARY non esiste sul sistema.
}
s.close();

// Ora usare JDBC per eseguire la procedura e richiamare i risultati. In
// questo caso stiamo richiamando le informazioni sulle procedure memorizzate
// di 'MYLIBRARY' (che costituiscono il punto in cui è stata creata questa procedura,
// quindi assicurarsi che esista qualche elemento da richiamare.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX1");

ResultSet rs = cs.executeQuery();

// Ora abbiamo il primo oggetto ResultSet che ha lasciato la procedura
// memorizzata aperta. Utilizzarlo.
int i = 1;
while (rs.next()) {
    System.out.println("MYLIBRARY stored procedure
                       " + i + " is " + rs.getString(1) + "." +
                       rs.getString(2));
    i++;

}
System.out.println("");


// Ora richiamare l'oggetto ResultSet successivo dal sistema - quello precedente
// viene chiuso automaticamente.
if (!cs.getMoreResults()) {
    System.out.println("Something went wrong.  There should have
                        been another ResultSet, exiting.");
    System.exit(0);
}
rs = cs.getResultSet();

 // Ora abbiamo il secondo oggetto ResultSet che ha lasciato la procedura
// memorizzata aperta. Utilizzarlo.
i = 1;
while (rs.next()) {
    System.out.println("MYLIBRARY procedure " + rs.getString(1)
                        + "." + rs.getString(2) +
                       " parameter:  " + rs.getInt(3) + " direction:
                       " + rs.getString(4) +
```

```
                                " data type: " + rs.getString(5));
                i++;

            }

            if (i == 1) {
                System.out.println("None of the stored procedures have any parameters.");
            }


            if (cs.getMoreResults()) {
                System.out.println("Something went wrong,
                                    there should not be another ResultSet.");
                System.exit(0);
            }

            cs.close();  // close the CallableStatement object
            c.close();   // close the Connection object.

        } catch (Exception e) {
            System.out.println("Something failed..");
            System.out.println("Reason: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Esempio: creazione di una procedura con i parametri di immissione ed emissione:

Quest'esempio mostra come accedere ad un database utilizzando JDBC e creare quindi una procedura con parametri di immissione ed emissione.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import java.util.Properties;

public class CallableStatementExample2 {

    public static void main(java.lang.String[] args) {

        // Registrare il programma di controllo JDBC nativa. Se non è possibile
        // registrare il programma di controllo, la verifica non può continuare.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Creare le proprietà di collegamento
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Collegarsi al database server locale
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Creare una procedura con i parametri in, out e in/out.
            String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX2 " +
                         "(IN P1 INTEGER, OUT P2 INTEGER, INOUT P3 INTEGER) " +
                         "LANGUAGE SQL SPECIFIC MYLIBRARY.SQLSPEX2 " +
                         "EX2: BEGIN " +
                         "   SET P2 = P1 + 1; " +
                         "   SET P3 = P3 + 1; " +
                         "END EX2 ";

            try {
```

```
            s.executeUpdate(sql);
        } catch (SQLException e) {
            // NOTA:  in questo momento stiamo ignorando l'errore. Presupponiamo
            //        che l'unico motivo per cui ha avuto esito negativo consista
            //        nel fatto che la procedura esiste già. Altri motivi di questo
            //        errore possono essere dati dal fatto che il compilatore C
            //        non è stato rilevato per compilare la procedura oppure la
            //        raccolta MYLIBRARY non esiste sul sistema.
        }
        s.close();

        // Preparare un'istruzione callable usata per eseguire la procedura.
        CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX2(?, ?, ?)");

        // Tutti i parametri di immissione devono essere impostati e quelli di emissione
        // devono essere registrati. Notare che ciò indica che devono essere eseguite
        // due chiamate per un parametro input output.
        cs.setInt(1, 5);
        cs.setInt(3, 10);
        cs.registerOutParameter(2, Types.INTEGER);
        cs.registerOutParameter(3, Types.INTEGER);

        // Eseguire la procedura
        cs.executeUpdate();

        // Verificare che i parametri output dispongano dei valori desiderati.
        System.out.println("The value of P2 should be P1 (5) + 1 = 6.  -->  " + cs.getInt(2));
        System.out.println("The value of P3 should be P3 (10) + 1 = 11.  -->  " + cs.getInt(3));

        cs.close();  // chiudere l'oggetto CallableStatement
        c.close();   // chiudere l'oggetto Connection.

    } catch (Exception e) {
        System.out.println("Something failed..");
        System.out.println("Reason: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

Esempio: creazione di una procedura con i valori di ritorno:

Questo esempio mostra come accedere a un database utilizzando JDBC e creare quindi una procedura con i valori di ritorno.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import java.util.Properties;

public class CallableStatementExample3 {

    public static void main(java.lang.String[] args) {

        // Registrare il programma di controllo JDBC nativo. Se non è possibile
        // registrare il programma di controllo, la verifica non può continuare.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Creare le proprietà di collegamento
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Collegarsi al database server locale
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);
```

```
            Statement s = c.createStatement();

            // Creare una procedura con un valore di ritorno.
            String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX3 " +
                         " LANGUAGE SQL SPECIFIC MYLIBRARY.SQLSPEX3 " +
                         " EX3: BEGIN " +
                         "      RETURN 1976; " +
                         " END EX3 ";

            try {
                s.executeUpdate(sql);
            } catch (SQLException e) {
                // NOTA:  l'errore viene ignorato. Si presuppone che l'unico motivo
                //        per cui ha avuto esito negativo sia dovuto al fatto che
                //        la procedura esiste già. Altri motivi per cui si è verificato un
                //        errore possono essere dati dal fatto che il compilatore C
                //        non è stato rilevato per compilare la procedura oppure la
                //        raccolta MYLIBRARY non esiste sul sistema.
            }
            s.close();

            // Preparare un'istruzione callable usata per eseguire la procedura.
            CallableStatement cs = c.prepareCall("? = CALL MYLIBRARY.SQLSPEX3");

            // È ancora necessario registrare il parametro output.
            cs.registerOutParameter(1, Types.INTEGER);

            // Eseguire la procedura.
            cs.executeUpdate();

            // Mostrare che è stato restituito il valore corretto.
            System.out.println("The return value
                               should always be 1976 for this example:
                               -->  " + cs.getInt(1));

            cs.close();  // chiudere l'oggetto CallableStatement
            c.close();   // chiudere l'oggetto Connection.

        } catch (Exception e) {
            System.out.println("Something failed..");
            System.out.println("Reason: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

ResultSet

L'interfaccia ResultSet fornisce l'accesso ai risultati generati eseguendo delle query. Concettualmente, è possibile concepire i dati di un ResultSet come una tabella con un numero specifico di colonne e un numero specifico di righe. Per impostazione predefinita, le righe della tabella vengono richiamate in sequenza. All'interno di una riga, è possibile accedere ai valori della colonna in qualsiasi ordine.

Caratteristiche del ResultSet:

Questo argomento descrive le caratteristiche di ResultSet come i tipi di ResultSet, la simultaneità, la capacità di chiudere il ResultSet eseguendo il commit dell'oggetto di collegamento e la specifica delle caratteristiche di ResultSet.

Per impostazione predefinita tutti i ResultSet creati hanno una tipologia di solo inoltro, una simultaneità di sola lettura e i cursori sono congelati sui limiti del commit. Un'eccezione è rappresentata dalle modifiche che WebSphere apporta correntemente all'impostazione predefinita della conservabilità del

cursore in modo tale che i cursori siano implicitamente chiusi quando sottoposti a commit. Queste caratteristiche sono configurabili tramite metodi accessibili negli oggetti Statement, PreparedStatement e CallableStatement.

Tipi di ResultSet

Il tipo di ResultSet specifica quanto segue sul ResultSet:

- Se il ResultSet si può scorrere.
- I tipi di ResultSet JDBC (Java Database Connectivity) che sono definiti da costanti sull'interfaccia ResultSet.

Le definizioni di questi tipi di ResultSet sono le seguenti:

TYPE_FORWARD_ONLY
Un cursore che è possibile utilizzare solo per elaborare un ResultSet dall'inizio alla fine. Questo è un tipo predefinito.

TYPE_SCROLL_INSENSITIVE
Un cursore che è possibile utilizzare per scorrere un ResultSet. Questo tipo di cursore non è sensibile alle modifiche apportate al database mentre è aperto. Esso contiene righe che soddisfano la query quando la query è stata elaborata o quando viene effettuata una selezione multipla sui dati.

TYPE_SCROLL_SENSITIVE
Un cursore che è possibile utilizzare per scorrere in vari modi un ResultSet. Questo tipo di cursore è sensibile alle modifiche apportate al database mentre è aperto. Le modifiche al database hanno un impatto diretto sui dati del ResultSet.

I ResultSet JDBC 1.0 hanno un'impostazione sempre di solo inoltro. I cursori che si possono aggiornare sono aggiunti in JDBC 2.0.

Nota: le proprietà di collegamento blocco abilitato e dimensione blocco influiscono sul grado di sensibilità di un cursore TYPE_SCROLL_SENSITIVE. Il blocco migliora le prestazioni memorizzando nella cache i dati a livello del programma di controllo JDBC stesso.

Simultaneità

La simultaneità determina se è possibile aggiornare il ResultSet. I tipi vengono nuovamente definiti da costanti nell'interfaccia ResultSet. Le impostazione disponibili della simultaneità sono le seguenti:

CONCUR_READ_ONLY
È possibile utilizzare un ResultSet solo per la lettura dei dati esterni al database. Questa risulta un'impostazione predefinita.

CONCUR_UPDATEABLE
Un ResultSet che consente di apportare modifiche ad esso. È possibile posizionare queste modifiche nel database sottostante.

I ResultSet JDBC 1.0 hanno un'impostazione sempre di solo inoltro. ResultSet aggiornabili sono stati aggiunti nel JDBC 2.0.

Nota: secondo la specifica JDBC, è consentito al programma di controllo JDBC di modificare il tipo di ResultSet dell'impostazione di simultaneità del ResultSet se non è possibile utilizzare i valori contemporaneamente. In tali casi, il programma di controllo JDBC invia un messaggio di avvertenza sull'oggetto Connection.

Esiste una situazione in cui l'applicazione specifica un ResultSet TYPE_SCROLL_INSENSITIVE, CONCUR_UPDATEABLE. La non sensibilità viene implementata nel motore del database facendo una

copia dei dati. All'utente non è quindi consentito di apportare aggiornamenti tramite quella copia al database sottostante. Se si specifica questa combinazione, il programma di controllo modifica la sensibilità a TYPE_SCROLL_SENSITIVE e crea un messaggio di avvertenza indicante che la richiesta è stata modificata.

Conservabilità

La caratteristica di conservabilità determina se la chiamata del commit sull'oggetto Connection chiude il ResultSet. L'API JDBC per la gestione di tale caratteristica è nuova nella versione 3.0. Il programma di controllo JDBC nativo, tuttavia, fornisce una proprietà di collegamento per vari release che consente di specificare quell'impostazione predefinita per tutti i ResultSet creati sotto il collegamento. Il supporto API sostituisce qualsiasi impostazione relativa alla proprietà di collegamento. I valori relativi alla caratteristica di conservabilità sono definiti dalle costanti del ResultSet e sono i seguenti:

HOLD_CURSOR_OVER_COMMIT
Tutti i cursori aperti rimangono aperti quando viene richiamata la clausola commit. Questo rappresenta il valore predefinito del JDBC nativo.

CLOSE_CURSORS_ON_COMMIT
Tutti i cursori aperti vengono chiusi quando viene richiamata la clausola commit.

Nota: il richiamo del rollback su un collegamento chiude sempre tutti i cursori aperti. Questo rappresenta un fatto poco conosciuto, ma una modalità molto comune che il database utilizza per gestire i cursori.

Secondo la specifica JDBC, l'impostazione predefinita per la conservabilità del cursore è definita dall'implementazione. Alcune piattaforme scelgono di utilizzare CLOSE_CURSORS_ON_COMMIT come impostazione predefinita. Ciò non rappresenta un problema per la maggior parte delle applicazioni, ma è necessario tenere presente ciò che il programma di controllo che si sta gestendo esegue se vengono gestiti cursori nei limiti di commit. Il programma di controllo JDBC di IBM Toolbox per Java utilizza anche il valore predefinito HOLD_CURSORS_ON_COMMIT, ma il programma di controllo JDBC per UDB per Windows NT possiede il valore predefinito CLOSE_CURSORS_ON_COMMIT.

Specifica delle caratteristiche di ResultSet

Le caratteristiche di un ResultSet non si modificano una volta che l'oggetto ResultSet è stato creato. Perciò le caratteristiche vengono specificate prima della creazione dell'oggetto. È possibile specificare queste caratteristiche tramite variazioni dei metodi createStatement, prepareStatement e prepareCall.

Nota: esistono metodi ResultSet per ottenere il tipo di ResultSet e la simultaneità del ResultSet, ma non esiste alcun metodo per ottenere la conservabilità del ResultSet.

Concetti correlati

"Oggetti Statement" a pagina 98
Si utilizza un oggetto Statement per l'elaborazione di un'istruzione SQL statica e l'ottenimento dei risultati prodotti da questa. È possibile aprire solo un ResultSet alla volta per ciascun oggetto Statement. Tutti i metodi dell'istruzione che elaborano un'istruzione SQL chiudono implicitamente un ResultSet corrente dell'istruzione se ne esiste uno aperto.

"CallableStatement" a pagina 106
L'interfaccia CallableStatement JDBC estende PreparedStatement e fornisce il supporto per i parametri di immissione/emissione e di emissione. L'interfaccia CallableStatement dispone inoltre del supporto per i parametri di immissione fornito dall'interfaccia PreparedStatement.

"PreparedStatement" a pagina 100
Le PreparedStatement estendono l'interfaccia Statement e forniscono un supporto per l'aggiunta di parametri alle istruzioni SQL.

"Movimenti del cursore" a pagina 121
I programmi di controllo JDBC (System i Java Database Connectivity) supportano i ResultSet scorribili.

Con un ResultSet che è possibile scorrere, si possono elaborare le righe di dati in qualsiasi ordine utilizzando una serie di metodi di posizionamento del cursore.

Attività correlate

"Modifica dei ResultSet" a pagina 124
Con i programmi di controllo JDBC System i, è possibile modificare i ResultSet eseguendo varie attività.

Riferimenti correlati

"Proprietà di collegamento del programma di controllo JDBC" a pagina 45
La tabella di seguito riportata contiene le proprietà di collegamento del programma di controllo JDBC e i relativi valori e descrizioni.

"Proprietà DataSource" a pagina 62
Per ciascuna proprietà di collegamento del programma di controllo JDBC, esiste un corrispondente metodo di origine dati. Questa tabella contiene le proprietà di origine dati valide.

Esempio: ResultSet sensibili e non sensibili:

L'esempio seguente indica le differenze tra i ResultSet sensibili e non sensibili quando vengono inserite righe in una tabella.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class Sensitive {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive test = new Sensitive();

        test.setup();
        test.run("sensitive");
        test.cleanup();

        test.setup();
        test.run("insensitive");
        test.cleanup();
    }


    public void setup() {

        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("drop table cujosql.sensitive");
            } catch (SQLException e) {
                // Ignorato.
            }

            s.executeUpdate("create table cujosql.sensitive(col1 int)");
            s.executeUpdate("insert into cujosql.sensitive values(1)");
            s.executeUpdate("insert into cujosql.sensitive values(2)");
            s.executeUpdate("insert into cujosql.sensitive values(3)");
            s.executeUpdate("insert into cujosql.sensitive values(4)");
            s.executeUpdate("insert into cujosql.sensitive values(5)");
```

```
        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {
        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

        // Selezionare i cinque valori presenti.
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        System.out.println("fetched the five rows...");

        // Nota:  se si seleziona l'ultima riga, ResultSet sembra
        //        chiuso e le nuove righe successive che vengono aggiunte
        //        non vengono riconosciute.

        // Consentire un'altra istruzione per l'inserimento di un nuovo valore.
        Statement s2 = connection.createStatement();
        s2.executeUpdate("insert into cujosql.sensitive values(6)");
        s2.close();

        // Se una riga viene riconosciuta si basa sull'impostazione della sensibilità.
        if (rs.next()) {
            System.out.println("There is a row now: " + rs.getInt(1));
        } else {
            System.out.println("No more rows.");
        }


    } catch (SQLException e) {
        System.out.println("SQLException exception: ");
        System.out.println("Message:....." + e.getMessage());
        System.out.println("SQLState:...." + e.getSQLState());
        System.out.println("Vendor Code:." + e.getErrorCode());
        System.out.println("-----------------------------------");
```

```
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Esempio: sensibilità del ResultSet:

Il seguente esempio indica come una modifica è in grado di influire su una clausola where di un'istruzione SQL in base alla sensibilità del ResultSet.

Alcune delle formattazioni di questo esempio possono non essere corrette al fine di inserire questo esempio in una pagina stampata.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class Sensitive2 {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive2 test = new Sensitive2();

        test.setup();
        test.run("sensitive");
        test.cleanup();

        test.setup();
        test.run("insensitive");
        test.cleanup();
    }


    public void setup() {

        try {
            System.out.println("Native JDBC used");
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("drop table cujosql.sensitive");
            } catch (SQLException e) {
                // Ignorato.
```

```
        }

        s.executeUpdate("create table cujosql.sensitive(col1 int)");
        s.executeUpdate("insert into cujosql.sensitive values(1)");
        s.executeUpdate("insert into cujosql.sensitive values(2)");
        s.executeUpdate("insert into cujosql.sensitive values(3)");
        s.executeUpdate("insert into cujosql.sensitive values(4)");
        s.executeUpdate("insert into cujosql.sensitive values(5)");


        try {
            s.executeUpdate("drop table cujosql.sensitive2");
        } catch (SQLException e) {
            // Ignorato.
        }

        s.executeUpdate("create table cujosql.sensitive2(col2 int)");
        s.executeUpdate("insert into cujosql.sensitive2 values(1)");
        s.executeUpdate("insert into cujosql.sensitive2 values(2)");
        s.executeUpdate("insert into cujosql.sensitive2 values(3)");
        s.executeUpdate("insert into cujosql.sensitive2 values(4)");
        s.executeUpdate("insert into cujosql.sensitive2 values(5)");

        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select col1, col2 From cujosql.sensitive,
                                       cujosql.sensitive2 where col1 = col2");

        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));

        System.out.println("fetched the four rows...");
```

```
            // Un'altra istruzione crea un valore che non corrisponde alla clausola where.
            Statement s2 =
              connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
              ResultSet.CONCUR_UPDATEABLE);
            ResultSet rs2 = s2.executeQuery("select *
            from cujosql.sensitive where col1 = 5 FOR UPDATE");
            rs2.next();
            rs2.updateInt(1, -1);
            rs2.updateRow();
            s2.close();

            if (rs.next()) {
                System.out.println("There is still a row: " + rs.getInt(1));
            } else {
                System.out.println("No more rows.");
            }

        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("----------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other
            than an SQLException was thrown: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Movimenti del cursore:

I programmi di controllo JDBC (System i Java Database Connectivity) supportano i ResultSet scorribili. Con un ResultSet che è possibile scorrere, si possono elaborare le righe di dati in qualsiasi ordine utilizzando una serie di metodi di posizionamento del cursore.

Il metodo ResultSet.next viene utilizzato per spostarsi in un ResultSet di una riga alla volta. Con JDBC (Java Database Connectivity) 2.0, i programmi di controllo System i JDBC supportano i ResultSet scorribili. I ResultSet che si possono scorrere consentono l'elaborazione delle righe di dati in qualsiasi ordine utilizzando i metodi previous, absolute, relative, first e last.

Per impostazione predefinita, i ResultSet JDBC sono sempre di solo inoltro, il che significa che l'unico metodo di posizionamento del cursore valido da richiamare è next(). È necessario richiedere esplicitamente un ResultSet che è possibile scorrere. Consultare Tipi di ResultSet per ulteriori informazioni.

Con un ResultSet che è possibile scorrere, si possono utilizzare i seguenti metodi di posizionamento del cursore:

Metodo	Descrizione
Next	Questo metodo sposta il cursore in avanti di una riga nel ResultSet. Il metodo restituisce ″true″ se il cursore viene posizionato su una riga valida, altrimenti restituisce ″false″.
Previous	Il metodo sposta il cursore indietro di una riga nel ResultSet. Il metodo restituisce ″true″ se il cursore viene posizionato su una riga valida, altrimenti restituisce ″false″.
First	Il metodo sposta il cursore alla prima riga nel ResultSet. Il metodo restituisce ″true″ se il cursore viene posizionato sulla prima riga e ″false″ se il ResultSet è vuoto.
Last	Il metodo sposta il cursore sull'ultima riga nel ResultSet. Il metodo restituisce ″true″ se il cursore viene posizionato sull'ultima riga e ″false″ se il ResultSet è vuoto.
BeforeFirst	Il metodo sposta il cursore immediatamente prima della prima riga nel ResultSet. Per un ResultSet vuoto, questo metodo non ha alcun effetto. Non viene fornito alcun valore di ritorno da questo metodo.
AfterLast	Il metodo sposta il cursore immediatamente dopo l'ultima riga nel ResultSet. Per un ResultSet vuoto, questo metodo non ha alcun effetto. Non viene fornito alcun valore di ritorno da questo metodo.
Relative (int rows)	Il metodo sposta il cursore in relazione alla sua posizione corrente. • Se il valore delle righe è pari a 0, questo metodo non ha effetto. • Se il valore delle righe è positivo, il cursore viene spostato in avanti tante volte quante sono le righe indicate. Se esistono meno righe tra la posizione corrente e la fine del ResultSet rispetto a quanto specificato dai parametri di immissione, questo metodo opera come afterLast. • Se il valore delle righe è negativo, il cursore viene spostato indietro tante volte quante sono le righe indicate. Se esistono meno righe tra la posizione corrente e la fine del ResultSet rispetto a quanto specificato dal parametro di immissione, questo metodo opera come beforeFirst. Il metodo restituisce ″true″ se il cursore è posizionato su una riga valida, altrimenti restituisce ″false″.
Absolute (int row)	Il metodo sposta il cursore sulla riga specificata dal valore della riga. Se il valore della riga è positivo, il cursore viene posizionato sulla riga corrispondente al numero di righe indicate dal valore a partire dell'inizio del ResultSet. La prima riga è indicata con il numero 1, la seconda con 2 e così via. Se esistono meno righe nel ResultSet di quanto specificato dal valore della riga, questo metodo opera come afterLast. Se il valore è negativo, il cursore viene posizionato sulla riga corrispondente al numero di righe indicate dal valore a partire dalla fine del ResultSet. L'ultima riga è indicata con il numero -1, la seconda con -2 e così via. Se esistono meno righe nel ResultSet di quanto specificato dal valore della riga, questo metodo opera come beforeLast. Se il valore della riga è 0, questo metodo opera come beforeFirst. Il metodo restituisce ″true″ se il cursore viene posizionato su una riga valida, altrimenti restituisce ″false″.

Richiamo di dati ResultSet:

L'oggetto ResultSet fornisce numerosi metodi per ottenere dati colonna per una riga. Tutti i metodi sono del formato get<*Type*>, dove <*Type*> è un tipo di dati Java. Alcuni esempi di questi metodi includono getInt, getLong, getString, getTimestamp e getBlob. Quasi tutti questi metodi utilizzano un parametro singolo che risulta essere l'indice di colonne all'interno del ResultSet o il nome di colonna.

Le colonne del ResultSet sono numerate, a partire da 1. Se il nome della colonna è utilizzato è non si trova più di una colonna nel ResultSet con lo stesso nome, viene restituito il primo. Esistono alcuni metodi get<Type> che possiedono parametri ulteriori, come l'oggetto Calendar facoltativo, che può essere inviato a getTime, getDate e getTimestamp. Fare riferimento al Javadoc per il pacchetto java.sql per conoscere tutti i dettagli.

Per ottenere i metodi che restituiscono gli oggetti, il valore di ritorno è null quando la colonna nel ResultSet è null. Per tipi primitivi, il valore null non può essere restituito. In questi casi, il valore è 0 o ″false″. Se è necessario che un'applicazione effettui una distinzione tra null e 0 o ″false″, è possibile utilizzare il metodo wasNull immediatamente dopo la chiamata. È possibile successivamente determinare che questo metodo stabilisca se il valore era uno 0 o un valore ″false″ oppure se quel valore è stato restituito perché il valore del ResultSet era in realtà null.

Supporto ResultSetMetaData

Quando viene richiamato il metodo getMetaData su un oggetto del ResultSet, il metodo restituisce un oggetto ResultSetMetaData che descrive le colonne di quell'oggetto del ResultSet. Quando l'istruzione SQL elaborata non è nota fino al tempo di esecuzione, è possibile utilizzare ResultSetMetaData per determinare quali metodi get è necessario utilizzare per richiamare i dati. Il codice seguente utilizza ResultSetMetaData per determinare ogni tipo di colonna nella serie di risultati:

```
ResultSet rs = stmt.executeQuery(sqlString);
ResultSetMetaData rsmd = rs.getMetaData();
int colType [] = new int[rsmd.getColumnCount()];
for (int idx = 0, int col = 1; idx < colType.length; idx++, col++)
colType[idx] = rsmd.getColumnType(col);
```

Esempio: interfaccia ResultSetMetaData per IBM Developer Kit per Java:

Questo programma dimostra l'utilizzo di un ResultSetMetaData e di un ResultSet per visualizzare tutti i metadati su un ResultSet creato interrogando una tabella. L'utente passa il valore per la tabella e la libreria.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

/**
ResultSetMetaDataExample.java

Questo programma mostra l'uso di un ResultSetMetaData e di
un ResultSet per visualizzare tutti i metadati relativi ad un ResultSet
creato interrogando una tabella. L'utente passa il valore per la
tabella e la libreria.
**/
public class ResultSetMetaDataExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage:  java ResultSetMetaDataExample <library> <table>");
            System.out.println("where <library> is the library that contains <table>");
            System.exit(0);
```

```
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Ottenere un collegamento al database e preparare l'istruzione.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
            int rowCount = 0;
        for (int i = 1; i <= colCount; i++) {
                System.out.println("Information about column " + i);
                System.out.println("   Name.........: " + rsmd.getColumnName(i));
                System.out.println("   Data Type.....: " + rsmd.getColumnType(i) +
                                   " ( " + rsmd.getColumnTypeName(i) + " )");
                System.out.println("   Precision.....: " + rsmd.getPrecision(i));
                System.out.println("   Scale.........: " + rsmd.getScale(i));
                System.out.print  ("   Allows Nulls..: ");
                if (rsmd.isNullable(i)==0)
                    System.out.println("false");
                else
                    System.out.println("true");
            }

        } catch (Exception e) {
            // Gestire gli errori.
            System.out.println("Oops... we have an error... ");
            e.printStackTrace();
        } finally {
            // Garantire una ripulitura continua. Se il collegamento viene chiuso,
            // anche l'istruzione presente in esso verrà terminata.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Critical error - cannot close connection object");
                }
            }
        }
    }
}
```

Modifica dei ResultSet:

Con i programmi di controllo JDBC System i, è possibile modificare i ResultSet eseguendo varie attività.

L'impostazione predefinita relativa ai ResultSet è di sola lettura. Tuttavia, con Java Database Connectivity (JDBC) 2.0, i programmi di controllo JDBC System i forniscono un supporto completo per i ResultSet aggiornabili.

È possibile fare riferimento a "Caratteristiche del ResultSet" a pagina 114 per informazioni su come aggiornare i ResultSet.

Aggiornare le righe

È possibile aggiornare le righe in una tabella di database tramite l'interfaccia ResultSet. I passaggi implicati in questo processo sono i seguenti:

1. Modificare i valori per una specifica riga utilizzando i vari metodi update<*Type*> dove <*Type*> rappresenta un tipo di dati Java. Questi metodi update<*Type*> corrispondono ai metodi get<*Type*> disponibili per il richiamo dei valori.
2. Applicare le righe al database sottostante.

Il database stesso non è aggiornato finché non viene raggiunta la seconda fase. L'aggiornamento delle colonne in un ResultSet senza il richiamo del metodo updateRow non comporta alcuna modifica al database.

È possibile eliminare gli aggiornamenti pianificati su una riga con il metodo cancelUpdates. Una volta che viene richiamato il metodo updateRow, le modifiche al database risultano finali e non possono essere annullate.

Nota: Il metodo rowUpdated restituisce sempre ″false″ poiché il database non possiede la capacità di indicare quali righe sono state aggiornate. In modo corrispondente, anche il metodo updatesAreDetected restituisce ″false″.

Cancellare le righe

È possibile cancellare le righe in una tabella di database tramite l'interfaccia ResultSet. Il metodo deleteRow viene fornito e cancella la riga corrente.

Inserire le righe

È possibile inserire le righe in una tabella di database tramite l'interfaccia ResultSet. Questo processo utilizza una ″riga di inserimento″ le cui applicazioni nello specifico spostano il cursore e creano i valori che si desidera inserire nel database. Le fasi implicate in questo processo sono le seguenti:

1. Posizionare il cursore sulla riga di inserimento.
2. Impostare ogni valore relativo alle colonne nella nuova riga.
3. Inserire la riga nel database e spostare facoltativamente il cursore nuovamente alla riga corrente all'interno del ResultSet.

Nota: le nuove righe non vengono inserite nella tabella in cui viene posizionato il cursore. Queste vengono solitamente aggiunte alla fine dello spazio dei dati di tabella. Un database relazionale non dipende dalla posizione per impostazione predefinita. Ad esempio, non bisogna supporre di poter spostare il cursore sulla terza riga e di inserire qualcosa da visualizzare prima della quarta riga quando utenti successivi selezionano i dati.

supporto per aggiornamenti posizionati

Oltre al metodo per aggiornare il database tramite un ResultSet, è possibile utilizzare le istruzioni SQL per immettere aggiornamenti della posizione. Questo supporto si affida all'utilizzo di cursori denominati. JDBC fornisce il metodo setCursorName da Statement e il metodo getCursorName da ResultSet per assicurare un accesso a questi valori.

Due metodi DatabaseMetaData, supportsPositionedUpdated e supportsPositionedDelete, restituiscono ″true″ poiché questa caratteristica viene supportata con il programma di controllo JDBC nativo.

Esempio: eliminazione di valori da una tabella tramite il cursore di un'altra istruzione:

Quest'esempio Java mostra come eliminare i valori da una tabella tramite il cursore di un'altra istruzione.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class UsingPositionedDelete {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {
        UsingPositionedDelete test = new UsingPositionedDelete();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }



/**
Gestire quanto necessario per il lavoro di impostazione necessario.
**/
    public void setup() {
        try {
            // Registrare il programma di controllo JDBC.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
            } catch (SQLException e) {
                // Ignorare i problemi.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
In questa sezione, deve essere aggiunto il codice necessario
per eseguire la verifica. se è necessaria solo un collegamento al database,
può essere utilizzata la variabile globale 'connection'.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Aggiornare ogni valore utilizzando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");
```

```
            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement
                                      ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
                                      rs.getCursorName ());


            // Eseguire il loop del ResultSet e aggiornare ogni voce.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Ripulire le risorse dopo averle utilizzate.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }



/**
In questa sezione, inserire tutti i lavori di ripulitura per la verifica.
**/
    public void cleanup() {
        try {
            // Chiudere il collegamento globale aperto in setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }



/**
Visualizzare il contenuto della tabella.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Esempio: modifica di valori con un'istruzione tramite il cursore di un'altra istruzione:

Quest'esempio Java mostra come modificare i valori con un'istruzione tramite il cursore di un'altra istruzione.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class UsingPositionedUpdate {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {

        UsingPositionedUpdate test = new UsingPositionedUpdate();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }



/**
Gestire quanto necessario per il lavoro di impostazione necessario.
**/
    public void setup() {
        try {
            // Registrare il programma di controllo JDBC.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
            } catch (SQLException e) {
                // Ignorare i problemi.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
In questa sezione, deve essere aggiunto il codice necessario
per la verifica. Se è necessario solo un collegamento al database,
può essere utilizzata la variabile globale 'connection'.
**/
    public void run() {
```

```
        try {
            Statement stmt1 = connection.createStatement();

            // Aggiornare ogni valore utilizzando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
                                                                   + " CUJOSQL.WHERECUREX
                                                                       SET COL_VALUE = 'CHANGED'
                                                                       WHERE CURRENT OF "
                                                                   + rs.getCursorName ());


            // Eseguire il loop del ResultSet e aggiornare ogni voce.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Ripulire le risorse dopo averle utilizzate.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }



/**
In questa sezione, inserire tutti i lavori di ripulitura per la verifica.
**/
    public void cleanup() {
        try {
            // Chiudere il collegamento globale aperto in setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }



/**
Visualizzare il contenuto della tabella.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
```

```
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Creazione di ResultSet:

Per creare un oggetto ResultSet, è possibile utilizzare i metodi executeQuery o altri metodi. Quest'argomento descrive le opzioni per creare i ResultSet.

Questi metodi provengono dalle interfacce Statement, PreparedStatement o CallableStatement. Esistono, tuttavia, altri metodi disponibili. Ad esempio, i metodi DatabaseMetaData come getColumns, getTables, getUDTs, getPrimaryKeys e così via, restituiscono i ResultSet. È anche possibile avere una singola istruzione SQL che restituisce più ResultSet per l'elaborazione. È possibile inoltre utilizzare il metodo getResultSet per richiamare un oggetto ResultSet dopo la chiamata al metodo execute fornito dalle interfacce Statement, PreparedStatement o CallableStatement.

Consultare "Esempio: creazione di una procedura con più ResultSet" a pagina 110 per ulteriori informazioni.

Chiusura di ResultSets

Sebbene un oggetto ResultSet venga automaticamente chiuso quando si chiude l'oggetto Statement al quale è associato, è consigliabile chiudere gli oggetti ResultSet quando è terminato l'utilizzo di questi ultimi. In questo modo, vengono immediatamente liberate le risorse del database interne che possono aumentare il rendimento dell'applicazione.

È anche importante chiudere i ResultSet creati dalle chiamate DatabaseMetaData. Poiché non si possiede un accesso diretto all'oggetto Statement utilizzato per creare questi ResultSet, non è possibile richiamare la chiusura direttamente sull'oggetto Statement. Questi oggetti sono collegati in modo tale che il programma di controllo JDBC chiuda l'oggetto interno Statement quando si chiude l'oggetto esterno ResultSet. Sebbene questi oggetti non vengono chiusi manualmente, il sistema continua a lavorare; tuttavia utilizza più risorse di quanto non sia necessario.

Nota: è possibile, inoltre, che la caratteristica di conservabilità dei ResultSet chiuda automaticamente i ResultSet per conto dell'utente. È consentita la chiamata della chiusura più volte su un ResultSet.

"Oggetti Statement" a pagina 98
Si utilizza un oggetto Statement per l'elaborazione di un'istruzione SQL statica e l'ottenimento dei risultati prodotti da questa. È possibile aprire solo un ResultSet alla volta per ciascun oggetto Statement. Tutti i metodi dell'istruzione che elaborano un'istruzione SQL chiudono implicitamente un ResultSet corrente dell'istruzione se ne esiste uno aperto.

"PreparedStatement" a pagina 100
Le PreparedStatement estendono l'interfaccia Statement e forniscono un supporto per l'aggiunta di parametri alle istruzioni SQL.

"CallableStatement" a pagina 106
L'interfaccia CallableStatement JDBC estende PreparedStatement e fornisce il supporto per i parametri di immissione/emissione e di emissione. L'interfaccia CallableStatement dispone inoltre del supporto per i parametri di immissione fornito dall'interfaccia PreparedStatement.

"Interfaccia DatabaseMetaData per IBM Developer Kit per Java" a pagina 65
L'interfaccia DatabaseMetaData è implementata dal programma di controllo JDBC IBM Developer Kit per Java per fornire informazioni sulle sue origini di dati sottostanti. Essa viene utilizzata principalmente dagli strumenti e server dell'applicazione per stabilire come interagire con una

specifica origine dati. Le applicazioni possono inoltre utilizzare i metodi DatabaseMetaData per ottenere informazioni su una origine dati, ma ciò avviene meno frequentemente.

Esempio: interfaccia ResultSet per IBM Developer Kit per Java:

Questo è un esempio su come utilizzare l'interfaccia ResultSet.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

/**
ResultSetExample.java

Questo programma mostra l'uso di un ResultSetMetaData e di
una ResultSet per visualizzare tutti i dati nella tabella sebbene
il programma che richiama i dati non sa quali elementi verranno
visualizzati nella tabella (l'utente inoltra i valori per la
tabella e la libreria).
**/
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage:  java ResultSetExample <library> <table>");
            System.out.println(" where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Ottenere un collegamento al database e preparare l'istruzione.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
            int rowCount = 0;
            while (rs.next()) {
                rowCount++;
                System.out.println("Data for row " + rowCount);
                for (int i = 1; i <= colCount; i++)
                    System.out.println("   Row " + i + ": " + rs.getString(i));
            }

        } catch (Exception e) {
            // Gestire gli errori.
            System.out.println("Oops... we have an error... ");
            e.printStackTrace();
        } finally {
            // Garantire una ripulitura continua. Se il collegamento viene chiuso,
            // anche l'istruzione presente in esso verrà terminata.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
```

```
                    System.out.println("Critical error - cannot close connection object");
                }
            }
        }
    }
}
```

Lotto di oggetti JDBC

Il lotto di oggetti è una considerazione importante per JDBC (Java Database Connectivity) e per le prestazioni. Dal momento che molti oggetti utilizzati in JDBC, come Connection, Statement e ResultSet, sono dispendiosi da creare, è possibile ottenere benefici significativi sulle prestazioni riutilizzando questi oggetti invece di crearli ogni volta che sono necessari.

Molte applicazioni gestiscono già una creazione di lotti di oggetti per conto dell'utente. Ad esempio, WebSphere ha un supporto estensivo per la creazione di lotti di oggetti JDBC e consente all'utente di controllare in che modo viene gestito il lotto. Per questo motivo, è possibile ottenere la funzionalità che si desidera senza preoccuparsi dei meccanismi di creazione dei lotti. Tuttavia, quando non viene fornito il supporto, è necessario trovare una soluzione per tutte le applicazioni ad eccezione delle più banali.

Utilizzo del supporto DataSource per la creazione di lotti di oggetti:

È possibile utilizzare i DataSource per ottenere che più applicazioni condividano una configurazione comune per accedere ad un database. Ciò si realizza perché ogni applicazione fa riferimento allo stesso nome DataSource.

Utilizzando DataSource, è possibile modificare molte applicazioni da un'ubicazione centrale. Ad esempio, se si modifica il nome di una libreria predefinita utilizzata da tutte le applicazioni e se è stato utilizzato un singolo DataSource per ottenere collegamenti per tutte le applicazioni, è possibile aggiornare il nome della raccolta in quel DataSource. Tutte le applicazioni successivamente cominciano a utilizzare la nuova libreria predefinita.

Quando si utilizza DataSource per ottenere i collegamenti relativi a un'applicazione, è possibile utilizzare il supporto incorporato del programma di controllo JDBC nativa relativo alla creazione di lotti di collegamenti. Questo supporto è fornito come un'implementazione dell'interfaccia ConnectionPoolDataSource.

La creazione di lotti si realizza fornendo liberamente oggetti Connection ″logici″ invece di oggetti Connection fisici. Un **oggetto Connection logico** rappresenta un collegamento che viene restituito da un oggetto Connection inserito in un lotto. Ogni collegamento logico agisce come un handle temporaneo al collegamento fisico rappresentato da un oggetto di collegamento inserito nel lotto. Per l'applicazione, quando viene restituito l'oggetto Connection, non esiste alcuna notevole differenza tra i due oggetti. La differenza sottile si verifica quando si chiama il metodo close sull'oggetto Connection. Questa chiamata annulla il collegamento logico e restituisce il collegamento fisico al lotto nel quale un'altra applicazione è in grado di utilizzare il collegamento fisico. Questa tecnica consente a molti oggetti dei collegamenti logici di riutilizzare un collegamento fisico singolo.

Impostazione della creazione di lotti di collegamenti

La creazione di lotti di collegamenti si realizza creando un oggetto DataSource che fa riferimento a un oggetto ConnectionPoolDataSource. Gli oggetti ConnectionPoolDataSource hanno proprietà che non è possibile impostare per la gestione di vari aspetti della manutenzione del lotto.

Fare riferimento all'esempio su come impostare la creazione di lotti di collegamenti con UDBDataSource e UDBConnectionPoolDataSource per ulteriori dettagli. È anche possibile consultare la JNDI (Java Naming and Directory Interface) per ulteriori dettagli sul ruolo che la JNDI assume in quest'esempio.

Da questo esempio, il collegamento che unisce i due oggetti DataSource è il dataSourceName. Il collegamento informa l'oggetto DataSource di ritardare la creazione di collegamenti per l'oggetto ConnectionPoolDataSource che gestisce il lotto automaticamente.

Applicazioni della creazione di lotti e della mancata creazione di lotti

Non esiste alcuna differenza tra un'applicazione che utilizza la creazione di lotti di collegamenti e una che non lo utilizza. Quindi è possibile aggiungere il supporto della creazione di lotti dopo che il codice di applicazione è completo, senza apportare alcuna modifica al codice di applicazione.

L'esempio seguente viene emesso dall'esecuzione del programma precedente in modo locale durante lo sviluppo.

Start timing the non-pooling DataSource version... Time spent: 6410

Start timing the pooling version... Time spent: 282

programma Java completato.

Un UDBConnectionPoolDataSource inserisce in un lotto un collegamento singolo in modo predefinito. Se un'applicazione necessita più volte di un collegamento e necessita solo di un collegamento alla volta, l'utilizzo di UDBConnectionPoolDataSource è una soluzione perfetta. Se sono necessari molti collegamenti simultanei, è necessario configurare ConnectionPoolDataSource"Proprietà di ConnectionPoolDataSource" a pagina 134 per far corrispondere esigenze e risorse.

Concetti correlati

"JNDI (Java Naming and Directory Interface)" a pagina 578
La JNDI (Java Naming and Directory Interface) fa parte dell'API (Application program interface) della piattaforma JavaSoft. Con JNDI, è possibile collegarsi, senza connessioni fisiche, a più servizi dell'indirizzario e di denominazione. È possibile creare applicazioni Java abilitate all'indirizzario trasferibili e potenti utilizzando questa interfaccia.

Riferimenti correlati

"Esempio: impostazione del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource"
Questo è un esempio della modalità di utilizzo del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource.

"Proprietà di ConnectionPoolDataSource" a pagina 134
È possibile configurare l'interfaccia ConnectionPoolDataSource utilizzando la serie di proprietà che essa fornisce.

Esempio: impostazione del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource:

Questo è un esempio della modalità di utilizzo del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class ConnectionPoolingSetup
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
```

```
        // Creare un'implementazione ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object");

        // Stabilire un contesto JNDI e collegare l'origine dati del lotto di collegamenti
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Creare una origine dati standard che vi faccia riferimento.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting pooling");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
  }
}
```

Esempio: esecuzione della verifica sulle prestazioni del lotto di collegamenti:

Questo è un esempio di come sottoporre a verifica le prestazioni dell'esempio della creazione di lotti rispetto a quelle dell'esempio in cui non vengono creati lotti.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;

public class ConnectionPoolingTest
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        Context ctx = new InitialContext();
        // Eseguire il lavoro senza un lotto:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("\nStart timing the non-pooling DataSource version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));

        // Eseguire il lavoro con il lotto:
        ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
}
```

Proprietà di ConnectionPoolDataSource:

È possibile configurare l'interfaccia ConnectionPoolDataSource utilizzando la serie di proprietà che essa fornisce.

Le descrizioni di queste proprietà vengono fornite nella seguente tabella.

Proprietà	Descrizione
initialPoolSize	Quando un lotto viene assegnato per la prima volta ad un'istanza, questa proprietà determina il numero di collegamenti inseriti nel lotto. Se questo valore viene specificato al di fuori dell'intervallo compreso tra minPoolSize e maxPoolSize, o minPoolSize o maxPoolSize viene utilizzato come il numero di collegamenti iniziale da creare.
maxPoolSize	Quando il lotto viene utilizzato, possono essere richiesti più collegamenti di quanti ne siano presenti all'interno del lotto. Questa proprietà specifica il numero massimo di collegamenti la cui creazione nel lotto è consentita. Le applicazioni non "effettuano il blocco" e attendono che sia restituito un collegamento al lotto quando il lotto raggiunge la dimensione massima e tutti i collegamenti sono in uso. Al contrario, il programma di controllo JDBC crea un nuovo collegamento basato sulle proprietà DataSource e restituisce il collegamento. Se viene specificata una maxPoolSize pari a 0, viene consentito al lotto di aumentare le proprie dimensioni senza vincoli fin tanto che il sistema possiede risorse disponibili per la distribuzione.
minPoolSize	I picchi dell'utilizzo del lotto possono determinare un aumento nel numero di collegamenti in esso. Se il livello di attività diminuisce fino al punto in cui alcune Connection non vengono mai eliminate dal lotto, le risorse vengono raccolte senza una particolare ragione. In tali casi, il programma di controllo JDBC possiede la capacità di rilasciare alcuni dei collegamenti che ha accumulato. Questa proprietà consente di indicare al JDBC di rilasciare i collegamenti assicurandosi che esista sempre un certo numero di collegamenti disponibili per l'utilizzo. Se si specifica una minPoolSize pari a 0, è possibile che il lotto liberi tutti i collegamenti e che l'applicazione provveda realmente al tempo di collegamento per ogni richiesta di collegamento.
maxIdleTime	I collegamenti tengono traccia del tempo in cui sono rimasti in sospeso senza essere utilizzati. Questa proprietà specifica il tempo che un'applicazione fornisce ai collegamenti non utilizzati prima che questi vengano rilasciati (cioè, esistono più collegamenti rispetto al necessario). Questa proprietà rappresenta un periodo di tempo espresso in secondi e non specifica quando si verifica la chiusura effettiva. Questo specifica quanto tempo debba trascorrere prima che un collegamento venga rilasciato.
propertyCycle	Questa proprietà rappresenta il numero di secondi consentiti che devono trascorrere prima dell'acquisizione forzata di queste regole.

Nota: l'impostazione del tempo di maxIdleTime o propertyCycle su 0 significa che il programma di controllo JDBC non controlla l'eliminazione dei collegamenti dal lotto. Le regole specificate per la dimensione initial, min e max sono ancora forzate.

Quando maxIdleTime e propertyCycle non hanno un valore pari a 0, si utilizza un sottoprocesso di gestione per controllare il lotto. Il sottoprocesso si attiva a ogni secondo del propertyCycle e controlla tutti i collegamenti presenti nel lotto per individuare quali tra essi non vengono utilizzati per più secondi del maxIdleTime. I collegamenti che rispondono a tali criteri vengono eliminati dal lotto finché non viene raggiunto il valore di minPoolSize.

Creazione di lotti di istruzioni basati su DataSource:

La proprietà maxStatements, disponibile sull'interfaccia UDBConnectionPoolDataSource, consente la creazione di lotti di istruzioni all'interno del lotto di collegamenti. La creazione di lotti di istruzioni ha effetto solo sulle PreparedStatement e CallableStatement. Gli oggetti Statement non vengono inseriti nel lotto.

L'implementazione della creazione di lotti di istruzioni è simile a quella della creazione di lotti di collegamenti. Quando l'applicazione chiama Connection.prepareStatement("select * from tablex"), il modulo della creazione di lotti controlla se l'oggetto Statement è già stato preparato all'interno del collegamento. Se è stato preparato, un oggetto logico PreparedStatement viene inoltrato all'utente invece dell'oggetto fisico. Quando si chiama la chiusura, l'oggetto Connection viene restituito al lotto, l'oggetto logico Connection viene eliminato e l'oggetto Statement può essere riutilizzato.

La proprietà maxStatements consente al DataSource di specificare quante istruzioni possono essere inserite in un lotto all'interno del collegamento. Un valore pari a 0 indica che non è possibile utilizzare la creazione di lotti di istruzioni. Quando il lotto di istruzioni è completo, almeno gli algoritmi utilizzati recentemente vengono applicati per determinare quale istruzione deve essere eliminata.

Il seguente esempio esegue la verifica di un DataSource che utilizza solo lotti di collegamenti e dell'altro DataSource che utilizza sia lotti di collegamenti che lotti di istruzioni.

L'esempio seguente è emesso dall'esecuzione di questo programma localmente durante lo sviluppo.

Deploying statement pooling data source Start timing the connection pooling only version... Time spent: 26312

Starting timing the statement pooling version... Time spent: 2292 Java program completed

Esempio: verifica delle prestazioni di due DataSource:

Questo è un esempio di verifica di un DataSource che utilizza solo lotti di collegamenti e di un altro DataSource che utilizza sia lotti di collegamenti che lotti di istruzioni.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class StatementPoolingTest
{

    public static void main(java.lang.String[] args)
    throws Exception
```

```
    {
        Context ctx = new InitialContext();

        System.out.println("deploying statement pooling data source");
        deployStatementPoolDataSource();

        // Gestire solo la creazione lotto di collegamenti.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the connection pooling only version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));


        // Gestire la creazione lotti di istruzioni aggiunti.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("\nStart timing the statement pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }



    private static void deployStatementPoolDataSource()
    throws Exception
    {
        // Creare un'implementazione ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object with Statement pooling");
        cpds.setMaxStatements(10);

        // Stabilire un contesto JNDI e collegare l'origine dati del lotto di collegamenti
        Context ctx = new InitialContext();
        ctx.rebind("StatementSupport", cpds);


        // Creare un datasource standard che vi faccia riferimento.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting statement pooling");
        ds.setDataSourceName("StatementSupport");
        ctx.rebind("StatementPoolingDataSource", ds);

    }
}
```

Creazione del lotto di collegamenti:

È possibile sviluppare il proprio collegamento e la propria creazione di lotti di istruzioni senza richiedere il supporto per i DataSource o fare affidamento su un altro prodotto.

Senza la creazione di lotti di collegamenti, il lavoro del database è eccessivo per ogni richiesta. In pratica, si apre un collegamento, si apre un'istruzione, si elabora l'istruzione, si chiude l'istruzione e si chiude il collegamento. Invece di eliminare tutto dopo ogni richiesta, esiste un modo per poter riutilizzare le parti di questo processo. La creazione di **lotti di collegamenti** sostituisce il codice di creazione collegamento con altro codice per ottenere un collegamento dal lotto e successivamente sostituisce il codice di chiusura collegamento con altro codice per riportare il collegamento al lotto per il relativo utilizzo.

Il programma di creazione di lotti di collegamenti crea i collegamenti e li colloca all'interno del lotto. La classe di lotti ha metodi take e put per individuare un collegamento da utilizzare e per restituire il collegamento al lotto al termine della gestione del collegamento. Questi metodi vengono sincronizzati perché l'oggetto del lotto risulta essere una risorsa condivisa, ma si dovrebbe evitare che più sottoprocessi tentino di gestire simultaneamente le risorse inserite nel lotto.

Creazione del proprio lotto di istruzioni

Quando si utilizza la creazione di lotti di collegamenti, si perde del tempo durante la creazione e la chiusura di un'istruzione quando essa viene elaborata. Questo rappresenta un esempio di perdita di un oggetto che può essere riutilizzato.

Per riutilizzare un oggetto, è possibile utilizzare la classe di istruzioni preparata. Nella maggior parte delle applicazioni, vengono riutilizzate le stesse istruzioni SQL con un numero di modifiche minore. Ad esempio, una iterazione tramite un'applicazione potrebbe generare la seguente query:

```
SELECT * from employee where salary > 100000
```

L'iterazione seguente potrebbe generare la seguente query:

```
SELECT * from employee where salary > 50000
```

Questa è la stessa query, ma utilizza un parametro differente. È possibile che entrambe le query siano realizzate con la seguente query.

```
SELECT * from employee where salary > ?
```

È possibile successivamente impostare il contrassegno di parametro (denotato dal punto interrogativo) su 100000 durante l'elaborazione della prima query e su 50000 durante l'elaborazione della seconda query. Ciò migliora le prestazioni per tre ragioni oltre a quella che il lotto di collegamenti può offrire:

- Vengono creati meno oggetti. Un oggetto PreparedStatement viene creato e riutilizzato invece della creazione di un oggetto Statement per ogni richiesta. Inoltre vengono eseguiti meno programmi di creazione.
- È possibile riutilizzare il lavoro del database per impostare l'istruzione SQL (chiamata **prepare**). La preparazione di istruzioni SQL è chiaramente dispendiosa perché implica la determinazione del contenuto del testo dell'istruzione SQL e del modo in cui il sistema debba realizzare l'attività richiesta.
- Quando le creazioni di oggetti aggiuntivi vengono eliminate, si verifica un vantaggio che non viene considerato spesso. Non c'è alcuna necessità di eliminare ciò che non è stato creato. Questo modello risulta di più semplice utilizzo nel programma di raccolta dati inutili Java ed inoltre incrementa nel tempo le prestazioni rispetto a molti utenti.

Considerazioni

Le prestazioni migliorano tramite la duplicazione. Se una voce non viene riutilizzata, allora inserirla nel lotto significa perdere risorse.

La maggior parte delle applicazioni contiene sezioni critiche del codice. In genere, un'applicazione utilizza dall'80 al 90 percento del tempo di elaborazione su solo il 10, 20 percento del codice. Se ci sono 10,000 istruzioni SQL utilizzate potenzialmente in un'applicazione, non tutte vengono inserite nel lotto. L'obiettivo è quello d'identificare e inserire in un lotto le istruzioni SQL utilizzate nelle sezioni critiche dell'applicazione del codice.

È possibile che la creazione di oggetti in un implementazione Java comporti dei costi rilevanti. La soluzione di un inserimento in un lotto può essere utilizzata con vantaggio. Gli oggetti utilizzati nel processo vengono creati all'inizio, prima che altri utenti tentino di utilizzare il sistema. Questi oggetti vengono riutilizzati ogni volta che è necessario. Le prestazioni risultano eccellenti ed è possibile ottimizzare nel tempo l'applicazione per facilitare l'utilizzo per il maggiore numero di utenti. Ne consegue che più oggetti vengono inseriti nel lotto. Inoltre consente una più efficace esecuzione di più sottoprocessi dell'accesso al database dell'applicazione in modo da ottenere un rendimento maggiore.

Java (utilizzando JDBC) si basa sull'SQL dinamico e può risultare lento. È possibile che la creazione di lotti possa ridurre questo problema. Preparando le istruzioni all'avvio, è possibile rendere statico l'accesso al database. Esiste poca differenza nelle prestazioni tra l'SQL statico e dinamico dopo che l'istruzione è stata preparata.

Le prestazioni dell'accesso al database in Java possono essere efficaci e regolate senza rinunciare alla conservabilità del codice o della progettazione orientata agli oggetti. Non è difficile scrivere il codice per la creazione di lotti di istruzioni e collegamenti. Inoltre, è possibile modificare il codice in modo tale che supporti più applicazioni e tipi di applicazioni (basate sul web, client/server) e così via.

Aggiornamenti batch

Il supporto di aggiornamento batch consente di inoltrare qualsiasi aggiornamento al database come una singola transazione tra il programma utente e il database. Questa procedura può migliorare considerevolmente le prestazioni quando è necessario eseguire più aggiornamenti alla volta.

Ad esempio, se una grande società richiede ai propri impiegati appena assunti di iniziare il lavoro di lunedì, questo requisito rende necessario elaborare molti aggiornamenti (in questo caso inserzioni) al database degli impiegati alla volta. Creare un batch di aggiornamenti e inoltrarli al database come un'unità può far risparmiare del tempo.

Esistono due tipi di aggiornamenti batch:

- Aggiornamenti batch che utilizzano oggetti Statement.
- Aggiornamenti batch che utilizzano oggetti PreparedStatement.

Aggiornamento batch Statement:

Per eseguire un aggiornamento batch Statement, è necessario disattivare il commit automatico. In JDBC ovvero Java Database Connectivity, il commit automatico è attivo per impostazione predefinita. Il commit automatico indica che ogni aggiornamento al database viene sincronizzato dopo l'elaborazione di ogni istruzione SQL. Se si intende gestire un gruppo di istruzioni passato al database come un gruppo funzionale, non si desidera che il database sincronizzi ogni istruzione singolarmente. Se non si disattiva il commit automatico e un'istruzione al centro del batch ha esito negativo, non è possibile eseguire il rollback dell'intero batch e ritentare in quanto la metà delle istruzioni sono state rese finali. Inoltre, il lavoro aggiuntivo di sincronizzare ogni istruzione in un batch crea un notevole sovraccarico.

Per ulteriori dettagli, consultare "Transazioni JDBC" a pagina 78.

Dopo aver disattivato il commit automatico, è possibile creare un oggetto Statement standard. Invece di elaborare le istruzioni con metodi come executeUpdate, l'utente le aggiunge al batch con il metodo addBatch. Dopo aver aggiunto tutte le istruzioni che si desidera al batch, è possibile elaborarle tutte con il metodo executeBatch. È possibile svuotare il batch in qualsiasi momento con il metodo clearBatch.

Il seguente esempio mostra come è possibile utilizzare questi metodi:

Esempio: aggiornamento batch Statement

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
connection.setAutoCommit(false);
Statement statement = connection.createStatement();
statement.addBatch("INSERT INTO TABLEX VALUES(1, 'Cujo')");
statement.addBatch("INSERT INTO TABLEX VALUES(2, 'Fred')");
statement.addBatch("INSERT INTO TABLEX VALUES(3, 'Mark')");
int [] counts = statement.executeBatch();
connection.commit();
```

In questo esempio, viene restituita una schiera di numeri interi dal metodo executeBatch. Tale schiera ha un valore intero per ogni istruzione elaborata nel batch. Se si stanno inserendo i valori nel database, il valore per ogni istruzione è 1 (cioè, presume l'elaborazione con esito positivo). Tuttavia, alcune istruzioni possono essere istruzioni di aggiornamento che influenzano più righe. Se si inserisce una qualsiasi istruzione nel batch diversa da INSERT, UPDATE o DELETE, si verifica un'eccezione.

Aggiornamento batch preparedStatement:

Un batch preparedStatement è simile ad un batch Statement; tuttavia, un batch preparedStatement funziona sempre fuori dalla stessa istruzione preparata e si modificano solo i parametri in quell'istruzione.

Segue un esempio che utilizza un batch preparedStatement.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
connection.setAutoCommit(false);
PreparedStatement statement =
  connection.prepareStatement("INSERT INTO TABLEX VALUES(?, ?)");
statement.setInt(1, 1);
statement.setString(2, "Cujo");
statement.addBatch();
statement.setInt(1, 2);
statement.setString(2, "Fred");
statement.addBatch();
statement.setInt(1, 3);
statement.setString(2, "Mark");
statement.addBatch();
int [] counts = statement.executeBatch();
connection.commit();
```

BatchUpdateException JDBC:

Un'importante considerazione sugli aggiornamenti batch è quale operazione effettuare quando una chiamata ad un metodo executeBatch ha esito negativo. In questo caso, viene emesso un nuovo tipo di eccezione, denominato BatchUpdateException. BatchUpdateException è una sottoclasse della SQLException e consente di chiamare gli stessi metodi che sono sempre stati chiamati per ricevere il messaggio, SQLState e il codice fornitore.

BatchUpdateException fornisce inoltre il metodo getUpdateCounts che restituisce una schiera di numeri interi. Tale schiera contiene conteggi di aggiornamento da tutte le istruzioni nel batch che sono state elaborate fino al punto in cui si è verificato l'errore. La lunghezza della schiera indica quale istruzione nel batch ha avuto esito negativo. Ad esempio, se la schiera restituita nell'eccezione ha una lunghezza di tre, la quarta istruzione nel batch ha avuto esito negativo. Quindi, dal singolo oggetto BatchUpdateException che viene restituito, è possibile determinare i conteggi di aggiornamento per tutte le istruzioni con esito positivo, quale istruzione ha avuto esito positivo e tutte le informazioni sull'errore.

le prestazioni standard dell'elaborazione degli aggiornamenti sottoposti a batch sono equivalenti a quelle dell'elaborazione di ogni istruzione in modo indipendente. È possibile far riferimento al Supporto di inserimento bloccato per ulteriori informazioni sul supporto ottimizzato per gli aggiornamenti batch. Sarebbe opportuno utilizzare ancora il nuovo modello nella codificazione e sfruttare le ottimizzazioni delle prestazioni future.

Nota: nella specifica JDBC 2.1, viene fornita un'opzione differente sulla modalità di gestione delle condizioni di eccezione relative agli aggiornamenti batch. JDBC 2.1 presenta un modello dove il batch di elaborazione continua dopo che un'immissione batch ha avuto esito negativo. Un conteggio di aggiornamento speciale è inserito nella schiera dei numeri interi di conteggio dell'aggiornamento restituita per ogni immissione che ha esito negativo. Ciò consente a batch ampi di continuare l'elaborazione anche se una delle immissioni ha esito negativo. Consultare la specifica JDBC 2.1 o JDBC 3.0 per dettagli su queste due modalità di operazione. Per impostazione predefinita, il programma di controllo JDBC nativa utilizza la definizione JDBC 2.0. L'unità fornisce una proprietà Connection utilizzata quando si usa DriverManager per stabilire collegamenti. L'unità fornisce inoltre una proprietà DataSource utilizzata quando si usa DataSource per stabilire collegamenti. Tali proprietà consentono alle applicazioni di scegliere in che modo desiderano che le operazioni batch gestiscano gli errori.

Inserimenti bloccati con JDBC:

È possibile utilizzare un'operazione di inserimento bloccato per inserire svariate righe per volta in una tabella di database.

Un inserimento bloccato è un tipo speciale di operazione su System i che fornisce un metodo altamente ottimizzato per inserire svariate righe per volta in una tabella di database. È possibile concepire gli inserimenti bloccati come un sottoinsieme di aggiornamenti sottoposti a batch. Gli aggiornamenti sottoposti a batch possono essere una qualsiasi forma di richiesta di aggiornamento, ma gli inserimenti bloccati sono specifici. Tuttavia, i tipi di inserimenti bloccati degli aggiornamenti sottoposti a batch sono comuni; il programma di controllo JDBC nativo è stato modificato per sfruttare questa funzione.

A causa di limitazioni del sistema durante l'utilizzo del supporto di inserimento bloccato, l'impostazione predefinita per il programma di controllo JDBC nativo deve avere l'inserimento bloccato disabilitato. È possibile abilitarlo tramite la proprietà Connection o DataSource. È possibile gestire e controllare la maggior parte delle limitazioni per conto proprio, ma non tutte; per questo motivo è consigliabile disattivare il supporto di inserimento per impostazione predefinita. L'elenco di limitazioni è il seguente:

- L'istruzione SQL utilizzata deve essere un'istruzione INSERT con una clausola VALUES, ad indicare che non è un'istruzione INSERT con SUBSELECT. Il programma di controllo JDBC riconosce questa limitazione ed agisce di conseguenza.
- È necessario utilizzare PreparedStatement, ad indicare che non esiste un supporto ottimizzato per gli oggetti dell'istruzione. Il programma di controllo JDBC riconosce questa limitazione ed agisce di conseguenza.
- L'istruzione SQL deve specificare i contrassegni di parametro per tutte le colonne nella tabella. Ciò significa che non è possibile utilizzare i valori della costante per una colonna o consentire al database di inserire i valori predefiniti per qualsiasi colonna. Il programma di controllo JDBC non dispone di un meccanismo per gestire la verifica dei contrassegni di parametri specifici nella propria istruzione SQL. Se si imposta la proprietà affinché esegua inserimenti bloccati ottimizzati e non si evitano valori predefiniti o costanti nelle proprie istruzioni SQL, i valori che finiscono nella tabella database non sono corretti.
- È necessario il collegamento al sistema locale. Ciò significa che non è possibile utilizzare un collegamento utilizzando DRDA per accedere ad un sistema remoto in quanto DRDA non supporta un'operazione di inserimento bloccato. Il programma di controllo JDBC non dispone di un meccanismo per gestire la verifica relativa al collegamento ad un sistema locale. Se si imposta la proprietà affinché esegua un inserimento bloccato ottimizzato e si tenta di collegarsi ad un sistema remoto, l'elaborazione dell'aggiornamento batch ha esito negativo.

Questo esempio di codice mostra come abilitare il supporto per l'elaborazione dell'inserimento bloccato. L'unica differenza tra questo codice ed una versione che non utilizza il supporto di inserimento bloccato è `use block insert=true` che viene aggiunto all'URL di collegamento.

Esempio: elaborazione dell'inserimento bloccato

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle “Informazioni sull’esonero di responsabilità e licenza del codice” a pagina 580.

```
// Creare un collegamento database
Connection c = DriverManager.getConnection("jdbc:db2:*local;use block insert=true");
BigDecimal bd = new BigDecimal("123456");

// Creare PreparedStatement per inserimento in una tabella con 4 colonne
PreparedStatement ps =
  c.prepareStatement("insert into cujosql.xxx values(?, ?, ?, ?)");

// Avvio sincronizzazione...
for (int i = 1; i <= 10000; i++) {
  ps.setInt(1, i);                        // Impostare tutti i parametri di una riga
  ps.setBigDecimal(2, bd);
  ps.setBigDecimal(3, bd);
  ps.setBigDecimal(4, bd);
  ps.addBatch();                          //Aggiungere i parametri al batch
}

// Elaborare il batch
int[] counts = ps.executeBatch();

// Fine sincronizzazione...
```

In simili casi di verifica, l’elaborazione di operazioni con un inserimento bloccato è molto più veloce rispetto a quella in cui tale inserimento non viene utilizzato. Ad esempio, la verifica eseguita sul codice precedente è stata di nove volte più veloce utilizzando inserimenti bloccati. I casi che utilizzano solo tipi primitivi invece di oggetti possono essere sedici volte più veloci. In applicazioni in cui c’è una quantità di lavoro considerevole in esecuzione è opportuno rivedere le proprie aspettative.

Tipi di dati avanzati

I tipi di dati SQL3 avanzati forniscono all’utente estrema flessibilità. Essi sono ideali per archiviare oggetti Java serializzati, documenti XML (Extensible Markup Language) e dati multimediali come canzoni, immagini del prodotto, fotografie degli impiegati e filmati. JDBC Java Database Connectivity 2.0 e le versioni successive forniscono il supporto per gestire questo tipo di dati che sono parte dello standard SQL99.

Tipi distinti

Il tipo distinto è un tipo definito dall’utente che si basa su un database standard. Ad esempio, è possibile definire un tipo SSN (Social Security Number) che è internamente un CHAR(9). La seguente istruzione SQL crea questo tipo DISTINCT.

```
CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)
```

Un tipo distinto è sempre correlato ad un tipo di dati incorporati. Per ulteriori informazioni su come e quando utilizzare tipi distinti nel contesto di SQL, consultare i manuali di riferimento SQL.

Per utilizzare i tipi distinti in JDBC, l’utente vi accede nello stesso modo in cui accede ad un tipo sottostante. Il metodo getUDTs è un nuovo metodo che consente di interrogare quali tipi distinti sono disponibili sul sistema. Il programma Esempio: tipi distinti mostra quanto segue:

- La creazione di un tipo distinto.
- La creazione di una tabella che utilizza tale tipo.
- L’utilizzo di PreparedStatement per impostare un parametro del tipo distinto.
- L’utilizzo di ResultSet per restituire un tipo distinto.
- L’utilizzo della chiamata API (Application Programming Interface) metadata a getUDT per informazioni su un tipo distinto.

Per ulteriori informazioni, consultare l'argomento secondario Esempio: tipi distinti che mostra varie attività comuni che è possibile eseguire utilizzando i tipi distinti.

LOB (Large Object)

Esistono tre tipi di LOB (Large Object):
- Binary Large Object (BLOB)
- Character Large Object (CLOB)
- Double Byte Character Large Object (DBCLOB)

I DBCLOB sono simili ai CLOB ad eccezione della relativa rappresentazione di memoria interna dei dati di caratteri. Dato che Java e JDBC esternano tutti i dati di caratteri come Unicode, esiste solo un supporto JDBC per i CLOB. I DBCLOB funzionano in modo intercambiabile con il supporto CLOB da una prospettiva JDBC.

BLOB (Binary Large Object)

In molti modi, una colonna BLOB (Binary Large Object) è simile ad una colonna CHAR FOR BIT DATA che è possibile ingrandire. In queste colonne è possibile memorizzare qualsiasi dato che possa essere rappresentato come un flusso di byte non convertiti. Spesso, le colonne BLOB vengono utilizzate per memorizzare oggetti Java serializzati, immagini, canzoni e altri dati binari.

È possibile utilizzare i BLOB nello stesso modo degli altri tipi di database standard. È possibile passarli a procedure memorizzate, utilizzarli in istruzioni preparate e aggiornarli in serie di risultati. La classe PreparedStatement dispone di un metodo setBlob per inoltrare i BLOB al database e la classe ResultSet aggiunge una classe getBlob per richiamarli dal database. Un BLOB viene rappresentato in un programma Java da un oggetto BLOB che è un'interfaccia JDBC.

CLOB (Character Large Object)

I CLOB (Character Large Object) sono il complemento di dati di caratteri ai BLOB. Invece di memorizzare i dati nel database senza la conversione, i dati vengono memorizzati nel database come testo ed elaborati nello stesso modo di una colonna CHAR. Come avviene con i BLOB, JDBC 2.0 fornisce funzioni per trattare direttamente con i CLOB. L'interfaccia PreparedStatement contiene un metodo setClob e l'interfaccia ResultSet contiene un metodo getClob.

Sebbene le colonne BLOB e CLOB funzionino come le colonne CHAR FOR BIT DATA e CHAR, questo è concettualmente il modo in cui esse funzionano da una prospettiva di un utente esterno. Internamente, esse sono differenti; a causa della dimensione potenzialmente imponente delle colonne LOB (Large Object), generalmente si lavora in modo indiretto con i dati. Ad esempio, quando si acquisisce un blocco di righe dal database, non si sposta un blocco di LOB in ResultSet. Si spostano, invece, i puntatori denominati localizzatori LOB (cioè, numeri interi di quattro byte) nel ResultSet. Tuttavia, non è necessario conoscere i localizzatori quando si gestiscono i LOB in JDBC.

Datalink

I **Datalink** sono valori compressi che contengono un riferimento logico dal database ad un file memorizzato fuori dal database. I Datalink vengono rappresentati e utilizzati da una prospettiva JDBC in due modi differenti, a seconda se si sta utilizzando JDBC 2.0 o versione precedente o se si sta utilizzando JDBC 3.0 o versione successiva.

Tipi di dati SQL3 non supportati

Esistono altri tipi di dati SQL3 che sono stati definiti e per cui l'API JDBC fornisce un supporto. Questi sono ARRAY, REF e STRUCT. Attualmente, System i non supporta questi tipi. Quindi, il programma di

controllo JDBC non fornisce alcuna forma di supporto per essi.

Riferimenti correlati

"Esempio: tipi distinti" a pagina 153
Questo esempio illustra come utilizzare tipi distinti.

Scrittura del codice che utilizza i BLOB:

Esistono alcune attività che è possibile realizzare con colonne BLOB (Binary Large Object) database tramite l'API (Application Programming Interface) JDBC Java Database Connectivity. I seguenti argomenti discutono brevemente queste attività e includono esempi su come realizzarle.

Lettura dei BLOB dal database e inserimento dei BLOB nel database

Con l'API JDBC, esistono metodi per richiamare i BLOB fuori dal database e metodi per inserirli nel database. Tuttavia, non esiste una modalità standardizzata per creare un oggetto Blob. Questo non è un problema se il proprio database è già pieno di BLOB, ma costituisce un problema se si desidera gestire i BLOB da zero tramite JDBC. Invece di definire un programma di creazione per le interfacce Blob e Clob dell'API JDBC, viene fornito un supporto per inserire i BLOB nel database e richiamarli fuori dal database direttamente come altri tipi. Ad esempio, il metodo setBinaryStream può gestire una colonna del database di tipo Blob. L'argomento Esempio: Blob mostra alcuni dei modi comuni in cui è possibile inserire un BLOB nel database o richiamarlo dal database.

Gestione dell'API dell'oggetto Blob

I BLOB sono definiti in JDBC come un'interfaccia di cui i programmi di controllo forniscono implementazioni. Tale interfaccia dispone di una serie di metodi che è possibile utilizzare per interagire con l'oggetto Blob. L'Esempio: utilizzo di BLOB mostra alcune delle attività comuni che possono essere eseguite utilizzando questa API. Consultare javadoc JDBC per un elenco completo dei metodi disponibili sull'oggetto Blob.

Utilizzo del supporto JDBC 3.0 per aggiornare i BLOB

In JDBC 3.0, esiste un supporto per modificare gli oggetti LOB. Queste modifiche possono essere memorizzate nelle colonne BLOB nel database. L'argomento Esempio: aggiornamento di BLOB mostra alcune delle attività che possono essere seguite con il supporto BLOB in JDBC 3.0.

Riferimenti correlati

"Esempio: BLOB"
Questo è un esempio del modo in cui è possibile inserire un BLOB nel database o richiamarlo dal database.

"Esempio: aggiornamento di BLOB" a pagina 146
Questo è un esempio di come aggiornare i BLOB nelle applicazioni Java.

Esempio: BLOB:

Questo è un esempio del modo in cui è possibile inserire un BLOB nel database o richiamarlo dal database.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// PutGetBlobs è un'applicazione di esempio
// che mostra come gestire l'API JDBC
// per ottenere e inserire i BLOB in e dalle
// colonne database.
//
// I risultati dell'esecuzione di questo programma
```

```
// consistono in due valori BLOB in una
// nuova tabella. Sono identici e
// contengono 500k di dati byte
// casuali.
//////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      // Stabilire una Connection e una Statement con cui operare.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Ripulire l'esecuzione precedente di questa applicazione.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
      } catch (SQLException e) {
          // Ignorarlo - presupporre che la tabella non esista.
      }

      // Creare una tabella con una colonna BLOB. La dimensione predefinita della
      // colonna BLOB è 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

      // Creare un oggetto PreparedStatement che consenta di inserire un nuovo
      // oggetto Blob nel database.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

      // Creare un valore BLOB grande...
      Random random = new Random ();
      byte [] inByteArray = new byte[500000];
      random.nextBytes (inByteArray);

      // Impostare il parametro PreparedStatement. Nota: non è trasferibile
      // in tutti i programmi di controllo JDBC. Questi ultimi non dispongono di
      // supporto quando si usa setBytes per le colonne BLOB. Viene utilizzato
      // per consentire la creazione di nuovi BLOB. Consente inoltre ai programmi di
      // controllo JDBC 1.0 di gestire le colonne contenenti i dati BLOB.
      ps.setBytes(1, inByteArray);

      // Elaborare l'istruzione, inserendo il BLOB nel database.
      ps.executeUpdate();

      // Elaborare una query e ottenere il BLOB inserito fuori dal
      // database come un oggetto Blob.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
      rs.next();
      Blob blob = rs.getBlob(1);


      // Inserire di nuovo Blob nel database tramite
      // PreparedStatement.
      ps.setBlob(1, blob);
      ps.execute();
```

```
      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: aggiornamento di BLOB:

Questo è un esempio di come aggiornare i BLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// UpdateBlobs è un'applicazione di esempio
// che mostra alcune API fornendo il supporto
// per modificare gli oggetti Blob e
// riflettere tali modifiche nel
// database.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetBlobs.
//////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

      rs.next();
      Blob blob1 = rs.getBlob(1);
      rs.next();
      Blob blob2 = rs.getBlob(1);


      // Troncare un BLOB.
      blob1.truncate((long) 150000);
      System.out.println("Blob1's new length is " + blob1.length());

      // Aggiornare parte del BLOB con una nuova schiera di byte.
      // Il seguente codice contiene i byte che si trovano nelle
      // posizioni 4000-4500 e li imposta nelle posizioni 500-1000.

      // Ottenere parte del BLOB come una schiera di byte.
      byte[] bytes = blob1.getBytes(4000L, 4500);

      int bytesWritten = blob2.setBytes(500L, bytes);

      System.out.println("Bytes written is " + bytesWritten);

      // I byte vengono rilevati nella posizione 500 in blob2
      long startInBlob2 = blob2.position(bytes, 1);

      System.out.println("pattern found starting at position " + startInBlob2);
```

```
      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: utilizzo di BLOB:

Questo è un esempio di come utilizzare i BLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// UseBlobs è un'applicazione di esempio
// che mostra alcune API associate
// agli oggetti Blob.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetBlobs.
//////////////////////////////////////
import java.sql.*;

public class UseBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

      rs.next();
      Blob blob1 = rs.getBlob(1);
      rs.next();
      Blob blob2 = rs.getBlob(1);


      // Determinare la lunghezza di un LOB.
      long end = blob1.length();
      System.out.println("Blob1 length is " + blob1.length());

      // Quando si gestiscono i LOB, l'indicizzazione che si riferisce ad essi
      // si basa su 1 e non su 0 come per le stringhe e le schiere.
      long startingPoint = 450;
      long endingPoint = 500;

      // Ottenere parte del BLOB come una schiera di byte.
      byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

      // Rilevare dove viene individuato un BLOB secondario o una schiera di byte all'interno di
      // un BLOB. L'impostazione di questo programma inserisce due copie identiche di un
      // BLOB casuale nel database. Quindi, la posizione iniziale della schiera di byte
      // estratta dal blob1 può essere rilevata nella posizione iniziale
      // nel blob2. L'eccezione si verificherebbe se esistessero 50 byte casuali
      // identici nei LOB in precedenza.
      long startInBlob2 = blob2.position(outByteArray, 1);

      System.out.println("pattern found starting at position " + startInBlob2);
```

```
      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Scrittura del codice che utilizza i CLOB:

Esistono alcune attività che è possibile eseguire con colonne CLOB e DBCLOB database tramite l'API (Application Programming Interface) JDBC Java Database Connectivity. I seguenti argomenti discutono brevemente queste attività e includono esempi su come realizzarle.

Lettura dei CLOB dal database e inserimento dei CLOB nel database

Con l'API JDBC, esistono metodi per richiamare i CLOB fuori dal database e metodi per inserirli nel database. Tuttavia, non esiste una modalità standardizzata per creare un oggetto Clob. Questo non è un problema se il proprio database è già pieno di CLOB, ma costituisce un problema se si desidera gestire i CLOB da zero tramite JDBC. Invece di definire un programma di creazione per le interfacce Blob e Clob dell'API JDBC, viene fornito un supporto per inserire i CLOB nel database e richiamarli fuori dal database direttamente come altri tipi. Ad esempio, il metodo setCharacterStream può gestire una colonna del database di tipo CLOB. L'argomento Esempio: CLOB mostra alcuni dei modi comuni in cui è possibile inserire un CLOB nel database o richiamarlo dal database.

Gestione dell'API dell'oggetto Clob

I CLOB sono definiti in JDBC come un'interfaccia di cui i programmi di controllo forniscono implementazioni. Tale interfaccia dispone di una serie di metodi che è possibile utilizzare per interagire con l'oggetto Clob. L'Esempio: utilizzo di CLOB mostra alcune delle attività comuni che possono essere eseguite utilizzando questa API. Consultare Javadoc JDBC per un elenco completo dei metodi disponibili sull'oggetto Clob.

Utilizzo del supporto JDBC 3.0 per aggiornare i CLOB

In JDBC 3.0, esiste un supporto per modificare gli oggetti LOB. Queste modifiche possono essere memorizzate nelle colonne CLOB nel database. L'argomento Esempio: aggiornamento di CLOB mostra alcune delle attività che possono essere seguite con il supporto CLOB in JDBC 3.0.

Riferimenti correlati

"Esempio: CLOB"
Questo è un esempio del modo in cui è possibile inserire un CLOB in un database o richiamarlo da un database.

"Esempio: utilizzo di CLOB" a pagina 151
Questo è un esempio di come utilizzare i CLOB nelle applicazioni Java.

"Esempio: aggiornamento dei CLOB" a pagina 150
Questo è un esempio di come aggiornare i CLOB nelle applicazioni Java.

Esempio: CLOB:

Questo è un esempio del modo in cui è possibile inserire un CLOB in un database o richiamarlo da un database.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// PutGetClobs è un esempio di applicazione
// che mostra come gestire l'API JDBC
// per ottenere e inserire i CLOB in e dalle
// colonne database.
```

```
//
// I risultati dell'esecuzione di questo programma
// consistono in due valori CLOB in una
// nuova tabella. Sono identici e
// contengono 500k di dati di testo
// ripetitivo.
/////////////////////////////////////
import java.sql.*;

public class PutGetClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      // Stabilire una Connection e una Statement con cui operare.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Ripulire l'esecuzione precedente di questa applicazione.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
      } catch (SQLException e) {
          // Ignorarlo - presupporre che la tabella non esista.
      }

      // Creare una tabella con una colonna CLOB. La dimensione predefinita della
      // colonna BLOB è 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

      // Creare un oggetto PreparedStatement che consente di inserire un nuovo
      // oggetto Clob nel database.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

      // Creare un valore CLOB grande...
      StringBuffer buffer = new StringBuffer(500000);
      while (buffer.length() < 500000) {
          buffer.append("All work and no play makes Cujo a dull boy.");
      }
      String clobValue = buffer.toString();

      // Impostare il parametro PreparedStatement. Non è trasferibile
      // in tutti i programmi di controllo JDBC. Questi ultimi non dispongono di
      // supporto setBytes per le colonne CLOB. Ciò viene effettuato per
      // consentire all'utente di creare nuovi CLOB. Consente inoltre alle unità di
      // controllo JDBC 1.0 un modo per gestire le colonne contenenti
      // dati Clob.
      ps.setString(1, clobValue);

      // Elaborare l'istruzione, inserendo il clob nel database.
      ps.executeUpdate();

      // Elaborare una query e ottenere il CLOB inserito nel
      // database come oggetto Clob.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
      rs.next();
      Clob clob = rs.getClob(1);


      // Inserire di nuovo Clob nel database tramite
      // PreparedStatement.
      ps.setClob(1, clob);
```

```
      ps.execute();

      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: aggiornamento dei CLOB:

Questo è un esempio di come aggiornare i CLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// UpdateClobs è un'applicazione di esempio
// che mostra alcune API fornendo il supporto
// per modificare gli oggetti Clob e
// riflettere tali modifiche nel
// database.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetClobs.
//////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");

      rs.next();
      Clob clob1 = rs.getClob(1);
      rs.next();
      Clob clob2 = rs.getClob(1);


      // Troncare un CLOB.
      clob1.truncate((long) 150000);
      System.out.println("Clob1's new length is " + clob1.length());

      // Aggiornare una parte del CLOB con un nuovo valore String.
      String value = "Some new data for once";
      int charsWritten = clob2.setString(500L, value);

      System.out.println("Characters written is " + charsWritten);

      // I byte possono essere rilevati nella posizione 500 in clob2
      long startInClob2 = clob2.position(value, 1);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: utilizzo di CLOB:

Questo è un esempio di come utilizzare i CLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// UpdateClobs è un'applicazione di esempio
// che mostra alcune API fornendo il supporto
// per modificare gli oggetti Clob e
// riflettere tali modifiche nel
// database.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetClobs.
//////////////////////////////////////
import java.sql.*;

public class UseClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");

      rs.next();
      Clob clob1 = rs.getClob(1);
      rs.next();
      Clob clob2 = rs.getClob(1);


      // Determinare la lunghezza di un LOB.
      long end = clob1.length();
      System.out.println("Clob1 length is " + clob1.length());

      // Quando si gestiscono i LOB, l'indicizzazione che si riferisce ad essi
      // si basa su 1 e non su 0 come per le stringhe e le schiere.
      long startingPoint = 450;
      long endingPoint = 50;

      // Ottenere parte del CLOB come una schiera di byte.
      String outString = clob1.getSubString(startingPoint, (int)endingPoint);
      System.out.println("Clob substring is " + outString);

      // Rilevare dove viene trovato un CLOB secondario o una stringa all'interno di
      // un CLOB. L'impostazione di questo programma inserisce due copie identiche di un
      // CLOB ripetitivo nel database. Quindi, la posizione iniziale della stringa
      // estratta dal clob1 può essere rilevata nella posizione iniziale
      // nel clob2 se la ricerca inizia vicino alla posizione in cui inizia
   // la stringa.
      long startInClob2 = clob2.position(outString, 440);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Scrittura del codice che utilizza i Datalink:

La gestione dei Datalink dipende dal release che si sta utilizzando. In JDBC 3.0, esiste un supporto per gestire direttamente le colonne Datalink utilizzando i metodi getURL e putURL.

Con le precedenti versioni JDBC, era stato necessario gestire le colonne Datalink come se fossero colonne String. Attualmente, il database non supporta le conversioni automatiche tra tipi di dati di caratteri e Datalink. Come risultato, è necessario eseguire un'assegnazione del tipo nelle istruzioni SQL.

Esempio: Datalink:

Quest'applicazione di esempio mostra come utilizzare la API JDBC per gestire le colonne database del datalink.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// PutGetDatalinks è un'applicazione di esempio
// che mostra come utilizzare l'API JDBC
// per gestire le colonne database del datalink.
//////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrare il programma di controllo JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Errore di impostazione.
        }

        // Stabilire una Connection e una Statement con cui operare.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Ripulire l'esecuzione precedente di questa applicazione.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
        } catch (SQLException e) {
            // Ignorarlo - presupporre che la tabella non esista.
        }

        // Creare una tabella con una colonna datalink.
        s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

        // Creare un oggetto PreparedStatement che consente di aggiungere un nuovo
        // datalink al database. Poiché la conversione in un datalink non può
        // essere effettuata direttamente nel database, è possibile codificare
        // l'istruzione SQL per eseguire la conversione esplicita.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
                                                   VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

        // Impostare il datalink. Questo URL punta ad un argomento relativo alle
        // nuove funzioni di JDBC 3.0.
        ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

        // Elaborare l'istruzione, inserendo il CLOB nel database.
        ps.executeUpdate();
```

```
        // Elaborare una query e ottenere il CLOB appena inserito nel
        // database come oggetto Clob.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        String datalink = rs.getString(1);


        // Inserire tale valore di datalink nel database tramite
        // PreparedStatement. Nota: questa funzione richiede JDBC 3.0
        // supporto di JDBC 2.0.
        /*
        try {
            URL url = new URL(datalink);
            ps.setURL(1, url);
            ps.execute();
        } catch (MalformedURLException mue) {
            // Gestire in questo punto l'immissione.
        }

        rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        URL url = rs.getURL(1);
        System.out.println("URL value is " + url);
        */

        c.close(); // La chiusura del collegamento chiude anche stmt e rs.
    }
}
```

Esempio: tipi distinti:

Questo esempio illustra come utilizzare tipi distinti.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// Questo esempio di programma mostra esempi
// di diverse attività comuni che possono
// essere eseguite con tipi distinti.
////////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrare il programma di controllo JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Errore di impostazione.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Ripulire le vecchie esecuzioni.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignorarlo e presupporre che la tabella non esista.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
```

```
        } catch (SQLException e) {
            // Ignorarlo e presupporre che la tabella non esista.
        }

        // Creare il tipo, creare la tabella e inserire un valore.
        s.executeUpdate("CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)");
        s.executeUpdate("CREATE TABLE CUJOSQL.SERIALNOS (COL1 CUJOSQL.SSN)");

        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.SERIALNOS VALUES(?)");
        ps.setString(1, "399924563");
        ps.executeUpdate();
        ps.close();

        // È possibile ottenere dettagli sui tipi disponibili con i nuovi metadati in
        // JDBC 2.0
        DatabaseMetaData dmd = c.getMetaData();

        int types[] = new int[1];
        types[0] = java.sql.Types.DISTINCT;

        ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN", types);
        rs.next();
        System.out.println("Type name " + rs.getString(3) +
                           " has type " + rs.getString(4));


        // Accedere ai dati inseriti.
        rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
        rs.next();
        System.out.println("The SSN is " + rs.getString(1));

        c.close(); // La chiusura del collegamento chiude anche stmt e rs.
    }
}
```

RowSet JDBC

I RowSet sono stati inizialmente aggiunti al pacchetto facoltativo JDBC Java Database Connectivity 2.0. A differenza di alcune tra le più conosciute interfacce della specifica JDBC, la specifica RowSet è stata concepita per essere più una framework che una reale implementazione. Le interfacce RowSet definiscono una serie di funzionalità principale che possiedono tutti i RowSet. I fornitori dell'implementazione RowSet hanno una grande libertà nel definire la funzionalità necessaria per le esigenze in uno spazio specifico del problema.

Caratteristiche del RowSet:

È possibile richiedere che alcune proprietà vengano soddisfatte dai rowset. Le proprietà comuni includono la serie di interfacce da supportare tramite il rowset risultante.

I RowSet corrispondono ai ResultSet

L'interfaccia RowSet si estende all'interfaccia ResultSet, il che significa che i RowSet possiedono la capacità di eseguire tutte le funzioni che sono in grado di eseguire i ResultSet. Ad esempio, i RowSet si possono scorrere e aggiornare.

RowSet non collegati al database

Esistono due categorie di RowSet:

Collegati
: Sebbene i RowSet collegati vengano popolati con dati, questi dispongono sempre di collegamenti interni aperti al database sottostante e vengono utilizzati come wrapper per l'implementazione di ResultSet.

Scollegati

I RowSet scollegati non sono necessari per mantenere i collegamenti con la relativa sorgente dati in qualsiasi momento. È possibile scollegare i RowSet sconnessi dal database, utilizzarli in una varietà di modi e ricollegarli al database per sottoporre a mirror le modifiche apportate ad essi.

I RowSet sono componenti JavaBeans

I RowSets possiedono il supporto per la gestione dell'evento in base al modello di gestione dell'evento JavaBeans. Possiedono inoltre proprietà che è possibile impostare. È possibile utilizzare queste proprietà dal RowSet per eseguire quanto segue:

- Stabilire un collegamento al database.
- Elaborare un'istruzione SQL.
- Determinare le caratteristiche dei dati che il RowSet rappresenta e gestire altre caratteristiche interne dell'oggetto RowSet.

I RowSet sono serializzabili

È possibile serializzare e non i RowSet per consentirgli di passare in un collegamento di rete, di essere scritti su un file flat (cioè un documento di testo senza alcuna elaborazione testuale o altri caratteri di struttura), e così via.

DB2CachedRowSet:

L'oggetto DB2CachedRowSet è un RowSet scollegato, il che significa che può essere utilizzato senza essere collegato al database. La relativa implementazione aderisce in maniera precisa alla descrizione di un CachedRowSet. Il DB2CachedRowSet è un contenitore di righe di dati da un ResultSet. Il DB2CachedRowSet conserva tutti i relativi dati in modo che non abbia necessità di mantenere un collegamento diverso da quello esplicito al database durante la lettura o scrittura dati al database.

Utilizzo di DB2CachedRowSet:

Poiché è possibile scollegare e serializzare l'oggetto DB2CachedRowSet, tale oggetto è utile in ambienti in cui non è sempre pratico eseguire un programma di controllo JDBC completo (ad esempio, su PDA (Personal Digital Assistant) e su telefoni cellulari abilitati a Java.

Dal momento che l'oggetto DB2CachedRowSet è contenuto in memoria e si conoscono i relativi dati, esso può funzionare come un formato altamente ottimizzato di un ResultSet che è possibile scorrere per le applicazioni. Mentre i DB2ResultSet che è possibile scorrere normalmente subiscono una penalizzazione delle prestazioni in quanto i relativi movimenti casuali interferiscono con la capacità del programma di controllo JDBC di memorizzare nella cache righe di dati, i RowSet non hanno questo problema.

Vengono forniti due metodi su DB2CachedRowSet che creano nuovi RowSet:

- Il metodo createCopy crea un nuovo RowSet identico a quello copiato.
- Il metodo createShared crea un nuovo RowSet che condivide gli stessi dati sottostanti dell'originale.

È possibile utilizzare il metodo createCopy che distribuisce ai client i ResultSet comuni. Se non si modificano i dati della tabella, creare una copia di RowSet e passarla ad ogni client è più efficace di eseguire una query al database ogni volta.

È possibile utilizzare il metodo createShared per migliorare le prestazioni del database consentendo a vari utenti di utilizzare gli stessi dati. Ad esempio, supponiamo di avere un sito Web che mostra i venti prodotti più venduti sulla propria home page quando un client si collega. Si desidera aggiornare regolarmente le informazioni sulla propria pagina principale ma, eseguire la query per conoscere gli articoli acquistati più frequentemente ogni volta che un cliente visita la propria pagina principale, non è pratico. Utilizzando il metodo createShared, è possibile creare "cursori" in modo efficace per ogni cliente

senza dovere rielaborare la query o memorizzare una notevole quantità di informazioni in memoria. Quando è corretto, è possibile eseguire nuovamente la query per trovare i prodotti acquistati più frequentemente. I nuovi dati possono popolare il RowSet utilizzato per creare i cursori condivisi e i servlet possono utilizzarli.

I DB2CachedRowSet forniscono una funzione di elaborazione ritardata. Questa funzione consente di raggruppare ed elaborare più richieste della query al database come una singola richiesta. Consultare l'argomento "Creazione e popolamento di un DB2CachedRowSet" per eliminare una parte del sovraccarico di elaborazione cui andrebbe altrimenti sottoposto il database.

Dal momento che RowSet deve tenere un'accurata traccia di tutte le modifiche che avvengono su di esso in modo che queste vengano riflesse sul database, esiste un supporto per funzioni che annullano tali modifiche o che consentono all'utente di vedere tutte le modifiche apportate. Ad esempio, esiste un metodo showDeleted che è possibile utilizzare per indicare al RowSet di consentire all'utente di selezionare le righe cancellate. Esistono inoltre i metodi cancelRowInsert e cancelRowDelete per annullare rispettivamente gli inserimenti e le cancellazioni di righe effettuati.

L'oggetto DB2CachedRowSet offre una migliore interoperatività con altre API Java grazie al relativo supporto di gestione eventi e ai relativi metodi toCollection che consentono la conversione di un RowSet o di una parte di esso in una raccolta Java.

È possibile utilizzare il supporto di gestione eventi di DB2CachedRowSet in applicazioni GUI (graphical user interface) per controllare i pannelli, per registrare le informazioni sulle modifiche al RowSet nel momento in cui vengono apportate o per reperire informazioni sulle modifiche a origini diverse dai RowSet. Consultare "Eventi del DB2JdbcRowSet" a pagina 175 per ulteriori dettagli.

Per ulteriori informazioni sulla gestione ed il modello di eventi, consultare "DB2JdbcRowSet" a pagina 172 in quanto questo supporto funziona in modo identico per entrambi i tipi di RowSet.

Creazione e popolamento di un DB2CachedRowSet:

Esistono vari modi per inserire i dati in un DB2CachedRowSet: il metodo populate, le proprietà DB2CachedRowSet con DataSource, le proprietà DB2CachedRowSet e gli URL JDBC, il metodo setConnection(Connection), il metodo execute(Connection) e il metodo execute(int).

Utilizzo del metodo populate

I DB2CachedRowSet dispongono di un metodo populate che può essere utilizzato per inserire i dati nel RowSet da un oggetto DB2ResultSet. Segue un esempio di questo approccio.

Esempio: utilizzo del metodo populate

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Stabilire un collegamento al database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Creare un'istruzione ed utilizzarla per eseguire la query.
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");

// Creare e popolare un DB2CachedRowSet.
DB2CachedRowSet crs = new DB2CachedRowSet();
crs.populate(rs);

// Nota: scollegare ResultSet, Statement
// e Connection utilizzati per creare il RowSet.
rs.close();
```

```
stmt.close();
conn.close();

// Loop dei dati nel RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

crs.close();
```

Utilizzo delle proprietà DB2CachedRowSet e DataSource

I DB2CachedRowSet dispongono di proprietà che consentono ai DB2CachedRowSet di accettare una query SQL e un nome DataSource. Quindi essi utilizzano la query SQL e il nome DataSource per creare dati per il proprio utilizzo. Segue un esempio di questo approccio. Si presume che il riferimento al DataSource denominato BaseDataSource sia un DataSource valido precedentemente impostato.

Esempio: utilizzo delle proprietà DB2CachedRowSet e DataSource

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Creare un nuovo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Impostare le proprietà necessarie affinché
// RowSet utilizzi un DataSource per popolarsi.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet. Ciò fa in modo che
// RowSet utilizzi il DataSource e la query SQL
// specificati per popolare i propri dati. Una volta
// popolato il RowSet, si scollega dal database.
crs.execute();

// Loop dei dati nel RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventualmente, chiudere il RowSet.
crs.close();
```

Utilizzo delle proprietà DB2CachedRowSet e degli URL JDBC

I DB2CachedRowSet dispongono di proprietà che consentono ai DB2CachedRowSet di accettare una query SQL e un URL JDBC. Quindi essi utilizzano la query e l'URL JDBC per creare dati per il proprio utilizzo. Segue un esempio di questo approccio.

Esempio: utilizzo delle proprietà DB2CachedRowSet e gli URL JDBC

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Creare un nuovo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Impostare le proprietà necessarie affinché
// RowSet utilizzi un URL JDBC per popolarsi.
crs.setUrl("jdbc:db2:*local");
crs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet. Ciò fa in modo che
// RowSet utilizzi il DataSource e la query SQL
// specificati per popolare i propri dati. Una volta
```

```
// popolato il RowSet, si scollega dal database.
crs.execute();

// Loop dei dati nel RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventualmente, chiudere il RowSet.
crs.close();
```

Utilizzo del metodo setConnection(Connection) per utilizzare un collegamento database esistente

Per favorire il riutilizzo degli oggetti Connection JDBC, DB2CachedRowSet fornisce un meccanismo per inoltrare un oggetto Connection stabilito al DB2CachedRowSet utilizzato per popolare il RowSet. Se un oggetto Connection fornito dall'utente viene passato, il DB2CachedRowSet non lo scollega dopo essersi popolato.

Esempio: utilizzo del metodo setConnection(Connection) per usare un collegamento database esistente

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Stabilire un collegamento JDBC al database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Creare un nuovo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Impostare le proprietà necessarie affinché
// RowSet utilizzi un collegamento già stabilito
// per popolarsi.
crs.setConnection(conn);
crs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet. Ciò fa in modo che
// RowSet utilizzi il collegamento fornito
// precedentemente. Una volta popolato il RowSet, non
// chiude il collegamento fornito dall'utente.
crs.execute();

// Loop dei dati nel RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventualmente, chiudere il RowSet.
crs.close();
```

Utilizzo del metodo execute(Connection) per usare un collegamento database esistente

Per favorire il riutilizzo degli oggetti Connection JDBC, il DB2CachedRowSet fornisce un meccanismo per inoltrare un oggetto Connection stabilito al DB2CachedRowSet quando viene chiamato il metodo execute. Se un oggetto Connection fornito dall'utente viene passato, il DB2CachedRowSet non lo scollega dopo essersi popolato.

Esempio: utilizzo del metodo execute(Connection) per usare un collegamento database esistente

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Stabilire un collegamento JDBC al database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Creare un nuovo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();
```

```
// Impostare l'istruzione SQL che deve essere utilizzata per
// popolare il RowSet.
crs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet, trasferendo il collegamento
// che deve essere utilizzato. Una volta popolato il Rowset, non
// chiude il collegamento fornito dall'utente.
crs.execute(conn);

// Loop dei dati nel RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventualmente, chiudere il RowSet.
crs.close();
```

Utilizzo del metodo execute(int) per raggruppare le richieste database

Per ridurre il carico di lavoro del database, il DB2CachedRowSet fornisce un meccanismo per raggruppare le istruzioni SQL per vari sottoprocessi in una richiesta di elaborazione per il database.

Esempio: utilizzo del metodo execute(int) per raggruppare le richieste database

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Creare un nuovo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Impostare le proprietà necessarie affinché
// RowSet utilizzi un DataSource per popolarsi.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet. Ciò fa in modo che
// RowSet utilizzi il DataSource e la query SQL
// specificati per popolare i propri dati. Una volta
// popolato il RowSet, si scollega dal database.
// Questa versione del metodo execute accetta il numero di secondi
// di attesa dei risultati. Consentendo un ritardo,
// il RowSet può raggruppare le richieste di diversi
// utenti e elaborare solo una richiesta alla volta sul
// database sottostante.
crs.execute(5);

// Loop dei dati nel RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventualmente, chiudere il RowSet.
crs.close();
```

Accesso ai dati del DB2CachedRowSet e alla manipolazione del cursore:

Questo argomento fornisce informazioni sull'accesso ai dati DB2CachedRowSet e varie funzioni di manipolazione del cursore.

I RowSet dipendono dai metodi ResultSet. Per molte operazioni come l'accesso ai dati DB2CachedRowSet e gli spostamenti del cursore, non esiste alcuna differenza al livello di applicazione tra l'utilizzo di ResultSet e l'utilizzo di RowSet.

Accesso ai dati DB2CachedRowSet

RowSet e ResultSet accedono ai dati nella stessa maniera. Nel seguente esempio, il programma crea una tabella e la popola con vari tipi di dati utilizzando JDBC. Quando la tabella è pronta, DB2CachedRowSet viene creato e popolato con le informazioni ricavate dalla tabella. L'esempio utilizza inoltre vari metodi get della classe RowSet.

Esempio: accesso ai dati DB2CachedRowSet

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;
import java.math.*;

public class TestProgram
{
  public static void main(String args[])
  {
    // Registrare il programma di controllo.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // Non è necessario andare oltre.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Ripulire le precedenti esecuzioni
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Creare la tabella di verifica
      stmt.execute("Create table cujosql.test_table (col1 smallint, col2 int, " +
              "col3 bigint, col4 real, col5 float, col6 double, col7 numeric, " +
              "col8 decimal, col9 char(10), col10 varchar(10), col11 date, " +
              "col12 time, col13 timestamp)");
      System.out.println("Table created.");

      // Inserire alcune righe di verifica
      stmt.execute("insert into cujosql.test_table values (1, 1, 1, 1.5, 1.5, 1.5, 1.5, 1.5, 'one', 'one',
                  {d '2001-01-01'}, {t '01:01:01'}, {ts '1998-05-26 11:41:12.123456'})");

      stmt.execute("insert into cujosql.test_table values (null, null, null, null, null, null, null, null,
                  null, null, null, null, null)");
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
      System.out.println("Query executed");

      // Creare un nuovo rowset e popolarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
```

```
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Test with getObject");
      int count = 0;
      while (crs.next()) {
        System.out.println("Row " + (++count));
        for (int i = 1; i <= 13; i++) {
          System.out.println("  Col " + i + " value " + crs.getObject(i));
        }
      }

      System.out.println("Test with getXXX... ");
      crs.first();
      System.out.println("Row 1");
      System.out.println("  Col 1 value " + crs.getShort(1));
      System.out.println("  Col 2 value " + crs.getInt(2));
      System.out.println("  Col 3 value " + crs.getLong(3));
      System.out.println("  Col 4 value " + crs.getFloat(4));
      System.out.println("  Col 5 value " + crs.getDouble(5));
      System.out.println("  Col 6 value " + crs.getDouble(6));
      System.out.println("  Col 7 value " + crs.getBigDecimal(7));
      System.out.println("  Col 8 value " + crs.getBigDecimal(8));
      System.out.println("  Col 9 value " + crs.getString(9));
      System.out.println("  Col 10 value " + crs.getString(10));
      System.out.println("  Col 11 value " + crs.getDate(11));
      System.out.println("  Col 12 value " + crs.getTime(12));
      System.out.println("  Col 13 value " + crs.getTimestamp(13));
      crs.next();
      System.out.println("Row 2");
      System.out.println("  Col 1 value " + crs.getShort(1));
      System.out.println("  Col 2 value " + crs.getInt(2));
      System.out.println("  Col 3 value " + crs.getLong(3));
      System.out.println("  Col 4 value " + crs.getFloat(4));
      System.out.println("  Col 5 value " + crs.getDouble(5));
      System.out.println("  Col 6 value " + crs.getDouble(6));
      System.out.println("  Col 7 value " + crs.getBigDecimal(7));
      System.out.println("  Col 8 value " + crs.getBigDecimal(8));
      System.out.println("  Col 9 value " + crs.getString(9));
      System.out.println("  Col 10 value " + crs.getString(10));
      System.out.println("  Col 11 value " + crs.getDate(11));
      System.out.println("  Col 12 value " + crs.getTime(12));
      System.out.println("  Col 13 value " + crs.getTimestamp(13));

      crs.close();
    }
    catch (Exception ex) {
      System.out.println("SQLException: " + ex.getMessage());
                 ex.printStackTrace();
    }
  }
}
```

Manipolazione del cursore

È possibile scorrere i RowSet, i quali si comportano esattamente come un ResultSet da scorrere. Nel seguente esempio, il programma crea una tabella e la popola con i dati utilizzando JDBC. Quando la tabella è pronta, DB2CachedRowSet viene creato e popolato con le informazioni dalla tabella. L'esempio utilizza inoltre varie funzioni di manipolazione del cursore.

Esempio: manipolazione del cursore

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample1
{
  public static void main(String args[])
  {
    // Registrare il programma di controllo.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // Non è necessario andare oltre.
      System.exit(1);
  }

  try {
    Connection conn = DriverManager.getConnection("jdbc:db2:*local");

    Statement stmt = conn.createStatement();

    // Ripulire le precedenti esecuzioni
    try {
      stmt.execute("drop table cujosql.test_table");
    }
    catch (SQLException ex) {
      System.out.println("Caught drop table: " + ex.getMessage());
    }

    // Creare una tabella di verifica
    stmt.execute("Create table cujosql.test_table (col1 smallint)");
    System.out.println("Table created.");

    // Inserire alcune righe di verifica
    for (int i = 0; i < 10; i++) {
      stmt.execute("insert into cujosql.test_table values (" + i + ")");
    }
    System.out.println("Rows inserted");

    ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
    System.out.println("Query executed");

    // Creare un nuovo rowset e popolarlo...
    DB2CachedRowSet crs = new DB2CachedRowSet();
    crs.populate(rs);
    System.out.println("RowSet populated.");

    conn.close();
    System.out.println("RowSet is detached...");

    System.out.println("Use next()");
    while (crs.next()) {
      System.out.println("v1 is " + crs.getShort(1));
    }

    System.out.println("Use previous()");
    while (crs.previous()) {
      System.out.println("value is " + crs.getShort(1));
    }

    System.out.println("Use relative()");
    crs.next();
    crs.relative(9);
    System.out.println("value is " + crs.getShort(1));
```

```
    crs.relative(-9);
    System.out.println("value is " + crs.getShort(1));

    System.out.println("Use absolute()");
    crs.absolute(10);
    System.out.println("value is " + crs.getShort(1));
    crs.absolute(1);
    System.out.println("value is " + crs.getShort(1));
    crs.absolute(-10);
    System.out.println("value is " + crs.getShort(1));
    crs.absolute(-1);
    System.out.println("value is " + crs.getShort(1));

    System.out.println("Test beforeFirst()");
    crs.beforeFirst();
    System.out.println("isBeforeFirst is " + crs.isBeforeFirst());
    crs.next();
    System.out.println("move one... isFirst is " + crs.isFirst());

    System.out.println("Test afterLast()");
    crs.afterLast();
    System.out.println("isAfterLast is " + crs.isAfterLast());
    crs.previous();
    System.out.println("move one... isLast is " + crs.isLast());

    System.out.println("Test getRow()");
    crs.absolute(7);
    System.out.println("row should be (7) and is " + crs.getRow() +
                       " value should be (6) and is " + crs.getShort(1));

    crs.close();
  }
  catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
  }
 }
}
```

Modifica dei dati DB2CachedRowSet e riflesso delle modifiche nell'origine dati:

Questo argomento fornisce informazioni su come modificare le righe in DB2CachedRowSet e poi aggiornare il database sottostante.

Il DB2CachedRowSet utilizza gli stessi metodi dell'interfaccia ResultSet standard per effettuare le modifiche ai dati nell'oggetto RowSet. Non esiste alcuna differenza a livello di applicazione tra la modifica dei dati di un RowSet e la modifica dei dati di un ResultSet. Il DB2CachedRowSet fornisce il metodo acceptChanges utilizzato per riflettere le modifiche al RowSet riportate nel database da cui i dati provengono.

Cancellazione, inserimento e aggiornamento delle righe in un DB2CachedRowSet

È possibile aggiornare i DB2CachedRowSet. Nel seguente esempio, il programma crea una tabella e la popola con i dati utilizzando JDBC. Quando la tabella è pronta, DB2CachedRowSet viene creato e popolato con le informazioni ricavate dalla tabella. L'esempio fornisce inoltre vari metodi che è possibile utilizzare per aggiornare il Rowset e mostra come usare la proprietà showDeleted che consente all'applicazione di selezionare righe anche dopo che sono state cancellate. Inoltre, i metodi cancelRowInsert e cancelRowDelete vengono utilizzati nell'esempio per consentire l'annullamento dell'inserimento o della cancellazione di righe.

Esempio: cancellazione, inserimento e aggiornamento delle righe in un DB2CachedRowSet

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample2
{
  public static void main(String args[])
  {
    // Registrare il programma di controllo.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
              ex.getMessage());

      // Non è necessario andare oltre.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Ripulire le precedenti esecuzioni
      try {
        stmt.execute("drop table cujosql.test_table");
      }

      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Creare la tabella di verifica
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Table created.");

      // Inserire alcune righe di verifica
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Query executed");

      // Creare un nuovo rowset e popolarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Delete the first three rows");
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();

      crs.beforeFirst();
      System.out.println("Insert the value -10 into the RowSet");
      crs.moveToInsertRow();
```

```
  crs.updateShort(1, (short)-10);
  crs.insertRow();
  crs.moveToCurrentRow();

  System.out.println("Update the rows to be the negative of what they now are");
  crs.beforeFirst();
  while (crs.next())
    short value = crs.getShort(1);
    value = (short)-value;
    crs.updateShort(1, value);
    crs.updateRow();
  }

  crs.setShowDeleted(true);

  System.out.println("RowSet is now (value - inserted - updated - deleted)");
  crs.beforeFirst();
  while (crs.next()) {
    System.out.println("value is " + crs.getShort(1) + " " +
                crs.rowInserted() + " " +
                crs.rowUpdated() + " " +
                crs.rowDeleted());
  }

  System.out.println("getShowDeleted is " + crs.getShowDeleted());

  System.out.println("Now undo the inserts and deletes");
  crs.beforeFirst();
  crs.next();
  crs.cancelRowDelete();
  crs.next();
  crs.cancelRowDelete();
  crs.next();
  crs.cancelRowDelete();
  while (!crs.isLast()) {
    crs.next();
  }

  crs.cancelRowInsert();

  crs.setShowDeleted(false);

  System.out.println("RowSet is now (value - inserted - updated - deleted)");
  crs.beforeFirst();
  while (crs.next()) {
    System.out.println("value is " + crs.getShort(1) + " " +
                crs.rowInserted() + " " +
                crs.rowUpdated() + " " +
                crs.rowDeleted());
  }

  System.out.println("finally show that calling cancelRowUpdates works");
  crs.first();
  crs.updateShort(1, (short) 1000);
  crs.cancelRowUpdates();
  crs.updateRow();
  System.out.println("value of row is " + crs.getShort(1));
  System.out.println("getShowDeleted is " + crs.getShowDeleted());

  crs.close();

}

catch (SQLException ex) {
```

```
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

Riflesso delle modifiche ad un DB2CachedRowSet sul database sottostante

Una volta effettuate le modifiche ad un DB2CachedRowSet, esse esistono solo finché esiste l'oggetto RowSet. In pratica, eseguire delle modifiche ad un RowSet scollegato non ha alcun effetto sul database. Per riflettere le modifiche di un RowSet nel database sottostante, viene utilizzato il metodo acceptChanges. Tale metodo comunica al RowSet scollegato di ristabilire un collegamento al database e tentare di effettuare modifiche apportate al RowSet al database sottostante. Se non è possibile effettuare le modifiche in modo sicuro al database a causa di conflitti con altre modifiche di database dopo che il RowSet è stato creato, viene emessa un'eccezione e viene effettuato il rollback della transazione.

Esempio: riflesso delle modifiche ad un DB2CachedRowSet sul database sottostante

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample3
{
  public static void main(String args[])
  {
    // Registrare il programma di controllo.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // Non è necessario andare oltre.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Ripulire le precedenti esecuzioni
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Creare la tabella di verifica
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Table created.");

      // Inserire alcune righe di verifica
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Query executed");

      // Creare un nuovo rowset e popolarlo...
```

```
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Delete the first three rows");
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();

      crs.beforeFirst();
      System.out.println("Insert the value -10 into the RowSet");
      crs.moveToInsertRow();
      crs.updateShort(1, (short)-10);
      crs.insertRow();
      crs.moveToCurrentRow();

      System.out.println("Update the rows to be the negative of what they now are");
      crs.beforeFirst();
      while (crs.next()) {
        short value = crs.getShort(1);
        value = (short)-value;
        crs.updateShort(1, value);
        crs.updateRow();
      }

      System.out.println("Now accept the changes to the database");

      crs.setUrl("jdbc:db2:*local");
      crs.setTableName("cujosql.test_table");

      crs.acceptChanges();
      crs.close();

      System.out.println("And the database table looks like this:");
      conn = DriverManager.getConnection("jdbc:db2:localhost");
      stmt = conn.createStatement();
      rs = stmt.executeQuery("select col1 from cujosql.test_table");
      while (rs.next()) {
        System.out.println("Value from table is " + rs.getShort(1));
      }

      conn.close();

    }
    catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

Funzioni DB2CachedRowSet:

Oltre a funzionare come un ResultSet, la classe DB2CachedRowSet dispone di funzionalità aggiuntive che ne rende l'utilizzo più flessibile. Vengono forniti dei metodi per convertire l'intero Rowset JDBC (Java Database Connectivity) o solo una parte di esso in una raccolta Java. Inoltre, a causa della loro natura scollegata, i DB2CachedRowSet non hanno una relazione biunivoca rigida con ResultSet.

Oltre a funzionare come un ResultSet, come è stato mostrato da vari esempi, la classe DB2CachedRowSet dispone di una funzionalità aggiuntiva che ne rende l'utilizzo più flessibile. Vengono forniti dei metodi

per convertire l'intero Rowset JDBC (Java Database Connectivity) o solo una parte di esso in una raccolta Java. Inoltre, a causa della loro natura scollegata, i DB2CachedRowSet non hanno una relazione biunivoca rigida con ResultSet.

Con i metodi forniti da DB2CachedRowSet, è possibile effettuare le seguenti attività:

Acquisizione delle raccolte dai DB2CachedRowSet

Esistono tre metodi che restituiscono una sorta di raccolta da un oggetto DB2CachedRowSet. Essi sono:

- **toCollection** restituisce un ArrayList (cioè, una voce per ogni riga) di vettori (cioè, una voce per ogni colonna).
- **toCollection(int columnIndex)** restituisce un vettore contenente il valore relativo ad ogni riga della colonna.
- **getColumn(int columnIndex)** restituisce una schiera contenente il valore relativo ad ogni colonna per una colonna stabilita.

La differenza principale tra toCollection(int columnIndex) e getColumn(int columnIndex) è che il metodo getColumn può restituire una schiera di tipi primitivi. Quindi, se columnIndex rappresenta una colonna che contiene dati interi, viene restituita una schiera di numeri interi e non una contenente oggetti java.lang.Integer.

Il seguente esempio mostra come è possibile utilizzare questi metodi.

Esempio: acquisizione di raccolte dai DB2CachedRowSet

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;
import java.util.*;

public class RowSetSample4
{
  public static void main(String args[])
  {
    // Registrare il programma di controllo.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // Non è necessario andare oltre.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");
      Statement stmt = conn.createStatement();

      // Ripulire le precedenti esecuzioni
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Creare la tabella di verifica
      stmt.execute("Create table cujosql.test_table (col1 smallint, col2 smallint)");
      System.out.println("Table created.");
```

```
      // Inserire alcune righe di verifica
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ", " + (i + 100) + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
      System.out.println("Query executed");

      // Creare un nuovo rowset e popolarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Test the toCollection() method");
      Collection collection = crs.toCollection();
      ArrayList map = (ArrayList) collection;

      System.out.println("size is " + map.size());
      Iterator iter = map.iterator();
      int row = 1;
      while (iter.hasNext()) {
        System.out.print("row [" + (row++) + "]: \t");

        Vector vector = (Vector)iter.next();
        Iterator innerIter = vector.iterator();
        int i = 1;
        while (innerIter.hasNext()) {
          System.out.print(" [" + (i++) + "]=" + innerIter.next() + "; \t");
        }
        System.out.println();
      }
      System.out.println("Test the toCollection(int) method");
      collection = crs.toCollection(2);
      Vector vector = (Vector) collection;

      iter = vector.iterator();

      while (iter.hasNext()) {
        System.out.println("Iter: Value is " + iter.next());
      }

      System.out.println("Test the getColumn(int) method");
      Object values = crs.getColumn(2);
      short[] shorts = (short [])values;

      for (int i =0; i < shorts.length; i++) {
        System.out.println("Array: Value is " + shorts[i]);
      }
    }
    catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

Creazione di copie di RowSet

Il metodo createCopy crea una copia del DB2CachedRowSet. Tutti i dati associati al Rowset vengono duplicati insieme a tutte le strutture di controllo, proprietà e indicatori dello stato.

Il seguente esempio mostra come è possibile utilizzare questo metodo.

Esempio: creazione di copie di RowSet

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;

public class RowSetSample5
{
  public static void main(String args[])
  {
    // Registrare il programma di controllo.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // Non è necessario andare oltre.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Ripulire le precedenti esecuzioni
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Creare la tabella di verifica
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Table created.");

      // Inserire alcune righe di verifica
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Query executed");

      // Creare un nuovo rowset e popolarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Now some new RowSets from one.");
      DB2CachedRowSet crs2 = crs.createCopy();
      DB2CachedRowSet crs3 = crs.createCopy();

      System.out.println("Change the second one to be negated values");
      crs2.beforeFirst();
      while (crs2.next()) {
        short value = crs2.getShort(1);
        value = (short)-value;
```

```
        crs2.updateShort(1, value);
        crs2.updateRow();
      }

      crs.beforeFirst();
      crs2.beforeFirst();
      crs3.beforeFirst();
      System.out.println("Now look at all three of them again");

      while (crs.next()) {
        crs2.next();
        crs3.next();
        System.out.println("Values: crs: " + crs.getShort(1) + ", crs2: " + crs2.getShort(1) +
                   ", crs3: " + crs3.getShort(1));
      }
    }
    catch (Exception ex) {
      System.out.println("SQLException: " + ex.getMessage());
      ex.printStackTrace();
    }

  }

}
```

Creazione di condivisioni per RowSet

Il metodo createShared crea un nuovo oggetto RowSet con informazioni sullo stato ad alto livello e consente a due oggetti RowSet di condividere gli stessi dati fisici sottostanti.

Il seguente esempio mostra come è possibile utilizzare questo metodo.

Esempio: creazione di condivisioni di RowSet

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;

public class RowSetSample5
{
  public static void main(String args[])
  {
    // Registrare il programma di controllo.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // Non è necessario andare oltre.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Ripulire le precedenti esecuzioni
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
```

```
      System.out.println("Caught drop table: " + ex.getMessage());
    }

    // Creare la tabella di verifica
    stmt.execute("Create table cujosql.test_table (col1 smallint)");
    System.out.println("Table created.");

    // Inserire alcune righe di verifica
    for (int i = 0; i < 10; i++) {
      stmt.execute("insert into cujosql.test_table values (" + i + ")");
    }
    System.out.println("Rows inserted");

    ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
    System.out.println("Query executed");

    // Creare un nuovo rowset e popolarlo...
    DB2CachedRowSet crs = new DB2CachedRowSet();
    crs.populate(rs);
    System.out.println("RowSet populated.");

    conn.close();
    System.out.println("RowSet is detached...");

    System.out.println("Test the createShared functionality (create 2 shares)");
    DB2CachedRowSet crs2 = crs.createShared();
    DB2CachedRowSet crs3 = crs.createShared();

    System.out.println("Use the original to update value 5 of the table");
    crs.absolute(5);
    crs.updateShort(1, (short)-5);
    crs.updateRow();

    crs.beforeFirst();
    crs2.afterLast();

    System.out.println("Now move the cursors in opposite directions of the same data.");

    while (crs.next()) {
      crs2.previous();
      crs3.next();
      System.out.println("Values: crs: " + crs.getShort(1) + ", crs2: " + crs2.getShort(1) +
                  ", crs3: " + crs3.getShort(1));
    }
    crs.close();
    crs2.close();
    crs3.close();
   }
   catch (Exception ex) {
    System.out.println("SQLException: " + ex.getMessage());
               ex.printStackTrace();
   }
  }
}
```

DB2JdbcRowSet:

Il DB2JdbcRowSet è un RowSet collegato, il che significa che è possibile utilizzarlo solo con il supporto di un oggetto Connection, PreparedStatement o ResultSet sottostante. La relativa implementazione si conforma in maniera precisa alla descrizione di un JdbcRowSet.

Utilizzo di DB2JdbcRowSet

Dal momento che l'oggetto DB2JdbcRowSet supporta eventi descritti nella specifica 3.0 di JDBC (Java Database Connectivity) per tutti i RowSet, è possibile che esso funzioni come oggetto intermedio tra un database locale e altri oggetti cui devono essere notificate le modifiche ai dati del database.

Come esempio, supponiamo di lavorare in un ambiente dove si dispone di un database principale e vari PDA (Personal Digital Assistants) che utilizzano un protocollo wireless per collegarsi ad esso. È possibile utilizzare un oggetto DB2JdbcRowSet per spostarsi su una riga e aggiornarla utilizzando un'applicazione principale in esecuzione sul server. L'aggiornamento della riga fa in modo che il componente RowSet generi un evento. Se esiste un servizio in esecuzione che è responsabile dell'invio degli aggiornamenti ai PDA, esso può registrarsi come ″listener″ del RowSet. Ogni volta che esso riceve un evento RowSet, è possibile che generi l'aggiornamento adeguato e lo invii alle unità wireless.

Far riferimento a Esempio: eventi DB2JdbcRowSet per ulteriori informazioni.

Creazione di JDBCRowSet

Esistono vari metodi forniti per creare un oggetto DB2JDBCRowSet. Ogni oggetto viene delineato come segue.

Utilizzare le proprietà DB2JdbcRowSet e DataSources

I DB2JdbcRowSet dispongono di proprietà che accettano una query SQL e un nome DataSource. È possibile quindi utilizzare i DB2JdbcRowSet. Segue un esempio di questo approccio. Si presume che il riferimento al DataSource denominato BaseDataSource sia un DataSource valido precedentemente impostato.

Esempio: utilizzo delle proprietà DB2JdbcRowSet e DataSource

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Creare un nuovo DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Impostare le proprietà necessarie affinché
// RowSet venga elaborato.
jrs.setDataSourceName("BaseDataSource");
jrs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet. Ciò fa in modo che
// RowSet utilizzi il DataSource e la query SQL
// specificati per prepararsi all'elaborazione dati.
jrs.execute();

// Loop dei dati nel RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventualmente, chiudere il RowSet.
jrs.close();
```

Utilizzare le proprietà DB2JdbcRowSet e gli URL JDBC

I DB2JdbcRowSet dispongono di proprietà che accettano una query SQL e un URL JDBC. È possibile quindi utilizzare i DB2JdbcRowSet. Segue un esempio di questo approccio:

Esempio: utilizzo delle proprietà DB2JdbcRowSet e gli URL JDBC

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Creare un nuovo DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Impostare le proprietà necessarie affinché
// RowSet venga elaborato.
jrs.setUrl("jdbc:db2:*local");
jrs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet. Ciò fa in modo che
// RowSet utilizzi l'URL e la query SQL specificati
// precedentemente per prepararsi all'elaborazione dati.
jrs.execute();

// Loop dei dati nel RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventualmente, chiudere il RowSet.
jrs.close();
```

Utilizzare il metodo setConnection(Connection) per utilizzare un collegamento database esistente

Per favorire il riutilizzo degli oggetti Connection JDBC, il DB2JdbcRowSet consente di inoltrare un collegamento stabilito al DB2JdbcRowSet. DB2JdbcRowSet utilizza tale collegamento per prepararsi ad usarlo quando viene chiamato il metodo execute.

Esempio: utilizzo del metodo setConnection

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Stabilire un collegamento JDBC al database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Creare un nuovo DB2JdbcRowSet.
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Impostare le proprietà necessarie affinché
// RowSet utilizzi un collegamento stabilito.
jrs.setConnection(conn);
jrs.setCommand("select col1 from cujosql.test_table");

// Chiamare il metodo execute RowSet. Ciò fa in modo che
// RowSet utilizzi il collegamento fornito
// precedentemente per prepararsi all'elaborazione dati.
jrs.execute();

// Loop dei dati nel RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventualmente, chiudere il RowSet.
jrs.close();
```

Accesso ai dati e agli spostamenti del cursore

La manipolazione delle posizioni del cursore e l'accesso ai dati del database tramite un DB2JdbcRowSet vengono gestiti dall'oggetto ResultSet sottostante. Le attività che è possibile eseguire con un oggetto ResultSet si applicano anche all'oggetto DB2JdbcRowSet.

Modifica dei dati e riflesso delle modifiche sul database sottostanti

Il supporto per l'aggiornamento del database tramite DB2JdbcRowSet viene gestito completamente dall'oggetto ResultSet sottostante. Le attività che è possibile eseguire con un oggetto ResultSet si applicano anche all'oggetto DB2JdbcRowSet.

Eventi del DB2JdbcRowSet:

Tutte le implementazioni RowSet supportano la gestione eventi per situazioni che interessano altri componenti. Tale supporto consente ai componenti dell'applicazione di ″comunicare″ tra di loro quando si verificano degli eventi su di essi. Ad esempio, l'aggiornamento di una riga database tramite un RowSet può provocare che una tabella GUI (Graphical User Interface) mostrata all'utente si aggiorni.

Nel seguente esempio, il metodo principale effettua l'aggiornamento sul RowSet ed è l'applicazione principale dell'utente. Il listener è parte del proprio server wireless utilizzato dai client scollegati nel campo. È possibile collegare questi due aspetti dell'attività aziendale senza confondere il codice per i due processi. Anche se il supporto dell'evento di Rowset è stato progettato principalmente per aggiornare le GUI con i dati del database, esso funziona perfettamente per questo tipo di problema dell'applicazione.

Esempio: eventi DB2JdbcRowSet

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2JdbcRowSet;

public class RowSetEvents {
    public static void main(String args[])
    {
        // Registrare il programma di controllo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                               ex.getMessage());
            // Non è necessario andare oltre.
            System.exit(1);
        }


        try {
            // Ottenere Connection e Statement JDBC necessari per
            // impostare questo esempio.
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");
            Statement stmt = conn.createStatement();

            // Ripulire le esecuzioni precedenti.
            try {
                stmt.execute("drop table cujosql.test_table");
            } catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Creare la tabella di verifica
            stmt.execute("Create table cujosql.test_table (col1 smallint)");
            System.out.println("Table created.");

            // Popolare la tabella con i dati.
            for (int i = 0; i < 10; i++) {
                stmt.execute("insert into cujosql.test_table values (" + i + ")");
            }
            System.out.println("Rows inserted");
```

```
            // Eliminare gli oggetti di impostazione.
            stmt.close();
            conn.close();

            // Creare un nuovo rowset e impostare le proprietà necessarie per
            // elaborarlo.
            DB2JdbcRowSet jrs = new DB2JdbcRowSet();
            jrs.setUrl("jdbc:db2:*local");
            jrs.setCommand("select col1 from cujosql.test_table");
            jrs.setConcurrency(ResultSet.CONCUR_UPDATEABLE);

            // Fornire un listener all'oggetto RowSet. Questo oggetto gestisce
            // un'elaborazione speciale quando vengono effettuate alcune operazioni
            // sul RowSet.
            jrs.addRowSetListener(new MyListener());

            // Elaborare il RowSet per fornire l'accesso ai dati del database.
            jrs.execute();

            // Provocare alcuni eventi di modifica del cursore. Questi eventi fanno
            // in modo che il metodo cursorMoved nell'oggetto listener venga controllato.
            jrs.next();
            jrs.next();
            jrs.next();

            // Provocare alcuni eventi di modifica della riga. Questi eventi fanno
            // in modo che il metodo rowChanged nell'oggetto listener venga controllato.
            jrs.updateShort(1, (short)6);
            jrs.updateRow();

            // Infine, fare in modo che si verifichi un evento di modifica di RowSet. Ciò
            // fa in modo che il metodo rowSetChanged nell'oggetto listener venga controllato.
            jrs.execute();

            // Una volta completato, chiudere il RowSet.
            jrs.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}


/**
 * Questo è un esempio di listener. Questo esempio stampa messaggi che mostrano come
 * controllare un flusso dell'applicazione e offre alcuni suggerimenti
 * su cosa è possibile effettuare se l'applicazione è stata completamente implementata.
 */
class MyListener
implements RowSetListener {
    public void cursorMoved(RowSetEvent rse) {
        System.out.println("Event to do:  Cursor position changed.");
        System.out.println("  For the remote system, do nothing ");
        System.out.println("  when this event happened. The remote view of the data");
        System.out.println("  could be controlled separately from the local view.");
        try {
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("row is " + rs.getRow() + ". \n\n");
        } catch (SQLException e) {
            System.out.println("To do:  Properly handle possible problems.");
        }
    }

    public void rowChanged(RowSetEvent rse) {
        System.out.println("Event to do:  Row changed.");
        System.out.println("  Tell the remote system that a row has changed. Then,");
        System.out.println("  pass all the values only for that row to the  ");
```

```
        System.out.println("  remote system.");
        try {
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("new values are " + rs.getShort(1) + ". \n\n");
        } catch (SQLException e) {
              System.out.println("To do:  Properly handle possible problems.");
        }
    }

    public void rowSetChanged(RowSetEvent rse) {
        System.out.println("Event to do:  RowSet changed.");
        System.out.println("  If there is a remote RowSet already established, ");
        System.out.println("  tell the remote system that the values it ");
        System.out.println("  has should be thrown out. Then, pass all ");
        System.out.println("  the current values to it.\n\n");
    }
}
```

Suggerimenti sulle prestazioni per il programma di controllo JDBC diIBM Developer Kit per Java

Il programma di controllo JDBC di IBM Developer Kit per Java è progettato per essere un'interfaccia Java dalle elevate prestazioni per la gestione del database. Tuttavia, per ottenere le migliori prestazioni possibili, è necessario creare le proprie applicazioni in modo da sfruttare le potenzialità di cui dispone il programma di controllo JDBC. I seguenti suggerimenti sono considerati una buona pratica di programmazione JDBC. La maggior parte di essi non sono specifici per il programma di controllo JDBC nativo. Quindi le applicazioni scritte secondo queste direttive forniscono delle buone prestazioni anche se utilizzate con programmi di controllo JDBC diversi dal programma di controllo JDBC nativo.

Come evitare le query SELECT * SQL

SELECT * FROM... è una modalità comune per specificare una query in SQL. Spesso, non è necessario interrogare esplicitamente tutti i campi. Per ogni colonna da restituire, il programma di controllo JDBC deve eseguire un lavoro aggiuntivo di collegamento e restituzione della riga. Anche se la propria applicazione non utilizza mai una colonna particolare, è necessario che il programma di controllo JDBC ne sia a conoscenza e riservi uno spazio per il relativo utilizzo. Se le tabelle dispongono di poche colonne non utilizzate, ciò non costituisce un sovraccarico significativo. Per un numero considerevole di colonne non utilizzate, tuttavia, è possibile che il sovraccarico sia significativo. Una soluzione migliore è elencare singolarmente le colonne che interessano la propria applicazione, come riportato di seguito:

```
SELECT COL1, COL2, COL3 FROM...
```

Utilizzo di getXXX(int) invece di getXXX(String)

Utilizzare i metodi getXXX ResultSet che acquisiscono valori numerici invece delle versioni che acquisiscono i nomi colonna. Mentre la possibilità di utilizzare i propri nomi colonna invece delle costanti numeriche sembra un vantaggio, il database stesso è solo in grado di gestire gli indici colonna. Quindi, è necessario che ogni metodo getXXX che viene chiamato con un nome colonna venga definito dal programma di controllo JDBC prima che possa essere passato al database. Dal momento che i metodi getXXX vengono solitamente chiamati in loop che possono essere eseguiti milioni di volte, questo sovraccarico minimo può accumularsi rapidamente.

Come evitare le chiamate getObject per i tipi primitivi Java

Quando si acquisiscono dal database valori di tipi primitivi (int, long, float e via di seguito), è più rapido utilizzare il metodo Get specifico per il tipo primitivo (getInt, getLong, getFloat) piuttosto che utilizzare getObject. La chiamata getObject effettua il lavoro di Get per il tipo primitivo e quindi crea un oggetto da restituire all'utente. Ciò avviene normalmente nei loop, potenzialmente creando milioni di oggetti con breve durata media. L'utilizzo di getObject per comandi primitivi ha l'ulteriore svantaggio di attivare frequentemente il programma di raccolta di dati inutili, oltre a peggiorare le prestazioni.

Utilizzo di PreparedStatement piuttosto che Statement

Se si sta scrivendo un'istruzione SQL utilizzata più di una volta, essa viene eseguita meglio come PreparedStatement che come oggetto Statement. Ogni volta che si esegue un'istruzione, si esegue un processo a due fasi: l'istruzione viene preparata e quindi elaborata. Quando si utilizza un'istruzione preparata, essa viene preparata solo nel momento in cui viene creata, non ogni volta che viene eseguita. Sebbene sia comprovato che una PreparedStatement si esegue più rapidamente di una Statement, questo vantaggio viene spesso ignorato dai programmatori. Grazie all'incremento delle prestazioni fornito dalle PreparedStatement, è consigliabile utilizzarle nel disegno delle proprie applicazioni quando possibile (consultare più avanti "Utilizzo della creazione di lotti PreparedStatement" a pagina 179).

Come evitare chiamate DatabaseMetaData

È necessario essere consapevoli che alcune chiamate DatabaseMetaData possono risultare dispendiose. In particolare, i metodi getBestRowIdentifier, getCrossReference, getExportedKeys e getImportedKeys potrebbero richiedere molte risorse. Alcune chiamate DataBaseMetaData implicano condizioni di unione complesse su tabelle a livello di sistema. Utilizzarle soltanto se l'utente desidera le relative informazioni, non esclusivamente per convenienza.

Utilizzo del livello di commit corretto per la propria applicazione

JDBC fornisce numerosi livelli di commit che stabiliscono l'influenza reciproca di più transazioni rispetto al sistema (consultare Transazioni per ulteriori dettagli). Il valore predefinito consente di utilizzare il livello di commit minimo. Ciò significa che le transazioni possono esaminare una parte del lavoro reciproco tramite limiti di commit. Ciò introduce la possibilità di alcune anomalie del database. Alcuni programmatori aumentano il livello di commit in modo da non preoccuparsi quando si verificano queste anomalie. Occorre tenere presente che livelli più elevati di commit implicano che il database dipenda da vincoli particolareggiati. Ciò limita la capacità di agire simultaneamente di cui può disporre il sistema, rallentando in modo considerevole le prestazioni di alcune applicazioni. Spesso non è possibile che si verifichino in primo luogo le condizioni dell'anomalia a causa della progettazione dell'applicazione. Cercare di capire cosa si sta tentando di portare a termine e limitare il proprio livello di isolamento della propria transazione al livello più basso che è possibile utilizzare in modo sicuro.

Memorizzazione dei dati in Unicode

Java richiede che tutti i dati carattere che gestisce (Stringhe) siano in Unicode. Quindi, qualsiasi tabella che non dispone di dati Unicode richiede che il programma di controllo JDBC converta i dati in un senso e nell'altro, in modo da inserirli nel database e richiamarli da esso. Se la tabella è già in Unicode, il programma di controllo JDBC non ha necessità di convertire i dati e può, quindi, inserirli dal database più rapidamente. Occorre tenere presente che i dati in Unicode non possono gestire applicazioni diverse da Java, che non sono in grado di operare con Unicode. Occorre inoltre tenere a mente che i dati non di caratteri non hanno un'esecuzione più rapida, dato che non esiste mai alcuna conversione di essi. Un'ulteriore considerazione è che i dati memorizzati in Unicode impiegano il doppio dello spazio rispetto a dati ad un byte singolo. Se si dispone di più colonne di caratteri lette numerose volte, tuttavia, è possibile che le prestazioni ottenute memorizzando i propri dati in Unicode siano significative.

Utilizzo delle procedure memorizzate

L'utilizzo delle procedure memorizzate è supportato in Java. Le procedure memorizzate possono essere eseguite più rapidamente consentendo al programma di controllo JDBC di effettuare SQL statici invece di SQL dinamici. Non creare procedure memorizzate per ogni singola istruzione SQL che si esegue nel proprio programma. Dove possibile, tuttavia, creare una procedura memorizzata che esegua un gruppo di istruzioni SQL.

Utilizzo di BigInt invece di Numeric o Decimal

Invece di utilizzare campi Numeric o Decimal che hanno una scala di 0, utilizzare il tipo di dati BigInt. BigInt effettua direttamente la conversione nel tipo primitivo Java Long mentre i tipi di dati Numeric o Decimal effettuano la conversione in oggetti String o BigDecimal. Come notato in "Come evitare chiamate DatabaseMetaData" a pagina 178, l'utilizzo di tipi di dati primitivi è preferibile rispetto all'utilizzo di tipi che richiedono una creazione dell'oggetto.

Chiusura esplicita delle proprie risorse JDBC quando si è terminato di utilizzarle

È necessario che l'applicazione chiuda esplicitamente ResultSet, Statement e Connection quando non sono più necessari. Ciò consente la ripulitura delle risorse nel modo più efficiente possibile e può migliorare le prestazioni. Inoltre, le risorse possono fare in modo che le perdite di risorse e vincoli database siano mantenuti più del necessario. Ciò può provocare errori dell'applicazione o simultaneità ridotta nelle applicazioni.

Utilizzo della creazione di lotti di collegamenti

La creazione di lotti di collegamenti è una strategia tramite cui è possibile riutilizzare oggetti Connection JDBC per più utenti invece della richiesta di ogni utente che crea il relativo oggetto Connection. Gli oggetti Connection sono dispendiosi da creare. Invece di far creare ad ogni utente un nuovo oggetto, sarebbe opportuno condividere un lotto di oggetti in applicazioni dove le prestazioni sono critiche. Molti prodotti (come WebSphere) forniscono il supporto di creazione di lotti Connection che è possibile utilizzare con un piccolo sforzo supplementare da parte dell'utente. Se non si intende utilizzare un prodotto con il supporto di creazione dei lotti di collegamenti o si preferisce crearne uno proprio per un migliore controllo sulla modalità di lavoro ed esecuzione del lotto, è abbastanza semplice effettuare queste operazioni.

Utilizzo della creazione di lotti PreparedStatement

La creazione di lotti Statement funziona come la creazione di lotti Connection. Invece di inserire soltanto oggetti Connection in un lotto, inserirvi un oggetto che contiene Connection e PreparedStatement. Quindi, richiamare tale oggetto ed accedere all'istruzione specifica che si desidera utilizzare. Ciò può migliorare considerevolmente le prestazioni.

Utilizzo di un SQL efficace

Dal momento che JDBC è creato su SQL, quasi tutto ciò che si applica per un SQL efficace, si applica anche per un JDBC efficace. Quindi, JDBC trae vantaggi dalle query ottimizzate, dagli indici scelti in modo appropriato e da altri aspetti di una buona progettazione SQL.

Accesso ai database utilizzando il supporto IBM Developer Kit per Java DB2 SQLJ

Il supporto DB2 SQLJ (Structured Query Language for Java) si basa sullo standard ANSI SQLJ. Il supportoDB2 SQLJ è contenuto all'interno di IBM Developer Kit per Java. Il supporto DB2 SQLJ consente la scrittura, la creazione e l'esecuzione di applicazioni incorporate SQL per Java.

Il supporto SQLJ fornito da IBM Developer Kit per Java comprende le classi del tempo di esecuzione SQLJ ed è disponibile in /QIBM/ProdData/Java400/ext/runtime.zip.

Configurazione SQLJ

Prima di poter utilizzare SQLJ nelle applicazioni Java sul server, è necessario prepararlo ad utilizzare SQLJ. Per ulteriori informazioni, consultare l'argomento relativo alla configurazione di SQLJ.

Strumenti SQLJ

Anche gli strumenti seguenti sono inclusi nel supporto SQLJ fornito da IBM Developer Kit per Java:

- Il programma di conversione SQLJ, `sqlj`, sostituisce le istruzioni incorporate SQL nel programma SQLJ con istruzioni di origine Java e genera un profilo serializzato che contiene le informazioni sulle operazioni SQLJ rilevate nel programma SQLJ.
- Il programma di personalizzazione del profilo DB2 SQLJ, `db2profc`, precompila le istruzioni SQL memorizzate nel profilo generato e genera un pacchetto nel database DB2.
- La stampante del profilo DB2 SQLJ, `db2profp`, stampa il contenuto di un profilo personalizzato DB2 in testo normale.
- Il programma di installazione del programma di controllo del profilo SQLJ, `profdb`, installa e disinstalla i programmi di controllo della classe di debug in una serie esistente di profili binari.
- Lo strumento di conversione del profilo SQLJ, `profconv`, converte un'istanza di profilo serializzata in un formato di classe Java.

Nota: è necessario che questi strumenti siano in esecuzione nel Qshell Interpreter.

Limitazioni DB2 SQLJ

Quando le applicazioni DB2 vengono create con SQLJ, è necessario tenere presente le seguenti limitazioni:

- Il supporto DB2 SQLJ, si adegua alle limitazioni standard del Database universale DB2 sull'immissione delle istruzioni SQL.
- Il programma di personalizzazione del profilo DB2 SQLJ, deve essere eseguito solo su profili associati a collegamenti con il database locale.
- L'implementazione di riferimento SQLJ richiede JDK 1.1 o una versione successiva. Consultare l'argomento relativo al Supporto di più JDK (Java Development Kit) per ulteriori informazioni sull'esecuzione di più versioni del JDK (Java Development Kit).

Concetti correlati

"Profili SQLJ (Structured Query Language for Java)"
I profili sono creati dal programma di conversione SQLJ, sqlj, quando si converte il file origine SQLJ. I profili sono file binari serializzati. Questo è il motivo per cui questi file possiedono un'estensione .ser. Questi file contengono le istruzioni SQL dal file origine SQLJ associato.

"Supporto per più opzioni 5761-JV1 LP" a pagina 6
La piattaforma System i5 supporta più versioni dei JDK (Java Development Kit) e di J2SE (Java 2 Platform, Standard Edition).

"Inserimento delle istruzioni SQL nell'applicazione Java" a pagina 185
Le istruzioni SQL statiche in SQLJ si trovano nelle clausole SQLJ. Le clausole SQLJ iniziano con `#sql` e terminano con un carattere punto e virgola (;).

Attività correlate

"Configurazione del system per l'utilizzo di SQLJ" a pagina 192
Prima di eseguire un programma Java che contiene istruzioni SQLJ incorporate, assicurarsi di configurare il server all'utilizzo di SQLJ. Il supporto SQLJ richiede di modificare la variabile di ambiente CLASSPATH per il server.

"Compilazione ed esecuzione dei programmi SQLJ" a pagina 189
Se il programma Java possiede istruzioni SQLJ incorporate, è necessario seguire una procedura speciale per sua compilazione ed esecuzione.

Profili SQLJ (Structured Query Language for Java)

I profili sono creati dal programma di conversione SQLJ, sqlj, quando si converte il file origine SQLJ. I profili sono file binari serializzati. Questo è il motivo per cui questi file possiedono un'estensione .ser. Questi file contengono le istruzioni SQL dal file origine SQLJ associato.

Per generare dei profili dal codice sorgente SQLJ, eseguire “Programma di conversione SQLJ (structured query language for Java) (sqlj)” sul file .sqlj.

Per ulteriori informazioni, consultare “Compilazione ed esecuzione dei programmi SQLJ” a pagina 189.

Programma di conversione SQLJ (structured query language for Java) (sqlj)

Il programma di conversione SQLJ, sqlj, crea un profilo serializzato contenente informazioni sulle operazioni SQL rilevate nel programma SQLJ. Il programma di conversione SQLJ utilizza il file /QIBM/ProdData/Java400/ext/translator.zip.

Per ulteriori informazioni sul profilo, seguire questo collegamento: Profilo.

Precompilazione delle istruzioni SQL in un profilo utilizzando il programma di personalizzazione del profilo DB2 SQLJ, db2profc

È possibile utilizzare il programma di personalizzazione del profilo DB2 SQLJ, db2profc, per fare in modo che l’applicazione Java operi in maniera più efficiente con il database.

Il programma di personalizzazione DB2 SQLJ effettua le seguenti operazioni:

- Precompila le istruzioni SQL memorizzate in un profilo e crea un pacchetto nel database DB2.
- Personalizza il profilo SQLJ sostituendo le istruzioni SQL con riferimenti all’istruzione associata nel pacchetto creato.

Per precompilare le istruzioni SQL in un profilo, immettere quanto segue sulla richiesta comandi Qshell:

```
db2profc MyClass_SJProfile0.ser
```

Dove *MyClass_SJProfile0.ser* rappresenta il nome del profilo che si desidera precompilare.

Utilizzo e sintassi del programma di personalizzazione del profilo DB2 SQLJ

db2profc[*options*] <*SQLJ_profile_name*>

Dove *SQLJ_profile_name* rappresenta il nome del profilo da stampare e *options* rappresenta l’elenco di opzioni desiderate.

Le opzioni disponibili per db2profp sono le seguenti:

- `-URL=<JDBC_URL>`
- `-user=<username>`
- `-password=<password>`
- `-package=<library_name/package_name>`
- `-commitctrl=<commitment_control>`
- `-datefmt=<date_format>`
- `-datesep=<date_separator>`
- `-timefmt=<time_format>`
- `-timesep=<time_separator>`
- `-decimalpt=<decimal_point>`
- `-stmtCCSID=<CCSID>`
- `-sorttbl=<library_name/sort_sequence_table_name>`
- `-langID=<language_identifier>`

Quanto segue rappresenta le descrizioni di queste opzioni:

-URL=<*JDBC_URL***>**

Dove *JDBC_URL* rappresenta l’URL del collegamento JDBC. La sintassi relativa all’URL è:

```
"jdbc:db2:systemName"
```

Per ulteriori informazioni, consultare “Accesso al database System i5 con il programma di controllo JDBC IBM Developer Kit per Java” a pagina 33.

-user=<*username*>
Dove *username* rappresenta il nome utente. Il valore predefinito è l'ID dell'utente corrente che è collegato per un collegamento locale.

-password=<*password*>
Dove *password* rappresenta la parola d'ordine. Il valore predefinito è la parola d'ordine dell'utente corrente collegato per il collegamento locale.

-package=<*library name/package name*>
Dove *library name* rappresenta la libreria in cui è stato inserito il pacchetto e *package name* rappresenta il nome del pacchetto da creare. Il nome della libreria predefinito è QUSRSYS. Il nome del pacchetto predefinito è creato dal nome del profilo. La lunghezza massima relativa al nome del pacchetto è pari a 10 caratteri. Dal momento che il nome del profilo SQLJ risulta sempre superiore a 10 caratteri, il nome del pacchetto predefinito creato è diverso dal nome del profilo. Il nome del pacchetto predefinito viene creato concatenando le prime lettere del nome del profilo con il numero chiave del profilo. Se il numero chiave del profilo è superiore a 10 caratteri, allora vengono utilizzati gli ultimi 10 caratteri di tale numero per il nome del pacchetto predefinito. Ad esempio, la seguente tabella indica alcuni nomi di profilo e i relativi nomi del pacchetto predefiniti:

Nome profilo	Nome pacchetto predefinito
App_SJProfile0	App_SJPro0
App_SJProfile01234	App_S01234
App_SJProfile012345678	A012345678
App_SJProfile01234567891	1234567891

-commitctrl=<*commitment_control*>
Dove *commitment_control* rappresenta il livello del controllo di commit che si desidera. È possibile che il controllo di commit abbia uno qualunque dei seguenti valori carattere:

Valore	Definizione
C	*CHG. Sono possibili letture errate, non ripetibili e fantasma.
S	*CS. Non sono possibili letture errate, ma sono possibili letture non ripetibili e fantasma.
A	*ALL. Non sono possibili letture errate e non ripetibili, ma sono possibili letture fantasma.
N	*NONE. Non sono possibili letture errate, non ripetibili e fantasma. Questo è il valore predefinito

-datefmt=<*date_format*>
Dove *date_format* rappresenta il tipo di formattazione della data che si desidera. È possibile che il formato della data possegga uno qualunque dei seguenti valori:

Valore	Definizione
USA	IBM USA standard (mm.dd.aaaa,hh:mm a.m., hh:mm p.m.)
ISO	International Standards Organization (aaaa-mm-gg, hh.mm.ss) Questo è il valore predefinito.

Valore	Definizione
EUR	IBM European Standard (dd.mm.aaaa, hh.mm.ss)
JIS	Japanese Industrial Standard Christian Era (aaaa-mm-dd, hh:mm:ss)
MDY	Mese/Giorno/Anno (mm/g/aa)
DMY	Giorno/Mese/Anno (gg/mm/aa)
YMD	Anno/Mese/Giorno (aa/mm/gg)
JUL	Giuliano (aa/ggg)

Il formato della data viene utilizzato durante l'accesso alle colonne di risultati della data. Tutti i campi di emissione della data vengono restituiti nel formato specifico. Per le stringhe di immissione della data, il valore specificato viene utilizzato per determinare se la data è specificata o meno in un formato valido. Il valore predefinito è ISO.

-datesep=<*date_separator***>**

Dove *date_separator* rappresenta il tipo di separatore che si desidera utilizzare. Il separatore della data è utilizzato durante l'accesso alle colonne di risultati della data. È possibile che tale separatore assuma uno qualunque dei seguenti valori:

Valore	Definizione
/	Si utilizza una barra.
.	Si utilizza un punto.
,	Si utilizza una virgola.
-	Si utilizza un trattino. Questo è il valore predefinito
spazio	Si utilizza uno spazio.

-timefmt=<*time_format***>**

Dove *time_format* rappresenta il formato che si desidera utilizzare per visualizzare i campi dell'ora. Il formato dell'ora si utilizza durante l'accesso alle colonne di risultati dell'ora. Per le stringhe di immissione dell'ora, viene specificato il valore utilizzato per stabilire se l'ora è determinata in un formato valido. È possibile che il formato dell'ora assuma uno qualunque dei seguenti valori:

Valore	Definizione
USA	IBM USA standard (mm.dd.aaaa,hh:mm a.m., hh:mm p.m.)
ISO	International Standards Organization (aaaa-mm-gg, hh.mm.ss) Questo è il valore predefinito.
EUR	IBM European Standard (dd.mm.aaaa, hh.mm.ss)
JIS	Japanese Industrial Standard Christian Era (aaaa-mm-dd, hh:mm:ss)
HMS	Ora/Minuto/Secondo (hh:mm:ss)

-timesep=<*time_separator***>**

Dove *time_separator* rappresenta il carattere che si desidera utilizzare per l'accesso alle colonne dei risultati dell'ora. È possibile che il separatore dell'ora assuma uno qualunque dei seguenti valori:

Valore	Definizione
:	Si utilizzano i due punti.
.	Si utilizza un punto. Questo è il valore predefinito

Valore	Definizione
,	Si utilizza una virgola.
spazio	Si utilizza uno spazio.

-decimalpt=<*decimal_point***>**
Dove *decimal_point* rappresenta il punto decimale che si desidera utilizzare. Si utilizza il punto decimale per le costanti numeriche nelle istruzioni SQL. È possibile che il punto decimale assuma uno qualunque dei seguenti valori:

Valore	Definizione
.	Si utilizza un punto. Questo è il valore predefinito
,	Si utilizza una virgola.

-stmtCCSID=<*CCSID***>**
Dove *CCSID* rappresenta l'identificativo della serie di caratteri codificati per le istruzioni SQL preparate nel pacchetto. Il valore del lavoro durante la personalizzazione rappresenta il valore predefinito.

-sorttbl=<*library_name/sort_sequence_table_name***>**
Dove *library_name/sort_sequence_table_name* rappresenta il nome della tabella e dell'ubicazione della tabella di sequenze di ordinamento che si desidera utilizzare. La tabella di sequenze di ordinamento viene utilizzata per il confronto di stringhe nelle istruzioni SQL. Il nome della libreria e il nome della tabella di sequenze di ordinamento possiedono ognuno un limite di 10 caratteri. Il valore predefinito viene preso dal lavoro durante la personalizzazione.

-langID=<*language_identifier***>**
Dove *language identifier* rappresenta l'identificativo della lingua che si desidera utilizzare. Il valore predefinito relativo all'identificativo della lingua viene rilevato dal lavoro corrente durante la personalizzazione. L'identificativo della lingua si utilizza insieme alla tabella di sequenze di ordinamento.

Informazioni correlate

Programmazione SQL

Stampa del contenuto dei profili DB2 SQLJ (db2profp e profp)

La stampante del profilo DB2 SQLJ, db2profp, stampa il contenuto di un profilo personalizzato DB2 in testo chiaro. La stampante di profilo, profp, stampa il contenuto dei profili creati dal programma di conversione SQLJ in testo chiaro.

Per stampare il contenuto dei profili creati dal programma di conversione SQLJ in testo chiaro, utilizzare il programma di utilità profp come segue:

```
profp MyClass_SJProfile0.ser
```

Dove *MyClass_SJProfile0.ser* rappresenta il nome del profilo che si desidera stampare.

Per stampare il contenuto di una versione personalizzata DB2 del profilo in testo chiaro, utilizzare il programma di utilità db2profp come segue:

```
db2profp MyClass_SJProfile0.ser
```

Dove *MyClass_SJProfile0.ser* rappresenta il nome del profilo che si desidera stampare.

Nota: se si esegue db2profp su un profilo non personalizzato, questo indica che il profilo non è stato personalizzato. Se si esegue profp su un profilo personalizzato, questo visualizza il contenuto del profilo senza le personalizzazioni.

Utilizzo e sintassi della Stampante di profilo DB2 SQLJ:

```
db2profp [opzioni] <nome_profilo_SQLJ>
```

Dove *nome_profilo_SQLJ* rappresenta il nome del profilo da stampare e *opzioni* rappresenta l'elenco di opzioni desiderate.

Le opzioni disponibili per db2profp sono le seguenti:

-URL=<*JDBC_URL***>**
: Dove *JDBC_URL* rappresenta l'URL cui ci si desidera collegare. Per ulteriori informazioni, consultare "Accesso al database System i5 con il programma di controllo JDBC IBM Developer Kit per Java" a pagina 33.

-user=<*nomeutente***>**
: Dove *nomeutente* rappresenta il nome dell'utente presente nel profilo utente.

-password=<*parolad'ordine***>**
: Dove *parolad'ordine* rappresenta la parola d'ordine del profilo utente.

Programma di installazione del programma di controllo del profilo SQLJ (profdb)

Il programma di installazione del programma di controllo del profilo SQLJ (profdb) installa e disinstalla i programmi di controllo della classe di debug. Tali programmi di controllo vengono installati in una serie esistente di profili binari. Una volta che i programmi di controllo della classe di debug sono stati installati, tutte le chiamate RTStatement e RTResultSet eseguite durante il tempo di esecuzione dell'applicazione vengono registrate. Queste possono essere registrate su un file o su un'emissione standard. È possibile che le registrazioni siano poi analizzate per verificare gli errori nella traccia e nella funzionalità dell'applicazione. Tenere presente che solo le chiamate eseguite sull'interfaccia sottostante RTStatement e RTResultSetcall nel tempo di esecuzione vengono controllate.

Per installare i programmi di controllo della classe di debug, immettere quanto segue nella richiesta comandi Qshell:

```
profdb MyClass_SJProfile0.ser
```

Dove *MyClass_SJProfile0.ser* rappresenta il nome del profilo generato dal programma di conversione SQLJ.

Per disinstallare i programmi di controllo della classe di debug, immettere quanto segue nella richiesta comandi di Qshell:

```
profdb -Cuninstall MyClass_SJProfile.ser
```

Dove *MyClass_SJProfile0.ser* rappresenta il nome del profilo generato dal programma di conversione SQLJ.

Conversione di un'istanza di profilo serializzata in un formato di classe Java utilizzando lo strumento di conversione del profilo SQLJ (profconv)

Lo strumento di conversione del profilo SQLJ (profconv) converte un'istanza di profilo serializzata in un formato di classeJava. Lo strumento profconv è necessario perché alcuni browser non supportano il caricamento di un oggetto serializzato da un file di risorsa associato ad un'applet. Eseguire il programma di utilità profconv per operare la conversione.

Per eseguire il programma di utilità profconv, immettere quanto segue sulla riga comandi di Qshell:

```
profconv MyApp_SJProfile0.ser
```

dove *MyApp_SJProfile0.ser* rappresenta il nome dell'istanza del profilo che si desidera convertire.

Lo strumento `profconv` richiama `sqlj -ser2class`. Consultare sqlj per le opzioni sulla riga comandi.

Inserimento delle istruzioni SQL nell'applicazione Java

Le istruzioni SQL statiche in SQLJ si trovano nelle clausole SQLJ. Le clausole SQLJ iniziano con `#sql` e terminano con un carattere punto e virgola (;).

Prima di creare qualsiasi clausola SQLJ nell'applicazione Java, importare i seguenti pacchetti:
- import java.sql.*;
- import sqlj.runtime.*;
- import sqlj.runtime.ref.*;

Le clausole SQLJ più semplici sono le clausole che è possibile elaborare e che sono composte dal token `#sql` seguito da un'istruzione SQL racchiusa tra parentesi graffe. Ad esempio, la clausola SQLJ seguente può comparire se un'istruzione Java viene visualizzata in modo valido:

```
#sql { DELETE FROM TAB };
```

L'esempio precedente cancella tutte le righe presenti nella tabella denominata TAB.

In una clausola di elaborazione SQLJ, i token che compaiono all'interno delle parentesi graffe sono token SQL o variabili host. Tutte le variabili host sono distinte dal carattere due punti (:). I token SQL non compaiono mai fuori dalle parentesi graffe di una clausola di elaborazione SQLJ. Ad esempio, il metodo Java seguente inserisce degli argomenti all'interno di una tabella SQL:

```
public void insertIntoTAB1 (int x, String y, float z) throws SQLException
{
    #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

Il corpo del metodo consiste in una clausola di elaborazione SQLJ contenente le variabili host x, y e z. .

In generale, i token SQL non sono sensibili al maiuscolo e al minuscolo (eccetto per quanto riguarda gli identificativi delimitati da virgolette) ed è possibile scriverli in maiuscolo, in minuscolo o con entrambi i caratteri. I token *Java*, tuttavia, sono sensibili al maiuscolo e al minuscolo. Per una maggiore chiarezza negli esempi, i token SQL non sensibili al maiuscolo e al minuscolo sono scritti in maiuscolo, mentre i token Java sono scritti in minuscolo o con entrambi i caratteri. In tutto questo argomento, il minuscolo `null` si utilizza per rappresentare il valore Java ″null″ e il maiuscolo `NULL` si utilizza per rappresentare il valore SQL ″null″.

I tipi seguenti di creazioni SQL possono comparire nei programmi SQLJ:
- Query, ad esempio: istruzioni ed espressioni SELECT.
- Istruzioni di modifica dati SQL (DML), ad esempio: INSERT, UPDATE, DELETE.
- Istruzioni di dati, ad esempio: FETCH, SELECT..INTO.
- Istruzioni di controllo transazione, ad esempio: COMMIT, ROLLBACK, ecc.
- Istruzioni DDL (Data Definition Language, anche noti come Schema Manipulation Language), ad esempio: CREATE, DROP, ALTER.
- Chiamate a procedure memorizzate, ad esempio: CALL MYPROC(:x, :y, :z)
- Richiamo di funzioni memorizzate, ad esempio: VALUES(MYFUN(:x))

Variabili host in SQLJ (Structured Query Language for Java):

Gli argomenti per le istruzioni SQL incorporate vengono passati attraverso le variabili host. Le variabili host sono variabili del linguaggio host e possono comparire in istruzioni SQL.

Le variabili host sono composte di massimo tre parti:
- Un prefisso rappresentato dai due punti (:).
- Una variabile host Java che è un identificativo Java per un parametro, una variabile o un campo.
- Un identificativo della modalità del parametro facoltativo.

  È possibile che questo identificativo della modalità sia uno dei seguenti:
  IN, OUT o INOUT.

La valutazione di un identificativo Java non ha effetti indiretti in un programma Java, in questo modo può comparire più volte in un codice Java generato per sostituire una clausola SQLJ.

La query seguente contiene la variabile host, `:x`. Questa variabile host è la variabile, il campo o il parametro Java `x` visibile nell'ambito contenente la query.

```
SELECT COL1, COL2 FROM TABLE1 WHERE :x > COL3
```

Esempio: inserimento di istruzioni SQL nell'applicazione Java:

La seguente applicazione SQLJ di esempio, App.sqlj, utilizza SQL statico per richiamare e aggiornare i dati dalla tabella EMPLOYEE del database di esempio DB2.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ; // 1
#sql iterator App_Cursor2 (String) ;

class App
{

  /**********************
   ** Registrare unità **
   **********************/

 static
 {
   try
   {
     Class.forName("com.ibm.db2.jdbc.app.DB2Driver").newInstance();
   }
   catch (Exception e)
   {
     e.printStackTrace();
   }
 }

  /********************
   **      Main      **
   ********************/

 public static void main(String argv[])
 {
   try
   {
     App_Cursor1 cursor1;
     App_Cursor2 cursor2;

     String str1 = null;
     String str2 = null;
     long   count1;

     // L'URL è jdbc:db2:dbname
     String url = "jdbc:db2:sample";

     DefaultContext ctx = DefaultContext.getDefaultContext();
     if (ctx == null)
     {
       try
       {
         // collegarsi con id/parole d'ordine predefiniti
```

```
      Connection con = DriverManager.getConnection(url);
      con.setAutoCommit(false);
      ctx = new DefaultContext(con);
    }
    catch (SQLException e)
    {
      System.out.println("Error: could not get a default context");
      System.err.println(e) ;
      System.exit(1);
    }
    DefaultContext.setDefaultContext(ctx);
  }

  // richiamare i dati dal database
  System.out.println("Retrieve some data from the database.");
  #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

  // visualizzare la serie di risultati
  // cursor1.next() restituisce false quando non ci sono più righe
  System.out.println("Received results:");
  while (cursor1.next()) // 3
  {
    str1 = cursor1.empno(); // 4
    str2 = cursor1.firstnme();

    System.out.print (" empno= " + str1);
    System.out.print (" firstname= " + str2);
    System.out.println("");
  }
  cursor1.close(); // 9

  // richiamare il numero di impiegati dal database
  #sql { SELECT count(*) into :count1 FROM employee }; // 5
  if (1 == count1)
    System.out.println ("There is 1 row in employee table");
  else
    System.out.println ("There are " + count1
                        + " rows in employee table");

  // aggiornare il database
  System.out.println("Update the database.");
  #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

  // richiamare i dati aggiornati dal database
  System.out.println("Retrieve the updated data from the database.");
  str1 = "000010";
  #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

  // visualizzare la serie di risultati
  // cursor2.next() restituisce false quando non ci sono più righe
  System.out.println("Received results:");
  while (true)
  {
    #sql { FETCH :cursor2 INTO :str2 }; // 7
    if (cursor2.endFetch()) break; // 8

    System.out.print (" empno= " + str1);
    System.out.print (" firstname= " + str2);
    System.out.println("");
  }
  cursor2.close(); // 9

  // rollback dell'aggiornamento
  System.out.println("Rollback the update.");
  #sql { ROLLBACK work };
  System.out.println("Rollback done.");
}
```

```
   catch( Exception e )
   {
     e.printStackTrace();
   }
 }
}
```

[1]Dichiarazione di iteratori. Questa sezione dichiara due tipi di iteratori:

- App_Cursor1: dichiara i tipi e i nomi dei dati di colonna e restituisce i valori delle colonne a seconda del nome di colonna (collegamento denominato a colonne).
- App_Cursor2: dichiara i tipi dei dati di colonna e restituisce i valori delle colonne tramite la posizione della colonna (collegamento di posizione alle colonne).

[2]Inizializzazione dell'iteratore. L'oggetto iteratore cursor1 viene inizializzato utilizzando il risultato di una query. La query memorizza il risultato in cursor1.

[3]Avanzamento dell'iteratore alla riga successiva. Il metodo cursor1.next() restituisce il valore Booleano `false` se non esistono più righe da richiamare.

[4]Spostamento di dati. Il metodo del programma di accesso denominato empno() restituisce il valore della colonna denominata empno sulla riga corrente. Il metodo di accesso denominato firstnme() restituisce il valore della colonna denominata firstnme sulla riga corrente.

[5]Dati SELECT in una variabile host. L'istruzione SELECT trasferisce il numero di righe presente in una tabella in una variabile host count1.

[6] Inizializzazione dell'iteratore. L'oggetto iteratore cursor2 viene inizializzato utilizzando il risultato di una query. La query memorizza il risultato in cursor2.

[7]Richiamo di dati. L'istruzione FETCH restituisce il valore corrente della prima colonna dichiarata nel cursore ByPos dalla tabella dei risultati nella variabile host str2.

[8]Controllo della riuscita di un'istruzione FETCH.INTO. Il metodo endFetch() restituisce un valore Booleano true se l'iteratore non è posizionato su una riga, cioè se l'ultimo tentativo di selezionare una riga ha avuto esito negativo. Il metodo endFetch() restituisce false se l'ultimo tentativo di selezionare una riga ha avuto esito positivo. DB2 tenta di selezionare una riga quando viene richiamato il metodo next(). Un'istruzione FETCH...INTO chiama implicitamente il metodo next().

[9]Chiusura degli iteratori. Il metodo close() rilascia qualsiasi risorsa mantenuta dagli iteratori. È necessario chiudere in modo esplicito gli iteratori per assicurarsi che le risorse di sistema vengano rilasciate in modo tempestivo.

Compilazione ed esecuzione dei programmi SQLJ

Se il programma Java possiede istruzioni SQLJ incorporate, è necessario seguire una procedura speciale per sua compilazione ed esecuzione.

Se il programma Java possiede istruzioni SQLJ incorporate, è necessario seguire una procedura speciale per la compilazione e l'esecuzione.

1. Configurare il server per l'uso di SQLJ.
2. Utilizzare il programma di conversione SQLJ, `sqlj`, sul codice sorgente Java con l'SQL incorporato, per generare il codice sorgente Java e i profili associati. Esiste un profilo creato per ciascun collegamento.

   Ad esempio, immettere il comando seguente:

   ```
   sqlj MyClass.sqlj
   ```

   dove *MyClass.sqlj* rappresenta il nome del file SQLJ.

In questo esempio, il programma di conversione SQLJ crea un file codice sorgente MyClass.java e ogni profilo associato. I profili associati sono denominati MyClass_SJProfile0.ser, MyClass_SJProfile1.ser, MyClass_SJProfile2.ser e così via.

Nota: il programma di conversione SQLJ compila automaticamente il codice sorgente Java convertito in un file di classe, a meno che non venga esplicitamente disattivata l'opzione di compilazione con la clausola `-compile=false`.

3. Utilizzare lo strumento Programma di personalizzazione del profilo SQLJ,`db2profc`, per installare i Programmi di personalizzazione DB2 SQLJ sui profili generati e creare i pacchetti DB2 sul sistema locale.

   Ad esempio, immettere il comando:

   ```
   db2profc MyClass_SJProfile0.ser
   ```

   dove *MyClass_SJProfile0.ser* è il nome del profilo sul quale il programma di personalizzazione DB2 SQLJ è in esecuzione.

   Nota: questa fase è facoltativa ma è consigliata per migliorare le prestazioni del tempo di esecuzione.

4. Eseguire il file di classe Java come un qualsiasi altro file di classe Java.

   Ad esempio, immettere il comando:

   ```
   java MyClass
   ```

   dove *MyClass* è il nome del file di classe Java.

Concetti correlati

"Inserimento delle istruzioni SQL nell'applicazione Java" a pagina 185
Le istruzioni SQL statiche in SQLJ si trovano nelle clausole SQLJ. Le clausole SQLJ iniziano con `#sql` e terminano con un carattere punto e virgola (;).

Utilizzo delle routine SQL Java

Il sistema consente di accedere ai programmi Java da istruzioni e programmi SQL. È possibile effettuare ciò utilizzando le procedure memorizzate Java e le UDF (user-defined function - funzioni definite dall'utente) Java. System i5 supporta sia le convenzioni DB2 che quelle SQLJ per richiamare procedure memorizzate Java e UDF Java. Sia le procedure memorizzate Java che le UDF Java possono utilizzare classi Java memorizzate nei file JAR. System i5 utilizza le procedure memorizzate definite dallo standard *SQLJ Part 1* per registrare i file JAR con il database.

Utilizzo delle routine SQL Java

È possibile accedere ai programmi Java dai programmi e dalle istruzioni SQL. È possibile effettuare ciò utilizzando le procedure memorizzate Java e le UDF (user-defined function - funzioni definite dall'utente) Java.

Per utilizzare le routine SQL Java, completare le seguenti attività:

1. Abilitare SQLJ

   Poiché qualsiasi routine SQL Java può utilizzare SQLJ, rendere il supporto tempo di esecuzione SQLJ sempre disponibile quando si utilizza J2SE (Java 2 Platform, Standard Edition). Per abilitare il supporto tempo di esecuzione per SQLJ in J2SE, aggiungere un collegamento al file SQLJ runtime.zip dall'indirizzario delle estensioni. Per ulteriori informazioni, consultare l'argomento relativo alla configurazione del sistema per utilizzare SQLJ.

2. Scrivere i metodi Java relativi alle routine

   Una routine SQL Java elabora un metodo Java da SQL. Questo metodo deve essere scritto utilizzando le convenzioni di inoltro dei parametri DB2 for i5/OS o SQLJ. Consultare Procedure memorizzate Java, Funzioni Java definite dall'utente e Funzioni della tabella Java definite dall'utente per ulteriori informazioni sulla codifica di un metodo utilizzato da una routine SQL Java.

3. Compilare le classi Java

   È possibile compilare le routine SQL Java scritte utilizzando lo stile di parametro Java senza alcuna impostazione aggiuntiva. Tuttavia, è necessario che le routine SQL Java che utilizzano lo stile di parametro DB2GENERAL estendano la classe com.ibm.db2.app.UDF o la classe com.ibm.db2.app.StoredProc. Tali classi sono contenute nel file JAR, /QIBM/ProdData/Java400/ext/db2routines_classes.jar. Quando si utilizza javac per compilare queste routine, è indispensabile che questo file JAR esista nel CLASSPATH. Ad esempio, il seguente comando compila un file di origine Java che contiene una routine che utilizza lo stile di parametro DB2GENERAL:

   ```
   javac -DCLASSPATH=/QIBM/ProdData/Java400/ext/db2routines_classes.jar
   source.java
   ```

4. Rendere le classi compilate accessibili alla JVM (Java virtual machine) utilizzata dal database

   Le classi definite dall'utente utilizzate dalla JVM del database possono trovarsi nell'indirizzario /QIBM/UserData/OS400/SQLLib/Function o in un file JAR registrato nel database.

   /QIBM/UserData/OS400/SQLLib/Function è l'equivalente System i5 di /sqllib/function, l'indirizzario dove DB2 for i5/OS memorizza le procedure memorizzate Java e le UDF Java su altre piattaforme. Se la classe fa parte di un pacchetto Java, è necessario che essa si trovi nel sottoindirizzario appropriato. Ad esempio, se la classe runit viene creata come parte del pacchetto foo.bar, sarebbe opportuno che il file runnit.class si trovi nell'indirizzario dell'IFS (Integrated File System), /QIBM/ProdData/OS400/SQLLib/Function/foo/bar.

   È inoltre possibile inserire il file di classe in un file JAR che viene registrato nel database. Il file JAR viene registrato utilizzando la procedura memorizzata SQLJ.INSTALL_JAR. Tale procedura viene utilizzata per assegnare un ID JAR ad un file JAR. Tale ID JAR viene utilizzato per identificare il file JAR in cui si trova il file di classe. Consultare Procedure SQLJ che manipolano i file JAR per ulteriori informazioni su SQLJ.INSTALL_JAR e su altre procedure memorizzate per gestire i file JAR.

5. Registrare la routine con il database.

   Le routine SQL Java vengono registrate con il database utilizzando le istruzioni SQL CREATE PROCEDURE e CREATE FUNCTION. Tali istruzioni contengono i seguenti elementi:

   Parole chiave CREATE
   Le istruzioni SQL per creare una routine SQL Java iniziano con CREATE PROCEDURE o CREATE STATEMENT.

   Nome della routine
   L'istruzione SQL identifica il nome della routine nota al database. Questo è il nome utilizzato per accedere alla routine Java da SQL.

   Parametri e valore di ritorno
   L'istruzione SQL identifica i parametri e i valori di ritorno, se applicabili, per la routine Java.

   LANGUAGE JAVA
   L'istruzione SQL utilizza le parole chiave LANGUAGE JAVA per indicare che la routine è stata scritta in Java.

   PAROLE CHIAVE PARAMETER STYLE
   L'istruzione SQL identifica lo stile di parametro utilizzando le parole chiave PARAMETER STYLE JAVA o PARAMETER STYLE DB2GENERAL.

   Nome esterno
   L'istruzione SQL identifica il metodo Java da elaborare come routine SQL Java. Il nome esterno ha uno dei due formati:

   - Se il metodo è in un file classe ubicato nell'indirizzario /QIBM/UserData/OS400/SQLLib/Function, il metodo viene identificato utilizzando il formato *classname.methodname*, dove *classname* è il nome completo della classe e *methodname* è il nome del metodo.
   - Se il metodo è nel file JAR registrato nel database, esso viene identificato utilizzando il formato *jarid*:*classname.methodname*, dove *jarid* è l'ID JAR del file JAR registrato, *classname* è il nome classe e *methodname* è il nome del metodo.

È possibile utilizzare System i Navigator per creare una procedura memorizzata o una funzione definita dall'utente che utilizza lo stile di parametro Java.

6. Utilizzare la procedura Java

   Una procedura memorizzata Java viene chiamata utilizzando l'istruzione SQL CALL. Un'UDF Java è una funzione chiamata come parte di un'altra istruzione SQL.

   "Configurazione del system per l'utilizzo di SQLJ"
   Prima di eseguire un programma Java che contiene istruzioni SQLJ incorporate, assicurarsi di configurare il server all'utilizzo di SQLJ. Il supporto SQLJ richiede di modificare la variabile di ambiente CLASSPATH per il server.

   "Procedure memorizzate Java" a pagina 193
   Quando si utilizza Java per scrivere procedure memorizzate, è possibile utilizzare due stili di inoltro dei parametri.

   "Funzioni scalari Java definite dall'utente" a pagina 197
   Una funzione scalare Java restituisce un valore da un programma Java al database. Ad esempio, è stato possibile creare una funzione scalare che restituisce la somma di due numeri.

   "Funzioni della tabella Java definite dall'utente" a pagina 202
   DB2 permette a una funzione di restituire una tabella. Questa possibilità risulta utile per presentare informazioni dall'esterno del database al database in formato tabella. Ad esempio, è possibile creare una tabella che presenta la serie di proprietà nella JVM (Java virtual machine) utilizzata per le procedure memorizzate Java e le UDF Java (sia tabella che scalare).

   "Procedure SQLJ che manipolano i file JAR" a pagina 203
   Sia le procedure memorizzate Java che le UDF Java possono utilizzare le classi Java memorizzate in file JAR Java.

Configurazione del system per l'utilizzo di SQLJ:

Prima di eseguire un programma Java che contiene istruzioni SQLJ incorporate, assicurarsi di configurare il server all'utilizzo di SQLJ. Il supporto SQLJ richiede di modificare la variabile di ambiente CLASSPATH per il server.

Per ulteriori informazioni sulla gestione dei classpath Java, consultare la pagina che segue:

Classpath Java

Utilizzo di SQLJ e J2SE

Per configurare SQLJ su un server su cui è in esecuzione una qualsiasi versione supportata di J2SE, completare queste istruzioni:

1. Aggiungere i seguenti file alla variabile di ambiente CLASSPATH per il server:
   - /QIBM/ProdData/Os400/Java400/ext/sqlj_classes.jar
   - /QIBM/ProdData/Os400/Java400/ext/translator.zip

   Nota: è necessario aggiungere translator.zip solo se si desidera eseguire il programma di conversione SQLJ (comando sqlj). Non è necessario aggiungere translator.zip se si desidera solo eseguire i programmi Java compilati che utilizzano SQLJ. Per ulteriori informazioni, consultare l'argomento relativo al programma di conversione SQLJ (sqlj)
2. Su una richiesta comandi i5/OS, utilizzare il seguente comando per aggiungere un collegamento a runtime.zip dall'indirizzario estensioni. Immettere il comando su una sola riga e premere **Invio**.

   ```
   ADDLNK OBJ('/QIBM/ProdData/Os400/Java400/ext/runtime.zip')
   NEWLNK('/QIBM/UserData/Java400/ext/runtime.zip')
   ```

   Per ulteriori informazioni sull'installazione delle estensioni, consultare la seguente pagina:

Procedure memorizzate Java

Quando si utilizza Java per scrivere procedure memorizzate, è possibile utilizzare due stili di inoltro dei parametri.

Lo stile consigliato è lo stile di parametro JAVA, che corrisponde allo stile di parametro specificato in SQLj: standard delle routine SQL. Il secondo stile, DB2GENERAL, è uno stile di parametro definito da UDB DB2. Lo stile di parametro determina inoltre le convenzioni che è necessario utilizzare quando si codifica una procedura memorizzata Java.

È inoltre opportuno essere a conoscenza di alcune limitazioni esistenti sulle procedure memorizzate Java.

Stile del parametro JAVA:

Quando si codifica una procedura memorizzata Java che utilizza lo stile del parametro Java, è necessario utilizzare le seguenti convenzioni.

- Il metodo Java deve essere un metodo static (non di istanza) public void.
- I parametri del metodo Java devono essere tipi compatibili con SQL.
- È possibile che un metodo Java verifichi un valore NULL SQL quando il parametro è di tipo a capacità null (come String).
- I parametri di emissione vengono restituiti utilizzando schiere ad elemento singolo.
- Il metodo Java può accedere al database corrente utilizzando il metodo getConnection.

Le procedure di memorizzazione Java che utilizzano lo stile del parametro JAVA sono metodi public static. All'interno delle classi, le procedure memorizzate vengono identificate dalla relativa firma e nome metodo. Quando si chiama una procedura memorizzata, la relativa firma viene generata in modo dinamico, in base ai tipi di variabile definiti dall'istruzione CREATE PROCEDURE.

Se un parametro viene passato in un tipo Java che consente il valore null, è possibile che un metodo Java confronti il parametro con null per stabilire se un parametro di immissione è un NULL SQL.

I seguenti tipi Java non supportano il valore null:

- short
- int
- long
- float
- double

Se un valore null viene passato in un tipo Java che non supporta il valore null, verrà restituita un'SQL exception con il codice di errore -20205.

I parametri di emissione vengono passati come schiere che contengono un solo elemento. La procedura memorizzata Java può impostare il primo elemento della schiera per impostare il parametro di emissione.

Si accede ad un collegamento nel contesto dell'applicazione di incorporazione utilizzando la seguente chiamata JDBC (Java Database Connectivity):

```
connection=DriverManager.getConnection("jdbc:default:connection");
```

Questo collegamento esegue, quindi, istruzioni SQL con API JDBC.

Segue una breve procedura memorizzata con un'immissione e due emissioni. Questa procedura esegue la query SQL specificata e restituisce sia il numero di righe nel risultato che SQLSTATE.

Esempio: procedura memorizzata con un'immissione e due emissioni

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
package mystuff;

import java.sql.*;
public class sample2 {
  public static void donut(String query, int[] rowCount,
    String[] sqlstate) throws Exception {
  try {
    Connection c=DriverManager.getConnection("jdbc:default:connection");
    Statement s=c.createStatement();
    ResultSet r=s.executeQuery(query);
    int counter=0;
    while(r.next()){
      counter++;
    }
    r.close(); s.close();
    rowCount[0] = counter;
    }catch(SQLException x){
    sqlstate[0]= x.getSQLState();
   }
  }
 }
```

Nello standard SQLj, per restituire una serie di risultati nelle routine che utilizzano lo stile del parametro JAVA, è necessario impostare esplicitamente la serie dei risultati. Quando viene creata una procedura che restituisce serie di risultati, vengono aggiunti ulteriori parametri della serie di risultati alla fine dell'elenco di parametri. Ad esempio, l'istruzione

```
CREATE PROCEDURE RETURNTWO()
DYNAMIC RESULT SETS 2
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'javaClass!returnTwoResultSets'
```

chiama un metodo Java con la firma `public static void returnTwoResultSets(ResultSet[] rs1, ResultSet[] rs2)`.

È necessario impostare i parametri di emissione delle serie di risultati come dimostrato nel seguente esempio. Come nello stile DB2GENERAL, sarebbe opportuno non chiudere le serie di risultati e le istruzioni corrispondenti.

Esempio: procedura memorizzata che restituisce due serie di risultati

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
public class javaClass {
  /**
    * Procedura Java memorizzata, con parametri stile JAVA,
    * la quale elabora stringhe predefinite
    * e restituisce due serie di risultati.
    *
    * @param ResultSet[] rs1    primo ResultSet
    * @param ResultSet[] rs2    secondo ResultSet
    */
public static void returnTwoResultSets (ResultSet[] rs1, ResultSet[] rs2) throws Exception
{
  // ottenere il collegamento del chiamante al database; ereditato da StoredProc
  Connection con = DriverManager.getConnection("jdbc:default:connection");
```

```
  //definire ed elaborare la prima istruzione di selezione
  Statement stmt1 = con.createStatement();
  String sql1 = "select value from table01 where index=1";
  rs1[0] = stmt1.executeQuery(sql1);

  //definire ed elaborare la seconda istruzione di selezione
  Statement stmt2 = con.createStatement();
  Stringsql2 = "select value from table01 where index=2";
  rs2[0] = stmt2.executeQuery(sql2);
  }
}
```

Sul server, i parametri aggiuntivi della serie di risultati non vengono esaminati per stabilire l'ordine delle serie di risultati. Tali serie sul server vengono restituite nell'ordine in cui sono state aperte. Per assicurare la compatibilità con lo standard SQLj, sarebbe opportuno assegnare i risultati nell'ordine in cui vengono aperti, come mostrato precedentemente.

Stile del parametro DB2GENERAL:

Quando si codifica una procedure memorizzata Java che utilizza lo stile di parametro DB2GENERAL è necessario utilizzare le seguenti convenzioni.

- La classe che definisce una procedura memorizzata Java deve *essere un'estensione*, o una sottoclasse della classe com.ibm.db2.app.StoredProc Java.
- Il metodo Java deve essere un metodo di istanza public void.
- I parametri del metodo Java devono essere tipi compatibili con SQL.
- È possibile che un metodo Java verifichi un valore NULL SQL utilizzando il metodo isNull.
- Il metodo Java deve impostare esplicitamente i parametri di ritorno utilizzando il metodo set.
- Il metodo Java può accedere al database corrente utilizzando il metodo getConnection.

Una classe che include una procedura memorizzata Java deve essere un'estensione della classe com.ibm.db2.app.StoredProc. Le procedure memorizzate Java sono metodi di istanza public. All'interno delle classi, le procedure memorizzate vengono identificate dalla relativa firma e nome metodo. Quando si chiama una procedura memorizzata, la relativa firma viene generata in modo dinamico, in base ai tipi di variabile definiti dall'istruzione CREATE PROCEDURE.

La classe com.ibm.db2.app.StoredProc fornisce il metodo isNull, che consente a un metodo Java di stabilire se un parametro di immissione è un NULL SQL. La classe com.ibm.db2.app.StoredProc fornisce inoltre i metodi set...( ) che impostano i parametri di emissione. È necessario utilizzare questi metodi per impostare i parametri di emissione. Se non si imposta un parametro di emissione, tale parametro restituisce il valore NULL SQL.

La classe com.ibm.db2.app.StoredProc fornisce la seguente routine per selezionare un collegamento JDBC nel contesto dell'applicazione di incorporazione. Si accede ad un collegamento nel contesto dell'applicazione di incorporazione utilizzando la seguente chiamata JDBC:

```
public Java.sql.Connection getConnection( )
```

Questo collegamento esegue, quindi, istruzioni SQL con API JDBC.

Segue una breve procedura memorizzata con un'immissione e due emissioni. Questa procedura elabora la query SQL specificata e restituisce sia il numero di righe nel risultato che SQLSTATE.

Esempio: procedura memorizzata con un'immissione e due emissioni

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
package mystuff;

import com.ibm.db2.app.*;
import java.sql.*;
public class sample2 extends StoredProc {
  public void donut(String query, int rowCount,
    String sqlstate) throws Exception {
  try {
    Statement s=getConnection().createStatement();
    ResultSet r=s.executeQuery(query);
    int counter=0;
    while(r.next()){
      counter++;
    }
    r.close(); s.close();
    set(2, counter);
    }catch(SQLException x){
    set(3, x.getSQLState());
    }
  }
}
```

Per restituire una serie dei risultati in procedure che utilizzano lo stile del parametro DB2GENERAL, è necessario lasciare aperte le serie dei risultati e l'istruzione corrispondente alla fine della procedura. La serie dei risultati che viene restituita deve essere chiusa dall'applicazione client. Se vengono restituite più serie dei risultati, il loro ordine è lo stesso di quello in cui sono state aperte. Ad esempio, la seguente procedura memorizzata restituisce due serie dei risultati.

Esempio: procedura memorizzata che restituisce due serie dei risultati

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
public void returnTwoResultSets() throws Exception
{
  // ottenere il collegamento del chiamante al database; ereditato da StoredProc
  Connection con = getConnection ();
  Statement stmt1 = con.createStatement ();
  String sql1 = "select value from table01 where index=1";
  ResultSet rs1 = stmt1.executeQuery(sql1);
  Statement stmt2 = con.createStatement();
  String sql2 = "select value from table01 where index=2";
  ResultSet rs2 = stmt2.executeQuery(sql2);
}
```

Limitazioni sulle procedure memorizzate Java:

Tali limitazioni si applicano alle procedure memorizzate Java.

- Una procedura memorizzata Java non dovrebbe creare sottoprocessi aggiuntivi. È possibile creare un sottoprocesso aggiuntivo in un lavoro se il lavoro è capace di supportare più sottoprocessi. Dal momento che non esiste alcuna garanzia che un lavoro che chiama una procedura memorizzata SQL sia capace di supportare più sottoprocessi, una procedura memorizzata Java non dovrebbe creare sottoprocessi aggiuntivi.
- Non è possibile utilizzare un'autorizzazione adottata per accedere ai file di classe Java.
- Una procedura memorizzata Java utilizza la stessa versione predefinita di JDK del comando `java`. Se necessario, la versione di JDK utilizzata da una procedura memorizzata Java può essere modificata utilizzando un file SystemDefault.properties.
- Dal momento che le classi Blob e Clob si trovano in entrambi i pacchetti java.sql e com.ibm.db2.app, è necessario che il programmatore utilizzi l'intero nome di queste classi, se tutte e due vengono utilizzate nel programma. Il programma deve assicurare che le classi Blob e Clob da com.ibm.db2.app vengano utilizzate come parametri passati alla procedura memorizzata.

- Quando viene creata una procedura memorizzata Java, il sistema genera un programma di servizio nella libreria. Tale programma viene utilizzato per memorizzare la definizione della procedura. Il programma ha un nome generato dal sistema. È possibile ottenere tale nome esaminando la registrazione lavori che ha creato la procedura memorizzata. Se l'oggetto del programma viene salvato e quindi ripristinato, la definizione della procedura viene ripristinata. Se è necessario spostare una procedura memorizzata Java da un sistema a un altro, l'utente è responsabile di spostare il programma che contiene la definizione di procedura così come il file dell'IFS (integrated file system), che contiene la classe Java.
- Una procedura memorizzata Java non può impostare le proprietà (ad esempio, la denominazione di sistema) del collegamento JDBC utilizzato per collegarsi al database. Le proprietà di collegamento JDBC predefinite vengono utilizzate sempre, tranne quando il prefetch è disabilitato.

Funzioni scalari Java definite dall'utente

Una funzione scalare Java restituisce un valore da un programma Java al database. Ad esempio, è stato possibile creare una funzione scalare che restituisce la somma di due numeri.

Come altre procedure memorizzate Java le funzioni scalari Java utilizzano uno dei due stili di parametro, Java e DB2GENERAL. Quando si codifica un'UDF (user-defined function) Java, è necessario conoscere le limitazioni inserite sulla creazione delle funzioni scalari Java.

Stile di parametro Java

Lo stile di parametro Java viene specificato dallo standard *SQLJ Parte 1: routine SQL*. Quando si codifica un'UDF Java, utilizzare le seguenti convenzioni.

- Il metodo Java deve essere un metodo public static.
- Il metodo Java deve restituire un tipo compatibile a SQL. Il valore di ritorno è il risultato del metodo.
- I parametri del metodo Java devono essere tipi compatibili con SQL.
- È possibile che un metodo Java verifichi un valore NULL SQL per tipi Java che consentono il valore null.

Ad esempio, data un'UDF denominata sample!test3 che restituisce INTEGER e acquisisce argomenti di tipo CHAR(5), BLOB(10K) e DATE, DB2 prevede che l'implementazione Java dell'UDF abbia la seguente firma:

```
import com.ibm.db2.app.*;
public class sample {
   public static int test3(String arg1, Blob arg2, Date arg3) { ... }
}
```

I parametri del metodo Java devono essere tipi compatibili con SQL. Ad esempio se si dichiara che un'UDF acquisisce argomenti di tipi SQL t1, t2 e t3 e il tipo di ritorno t4, essa viene chiamata come un metodo Javacon la firma Java prevista:

```
public static T4 name (T1 a, T2 b, T3 c)  { .....}
```

dove:

- *name* è il nome del metodo
- I valori da T1 a T4 sono i tipi Java che corrispondono ai tipi SQL da t1 a t4.
- *a*, *b* e *c* sono nomi variabile arbitrari per gli argomenti di immissione.

La correlazione tra tipi SQL e tipi Java si trova in Convenzioni per inoltrare un parametro per le procedure memorizzate e le UDF.

I valori NULL SQL sono rappresentati dalle variabiliJava non inizializzate. Tali variabili possiedono un valore null Java se sono tipi di oggetti. Se un NULL SQL viene passato ad un tipo di dati scalare Java, come int, si verifica una condizione di eccezione.

Per restituire un risultato da un'UDF Java quando si utilizza lo stile di parametro JAVA, restituire semplicemente il risultato dal metodo.

```
{ ....
  return value;
}
```

Come i moduli C utilizzati nelle UDF e nelle procedure memorizzate, non è possibile utilizzare i flussi I/E standard Java (System.in, System.out e System.err) nelle UDF Java.

Stile di parametro DB2GENERAL

Lo stile di parametro DB2GENERAL è utilizzato dalle UDFJava. In questo stile, il valore di ritorno viene passato come l'ultimo parametro della funzione ed è necessario impostarlo utilizzando un metodo *set* della classe com.ibm.db2.app.UDF.

Quando si codifica un'UDF Java, è necessario seguire queste convenzioni:

- La classe, che include l'UDF Java, deve *essere un'estensione* o una sottoclasse della classe com.ibm.db2.app.UDF Java.
- Per lo stile di parametro DB2GENERAL, il metodo Java deve essere un metodo di istanza public void.
- I parametri del metodo Java devono essere tipi compatibili con SQL.
- È possibile che il metodo Java verifichi un valore NULL SQL utilizzando il metodo isNull.
- Per lo stile di parametro DB2GENERAL, il metodo Java deve impostare esplicitamente i parametri di ritorno utilizzando il metodo set().

Una classe che include un'UDF Java deve essere un'estensione della classe com.ibm.db2.app.UDF Java. Un'UDF Java che utilizza lo stile di parametro DB2GENERAL deve essere un metodo di istanza void della classe Java. Ad esempio, per un'UDF denominata sample!test3 che restituisce INTEGER e acquisisce argomenti di tipo CHAR(5), BLOB(10K) e DATE, DB2 si prevede che l'implementazione Java dell'UDF abbia la seguente firma:

```
import com.ibm.db2.app.*;
public class sample extends UDF {
   public void test3(String arg1, Blob arg2, String arg3, int result) { ... }
}
```

I parametri di un metodo Java devono essere tipi compatibili con SQL. Ad esempio se si dichiara che un'UDF acquisisce argomenti di tipi SQL t1, t2 e t3 e il tipo di ritorno t4, essa viene chiamata come un metodo Java con la firma Java prevista:

```
public void name (T1 a, T2 b, T3 c, T4 d)  { .....}
```

dove:

- *name* è il nome del metodo
- I valori da T1 a T4 sono i tipi Java che corrispondono ai tipi SQL da t1 a t4.
- *a*, *b* e *c* sono nomi variabile arbitrari per gli argomenti di immissione.
- *d* è un nome variabile arbitrario che rappresenta il risultato calcolato dell'UDF.

La correlazione tra tipi SQL e tipi Java è fornita nella sezione Convenzioni per inoltrare un parametro per le procedure memorizzate e le UDF.

I valori NULL SQL sono rappresentati dalle variabiliJava non inizializzate. Queste variabili hanno un valore di zero se sono tipi primitivi e un valore null Java se sono tipi di oggetti, secondo le regole Java. Per distinguere NULL SQL da uno zero ordinario, è possibile chiamare il metodo isNull per qualsiasi argomento di immissione:

```
{ ....
  if (isNull(1)) { /* argument #1 was a SQL NULL */ }
  else           { /* not NULL */ }
}
```

Nel precedente esempio, i numeri dell'argomento iniziano a uno. La funzione isNull(), come le altre funzioni che seguono, viene ricevuta dalla classe com.ibm.db2.app.UDF. Per restituire un risultato da un'UDF Java quando si utilizza lo stile di parametro DB2GENERAL, utilizzare il metodo set() nell'UDF, come nel seguente esempio:

```
{ ....
  set(2, value);
}
```

Dove 2 è l'indice di un argomento di emissione e *value* è una costante letterale o variabile di un tipo compatibile. Il numero dell'argomento è l'indice nell'elenco argomenti dell'emissione selezionata. Nel primo esempio in questa sezione, la variabile del risultato int ha un indice di 4. Un argomento di emissione che non viene impostato prima che l'UDF venga restituita ha un valore NULL.

Come i moduli C utilizzati nelle UDF e nelle procedure memorizzate, non è possibile utilizzare i flussi I/E standard Java (System.in, System.out e System.err) nelle UDF Java.

Normalmente DB2 chiama un'UDF varie volte, una per ogni riga di un'immissione o una serie di risultati in una query. Se viene specificato SCRATCHPAD nell'istruzione CREATE FUNCTION dell'UDF, DB2 riconosce che è necessaria una certa "continuità" tra i richiami successivi dell'UDF e, quindi, per le funzioni dello stile di parametro DB2GENERAL, la classe Java di implementazione non viene dotata di istanze per ogni chiamata, ma, generalmente parlando, una volta per ogni riferimento UDF per istruzione. Tuttavia, se viene specificato NO SCRATCHPAD per un'UDF, viene emessa un'istanza di ripulitura per ogni chiamata all'UDF, tramite una chiamata al programma di creazione della classe.

Potrebbe essere utile uno scratchpad per salvare le informazioni attraverso le chiamate ad un'UDF. È possibile che le UDF Java utilizzino variabili di istanze o impostino lo scratchpad per ottenere continuità tra le chiamate. Le UDF Java accedono allo scratchpad con i metodi getScratchPad e setScratchPad disponibili in com.ibm.db2.app.UDF. Alla fine di una query, se si specifica l'opzione FINAL CALL sull'istruzione CREATE FUNCTION, viene chiamato il metodo public void close() dell'oggetto (per funzioni dello stile di parametro DB2GENERAL). Se non si definisce questo metodo, subentra una funzione stub e l'evento viene ignorato. La classe com.ibm.db2.app.UDF contiene variabili e metodi utili che è possibile utilizzare all'interno di un'UDF dello stile di parametro DB2GENERAL. Tali variabili e metodi vengono spiegati nella seguente tabella.

Variabili e Metodi	**Descrizione**
• public static final int SQLUDF_FIRST_CALL = -1; • public static final int SQLUDF_NORMAL_CALL = 0; • public static final int SQLUDF_TF_FIRST = -2; • public static final int SQLUDF_TF_OPEN = -1; • public static final int SQLUDF_TF_FETCH = 0; • public static final int SQLUDF_TF_CLOSE = 1; • public static final int SQLUDF_TF_FINAL = 2;	Per le UDF scalari, queste sono le costanti per determinare se viene effettuata una chiamata first o normal. Per le UDF della tabella, queste sono costanti per determinare se viene effettuata una chiamata first, open, fetch, close o final.
public Connection getConnection();	Il metodo ottiene l'handle del collegamento JDBC per questa chiamata alla procedura memorizzata e restituisce l'oggetto JDBC che rappresenta il collegamento dell'applicazione della chiamata al database. Esso è analogo al risultato di una chiamata SQLConnect() null in una procedura memorizzata C.

Variabili e Metodi	Descrizione
public void close();	Questo metodo viene chiamato dal database alla fine di una valutazione UDF, se l'UDF è stata creata con l'opzione FINAL CALL. Esso è analogo alla chiamata finale per un'UDF C. Se una classe UDF Java non implementa questo metodo, questo evento viene ignorato.
public boolean isNull(int i)	Questo metodo verifica se un argomento di immissione con l'indice assegnato è un NULL SQL.
• public void set(int i, short s); • public void set(int i, int j); • public void set(int i, long j); • public void set(int i, double d); • public void set(int i, float f); • public void set(int i, BigDecimal bigDecimal); • public void set(int i, String string); • public void set(int i, Blob blob); • public void set(int i, Clob clob); • public boolean needToSet(int i);	Questi metodi impostano un argomento di emissione al valore assegnato. Viene emessa un'eccezione se si verifica qualcosa di sbagliato, incluse le seguenti situazioni: • La chiamata UDF non è in corso • L'indice non fa riferimento ad un valido argomento di emissione • I tipi di dati non corrispondono • La lunghezza dei dati non corrisponde • Si verifica un errore di conversione della codepage
public void setSQLstate(String string);	È possibile chiamare questo metodo da un'UDF per impostare SQLSTATE da restituire dalla chiamata. Se la stringa non è accettabile come SQLSTATE, viene emessa un'eccezione. L'utente può impostare SQLSTATE nel programma esterno per restituire un errore o un'avvertenza dalla funzione. In questo caso, è necessario che SQLSTATE contenga uno dei seguenti elementi: • '00000' per indicare l'esito positivo • '01Hxx', dove xx è qualsiasi carattere formato da lettere maiuscole o a due cifre, per indicare un'avvertenza • '38yxx', dove y è una lettera maiuscola tra 'I' e 'Z' e xx è qualsiasi carattere formato da lettere maiuscole o a due cifre, per indicare un errore
public void setSQLmessage(String string);	Questo metodo è simile al metodo setSQLstate. Esso imposta il risultato del messaggio SQL. Se la stringa non è accettabile (ad esempio, più lunga di 70 caratteri), viene emessa un'eccezione.
public String getFunctionName();	Questo metodo restituisce il nome dell'UDF in elaborazione.
public String getSpecificName();	Questo metodo restituisce il nome specifico dell'UDF in elaborazione.
public byte[] getDBinfo();	Questo metodo restituisce una struttura DBINFO non elaborata per l'UDF in elaborazione, come una schiera di byte. È necessario che l'UDF sia stata registrata (utilizzando CREATE FUNCTION) con l'opzione DBINFO.

Variabili e Metodi	Descrizione
• public String getDBname(); • public String getDBauthid(); • public String getDBver_rel(); • public String getDBplatform(); • public String getDBapplid(); • public String getDBapplid(); • public String getDBtbschema(); • public String getDBtbname(); • public String getDBcolname();	Questi metodi restituiscono il valore del campo appropriato dalla struttura DBINFO dell'UDF in elaborazione. È necessario che l'UDF sia stata registrata (utilizzando CREATE FUNCTION) con l'opzione DBINFO. I metodi getDBtbschema(), getDBtbname() e getDBcolname() restituiscono informazioni significative solo se una funzione definita dall'utente viene specificata nella parte destra di una clausola SET in un'istruzione UPDATE.
public int getCCSID();	Questo metodo restituisce il CCSID del lavoro.
public byte[] getScratchpad();	Questo metodo restituisce una copia dello scratchpad dell'UDF correntemente in elaborazione. È necessario innanzitutto dichiarare l'UDF con l'opzione SCRATCHPAD.
public void setScratchpad(byte ab[]);	Questo metodo sostituisce lo scratchpad dell'UDF attualmente in elaborazione con il contenuto della schiera di byte fornita. È necessario innanzitutto dichiarare l'UDF con l'opzione SCRATCHPAD. È indispensabile che la schiera di byte abbia la stessa dimensione di quella restituita da getScratchpad().
public int getCallType();	Questo metodo restituisce il tipo di chiamata che viene correntemente effettuata. Questi valori corrispondono ai valori C definiti in sqludf.h. Il seguente elenco include i possibili valori di ritorno: • SQLUDF_FIRST_CALL • SQLUDF_NORMAL_CALL • SQLUDF_TF_FIRST • SQLUDF_TF_OPEN • SQLUDF_TF_FETCH • SQLUDF_TF_CLOSE • SQLUDF_TF_FINAL

Limitazioni sulle funzioni Java definite dall'utente:

Queste limitazioni si applicano alle UDF (user-defined function/funzioni definite dall'utente) Java.

- Una UDF Java non dovrebbe creare sottoprocessi aggiuntivi. È possibile creare un sottoprocesso aggiuntivo in un lavoro se il lavoro è capace di supportare più sottoprocessi. Dal momento che non esiste alcuna garanzia che un lavoro che richiama una procedura memorizzata SQL sia in grado di supportare più sottoprocessi, una procedura memorizzata Java non dovrebbe creare sottoprocessi aggiuntivi.
- Il nome completo della procedura memorizzata Java definita nel database è limitato a 279 caratteri. Tale limite è una conseguenza della colonna EXTERNAL_NAME, che ha una lunghezza massima di 279 caratteri.
- Non è possibile utilizzare un'autorizzazione adottata per accedere ai file di classeJava.
- Una UDF Java utilizza sempre l'ultima versione del JDK che è installato sul sistema.
- Dal momento che le classi Blob e Clob si trovano in entrambi i pacchetti java.sql e com.ibm.db2.app, è necessario che il programmatore utilizzi l'intero nome di queste classi, se tutte e due vengono utilizzate nel programma. Il programma deve assicurare che le classi Blob e Clob da com.ibm.db2.app vengano utilizzate come parametri passati alla procedura memorizzata.

- Come le funzioni origine, quando viene creata un'UDF Java, viene utilizzato un programma di servizio nella libreria per memorizzare la definizione della funzione. Il nome del programma di servizio viene generato dal sistema ed è possibile trovarlo nella registrazione lavori del lavoro che ha creato la funzione. Se questo oggetto viene salvato e ripristinato in un altro sistema, la definizione della funzione viene ripristinata. Se è necessario spostare un'UDF Java da un sistema ad un altro, l'utente è responsabile di spostare il programma di servizio che contiene la definizione della funzione così come il file dell'IFS (Integrated File System) che contiene la classe Java.
- Una UDF Java non può impostare le proprietà (ad esempio, la denominazione di sistema) del collegamento JDBC utilizzato per collegarsi al database. Le proprietà di collegamento JDBC predefinite vengono utilizzate sempre, tranne quando il prefetch è disabilitato.

Funzioni della tabella Java definite dall'utente:

DB2 permette a una funzione di restituire una tabella. Questa possibilità risulta utile per presentare informazioni dall'esterno del database al database in formato tabella. Ad esempio, è possibile creare una tabella che presenta la serie di proprietà nella JVM (Java virtual machine) utilizzata per le procedure memorizzate Java e le UDF Java (sia tabella che scalare).

Lo standard *SQLJ Parte 1: routine SQL* supporta le funzioni della tabella. Di conseguenza, le funzioni della tabella sono disponibili soltanto utilizzando lo stile di parametro DB2GENERAL.

Vengono effettuate cinque differenti tipi di chiamate ad una funzione della tabella. La seguente tabella spiega tali chiamate. Esse presumono che è stato specificato lo scratchpad sull'istruzione SQL di creazione funzione.

Punto nella scansione	**NESSUNO SCRATCHPAD IN LINGUAGGIO JAVA DELLA CHIAMATA FINAL**	**SCRATCHPAD IN LINGUAGGIO JAVA DELLA CHIAMATA FINAL**
Prima della prima istruzione OPEN della funzione della tabella	Nessuna chiamata	Viene chiamato il programma di creazione della classe (indica un nuovo scratchpad). Il metodo UDF viene chiamato con la chiamata FIRST.
Ad ogni istruzione OPEN della funzione della tabella.	Viene chiamato il programma di creazione della classe (indica un nuovo scratchpad). Il metodo UDF viene chiamato con la chiamata OPEN.	Il metodo UDF viene chiamato con la chiamata OPEN.
Ad ogni istruzione FETCH per una nuova riga di dati della funzione della tabella.	Il metodo UDF viene chiamato con la chiamata FETCH.	Il metodo UDF viene chiamato con la chiamata FETCH.
Ad ogni istruzione CLOSE della funzione della tabella	Il metodo UDF viene chiamato con la chiamata CLOSE. Anche il metodo close(), se esiste, viene chiamato.	Il metodo UDF viene chiamato con la chiamata CLOSE.
Dopo l'ultima istruzione CLOSE della funzione della tabella.	Nessuna chiamata	Il metodo UDF viene chiamato con la chiamata FINAL. Anche il metodo close(), se esiste, viene chiamato.

Esempio: funzione della tabella Java

Il seguente esempio mostra una funzione della tabella Java che determina le proprietà impostate nella JVM utilizzata per eseguire la funzione della tabella Java definita dall'utente.

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import com.ibm.db2.app.*;
import java.util.*;

public class JVMProperties extends UDF {
 Enumeration propertyNames;
 Properties properties ;

 public void dump (String property, String value) throws Exception
 {
   int callType = getCallType();
   switch(callType) {
     case SQLUDF_TF_FIRST:
       break;
     case SQLUDF_TF_OPEN:
       properties = System.getProperties();
       propertyNames = properties.propertyNames();
       break;
     case SQLUDF_TF_FETCH:
       if (propertyNames.hasMoreElements()) {
           property = (String) propertyNames.nextElement();
           value = properties.getProperty(property);
           set(1, property);
           set(2, value);
       } else {
          setSQLstate("02000");
       }
       break;
     case SQLUDF_TF_CLOSE:
       break;
     case SQLUDF_TF_FINAL:
       break;
     default:
       throw new Exception("UNEXPECT call type of "+callType);
   }
 }
}
```

Una volta compilata la funzione della tabella e copiato il relativo file di classe in /QIBM/UserData/OS400/SQLLib/Function, è possibile registrare la funzione nel database utilizzando la seguente istruzione SQL.

```
create function properties()
returns table (property varchar(500), value varchar(500))
external name 'JVMProperties.dump' language java
parameter style db2general fenced no sql
disallow parallel scratchpad
```

Una volta registrata la funzione, è possibile utilizzarla come parte di un'istruzione SQL. Ad esempio, la seguente istruzione SELECT restituisce la tabella generata dalla relativa funzione.

```
SELECT * FROM TABLE(PROPERTIES())
```

Procedure SQLJ che manipolano i file JAR

Sia le procedure memorizzate Java che le UDF Java possono utilizzare le classi Java memorizzate in file JAR Java.

Per utilizzare un file JAR, è necessario associare un *jar-id* al file JAR. Il sistema fornisce procedure memorizzate nello schema SQLJ che consentono ai *jar-id* e ai file JAR di essere manipolati. Queste procedure consentono ai file JAR di essere installati, sostituiti ed eliminati. Esse inoltre forniscono la capacità di utilizzare e aggiornare i cataloghi SQL associati ai file JAR.

SQLJ.INSTALL_JAR:

La procedura memorizzata SQLJ.INSTALL_JAR installa un file JAR in un sistema database. È possibile utilizzare questo file JAR nelle istruzioni successive CREATE FUNCTION e CREATE PROCEDURE.

Autorizzazione

Il privilegio mantenuto dall'ID di autorizzazione dell'istruzione CALL deve includere almeno una delle seguenti autorizzazioni per le tabelle catalogo SYSJAROBJECTS e SYSJARCONTENTS:

- Le seguenti autorizzazioni di sistema:
  - I privilegi INSERT e SELECT per la tabella
  - L'autorizzazione di sistema *EXECUTE per la libreria QSYS2
- Autorizzazione di gestione

Il privilegio mantenuto dall'ID di autorizzazione dell'istruzione CALL deve possedere le seguenti autorizzazioni:

- Accesso *R (alla lettura) al file JAR specificato nel parametro *jar-url* installato.
- Accesso *RWX (alla scrittura, all'esecuzione e alla lettura) all'indirizzario nel quale il file JAR è stato installato. Questo indirizzario è /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*, dove *schema* rappresenta la schema di *jar-id*.

Non è possibile utilizzare l'autorizzazione adottata per queste autorizzazioni.

Sintassi SQL

```
>>-CALL--SQLJ.INSTALL_JAR-- (--'jar-url'--,--'jar-id'--,--deploy--)-->

>-------------------------------------------------------------><
```

Descrizione

jar-url L'URL contenente il file JAR da installare o sostituire. L'unico schema URL supportato è 'file:'.

jar-id L'identificativo JAR nel database da associare al file specificato da *jar-url*. *jar-id* utilizza la denominazione SQL e il file JAR viene installato nello schema o nella libreria specificata dal qualificatore implicito o esplicito.

deploy Valore utilizzato per descrivere la install_action del file del descrittore di disposizione. Se questo numero intero risulta un valore diverso da zero, allora le install_actions di un file del descrittore di disposizione devono essere eseguite alla fine della procedura install_jar. La versione corrente di DB2 for i5/OS supporta solo un valore di zero.

Note sull'utilizzo

Quando viene installato un file JAR, DB2 for i5/OS registra il file JAR nel catalogo di sistema SYSJAROBJECTS. Estrae anche i nomi dei file di classe Java dal file JAR e registra ciascuna classe nel catalogo di sistema SYSJARCONTENTS. DB2 for i5/OS copia il file JAR in un sottoindirizzario jar/schema dell'indirizzario /QIBM/UserData/OS400/SQLLib/Function. DB2 for i5/OS fornisce alla nuova copia del file JAR il nome indicato nella clausola *jar-id*. Un file JAR che è stato installato da DB2 for i5/OS in un sottoindirizzario di /QIBM/UserData/OS400/SQLLib/Function/jar non deve essere modificato. Al contrario, non è necessario utilizzare i comandi CALL SQLJ.REMOVE_JAR e CALL SQLJ.REPLACE_JAR SQL per eliminare o sostituire un file JAR installato.

Esempio

Il comando seguente è immesso da una sessione interattiva SQL.

```
CALL SQLJ.INSTALL_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar', 0)
```

Il file Proc.jar che si trova nell'indirizzario file:/home/db2inst/classes/ è installato in DB2 for i5/OS con il nome myproc_jar. I comandi successivi SQL che utilizzano il file Procedure.jar fanno riferimento ad esso con il nome myproc_jar.

SQLJ.REMOVE_JAR:

La procedura SQLJ.REMOVE_JAR memorizzata elimina un file JAR dal sistema del database.

Autorizzazione

Il privilegio mantenuto dall'ID di istruzione CALL deve includere almeno una delle seguenti autorizzazioni per le tabelle di catalogo SYSJARCONTENTS e SYSJAROBJECTS:

- Le seguenti autorizzazioni di sistema:
  - I privilegi SELECT e DELETE per la tabella
  - L'autorizzazione di sistema *EXECUTE per la libreria QSYS2
- Autorizzazione di gestione

Il privilegio mantenuto dall'ID di autorizzazione dell'istruzione CALL deve possedere la seguente autorizzazione:

- Autorizzazione *OBJMGT al file JAR eliminato. Il file JAR è denominato /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile.

Non è possibile utilizzare l'autorizzazione adottata per questa autorizzazione.

Sintassi

```
>>-CALL--SQLJ.REMOVE_JAR--(--'jar-id'--,--undeploy--)----------><
```

Descrizione

jar-id L'identificativo JAR del file JAR da eliminare dal database.

undeploy
Il valore utilizzato per descrivere la remove_action del file del descrittore di disposizione. Se questo numero intero risulta un valore diverso da zero, allora le remove_action di un file del descrittore di disposizione devono essere eseguite alla fine della procedura install_jar. La versione corrente di DB2 for i5/OS supporta solo un valore di zero.

Esempio

Il comando seguente è immesso da una sessione interattiva SQL:

```
CALL SQLJ.REMOVE_JAR('myProc_jar', 0)
```

Il file JAR myProc_jar viene eliminato dal database.

SQLJ.REPLACE_JAR:

La procedura SQLJ.REPLACE_JAR memorizzata sostituisce un file JAR nel sistema del database.

Autorizzazione

Il privilegio mantenuto dall'ID di autorizzazione dell'istruzione CALL deve includere almeno una delle seguenti autorizzazioni per le tabelle catalogo SYSJAROBJECTS e SYSJARCONTENTS:

- Le seguenti autorizzazioni di sistema:
  - I privilegi SELECT, INSERT e DELETE per la tabella
  - L'autorizzazione di sistema *EXECUTE per la libreria QSYS2

- Autorizzazione di gestione

Il privilegio mantenuto dall'ID di autorizzazione dell'istruzione CALL deve possedere le seguenti autorizzazioni:
- Accesso *R (alla lettura) al file JAR specificato dal parametro *jar-url* installato.
- Autorizzazione *OBJMGT al file JAR eliminato. Il file JAR è denominato /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile.

Non è possibile utilizzare l'autorizzazione adottata per queste autorizzazioni.

Sintassi

```
>>-CALL--SQLJ.REPLACE_JAR--(--'jar-url'--,--'jar-id'--)--------><
```

Descrizione

jar-url L'URL contenente il file JAR da sostituire. L'unico schema URL supportato è 'file:'.

jar-id L'identificativo JAR nel database da associare al file specificato da *jar-url*. *jar-id* utilizza la denominazione SQL e il file JAR viene installato nello schema o nella libreria specificata dal qualificatore implicito o esplicito.

Note sull'utilizzo

La procedura SQLJ.REPLACE_JAR memorizzata sostituisce un file JAR che è stato precedentemente installato nel database utilizzando SQLJ.INSTALL_JAR.

Esempio

Il comando seguente è immesso da una sessione interattiva SQL:

```
CALL SQLJ.REPLACE_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar')
```

Il file JAR corrente cui ha fatto riferimento il *jar-id* myproc_jar viene sostituito con il file Proc.jar ubicato nell'indirizzario file:/home/db2inst/classes/.

SQLJ.UPDATEJARINFO:

SQLJ.UPDATEJARINFO aggiorna la colonna CLASS_SOURCE della tabella catalogo SYSJARCONTENTS. Questa procedura non fa parte dello standard SQLJ, ma viene utilizzata dal programma di creazione della procedura memorizzata DB2 for i5/OS.

Autorizzazione

Il privilegio mantenuto dall'ID di autorizzazione dell'istruzione CALL deve includere almeno una delle seguenti autorizzazioni per la tabella di catalogo SYSJARCONTENTS:
- Le seguenti autorizzazioni di sistema:
  - I privilegi SELECT e UPDATEINSERT per la tabella
  - L'autorizzazione di sistema *EXECUTE per la libreria QSYS2
- Autorizzazione di gestione

È necessario che l'utente che esegue l'istruzione CALL possieda anche le seguenti autorizzazioni:
- Accesso *R (alla lettura) al file JAR specificato nel parametro *jar-url*. Accesso *R (alla lettura) al file JAR installato.
- Accesso *RWX (alla scrittura, all'esecuzione e alla lettura) all'indirizzario nel quale il file JAR è stato installato. Questo indirizzario è /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*, dove *schema* rappresenta la schema di *jar-id*.

Non è possibile utilizzare l'autorizzazione adottata per queste autorizzazioni.

Sintassi

```
>>-CALL--SQLJ.UPDATEJARINFO--(--'jar-id'--,--'class-id'--,--'jar-url'--)-->

>-------------------------------------------------------------><
```

Descrizione

jar-id L'identificativo JAR nel database che deve essere aggiornato.

class-id
Il nome di classe completo del pacchetto relativo alla classe da aggiornare.

jar-url L'URL contenente il file di classe con cui aggiornare il file JAR. L'unico schema URL supportato è 'file:'.

Esempio

Il comando seguente è immesso da una sessione interattiva SQL:

```
CALL SQLJ.UPDATEJARINFO('myproc_jar', 'mypackage.myclass',
                        'file:/home/user/mypackage/myclass.class')
```

Il file JAR associato a *jar-id* myproc_jar, viene aggiornato con una nuova versione della classe mypackage.myclass. La nuova versione della classe si ottiene dal file /home/user/mypackage/myclass.class.

SQLJ.RECOVERJAR:

La procedura SQLJ.RECOVERJAR prende il file JAR memorizzato nel catalogo SYSJAROBJECTS e lo ripristina sul file /QIBM/UserData/OS400/SQLLib/Function/jar/*jarschema*/*jar_id*.jar.

Autorizzazione

Il privilegio mantenuto dall'ID di autorizzazione dell'istruzione CALL deve includere almeno una delle seguenti autorizzazioni per la tabella di catalogo SYSJAROBJECTS:

- Le seguenti autorizzazioni di sistema:
  - I privilegi SELECT e UPDATEINSERT per la tabella
  - L'autorizzazione di sistema *EXECUTE per la libreria QSYS2
- Autorizzazione di gestione

È necessario che l'utente che esegue l'istruzione CALL possieda anche le seguenti autorizzazioni:

- Accesso *RWX (alla scrittura, all'esecuzione e alla lettura) all'indirizzario nel quale il file JAR è stato installato. Questo indirizzario è /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*, dove *schema* rappresenta la schema di *jar-id*.
- Autorizzazione *OBJMGT al file JAR eliminato. Il file JAR è denominato /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile.

Sintassi

```
>>-CALL--SQLJ.RECOVERJAR--(--'jar-id'--)---------------------><
```

Descrizione

jar-id L'identificativo JAR nel database che deve essere ripristinato.

Esempio

Il comando seguente è immesso da una sessione interattiva SQL:

```
CALL SQLJ.UPDATEJARINFO('myproc_jar')
```

Il file JAR associato a myproc_jar viene aggiornato con il contenuto preso dalla tabella SYSJARCONTENT. Il file viene copiato su /QIBM/UserData/OS400/SQLLib/Function/jar/jar_schema myproc_jar.jar.

SQLJ.REFRESH_CLASSES:

La procedura memorizzata SQLJ.REFRESH_CLASSES provoca il ricaricamento delle classi definite dall'utente utilizzate dalle procedure memorizzate Java o dagli UDF Java nel collegamento al database corrente. Questa procedura memorizzata deve essere chiamata da collegamenti al database esistenti per ottenere le modifiche effettuate da una chiamata alla procedura memorizzata SQLJ.REPLACE_JAR.

Autorizzazione

NONE

Sintassi

```
>>-CALL--SQLJ.REFRESH_CLASSES-- ()-->
>-------------------------------------------------------------><
```

Esempio

Chiamata di una procedura memorizzata Java, MYPROCEDURE, che utilizza una classe in un file jar registrato con jarid MYJAR:

```
CALL MYPROCEDURE()
```

Sostituire il file jar utilizzando la seguente chiamata:

```
CALL SQLJ.REPLACE_JAR('MYJAR', '/tmp/newjarfile.jar')
```

Per fare in modo che le successive chiamate della procedura memorizzata MYPROCEDURE utilizzino il file jar aggiornato, è necessario chiamare SQLJ.REFRESH_CLASSES:

```
CALL SQLJ.REFRESH_CLASSES()
```

Richiamare la procedura memorizzata. Vengono utilizzati i file classe aggiornati quando si chiama la procedura.

```
CALL MYPROCEDURE()
```

Convenzioni per inoltrare un parametro per le UDF e le procedure memorizzate Java

La seguente tabella elenca la modalità in cui i tipi di dati SQL vengono rappresentati nelle UDF e nelle procedure memorizzate Java.

Tipo di dati SQL	Stile di parametro Java JAVA	Stile di parametro Java DB2GENERAL
SMALLINT	short	short
INTEGER	int	int
BIGINT	long	long
DECIMAL(p,s)	BigDecimal	BigDecimal
NUMERIC(p,s)	BigDecimal	BigDecimal

Tipo di dati SQL	Stile di parametro Java JAVA	Stile di parametro Java DB2GENERAL
REAL o FLOAT(p)	float	float
DOUBLE PRECISION o FLOAT o FLOAT(p)	double	double
CHARACTER(n)	String	String
CHARACTER(n) FOR BIT DATA	byte[]	com.ibm.db2.app.Blob
VARCHAR(n)	String	String
VARCHAR(n) FOR BIT DATA	byte[]	com.ibm.db2.app.Blob
GRAPHIC(n)	String	String
VARGRAPHIC(n)	String	String
DATE	Date	String
TIME	Time	String
TIMESTAMP	Timestamp	String
Indicator Variable	-	-
CLOB	-	com.ibm.db2.app.Clob
BLOB	-	com.ibm.db2.app.Blob
DBCLOB	-	com.ibm.db2.app.Clob
DataLink	-	-

Java con altri linguaggi di programmazione

Con Java, esistono molti modi per richiamare i codici scritti in linguaggi diversi da Java.

Java Native Interface

Uno dei modo in cui è possibile richiamare i codici scritti in un altro linguaggio è quello di implementare i metodi Java come 'metodi nativi.' I metodi nativi sono procedure, scritte in un altro linguaggio, che forniscono la reale implementazione di un metodo Java. È possibile che i metodi nativi accedano alla Java virtual machine utilizzando JNI (Java Native Interface). Questi metodi nativi sono in esecuzione nell'ambito del sottoprocesso Java, che è un sottoprocesso kernel, quindi è necessario che questi siano sicuri durante il sottoprocesso. Una funzione risulta sicura durante il sottoprocesso se è possibile avviarla simultaneamente in più sottoprocessi all'interno dello stesso processo. Una funzione risulta sicura durante il sottoprocesso se e solo se anche tutte le funzioni che chiama sono sicure durante il sottoprocesso.

I metodi nativi sono un "ponte" per accedere alle funzioni del sistema che non sono direttamente supportate in Java o per interfacciarsi a un codice utente esistente. Fare attenzione durante l'utilizzo dei metodi nativi, perché è possibile che il codice chiamato non sia sicuro durante il sottoprocesso.

API di richiamo Java

L'utilizzo della API di richiamo Java, che è anche parte della specifica JNI (Java Native Interface), consente ad un'applicazione non Java di utilizzare la JVM (Java virtual machine). Consente inoltre l'utilizzo del codice Java come un'estensione dell'applicazione.

Metodi nativi i5/OS PASE

La JVM (i5/OS Java virtual machine) supporta l'utilizzo di metodi nativi in esecuzione nell'ambiente i5/OS PASE. I metodi nativi i5/OS per Java consentono di trasferire facilmente le applicazioni Java che vengono eseguite in AIX sul proprio server. È possibile copiare i file di classe e le librerie dei metodi

nativi di AIX nell'IFS (integrated file system) sul server ed eseguirli da qualsiasi richiesta comandi CL (Control Language), Qshell o sessione del terminale i5/OS PASE.

Metodi nativi teraspace

La JVM (i5/OS Java virtual machine) supporta l'utilizzo dei metodi nativi del modello di memoria teraspace. Il modello di memoria teraspace fornisce un ambiente indirizzo locale-elaborazione lunga per i programmi ILE. L'utilizzo del teraspace permette di trasferire il codice metodo nativo da altri sistemi operativi a i5/OS con modifiche minime o nulle al codice sorgente.

java.lang.Runtime.exec()

È possibile utilizzare java.lang.Runtime.exec() per richiamare i programmi o i comandi dall'interno di un programma Java. Il metodo exec() avvia un altro processo nel quale è possibile eseguire qualsiasi programma o comando System i5. In questo modello, è possibile utilizzare standard in, standard out e standard err del processo secondario per la comunicazioni tra processi.

Comunicazione tra processi

Una opzione è quella che consente di utilizzare i socket per la comunicazione tra processi che si verifica tra processi principali e secondari.

È possibile inoltre utilizzare i file di flusso per la comunicazione tra programmi. Altrimenti, consultare l'argomento Comunicazioni tra processi per una panoramica delle opzioni quando si comunica con programmi che sono in esecuzione in un altro processo.

Per richiamare Java da altri linguaggi, consultare le sezioni Esempio: richiamo di Java da C oppure Esempio: richiamo di Java da RPG.

È anche possibile utilizzare IBM Toolbox per Java per richiamare i programmi e i comandi esistenti sul server. Le code dati e i messaggi System i5 sono di norma utilizzati per le comunicazioni tra processi con IBM Toolbox per Java.

Nota: utilizzando Runtime.exec(), IBM Toolbox per Java o JNI, è possibile compromettere la trasferibilità del programma Java. È necessario evitare l'utilizzo di questi metodi in un ambiente Java "puro".

Concetti correlati

"API di richiamo Java" a pagina 213
L'API di richiamo, che fa parte della JNI (Java Native Interface), consente ad un codice diverso da Java di creare una JVM (Java virtual machine) e di caricare ed utilizzare classi Java. Questa funzione consente a un programma con più sottoprocessi di utilizzare le classi Java in esecuzione in una singola JVM (Java virtual machine), in più sottoprocessi.

"Utilizzo dei socket per le comunicazioni tra processi" a pagina 233
I flussi di socket comunicano tra i programmi in esecuzione su processi separati.

"Utilizzo dei flussi di immissione ed emissione per la comunicazione tra processi" a pagina 237
I flussi di immissione ed emissione comunicano tra programmi che sono in esecuzione in processi separati.

Riferimenti correlati

"Esempio: richiamo di Java da C" a pagina 238
Questo è un esempio di un programma C che utilizza la funzione system() per chiamare il programma Hello Java.

"Esempio: richiamo di Java da RPG" a pagina 238
Questo è un esempio di un programma RPG che utilizza l'API QCMDEXC per chiamare il programma Hello Java.

Informazioni correlate

Utilizzo della Java Native Interface per i metodi nativi

Sarebbe opportuno utilizzare i metodi nativi soltanto in casi in cui Java puro non è in grado di rispondere alle esigenze di programmazione dell'utente.

Limitare l'utilizzo dei metodi nativi a queste circostanze:

- Per accedere alle funzioni di sistema che non sono disponibili utilizzando Java puro.
- Per implementare i metodi molto sensibili alle prestazioni che possono ottenere vantaggi significativi da un'implementazione nativa.
- Per interfacciarsi con le API (application program interface) esistenti che consentono a Java di chiamare altre API.

Le istruzioni che seguono si applicano all'utilizzo della JNI (Java Native Interface) con il linguaggio C. Per informazioni sull'utilizzo della JNI con il linguaggio RPG, consultare il capitolo 11 del manuale WebSphere Development Studio: ILE RPG Programmer's Guide, SC09-2507.

Per utilizzare la JNI (Java Native Interface) per i metodi nativi, effettuare le seguenti operazioni:

1. Progettare la classe specificando quali metodi sono nativi con la sintassi di linguaggio Java standard.
2. Stabilire una libreria e un nome programma per il programma di servizio (*SRVPGM) che contiene le implementazioni del metodo nativo. Quando si scrive il codice per la chiamata del metodo System.loadLibrary() nel programma di inizializzazione statico per la classe, specificare il nome del programma di servizio.
3. Utilizzare lo strumento `javac` per compilare l'origine Java in un file di classe.
4. Utilizzare lo strumento `javah` per creare il file di intestazione (.h). Tale file contiene i prototipi esatti per creare le implementazioni del metodo nativo. L'opzione -d specifica l'indirizzario dove è necessario creare il file di intestazione.
5. Copiare il file di intestazione dall'IFS (Integrated File System) in un membro in un file di origine utilizzando il comando CPYFRMSTMF (Copia dal file di flusso). È necessario copiare il file di intestazione in un membro del file di origine affinché il compilatore C lo utilizzi. Utilizzare il nuovo supporto del file di flusso per il comando CRTCMOD (Creazione programma C/400 ILE di bind) per lasciare i file di intestazione C e di origine C nell'IFS (integrated file system). Per ulteriori informazioni sul comando CRTCMOD e sull'utilizzo dei file di flusso, consultare il manuale WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712.
6. Scrittura del codice del metodo nativo. Consultare l'argomento relativo alle Considerazioni sui sottoprocessi e sui metodi nativi Java per ulteriori dettagli sui linguaggi e sulle funzioni utilizzati per i metodi nativi.
   a. Includere il file di intestazione creato nelle fasi precedenti.
   b. Associare i prototipi nel file di intestazione in modo esatto.
   c. Convertire le stringhe in ASCII (American Standard Code for Information Interchange) se le stringhe devono essere passate alla JVM (Java virtual machine). Per ulteriori informazioni, consultare l'argomento relativo alla codifica di caratteri Java.
7. Se è necessario che il proprio metodo nativo interagisca con la JVM (Java virtual machine), utilizzare le funzioni fornite con JNI.
8. Compilare il proprio codice sorgente C, utilizzando il comando CRTCMOD, in un oggetto modulo (*MODULE).
9. Collegare uno o più oggetti moduli in un programma di servizio (*SRVPGM) utilizzando il comando CRTSRVPGM (Creazione programma di servizio). È necessario che il nome di tale programma corrisponda al nome fornito nel proprio codice Java i trova nelle chiamate alla funzione System.load() o System.loadLibrary().

10. Se è stata utilizzata la chiamata System.loadLibrary() nel proprio codice Java, effettuare una delle seguenti attività tra quelle più appropriate al J2SE in esecuzione:
    - Includere l'elenco delle librerie necessarie nella variabile di ambiente LIBPATH. È possibile modificare la variabile di ambiente LIBPATH in QShell e dalla riga comandi i5/OS.
      – Dalla richiesta comandi Qshell, immettere:

        ```
        export LIBPATH=/QSYS.LIB/MYLIB.LIB
        java -Djava.version=1.5 myclass
        ```

      – Oppure, dalla riga comandi:

        ```
        ADDENVVAR LIBPATH '/QSYS.LIB/MYLIB.LIB'
        JAVA PROP((java.version 1.5)) myclass
        ```

    - Oppure fornire l'elenco nella proprietà **java.library.path**. È possibile modificare la proprietà java.library.path in QShell e dalla riga comandi i5/OS.
      – Dalla richiesta comandi Qshell, immettere:

        ```
        java -Djava.library.path=/QSYS.LIB/MYLIB.LIB -Djava.version=1.5 myclass
        ```

      – In alternativa, dalla riga comandi i5/OS, immettere:

        ```
        JAVA PROP((java.library.path '/QSYS.LIB/MYLIB.LIB') (java.version '1.5')) myclass
        ```

    Dove */QSYS.LIB/MYLIB.LIB* è la libreria che si intende caricare utilizzando la chiamata System.loadLibrary() e *myclass* è il nome della propria applicazione Java.
11. La sintassi del percorso per System.load(String path) può essere una delle seguenti:
    - /qsys.lib/sysNMsp.srvpgm (per *SRVPGM QSYS/SYSNMSP)
    - /qsys.lib/mylib.lib/myNMsp.srvpgm (per *SRVPGM MYLIB/MYNMSP)
    - un collegamento simbolico, ad esempio /home/mydir/myNMsp.srvpgm che collega a /qsys.lib/mylib.lib/myNMsp.srvpgm

      Nota: ciò è equivalente all'utilizzo del metodo System.loadLibrary(″myNMsp″).

    Nota: il nome percorso, generalmente, è una costante letterale stringa racchiusa tra virgolette. Ad esempio, è possibile utilizzare il seguente codice:

    ```
    System.load("/qsys.lib/mylib.lib/myNMsp.srvpgm")
    ```

12. Il parametro libname per System.loadLibrary(String libname) è, generalmente, una costante letterale stringa tra virgolette che identifica la libreria del metodo nativo. Il sistema utilizza l'elenco librerie corrente e le variabili di ambiente LIBPATH e PASE_LIBPATH per ricercare un programma di servizio o l'eseguibile i5/OS PASE corrispondente al nome libreria. Ad esempio, `loadLibrary("myNMsp")` risulta nella ricerca di un *SRVPGM denominato MYNMSP o di un eseguibile i5/OS PASE denominato libmyNMsp.a o libmyMNsp.so.

Consultare Esempi: utilizzo della JNI (Java Native Interface) per metodi nativi per un esempio di come utilizzare la JNI per i metodi nativi.

Websphere Development Studio: ILE RPG Programmer's Guide, SC09-2507.

"Considerazioni sui sottoprocessi e i metodi nativi di Java" a pagina 218
È possibile utilizzare i metodi nativi per accedere alle funzioni non disponibili in Java. Per utilizzare meglio Java con i metodi nativi, bisogna tenere presente i seguenti concetti.

Java Native Interface di Sun Microsystems, Inc.

"Esempi: utilizzo della JNI (Java Native Interface) per i metodi nativi" a pagina 558
Questo programma è un semplice esempio di JNI (Java Native Interface) in cui viene utilizzato un metodo nativo C per visualizzare ″Hello, World.″ Utilizzare lo strumento `javah` con il file di classe

NativeHello per generare il file NativeHello.h. Questo esempio presume che l'implementazione C di NativeHello faccia parte di un programma di servizio denominato NATHELLO.

"Stringhe nei metodi nativi" a pagina 219
Molte funzioni JNI (Java Native Interface) accettano stringhe nello stile del linguaggio C come parametri. Ad esempio, la funzione JNI FindClass() accetta un parametro di stringa che specifica il nome completo di un file di classe. Se il file di classe viene rilevato, esso è caricato da FindClass e viene restituito un riferimento ad esso al chiamante di FindClass.

"Codifiche del carattere Java" a pagina 26
I programmi Java possono convertire i dati in diversi formati, consentendo alle applicazioni di trasferire e utilizzare informazioni da diversi tipi di serie di caratteri internazionali.

API di richiamo Java

L'API di richiamo, che fa parte della JNI (Java Native Interface), consente ad un codice diverso da Java di creare una JVM (Java virtual machine) e di caricare ed utilizzare classi Java. Questa funzione consente a un programma con più sottoprocessi di utilizzare le classi Java in esecuzione in una singola JVM (Java virtual machine), in più sottoprocessi.

IBM Developer Kit per Java supporta la API di richiamo Java per i seguenti tipi di chiamanti:

- Un programma ILE o un programma di servizio creato per `STGMDL(*SNGLVL)` e `DTAMDL(*P128)`
- Un programma ILE o un programma di servizio creato per `STGMDL(*TERASPACE)` e `DTAMDL(*LLP64)`
- Un i5/OS PASE eseguibile creato per AIX a 32-bit o 64-bit

L'applicazione controlla la JVM (Java virtual machine). L'applicazione può creare la JVM (Java virtual machine), chiamare metodi Java (nello stesso modo in cui un'applicazione chiama le sottoroutine) ed eliminare la JVM (Java virtual machine). Una volta creata la JVM (Java virtual machine), essa rimane in stato di attesa per l'esecuzione all'interno del processo finché l'applicazione non la elimina esplicitamente. Durante l'eliminazione, la JVM (Java virtual machine) esegue la ripulitura, come eseguire i programmi di chiusura, arrestare i sottoprocessi della JVM (Java virtual machine) e rilasciare le risorse della JVM (Java virtual machine).

Con una JVM (Java virtual machine) pronta per l'esecuzione, è possibile che un'applicazione scritta in linguaggio ILE, come C e RPG, effettui una chiamata nella JVM (Java virtual machine) per eseguire una qualsiasi funzione. È inoltre possibile che essa torni dalla JVM (Java virtual machine) all'applicazione C ed effettui nuovamente una chiamata nella JVM (Java virtual machine) e così via. La JVM (Java virtual machine) viene creata una volta e non è necessario ricrearla prima di effettuare una chiamata nella JVM (Java virtual machine) per eseguire una parte più o meno grande del codice Java.

Quando si utilizza l'API di richiamo per eseguire i programmi Java, la destinazione per STDOUT e STDERR viene controllata dall'utilizzo di una variabile di ambiente denominata QIBM_USE_DESCRIPTOR_STDIO. Se tale variabile viene impostata su Y o I (ad esempio, QIBM_USE_DESCRIPTOR_STDIO=Y), la JVM (Java virtual machine) utilizza descrittori del file STDIN (fd 0), STDOUT (fd 1) e STDERR (fd 2). In questo caso, è necessario impostare questi descrittori del file su valori validi aprendoli come i primi tre file o pipe in questo lavoro. Al primo file aperto nel lavoro viene dato il valore fd di 0, al secondo fd di 1 e al terzo fd di 2. Per i lavori iniziati con l'API Spawn, è possibile preassegnare questi descrittori utilizzando una correlazione del descrittore del file (esaminare la documentazione sull'API Spawn). Se la variabile di ambiente QIBM_USE_DESCRIPTOR_STDIO non è impostata o è impostata su qualsiasi altro valore, i descrittori del file non vengono utilizzati per STDIN, STDOUT o STDERR. Al contrario, STDOUT e STDERR vengono instradati su un file di spool di proprietà del lavoro corrente e l'utilizzo di STDIN provoca un'eccezione IE.

Funzioni dell'API di richiamo:

IBM Developer Kit per Java supporta queste funzioni dell'API di richiamo.

Nota: prima di utilizzare questa API, è necessario assicurarsi che l'utente si trovi in un lavoro capace di supportare più sottoprocessi. Consultare Applicazioni con più sottoprocessi per ulteriori informazioni su lavori capaci di supportare più sottoprocessi.

- **JNI_GetCreatedJavaVMs**

  Restituisce informazioni su tutte le JVM (Java virtual machines) create. Nonostante questa API preveda la restituzione di informazioni per più JVM (Java virtual machine), può esistere una sola JVM per processo. Pertanto, l'API restituirà solo una JVM.

  Firma:

  ```
  jint JNI_GetCreatedJavaVMs(JavaVM **vmBuf,
                             jsize bufLen,
                             jsize *nVMs);
  ```

  vmBuf è un'area di emissione la cui dimensione è determinata da bufLen, cioè il numero dei puntatori. Ogni JVM (Java virtual machine) ha associata una struttura JavaVM definita in java.h. Questa API memorizza un puntatore nella struttura JavaVM associata con ogni JVM (Java virtual machine) creata in vmBuf, a meno che vmBuf sia 0. I puntatori alla struttura JavaVM vengono memorizzati secondo l'ordine delle JVM (Java virtual machine) create. nVMs restituisce il numero di macchine virtuali correntemente create. Il server supporta la creazione di più di una JVM (Java virtual machine) ed è quindi possibile prevedere un valore maggiore di uno. Queste informazioni, insieme alla dimensione di vmBuf, determinano se vengono restituiti i puntatori alle strutture JavaVM per ogni JVM (Java virtual machine) creata.

- **JNI_CreateJavaVM**

  Consente di creare una JVM (Java virtual machine) e successivamente di utilizzarla in un'applicazione.

  Firma:

  ```
  jint JNI_CreateJavaVM(JavaVM **p_vm,
                        void **p_env,
                        void *vm_args);
  ```

  p_vm è l'indirizzo di un puntatore JavaVM per la JVM (Java virtual machine) appena creata. Molte altre API di richiamo JNI utilizzano p_vm per identificare la JVM (Java virtual machine). p_env è l'indirizzo di un puntatore ambiente JNI per la JVM (Java virtual machine) appena creata. Esso punta ad una tabella di funzioni JNI che avvia tali funzioni. vm_args è una struttura che contiene i parametri di inizializzazione della JVM (Java virtual machine).

  Se si avvia un comando RUNJVA (Esecuzione programma Java) o un comando JAVA e si specifica una proprietà che dispone di un parametro del comando equivalente, il parametro del comando ha la precedenza. La proprietà viene ignorata.

  Per un elenco di proprietà univoche supportate dalla API JNI_CreateJavaVM, consultare "Proprietà di sistema Java" a pagina 15.

  Nota: Java su System i5 supporta la creazione di una sola JVM (Java virtual machine) in un singolo lavoro o processo. Per ulteriori informazioni, consultare "Supporto per più JVM (Java virtual machine)" a pagina 215

- **DestroyJavaVM**

  Elimina la JVM (Java virtual machine).

  Firma:

  ```
  jint DestroyJavaVM(JavaVM *vm)
  ```

  Quando viene creata la JVM (Java virtual machine), vm è il puntatore JavaVM restituito.

- **AttachCurrentThread**

  Collega un sottoprocesso ad una JVM (Java virtual machine), in modo che possa utilizzare i servizi JVM (Java virtual machine).

  Firma:

  ```
  jint AttachCurrentThread(JavaVM *vm,
                           void **p_env,
                           void *thr_args);
  ```

Il puntatore JavaVM, vm, identifica la JVM (Java virtual machine) cui è collegato il sottoprocesso. p_env è il puntatore all'ubicazione dove si trova il puntatore all'interfaccia JNI del corrente sottoprocesso. thr_args contiene argomenti di collegamento del sottoprocesso specifico VM.

- **DetachCurrentThread**

  Firma:

  ```
  jint DetachCurrentThread(JavaVM *vm);
  ```

  vm identifica la JVM (Java virtual machine) da cui il sottoprocesso è stato scollegato.

 Java Native Interface di Sun Microsystems, Inc.

Supporto per più JVM (Java virtual machine):

Java sulla piattaforma System i5 non supporta più la creazione di più di una JVM (Java virtual machine) in un singolo lavoro o processo. Questa limitazione influenza solo gli utenti che creano le JVM utilizzando l'API di richiamo JNI (Java Native Interface). Questa modifica non influisce sulla modalità di utilizzo del comando java per eseguire i programmi Java.

La chiamata di JNI_CreateJavaVM() più di una volta in un lavoro ha esito negativo e JNI_GetCreatedJavaVMs() non restituisce più di una JVM in un elenco di risultati.

Il supporto per la creazione di una sola JVM in un singolo lavoro o processo segue gli standard dell'implementazione di riferimento di Java di Sun Microsystems, Inc.

Esempio: API di richiamo Java:

Questo esempio segue il paradigma API di richiamo standard.

Effettua quanto segue:

- Crea una JVM (Java virtual machine) utilizzando JNI_CreateJavaVM.
- Utilizza la JVM (Java virtual machine) per trovare il file di classe che si intende eseguire.
- Rileva il methodID per il metodo principale della classe.
- Chiama il metodo principale della classe.
- Notifica gli errori se si verifica un'eccezione.

Quando si crea il programma, il programma di servizio QJVAJNI o QJVAJNI64 fornisce le funzioni dell'API di richiamo JNI_CreateJavaVM. JNI_CreateJavaVM crea la JVM (Java virtual machine).

Nota: QJVAJNI64 è un nuovo programma di servizio per il metodo nativo teraspace/LLP64 e per il supporto dell'API di richiamo.

Questi programmi di servizio risiedono nell'indirizzario di collegamento del sistema e per tanto non occorre identificarli in modo esplicito su un comando di creazione CL (Control Language). Ad esempio, non è necessario identificarli in modo esplicito quando si utilizza il comando CRTPGM (Creazione programma) o il comando CRTSRVPGM (Creazione programma di servizio.

Per eseguire il programma, una delle opzioni possibili è quella di utilizzare il seguente comando CL:

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```

Qualsiasi lavoro che crei una JVM (Java virtual machine) deve essere in grado di supportare più sottoprocessi. L'emissione dal programma principale, così come una qualsiasi emissione dal programma, viene inserita in file di spool QPRINT. I file di spool sono visibili quando si utilizza il comando CL WRKSBMJOB (Gestione lavori inoltrati) e si visualizza il lavoro avviato utilizzando il comando CL SBMJOB (Inoltro lavoro).

Esempio: utilizzo dell'API di richiamo Java

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
#define OS400_JVM_15
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

/* Specificare il programma che fa in modo che tutte le stringhe letterali nel codice
 * sorgente vengano memorizzate in ASCII (il quale, per le stringhe
 * utilizzare, equivale a UTF-8)
 */

#pragma convert(819)

/* Procedura: Oops
 *
 * Descrizione: La routine del programma di aiuto viene chiamata quando una funzione JNI
 *              restituisce un valore zero, indicando un errore serio.
 *              Questa routine riporta l'eccezione a stderr e
 *              chiude senza preavviso la JVM con un FatalError.
 *
 * Parametri:   env -- JNIEnv* da utilizzare per le chiamate JNI
 *              msg -- char* che punta alla descrizione errore in UTF-8
 *
 * Nota:        Il controllo non viene restituito dopo la chiamata a FatalError
 *              e non viene restituito da questa procedura.
 */

void Oops(JNIEnv* env, char *msg) {
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*env)->FatalError(env, msg);
}

/* Questa è la routine "main" del programma. */
int main (int argc, char *argv[])
{

    JavaVMInitArgs initArgs; /* Struttura di inizializzazione VM (Virtual Machine),
                              * passata dal riferimento a JNI_CreateJavaVM(). Vedere jni.h per dettagli
                              */
    JavaVM* myJVM;           /* Puntatore JavaVM impostato dalla chiamata a JNI_CreateJavaVM */
    JNIEnv* myEnv;           /* Puntatore JNIEnv impostato dalla chiamata a JNI_CreateJavaVM */
    char*   myClasspath;     /* Classpath 'string' modificabile */
    jclass  myClass;         /* La classe da chiamare, 'NativeHello'. */
    jmethodID mainID;        /* L'ID metodo della routine 'main'. */
    jclass  stringClass;     /* Necessario per creare l'arg String[] per main */
    jobjectArray args;       /* String[] stesso */
    JavaVMOption options[1]; /* Schiera opzioni -- usare le opzioni per impostare classpath */
    int     fd0, fd1, fd2;   /* descrittore file per IO */

    /* Aprire i descrittori file in modo che IO sia operativo. */
    fd0 = open("/dev/null1", O_CREAT|O_TRUNC|O_RDWR,   S_IRUSR|S_IROTH);
    fd1 = open("/dev/null2", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);
    fd2 = open("/dev/null3", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);

    /*  Impostare il campo versione degli argomenti di inizializzazione per J2SE v1.5. */
    initArgs.version = 0x00010005;
 /*  Per utilizzare J2SDK v1.4, impostare initArgs.version = 0x00010004; */

```

```
|      /* Ora, si desidera specificare l'indirizzario per la classe da eseguire nel classpath.
|       * Con Java2, classpath viene passato come una opzione.
|       * Nota: specificare il nome indirizzario in formato UTF-8. Quindi, raggruppare
|       *       i blocchi di codice in istruzioni #pragma convert.
|       */
|      options[0].optionString="-Djava.class.path=/CrtJvmExample";
|    /* Per utilizzare J2SDK v1.4, sostituire '1.5' con '1.4'.
|    options[1].optionString="-Djava.version=1.5" */
|
|      initArgs.options=options;  /* Inoltrare il classpath impostato. */
|      initArgs.nOptions = 1;     /* Inoltrare le opzioni di classpath e versione */
|
|      /*  Creare la JVM -- un codice di ritorno diverso da zero indica che si è verificato
|       *  un errore. Ritornare a EBCDIC e scrivere un messaggio in stderr
|       *  prima di uscire dal programma.
|       */
|      if (JNI_CreateJavaVM("myJVM, (void **)"myEnv, (void *)"initArgs)) {
|     #pragma convert(0)
|            fprintf(stderr, "Failed to create the JVM\n");
|     #pragma convert(819)
|            exit(1);
|      }
|
|      /*  Utilizzare la JVM appena creata per trovare la classe di esempio,
|       *  chiamata 'NativeHello'.
|       */
|      myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
|      if (! myClass) {
|          Oops(myEnv, "Failed to find class 'NativeHello'");
|      }
|
|      /* Ora, richiamare l'identificativo del metodo per il punto di entrata 'main'
|       * della classe.
|       * Nota: la firma di 'main' è sempre uguale per qualsiasi
|       *       classe chiamata dal seguente comando java:
|       *           "main" , "([Ljava/lang/String;)V"
|       */
|      mainID = (*myEnv)->GetStaticMethodID(myEnv,myClass,"main",
|                                          "([Ljava/lang/String;)V");
|      if (! mainID) {
|          Oops(myEnv, "Failed to find jmethodID of 'main'");
|      }
|
|      /* Richiamare jclass per String per creare la schiera
|       * di String da inoltrare a 'main'.
|       */
|      stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
|      if (! stringClass) {
|          Oops(myEnv, "Failed to find java/lang/String");
|      }
|
|      /*  Ora, è necessario creare una schiera di stringhe vuota,
|       *  poiché main richiede una schiera di questo tipo come parametro.
|       */
|      args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
|      if (! args) {
|          Oops(myEnv, "Failed to create args array");
|      }
|
|      /* Ora, si ha l'ID metodo di main e la classe, quindi è possibile
|       * chiamare il metodo main.
|       */
|      (*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);
|
|      /* Controllare errori. */
|      if ((*myEnv)->ExceptionOccurred(myEnv)) {
|          (*myEnv)->ExceptionDescribe(myEnv);
```

```
    }

    /* Infine, eliminare la JavaVM creata. */
    (*myJVM)->DestroyJavaVM(myJVM);

    /* Eseguite tutte le operazioni. */
    return 0;
}
```

Per ulteriori informazioni, consultare "API di richiamo Java" a pagina 213.

Considerazioni sui sottoprocessi e i metodi nativi di Java

È possibile utilizzare i metodi nativi per accedere alle funzioni non disponibili in Java. Per utilizzare meglio Java con i metodi nativi, bisogna tenere presente i seguenti concetti.

- Un sottoprocesso Java, se creato da Java o da un sottoprocesso nativo collegato, ha tutte le eccezioni a virgola mobile disabilitate. Se il sottoprocesso esegue un metodo nativo che abilita nuovamente le eccezioni a virgola mobile, Java non le disattiva una seconda volta. Se l'applicazione dell'utente non le disabilita prima di ritornare all'esecuzione del codice Java, è possibile che il codice Java non funzioni correttamente se si verifica un'eccezione a virgola mobile. Quando un sottoprocesso nativo si scollega dalla Java virtual machine, la maschera dell'eccezione a virgola mobile viene ripristinata al valore che aveva quando il sottoprocesso era collegato.
- Quando un sottoprocesso nativo si collega alla Java virtual machine, la Java virtual machine modifica la priorità dei sottoprocessi, se lo ritiene necessario, in modo tale da conformarsi a uno dei dieci schemi di priorità che Java definisce. Quando il sottoprocesso si scollega, la priorità viene ripristinata. Dopo il collegamento, è possibile che il sottoprocesso modifichi la relativa priorità utilizzando un'interfaccia del metodo nativo (ad esempio una API POSIX). Java non riporta la priorità del sottoprocesso sulle transazioni alla Java virtual machine.
- Il componente API di richiamo di JNI (Java Native Interface) consente a un utente di incorporare una Java virtual machine all'interno dell'applicazione. Se un'applicazione crea una JVM (Java virtual machine) e la JVM (Java virtual machine) termina in modo anomalo, viene segnalata l'eccezione MCH74A5 ″Java Virtual Machine Terminated″ System i5 al sottoprocesso iniziale del processo se quel sottoprocesso era collegato alla JVM (Java virtual machine) quando la JVM (Java virtual machine) ha terminato l'esecuzione. È possibile che la Java venga arrestata in maniera anomala per una delle seguenti ragioni:
  - L'utente chiama il metodo java.lang.System.exit().
  - Un sottoprocesso necessario alla Java virtual machine è terminato.
  - Si verifica un errore interno nella Java virtual machine.

  Questa funzionalità differisce dalla maggior parte delle altre piattaforme Java. Sulla maggior parte delle altre piattaforme, il processo che crea automaticamente la Java termina in modo anomalo nel momento in cui si arresta la Java virtual machine. Se l'applicazione controlla e gestisce un'eccezione MCH74A5 segnalata, è possibile proseguire l'esecuzione. Altrimenti il processo termina quando l'eccezione diventa non gestita. Aggiungendo il codice che gestisce l'eccezione MCH74A5 specifica per il sistema System i5, è possibile che l'applicazione diventi meno trasportabile su altre piattaforme.

Poiché l'esecuzione dei metodi nativi avviene sempre in un processo con più sottoprocessi, è necessario che il codice che questi contengono sia protetto durante il sottoprocesso. Ciò impone le seguenti limitazioni riguardo i linguaggi e le funzioni utilizzate per i metodi nativi:

- Non bisogna utilizzare CL ILE in relazione ai metodi nativi, perché questo linguaggio non è protetto durante il sottoprocesso. Per eseguire comandi CL protetti durante il sottoprocesso, è possibile utilizzare la funzione system() di linguaggio C o il metodo java.lang.Runtime.exec().
  - Utilizzare la funzione system() di linguaggio C per eseguire comandi CL protetti durante il sottoprocesso dall'ambito di un metodo nativo C o C++.
  - Utilizzare il metodo java.lang.Runtime.exec() per eseguire comandi CL protetti direttamente da Java.

- È possibile utilizzare ILE C, ILE C++, ILE COBOL e ILE RPG per scrivere un metodo nativo, ma è necessario che tutte le funzioni chiamate dall'ambito del metodo nativo siano protette durante il sottoprocesso.

  Nota: il supporto al tempo di compilazione per la scrittura dei metodi nativi attualmente viene fornito solo per i linguaggi C, C++ e RPG. Anche se possibile, la scrittura dei metodi nativi in altri linguaggi sarebbe molto più complicato.

  Attenzione: ***non tutte le funzioni standard C, C++, COBOL o RPG sono protette durante il sottoprocesso.***
- Le funzioni C e C++ exit() e abort() non devono essere utilizzate nell'ambito di un metodo nativo. Queste funzioni determinano l'arresto dell'intero processo che esegue la Java virtual machine. Ciò include tutti i sottoprocessi presenti nel processo, indipendentemente dal fatto che la creazione è avvenuta da Java o meno.

  Nota: la funzione exit() è la funzione C e C++ e non è uguale al metodo java.lang.Runtime.exit().

Per ulteriori informazioni sui sottoprocessi sul server, consultare Applicazioni a più sottoprocessi.

Metodi nativi e JNI (Java Native Interface)

I metodi nativi sono quei metodi Java che si avviano in un linguaggio diverso da Java. È possibile che i metodi nativi accedano alle funzioni e alle API specifiche per il sistema che non sono disponibili direttamente in Java.

L'utilizzo di metodi nativi limita la trasferibilità di un'applicazione, perché implica un codice specifico per il sistema. I metodi nativi possono essere istruzioni di codice nativo nuovo oppure istruzioni di codice nativo che chiamano un codice nativo esistente.

Quando si decide che è necessario un metodo nativo, è possibile che sia necessario interagire con la JVM (Java virtual machine) nella quale esso viene eseguito. La JNI (Java Native Interface) facilita questa interazione in modo indipendente dalla piattaforma.

La JNI è una serie di interfacce che consentono a un metodo nativo di interagire con la JVM (Java virtual machine) in vari modi. Ad esempio, la JNI include interfacce che creano nuovi oggetti e chiamano metodi che individuano e impostano campi, elaborano eccezioni e manipolano stringhe e schiere.

Per una descrizione completa della JNI, consultare l'argomento relativo alla Java Native Interface di Sun Microsystems, Inc.

Informazioni correlate

Java Native Interface di Sun Microsystems, Inc.

Stringhe nei metodi nativi

Molte funzioni JNI (Java Native Interface) accettano stringhe nello stile del linguaggio C come parametri. Ad esempio, la funzione JNI FindClass() accetta un parametro di stringa che specifica il nome completo di un file di classe. Se il file di classe viene rilevato, esso è caricato da FindClass e viene restituito un riferimento ad esso al chiamante di FindClass.

Tutte le funzioni JNI presuppongono che i parametri di stringa siano codificati in UTF-8. Per dettagli su UTF-8, è possibile fare riferimento alla Specifica JNI, ma nella maggior parte dei casi è sufficiente osservare che l'ASCII (American Standard Code for Information Interchange) a 7-bit è equivalente alla rappresentazione di UTF-8. I caratteri ASCII a 7-bit sono in realtà caratteri a 8-bit ma il primo bit è sempre 0. Perciò, la maggior parte di stringhe C ASCII sono già in UTF-8.

Il compilatore C sul server opera in EBCDIC (extended binary-coded decimal interchange code) per impostazione predefinita, per cui è possibile fornire stringhe alle funzioni JNI in UTF-8. Esistono due

modi di effettuare questa operazione. È possibile utilizzare le stringhe di costanti letterali oppure è possibile utilizzare stringhe dinamiche. Le stringhe di costanti letterali sono stringhe il cui valore è noto quando il codice sorgente viene compilato. Le stringhe dinamiche sono stringhe il cui valore non è noto in fase di compilazione, ma è in realtà elaborato durante il tempo di esecuzione.

Stringhe di costanti letterali nei metodi nativi:

È più semplice codificare le stringhe di costanti letterali in formato UTF-8 se la stringa è composta da caratteri con rappresentazione ASCII (American Standard Code for Information Interchange) di 7-bit.

Se è possibile rappresentare la stringa in ASCII, come avviene per la maggior parte di esse, allora la stringa può essere racchiusa tra parentesi da istruzioni 'pragma' che modificano la codepage corrente del compilatore. Successivamente, il compilatore memorizza la stringa internamente nel formato UTF-8 richiesto dal JNI. Se non è possibile rappresentare la stringa in ASCII, è più semplice trattare la stringa originale EBCDIC (extended binary-coded decimal interchange code) come una stringa dinamica ed elaborarla utilizzando iconv() prima di inoltrarla a JNI.

Ad esempio, per rilevare la classe denominata `java/lang/String`, il codice risulta in questo modo:

```
#pragma convert(819)
myClass = (*env)->FindClass(env,"java/lang/String");
#pragma convert (0)
```

Il primo pragma, con il numero 819, indica al compilatore di memorizzare tutte le stringhe tra virgolette successive (stringhe di costanti letterali) in ASCII. Il secondo pragma, con il numero 0, indica al compilatore di ritornare alla code page predefinita del compilatore per le stringhe tra virgolette, che solitamente rappresenta la code page EBCDIC 37. Così, racchiudendo tra virgolette questa chiamata con queste pragma, si soddisfano i requisiti JNI che richiede la codifica dei parametri di stringa in UTF-8.

Attenzione: attenzione alle sostituzioni di testo. Ad esempio, se il codice risulta in questo modo:

```
#pragma convert(819)
#define MyString "java/lang/String"
#pragma convert(0)
myClass = (*env)->FindClass(env,MyString);
```

Allora la stringa risultante è EBCDIC, perché il valore di MyString viene sostituito nella chiamata FindClass durante la compilazione. Al momento della sostituzione, il pragma, numero 819, non è in funzione. In questo modo le stringhe di costanti letterali non vengono memorizzate in ASCII.

Conversione delle stringhe dinamiche in e da EBCDIC, Unicode e UTF-8:

Per gestire le variabili di stringhe calcolate durante il tempo di esecuzione, potrebbe essere necessario convertire le stringhe in e da EDCDIC (extended binary-coded decimal interchange), Unicode e UTF-8.

L'API del sistema che fornisce la funzione di conversione pagina del codice è iconv(). Per utilizzare iconv(), seguire queste fasi:

1. Creare un descrittore della conversione con `QtqIconvOpen()`.
2. Chiamare `iconv()` per utilizzare il descrittore da convertire in una stringa.
3. Chiudere il descrittore utilizzando `iconv_close`.

Nell'Esempio 3 dell'utilizzo di Java Native Interface per esempi dei metodi nativi, la routine crea, utilizza e quindi elimina il descrittore di conversione iconv all'interno di essa. Questo schema evita i problemi con un utilizzo sottoposto a più sottoprocessi di un descrittore iconv_t, ma per il codice sensibile alle prestazioni è meglio creare un descrittore di conversione in memoria statica e moderare l'accesso multiplo ad esso utilizzando un mutex (mutual exclusion) o un'altra funzione di sincronizzazione.

Metodi nativi IBM i5/OS PASE per Java

La JVM (i5/OS Java virtual machine) supporta l'utilizzo di metodi nativi in esecuzione nell'ambiente i5/OS PASE. Prima della V5R2, la JVM i5/OS nativa utilizzava solo i metodi nativi ILE.

Il supporto per i metodi nativi di i5/OS PASE include:

- L'uso completo della JNI (System i5 Java Native Interface) dai metodi nativi di i5/OS PASE
- La capacità di richiamare i metodi nativi i5/OS PASE dalla JVM i5/OS nativa

Questo nuovo supporto consente di trasferire facilmente le applicazioni Java che vengono eseguite in AIX sul server. È possibile copiare i file di classe e le librerie dei metodi nativi di AIX nell'IFS (integrated file system) sul server ed eseguirli da qualsiasi richiesta comandi CL (Control Language), Qshell o sessione del terminale i5/OS PASE.

Informazioni correlate

i5/OS PASE
Queste informazioni presumono che l'utente abbia già una certa familiarità con i5/OS PASE. In caso contrario, consultare questo argomento per ottenere ulteriori informazioni sull'utilizzo dei metodi nativi IBM i5/OS PASE con Java.

Variabili di ambiente Java i5/OS PASE

La JVM (Java virtual machine) utilizza le variabili che seguono per avviare gli ambienti i5/OS PASE. È necessario impostare la variabile QIBM_JAVA_PASE_STARTUP per poter eseguire l'esempio per il metodo nativoIBM i5/OS PASE per Java.

QIBM_JAVA_PASE_STARTUP
È necessario impostare questa variabile di ambiente quando si verificano entrambe le condizioni che seguono:

- Si stanno utilizzando i metodi nativi di i5/OS PASE
- Si sta avviando Java da una richiesta comandi i5/OS o da una richiesta comandi Qshell

JVM utilizza questa variabile di ambiente per avviare un ambiente PASE. Il valore della variabile identifica un programma di avvio di i5/OS PASE. Il server include due programmi di avvio i5/OS PASE:

- /usr/lib/start32: avvia un ambiente i5/OS PASE a 32 bit
- /usr/lib/start64: avvia un ambiente i5/OS PASE a 64 bit

Il formato bit di tutti gli oggetti della libreria condivisa utilizzati da un ambiente i5/OS deve corrispondere al formato di bit dell'ambiente i5/OS PASE.

Non è possibile utilizzare questa variabile se si avvia Java da una sessione del terminale i5/OS PASE. Una sessione del terminale i5/OS PASE utilizza sempre un ambiente i5/OS PASE a 32 bit. Qualsiasi JVM avviata da una sessione del terminale i5/OS PASE utilizza lo stesso tipo di ambiente PASE come sessione del terminale.

QIBM_JAVA_PASE_CHILD_STARTUP
Impostare questa variabile di ambiente facoltativa quando è necessario che l'ambiente i5/OS PASE per una JVM secondaria sia diverso dall'ambiente i5/OS PASE della JVM principale. Una chiamata di Runtime.exec() in Java avvia una JVM secondaria (o child).

QIBM_JAVA_PASE_ALLOW_PREV
Impostare questa variabile di ambiente facoltativa quando si desidera utilizzare l'ambiente i5/OS PASE corrente, se ne esiste uno. In alcune situazioni è difficile determinare se è presente o meno un ambiente i5/OS PASE. L'utilizzo di QIBM_JAVA_PASE_ALLOW_PREV e QIBM_JAVA_PASE_STARTUP in combinazione, consente alla JVM di utilizzare un ambiente i5/OS PASE esistente o di avviare un nuovo ambiente i5/OS PASE.

Riferimenti correlati

"Esempi: variabili di ambiente per IBM i5/OS PASE"
Per utilizzare l'esempio dei metodi nativi di IBM i5/OS PASE per Java, è necessario impostare le seguenti variabili di ambiente.

Esempi: variabili di ambiente per IBM i5/OS PASE:

Per utilizzare l'esempio dei metodi nativi di IBM i5/OS PASE per Java, è necessario impostare le seguenti variabili di ambiente.

PASE_LIBPATH
Il server utilizza questa variabile di ambiente i5/OS PASE per identificare l'ubicazione delle librerie dei metodi nativi di i5/OS PASE. È possibile impostare il percorso ad un singolo indirizzario o a più indirizzari. Per più indirizzari, utilizzare i due punti (:) per separare le voci. Il server può anche utilizzare la variabile di ambiente LIBPATH.

Per ulteriori informazioni sull'utilizzo di Java, delle librerie di metodi nativi e di PASE_LIBPATH con quest'esempio, consultare "Gestione delle librerie di metodi nativi" a pagina 225.

PASE_THREAD_ATTACH
Se si imposta questa variabile di ambiente i5/OS PASE su Y, un sottoprocesso ILE, non avviato da i5/OS PASE, verrà automaticamente collegato a i5/OS PASE, quando richiama una procedura i5/OS PASE.

Per ulteriori informazioni sulle variabili di ambiente PASE i5/OS, consultare le voci appropriate in Gestione delle variabili di ambiente PASE i5/OS.

QIBM_JAVA_PASE_STARTUP
JVM utilizza questa variabile di ambiente per avviare un ambiente i5/OS PASE. Il valore della variabile identifica un programma di avvio di i5/OS PASE.

Per ulteriori informazioni, consultare "Variabili di ambiente Java i5/OS PASE" a pagina 221.

Utilizzo di QIBM_JAVA_PASE_CHILD_STARTUP:

La variabile di ambiente QIBM_JAVA_PASE_CHILD_STARTUP indica il programma di avvio i5/OS PASE per qualsiasi JVM secondaria.

Utilizzare QIBM_JAVA_PASE_CHILD_STARTUP quando si verificano tutte le condizioni che seguono:
- L'applicazione Java che si desidera eseguire, crea delle JVM (Java virtual machine) tramite chiamate Java a Runtime.exec()
- Sia la JVM principale che la secondaria, utilizzano i metodi nativi di i5/OS PASE
- L'ambiente i5/OS PASE delle JVM secondarie deve essere differente dall'ambiente i5/OS PASE della JVM principale

Se si verificano tutte le condizioni precedentemente elencate, effettuare quanto segue:
- Impostare la variabile di ambiente QIBM_JAVA_PASE_CHILD_STARTUP sul programma di avvio i5/OS PASE delle JVM secondarie.
- Quando si avvia la JVM principale da una richiesta comandi i5/OS o da una richiesta comandi Qshell, impostare la variabile di ambiente QIBM_JAVA_PASE_STARTUP sul programma di avvio i5/OS PASE della JVM principale.

Nota: quando si avvia la JVM principale da una sessione del terminale i5/OS PASE, non impostare QIBM_JAVA_PASE_STARTUP.

Il processo della JVM secondaria eredita la variabile di ambiente QIBM_JAVA_PASE_CHILD_STARTUP. Inoltre, i5/OS imposta la variabile di ambiente QIBM_JAVA_PASE_STARTUP del processo della JVM secondaria sul valore della variabile di ambiente QIBM_JAVA_PASE_CHILD_STARTUP dal processo principale (parent).

La tabella che segue identifica gli ambienti i5/OS PASE risultanti (se ne esistono), per le diverse combinazioni di ambienti di comandi e definizioni di QIBM_JAVA_PASE_STARTUP e QIBM_JAVA_PASE_CHILD_STARTUP:

Tabella 7. Ambienti PASE risultanti per QIBM_JAVA_PASE_STARTUP e QIBM_JAVA_PASE_CHILD_STARTUP

Ambiente di avvio			**Funzionalità risultante**	
Ambiente comandi	**QIBM_JAVA _PASE_STARTUP**	**QIBM_JAVA_PASE_ CHILD_STARTUP**	**Avvio i5/OS PASE della JVM principale**	**Avvio i5/OS PASE della JVM secondaria**
CL o QSH	StartX definito	StartY definito	Utilizzare startX	Utilizzo startY
CL o QSH	StartX definito	Non definito	Utilizzare startX	Utilizzare startX
CL o QSH	Non definito	StartY definito	Nessun ambiente i5/OS PASE	Utilizzo startY
CL o QSH	Non definito	Non definito	Nessun ambiente i5/OS PASE	Nessun ambiente i5/OS PASE
Sessione terminale i5/OS PASE	StartX definito	StartY definito	Non consentito*	Non consentito*
Sessione terminale i5/OS PASE	StartX definito	Non definito	Non consentito*	Non consentito*
Sessione terminale i5/OS PASE	Non definito	StartY definito	Utilizzo ambiente sessione terminale i5/OS PASE	Utilizzo startY
Sessione terminale i5/OS PASE	Non definito	Non definito	Utilizzo ambiente sessione terminale i5/OS PASE	Nessun ambiente i5/OS PASE

* Le righe contrassegnate con ″Non consentito″ indicano le situazioni in cui la variabile di ambiente QIBM_JAVA_PASE_STARTUP potrebbe entrare in conflitto con la sessione del terminale i5/OS PASE. A causa di tale potenziale conflitto, non è consentito l'utilizzo di QIBM_JAVA_PASE_STARTUP da una sessione del terminale i5/OS PASE.

Utilizzo di QIBM_JAVA_PASE_ALLOW_PREV:

A volte è difficile determinare se un ambientei5/OS PASE esiste già. L'utilizzo della variabile di ambiente facoltativa QIBM_JAVA_PASE_ALLOW_PREV insieme a QIBM_JAVA_PASE_STARTUP permette alla JVM di determinare se utilizzare l'ambiente i5/OS PASE (se esistente) o di avviare un nuovo ambiente i5/OS PASE.

Per utilizzare queste due variabili di ambiente in combinazione, impostarle sui seguenti valori:

- Impostare QIBM_JAVA_PASE_STARTUP sul programma di avvio predefinito
- Impostare QIBM_JAVA_PASE_ALLOW_PREV su 1

Ad esempio, un'applicazione che facoltativamente avvia un ambiente i5/OS PASE, chiama il programma che avvia la JVM. In questo caso, utilizzando le precedenti impostazioni, il programma è in grado di utilizzare l'ambiente i5/OS PASE corrente se esiste, o di avviarne uno nuovo i5/OS PASE. P

La tabella che segue identifica qualsiasi ambiente i5/OS PASE risultante dalle diverse combinazioni dell'ambiente di i5/OS PASE e dalle definizioni di QIBM_JAVA_PASE_STARTUP e QIBM_JAVA_PASE_ALLOW_PREV:

Tabella 8. Ambienti i5/OS PASE risultanti dalle combinazioni dell'ambiente i5/OS PASE e dalle definizioni di QIBM_JAVA_PASE_STARTUP e QIBM_JAVA_PASE_ALLOW_PREV

Ambiente di avvio			Funzionalità risultante
Ambiente i5/OS PASE	**QIBM_JAVA _PASE_STARTUP**	**QIBM_JAVA_PASE _ALLOW_PREV**	**Avvio i5/OS PASE della JVM**
Nessuno	Non definito	Non definito*	Nessun ambiente i5/OS PASE
Nessuno	Non definito	Definito '1'	Nessun ambiente i5/OS PASE
Nessuno	StartX definito	Non definito*	Utilizzare startX
Nessuno	StartX definito	Definito '1'	Utilizzare startX
Avviato	Non definito	Non definito*	Utilizzare l'ambiente i5/OS PASE esistente
Avviato	Non definito	Definito '1'	Utilizzare l'ambiente i5/OS PASE esistente
Avviato	StartX definito	Non definito*	Non consentito: errore JVM durante l'avvio
Avviato	StartX definito	Definito '1'	Utilizzare l'ambiente i5/OS PASE esistente

* "Non definito" significa che QIBM_JAVA_PASE_ALLOW_PREV non è incluso o possiede un valore diverso da 1.

Le ultime due righe della precedente tabella indicano delle situazioni in cui è utile impostare QIBM_JAVA_PASE_ALLOW_PREV. La JVM verifica QIBM_JAVA_PASE_ALLOW_PREV quando è già presente un ambiente i5/OS PASE ed è stato definito QIBM_JAVA_PASE_STARTUP. In caso contrario, la JVM ignora QIBM_JAVA_PASE_ALLOW_PREV.

Le variabili di ambiente QIBM_JAVA_PASE_ALLOW_PREV e QIBM_JAVA_PASE_CHILD_STARTUP sono indipendenti l'una dall'altra.

Codici di errore Java i5/OS PASE

Questo argomento, fornisce una guida su come risolvere i problemi relativi ai metodi nativi di i5/OS PASE, descrive le condizioni di errore indicate nei messaggi della registrazione lavoro i5/OS e le eccezioni in fase di esecuzione Java. Gli elenchi che seguono descrivono gli errori che si possono verificare all'avvio o al tempo di esecuzione quando si utilizzano i metodi nativi di i5/OS PASE per Java.

Errori all'avvio

Per gli errori di avvio, esaminare i messaggi nella registrazione lavoro appropriata.

Errori al tempo di esecuzione

Oltre agli errori dell'avvio, possono essere visualizzati errori Java PaseInternalError o PaseExit nell'emissione Qshell della JVM:

- PaseInternalError - indica un errore interno al sistema. Controllare le voci della Registrazione LIC (Licensed Internal Code).

  Per ulteriori informazioni sul codice di errore PaseInternalError, consultare Qp2CallPase.
- PaseExit - l'applicazione i5/OS PASE ha richiamato la funzione exit() oppure l'ambiente i5/OS PASE è stato chiuso in modo anomalo. Per ulteriori informazioni controllare la Registrazione lavoro e la Registrazione LIC (Licensed Internal Code).

Gestione delle librerie di metodi nativi

Per utilizzare le librerie di metodi nativi, in particolar modo quando si desidera gestire più versioni di una libreria di metodi nativi sul server iSeries, è necessario conoscere le convenzioni di denominazione della libreria Java e l'algoritmo di ricerca della libreria.

i5/OS utilizza il primo nome libreria metodi nativi corrispondente a quello della libreria caricata dalla JVM (Java virtual machine). Per assicurarsi che i5/OS rilevi i metodi nativi corretti, è necessario evitare conflitti tra i nomi libreria e confusioni su quale libreria metodi nativi utilizza la JVM.

Convenzioni di denominazione della libreria i5/OS PASE e AIX Java

Se il codice Java carica una libreria denominata Sample, è necessario che il file eseguibile corrispondente si chiami libSample.a o libSample.so.

Ordine di ricerca della libreria Java

Quando si abilitano i metodi nativi di i5/OS PASE per la JVM, il server utilizza tre diversi elenchi (nell'ordine che segue) per creare un singolo percorso di ricerca della libreria metodi nativi:

1. elenco librerie i5/OS
2. variabile di ambiente LIBPATH
3. variabile di ambiente PASE_LIBPATH

Per effettuare la ricerca i5/OS converte l'elenco librerie nel formato dell'IFS (integrated file system). Gli oggetti del file system QSYS hanno nomi equivalenti nell'IFS (integrated file system), ma alcuni oggetti dell'IFS non hanno nomi equivalenti nel file system QSYS. Poiché il programma di caricamento della libreria ricerca gli oggetti sia nel file system QSYS che nell'IFS, i5/OS utilizza il formato IFS per ricercare le librerie di metodi nativi.

La tabella che segue illustra il modo in cui i5/OS converte le voci dell'elenco librerie nel formato IFS:

Voce elenco librerie	Formato IFS (integrated file system)
QSYS	/qsys.lib
QSYS2	/qsys.lib/qsys2.lib
QGPL	/qsys.lib/qgpl.lib
QTEMP	/qsys.lib/qtemp.lib

Esempio: ricerca della libreria Sample2

Nell'esempio che segue, LIBPATH è impostato su /home/user1/lib32:/samples/lib32 e PASE_LIBPATH su /QOpenSys/samples/lib.

La tabella che segue, letta dall'alto verso il basso, indica il percorso di ricerca completo:

Sorgente	Indirizzari IFS (integrated file system)
Elenco librerie	/qsys.lib /qsys.lib/qsys2.lib /qsys.lib/qgpl.lib /qsys.lib/qtemp.lib
LIBPATH	/home/user1/lib32 /samples/lib32
PASE_LIBPATH	/QOpenSys/samples/lib

Nota: i caratteri in maiuscolo e minuscolo sono determinanti solo nel percorso /QOpenSys.

Per ricercare la libreria Sample2, il programma di caricamento della libreria Java ricerca i file candidati nell'ordine che segue:

1. /qsys.lib/sample2.srvpgm
2. /qsys.lib/libSample2.a
3. /qsys.lib/libSample2.so
4. /qsys.lib/qsys2.lib/sample2.srvpgm
5. /qsys.lib/qsys2.lib/libSample2.a
6. /qsys.lib/qsys2.lib/libSample2.so
7. /qsys.lib/qgpl.lib/sample2.srvpgm
8. /qsys.lib/qgpl.lib/libSample2.a
9. /qsys.lib/qgpl.lib/libSample2.so
10. /qsys.lib/qtemp.lib/sample2.srvpgm
11. /qsys.lib/qtemp.lib/libSample2.a
12. /qsys.lib/qtemp.lib/libSample2.so
13. /home/user1/lib32/sample2.srvpgm
14. /home/user1/lib32/libSample2.a
15. /home/user1/lib32/libSample2.so
16. /samples/lib32/sample2.srvpgm
17. /samples/lib32/libSample2.a
18. /samples/lib32/libSample2.so
19. /QOpenSys/samples/lib/SAMPLE2.srvpgm
20. /QOpenSys/samples/lib/libSample2.a
21. /QOpenSys/samples/lib/libSample2.so

i5/OS carica il primo candidato nell'elenco effettivamente esistente nella JVM, come libreria di metodi nativi. Anche se vengono rilevati nella ricerca candidati come '/qsys.lib/libSample2.a' e '/qsys.lib/libSample2.so', non è possibile creare file IFS o collegamenti simbolici negli indirizzari /qsys.lib. Per questo motivo, anche se i5/OS ricerca questi file candidati, non li troverà mai negli indirizzari IFS che cominciano con /qsys.lib.

Tuttavia, è possibile creare collegamenti simbolici arbitrari da altri indirizzari IFS agli oggetti i5/OS nel file system QSYS. Ne risulta che i file candidati validi includono file del tipo /home/user1/lib32/sample2.srvpgm.

Esempio: metodo nativo IBM i5/OS PASE per Java

L'esempio del metodo nativo IBM i5/OS PASE per Java chiama un'istanza di un metodo C nativo che, in seguito, utilizza la JNI (Java Native Interface) per richiamarlo nel codice Java. Invece di accedere alla stringa direttamente dal codice Java, l'esempio chiama un metodo nativo che, in seguito, richiama in Java, attraverso la JNI, per ottenere il valore stringa.

Per visualizzare le versioni HTML dei file di origine dell'esempio, utilizzare i collegamenti che seguono:

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

- "Esempio: PaseExample1.java" a pagina 555
- "Esempio: PaseExample1.c" a pagina 555

Prima di poter eseguire l'esempio di metodo nativo PASE i5/OS, è necessario completare le attività nelle seguenti sezioni:

1. "Esempio: scaricamento del codice sorgente dell'esempio sulla stazione di lavoro AIX" a pagina 556
2. "Esempio: preparazione del codice sorgente di esempio" a pagina 556
3. "Esempio: preparazione di System i5 all'esecuzione dell'esempio di metodo nativo PASE per Java" a pagina 557

Esecuzione dell'esempio di metodo nativo PASE i5/OS per Java

Una volta completate le attività precedentemente descritte, è possibile eseguire l'esempio. Utilizzare uno dei comandi che seguono per eseguire il programma di esempio:

- Da una richiesta comandi i5/OS:

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- Da una richiesta comandi Qshell o da una sessione del terminalei5/OS PASE:

```
cd /home/example
java PaseExample1
```

Metodi nativi del modello di memoria teraspace perJava

La JVM (i5/OS Java virtual machine) ora supporta l'utilizzo dei metodi nativi del modello di memoria teraspace. Il modello di memoria teraspace fornisce un ambiente indirizzo locale-elaborazione lunga per i programmi ILE. L'utilizzo del modello di memoria teraspace consente di trasferire il codice di metodo nativo da altri sistemi operativi a i5/OS con modifiche minime o nulle al codice sorgente.

Nota: Il modello di memoria teraspace fornisce un ambiente in cui la memoria statica, le variabili locali e le allocazioni di heap si trovano automaticamente nella memoria teraspace. Se si deve solo abilitare l'accesso alla memoria teraspace, **non** è necessario utilizzare il modello di memoria teraspace. È sufficiente abilitare a teraspace il proprio codice di metodo nativo. Per abilitare a teraspace il metodo nativo, utilizzare il parametro TERASPACE(*YES) in CRTCMOD (Creazione modulo C), CRTCPPMOD (Creazione modulo C++) o un altro comando di creazione di modulo.

Per dettagli sulla programmazione con il modello di memoria teraspace consultare le seguenti informazioni:

- Capitolo 4 di ILE Concepts
- Capitolo 17 di WebSphere Development Studio ILE C/C++ Programmer's Guide

Il concetto di metodo nativo Java creato per il modello di memoria teraspace è molto simile a quello di metodo nativo che utilizza una memoria a livello singolo. La JVM inoltra ai metodi nativi del modello di memoria teraspace un puntatore all'ambiente JNI (Java Native Interface) che i metodi possono utilizzare per richiamare le funzioni JNI.

Per i metodi nativi del modello di memoria teraspace la JVM fornisce delle implementazioni della funzione JNI che utilizzano il modello di memoria teraspace e i puntatori a 8 byte.

Creazione dei metodi nativi del modello di memoria teraspace

Per creare un metodo nativo di un modello di memoria teraspace, il comando di creazione del modulo del modello di memoria teraspace deve utilizzare le seguenti opzioni:

```
TERASPACE(*YES) STGMDL(*TERASPACE) DTAMDL(*LLP64)
```

L'opzione che segue (*TSIFC), per utilizzare le funzioni della memoria teraspace, è facoltativa:

```
TERASPACE(*YES *TSIFC)
```

Nota: se non si specifica `DTAMDL(*LLP64)` quando si utilizzano i metodi nativi Java del modello di memoria teraspace, il richiamo di un metodo nativo provoca un'eccezione al tempo di esecuzione.

Creazione di programmi di servizio del modello di memoria teraspace che utilizzano metodi nativi

Per creare un programma di servizio del modello di memoria teraspace utilizzare la seguente opzione sul comando CL Creazione programma servizio (CRTSRVPGM):

```
CRTSRVPGM STGMDL(*TERASPACE)
```

Inoltre, è necessario utilizzare l'opzione `ACTGRP(*CALLER)` che permette alla JVM di attivare tutti i programmi di servizio del metodo nativo del modello di memoria teraspace nello stesso gruppo di attivazione teraspace. Un simile utilizzo di un gruppo di attivazione teraspace può assicurare un'efficiente gestione delle eccezioni da parte dei metodi nativi.

Per ulteriori dettagli sull'attivazione del programma e sui gruppi di attivazione, consultare il capitolo 3 di ILE Concepts.

Utilizzo delle API di richiamo Java con i metodi nativi del modello di memoria teraspace

Utilizzare la funzione GetEnv dell'API di richiamo quando il puntatore dell'ambiente JNI non corrisponde al modello di memoria del programma di servizio. La funzione GetEnv dell'API di richiamo restituisce sempre il puntatore dell'ambiente JNI corretto.

La JVM supporta sia i metodi nativi del modello di memoria teraspace che a livello singolo, ma i due modelli di memoria utilizzano ambienti JNI differenti. Poiché i due modelli di memoria utilizzano ambienti JNI differenti, non inoltrare il puntatore di ambiente JNI come parametro tra i metodi nativi nei due modelli di memoria.

Concetti correlati

"API di richiamo Java" a pagina 213
L'API di richiamo, che fa parte della JNI (Java Native Interface), consente ad un codice diverso da Java di creare una JVM (Java virtual machine) e di caricare ed utilizzare classi Java. Questa funzione consente a un programma con più sottoprocessi di utilizzare le classi Java in esecuzione in una singola JVM (Java virtual machine), in più sottoprocessi.

Informazioni correlate

Java Native Interface di Sun Microsystems, Inc.

Comando CL CRTCMOD (Creazione modulo C)

Comando CL CRTCPPMOD (Creazione modulo C++)

Capitolo 3 di ILE Concepts

Websphere Development Studio ILE C/C++ Programmer's Guide

Confronto tra ILE (Integrated Language Environment) e Java

L'ambiente Java su un System i5 è separato dall'ILE (integrated language environment). Java non è un linguaggio ILE e non può collegarsi ai moduli oggetti ILE per creare programmi o programmi di servizio su un System i5.

ILE	Java
I membri che fanno parte della struttura file o libreria su un server System i5 memorizzano dei codici sorgente.	I file del flusso nell'IFS (Integrated File System) contengono un codice sorgente.

ILE	Java
SEU (source entry utility) modifica file di origine EBCDIC (extended binary-coded decimal interchange code).	I file di origine ASCII (American Standard Code for Information Interchange) vengono solitamente modificati utilizzando un editor della stazione di lavoro.
I file di origine vengono compilati in moduli del codice oggetto, memorizzati nelle librerie su un server System i5.	Il codice sorgente si compila nei file di classe, che l'IFS memorizza.
I moduli oggetto sono collegati tra di loro nei programmi o nei programmi di servizio.	Le classi vengono caricate dinamicamente, quando necessario, durante il tempo di esecuzione.
È possibile chiamare direttamente le funzioni scritte in altri linguaggi di programmazione ILE.	È necessario utilizzare JNI (Java Native Interface) per chiamare altri linguaggi da Java.
I linguaggi ILE vengono sempre compilati ed eseguiti come istruzioni macchina.	È possibile interpretare e compilare i programmi Java.

Utilizzo di java.lang.Runtime.exec()

Utilizzare il metodo java.lang.Runtime.exec per richiamare comandi o programmi dall'interno di un programma Java. Il metodo java.lang.Runtime.exec() consente di creare uno o più lavori aggiuntivi abilitati a sottoprocessi. Tali lavori elaborano la stringa dei comandi trasmessa al metodo.

Nota: il metodo java.lang.Runtime.exec esegue i programmi in un lavoro distinto, diversamente dalla funzione C system (). La funzione C system, infatti, esegue i programmi nello stesso lavoro.

L'elaborazione vera e propria dipende dai seguenti fattori:

- Il tipo di comando immesso su su java.lang.Runtime.exec()
- Il valore della proprietà di sistema os400.runtime.exec

Elaborazione di diversi tipi di comandi

La tabella di seguito riportata indica il modo in cui java.lang.Runtime.exec() elabora diversi tipi di comandi e mostra gli effetti della proprietà di sistema os400.runtime.exec.

	Valore della proprietà di sistema os400.runtime.exec	
Tipo di comando	**EXEC (valore predefinito)**	**QSHELL**
comando java	Avvia un secondo lavoro che esegue la JVM (Java Virtual Machine). La JVM avvia un terzo lavoro che esegue l'applicazione Java.	Avvia un secondo lavoro che esegue Qshell, l'interprete shell. Qshell avvia un terzo lavoro per eseguire l'applicazione, il programma o il comando Java.
programma	Avvia un secondo lavoro che esegue il programma eseguibile (i5/OS program o i5/OS PASE).	
Comando CL	Avvia un secondo lavoro che esegue un programma i5/OS. Il programma i5/OS esegue il comando CL nel secondo lavoro.	

Nota: nel richiamare un comando o un programma CL, occorre assicurarsi che il CCSID del lavoro contenga i caratteri trasmessi come parametri al comando richiamato.

L'elaborazione nel secondo o terzo lavoro si verifica contemporaneamente con una qualsiasi JVM (Java virtual machine) nel lavoro originario. Qualsiasi elaborazione di uscita o di chiusura in questi lavori non influenza la JVM originale.

Proprietà di sistema os400.runtime.exec

È possibile impostare il valore della proprietà di sistema os400.runtime.exec su EXEC (valore predefinito) o QSHELL. Il valore di os400.runtime.exec determina se java.lang.Runtime.exec() utilizza l'interfaccia EXEC o Qshell.

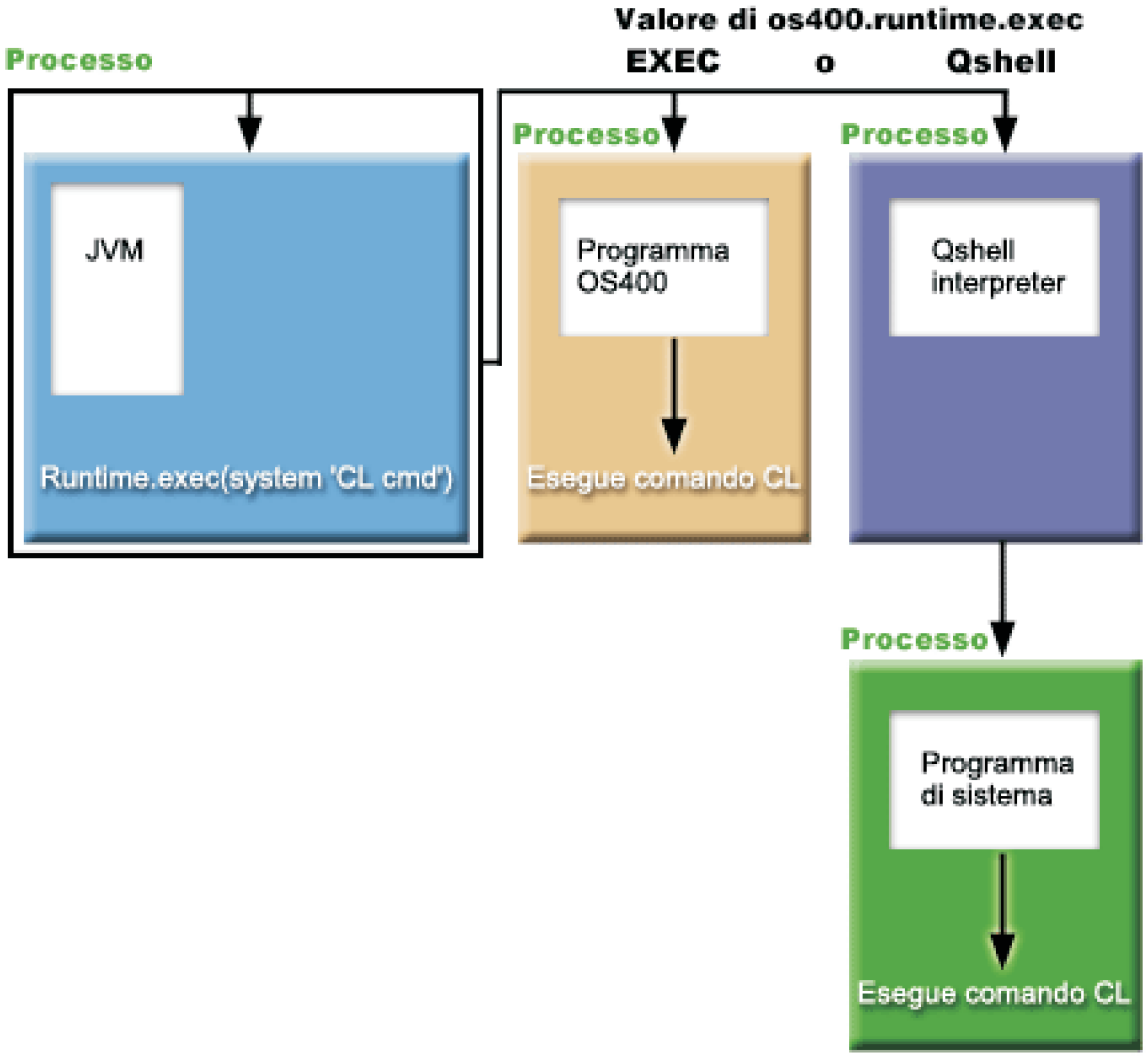
Rispetto a QSHELL, il valore EXEC presenta i seguenti vantaggi:

- Un programma Java che richiami java.lang.Runtime.exec() è più facilmente trasferibile
- L'utilizzo di java.lang.Runtime.exec() per richiamare un comando CL comporta l'impiego di un numero minore di risorse di sistema

È consigliabile utilizzare java.lang.Runtime.exec() per eseguire Qshell solo quando lo richiede la compatibilità con le versioni precedenti. Se si utilizza java.lang.Runtime.exec() per eseguire Qshell è necessario impostare os400.runtime.exec su QSHELL.

Come mostra l'illustrazione di seguito riportata, quando si utilizza QSHELL viene lanciato un terzo lavoro con conseguente consumo di ulteriori risorse di sistema. Occorre notare che il valore QSHELL diminuisce la trasferibilità del programma Java.

Figura 1. Utilizzare un valore QSHELL per la proprietà di sistema os400.runtime.exec



Inoltre quando si utilizza questo valore è necessaria una sintassi specifica per trasferire un comando CL a java.lang.Runtime.exec(). Per ulteriori informazioni consultare il seguente esempio su come richiamare un comando CL.

Per informazioni su come impostare os400.runtime.exec, fare riferimento a Elenco delle proprietà di sistema Java .

Esempio: richiamo di un altro programma Java con java.lang.Runtime.exec()

Questo esempio descrive come chiamare un altro programma Java con java.lang.Runtime.exec(). Questa classe richiama il programma Hello fornito come parte di IBM Developer Kit per Java. Quando la classe Hello scrive su System.out, questo programma ottiene un handle al flusso e può leggere da quest'ultimo.

Nota: viene utilizzato Qshell Interpreter per richiamare il programma.

Esempio 1: classe CallHelloPgm

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallHelloPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallHelloPgm.main() invoked");

      // chiamare la classe Hello
      try
      {
          theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }

      // leggere dal flusso di emissione standard del programma chiamato
      try
      {
         inStream = new BufferedReader(
                           new InputStreamReader( theProcess.getInputStream() ));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error on inStream.readLine()");
         e.printStackTrace();
      }

   } // end method

} // end class
```

Esempio: richiamo di un programma CL con java.lang.Runtime.exec()

Questo esempio mostra come eseguire i programmi CL dall'interno di un programma Java. In questo esempio, la classe Java class CallCLPgm esegue un programma CL.

Il programma CL utilizza il comando DSPJVAPGM (Visualizzazione programma Java) per visualizzare il programma associato al file di classe Hello. Questo esempio presume che il programma CL sia stato compilato ed esista in una libreria denominata JAVSAMPLIB. L'emissione dal programma CL è nel file di spool QSYSPRT.

Consultare "Esempio: richiamo di un comando CL con java.lang.Runtime.exec()" per un esempio di come richiamare un comando CL dall'interno di un programma Java.

Nota: JAVSAMPLIB non viene creato come parte del processo di installazione del programma su licenza (LP - licensed program) IBM Developer Kit numero 5761-JV1. È necessario creare la libreria esplicitamente.

Esempio 1: classe CallCLPgm

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallCLPgm
{
   public static void main(String[] args)
   {
      try
      {
         Process theProcess =
                 Runtime.getRuntime().exec("/QSYS.LIB/JAVSAMPLIB.LIB/DSPJVA.PGM");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }
   }  // end main() method
} // end class
```

Esempio 2: visualizzazione del programma CL Java

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
               OUTPUT(*PRINT)
ENDPGM
```

Esempio: richiamo di un comando CL con java.lang.Runtime.exec()

Questo esempio mostra come eseguire un comando CL (Control Language) dall'interno di un programma Java.

In questo esempio, la classe Java esegue un comando CL. Il comando CL utilizza il comando CL DSPJVAPGM (Visualizzazione programma Java) per visualizzare il programma associato al file di classe Hello. L'emissione dal comando CL è nel file di spool QSYSPRT.

Quando si imposta la proprietà di sistema os400.runtime.exec su EXEC (valore predefinito), i comandi che si immettono nella funzione `Runtime.getRuntime().exec()` utilizzano il seguente formato:

```
Runtime.getRuntime()Exec("system CLCOMMAND");
```

dove *CLCOMMAND* è il comando CL che si desidera eseguire.

Nota: quando si imposta os400.runtime.exec su QSHELL, è necessario aggiungere barra e virgolette (\"). Ad esempio il comando precedente appare in questo modo:

```
Runtime.getRuntime()Exec("system \"CLCOMMAND\"");
```

Esempio: classe per richiamare un comando CL

Il seguente codice implica l'utilizzo del valore predefinito di EXEC per la proprietà di sistema os400.runtime.exec.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallCLCom
{
   public static void main(String[] args)
   {
      try
      {
         Process theProcess =
            Runtime.getRuntime().exec("system DSPJVAPGM CLSF('/com/ibm/as400/system/Hello.class')
                                      OUTPUT(*PRINT)");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }
   }  // end main() method
} // end class
```

Concetti correlati

"Utilizzo di java.lang.Runtime.exec()" a pagina 229
Utilizzare il metodo java.lang.Runtime.exec per richiamare comandi o programmi dall'interno di un programma Java. Il metodo java.lang.Runtime.exec() consente di creare uno o più lavori aggiuntivi abilitati a sottoprocessi. Tali lavori elaborano la stringa dei comandi trasmessa al metodo.

"Elenco delle proprietà di sistema Java" a pagina 16
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Comunicazioni tra processi

Quando si comunica con programmi in esecuzione su un altro processo, esistono varie opzioni.

Un'opzione consiste nell'utilizzare i socket per le comunicazioni tra processi. Un programma può funzionare come programma server in ascolto su un collegamento socket per l'immissione dal programma client. Il programma client si collega al server con un socket. Una volta stabilito il collegamento socket, il programma può inviare o ricevere informazioni.

Un'altra opzione consiste nell'utilizzare i file di flusso per le comunicazioni tra programmi. Per fare ciò, utilizzare le classi System.in, System.out e System.err.

Una terza opzione consiste nell'utilizzare IBM Toolbox per Java, che fornisce le code di dati e gli oggetti dei messaggi System i5.

È anche possibile richiamare Java da altri linguaggi, come dimostrato nei seguenti esempi.

Informazioni correlate

IBM Toolbox per Java

Utilizzo dei socket per le comunicazioni tra processi

I flussi di socket comunicano tra i programmi in esecuzione su processi separati.

I programmi possono essere avviati separatamente oppure utilizzando il metodo java.lang.Runtime.exec() dall'interno del programma principale Java. Se un programma è scritto in un linguaggio diverso da Java, è necessario assicurarsi che si verifichi qualsiasi conversione ASCII (American Standard Code for Information Interchange) o EBCDIC (extended binary-coded decimal interchange code). Consultare le sezioni relative alle codifiche di caratteri Java per ulteriori dettagli.

Concetti correlati

"Utilizzo di java.lang.Runtime.exec()" a pagina 229
Utilizzare il metodo java.lang.Runtime.exec per richiamare comandi o programmi dall'interno di un programma Java. Il metodo java.lang.Runtime.exec() consente di creare uno o più lavori aggiuntivi abilitati a sottoprocessi. Tali lavori elaborano la stringa dei comandi trasmessa al metodo.

"Codifiche del carattere Java" a pagina 26
I programmi Java possono convertire i dati in diversi formati, consentendo alle applicazioni di trasferire e utilizzare informazioni da diversi tipi di serie di caratteri internazionali.

Esempio: utilizzo dei socket per la comunicazione tra processi:

Questo esempio utilizza i socket per comunicare tra un programmaJava e un programma C.

È necessario avviare prima il programma C, in ascolto su un socket. Una volta che il programma Java si collega al socket, il programma C invia a questo una stringa utilizzando quel collegamento del socket. La stringa inviata dal programma C rappresenta una stringa ASCII (American Standard Code for Information Interchange) nella codepage 819.

Il programma Java deve essere avviato utilizzando questo comando, `java TalkToC xxxxx nnnn` sulla riga comandi Qshell Interpreter o su un'altra piattaforma Java. Altrimenti immettere `JAVA TALKTOC PARM(xxxxx nnnn)` sulla riga comandi i5/OS per avviare il programma Java. `xxxxx` rappresenta il nome del dominio o l'indirizzo IP (Internet Protocol) del sistema sul quale il programma C è in esecuzione. `nnnn` rappresenta il numero della porta del socket che il programma C sta utilizzando. È necessario utilizzare inoltre questo numero porta come primo parametro sulla chiamata al programma C.

Esempio 1: classe client TalkToC

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.net.*;
import java.io.*;

class TalkToC
{
   private String host = null;
   private int port = -999;
   private Socket socket = null;
   private BufferedReader inStream = null;


   public static void main(String[] args)
   {
      TalkToC caller = new TalkToC();
      caller.host = args[0];
      caller.port = new Integer(args[1]).intValue();
      caller.setUp();
      caller.converse();
      caller.cleanUp();

   }  // end main() method

   public void setUp()
   {
      System.out.println("TalkToC.setUp() invoked");
```

```
      try
      {
         socket = new Socket(host, port);
         inStream = new BufferedReader(new InputStreamReader(
                                     socket.getInputStream()));
      }
      catch(UnknownHostException e)
      {
         System.err.println("Cannot find host called: " + host);
         e.printStackTrace();
         System.exit(-1);
      }
      catch(IOException e)
      {
         System.err.println("Could not establish connection for " + host);
         e.printStackTrace();
         System.exit(-1);
      }
   } // end setUp() method

   public void converse()
   {
      System.out.println("TalkToC.converse() invoked");

      if (socket != null && inStream != null)
      {
         try
         {
            System.out.println(inStream.readLine());
         }
         catch(IOException e)
         {
            System.err.println("Conversation error with host " + host);
            e.printStackTrace();
         }

      } // end if

   }  // end converse() method

   public void cleanUp()
   {
      try
      {

         if(inStream != null)
         {
            inStream.close();
         }
         if(socket != null)
         {
            socket.close();
         }
      } // end try
      catch(IOException e)
      {
         System.err.println("Error in cleanup");
         e.printStackTrace();
         System.exit(-1);
      }
   } // end cleanUp() method

} // end TalkToC class
```

SockServ.C viene avviato con l'inoltro in un parametro relativo al numero di porta. Ad esempio, `CALL SockServ '2001'`.

Esempio 2: programma del server SockServ.C

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/time.h>

void main(int argc, char* argv[])
{
   int    portNum = atoi(argv[1]);
   int    server;
   int    client;
   int    address_len;
   int    sendrc;
   int    bndrc;
   char*  greeting;
   struct sockaddr_in  local_Address;
   address_len = sizeof(local_Address);

   memset(&local_Address,0x00,sizeof(local_Address));
   local_Address.sin_family = AF_INET;
   local_Address.sin_port = htons(portNum);
   local_Address.sin_addr.s_addr = htonl(INADDR_ANY);

   #pragma convert (819)
   greeting = "This is a message from the C socket server.";
   #pragma convert (0)

   /*  allocate socket    */
   if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
   {
      printf("failure on socket allocation\n");
      perror(NULL);
      exit(-1);
   }

   /* do bind   */
   if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
   {
     printf("Bind failed\n");
     perror(NULL);
     exit(-1);
   }

   /* invoke listen   */
   listen(server, 1);

   /* wait for client request */
   if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
   {
     printf("accept failed\n");
     perror(NULL);
     exit(-1);
   }


   /* send greeting to client    */
   if((sendrc = send(client, greeting, strlen(greeting),0))<0)
   {
      printf("Send failed\n");
```

```
      perror(NULL);
      exit(-1);
   }

   close(client);
   close(server);

}
```

Utilizzo dei flussi di immissione ed emissione per la comunicazione tra processi

I flussi di immissione ed emissione comunicano tra programmi che sono in esecuzione in processi separati.

Il metodo java.lang.Runtime.exec() esegue un programma. Il programma principale può richiamare handle nei flussi di emissione e immissione del processo secondario e può registrare in tali flussi o leggere da essi. Se il programma secondario è scritto in un linguaggio diverso da Java, è necessario assicurare che avvenga una conversione ASCII (American Standard Code for Information Interchange) o EBCDIC (extended binary-coded decimal interchange code). Consultare le codifiche di caratteri Java per ulteriori dettagli.

Concetti correlati

"Utilizzo di java.lang.Runtime.exec()" a pagina 229
Utilizzare il metodo java.lang.Runtime.exec per richiamare comandi o programmi dall'interno di un programma Java. Il metodo java.lang.Runtime.exec() consente di creare uno o più lavori aggiuntivi abilitati a sottoprocessi. Tali lavori elaborano la stringa dei comandi trasmessa al metodo.

"Codifiche del carattere Java" a pagina 26
I programmi Java possono convertire i dati in diversi formati, consentendo alle applicazioni di trasferire e utilizzare informazioni da diversi tipi di serie di caratteri internazionali.

Esempio: utilizzo dei flussi di immissione ed emissione per la comunicazione tra processi:

Questo esempio mostra come chiamare un programma C da Java e utilizzare flussi di immissione ed emissione per la comunicazione tra processi.

In questo esempio, il programma C registra una stringa sul relativo flusso di emissione standard e il programma Java legge questa stringa e la visualizza. Questo esempio presume che sia stata creata una libreria denominata JAVSAMPLIB e che sia stato creato il programma CSAMP1 al suo interno.

Nota: JAVSAMPLIB non viene creato come parte del processo di installazione del programma su licenza (LP - licensed program) IBM Developer Kit numero 5761-JV1. È necessario crearlo esplicitamente.

Esempio 1: classe CallPgm

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallPgm.main() invoked");

      // chiamare il programma CSAMP1
      try
      {
          theProcess = Runtime.getRuntime().exec(
```

```
                    "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }

      // leggere dal flusso di emissione standard del programma chiamato
      try
      {
         inStream = new BufferedReader(new InputStreamReader
                         (theProcess.getInputStream()));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error on inStream.readLine()");
         e.printStackTrace();
      }

   } // end method

 } // end class
```

Esempio 2: programma C CSAMP1

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
     /* Convertire la stringa in ASCII nel tempo di compilazione */
#pragma convert(819)
     printf("Program JAVSAMPLIB/CSAMP1 was invoked\n");
#pragma convert(0)
     /* Stdout può essere memorizzato nel buffer, quindi svuotare il buffer */

     fflush(stdout);

}
```

Esempio: richiamo di Java da C

Questo è un esempio di un programma C che utilizza la funzione system() per chiamare il programma Hello Java.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
#include <stdlib.h>

 int main(void)
 {
   int result;

 /* La funzione di sistema passa la stringa fornita al processore comandi CL
    per l'elaborazione. */

   result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
 }
```

Esempio: richiamo di Java da RPG

Questo è un esempio di un programma RPG che utilizza l'API QCMDEXC per chiamare il programma Hello Java.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle “Informazioni sull’esonero di responsabilità e licenza del codice” a pagina 580.

```
D*             DEFINE  THE PARAMETERS FOR THE QCMDEXC API
D*
DCMDSTRING        S             25    INZ('JAVA CLASS(''com.ibm.as400.system.Hello'')')
DCMDLENGTH        S             15P 5 INZ(25)
D*             NOW THE CALL TO QCMDEXC WITH THE 'JAVA' CL COMMAND
C                   CALL      'QCMDEXC'
C                   PARM                    CMDSTRING
C                   PARM                    CMDLENGTH
C*       This next line displays 'DID IT' after you exit the
C*       Java Shell via F3 or F12.
C     'DID IT'      DSPLY
C*       Set On LR to exit the RPG program
C                   SETON                                        LR
C
```

Piattaforma Java

La piattaforma Java è l’ambiente per lo sviluppo e la gestione delle applicazioni e applet Java. Questa consiste di tre componenti principali: il linguaggio Java, i pacchetti Java e la Java virtual machine.

Il linguaggio e i pacchetti Java sono simili a C++ e alle relative librerie di classe. I pacchetti Java contengono classi, disponibili in qualsiasi implementazione Java compatibile. L’API (application programming interface) deve essere uguale su qualsiasi sistema che supporta Java.

Java differisce da un linguaggio tradizionale, come C++, nel modo in cui esegue la compilazione e l’esecuzione. In un ambiente di programmazione tradizionale, il codice sorgente di un programma viene scritto e compilato nel codice dell’oggetto per uno specifico hardware e sistema operativo. Il codice dell’oggetto si collega agli altri moduli di codice dell’oggetto per creare un programma di esecuzione. Il codice risulta specifico per una serie specifica di hardware del computer che non può essere in esecuzione su altri sistemi senza essere modificato. Questa figura illustra l’ambiente di disposizione del linguaggio tradizionale.

Applicazioni e applet Java

Un’applet è un programma Java progettato per essere incluso in un documento Web HTML. È possibile scrivere l’applet Java e includerla in una pagina HTML, nello stesso modo in cui viene inclusa un’immagine. Quando si utilizza un browser abilitato per Java per visualizzare una pagina HTML che contiene un’applet, il codice dell’applet viene trasferito al sistema ed è eseguito dalla Java virtual machine del browser.

Il documento HTML contiene tag, che specificano il nome dell’applet Java e il relativo URL (Uniform Resource Locator). L’URL è l’ubicazione in cui si trovano i bytecode dell’applet su Internet. Quando viene visualizzato un documento HTML contenente una tag applet Java, il browser Web abilitato a Java scarica i bytecode Java da Internet e utilizza la JVM (Java virtual machine) per elaborare il codice dall’interno del documento Web. Queste applet Java abilitano le pagine Web per contenere una parte interattiva o una grafica animata.

È possibile inoltre scrivere un’applicazione Java che non richiede l’utilizzo di un browser web.

Per ulteriori informazioni, consultare Writing Applets, supporto didattico di Sun Microsystems per le applet Java. Esso include una panoramica delle applet, indicazioni per scrivere le applet e alcuni problemi applet comuni.

Le **Applicazioni** sono programmi autonomi che non richiedono l’utilizzo di un browser. Le applicazioni Java vengono eseguite attivando l’interpreter Java dalla riga comandi e specificando il file che contiene

l'applicazione compilata. Le applicazioni solitamente risiedono sul sistema su cui vengono sviluppate. Le applicazioni accedono alle risorse sul sistema e sono limitate dal modello di sicurezza Java .

JVM (Java virtual machine)

La JVM (Java virtual machine) è un ambiente del tempo di esecuzione che è possibile aggiungere in un browser web o in qualsiasi sistema operativo, come IBM i5/OS. La JVM (Java virtual machine) esegue istruzioni generate da un compilatore Java. Essa consiste in un interpreter bytecode e tempo di esecuzione che consente di eseguire i file di classe Java su qualsiasi piattaforma, indipendentemente dalla piattaforma su cui sono stati sviluppati in origine.

Il programma di caricamento classi e il responsabile della riservatezza, che fanno parte del tempo di esecuzione Java, isolano il codice che proviene da un'altra piattaforma. Essi possono anche limitare le risorse di sistema cui può accedere ogni classe caricata.

Nota: le applicazioni Java non vengono limitate; la limitazione riguarda soltanto le applet. Le applicazioni possono accedere liberamente alle risorse di sistema e utilizzare i metodi nativi. La maggior parte dei programmi di IBM Developer Kit per Java sono applicazioni.

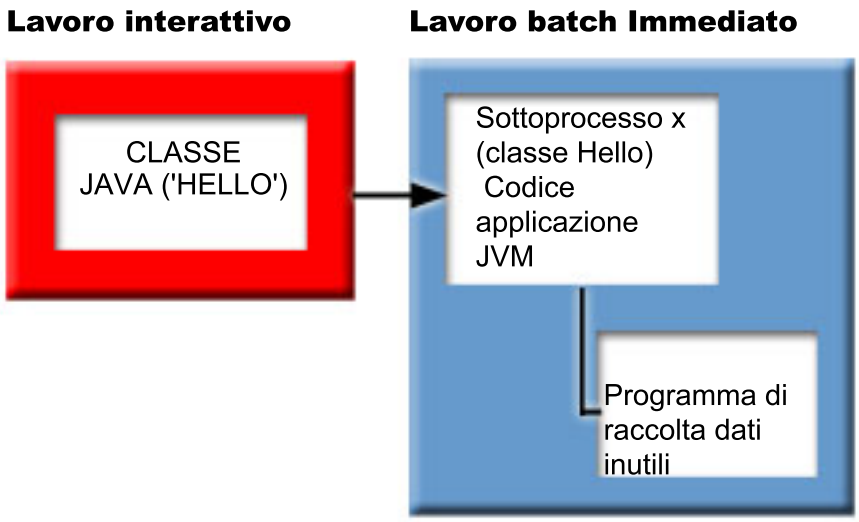
È possibile utilizzare il comando CRTJVAPGM (Creazione programma Java) per assicurare che il codice rispetti i requisiti di sicurezza che il tempo di esecuzione Java impone per verificare i bytecode. Ciò include il controllo di limitazioni del tipo, il controllo di conversioni di dati, l'assicurarsi che non si verifichi un'eccedenza o un'insufficienza dello stack del parametro e il controllo delle violazioni di accesso. Tuttavia, non è necessario verificare esplicitamente i bytecode. Se non si utilizza il comando CRTJVAPGM in anticipo, i controlli si verificano durante il primo utilizzo di una classe. Una volta verificati i bytecode, l'interpreter li decodifica ed esegue le istruzioni macchina necessarie per effettuare le operazioni desiderate.

Oltre caricamento e all'esecuzione dei bytecode, la JVM (Java virtual machine) include un programma di raccolta dati inutili che gestisce la memoria. La "Raccolta di dati inutili Java" a pagina 416 viene eseguita nello stesso momento del caricamento e dell'interpretazione dei bytecode.

JRE (Java runtime environment)

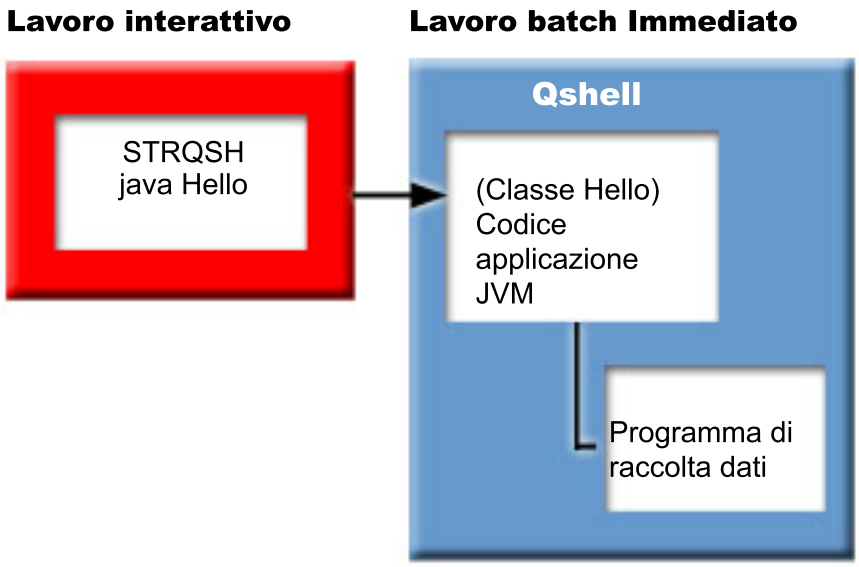
L'ambiente di tempo di esecuzione Java viene avviato ogni qualvolta si immette il comando RUNJVA (Esecuzione Java) oppure il comando JAVA sulla riga comandi i5/OS. Dal momento che l'ambiente Java dispone di più sottoprocessi, è necessario eseguire la JVM (Java virtual machine) su un lavoro che supporti i sottoprocessi, come il lavoro BCI (batch immediato). Come mostra la seguente figura, una volta avviata la JVM (Java virtual machine), è possibile avviare ulteriori sottoprocessi su cui viene eseguito il programma di raccolta di dati inutili.

Figura 1: il tipico ambiente Java quando si utilizza il comando CL RUNJVA o JAVA



È inoltre possibile avviare JRE (Java runtime environment) utilizzando il comando `java` in Qshell dal Qshell Interpreter. In questo ambiente, Qshell Interpreter è in esecuzione su un lavoro BCI associato ad un lavoro interattivo. JRE (Java runtime environment) viene avviato sul lavoro su cui è in esecuzione Qshell Interpreter.

Figura 2: ambiente Java environment quando si utilizza il comando java in Qshell



Quando JRE (Java runtime environment) viene avviato da un lavoro interattivo, viene visualizzato il pannello Java Shell. Tale pannello fornisce una riga di immissione per immettere i dati nel flusso System.in e per visualizzare i dati registrati nel flusso System.out e System.err.

Interprete Java

L'interpreter Java fa parte della JVM (Java virtual machine) che interpreta i file di classe Java per una specifica piattaforma hardware. L'interpreter Java decodifica ogni bytecode ed esegue l'operazione corrispondente.

Esecuzione di un programma PASE i5/OS con QP2TERM()

"Funzioni dell'API di richiamo" a pagina 213
IBM Developer Kit per Java supporta queste funzioni dell'API di richiamo.

File di classe e JAR Java

Un file JAR (Java ARchive) è un formato file che combina più file in uno. L'ambiente Java differisce dagli altri ambienti di programmazione per il fatto che il compilatore Java non crea un codice macchina per una serie di istruzioni specifiche per l'hardware. Il compilatore Java, invece, converte il codice sorgente Java in istruzioni JVM (Java virtual machine), che i file di classe Java memorizzano. È possibile utilizzare i file JAR per memorizzare i file di classe. Il file di classe non ha come destinazione una piattaforma hardware specifica, ma al contrario ha come destinazione la struttura della JVM (Java virtual machine).

È possibile utilizzare JAR come strumento di archiviazione generale e anche per distribuire i programmi Java di tutti i tipi, incluse le applet. Le applet Java vengono scaricate in un browser in una singola transazione HTTP (Hypertext Transfer Protocol) piuttosto che aprendo un nuovo collegamento per ogni elemento. Tale metodo di scaricare incrementa la velocità con cui l'applet viene caricata su una pagina Web e inizia a funzionare.

JAR è l'unico formato di archivio che è a piattaforma incrociata. JAR è inoltre l'unico formato che gestisce i file audio e immagine, così come i file di classe. JAR è un formato completamente estensibile, standard aperto scritto in Java.

Il formato JAR supporta inoltre la compressione, che riduce la dimensione del file e il tempo di scaricamento. In aggiunta, un creatore di applet può apporre la firma digitale su singole voci in un file JAR per autenticarne l'origine.

Per aggiornare le classi nei file JAR, utilizzare lo strumento `jar`.

I **file di classe Java** sono file di flusso prodotti quando un file di origine viene compilato dal compilatore Java. Il file di classe contiene tabelle che descrivono ogni campo e metodo della classe. Il file contiene inoltre i bytecode per ogni metodo, dati statici e descrizioni utilizzate per rappresentare gli oggetti Java.

Informazioni correlate

Strumento jar Java di Sun Microsytems, Inc.

Sottoprocessi Java

Un sottoprocesso è un flusso singolo indipendente che viene eseguito all'interno di un programma. Java è un linguaggio di programmazione con più sottoprocessi, per cui è possibile eseguire più di un sottoprocesso all'interno della JVM (Java virtual machine) alla volta. I sottoprocessi Java forniscono la possibilità a un programma Java di eseguire più attività contemporaneamente. Un sottoprocesso è essenzialmente un flusso di controllo in un programma.

I sottoprocessi rappresentano una struttura di programmazione moderna e sono utilizzati per supportare programmi simultanei e per migliorare le prestazioni e la scalabilità delle applicazioni. La maggior parte dei linguaggi di programmazione supportano i sottoprocessi tramite l'utilizzo di librerie di programmazione aggiunte. Java supporta i sottoprocessi come le API (application program interface) incorporate.

Nota: l'utilizzo dei sottoprocessi fornisce il supporto per aumentare l'interattività, nel senso di un'attesa minore alla tastiera perché più attività sono in esecuzione in parallelo. Tuttavia il programma non è necessariamente più interattivo solo perché possiede dei sottoprocessi.

I sottoprocessi sono il meccanismo per l'attesa su lunghe interazioni in esecuzione, mentre viene consentito ancora al programma di gestire altri lavori. I sottoprocessi hanno la capacità di supportare più flussi tramite lo stesso flusso di codice. Questi vengono a volte denominati **processi leggeri**. Il linguaggio

Java include un supporto diretto ai sottoprocessi. Tuttavia, per progettazione, non supporta l'immissione e l'emissione asincrona non vincolante con interruzioni e più attese.

I sottoprocessi consentono lo sviluppo di programmi paralleli che si adattano bene in un ambiente nel quale una macchina ha più processori. Se creati in modo appropriato, questi forniscono inoltre un modello per la gestione di più transazioni e utenti.

È possibile utilizzare i sottoprocessi in un programma Java per numerose situazioni. È necessario che alcuni programmi siano in grado di impegnarsi in più attività e siano ancora in grado di rispondere all'immissione ulteriore da parte dell'utente. Ad esempio, un browser Web deve essere in grado di rispondere all'immissione dell'utente mentre sta emettendo dei suoni.

È possibile inoltre che i sottoprocessi utilizzino metodi asincroni. Quando viene chiamato un secondo metodo, non è necessario attendere il completamento del primo prima che il secondo metodo continui con l'attività.

Esistono anche molte ragioni per non utilizzare i sottoprocessi. Se un programma utilizza la logica sequenziale in modo inerente, è possibile che un sottoprocesso realizzi l'intera sequenza. L'utilizzo di più sottoprocessi in tali casi determina un programma complesso senza alcun vantaggio. È necessario molto lavoro per la creazione e l'avvio di un sottoprocesso. Se un'operazione implica solo poche istruzioni, la gestione di essa in un singolo sottoprocesso è più veloce. Questo risulta vero persino quando l'operazione è concettualmente asincrona. Quando più sottoprocessi condividono oggetti, è necessario che gli oggetti siano sincronizzati all'accesso coordinato al sottoprocesso e che mantengano la coerenza. La sincronizzazione aggiunge complessità a un programma, diventa difficile ottimizzare prestazioni ottimali ed può essere fonte di errori di programmazione.

Per ulteriori informazioni sui sottoprocessi, consultare Sviluppare applicazioni con più sottoprocessi.

Sun Microsystems, Inc. Java Development Kit

Il JDK (Java Development Kit) è un software distribuito da Sun Microsystems, Inc. per gli sviluppatori Java. Esso include l'interpreter Java, le classi Java e gli strumenti di sviluppo Java: il compilatore, il programma di debug, il programma di disassemblaggio, l'appletviewer, il generatore di file stub e il generatore di documentazione.

Il JDK consente la scrittura di applicazioni che sono state sviluppate una volta e di effettuare l'esecuzione dovunque su qualunque Java virtual machine. Le applicazioni Java sviluppate con JDK su un sistema possono essere utilizzate su un altro sistema senza modificare o ricompilare il codice. I file di classe Java sono trasferibili su qualsiasi Java virtual machine standard.

Per trovare ulteriori informazioni sul JDK corrente, controllare la versione di IBM Developer Kit per Java sul server.

È possibile verificare la versione del IBM Developer Kit per JVM (Java Java virtual machine) predefinito sul server immettendo uno dei seguenti comandi:

- `java -version` sulla richiesta comandi Qshell.
- `RUNJVA CLASS(*VERSION)` sulla riga comandi CL.

Cercare quindi la stessa versione di Sun Microsystems, Inc. JDK su The Source for Java Technology java.sun.com per la documentazione specifica. IBM Developer Kit per Java è un'implementazione compatibile di Sun Microsystems, Inc. Java Technology ed occorre pertanto avere dimestichezza con la relativa documentazione JDK.

Pacchetti Java

Un pacchetto Java è un modo per raggruppare classi e interfacce correlate in Java. I pacchetti Java sono simili alle librerie di classi disponibili in altri linguaggi.

I pacchetti Java, che forniscono le API Java, sono disponibili come parte di Sun Microsystems, Inc. JDK (Java Development Kit). Per un elenco completo di pacchetti Java e per informazioni sulle API Java, consultare Pacchetti piattaforma Java 2.

Strumenti Java

Per un elenco completo degli strumenti forniti da Sun Microsystems, Inc. Java Development Kit, consultare la Guida di riferimento agli strumenti di Sun Microsystems, Inc. Per ulteriori informazioni su ogni singolo strumento supportato da IBM Developer Kit for Java, consultare Strumenti Java supportati da IBM Developer Kit per Java.

"Supporto per più opzioni 5761-JV1 LP" a pagina 6
La piattaforma System i5 supporta più versioni dei JDK (Java Development Kit) e di J2SE (Java 2 Platform, Standard Edition).

"Metodi nativi e JNI (Java Native Interface)" a pagina 219
I metodi nativi sono quei metodi Java che si avviano in un linguaggio diverso da Java. È possibile che i metodi nativi accedano alle funzioni e alle API specifiche per il sistema che non sono disponibili direttamente in Java.

Pacchetti piattaforma Java 2

Guida di riferimento agli strumenti di Sun Microsystems, Inc.

"Strumenti Java supportati da IBM Developer Kit per Java" a pagina 420
L'ambiente Qshell include gli strumenti di sviluppo Java normalmente richiesti per lo sviluppo di programmi.

Argomenti avanzati

Questo argomento fornisce istruzioni su come eseguire Java in un lavoro batch e descrive le autorizzazioni ai file Java necessarie nell'IFS (integrated file system) per visualizzare, eseguire o effettuare il debug di un programma Java.

Classi, pacchetti e indirizzari Java

Ogni classe Java fa parte di un pacchetto. La prima istruzione in un file di origine Java indica in quale pacchetto si trova una classe. Se il file di origine non contiene un'istruzione del pacchetto, la classe fa parte di un pacchetto predefinito non denominato.

Il nome del pacchetto fa riferimento alla struttura dell'indirizzario in cui si trova la classe. L'IFS (Integrated File System) supporta le classi Java in una struttura file gerarchica che è simile a quanto si trova nella maggior parte dei PC e sistemi UNIX. È necessario memorizzare una classe Java in un indirizzario con un percorso indirizzario relativo che corrisponda al nome del pacchetto per tale classe. Ad esempio, considerare la seguente classe Java:

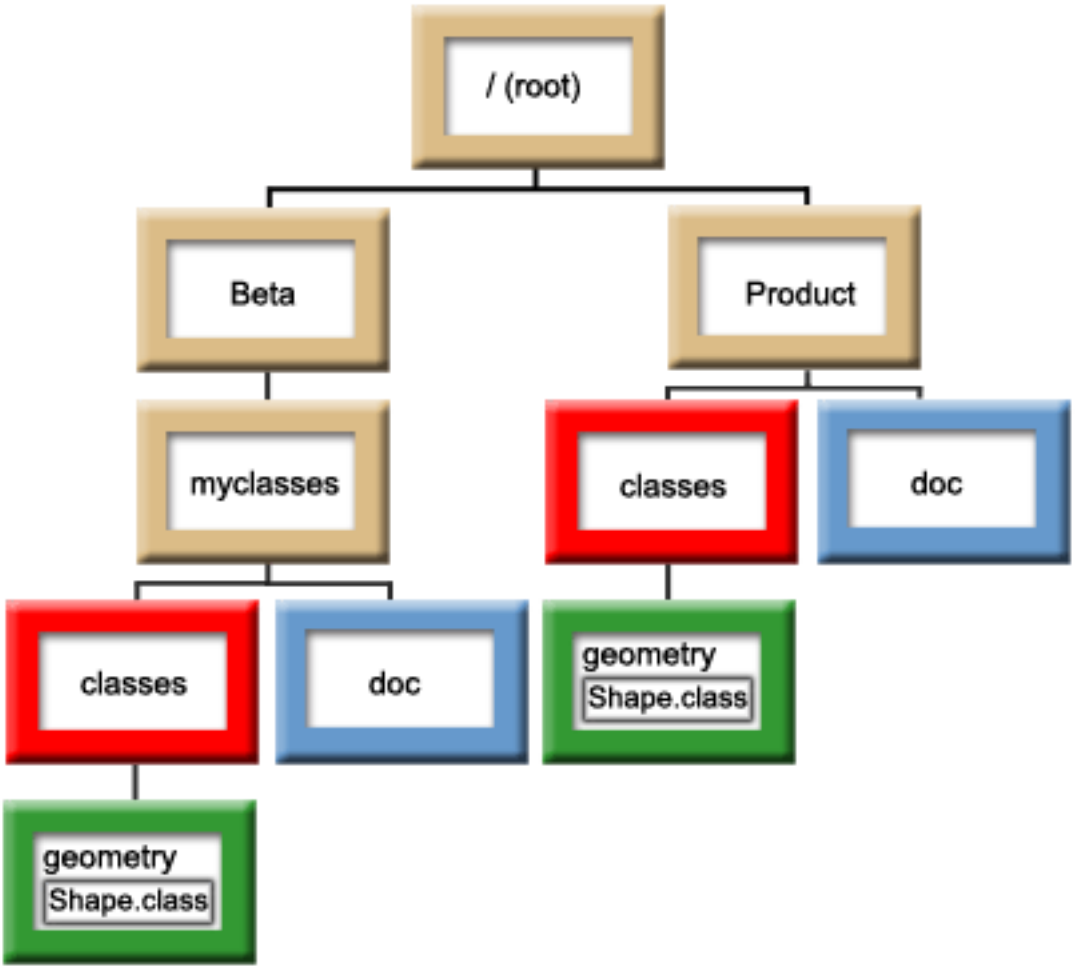
```
package classes.geometry;
import java.awt.Dimension;

public class Shape {

Dimension metrics;

   // L'implementazione per la classe Shape verrà codificata qui ...

}
```

L'istruzione del pacchetto nel codice precedente indica che la classe Shape fa parte del pacchetto classes.geometry. Affinché il tempo di esecuzione Java trovi la classe Shape, memorizzare tale classe nella struttura dell'indirizzario relativo classes/geometry.

Nota: il nome del pacchetto corrisponde al nome dell'indirizzario relativo in cui risiede la classe. Il programma di caricamento classi JVM (Java virtual machine) trova la classe accodando il nome percorso relativo ad ogni indirizzario specificato nel classpath. Inoltre il programma di caricamento classi JVM (Java virtual machine) trova la classe ricercandole nei file ZIP o JAR specificati nel classpath.

Ad esempio quando si memorizza la classe Shape nell'indirizzario /Product/classes/geometry nel file system ″root″ (/), è necessario specificare /Product nel classpath.

Figura 1: esempio di struttura dell'indirizzario per classi Java che hanno lo stesso nome in pacchetti diversi



Nota: è possibile che esistano più versioni della classe Shape nella struttura dell'indirizzario. Per utilizzare la versione Beta della classe Shape, inserire /Beta/myclasses nel classpath prima di qualsiasi altro indirizzario o file ZIP che contenga la classe Shape.

Il compilatore Java utilizza il classpath Java, il nome del pacchetto e la struttura dell'indirizzario per trovare pacchetti e classi quando compila il codice sorgente Java. Per ulteriori informazioni, consultare "Classpath Java" a pagina 13.

File correlati a Java nell'IFS

L'IFS (integrated file system) memorizza i file JAR, ZIP, di origine e di classe correlati a Java in una struttura gerarchica di file. L'IBM Developer Kit per Java supporta l'utilizzo di file system sicuri nell'IFS per memorizzare e gestire i file JAR, i file ZIP, i file di origine e i file di classe correlati a Java.

Informazioni correlate

Considerazioni sui file system per programmazione a sottoprocessi multipli

Confronto di file system

Autorizzazioni file Java nell'IFS (integrated file system).

Per eseguire o effettuare il debug di un programma Java è necessario che il file di classe, il file JAR o il file ZIP dispongano dell'autorizzazione alla lettura (*R). Gli indirizzari devono disporre delle autorizzazioni alla lettura e all'esecuzione (*RX).

Per utilizzare il comando CRTJVAPGM (Creazione programma Java) per ottimizzare un programma, è necessario che il file di classe, il file JAR o il file ZIP dispongano dell'autorizzazione alla lettura (*R) e che l'indirizzario disponga dell'autorizzazione all'esecuzione (*X). Se si utilizza un modello nel nome file di classe, è necessario che l'indirizzario disponga dell'autorizzazione alla lettura e all'esecuzione (*RX).

Per cancellare un programma Java utilizzando il comando DLTJVAPGM (Cancellazione programma Java), è necessario disporre dell'autorizzazione alla scrittura e lettura (*RW) al file di classe e che l'indirizzario disponga dell'autorizzazione all'esecuzione (*X). Se si utilizza un modello nel nome file di classe, è necessario che l'indirizzario disponga dell'autorizzazione alla lettura e all'esecuzione (*RX).

Per visualizzare un programma Java utilizzando il comando DSPJVAPGM (Visualizzazione programma Java), è necessario disporre dell'autorizzazione alla lettura (*R) al file di classe e che l'indirizzario disponga dell'autorizzazione all'esecuzione (*X).

Nota: i file e gli indirizzari che non dispongono dell'autorizzazione all'esecuzione (*X) vengono sempre visualizzati con l'autorizzazione all'esecuzione (*X) ad un utente con autorizzazione QSECOFR. Differenti utenti possono ottenere risultati differenti in alcune situazioni, anche se entrambi gli utenti sembrano avere lo stesso accesso agli stessi file. È importante sapere ciò quando si eseguono script shell utilizzando Qshell Interpreter o java.Runtime.exec().

Ad esempio, un utente scrive un programma Java che utilizza java.Runtime.exec() per chiamare uno script shell, quindi lo sottopone a verifica utilizzando un ID utente con autorizzazione QSECOFR. Se la modalità file dello script shell dispone dell'autorizzazione alla lettura e scrittura (*RW), l'IFS (Integrated File System) consente all'ID utente con autorizzazione QSECOFR di eseguirla. Tuttavia, è possibile che un utente senza autorizzazione QSECOFR tenti di eseguire lo stesso programma Java e che l'IFS indichi al codice java.Runtime.exec() che non è possibile eseguire lo script shell, in quanto manca *X. In questo caso, java.Runtime.exec() emette un'eccezione di immissione ed emissione.

È inoltre possibile assegnare autorizzazioni ai nuovi file creati dai programmi Java in un IFS (Integrated file system). Utilizzando la proprietà di sistema os400.file.create.auth per i file e os400.dir.create.auth per gli indirizzari, è possibile impiegare qualsiasi combinazione di autorizzazioni alla lettura, alla scrittura e all'esecuzione.

Per ulteriori informazioni, consultare le API del comando CL e del programma o l'IFS (Integrated file system).

Esecuzione di Java in un lavoro batch

I programmi Java vengono eseguiti su un lavoro batch utilizzando il comando SBMJOB (Inoltro lavoro). In questa modalità, il pannello di immissione comandi Qshell Java non è disponibile a gestire i flussi System.in, System.out e System.err.

È possibile reindirizzare questi flussi su altri file. La gestione predefinita invia i flussi System.out e System.err a un file di spool. Il lavoro batch, che risulta in un'eccezione di emissione e immissione per le richieste di lettura da System.in, possiede il file di spool. È possibile reindirizzare System.in, System.out e System.err all'interno del programma Java. È possibile inoltre utilizzare le proprietà di sistema os400.stdin, os400.stdout e os400.stderr per reindirizzare System.in, System.out e System.err.

Nota: SBMJOB imposta il CWD (current working directory/indirizzario di lavoro corrente) su un indirizzario HOME specificato nel profilo utente.

Esempio: Esecuzione Java in un lavoro Batch

```
SBMJOB CMD(JAVA Hello OPTION(*VERBOSE)) CPYENVVAR(*YES)
```

L'esecuzione del comando JAVA nell'esempio precedente effettua lo spawn di un secondo lavoro. Perciò il sottosistema in cui il lavoro batch viene eseguito deve essere in grado di eseguire più di un lavoro.

È possibile verificare se il lavoro batch è in grado di eseguire più di un lavoro seguendo queste fasi:

1. Sulla riga comandi CL, immettere `DSPSBSD(MYSBSD)`, dove `MYSBSD` rappresenta la descrizione del sottosistema del lavoro batch.
2. Scegliere l'opzione 6, Specifiche della coda lavori.
3. Consultare il campo Max attivi per la coda lavori.

Esecuzione dell'applicazione Java su un host che non dispone di una GUI (graphical user interface)

Se si desidera eseguire l'applicazione Java su un host che non dispone di una GUI (graphical user interface), come ad esempio un System i5, è possibile utilizzare il NAWT (Native Abstract Windowing Toolkit).

Utilizzare il NAWT per fornire alle applicazioni e ai servlet Java la funzionalità grafica AWT completa di J2SE (Java 2 Platform, Standard Edition).

NAWT (Native Abstract Windowing Toolkit)

NAWT (Native Abstract Windowing Toolkit), più che un toolkit, è un termine creato per indicare il supporto i5/OS nativo che fornisce ai servlet e alle applicazioni Java la capacità di utilizzare le funzioni grafiche AWT (Abstract Windowing Toolkit) offerte da J2SE (Java 2 Platform, Standard Edition).

Nota: In generale, le informazioni contenute in questa sezione sono applicabili solo alla JVM Classic originale e non alla IBM Technology for Java Virtual Machine. Facendo eccezione a questa norma, le informazioni di seguito indicate per avviare e utilizzare un X-server VNC sono applicabili per entrambe le JVM, se l'applicazione utilizza la API AWT per interagire direttamente con un utente oppure usa i componenti AWT heavyweight.

Nota: Quando si utilizza JDK 1.4, NAWT non supporta i caratteri e le serie di caratteri specifici per la lingua e le locali. Quando si utilizza NAWT, assicurarsi di rispettare i seguenti requisiti:

- Utilizzare solo caratteri definiti nella serie di caratteri ISO8859-1.
- Utilizzare il file font.properties. Il file font.properties risiede nell'indirizzario /QIBM/ProdData/Java400/jdknn/lib, dove *nn* è il numero di versione del J2SE che si sta utilizzando. Non utilizzare i file font.properties.xxx, dove *xxx* è la lingua o un altro qualificativo.

Quando si utilizza JDK versione 1.5 e successive, i caratteri e le serie di caratteri utilizzati sono determinati in fase di avvio di Java dal valore della proprietà file.encoding. Se non è impostata

esplicitamente, alla proprietà file.encoding viene dato un appropriato valore predefinito basato sul CCSID del lavoro. Per ulteriori informazioni, consultare “File predefinito e valori di codifica Java” a pagina 251.

Prerequisiti dei programmi su licenza

L'utilizzo della API AWT Java richiede che sia installato IBM i5/OS PASE (Portable Application Solutions Environment). Per ulteriori informazioni, consultare la sezione Installazione di i5/OS PASE.

Installazione di i5/OS PASE

Selezione di una modalità AWT

Quando si utilizza l'AWT sono disponibili due modalità: headless e normale. Un fattore che influenza la scelta della modalità da utilizzare è la necessità di utilizzare dei componenti AWT heavyweight o meno.

Modalità headless

La modalità headless può essere utilizzata se l'applicazione Java non interagisce direttamente con un utente. Questo significa che l'applicazione Java non visualizza finestre o caselle di dialogo, non accetta immissioni da tastiera o dal mouse e non utilizza componenti AWT heavyweight. Questa modalità viene selezionata specificando la proprietà Java *java.awt.headless=true* sul richiamo Java. L'utilizzo della modalità headless rende non necessario un VNC/X-server.

Degli esempi di applicazioni che possono utilizzare la modalità headless includono:

- I servlet o altri programmi basati sul server che utilizzano la API AWT solo per creare immagini da includere in un flusso di dati restituito ad un utente remoto
- Qualsiasi programma che si limita a creare o manipolare immagini o file immagine senza effettivamente visualizzarli utilizzando componenti AWT heavyweight

Modalità normale

La modalità normale deve essere utilizzata se l'applicazione utilizza la API AWT Java per visualizzare finestre, frame, caselle di dialogo o componenti heavyweight simili. Utilizzare la modalità normale se si prevede che l'applicazione riceverà eventi di operazioni del mouse o immissioni da tastiera. Se si sta utilizzando la IBM JVM Classic, la modalità normale deve essere abilitata specificando la proprietà Java *os400.awt.native=true* sul richiamo Java. Se si sta utilizzando IBM Technology for Java Virtual Machine, la proprietà os400.awt.native non è richiesta e viene ignorata, se specificata.

Componenti AWT heavyweight

I seguenti elementi sono considerati componenti AWT heavyweight. Se sono richiesti dall'applicazione, utilizzare la modalità normale:

Tabella 9. Componenti AWT heavyweight

Componenti AWT heavyweight			
Applet	Frame	List	Robot
Button	JApplet	Menu	Scrollbar
Checkbox	JDialog	MenuBar	ScrollPane
Choice	JFrame	MenuComponent	TextArea
Dialog	JWindow	MenuItem	TextComponent
FileDialog	Label	PopupMenu	Window

Utilizzo dell'AWT in modalità normale con il supporto GUI (graphical user interface) completo:

Per supportare una GUI (graphical user interface), è richiesto un sistema a finestre. La scelta supportata per i5/OS Java è VNC (Virtual Network Computing). Il server VNC è particolarmente adatto per il sistema perché non richiede un mouse, una tastiera e un monitor che supporta la grafica dedicati. IBM fornisce una versione del server VNC che viene eseguita in PASE. Attenersi alle seguenti istruzioni per assicurare che VNC sia installato e avviato e che la sessione Java sia configurata per farne uso.

Prima di poter testare o iniziare a utilizzare l'AWT, eseguire i seguenti passi obbligatori e facoltativi:

- Creare una parola d'ordine VNC. Questo è richiesto una volta per ciascun profilo utente che verrà utilizzato per avviare un server VNC.
- Avviare il server VNC, di norma dopo ciascun IPL del sistema.
- Configurare le variabili di ambiente dell'AWT, una volta in ciascuna sessione precedentemente alla prima esecuzione di Java e all'utilizzo della API AWT.
- Configurare le proprietà di sistema Java. Questa operazione deve essere eseguita ogni volta che si esegue Java.
- Facoltativo; per l'utilizzo interattivo: configurare l'iceWM (ice window manager)
- Facoltativo, per l'interazione diretta con un utente: utilizzare un programma di visualizzazione VNC o un browser Web per stabilire un collegamento a VNC
- Facoltativo: verificare la configurazione AWT

Creazione di un file parola d'ordine VNC:

Per utilizzare il NAWT (Native Abstract Windowing Toolkit) con un server VNC (Virtual Network Computing), è necessario creare un file della parola d'ordine VNC.

L'impostazione predefinita del server VNC richiede un file parola d'ordine utilizzato per proteggere il pannello VNC dagli accessi di utenti non autorizzati. È necessario creare il file parola d'ordine VNC sotto il profilo da utilizzare per avviare il server VNC. Immettere quanto segue su una richiesta comandi i5/OS:

1. `MKDIR DIR('/home/profiloVNC/.vnc')`
2. `QAPTL/VNCPASSWD USEHOME(*NO) PWDFILE('/home/profiloVNC/.vnc/passwd')` dove *profiloVNC* è il profilo che ha avviato il server VNC.

Per ottenere l'accesso interattivo al server VNC utilizzando un VNCviewer o un browser Web da un sistema remoto, gli utenti devono utilizzare la parola d'ordine specificata in questo passo.

Avvio del server VNC:

Per avviare il server VNC (Virtual Network Computing), completare i seguenti passi.

dove *n* è il numero pannello che si desidera utilizzare. i numeri pannello possono essere numeri interi compresi nell'intervallo 1-99.

Il file .Xauthority

Il processo di avvio del server VNC crea un nuovo file .Xauthority o ne modifica uno esistente. L'autorizzazione al server VNC utilizza il file .Xauthority, che contiene informazioni sulla chiave codificata, per impedire ad applicazioni di altri utenti di intercettare le richieste del proprio X-server. Per proteggere le comunicazioni tra la JVM (Java virtual machine) e VNC **È NECESSARIO** che entrambi, JVM e VNC, abbiano accesso alle informazioni sulla chiave codificata nel file .Xauthority.

Il file .Xauthority appartiene al profilo che ha avviato VNC. Il modo più semplice per permettere alla JVM e al server VNC di condividere l'accesso al file .Xauthority consiste nell'eseguire il server VNC e la

JVM sotto lo stesso profilo utente. Se non è possibile eseguire il server VNC e la JVM sotto lo stesso profilo utente, è possibile configurare la variabile di ambiente XAUTHORITY in modo da puntare al file .Xauthority corretto.

Per avviare il server VNC (Virtual Network Computing), immettere il seguente comando sulla riga comandi e premere **INVIO**: `CALL PGM(QSYS/QP2SHELL) PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n'))` dove *n* è il numero del pannello che si desidera utilizzare. I numeri del pannello possono essere numeri interi compresi nell'intervallo 1-99.

L'avvio del server VNC visualizza un messaggio che identifica il nome del sistema e il numero del pannello System i5; ad esempio, `Il nuovo 'X'desktop è systemname:1`. Annotare il nome di sistema ed il numero del pannello poiché serviranno successivamente per configurare la variabile di ambiente DISPLAY quando si esegue l'applicazione Java che utilizza AWT.

Se sono in esecuzione contemporaneamente più server VNC, ogni server VNC richiederà un numero del pannello univoco. Specificare esplicitamente il valore di visualizzazione quando si avvia il server VNC come mostrato consente di controllare quale numero del pannello viene utilizzato da ciascuna applicazione. In alternativa, se non si desidera specificare il numero del pannello, rimuovere '`:n`' dal comando precedente, lasciare che sia il programma vncserver_java a trovare un numero del pannello disponibile e annotarlo.

File .Xauthority

Il processo di avvio del server VNC crea un nuovo file .Xauthority oppure modifica un file .Xauthority esistente nell'indirizzario principale dell'utente che avvia il server. Il file .Xauthority contiene le informazioni di autorizzazione della chiave codificata che il server VNC utilizza per impedire alle applicazioni di altri utenti di intercettare le richieste del proprio X-server. Delle comunicazioni protette tra la JVM (Java virtual machine) e VNC **richiedono** che sia la JVM che VNC abbiano accesso allo stesso file .Xauthority.

Il file .Xauthority appartiene al profilo che ha avviato VNC. Per consentire sia alla JVM che al server VNC di condividere l'accesso, eseguire il server VNC e la JVM sotto lo stesso profilo utente. Se questo non è possibile, è possibile configurare la variabile di ambiente XAUTHORITY in modo che punti al file .Xauthority corretto.

Configurazione delle variabili di ambiente NAWT:

Quando si esegue Java e si desidera utilizzare il supporto GUI AWT completo, è necessario che le variabili di ambiente DISPLAY e XAUTHORITY siano definite per indicare a Java quale pannello X-server utilizzare e dove trovare il corretto file .Xauthority.

Variabile di ambiente DISPLAY

Nella sessione in cui si desidera eseguire i programmi Java, impostare la variabile di ambiente DISPLAY sul nome di sistema e sul numero di pannello. Immettere il seguente comando su una richiesta comandi i5/OS e premere INVIO:

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('nomesistema:n')
```

dove *nomesistema* è il nome host o l'indirizzo IP del sistema e *n* è il numero di pannello del server VNC da utilizzare.

Variabile di ambiente XAUTHORITY

Nella sessione dove si desidera eseguire i programmi Java, impostare la variabile di ambiente XAUTHORITY su `/home/VNCprofile/.Xauthority`, dove *VNCprofile* è il profilo che ha avviato il server VNC. Da una richiesta comandi i5/OS, eseguire il comando:

```
ADDENVVAR ENVVAR(XAUTHORITY) VALUE('/home/profiloVNC/.Xauthority')
```

sostituendo *profiloVNC* con l'appropriato nome profilo.

File predefinito e valori di codifica Java:

La seguente tabella mostra le codifiche Java utilizzate per specifiche impostazioni di CCSID lavoro o file.encoding. La codifica Java determina i font che verranno utilizzati per la visualizzazione delle stringhe di caratteri.

CCSID lavoro	Codifica file predefinita	Codifica Java
00037	ISO8859_1	ISO-8859-1
00420	Cp1046	IBM-1046
00424	ISO8859_8	ISO-8859-8
00870	ISO8859_2	ISO-8859-2
00875	ISO8859_7	ISO-8859-7
00933	Cp970	EUC-KR
00935	Cp1383	IBM-1383
00937	Cp950	IBM-950
00939	Cp943C	IBM-943C
01026	ISO8859_9	ISO-8859-9
01399	Cp943C	IBM-943C
Valore predefinito per tutti gli altri valori	ISO8859_1	ISO-8859-1

Configurazione delle proprietà di sistema Java AWT richieste:

Per utilizzare la API AWT Java, potrebbe essere necessario impostare alcune proprietà sul richiamo Java.

Sono qui indicate le proprietà che devono essere impostate per ciascuna versione JDK e ciascun tipo JVM:

Tabella 10. Proprietà di sistema Java per AWT

	JVM Classic, JDK 1.4 e successive	JVM IBM Technology for Java, JDK 1.5 e successive
Modalità normale (supporto GUI completo)	`os400.awt.native=true`	Nessuna[1]
Modalità headless	`java.awt.headless=true`	`java.awt.headless=true`

[1] Anche se non sono richieste proprietà aggiuntive per la JVM IBM Technology per Java in modalità normale, è comunque necessario eseguire la configurazione delle variabili di ambiente e del server VNC descritta in "Utilizzo dell'AWT in modalità normale con il supporto GUI (graphical user interface) completo" a pagina 248.

Configurazione dell'iceWM (ice window manager):

Configurare l'iceWM (ice window manager), come operazione facoltativa durante la configurazione di NAWT, quando si desidera utilizzare interattivamente il server VNC (Virtual Network Computing). Ad esempio, se si desidera eseguire un'applicazione Java che utilizza una GUI (graphical user interface). iceWM è un gestore di finestre piccolo ma potente incluso in System i Tools For Developers PRPQ.

Eseguito in background, iceWM controlla l'aspetto delle finestre in esecuzione nell'ambiente X Window del server VNC. iceWM fornisce un'interfaccia e una serie di funzioni simili a quelle di molti gestori di finestre noti. La funzionalità predefinita dello script vncserver_java incluso, avvia il server VNC ed esegue iceWM.

Una volta completata questa operazione, vengono creati diversi file di configurazione necessari a iceWM. Se lo si desidera, è anche possibile disabilitare iceWM.

Configurare iceWM

Per configurare l'iceWM, completare le operazioni che seguono su una richiesta comandi i5/OS. Assicurarsi di eseguire queste operazioni con il profilo utilizzato per avviare il server VNC.

1. Immettere il seguente comando e premere **INVIO** per avviare l'installazione:

   ```
   STRPTL CLIENT(IGNORE)
   ```

   Il valore IGNORE funziona come segnaposto che assicura che il comando attivi solo le funzioni di configurazione di STRPTL necessarie a NAWT.
2. Immettere il seguente comando e premere **INVIO** per scollegarsi:

   ```
   SIGNOFF
   ```

   Lo scollegamento assicura che i risultati specifici della sessione del comando STRPTL non influiscano sulle successive azioni intraprese per utilizzare o configurare NAWT.

Nota: eseguire il comando STRPTL una sola volta per ciascun profilo che avvia un server VNC. NAWT non richiede alcuno degli argomenti facoltativi disponibili per il comando. Queste istruzioni sovrascrivono eventuali istruzioni di configurazione per STRPTL associate a 5799-PTL System i Tools For Developers PRPQ.

Disabilitare iceWM

L'avvio del server VNC crea o modifica un file script esistente denominato xstartup_java contenente il comando per eseguire iceWM. Il file script xstartup_java risiede nel seguente indirizzario IFS (integrated file system):

```
/home/VNCprofile/.vnc/
```

dove *VNCprofile* è il nome del profilo che ha avviato il server VNC.

Per disabilitare completamente iceWM, utilizzare un editor di testo per isolare o rimuovere la riga nello script che avvia iceWM. Per isolare la riga, inserire il simbolo cancelletto (carattere #) all'inizio della riga.

Utilizzo di un VNCviewer o browser Web:

Per eseguire un'applicazione che utilizza una GUI (graphical user interface) su un System i, è necessario utilizzare un VNCviewer o un browser Web per collegarsi al server VNC (Virtual Network Computing). È necessario eseguire il VNCviewer o il browser Web su una piattaforma che supporta la grafica, ad esempio un personal computer.

Nota: per effettuare le operazioni che seguono è necessario conoscere il numero di pannello e la parola d'ordine VNC. L'avvio del server VNC (Virtual Network Computing) determina il valore per il numero di pannello. La creazione di un file della parola d'ordine VNC imposta la parola d'ordine VNC.

Utilizzare un VNCviewer per accedere al server VNC

Per utilizzare un VNCviewer per collegarsi al server VNC, completare le seguenti operazioni:

1. Scaricare e installare l'applicazione VNCviewer:

- I programmi di visualizzazione VNC sono disponibili dalla maggior parte delle piattaforme dal sito Web di RealVNC

2. Avviare il VNCviewer scaricato. Alla richiesta, immettere il nome del sistema e il numero del pannello e fare clic su **OK**.
3. Alla richiesta della parola d'ordine, immettere la parola d'ordine VNC per ottenere l'accesso al pannello server VNC.

Utilizzare un browser Web per accedere al server VNC

Per utilizzare un browser Web per collegarsi al server VNC, completare le seguenti operazioni:

1. Avviare il browser ed accedere al seguente URL:

   ```
   http://systemname:58nn
   ```

   dove:
   - *systemname* è il nome o l'indirizzo IP del sistema su cui è in esecuzione il server VNC
   - *nn* è la rappresentazione, sotto forma di due cifre, del numero del pannello del server VNC

   Ad esempio, se il nome di sistema è `system_one` e il numero di pannello è 2, l'URL è:

   ```
   http://system_one:5802
   ```

2. Se l'accesso all'URL ha esito positivo, viene visualizzata la richiesta della parola d'ordine del server VNC. Alla richiesta della parola d'ordine, immettere la parola d'ordine per ottenere l'accesso al pannello del server VNC.

Suggerimenti sull'utilizzo di VNC:

Utilizzare i comandi CL i5/OS per avviare ed arrestare un server VNC (Virtual Network Computing) e visualizzare le informazioni sui server VNC correntemente in esecuzione.

Avvio di un server del pannello VNC da un programma CL

Il seguente esempio è un modo per impostare la variabile di ambiente DISPLAY e avviare VNC automaticamente utilizzando i comandi CL:

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
```

dove:

- *systemname* è il nome host o l'indirizzo IP del sistema dove è in esecuzione VNC
- *n* è il valore numerico che rappresenta il numero del pannello che si intende avviare

Nota: l'esempio presuppone che non si stia eseguendo il pannello :*n* e che sia stato creato il file della parola d'ordine VNC necessario. Per ulteriori informazioni sulla creazione di un file della parola d'ordine, consultare Creazione di un file della parola d'ordine VNC.

Arresto di un server del pannello VNC da un programma CL

Il seguente codice indica uno dei modi per arrestare un server VNC da un programma CL:

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' '-kill' ':n')
```

dove *n* è il valore numerico che rappresenta il numero del pannello che si intende chiudere.

Ricerca di server pannello VNC in esecuzione

Per determinare quali eventuali server VNC sono attualmente in esecuzione sul sistema, completare i seguenti passi:

1. Da una riga comandi i5/OS, avviare una shell PASE:

```
CALL QP2TERM
```

2. Dalla richiesta comandi della shell PASE utilizzare il comando PASE ps per elencare i server VNC:

```
ps gaxuw | grep Xvnc
```

L'emissione risultante da questo comando indicherà i server VNC in esecuzione nel seguente formato:

```
john  418  0.9  0.0  5020  0  - A  Jan 31 222:26
  /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/Xvnc :1 -desktop X -httpd
jane   96  0.2  0.0   384  0  - A  Jan 30 83:54
  /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/Xvnc :2 -desktop X -httpd
```

Dove:

- La prima colonna è il profilo che ha avviato il server.
- La seconda colonna è l'ID processo PASE del server.
- Le informazioni che cominciano per */QOpensys/* corrispondono al comando che ha avviato il server VNC (compresi gli argomenti). Il numero del pannello è, generalmente, la prima voce nell'elenco degli argomenti per il comando Xvnc.

Nota: il processo Xvnc, visualizzato nella precedente emissione di esempio, è il nome del programma server VNC effettivo. Xvnc viene avviato quando si esegue lo script vncserver_java, che prepara l'ambiente e i parametri per Xvnc e poi lo avvia.

Suggerimenti per l'utilizzo di AWT con WebSphere Application Server:

Se si devono eseguire le applicazioni basate su WebSphere in modalità GUI completa, invece di utilizzare la modalità headless, utilizzare questi consigli per evitare problemi con il collegamento tra WebSphere ed il server VNC.

Come assicurare le comunicazioni protette

Il server VNC utilizza un metodo denominato X Authority checking (verifica autorizzazione X) per assicurare delle comunicazioni protette tra se stesso e l'applicazione d'uso, come ad esempio WebSphere.

Il processo di avvio del server VNC crea un file .Xauthority contenente le informazioni sulla chiave codificata. Perché WebSphere Application Server acceda a VNC, **deve** avere accesso e utilizzare lo stesso file .Xauthority che il server VNC sta utilizzando.

Per ottenere ciò, utilizzare uno dei seguenti metodi:

Eseguire WAS (WebSphere Application Server) e VNC utilizzando lo stesso profilo

Se sia WebSphere Application Server che il server VNC che deve utilizzare solo avviato dallo stesso profilo utente, per impostazione predefinita utilizzeranno entrambi lo stesso file .Xauthority. Questo richiede di avviare il server VNC dall'utente predefinito WebSphere (QEJBSVR) oppure di modificare l'utente predefinito WebSphere al profilo utilizzato per avviare il server VNC.

Per passare dal profilo utente predefinito di un server delle applicazioni (QEJBSVR) a uno differente, occorre effettuare le seguenti operazioni:

1. Utilizzare la console di gestione di WAS (WebSphere Application Server) per modificare la configurazione del server delle applicazioni
2. Utilizzare System i Navigator per abilitare il nuovo profilo

Eseguire WAS (WebSphere Application Server) e VNC utilizzando profili differenti

In questo caso, WebSphere Application Server viene avviato da un profilo utente e il file .Xauthority appartiene a un profilo utente differente. Per abilitare WebSphere Application Server ad avviare il server VNC, completare le seguenti operazioni:

1. Creare un nuovo file .Xauthority (oppure aggiornare un file .Xauthority esistente) avviando il server VNC dal profilo utente desiderato. Ad esempio, da una riga comandi CL (Control Language) i5/OS, immettere il seguente comando e premere INVIO:

   ```
   CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
   ```

   dove *n* è il numero del pannello (un valore numerico compreso nell'intervallo 1-99).

   Nota: Il file .Xauthority si trova nell'indirizzario per il profilo sotto il quale si sta eseguendo il server VNC.

2. Utilizzare i seguenti comandi CL per concedere al profilo con cui viene eseguito il WAS (WebSphere Application Server) l'autorizzazione alla lettura del file .Xauthority:

   ```
   CHGAUT OBJ('/home') USER(WASprofile) DTAAUT(*RX)
   CHGAUT OBJ('/home/VNCprofile') USER(WASprofile) DTAAUT(*RX)
   CHGAUT OBJ('/home/VNCprofile/.Xauthority') USER(WASprofile) DTAAUT(*R)
   ```

   dove *VNCprofile* e *WASprofile* sono i profili con cui sono in esecuzione il server VNC e il WAS (WebSphere Application Server).

   Nota: effettuare queste operazioni solo quando *VNCprofile* e *WASprofile* sono profili differenti. Effettuare queste operazioni quando *VNCprofile* e *WASprofile* sono lo stesso profilo può causare il malfunzionamento di VNC.

3. Dalla console di gestione del WAS (WebSphere Application Server), definire le variabili di ambiente DISPLAY e XAUTHORITY per l'applicazione:
   - Per DISPLAY, utilizzare: `sistema:n` o `localhost:n`

     dove *sistema* è il nome o l'indirizzo IP del sistema e *n* è il numero del pannello utilizzato per avviare il server VNC.
   - Per XAUTHORITY, utilizzare: `/home/VNCprofile/.Xauthority`

     dove *VNCprofile* è il profilo che ha avviato il server VNC.

4. Rendere effettive le modifiche alla configurazione riavviando il WAS (WebSphere Application Server).

   WebSphere Application Server per i5/OS

   Gestione di utenti e gruppi con Management Central

Variabili di ambiente i5/OS PASE correlate all'AWT:

Se si sta eseguendo l'IBM JVM Classic e si sta utilizzando la API AWT, l'ambiente PASE i5/OS è necessario e viene avviato automaticamente. In base ai propri requisiti, tuttavia, potrebbe essere necessario impostare delle variabili di ambiente.

Queste informazioni non sono applicabili quando si utilizza la JVM IBM Technology for Java perché è già in esecuzione nell'ambiente PASE.

QIBM_JAVA_PASE_STARTUP
Se si specifica *java.awt.headless=true* o *os400.awt.native=true* su un richiamo dell'IBM JVM Classic, l'ambiente PASE i5/OS PASE viene avviato automaticamente, ma sempre in modalità a 32 bit. Se l'applicazione richiede che l'ambiente PASE venga eseguito in modalità a 64 bit, è necessario impostare la variabile di ambiente QIBM_JAVA_PASE_STARTUP su un valore di */usr/lib/start64* prima di richiamare Java. Il valore predefinito è */usr/lib/start32*.

QIBM_JAVA_PASE_ALLOW_PREV
Per impostazione predefinita, la JVM Classic prevede di avviare l'ambiente i5/OS PASE, se necessario, e avrà esito negativo con un messaggio di errore se l'ambiente PASE è già attivo. Se l'applicazione richiede che l'AWT utilizzi un ambiente PASE preavviato, forse perché

l'applicazione Java viene avviata dall'interno di una shell QP2TERM, prima di richiamare Java è necessario impostare la variabile di ambiente QIBM_JAVA_PASE_ALLOW_PREV su un valore di *1*. Questo indica che è possibile utilizzare l'ambiente PASE esistente.

PASE_THREAD_ATTACH
Questa variabile di ambiente indica all'ambiente PASE i5/OS che dei sottoprocessi aggiuntivi avviati nel processo, spesso avviati dalla JVM, devono collegarsi implicitamente all'ambiente PASE in modo da avere accesso ai metodi nativi AWT, ecc. PASE_THREAD_ATTACH viene automaticamente impostato dal comando Java su una richiesta comandi i5/OS oppure nelle shell QSH o QP2TERM. Se l'applicazione crea direttamente una JVM da un programma C/C++ utilizzando l'interfaccia di richiamo JNI e i sottoprocessi secondari creati dalla JVM devono accedere a PASE (i metodi nativi AWT, ad esempio), occorrerà impostare la variabile di ambiente PASE_THREAD_ATTACH su un valore di *Y*. In caso contrario, verrà generato un errore UnsatisfiedLinkError o degli errori simili.

Concetti correlati

"Variabili di ambiente Java i5/OS PASE" a pagina 221
La JVM (Java virtual machine) utilizza le variabili che seguono per avviare gli ambienti i5/OS PASE. È necessario impostare la variabile QIBM_JAVA_PASE_STARTUP per poter eseguire l'esempio per il metodo nativoIBM i5/OS PASE per Java.

Verifica della configurazione AWT:

È possibile verificare la configurazione AWT eseguendo un programma di verifica Java.

Per eseguire il programma di verifica da una riga comandi i5/OS, immettere uno dei seguenti comandi, a seconda della modalità che si desidera verificare:

```
JAVA CLASS(NAWTtest) CLASSPATH('/QIBM/ProdData/Java400') PROP((os400.awt.native true))
```

Oppure

```
JAVA CLASS (NAWTtest) CLASSPATH('/QIBM/ProdData/Java400') PROP((java.awt.headless true))
```

Il programma di verifica crea un'immagine in formato JPEG e la salva nel seguente percorso nell'IFS (integrated file system):

```
/tmp/NAWTtest.jpg
```

Una volta eseguito il programma di verifica, controllare che abbia creato il file e che non abbia emesso alcuna eccezione Java. Per visualizzare l'immagine, utilizzare la modalità binaria per caricare il file immagine su un sistema con supporto grafico e visualizzarla con un browser, un programma grafico o uno strumento simile.

Sicurezza Java

Questo argomento fornisce i dettagli sull'autorizzazione adottata e spiega come è possibile utilizzare SSL per rendere sicuri i flussi di socket nell'applicazione Java.

Le applicazioni Java sono soggette alle stesse limitazioni relative alla sicurezza di qualsiasi altro programma sulla piattaforma System i5. Per eseguire un programma Java su un System i5, è necessario disporre dell'autorizzazione al file di classe nell'IFS (integrated file system). Una volta che il programma è avviato, viene eseguito sotto l'autorizzazione dell'utente.

Nei release precedenti alla V6R1, è possibile utilizzare l'autorizzazione adottata per accedere agli oggetti con l'autorizzazione dell'utente che sta eseguendo il programma e l'autorizzazione del proprietario del programma. L'autorizzazione adottata fornisce temporaneamente a un utente l'autorizzazione ad oggetti sui quali in precedenza non aveva alcuna autorizzazione. Consultare le informazioni sul comando

| CRTJVAPGM (Creazione programma Java) per informazioni dettagliate sui due parametri di
| autorizzazione adottata, che sono USRPRF e USEADPAUT.

| In V6R1, sarà necessaria una speciale PRPQ (programming request for price quotation), che è un prodotto
| programma personalizzato IBM per continuare ad utilizzare l'autorizzazione adottata nelle applicazioni
| Java. Nei futuri release, il supporto dell'autorizzazione adottata verrà ritirato. Per ulteriori informazioni
| sull'autorizzazione adottata, su come ottenere la PRPQ e sulla separazione delle applicazioni
| dall'autorizzazione adottata, consultare Modifiche all'autorizzazione adottata in V6R1.

La maggior parte dei programmi Java eseguiti su un System i5 sono applicazioni, non applet, e pertanto il modello di sicurezza "sandbox" non le limita.

Nota: per J2SDK, versione 1.4 e release successivi, JAAS, JCE, JGSS e JSSE fanno parte del JDK di base e non vengono considerati estensioni. Per le precedenti versioni di JDK, questi elementi di sicurezza rappresentano estensioni.

Descrizione del comando CL CRTJVAPGM (Creazione programma Java)

"Modifiche all'autorizzazione adottata nella V6R1"
Il supporto per l'autorizzazione adottata del profilo utente tramite i programmi Java è stato ritirato nella V6R1. Quest'argomento descrive come determinare se le applicazioni stanno utilizzando l'autorizzazione adottata e come modificare le applicazioni per accogliere questa modifica.

| Modifiche all'autorizzazione adottata nella V6R1

| Il supporto per l'autorizzazione adottata del profilo utente tramite i programmi Java è stato ritirato nella
| V6R1. Quest'argomento descrive come determinare se le applicazioni stanno utilizzando l'autorizzazione
| adottata e come modificare le applicazioni per accogliere questa modifica.

| In V6R1, le applicazioni Java non potranno più adottare l'autorizzazione di profilo utente tramite i
| programmi Java a meno che non sia installato PRPQ (programming request for price quotation) IBM
| Adopt Authority for Java for i5/OS 5799-AAJ. Il supporto per l'autorizzazione adottata Java e 5799-AAJ
| sarà ritirato in un futuro release. Utilizzare, pertanto, V6R1 come un release di transizione per
| l'autorizzazione adottata Java. Si invitano gli utenti a rimuovere tutte le dipendenze dall'autorizzazione
| adottata Java quanto prima.

| Queste sezioni descrivono alcune situazioni comuni dove viene utilizzata l'autorizzazione adottata Java e
| come è possibile modificare le applicazioni Java per rimuovere la dipendenza dall'autorizzazione adottata
| Java. Queste modifiche consentiranno alle applicazioni Java interessate di continuare a funzionare come
| previsto nei futuri release. Inoltre, queste modifiche consentiranno all'applicazione Java interessata di
| funzionare come previsto nel release corrente senza PRPQ 5799-AAJ installato.

| Come determinare se le applicazioni utilizzano l'autorizzazione adottata

| È possibile avvalersi di uno strumento nei release V5R3 e V5R4 per determinare se si hanno delle
| applicazioni Java che saranno interessate dalle modifiche all'autorizzazione adottata. Lo strumento
| funziona con la JVM per notificare ai programmi Java che la JVM sta eseguendo la registrazione come se
| stesse utilizzando l'autorizzazione adottata. Questo strumento è separato da PRPQ 5799-AAJ ed è
| disponibile tramite le seguenti PTF:

| **V5R3**

| Lo strumento è fornito nella PTF SI27769. In V5R3, la capacità di registrazione della JVM è fornita dalla
| Java Group PTF SF99269, livello 15, ed è limitata ai programmi Java che utilizzano JDK 5.0.

| **V5R4**

Lo strumento è fornito nella PTF SI27772. Nella V5R4, la capacità di registrazione della JVM è abilitata per tutti i JDK e non è richiesta alcuna PTF aggiuntiva.

Questo strumento aiuta anche ad identificare le applicazioni Java che potrebbero basarsi sull'autorizzazione adottata analizzando il sistema per rilevare la presenza di programmi Java creati con il supporto per l'autorizzazione adottata.

Per impostazione predefinita, questo strumento visualizza gli utilizzi registrati della JVM dell'autorizzazione adottata ed analizza anche l'intero sistema. Tuttavia, esso supporta anche varie opzioni dalla Qshell:

```
Utilizzo: /qsys.lib/qjava.lib/qjvaadptl.pgm [opzione]...
        Le opzioni valide includono
          -h             : visualizzare questa istruzione di utilizzo.
          -o <file>      : scrivere l'emissione nel file specificato.
          -d <directory> : analizzare solo la struttura ad albero degli indirizzari specificata.
          -noscan        : non analizzare il sistema. Notificare solo gli usi registrati.
```

L'emissione dallo strumento può aiutare a determinare quali applicazioni Java sul sistema stanno utilizzando l'autorizzazione adottata. Utilizzando queste informazioni, potrebbe essere necessario eseguire le seguenti operazioni:

- Se l'utilizzo è nel codice che è stato acquistato, rivolgersi al produttore per appurare cosa intendono fare con l'autorizzazione adottata.
- Se l'utilizzo è nel proprio codice, leggere le soluzioni illustrate in questo documento per determinare se si è disposti a, e in grado di, modificare il codice in modo che non utilizzi più l'autorizzazione adottata.
- Se, dopo aver considerato le varie alternative, si ritiene che si avrà bisogno di utilizzare l'autorizzazione adottata su V6R1, rivolgersi al tecnico di manutenzione IBM. L'utente verrà messo in contatto con il laboratorio di sviluppo di Rochester e verrà consultato in merito ai possibili modi per eseguire l'attività desiderata senza utilizzare l'autorizzazione adottata. Se necessario, all'utente verranno date istruzioni su come ottenere IBM Adopt Authority for Java for i5/OS 5799-AAJ.

Utilizzo dell'autorizzazione adottata

Poiché l'autorizzazione adottata è utile solo per eseguire operazioni sugli oggetti i5/OS e sui record di database oppure per accedere ai metodi nativi, gli esempio in questa raccolta di argomenti riguarderanno specificamente queste aree. Per una spiegazione di base dell'autorizzazione adottata, consultare Oggetti che adottano l'autorizzazione del proprietario nell'argomento Riferimento alla sicurezza.

Utilizzando l'autorizzazione adottata, un metodo può adottare l'autorizzazione del proprietario del programma eseguito piuttosto che l'autorizzazione della persona che esegue il programma al file di eseguire qualche operazione. Concettualmente, questo è molto simile a `Imposta UID` e `Imposta GID` su UNIX e l'esempio canonico di utilizzo è `Modifica parola d'ordine` così come creato in UNIX. Mentre non è ragionevole che ciascun utente disponga dell'autorizzazione a modificare il file parola d'ordine, è vantaggioso che il programma sia ritenuto attendibile e disponga dell'autorizzazione a modificare la parola d'ordine per conto degli utenti.

System i poteva fornire la funzione di autorizzazione adottata per i programmi Java perché la JVM faceva parte del TCB (Trusted Computing Base) nello SLIC (system licensed internal code). Con il passaggio di i5/OS ad una implementazione Javadove la JVM è un programma a livello utente, Java non può più fornire questa funzione.

Per pervenire alla stessa funzionalità, l'approccio più comune consisterà nell'aggiungere un metodo nativo ILE al programma Java che adotta l'autorizzazione di profilo utente equivalente ed esegue l'operazione richiesta. Potrebbe anche essere possibile pervenire agli stessi effetti dell'autorizzazione adottata senza aggiungere i metodi nativi eseguendo la funzione in un processo separato che ha un'autorizzazione aumentata e inviando le richieste a tale programma come necessario.

Attributi di adozione

Ciascun richiamo del metodo è rappresentato da un frame di stack sullo stack di tempo di esecuzione. Per i metodi DE (direct execution) e i metodi nativi Java, il frame di stack può essere correlato ad un oggetto di programma i5/OS che identifica gli attributi di adozione del richiamo. Questi attributi sono:

Adotta profilo utente
Questo corrisponde all'opzione USRPRF(*USER/*OWNER) nei comandi CRTJVAPGM, CRTPGM e CRTSRVPGM. Il valore *USER indica che i metodi nel programma **non** adottano le autorizzazioni del profilo utente proprietario in fase di esecuzione. Il valore *OWNER indica che i metodi nel programma adottano le autorizzazioni del profilo utente proprietario in fase di esecuzione.

Nei seguenti diagrammi, i frame di stack che adottano le autorizzazioni del profilo utente proprietario sono indicati con uno sfondo ombreggiato.

Ciascun frame di stack può essere descritto come uno che **adotta** o **non adotta** l'autorizzazione del profilo utente proprietario.

Consenti/rifiuta autorità adottata
Questo corrisponde all'opzione USEADPAUT(*YES/*NO) sui comandi CRTJVAPGM, CHGPGM e CHGSRVPGM. Il valore *YES indica che il richiamo per il frame di stack consentirà l'utilizzo delle autorizzazioni del profilo utente adottato dai frame di stack precedenti nel richiamo corrente. Il valore *NO significa che l'autorizzazione adottata dai richiami precedenti è tralasciata.

Quando l'autorizzazione adottata dai richiami precedenti è rilasciata, non viene utilizzata nel richiamo corrente o nei metodi richiamati dal frame corrente.

Ciascun frame di stack può essere descritto come uno che **consente** o **rilascia** l'autorizzazione adottata dai richiami precedenti sullo stack.

Oggetti che adottano l'autorizzazione del proprietario

Esempi: alternative all'autorizzazione adottata

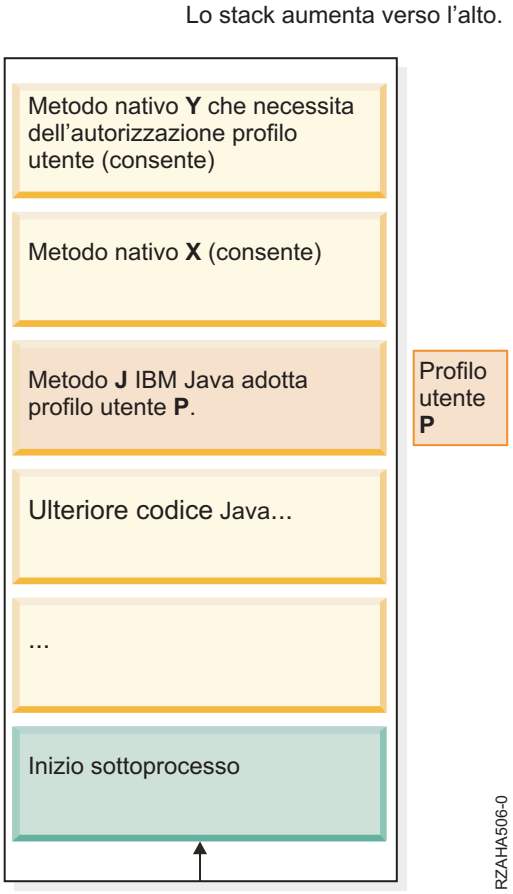
Quest'argomento contiene alcuni esempi dell'utilizzo dell'autorizzazione adottata Java e alcune alternative consigliate. Queste alternative utilizzano i metodi nativi nei programmi di servizio ILE per adottare l'autorizzazione in un modo analogo agli esempi Java.

Sono possibili delle altre alternative e, in alcune circostanze, potrebbero essere preferibili. Una possibilità consiste nell'interscambiare il profilo utente del processo per acquisire un'autorizzazione aggiuntiva. Quest'argomento non descrive l'interscambio del profilo utente, che presenta problemi e rischi specifici. Gli esempi compresi in questa raccolta di argomenti dimostreranno due situazioni comuni dove viene utilizzata l'autorizzazione adottata Java e offriranno possibili alternative. Questo documento presume che sul sistema non sia installata la PRPQ 5799-AAJ.

- "Esempio 1: il metodo Java adotta l'autorizzazione immediatamente prima di richiamare un metodo nativo" a pagina 260
- "Alternativa 1A: reimpacchettamento del metodo nativo X" a pagina 262
- "Alternativa 1B: nuovo metodo nativo N" a pagina 264
- "Esempio 2: Il metodo Java adotta l'autorizzazione e richiama altri metodi Java prima di richiamare un metodo nativo" a pagina 266
- "Alternativa 2: nuovo metodo nativo N" a pagina 268
- "Comandi di compilazione per gli esempi" a pagina 270

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute in "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

Esempio 1: il metodo Java adotta l'autorizzazione immediatamente prima di richiamare un metodo nativo



In quest'esempio, il metodo IBM Java **J** è contenuto in un programma Java che adotta il profilo utente **P** e richiama direttamente il metodo nativo **X**. Il metodo nativo **X** richiama il metodo ILE **Y**, che richiede l'autorizzazione adottata.

JA61Example1.java

```
public class JA61Example1 {
    public static void main(String args[]) {
 int returnVal = J();
 if (returnVal > 0) {
     System.out.println("Autorizzazione adottata correttamente.");
 }
 else {
     System.out.println("ERRORE: impossibile adottare l'autorizzazione.");
 }
    }

    static int J() {
 return X();
    }

    // Restituisce: 1 se capace di accedere correttamente a *DTAARA JADOPT61/DATAAREA
    static native int X();

    static {
 System.loadLibrary("EX1");
    }
}
```

JA61Example1.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class JA61Example1 */

#ifndef _Included_JA61Example1
#define _Included_JA61Example1
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     JA61Example1
 * Method:    X
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_JA61Example1_X
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

JA61Example1.c

```
/* Questo contiene il codice sorgente per il metodo nativo Java_JA61Example1_X. Questo
   modulo è concatenato al programma di manutenzione JADOPT61/EX1. */

#include "JA61Example1.h"
#include "JA61ModuleY.h"

/*
 * Class:     JA61Example1
 * Method:    X
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_JA61Example1_X(JNIEnv* env, jclass klass) {
   return methodY();
}
```

JA61ModuleY.h

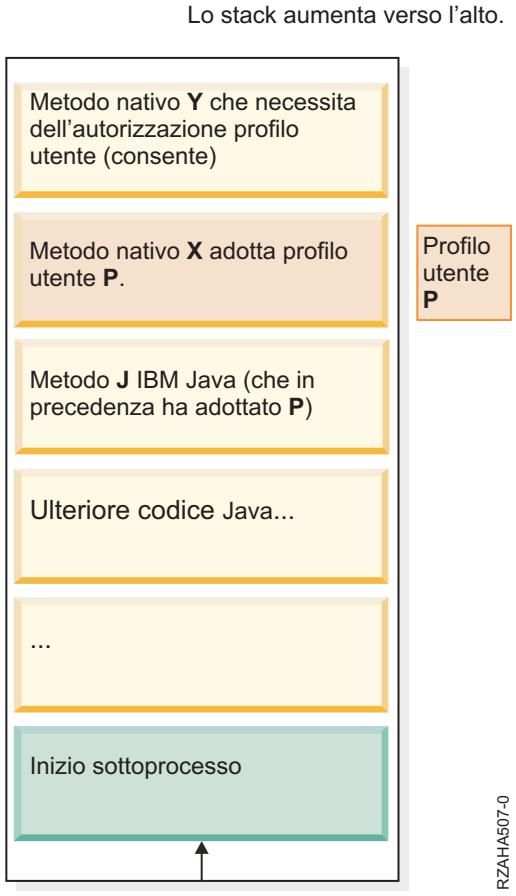
```
/* Questo metodo prova a modificare *DTAARA JADOPT61/DATAAREA. Questo
   sarà possibile solo se è stato adottato il profilo utente JADOPT61UP. */
int methodY(void);
```

JA61ModuleY.c

```
#include <except.h>
#include <stdio.h>
#include "JA61ModuleY.h"
#include <xxdtaa.h>

#define START 1
#define LENGTH 8


/* Questo metodo prova ad operare su *DTAARA JADOPT61/DATAAREA. Questo
 sarà possibile solo se è stato adottato il profilo utente JADOPT61UP. */
int methodY(void) {
   int returnValue;
   volatile int com_area;
   char newdata[LENGTH] = "new data";
   _DTAA_NAME_T dtaname = {"DATAAREA  ", "JADOPT61  "};

   /* Controllare la presenza di un'eccezione in quest'intervallo */
#pragma exception_handler(ChangeFailed, 0, _C1_ALL, _C2_MH_ESCAPE)
```

```
    /* modificare *DTAARA JADOPT61/DATAAREA */
    QXXCHGDA(dtaname, START, LENGTH, newdata);
#pragma disable_handler

ChangeCompleted:
    printf("Successfully updated data area\n");
    returnValue = 1;
    goto TestComplete;

ChangeFailed:      /* Control goes here for an exception */
    printf("Got an exception.\n");
    returnValue = 0;

TestComplete:
    printf("methodY completed\n");
    return returnValue;
}
```

Alternativa 1A: reimpacchettamento del metodo nativo X



Un modo per preservare l'adozione dell'autorizzazione consiste nel separare il metodo nativo **X** in un nuovo programma di servizio. Questo nuovo programma di servizio può quindi adottare il profilo utente **P**, che era precedentemente adottato dal metodo Java **J**.

JA61Alternative1A.java

```
public class JA61Alternative1A {
    public static void main(String args[]) {
 int returnVal = J();
 if (returnVal > 0) {
```

```
|       System.out.println("Autorizzazione adottata correttamente.");
|   }
|   else {
|       System.out.println("ERRORE: impossibile adottare l'autorizzazione.");
|   }
|      }
|
|      static int J() {
|   return X();
|      }
|
|      // Restituisce: 1 se capace di accedere correttamente a *DTAARA JADOPT61/DATAAREA
|      static native int X();
|
|      static {
|   System.loadLibrary("ALT1A");
|      }
|  }
```

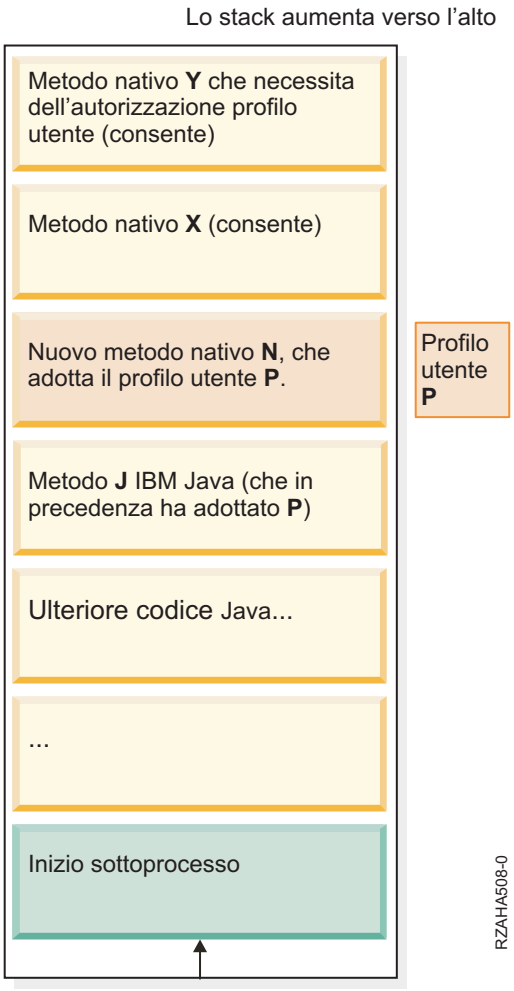
JA61Alternative1A.h

```
|  /* DO NOT EDIT THIS FILE - it is machine generated */
|  #include <jni.h>
|  /* Header for class JA61Alternative1A */
|
|  #ifndef _Included_JA61Alternative1A
|  #define _Included_JA61Alternative1A
|  #ifdef __cplusplus
|  extern "C" {
|  #endif
|  /*
|   * Class:     JA61Alternative1A
|   * Method:    X
|   * Signature: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative1A_X
|    (JNIEnv *, jclass);
|
|  #ifdef __cplusplus
|  }
|  #endif
|  #endif
```

JA61Alternative1A.c

```
|  /* Questo contiene il codice sorgente per il metodo nativo Java_JA61Alternative1A_X. Questo
|     modulo è concatenato al programma di manutenzione JADOPT61/ALT1A.*/
|
|  #include "JA61Alternative1A.h"
|  #include "JA61ModuleY.h"
|
|  /*
|   * Class:     JA61Alternative1A
|   * Method:    X
|   * Signature: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative1A_X(JNIEnv* env, jclass klass) {
|     return methodY();
|  }
```

Alternativa 1B: nuovo metodo nativo N



Un altro modo per preservare l'adozione del profilo utente consiste nel creare un metodo nativo **N** nuovo contenuto in un programma di servizio che adotta il profilo utente **P**. Questo nuovo metodo è richiamato dal metodo Java **J** e richiama il metodo nativo **X**. Il metodo Java **J** richiederebbe di essere modificato per richiamare **N** invece di **X**, ma il metodo nativo **X** non richiederebbe di essere modificato o reimpacchettato.

JA61Alternative1B.java

```
public class JA61Alternative1B {
    public static void main(String args[]) {
 int returnVal = J();
 if (returnVal > 0) {
     System.out.println("Autorizzazione adottata correttamente.");
 }
 else {
     System.out.println("ERRORE: impossibile adottare l'autorizzazione.");
 }
    }

    static int J() {
 return N();
    }

    // Restituisce: 1 se capace di accedere correttamente a *DTAARA JADOPT61/DATAAREA
```

```
    static native int N();

    static {
 System.loadLibrary("ALT1B");
    }
}
```

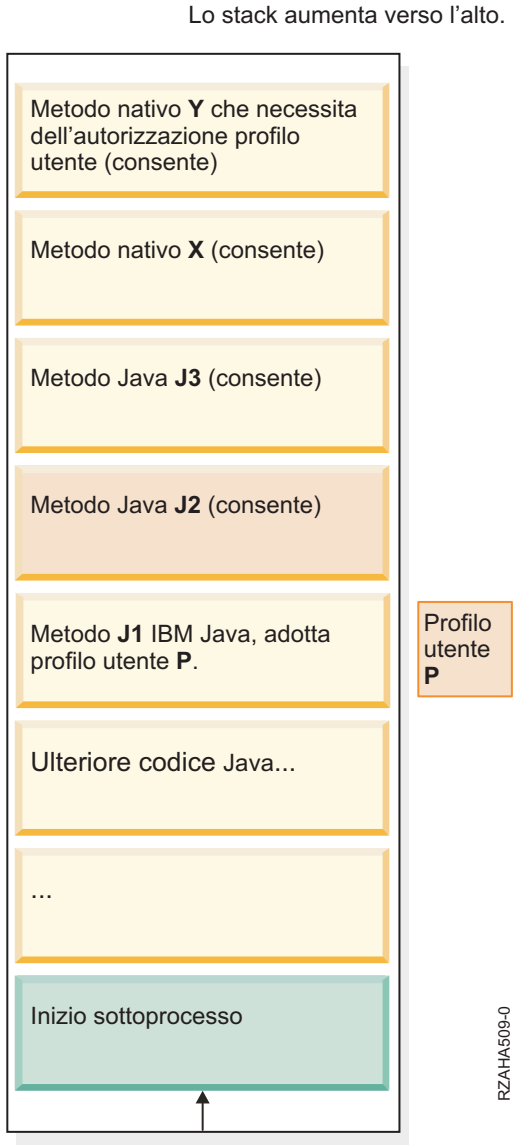
JA61Alternative1B.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class JA61Alternative1B */

#ifndef _Included_JA61Alternative1B
#define _Included_JA61Alternative1B
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     JA61Alternative1B
 * Method:    N
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_JA61Alternative1B_N
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

JA61Alternative1B.c

```
/* Questo contiene il codice sorgente per il metodo nativo Java_JA61Alternative1B_N. Questo
   modulo è concatenato al programma di manutenzione JADOPT61/ALT1B. */

#include "JA61Alternative1B.h"
#include "JA61Example1.h"

/*
 * Class:     JA61Alternative1B
 * Method:    N
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_JA61Alternative1B_N(JNIEnv* env, jclass klass) {
   return Java_JA61Example1_X(env, klass);  /* from JA61Example1.h */
}
```

Esempio 2: Il metodo Java adotta l'autorizzazione e richiama altri metodi Java prima di richiamare un metodo nativo



In quest'esempio, un metodo IBM Java **J1** è contenuto in un programma Java che adotta il profilo utente **P**, **J1** e chiama il metodo Java **J2** che chiama **J3**, che quindi chiama il metodo nativo **X**. Il metodo nativo **X** chiama il metodo ILE **Y**, che richiede l'autorizzazione adottata.

JA61Example2.java

```
public class JA61Example2 {
    public static void main(String args[]) {
 int returnVal = J1();
 if (returnVal > 0) {
     System.out.println("Autorizzazione adottata correttamente.");
 }
 else {
     System.out.println("ERRORE: impossibile adottare l'autorizzazione.");
 }
    }
```

```
|
|      static int J1() {
|   return JA61Example2Allow.J2();
|      }
|
|  }
```

JA61Example2Allow.java

```
|  public class JA61Example2Allow {
|      public static int J2() {
|   return J3();
|      }
|
|      static int J3() {
|   return X();
|      }
|
|      // Restituisce: 1 se capace di accedere correttamente a *DTAARA JADOPT61/DATAAREA
|      static native int X();
|
|      static {
|   System.loadLibrary("EX2ALLOW");
|      }
|  }
```

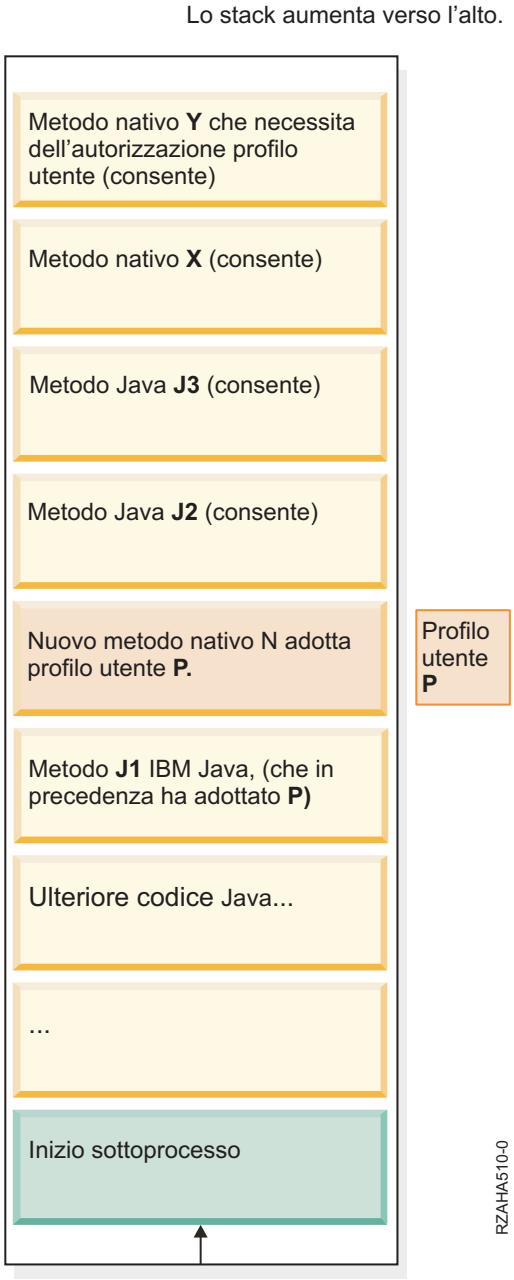
JA61Example2Allow.h

```
|  /* DO NOT EDIT THIS FILE - it is machine generated */
|  #include <jni.h>
|  /* Header for class JA61Example2Allow */
|
|  #ifndef _Included_JA61Example2Allow
|  #define _Included_JA61Example2Allow
|  #ifdef __cplusplus
|  extern "C" {
|  #endif
|  /*
|   * Class:     JA61Example2Allow
|   * Method:    X
|   * Signature: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Example2Allow_X
|    (JNIEnv *, jclass);
|
|  #ifdef __cplusplus
|  }
|  #endif
|  #endif
```

JA61Example2Allow.c

```
|  /* Questo contiene il codice sorgente per il metodo nativo Java_JA61Example2Allow_X. Questo
|     modulo è concatenato al programma di manutenzione JADOPT61/EX2ALLOW. */
|
|  #include "JA61Example2Allow.h"
|  #include "JA61ModuleY.h"
|
|  /*
|   * Class:     JA61Example2Allow
|   * Method:    X
|   * Signature: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Example2Allow_X(JNIEnv* env, jclass klass) {
|     return methodY();
|  }
```

Alternativa 2: nuovo metodo nativo N



Per conservare l'autorizzazione adottata in questo caso, può essere creato un nuovo metodo nativo **N**. Questo metodo nativo sarebbe contenuto in un programma di servizio che adotta il profilo utente **P**.

Il metodo nativo **N** utilizzerebbe quindi la JNI per richiamare il metodo Java **J2**, che non ha subito modifiche. Il metodo Java **J1** dovrebbe essere modificato per richiamare il metodo nativo **N** invece del metodo Java **J2**.

JA61Alternative2.java

```
public class JA61Alternative2 {
    public static void main(String args[]) {
 int returnVal = J1();
```

```
|   if (returnVal > 0) {
|       System.out.println("Autorizzazione adottata correttamente.");
|   }
|   else {
|       System.out.println("ERRORE: impossibile adottare l'autorizzazione.");
|   }
|      }
|
|      static native int N();
|
|      static int J1() {
|   return N();
|      }
|
|      static {
|   System.loadLibrary("ALT2");
|      }
|
|  }
```

JA61Alternative2.h

```
|  /* DO NOT EDIT THIS FILE - it is machine generated */
|  #include <jni.h>
|  /* Header for class JA61Alternative2 */
|
|  #ifndef _Included_JA61Alternative2
|  #define _Included_JA61Alternative2
|  #ifdef __cplusplus
|  extern "C" {
|  #endif
|  /*
|   * Class:     JA61Alternative2
|   * Method:    N
|   * Signature: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative2_N
|    (JNIEnv *, jclass);
|
|  #ifdef __cplusplus
|  }
|  #endif
|  #endif
```

JA61Alternative2.C

```
|  include "JA61Alternative2.h"
|
|  /*
|   * Class:     JA61Alternative2
|   * Method:    N
|   * Signature: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative2_N(JNIEnv* env, jclass klass) {
|  #pragma convert(819)
|      char* className = "JA61Example2Allow";
|      char* methodName = "J2";
|      char* methodSig = "()I";
|  #pragma convert(0)
|
|      // Individuare la classe JA61Example2Allow
|      jclass cls = env->FindClass(className);
|      // Ottenere l'ID metodo per J2()I e richiamarlo.
|      jmethodID methodID = env->GetStaticMethodID(cls, methodName, methodSig);
|      int result = env->CallStaticIntMethod(cls, methodID);
|      return result;
|  }
```

Comandi di compilazione per gli esempi

Tutte le seguenti istruzioni sono basate sul codice sorgente che si trova in un indirizzario denominato /home/javatests/adoptup/v6r1mig su una macchina System i.

In Qshell:

```
> cd /home/javatests/adoptup/v6r1mig
> javac -g *.java
```

Da CL:

```
> CRTLIB JADOPT61
> CRTUSRPRF USRPRF(JADOPT61UP) STATUS(*DISABLED)
> CRTUSRPRF USRPRF(JADOPT61) PASSWORD(j61adopt) INLPGM(QSYS/QCMD) SPCAUT(*NONE)

> CRTDTAARA DTAARA(JADOPT61/DATAAREA) TYPE(*CHAR) LEN(50) VALUE('Initial value')
> GRTOBJAUT OBJ(JADOPT61/DATAAREA) OBJTYPE(*DTAARA) USER(JADOPT61UP) AUT(*ALL)
> RVKOBJAUT OBJ(JADOPT61/DATAAREA) OBJTYPE(*DTAARA) USER(*PUBLIC) AUT(*ALL)
> RVKOBJAUT OBJ(JADOPT61/DATAAREA) OBJTYPE(*DTAARA) USER(YOUR_USER_ID) AUT(*ALL)
```

Creare SRVPGMY, che è utilizzato da tutti gli esempi:

```
> CRTCMOD MODULE(JADOPT61/MODULEY) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61ModuleY.c')
DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/SRVPGMY) MODULE(JADOPT61/MODULEY) EXPORT(*ALL)
```

Creare "Esempio 1: il metodo Java adotta l'autorizzazione immediatamente prima di richiamare un metodo nativo" a pagina 260:

```
> CRTCMOD MODULE(JADOPT61/EX1) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Example1.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/EX1) EXPORT(*ALL) BNDSRVPGM(JADOPT61/SRVPGMY)
> QSH CMD('chown JADOPT61UP /home/javatests/adoptup/v6r1mig/JA61Example1.class')
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Example1.class') USRPRF(*OWNER)
```

Creare "Alternativa 1A: reimpacchettamento del metodo nativo X" a pagina 262:

```
> CRTCMOD MODULE(JADOPT61/ALT1A) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Alternative1A.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/ALT1A) EXPORT(*ALL) BNDSRVPGM(JADOPT61/SRVPGMY) USRPRF(*OWNER)
> CHGOBJOWN OBJ(JADOPT61/ALT1A) OBJTYPE(*SRVPGM) NEWOWN(JADOPT61UP)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Alternative1A.class')
```

Creare "Alternativa 1B: nuovo metodo nativo N" a pagina 264:

```
> CRTCMOD MODULE(JADOPT61/ALT1B) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Alternative1B.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/ALT1B) EXPORT(*ALL) BNDSRVPGM(JADOPT61/EX1) USRPRF(*OWNER)
> CHGOBJOWN OBJ(JADOPT61/ALT1B) OBJTYPE(*SRVPGM) NEWOWN(JADOPT61UP)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Alternative1B.class')
```

Creare "Esempio 2: Il metodo Java adotta l'autorizzazione e richiama altri metodi Java prima di richiamare un metodo nativo" a pagina 266

```
> CRTCMOD MODULE(JADOPT61/EX2ALLOW) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Example2Allow.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/EX2ALLOW) EXPORT(*ALL) BNDSRVPGM(JADOPT61/SRVPGMY)
> QSH CMD('chown JADOPT61UP /home/javatests/adoptup/v6r1mig/JA61Example2.class')
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Example2.class') USRPRF(*OWNER)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Example2Allow.class') USEADPAUT(*YES)
```

Creare "Alternativa 2: nuovo metodo nativo N" a pagina 268:

```
> CRTCPPMOD MODULE(JADOPT61/ALT2) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Alternative2.C')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/ALT2) EXPORT(*ALL) USRPRF(*OWNER)
> CHGOBJOWN OBJ(JADOPT61/ALT2) OBJTYPE(*SRVPGM) NEWOWN(JADOPT61UP)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Alternative2.class')
```

Per eseguire gli esempi, eseguire i seguenti passi:

```
> collegarsi come JADOPT61
> ADDLIBLE JADOPT61
> ADDENVVAR ENVVAR(CLASSPATH) VALUE('/home/javatests/adoptup/v6r1mig')
> JAVA JA61Example1
> JAVA JA61Alternative1A
> JAVA JA61Alternative1B
> JAVA JA61Example2
> JAVA JA61Alternative2
```

Modello di sicurezza Java

È possibile scaricare le applet Java da qualsiasi sistema; in questo modo, esistono meccanismi di sicurezza all'interno della JVM (Java virtual machine) per la protezione da applet non affidabili. Il sistema del tempo di esecuzione Java verifica i bytecode quando la Java virtual machine li carica. Questo assicura che i bytecode siano validi e che il codice non violi nessuna delle limitazioni che la Java virtual machine pone nelle applet Java.

Così come avviene per le applet, il programma di caricamento e quello di verifica del bytecode controllano che i byte siano validi e che i tipi di dati vengano utilizzati in modo appropriato. Inoltre, questi controllano che esista un accesso corretto ai registri e alla memoria e che l'accumulo non sia in eccedenza o insufficiente. Questi controlli assicurano alla Java virtual machine un'esecuzione sicura della classe senza compromettere l'integrità del sistema.

Le applet Java sono limitate nelle operazioni che possono eseguire, nel modo in cui si verifica l'accesso alla memoria e nel modo in cui queste utilizzano la Java virtual machine. Le limitazioni hanno il compito di prevenire la situazione in cui un'applet Java ottenga l'accesso al sistema operativo sottostante o ai dati sul sistema. Ciò rappresenta un modello di sicurezza "sandbox", perché l'applet Java può solo "operare" nella propria sandbox.

Il modello di sicurezza "sandbox" rappresenta una combinazione del programma di caricamento classi, del programma di verifica del file di classe e della classe java.lang.SecurityManager.

Sicurezza della Sun Microsystems, Inc.

Applicazioni sicure con SSL

JCE (Java Cryptography Extension)

JCE (Java Cryptography Extension) 1.2 è un'estensione standard a J2SE (Java 2 Platform, Standard Edition). L'implementazione di JCE su System i è compatibile con l'implementazione di Sun Microsystems, Inc. Questa documentazione descrive gli aspetti univoci dell'implementazione di System i.

Allo scopo di capire queste informazioni, sarebbe auspicabile avere una certa familiarità con la documentazione generale relativa alle estensioni JCE. Consultare la documentazione su JCE di SUN per ulteriori informazioni sulle estensioni JCE.

Il fornitore JCE IBM fornisce un RNG (random number generator).

Esiste anche un fornitore JCE IBMJCEFIPS. Questo fornitore è stato approvato ed è compatibile con FIPS (Federal Information Processing standard) 140-2, "Requisiti di sicurezza per i moduli crittografici."

Il fornitore JCE IBMJCEFIPS supporta i seguenti algoritmi:

Tabella 11. Algoritmi supportati dal fornitore JCE IBMJCEFIPS

algoritmi di firma	algoritmi di codifica	MAC (Message authentication code)	Messaggio digest
SHA1withDSA SHA1withRSA	AES TripleDES RSA	HmacSHA1	MD5 SHA-1 SHA-256 SHA-384 SHA-512

Il fornitore JCE IBMJCEFIPS JCE supporta anche l'algoritmo IBMSecureRandom per la creazione di numeri casuali.

Per utilizzare IBMJCEFIPS, è necessario aggiungere un collegamento simbolico al proprio indirizzario dell'estensione tramite l'inoltro del seguente comando:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjcefips.jar')
NEWLNK(< il proprio indirizzario di estensione >)
```

Inoltre, sarà necessario aggiungere il fornitore all'elenco dei fornitori tramite l'inserimento di una voce nel file java.security file (ad esempio, `security.provider.4=com.ibm.crypto.fips.provider.IBMJCEFIPS`), o tramite l'utilizzo del metodo Security.addProvider().

Utilizzo della crittografia hardware

L'implementazione IBMJCECAI5OS estende JCE (Java) e JCA (Java Cryptography Architecture) per aggiungere le funzioni necessarie per utilizzare la crittografia hardware tramite le interfacce IBM CCA (Common Cryptographic Architecture).

Il fornitore IBMJCECAI5OS si avvale della crittografia hardware nell'architettura JCE esistente e dà ai programmatori Java 2 i notevoli vantaggi relativi a sicurezza e prestazioni della crittografia hardware con delle modifiche minime alle applicazioni Java esistenti. Grazie al fatto che gli aspetti complessi della crittografia hardware sono gestiti nella JCE normale, la sicurezza e le prestazioni avanzate per mezzo delle unità crittografiche hardware sono rese facilmente disponibili. Il fornitore IBMJCECCAI5OS si collega al framework JCE in modo analogo ai fornitori correnti. Per le richieste hardware, le API CCA vengono richiamate tramite i nuovi metodi nativi. Il fornitore IBMJCECCAI5OS memorizza le etichette di chiavi RSA CCA in un nuovo tipo di memorizzazione chiavi java JCECCAI5OSKS.

Requisiti per la crittografia hardware

Per utilizzare la crittografia hardware, è necessario che sul proprio sistema sia installato quanto segue:

- Un coprocessore crittografico modello 4764
- IBM i5/OS (5761-SS1) Opzione 35 - CCA Cryptographic Service Provider
- LPO (Licensed Program Offering) 5733-CY1 - IBM eServer iSeries Cryptographic Device Manager

Funzioni del fornitore di crittografia hardware IBM

Il fornitore IBMJCECCAI5OS supporta i seguenti algoritmi:

Tabella 12. Algoritmi supportati dal fornitore IBMJCECCAI5OS

algoritmi di firma	algoritmi di codifica	MAC (Message authentication code)	Messaggio digest
SHA1withRSA MD2WithRSA MD5WithRSA	RSA	HmacMD2 HmacMD5 HmacSHA1	MD2 MD5 SHA-1

Il fornitore IBMJCECCAI5OS comprende anche un potente PRNG (Pseudo Random Number Generator), una generazione delle chiavi tramite le factory di chiavi e la generazione e la gestione di chiavi/certificati tramite un'applicazione `keytool`.

Il fornitore di accesso crittografico hardware è disponibile utilizzando l'applicazione `hwkeytool`.

Nota: il fornitore IBMJCECCAI5OS non può essere aggiunto alla JVM utilizzando i metodi insertProviderAt() e addProviderAt().

Proprietà del sistema di crittografia

È possibile utilizzare le seguenti proprietà di sistema per gestire le unità crittografiche:

i5os.crypto.device
Specifica l'unità crittografica da utilizzare. Se questa proprietà non è impostata, viene utilizzata l'unità predefinita CRP01.

i5os.crypto.keystore
Specifica il file memorizzazione chiave CCA da utilizzare. Se questa proprietà non è impostata, viene utilizzato il file memorizzazione chiave indicato nella descrizione dell'unità crittografica.

Coprocessore crittografico 4764

"Applicazione hwkeytool Java" a pagina 422
L'applicazione `hwkeytool` consente di utilizzare le capacità di crittografia del coprocessore crittografico modello 4764 con JCE (Java Cryptography Extension) e JCA (Java Cryptography Architecture).

"Elenco delle proprietà di sistema Java" a pagina 16
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Coppie di chiavi e utilizzo dell'hardware:

Nell'ambiente di crittografia hardware, è possibile utilizzare il coprocessore crittografico con due tipi di coppie di chiavi, RETAINED e PKDS (Public Key Data Set).

Entrambi i tipi di coppie di chiavi hardware supportati dal fornitore IBMJCECAI5OS possono essere utilizzati da qualsiasi applicazione. Le coppie di chiavi RETAINED e PKDS restituiscono ognuna un'etichetta, che viene trattata come dal fornitore IBMJCECAI5OS in modo analogo ad una chiave. È possibile scegliere il meccanismo con il quale l'applicazione memorizzerà l'etichetta.

Il fornitore IBMJCECAI5OS memorizza le chiavi RSA in un file memorizzazione chiave CCA (Common Cryptographic Architecture) codificato sotto una chiave principale, che si trova all'interno del coprocessore crittografico 4764. La memorizzazione chiave JCECCAI5OSKS memorizza le etichette dei record di chiavi nella memorizzazione chiave CCA.

Coppie di chiavi hardware RETAINED
L'implementazione più sicura della crittografia supportata dal fornitore IBMJCECAI5OS consiste nel memorizzare le chiavi nell'unità crittografica hardware effettiva e non consentire mai il richiamo o la visualizzazione della chiave privata, la parte più delicata della coppia di chiavi. Questa è detta coppia di chiavi *conservata* (RETAINED) poiché la chiave privata è conservata sull'unità hardware senza che ne sia mai consentito il richiamo o la visualizzazione in formato non codificato. Quanto viene restituito all'applicazione o memorizzato nella memorizzazione chiave in fase di generazione della coppia di chiavi è solo un riferimento alla chiave privata detto *etichetta*.

Quando la chiave è necessaria, la richiesta viene inviata alla scheda hardware dove è conservata la chiave, accompagnata dall'etichetta della chiave. L'operazione crittografica viene eseguita su tale scheda utilizzando la chiave conservata e vengono quindi restituiti i risultati. Le chiavi conservate rappresentano il più sicuro dei tipi di chiavi. L'utilizzo delle chiavi conservate

presenta lo svantaggio che non è possibile eseguire operazioni di backup e di ripristino. Se si verifica un malfunzionamento della scheda, le chiavi vanno perdute.

Coppie di chiavi hardware PKDS (Public Key Data Set)
L'altra opzione per l'utilizzo delle chiavi RSA è tramite la coppia di chiavi PKDS. Quando viene generato questo tipo di coppia di chiavi, la chiave privata viene crittografata con la chiave principale del coprocessore in modo che la versione in testo non codificato di questa chiave non possa mai essere visualizzata o richiamata. La coppia di chiavi viene memorizzata in un file di database DB2. Quanto viene restituito all'applicazione o memorizzato nella memorizzazione chiave in fase di generazione della coppia di chiavi è solo un riferimento alla chiave privata detto *etichetta*. Memorizzando la coppia di chiavi in un file, si dispone di un metodo per eseguire il backup delle chiavi ed eseguire attività di ripristino in caso di malfunzionamento delle schede.

JSSE (Java Secure Socket Extension)

JSSE (Java Secure Socket Extension) è simile a un framework che astrae i meccanismi sottostanti sia di SSL (Secure Sockets Layer) che di TLS (Transport Layer Security). Astraendo la complessità e le peculiarità dei protocolli sottostanti, JSSE consente ai programmatori di utilizzare comunicazioni sicure e codificate, riducendo allo stesso tempo possibili vulnerabilità della sicurezza. JSSE (Java Secure Socket Extension) utilizza sia il protocollo SSL che il protocollo TLS per fornire delle comunicazioni sicure e codificate tra i client ed i server.

SSL/TLS fornisce un mezzo di autenticazione di un server e di un client per garantire riservatezza e integrità di dati. Tutte le comunicazioni SSL/TLS cominciano con una ″presentazione″ (handshake) tra il server e il client. Durante l'handshake, SSL/TLS negozia il pacchetto di crittografia che il client e il server utilizzano per comunicare tra di loro. Questo pacchetto di crittografia è una combinazione di varie funzioni di sicurezza disponibili tramite SSL/TLS.

JSSE effettua le seguenti operazioni per migliorare la sicurezza dell'applicazione:
- Protegge i dati di comunicazione tramite codifica.
- Autentica gli ID utente remoto.
- Autentica i nomi del sistema remoto.

Nota: JSSE utilizza un certificato digitale per codificare la comunicazione socket dell'applicazione Java. I certificati digitali sono uno standard Internet per identificare le applicazioni, gli utenti e i sistemi sicuri. È possibile controllare i certificati digitali utilizzando IBM Digital Certificate Manager. Per ulteriori informazioni, consultareIBM Digital Certificate Manager.

Per rendere l'applicazione Java più sicura utilizzando JSSE:
- Preparare System i5 a supportare JSSE.
- Progettare l'applicazione Java in modo che utilizzi JSSE, tramite le seguenti operazioni:
  - Modifica del codice socket Java per l'utilizzo delle produzioni socket, se non sono già utilizzate.
  - Modifica del codice Java per l'utilizzo di JSSE.
- Utilizzare un certificato digitale per rendere la propria applicazione Java più sicura effettuando le seguenti operazioni:
  1. Selezione di un tipo di certificato digitale da utilizzare.
  2. Utilizzo del certificato digitale quando si esegue l'applicazione.

È inoltre possibile registrare la propria applicazione Java come applicazione sicura utilizzando l'API QsyRegisterAppForCertUse.

Preparazione del sistema per il supporto SSL (secure sockets layer)

Per preparare System i5 per l'utilizzo di SSL (secure sockets layer), è necessario installare il Digital Certificate Manager LP.

Installare il Digital Certificate Manager LP, 5761-SS1 i5/OS - Digital Certificate Manager.

È necessario inoltre assicurarsi della possibilità di accedere o creare un certificato digitale sul sistema.

Informazioni correlate

Digital Certificate Manager

Modifica del codice Java per l'utilizzo delle produzioni socket

Per utilizzare SSL (secure socket layer) con il codice esistente, è necessario modificare il codice in modo da utilizzare le produzioni socket.

Per modificare il codice allo scopo di utilizzare le produzioni socket, effettuare quanto segue:

1. Aggiungere questa riga al programma per importare la classe SocketFactory:
   ```
   import javax.net.*;
   ```
2. Aggiungere una riga che dichiari un'istanza di un oggetto SocketFactory. Ad esempio:
   ```
   SocketFactory socketFactory
   ```
3. Inizializzare l'istanza SocketFactory impostandola come per il al metodo SocketFactory.getDefault(). Ad esempio:
   ```
   socketFactory = SocketFactory.getDefault();
   ```
   L'intera dichiarazione del SocketFactory deve risultare nel modo seguente:
   ```
   SocketFactory socketFactory = SocketFactory.getDefault();
   ```
4. Inizializzare i socket esistenti. Chiamare il metodo SocketFactory createSocket(host,port) nella produzione di socket per ogni socket dichiarato.

   Le dichiarazioni di socket devono adesso risultare nel modo seguente:
   ```
   Socket s = socketFactory.createSocket(host,port);
   ```

   Dove:
   - *s* rappresenta il socket che si sta creando.
   - *socketFactory* rappresenta la SocketFactory creata nella fase 2.
   - *host* rappresenta una variabile di stringa che indica il nome di un server host.
   - *port* rappresenta una variabile a numero intero che indica il numero porta del collegamento socket.

Quando tutte queste fasi sono state completate, il codice utilizza le produzioni socket. Non è necessario apportare altre modifiche ad esso. Funzionano ancora tutti i metodi chiamati e le sintassi con i socket.

Esempi: modifica del codice Java per l'utilizzo delle produzioni socket del server:

Questi esempi indicano il modo in cui modificare una classe semplice di socket, denominata simpleSocketServer, cosicché questa utilizzi le produzioni socket per creare tutti i socket. Il primo esempio mostra la classe simpleSocketServer senza produzioni socket. Il secondo esempio mostra la classe simpleSocketServer con produzioni socket. Nel secondo esempio, simpleSocketServer viene ridenominato factorySocketServer.

Esempio 1: programma del server di socket senza produzioni socket

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* File simpleSocketServer.java*/

import java.net.*;
import java.io.*;

public class simpleSocketServer {
  public static void main (String args[]) throws IOException {
```

```
    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    ServerSocket    serverSocket =
      new ServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete gli elementi inviati...

    byte buffer[] = new byte[4096];

    int bytesRead;

    // leggere fino all'"eof" restituito
    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead); // riscriverlo
      os.flush();    // flush del buffer di emissione
    }


    s.close();
    serverSocket.close();
  }       // end main()

}       // end class definition
```

Esempio 2: programma del server di socket con produzioni socket

```
/* File factorySocketServer.java  */

// importare javax.net per selezionare la classe ServerSocketFactory
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Modificare il simpleSocketServer originale per utilizzare un
    // ServerSocketFactory per creare socket server.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
```

```
    // Ora fare in modo che la produzione crei il socket server. Questa è l'ultima
    // modifica rispetto al programma originale.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete gli elementi inviati...

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }


    s.close();
    serverSocket.close();
  }

}
```

Esempi: modifica del codice Java per l'utilizzo delle produzioni socket del client:

Questi esempi indicano il modo in cui modificare una classe semplice di socket, denominata simpleSocketClient, cosicché questa utilizzi le produzioni socket per creare tutti i socket. Il primo esempio mostra la classe simpleSocketClient senza produzioni socket. Il secondo esempio mostra la classe simpleSocketClient con produzioni socket. Nel secondo esempio, simpleSocketClient viene ridenominato factorySocketClient.

Esempio 1: programma del client di socket senza produzioni socket

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* Programma client socket semplice */

import java.net.*;
import java.io.*;

public class simpleSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Creare il socket e collegarsi al server.
    Socket  s = new Socket(args[0], serverPort);
```

```
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Esempio 2: programma del client di socket semplice con produzioni socket

```
/* Programma client produzione socket semplice */

// Notare che javax.net.* viene importato per selezionare la classe SocketFactory.
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Modificare il programma simpleSocketClient originale per creare un
    // SocketFactory e utilizzarlo per creare i socket.

    SocketFactory socketFactory = SocketFactory.getDefault();

    // Ora la produzione crea il socket. Questa è l'ultima modifica
    // al programma simpleSocketClient originale.

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Modifica del codice Java per l'utilizzo di SSL (secure sockets layer)

Se il codice utilizza già produzioni socket per creare i socket relativi, è possibile aggiungere il supporto SSL (secure socket layer) al programma.

Se il codice non utilizza ancora le produzioni socket, consultare Modifica del codice Java per l'utilizzo delle produzioni socket.

Per modificare il codice in modo da utilizzare SSL, effettuare quanto segue:

1. Importare javax.net.ssl.* per aggiungere il supporto SSL:
   ```
   import javax.net.ssl.*;
   ```
2. Dichiarare una SocketFactory utilizzando SSLSocketFactory per inizializzarla:
   ```
   SocketFactory newSF = SSLSocketFactory.getDefault();
   ```
3. Utilizzare la nuova SocketFactory per inizializzare i socket nello stesso modo in cui si è utilizzata la vecchia SocketFactory:
   ```
   Socket s = newSF.createSocket(args[0], serverPort);
   ```

Il codice adesso utilizza il supporto SSL. Non è necessario apportare altre modifiche al codice.

Esempi: modifica del server Java per l'utilizzo di SSL (secure sockets layer):

Questi esempi indicano il modo in cui modificare una classe, denominata factorySocketServer, per utilizzare SSL (secure socket layer).

Il primo esempio mostra la classe factorySocketServer che non utilizza SSL. Il secondo esempio mostra la stessa classe, ridenominata factorySSLSocketServer, che utilizza SSL.

Esempio 1: classe factorySocketServer semplice senza supporto SSL

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* File factorySocketServer.java  */
// importare javax.net per selezionare la classe ServerSocketFactory
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Modificare il simpleSocketServer originale per utilizzare un
    // ServerSocketFactory per creare socket server.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ora fare in modo che la produzione crei il socket server. Questa è l'ultima
    // modifica rispetto al programma originale.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete solo gli elementi inviati.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

Esempio 2: classe factorySocketServer semplice con supporto SSL

```
/* File factorySocketServer.java  */

// importare javax.net per selezionare la classe ServerSocketFactory
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Modificare il simpleSocketServer originale per utilizzare un
    // ServerSocketFactory per creare socket server.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ora fare in modo che la produzione crei il socket server. Questa è l'ultima
    // modifica rispetto al programma originale.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete solo gli elementi inviati.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

Esempi: modifica del client Java per l'utilizzo di SSL (secure sockets layer):

Questi esempi indicano il modo in cui modificare una classe, denominata factorySocketClient, in modo da utilizzare SSL (secure socket layer). Il primo esempio mostra la classe factorySocketClient che non utilizza SSL. Il secondo esempio mostra la stessa classe, ridenominata factorySSLSocketClient, che utilizza SSL.

Esempio 1: classe factorySocketClient semplice senza supporto SSL

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* Programma client produzione socket semplice */

import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    SocketFactory socketFactory = SocketFactory.getDefault();

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Esempio 2: classe factorySocketClient semplice con supporto SSL

```
// Notare che è stato importato javax.net.ssl.* per selezionare il supporto SSL
import javax.net.ssl.*;
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySSLSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySSLSocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Modificare ciò per creare un SSLSocketFactory invece di un SocketFactory.
    SocketFactory socketFactory = SSLSocketFactory.getDefault();

    // Non è necessario modificare altro.
    // Questo è il vantaggio di utilizzare le produzioni!
    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Selezione di un certificato digitale

È necessario considerare numerosi fattori quando si decide quale certificato digitale utilizzare. È possibile utilizzare il certificato predefinito del sistema oppure specificare un altro certificato da utilizzare.

È possibile utilizzare il certificato predefinito del sistema se:

- Non si possiede alcun requisito specifico di sicurezza per l'applicazione Java.
- Non si conosce quale tipo di sicurezza è necessario per l'applicazione Java.
- Il certificato predefinito del sistema soddisfa i requisiti di sicurezza relativi all'applicazione Java.

Nota: se si desidera utilizzare il certificato predefinito del sistema, controllare con il responsabile di sistema per accertarsi che sia stato creato un certificato del sistema predefinito.

Se non si desidera utilizzare il certificato predefinito del sistema, è necessario scegliere un altro certificato da utilizzare. È possibile scegliere tra due tipi di certificati:

- **Certificato utente** che identifica l'utente dell'applicazione.
- **Certificato di sistema** che identifica il sistema su cui è in esecuzione l'applicazione.

È opportuno utilizzare un certificato utente se:

- l'applicazione è in esecuzione come un'applicazione client.
- si desidera che il certificato identifichi l'utente che gestisce l'applicazione.

È necessario utilizzare il certificato di sistema se:

- l'applicazione viene eseguita come un'applicazione del server.
- si desidera che il certificato identifichi il sistema sul quale l'applicazione è in esecuzione.

Una volta che si conosce il tipo di certificato necessario, è possibile scegliere da qualsiasi certificato digitale in qualsiasi memorizzazione certificati cui si è in grado di accedere.

Informazioni correlate

Digital Certificate Manager

Utilizzo del certificato digitale quando si esegue l'applicazione Java

Per utilizzare SSL (secure socket layer), è necessario eseguire l'applicazione Java utilizzando un certificato digitale.

Per specificare quale certificato digitale utilizzare, usare le seguenti proprietà:

- os400.certificateContainer
- os400.certificateLabel

Ad esempio, se si desidera eseguire l'applicazione Java MyClass.class utilizzando il certificato digitale MYCERTIFICATE e se MYCERTIFICATE si trovava nella memorizzazione certificati digitali YOURDCC, il comando `java` sarà:

```
java -Dos400.certificateContainer=YOURDCC
     -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

Se non si è ancora deciso quale certificato digitale utilizzare, consultare "Selezione di un certificato digitale". È possibile inoltre decidere di utilizzare il certificato predefinito del sistema, memorizzato nella relativa memorizzazione.

Per utilizzare il certificato digitale predefinito del sistema, non è necessario alcuna specificazione di un certificato o di una memorizzazione certificati. L'applicazione Java utilizza automaticamente il certificato digitale predefinito del sistema.

Certificati digitali e proprietà -os400.certificateLabel

I certificati digitali sono uno standard Internet per identificare le applicazioni, gli utenti e i sistemi sicuri. Essi vengono memorizzati nelle relative memorizzazioni. Se si desidera utilizzare un certificato predefinito della relativa memorizzazione, non è necessario specificare un'etichetta del certificato. Se si desidera utilizzare un certificato digitale specifico, è necessario specificare l'etichetta di tale certificato presente nel comando `java` utilizzando questa proprietà.

```
os400.certificateLabel=
```

Ad esempio, se il nome del certificato che si desidera utilizzare è MYCERTIFICATE, il comando `java` da immettere sarà:

```
java -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

In questo esempio, l'applicazione Java MyClass utilizza il certificato MYCERTIFICATE. È necessario che MYCERTIFICATE si trovi nella memorizzazione certificati predefinita del sistema per essere utilizzato da MyClass.

Memorizzazioni certificati digitali e proprietà -os400.certificateContainer

Le memorizzazioni certificati digitali memorizzano certificati digitali. Se si desidera utilizzare la memorizzazione certificati predefinita del sistema System i5, non è necessario specificare una memorizzazione certificati. Per utilizzare una memorizzazione certificati digitali specifica, è necessario specificare tale memorizzazione nel comando `java` utilizzando questa proprietà:

```
os400.certificateContainer=
```

Ad esempio, se il nome della memorizzazione certificati che contiene il certificato digitale che si desidera utilizzare è denominata MYDCC, allora il comando `java` da immettere potrebbe assumere questa forma:

```
java -Dos400.certificateContainer=MYDCC MyClass
```

In questo esempio, l'applicazione Java, denominata MyClass.class, viene eseguita nel sistema utilizzando il certificato digitale predefinito che si trova nella relativa memorizzazione denominata MYDCC. Qualsiasi socket creato nell'applicazione utilizza il certificato predefinito che si trova in MYDCC per l'identificazione e rende tutte le comunicazioni sicure.

Se si desiderasse utilizzare il certificato digitale MYCERTIFICATE nella relativa memorizzazione, allora il comando `java` che bisogna immettere dovrebbe assumere questa forma:

```
java -Dos400.certificateContainer=MYDCC
     -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

Informazioni correlate

Digital Certificate Manager

Utilizzo di Java Secure Socket Extension 1.4

Queste informazioni sono valide solo per l'utilizzo di JSSE su un System i5 che esegue J2SDK, versione 1.4. JSSE agisce come una framework che estrae i meccanismi sottostanti sia di SSL che di TLS. Estraendo la complessità e le peculiarità dei protocolli sottostanti, JSSE permette ai programmatori di utilizzare comunicazioni sicure e codificate, riducendo allo stesso tempo la vulnerabilità della sicurezza. JSSE (Java Secure Socket Extension) utilizza entrambi i protocolli SSL (Secure Sockets Layer) e TLS (Transport Layer Security) per fornire comunicazioni sicure e codificate tra i client e i server.

L'implementazione IBM di JSSE viene denominata IBM JSSE. IBM JSSE include un fornitore JSSE System i5 nativo ed un fornitore Java JSSE puro.

Configurazione del sistema per supportare JSSE 1.4:

Configurare il sistema per utilizzare IBM JSSE. Questo argomento include i requisiti software, le istruzioni su come modificare i fornitori JSSE e le proprietà di sistema e della sicurezza necessarie.

Quando si utilizza J2SDK (Java 2 Software Development Kit), versione 1.4 o una versione successiva su System i5, JSSE è già configurato. La configurazione predefinita utilizza il fornitore JSSE System i5 nativo.

Modifica dei fornitori JSSE

È possibile configurare JSSE per utilizzare il fornitore JSSE Java puro invece del fornitore JSSE System i5 nativo. Modificando specifiche proprietà della sicurezza JSSE e proprietà di sistema Java, è possibile passare da un fornitore all'altro.

Gestori della sicurezza

Se l'applicazione JSSE è in esecuzione con un gestore della sicurezza Java abilitato, potrebbe essere necessario impostare le autorizzazioni alla rete disponibile. Per ulteriori informazioni, consultare SSL in Permissions in the Java 2 SDK.

Fornitori JSSE 1.4:

JSSE IBM JSSE include un fornitore System i5 nativo e due fornitori JSSE Java nativi. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

I tre fornitori rispettano le specifiche dell'interfaccia JSSE. Sono in grado di comunicare tra di loro e con qualsiasi altra implementazione SSL o TLS, perfino con implementazioni non Java.

Fornitore JSSE Java puro

Il fornitore JGSS Java puro offre le seguenti caratteristiche:

- Gestisce qualsiasi tipo di oggetto KeyStore per controllare e configurare i certificati digitali (ad esempio, JKS, PKCS12 e così via).
- Permette di utilizzare contemporaneamente qualsiasi combinazione di componenti JSSE provenienti da più implementazioni.

IBMJSSE è il nome del fornitore per l'implementazione Java pura. È necessario inoltrare questo nome fornitore, rispettando i caratteri maiuscoli e minuscoli, al metodo java.security.Security.getProvider() o ai vari metodi getInstance() per diverse classi JSSE.

Fornitore JSSE FIPS 140-2 Java puro

Il fornitore JGSS FIPS 140-2 Java puro, fornisce le seguenti funzionalità:

- È compatibile con FIPS (Federal Information Processing Standards) 140-2 per i moduli crittografici.
- Gestisce qualsiasi tipo di oggetto KeyStore per il controllo e la configurazione dei certificati digitali.

Nota: il fornitore JSSE FIPS 140-2 Java puro non permette a componenti provenienti da un'altra implementazione di inserirsi nella relativa implementazione.

IBMJSSEFIPS è il nome fornitore per l'implementazione JSSE FIPS 140-2 Java puro. È necessario inoltrare questo nome fornitore, rispettando i caratteri maiuscoli e minuscoli, al metodo java.security.Securirty.getProvider() o ai vari metodi getInstance() per diverse classi di JSSE.

Fornitore JSSE System i5 nativo

Il fornitore System i5 JSSE nativo offre le seguenti funzioni:

- Utilizza il supporto SSL System i5 nativo.

- Permette l'utilizzo di DCM (Digital Certificate Manager) per configurare e controllare i certificati digitali. Ciò viene fornito tramite un tipo System i5 univoco di KeyStore (IbmISeriesKeyStore).
- Fornisce migliori prestazioni.
- Permette di utilizzare contemporaneamente qualsiasi combinazione di componenti JSSE provenienti da più implementazioni. Tuttavia, per ottenere delle prestazioni ottimali, utilizzare solo i componenti JSEE System i5 nativi.

IbmISeriesSslProvider è il nome per l'implementazione nativa di System i5. È necessario inoltrare questo nome fornitore, rispettando i caratteri maiuscoli e minuscoli, al metodo java.security.Security.getProvider() o ai vari metodi getInstance() per diverse classi JSSE.

Modifica del fornitore JSSE predefinito

È possibile cambiare il fornitore JSSE predefinito apportando le modifiche appropriate alle proprietà della sicurezza. Per ulteriori informazioni, consultare l'argomento relativo alle proprietà di sicurezza JSSE.

Una volta modificato il fornitore JSSE, assicurarsi che le proprietà di sistema specifichino la corretta configurazione per le informazioni sul certificato digitale (memorizzazione chiave) richiesta dal nuovo fornitore. Per ulteriori informazioni, consultare l'argomento relativo alle proprietà di sistema Java.

Proprietà di sicurezza JSSE 1.4:

Una JVM (Java virtual machine) utilizza diverse importanti proprietà della sicurezza che è possibile impostare modificando il file delle proprietà della sicurezza principale Java.

Questo file, denominato java.security, si trova, generalmente, nell'indirizzario /QIBM/ProdData/Java400/jdk14/lib/security sul server iSeries.

L'elenco che segue descrive diverse importanti proprietà della sicurezza per l'utilizzo di JSSE. Utilizzare le descrizioni come guida alla modifica del file java.security.

security.provider.<numero intero>

Il fornitore JSSE che si desidera utilizzare. Registra anche staticamente le classi del fornitore crittografico. Specificare i diversi fornitori JSSE esattamente come nell'esempio che segue:

```
security.provider.5=com.ibm.as400.ibmonly.net.ssl.Provider
security.provider.6=com.ibm.jsse.IBMJSSEProvider
security.provider.7=com.ibm.fips.jsse.IBMJSSEFIPSProvider
```

ssl.KeyManagerFactory.algorithm

Specifica l'algoritmo KeyManagerFactory predefinito. Per il fornitore JSSE nativo di iSeries, utilizzare:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

Per il fornitore JSSE nativo di Java, utilizzare:

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

Per ulteriori informazioni, consultare javadoc per javax.net.ssl.KeyManagerFactory.

ssl.TrustManagerFactory.algorithm

Specifica l'algoritmo TrustManagerFactory predefinito. Per il fornitore JSSE nativo di iSeries, utilizzare:

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

Per il fornitore JSSE nativo di Java, utilizzare:

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

Per ulteriori informazioni, consultare javadoc per javax.net.ssl.TrustManagerFactory.

ssl.SocketFactory.provider

Specifica la produzione socket SSL predefinita. Per il fornitore JSSE nativo di iSeries, utilizzare:

```
ssl.SocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLSocketFactoryImpl
```

Per il fornitore JSSE nativo di Java, utilizzare:

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
```

Per ulteriori informazioni, consultare javadoc per javax.net.ssl.SSLSocketFactory.

ssl.ServerSocketFactory.provider

Specifica la produzione socket del server SSL predefinita. Per il fornitore JSSE nativo di iSeries, utilizzare:

```
ssl.ServerSocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLServerSocketFactoryImpl
```

Per il fornitore JSSE nativo di Java, utilizzare:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

Per ulteriori informazioni, consultare javadoc per javax.net.ssl.SSLServerSocketFactory.

Proprietà di sistema Java JSSE 1.4:

Per utilizzare JSSE nelle applicazioni, è necessario specificare diverse proprietà di sistema necessarie agli oggetti SSLContext predefiniti per fornire la conferma della configurazione. Alcune proprietà si applicano ad entrambi i fornitori, mentre altre si applicano solo al fornitore nativo di System i5.

Quando si utilizza il fornitore JSSE nativo di System i5, se non si specifica alcuna proprietà, os400.certificateContainer viene impostato sul valore predefinito *SYSTEM, cioè JSSE utilizzerà la voce predefinita nella memoria certificati di sistema.

Proprietà che si applicano ad entrambi i fornitori

Le proprietà che seguono si applicano ad entrambi i fornitori JSSE. Ciascuna descrizione comprende la proprietà predefinita, se applicabile.

javax.net.ssl.trustStore

Il nome del file che contiene l'oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi. Il valore predefinito è jssecacerts o cacerts (se jssecacerets non esiste).

javax.net.ssl.trustStoreType

Il tipo di oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi. Il valore predefinito è quello restituito dal metodo KeyStore.getDefaultType.

javax.net.ssl.trustStorePassword

La parola d'ordine per l'oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi.

javax.net.ssl.keyStore

Il nome del file che contiene l'oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi.

javax.net.ssl.keyStoreType

Il tipo di oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi. Il valore predefinito è quello restituito dal metodo KeyStore.getDefaultType.

javax.net.ssl.keyStorePassword

La parola d'ordine per l'oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi.

Proprietà valide solo per il fornitore System i5 JSSE nativo

Le seguenti proprietà si applicano solo al fornitore System i5 JSSE nativo.

os400.secureApplication

L'identificativo dell'applicazione. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType

os400.certificateContainer

Il nome del file di chiavi che si desidera utilizzare. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

os400.certificateLabel

L'etichetta del file di chiavi che si desidera utilizzare. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword

- javax.ssl.net.trustStoreType
- os400.secureApplication

Concetti correlati

"Elenco delle proprietà di sistema Java" a pagina 16
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Informazioni correlate

Proprietà di sistema sul sito Web di Sun Java

Utilizzo del fornitore System i5 JSSE 1.4 nativo:

Il fornitore System i5 JSSE nativo offre la serie completa di classi e interfacce JSSE, comprese le implementazioni delle classi SSLConfiguration e Keystore JSSE.

Per utilizzare in modo efficace il fornitore System i5 nativo, utilizzare le informazioni contenute in quest'argomento e consultare anche le informazioni Javadoc su SSLConfiguration per JSSE 1.4.

Valori protocollo per il metodo SSLContext.getInstance

La seguente tabella identifica e descrive i valori di protocollo per il metodo SSLContext.getInstance del fornitore System i5 JSSE nativo.

I protocolli SSL supportati possono essere limitati dai valori di sistema impostati sul sistema. Per ulteriori dettagli, consultare l'argomento secondario relativo ai valori di sistema di sicurezza: protocolli SSL (Secure Sockets Layer) nelle informazioni sulla gestione dei sistemi.

Valore protocollo	Protocolli SSL supportati
SSL	SSL versione 2, SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.
SSLv2	SSL versione 2
SSLv3	Protocollo SSL versione 3. Accetterà l'hello SSLv3 incapsulato in un hello formato SSLv2.
TLS	SSL versione 2, SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.
TLSv1	Protocollo TLS versione 1, definito in RFC (Request for Comments) 2246. Accetterà l'hello TLSv1 incapsulato in un hello formato SSLv2.
SSL_TLS	SSL versione 2, SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.

Implementazione KeyStore nativa di System i5

Il fornitore System i5 nativo offre un'implementazione della classe KeyStore di tipo IbmISeriesKeyStore. Tale implementazione memorizzazione chiave fornisce un wrapper al supporto DCM (Digital Certificate Manager). Il contenuto della memorizzazione chiave si basa su uno specifico identificativo dell'applicazione o file di chiavi, parola d'ordine ed etichetta. JSSE carica le voci memorizzazione chiave dal DCM. Per caricare le voci, JSSE utilizza l'appropriato identificativo dell'applicazione o informazioni del file di chiavi quando l'applicazione effettua il primo tentativo di accesso alle voci o alle informazioni memorizzazione chiave. Non è possibile modificare la memorizzazione chiave ed è necessario effettuare

tutte le modifiche alla configurazione utilizzando il DCM (Digital Certificate Manager).

Consigli quando si utilizza il fornitore System i5 nativo

Quelli che seguono sono dei consigli per eseguire nel modo più efficiente possibile il fornitore System i5 nativo.

- Per fare in modo che il fornitore JSSE nativo di System i5 funzioni, è necessario che l'applicazione JSSE utilizzi solo componenti provenienti dall'implementazione nativa. Ad esempio, l'applicazione abilitata a JSSE nativo di System i5 non può utilizzare un oggetto X509KeyManager creato utilizzando il fornitore JSSE Java puro per inizializzare con esito positivo un oggetto SSLContext creato utilizzando il fornitore JSSE nativo di System i5.
- Inoltre, è necessario inizializzare le implementazioni di X509KeyManager e X509TrustManager nel fornitore nativo di System i5 utilizzando un oggetto IbmISeriesKeyStore oppure un oggetto com.ibm.as400.SSLConfiguration.

Nota: le raccomandazioni indicate potrebbero cambiare nei futuri release, e quindi il fornitore JSSE nativo di System i5 potrebbe consentire di collegare dei componenti non nativi (ad esempio, JKS KeyStore o IbmX509 TrustManagerFactory).

Riferimenti correlati

"Informazioni Javadoc su SSLConfiguration per JSSE 1.4"

Informazioni correlate

Digital Certificate Manager

Valori di sistema di sicurezza: protocolli SSL (Secure Sockets Layer)

Informazioni Javadoc su SSLConfiguration per JSSE 1.4:

com.ibm.as400

Classe SSLConfiguration

```
java.lang.Object
  |
  +--com.ibm.as400.SSLConfiguration
```

Tutte le interfacce implementate:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

public final class **SSLConfiguration**
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable

Questa classe si occupa della specifica della configurazione richiesta dall'implementazione System i5 JSSE nativa.

L'implementazione System i5 JSSE nativa funziona in modo ottimale utilizzando un oggetto `KeyStore` di tipo ″IbmISeriesKeyStore″. Questo tipo di oggetto `KeyStore` contiene voci chiavi e voci certificati sicuri basate su un'identificazione dell'applicazione registrata con DCM (Digital Certificate Manager) o su un file di chiavi (memorizzazione certificati digitali). Un oggetto `KeyStore` di questo tipo può essere utilizzato per inizializzare un oggetto `X509KeyManger` e `X509TrustManager` dal fornitore ″IbmISeriesSslProvider″ . Gli oggetti `X509KeyManager` e `X509TrustManager` possono essere utilizzati per inizializzare un oggetto `SSLContext` da ″IbmISeriesSslProvider″. L'oggetto `SSLContext` fornisce quindi l'accesso all'implementazione System i5 JSSE nativa in base alle informazioni relative alla configurazione specificate per l'oggetto `KeyStore`. Ogni volta che si esegue un caricamento per un `KeyStore`

″IbmISeriesKeyStore″, il KeyStore viene inizializzato in base alla configurazione corrente specificata dall'identificativo dell'applicazione o dal file di chiavi.

Questa classe può essere utilizzata anche per creare un oggetto KeyStore valido di qualsiasi tipo. Il KeyStore viene inizializzato in base alla configurazione corrente specificata dall'identificativo dell'applicazione o dal file di chiavi. Qualsiasi modifica effettuata alla configurazione specificata da un identificativo di applicazione o da un file di chiavi richiede che l'oggetto KeyStore venga creato nuovamente per poter applicare la modifica. Si tenga presente che una parola d'ordine del file di chiavi deve essere specificata (per la memoria certificati *SYSTEM quando si utilizza un'ID di applicazione) per poter creare in modo corretto un KeyStore di tipo diverso da ″IbmISeriesKeyStore″. La parola d'ordine del file di chiavi deve essere specificata per poter ottenere l'accesso a qualsiasi chiave privata per qualsiasi KeyStore di tipo ″IbmISeriesKeyStore″ creato.

Da: SDK 1.4

Consultare anche:
KeyStore, X509KeyManager, X509TrustManager, SSLContext

Riepilogo del programma di creazione

SSLConfiguration() crea una nuova SSLConfiguration. Per ulteriori informazioni, consultare "Dettagli sul programma di creazione " a pagina 291.

Tabella 13. **Riepilogo dei metodi**

void	"clear" a pagina 294() Cancella tutte le informazioni all'interno dell'oggetto in modo che tutti i metodi ottenuti restituiscano null.
java.lang.Object	"clone" a pagina 295() Crea una nuova copia di questa configurazione SSL.
boolean	"equals" a pagina 295(java.lang.Objectobj) Indica se alcuni oggetti sono ″uguali a″ questo.
protected void	"finalize" a pagina 294() Richiamato dal programma di raccolta di dati su un oggetto quando la raccolta di dati determina che non esistono più riferimenti all'oggetto.
java.lang.String	"getApplicationId" a pagina 293() Restituisce l'ID dell'applicazione.
java.lang.String	"getKeyringLabel" a pagina 294() Restituisce l'etichetta del file di chiavi.
java.lang.String	"getKeyringName" a pagina 293() Restituisce il nome del file di chiavi.
char[]	"getKeyringPassword" a pagina 294() Restituisce la parola d'ordine del file di chiavi.
java.security.KeyStore	"getKeyStore" a pagina 296(char[]password) Restituisce una memorizzazione chiave di tipo ″IbmISeriesKeyStore″ utilizzando la parola d'ordine specifica.
java.security.KeyStore	" getKeyStore" a pagina 296(java.lang.Stringtype, char[]password) Restituisce una memorizzazione chiave di tipo richiesto utilizzando la parola d'ordine specifica.
int	"hashCode" a pagina 295() Restituisce un valore del codice hash per l'oggetto.
staticvoid	(java.lang.String[]args) Esegue le funzioni SSLConfiguration.
void	(java.lang.String[]args, java.io.PrintStreamout) Esegue le funzioni SSLConfiguration.
void	"setApplicationId" a pagina 295(java.lang.StringapplicationId) Imposta l'ID di applicazione.
void	"setApplicationId" a pagina 295(java.lang.StringapplicationId, char[]password) Imposta l'ID di applicazione e la parola d'ordine del file di chiavi.

Tabella 13. **Riepilogo dei metodi** *(Continua)*

`void`	"setKeyring" a pagina 294(java.lang.Stringname,java.lang.Stringlabel, char[]password) Imposta le informazioni del file di chiavi.

Metodi ereditati dalla classe java.lang.Object
`getClass, notify, notifyAll, toString, wait, wait, wait`

Dettagli sul programma di creazione

SSLConfiguration

```
public SSLConfiguration()
```

Crea una nuova SSLConfiguration. L'identificazione d'applicazione e le informazioni sul file di chiavi vengono inizializzate su valori predefiniti.

Il valore predefinito per l'identificazione d'applicazione è il valore specificato per la proprietà ″os400.secureApplication″.

I valori predefiniti per le informazioni del file di chiavi sono null se la proprietà ″os400.secureApplication″ viene specificata. Se la proprietà ″os400.secureApplication″ non viene specificata, il valore predefinito per il nome del file di chiavi è il valore specificato per la proprietà ″os400.certificateContainer″. Se la proprietà ″os400.secureApplication″ non viene specificata, l'etichetta del file di chiavi viene inizializzata sul valore della proprietà ″os400.certificateLabel″. Se non viene impostata la proprietà ″os400.secureApplication″ o ″os400.certificateContainer″, il nome del file di chiavi verrà inizializzato su ″*SYSTEM″.

Dettagli del metodo

---

main

```
public static void main(java.lang.String[]args)
```

Esegue le funzioni SSLConfiguration. Esistono quattro comandi che possono essere eseguiti: -help, -create, -display e -update. Il comando deve essere il primo parametro specificato.

Le seguenti opzioni possono essere specificate (in qualsiasi ordine):

-keystore **keystore-file-name**
: Specifica il nome del file di chiavi da creare, aggiornare o visualizzare. Questa opzione viene richiesta per tutti i comandi.

-storepass **keystore-file-password**
: Specifica la parola d'ordine associata al file di chiavi da creare, aggiornare o visualizzare. Questa opzione viene richiesta per tutti i comandi.

-storetype **keystore-type**
Specifica il tipo di file di chiavi da creare, aggiornare o visualizzare. Questa opzione può essere specificata per qualsiasi comando. Se questa opzione non viene specificata, viene utilizzato un valore di "IbmISeriesKeyStore".

-appid **application-identifier**
Specifica l'identificazione di applicazione da utilizzare per inizializzare un file di chiavi che si stava creando o aggiornando. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-keyring **keyring-file-name**
Specifica il nome del file di chiavi da utilizzare per inizializzare un file di chiavi che si stava creando o aggiornando. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-keyringpass **keyring-file-password**
Specifica la parola d'ordine del file di chiavi da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione può essere specificata per i comandi *-create* e *-update*, viene richiesta quando viene specificato un tipo memorizzazione chiave diverso da "IbmISeriesKeyStore". Se questa opzione non viene specificata, viene utilizzata la parola d'ordine nascosta del file di chiavi.

-keyringlabel **keyring-file-label**
Specifica l'etichetta del file di chiavi da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione può essere specificata solo quando viene specificata anche l'opzione *-keyring*. Se questa opzione non viene specificata quando viene specificata l'opzione *keyring*, viene utilizzata l'etichetta predefinita all'interno del file di chiavi.

-systemdefault
Specifica il valore predefinito del sistema da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-v Specifica che l'emissione verbose deve essere prodotta. Questa opzione può essere specificata per qualsiasi comando.

Il comando di aiuto visualizza le informazioni sull'utilizzo per specificare i parametri in questo metodo. I parametri per richiamare la funzione di aiuto vengono specificati come segue:

```
-help
```

Il comando di creazione crea un nuovo file memorizzazione chiave. Esistono tre variazioni del comando di creazione. La prima variazione per creare una memorizzazione chiave si basa su una specifica identificazione di applicazione, la seconda su un nome, etichetta, e parola d'ordine del file di chiavi e la terza sulla configurazione predefinita del sistema.

Per creare una parola d'ordine basata su un'identificazione di applicazione specifica, deve essere specificata l'opzione *-appid*. I seguenti parametri creano un file memorizzazione chiave di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base all'identificazione di applicazione "APPID":

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -appid APPID
```

Per creare una memorizzazione chiave basata su un file di chiavi specifico, deve essere specificata l'opzione *-keyring*. Devono essere specificate anche le opzioni *-keyringpass* e *keyringlabel*. I seguenti parametri creano un file di chiavi di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base al file di chiavi denominato "keyring.file", alla parola d'ordine del file di chiavi "ringpass" e all'etichetta del file di chiavi "keylabel":

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

Per creare una memorizzazione chiave basata sulla configurazione predefinita del sistema, deve essere specificata l'opzione the *-systemdefault*. I seguenti parametri creano un file memorizzazione chiave di tipo ″IbmISeriesKeyStore″ denominato ″keystore.file″ con una parola d'ordine di ″keypass″ che viene inizializzata in base alla configurazione predefinita del sistema:

```
-create -keystore keystore.file -storepass keypass -systemdefault
```

Il comando di aggiornamento, aggiorna un file memorizzazione chiave esistente di tipo ″IbmISeriesKeyStore″. Esistono tre variazioni del comando di aggiornamento identiche alle variazioni del comando di creazione. Le opzioni per il comando di aggiornamento sono identiche alle opzioni utilizzate per il comando di creazione. Il comando di visualizzazione, visualizza la configurazione specificata per un file memorizzazione chiave esistente. I seguenti parametri visualizzano la configurazione specificata da un file memorizzazione chiave di tipo ″IbmISeriesKeyStore″ denominato ″keystore.file″ con una parola d'ordine di ″keypass″:

```
-display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

Parametri:

`args` - gli argomenti della riga comandi

---

run

```
public void run(java.lang.String[]args,
                java.io.PrintStreamout)
```

Esegue le funzioni SSLConfiguration. I parametri e la funzionalità di questo metodo sono identici al metodo main().

Parametri:

`args` - gli argomenti del comando

`out` - il flusso di emissione i cui risultati devono essere scritti

Inoltre, consultare:`com.ibm.as400.SSLConfiguration.main()`

---

getApplicationId

```
public java.lang.String getApplicationId()
```

Restituisce l'ID di applicazione.

Restituisce:

l'ID di applicazione.

---

getKeyringName

```
public java.lang.String getKeyringName()
```

Restituisce il nome del file di chiavi.

Restituisce:

il nome del file di chiavi.

---

getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

Restituisce l'etichetta del file di chiavi.

Restituisce:
l'etichetta del file di chiavi.

---

getKeyringPassword

```
public final char[] getKeyringPassword()
```

Restituisce la parola d'ordine del file di chiavi.

Restituisce:
la parola d'ordine del file di chiavi.

---

finalize

```
protected void finalize()
               throws java.lang.Throwable
```

Richiamato dal programma di raccolta di dati su un oggetto quando la raccolta di dati determina che non esistono più riferimenti all'oggetto.

Sovrascrive:
`finalize` nella classe `java.lang.Object`

Emette:
`java.lang.Throwable` - l'eccezione provocata da questo metodo.

---

clear

```
public void clear()
```

Cancella tutte le informazioni all'interno dell'oggetto in modo che tutti i metodi ottenuti restituiscano null.

---

setKeyring

```
public void setKeyring(java.lang.Stringname,
                       java.lang.Stringlabel,
                       char[]password)
```

Imposta le informazioni sul file di chiavi.

Parametri:
`name` - il nome del file di chiavi

`label` - l'etichetta del file di chiavi o null se deve essere utilizzata la voce del file di chiavi predefinito.

`password` - la parola d'ordine del file di chiavi o null se deve essere utilizzata la parola d'ordine nascosta.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId)
```

Imposta l'ID di applicazione.

Parametri:
applicationId - l'ID di applicazione.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId,
                             char[]password)
```

imposta l'ID di applicazione e la parola d'ordine del file di chiavi. La specifica della parola d'ordine del file di chiavi consente a qualsiasi memorizzazione chiave creata di consentire l'accesso alla chiave privata.

Parametri:
applicationId - l'ID di applicazione.

password - la parola d'ordine del file di chiavi.

---

equals

```
public boolean equals(java.lang.Objectobj)
```

Indica se altri oggetti sono ″uguali a″ questo.

Sovrascrive:
equals nella classe java.lang.Object

Parametri:
obj - oggetto da confrontare

Restituisce:
un indicatore che conferma se gli oggetti specificano le stesse informazioni di configurazione

---

hashCode

```
public int hashCode()
```

Restituisce un valore del codice hash per l'oggetto.

Sovrascrive:
hashCode nella classe java.lang.Object

Restituisce:
un valore del codice hash per questo oggetto.

---

clone

```
public java.lang.Object clone()
```

Crea una nuova copia di questa configurazione SSL. Le modifiche successive effettuate ai componenti di questa configurazione SSL, non interessano la nuova copia e viceversa.

Sovrascrive:
`clone` nella classe `java.lang.Object`

Restituisce:
una copia di questa configurazione SSL

---

getKeyStore

```
public java.security.KeyStore getKeyStore(char[]password)
                                  throws java.security.KeyStoreException
```

Restituisce una memorizzazione chiave di tipo "IbmISeriesKeyStore" utilizzando la parola d'ordine specifica. La memorizzazione chiave viene inizializzata in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto.

Parametri:
`password` - utilizzata per inizializzare la memorizzazione chiave

Restituisce:
KeyStore, inizializzato in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto

Emette:
`java.security.KeyStoreException` - se non è stato possibile creare la memorizzazione chiave

---

getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                           char[]password)
                                  throws java.security.KeyStoreException
```

Restituisce una memorizzazione chiave del tipo richiesto utilizzando la parola d'ordine specifica. La memorizzazione chiave viene inizializzata in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto.

Parametri:
`type` - tipo di memorizzazione chiave da restituire

`password` - utilizzata per inizializzare la memorizzazione chiave

Restituisce:
KeyStore, inizializzato in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto

Emette:
`java.security.KeyStoreException` - se non è stato possibile creare la memorizzazione chiave

Esempi: IBM Java Secure Sockets Extension 1.4:

Gli esempi JSSE mostrano il modo in cui un client e un server possono utilizzare il fornitore System i5 JSSE nativo per creare un contesto che abilita le comunicazioni sicure.

Nota: entrambi gli esempi utilizzano il fornitore JSSE nativo di System i5, indipendentemente dalle proprietà specificate dal file java.security.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

Esempio: client SSL che utilizza un oggetto SSLContext per la versione 1.4:

Questo programma client di esempio utilizza un oggetto SSLContext, da esso inizializzato per utilizzare l'ID applicazione ″MY_CLIENT_APP″. Questo programma utilizzerà l'implementazione System i5 nativa indipendentemente da quanto specificato nel file java.security.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
| ////////////////////////////////////////////////////////////////////////////
| //
| // Questo programma client di esempio utilizza un oggetto SSLContext, da esso inizializzato
| // per utilizzare l'ID applicazione "MY_CLIENT_APP".
| //
| // L'esempio utilizza il fornitore JSSE nativo di iSeries, indipendentemente dalle
| // proprietà specificate dal file java.security.
| //
| // Sintassi del comando:
| //     java -Djava.version=1.4 SslClient
| //
| // Notare che "-Djava.version=1.4" non è necessario se è stato configurato
| // come valore predefinito l'utilizzo di J2SDK versione 1.4.
| //
| ////////////////////////////////////////////////////////////////////////////
|
| import java.io.*;
| import javax.net.ssl.*;
| import java.security.*;
| import com.ibm.as400.SSLConfiguration;
|
| /**
|  * Programma client SSL.
|  */
| public class SslClient {
|
|    /**
|     * Metodo principale SslClient.
|     *
|     * @param args gli argomenti della riga comandi (non utilizzato)
|     */
|    public static void main(String args[]) {
|        /*
|         * Impostare in modo da catturare gli errori inviati.
|         */
|        try {
|            /*
|             * Inizializzare un oggetto SSLConfiguration per specificare un ID
|             * ID.  "MY_CLIENT_APP" deve essere registrato e configurato
|             * correttamente con il DCM (Digital Certificate Manager).
|             */
|            SSLConfiguration config = new SSLConfiguration();
|            config.setApplicationId("MY_CLIENT_APP");
|            /*
|             * Richiamare un oggetto KeyStore dall'oggetto SSLConfiguration.
|             */
|            char[] password = "password".toCharArray();
|            KeyStore ks = config.getKeyStore(password);
|            /*
|             * Assegnare ed inizializzare un KeyManagerFactory.
|             */
|            KeyManagerFactory kmf =
|                              KeyManagerFactory.getInstance("IbmISeriesX509");
|            kmf.init(ks, password);
```

```
            /*
             * Assegnare ed inizializzare un TrustManagerFactory.
             */
            TrustManagerFactory tmf =
                            TrustManagerFactory.getInstance("IbmISeriesX509");
            tmf.init(ks);
            /*
             * Assegnare ed inizializzare un SSLContext.
             */
            SSLContext c =
                       SSLContext.getInstance("SSL", "IbmISeriesProvider");
            c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
            /*
             * Ottenere l'SSLSocketFactory dall'SSLContext.
             */
            SSLSocketFactory sf = c.getSocketFactory();
            /*
             * Creare un SSLSocket.
             *
             * Modificare l'indirizzo IP codificato con l'indirizzo IP o il nome host
             * del server.
             */
            SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
            /*
             * Inviare un messaggio al server utilizzando la sessione protetta.
             */
            String sent = "Test of java SSL write";
            OutputStream os = s.getOutputStream();
            os.write(sent.getBytes());
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Wrote " + sent.length() + " bytes...");
            System.out.println(sent);
            /*
             * Ricevere un messaggio dal server utilizzando la sessione protetta.
             */
            InputStream is = s.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead = is.read(buffer);
            if (bytesRead == -1)
                throw new IOException("Unexpected End-of-file Received");
            String received = new String(buffer, 0, bytesRead);
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Read " + received.length() + " bytes...");
            System.out.println(received);
        } catch (Exception e) {
            System.out.println("Unexpected exception caught: " +
                               e.getMessage());
            e.printStackTrace();
        }
    }

}
```

Esempio: server SSL che utilizza un oggetto SSLContext per la versione 1.4:

Il seguente programma server utilizza un oggetto SSLContext da esso inizializzato, con un file memorizzazione chiave precedentemente creato.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////////
//
// Il seguente programma server utilizza un oggetto SSLContext da esso
// inizializzato, con un file memorizzazione chiave precedentemente creato.
//
// Il nome e la parola d'ordine memorizzazione chiave del file memorizzazione chiave sono:
//   Nome file: /home/keystore.file
//   Parola d'ordine:  password
//
// È necessario che il programma di esempio disponga del file memorizzazione chiave per poter creare un
// oggetto IbmISeriesKeyStore. L'oggetto KeyStore deve specificare MY_SERVER_APP come
// identificativo dell'applicazione.
//
// Per creare il file memorizzazione chiave, è possibile utilizzare il seguente comando Qshell:
//
//   java com.ibm.as400.SSLConfiguration  -create  -keystore /home/keystore.file
//     -storepass password  -appid MY_SERVER_APP
//
// Sintassi del comando:
//     java -Djava.version=1.4 JavaSslServer
//
// Notare che "-Djava.version=1.4" non è necessario se è stato configurato
// come valore predefinito l'utilizzo di J2SDK versione 1.4.
//
// È anche possibile creare il file memorizzazione chiave immettendo questo comando su una richiesta comandi i5/OS:
//
//  RUNJVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
//      '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
/**
* Programma server SSL Java che utilizza l'ID applicazione.
*/
public class JavaSslServer {

   /**
    * Metodo principale JavaSslServer.
    *
    * @param args gli argomenti della riga comandi (non utilizzato)
    */
   public static void main(String args[]) {
       /*
        * Impostare in modo da catturare gli errori inviati.
        */
       try {
           /*
            * Assegnare ed inizializzare un oggetto KeyStore.
            */
           char[] password = "password".toCharArray();
           KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
           FileInputStream fis = new FileInputStream("/home/keystore.file");
           ks.load(fis, password);
           /*
            * Assegnare ed inizializzare un KeyManagerFactory.
            */
           KeyManagerFactory kmf =
                           KeyManagerFactory.getInstance("IbmISeriesX509");
           kmf.init(ks, password);
           /*
            * Assegnare ed inizializzare un TrustManagerFactory.
            */
           TrustManagerFactory tmf =
                       TrustManagerFactory.getInstance("IbmISeriesX509");
```

```
            tmf.init(ks);
            /*
             * Assegnare ed inizializzare un SSLContext.
             */
            SSLContext c =
                       SSLContext.getInstance("SSL", "IbmISeriesProvider");
            c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
            /*
             * Richiamare un SSLServerSocketFactory dal SSLContext.
             */
            SSLServerSocketFactory sf = c.getServerSocketFactory();
            /*
             * Creare un SSLServerSocket.
             */
            SSLServerSocket ss =
                (SSLServerSocket) sf.createServerSocket(13333);
            /*
             * Eseguire un accept() per creare un SSLSocket.
             */
            SSLSocket s = (SSLSocket) ss.accept();
            /*
             * Ricevere un messaggio dal client utilizzando la sessione protetta.
             */
            InputStream is = s.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead = is.read(buffer);
            if (bytesRead == -1)
                throw new IOException("Unexpected End-of-file Received");
            String received = new String(buffer, 0, bytesRead);
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Read " + received.length() + " bytes...");
            System.out.println(received);
            /*
             * Rimandare il messaggio al client utilizzando la sessione protetta.
             */
            OutputStream os = s.getOutputStream();
            os.write(received.getBytes());
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Wrote " + received.length() + " bytes...");
            System.out.println(received);
        } catch (Exception e) {
            System.out.println("Unexpected exception caught: " +
                               e.getMessage());
            e.printStackTrace();
        }
    }

}
```

Utilizzo di Java Secure Socket Extension 1.5

Queste informazioni sono valide solo quando si utilizza JSSE sui sistemi che eseguono Java 2 Platform, Standard Edition (J2SE), versione 1.5 e release successivi. JSSE agisce come una framework che estrae i meccanismi sottostanti sia di SSL che di TLS. Estraendo la complessità e le peculiarità dei protocolli sottostanti, JSSE permette ai programmatori di utilizzare comunicazioni sicure e codificate, riducendo allo stesso tempo la vulnerabilità della sicurezza. JSSE (Java Secure Socket Extension) utilizza entrambi i protocolli SSL (Secure Sockets Layer) e TSL (Transport Layer Security) per fornire comunicazioni sicure e codificate tra i client e i server.

L'implementazione IBM di JSSE viene denominata IBM JSSE. IBM JSSE include un fornitore JSSE iSeries nativo e un fornitore JSSE Java puro IBM. Per quanto riguarda inoltre JSSE di Sun Microsystems, Inc., era in precedenza fornito con System i e continuerà ad essere fornito.

Configurazione del server per supportare JSSE 1.5:

Configurare System i5 per utilizzare implementazioni JSSE differenti. Questo argomento include i requisiti software, le istruzioni su come modificare i fornitori JSSE e le proprietà di sistema e della sicurezza necessarie. La configurazione predefinita utilizza il fornitore JSSE Java puro IBM noto come IBMJSSE2.

Quando si utilizza J2SE (Java 2 Platform, Standard Edition), versione 1.5 su System i5, JSSE è già configurato. La configurazione predefinita utilizza il fornitore JSSE System i5 nativo.

Modifica dei fornitori JSSE

È possibile configurare JSSE per utilizzare il fornitore JSSE System i5 nativo oppure il JSSE di Sun Microsystems, Inc. invece del fornitore JSSE Java puro IBM. Modificando specifiche proprietà della sicurezza JSSE e proprietà di sistema Java, è possibile passare da un fornitore all'altro.

Gestori della sicurezza

Se l'applicazione JSSE è in esecuzione con un gestore della sicurezza Java abilitato, potrebbe essere necessario impostare le autorizzazioni alla rete disponibile. Per ulteriori informazioni, consultare SSL in Permissions in the Java 2 SDK.

Fornitori JSSE 1.5:

IBM JSSE include un fornitore System i5 JSSE nativo ed un fornitore Java JSSE puro IBM. JSSE Sun Microsystems, Inc. viene fornito anche con System i5. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

Tutti i fornitori rispettano la specifica dell'interfaccia JSSE. Sono in grado di comunicare tra di loro e con qualsiasi altra implementazione SSL o TLS, perfino con implementazioni non Java.

Fornitore JSSE Java puro Sun Microsystems, Inc.

Questa è l'implementazione Sun Java di JSSE. Era inizialmente fornito con System i e continuerà ad essere fornito. Per ulteriori informazioni su JSSE Sun Microsystems, Inc., consultare il manuale Java Secure Socket Extension (JSSE) Reference Guide di Sun Microsystems, Inc.

Fornitore JSEE Java puro IBM

Il fornitore JSSE Java puro IBM offre le seguenti funzioni:

- Gestisce qualsiasi tipo di oggetto KeyStore per controllare e configurare i certificati digitali (ad esempio, JKS, PKCS12 e così via).
- Permette di utilizzare contemporaneamente qualsiasi combinazione di componenti JSSE provenienti da più implementazioni.

IBMJSSEProvider2 è il nome del fornitore per l'implementazione Java pura. È necessario inoltrare questo nome fornitore, rispettando i caratteri maiuscoli e minuscoli, al metodo java.security.Security.getProvider() o ai vari metodi getInstance() per diverse classi JSSE.

Fornitore JSSE System i5 nativo

Il fornitore System i5 JSSE nativo offre le seguenti funzioni:

- Utilizza il supporto SSL System i5 nativo.
- Permette l'utilizzo di DCM (Digital Certificate Manager) per configurare e controllare i certificati digitali. Ciò viene fornito tramite un tipo System i5 univoco di KeyStore (IbmISeriesKeyStore).

- Permette di utilizzare contemporaneamente qualsiasi combinazione di componenti JSSE provenienti da più implementazioni.

IBMi5OSJSSEProvider è il nome per l'implementazione nativa di System i5. È necessario inoltrare questo nome fornitore, rispettando i caratteri maiuscoli e minuscoli, al metodo java.security.Security.getProvider() o ai vari metodi getInstance() per diverse classi JSSE.

Modifica del fornitore JSSE predefinito

È possibile cambiare il fornitore JSSE predefinito apportando le modifiche appropriate alle proprietà della sicurezza. Per ulteriori informazioni, consultare "Proprietà di sicurezza JSSE 1.5".

Una volta modificato il fornitore JSSE, assicurarsi che le proprietà di sistema specifichino la corretta configurazione per le informazioni sul certificato digitale (memorizzazione chiave) richiesta dal nuovo fornitore. Per ulteriori informazioni, consultare "Proprietà di sistema Java JSSE 1.5" a pagina 303.

Proprietà di sicurezza JSSE 1.5:

Una JVM (Java virtual machine) utilizza diverse importanti proprietà della sicurezza che è possibile impostare modificando il file delle proprietà della sicurezza principale Java.

Questo file, denominato java.security, si trova di solito nell'indirizzario /QIBM/ProdData/Java400/jdk15/lib/security sul server.

L'elenco che segue descrive diverse importanti proprietà della sicurezza per l'utilizzo di JSSE. Utilizzare le descrizioni come guida alla modifica del file java.security.

security.provider.<numero intero>

Il fornitore JSSE che si desidera utilizzare. Registra anche staticamente le classi del fornitore crittografico. Specificare i diversi fornitori JSSE esattamente come nell'esempio che segue:

```
security.provider.5=com.ibm.i5os.jsse.JSSEProvider
security.provider.6=com.ibm.jsse2.IBMJSSEProvider2
security.provider.7=com.sun.net.ssl.internal.ssl.Provider
```

ssl.KeyManagerFactory.algorithm

Specifica l'algoritmo KeyManagerFactory predefinito. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

Per il fornitore Java JSSE puro IBM, utilizzare quanto segue:

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

Per il fornitore Sun Microsystems, Inc. Java JSSE puro, utilizzare quanto segue:

```
ssl.KeyManagerFactory.algorithm=SunX509
```

Per ulteriori informazioni, consultare il Javadoc per javax.net.ssl.KeyManagerFactory.

ssl.TrustManagerFactory.algorithm

Specifica l'algoritmo TrustManagerFactory predefinito. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

Per il fornitore Java JSSE puro IBM, utilizzare quanto segue:

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

Per ulteriori informazioni, consultare Javadoc per javax.net.ssl.TrustManagerFactory.

ssl.SocketFactory.provider

Specifica la produzione socket SSL predefinita. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.SocketFactory.provider=com.ibm.i5os.jsse.JSSESocketFactory
```

Per il fornitore Java JSSE puro IBM, utilizzare quanto segue:

```
ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
```

Per ulteriori informazioni, consultare il Javadoc per javax.net.ssl.SSLSocketFactory.

ssl.ServerSocketFactory.provider

Specifica la produzione socket del server SSL predefinita. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.ServerSocketFactory.provider=com.ibm.i5os.jsse.JSSEServerSocketFactory
```

Per il fornitore JSSE nativo di Java, utilizzare:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
```

Per ulteriori informazioni, consultare Javadoc per javax.net.ssl.SSLServerSocketFactory.

Informazioni correlate

- Javadoc javax.net.ssl.KeyManagerFactory
- Javadoc javax.net.ssl.TrustManagerFactory
- Javadoc javax.net.ssl.SSLSocketFactory
- Javadoc javax.net.ssl.SSLServerSocketFactory

Proprietà di sistema Java JSSE 1.5:

Per utilizzare JSSE nelle applicazioni, è necessario specificare diverse proprietà di sistema necessarie agli oggetti SSLContext predefiniti per fornire la conferma della configurazione. Alcune proprietà si applicano a tutti i fornitori, mentre altre si applicano solo al fornitore nativo di System i5.

Quando si utilizza il fornitore System i5 JSSE nativo, se non si specificano le proprietà, os400.certificateContainer viene impostato sul valore predefinito *SYSTEM, cioè JSSE utilizzerà la voce predefinita nella memoria certificati di sistema.

Proprietà che si applicano al fornitore System i5 JSSE nativo e al fornitore Java JSSE puro IBM

Le proprietà che seguono si applicano ad entrambi i fornitori JSSE. Ciascuna descrizione comprende la proprietà predefinita, se applicabile.

javax.net.ssl.trustStore

Il nome del file che contiene l'oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi. Il valore predefinito è jssecacerts o cacerts (se jssecacerets non esiste).

javax.net.ssl.trustStoreType

Il tipo di oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi. Il valore predefinito è quello restituito dal metodo KeyStore.getDefaultType.

javax.net.ssl.trustStorePassword

La parola d'ordine per l'oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi.

javax.net.ssl.keyStore

Il nome del file che contiene l'oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi. Il valore predefinito è jssecacerts o cacerts (se jssecacerets non esiste).

javax.net.ssl.keyStoreType

Il tipo di oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi. Il valore predefinito è quello restituito dal metodo KeyStore.getDefaultType.

javax.net.ssl.keyStorePassword

La parola d'ordine per l'oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi.

Proprietà valide solo per il fornitore System i5 JSSE nativo

Le seguenti proprietà si applicano solo al fornitore System i5 JSSE nativo.

os400.secureApplication

L'identificativo dell'applicazione. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType

os400.certificateContainer

Il nome del file di chiavi che si desidera utilizzare. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

os400.certificateLabel

L'etichetta del file di chiavi che si desidera utilizzare. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

Concetti correlati

"Elenco delle proprietà di sistema Java" a pagina 16
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Informazioni correlate

Proprietà di sistema Sun Microsystems, Inc.

Utilizzo del fornitore System i5 JSSE 1.5 nativo:

Il fornitore System i5 JSSE nativo offre la serie completa di classi e interfacce JSSE, comprese le implementazioni delle classi SSLConfiguration e Keystore JSSE.

Valori protocollo per il metodo SSLContext.getInstance

La seguente tabella identifica e descrive i valori di protocollo per il metodo SSLContext.getInstance del fornitore System i5 JSSE nativo.

I protocolli SSL supportati possono essere limitati dai valori di sistema impostati sul sistema. Per ulteriori dettagli, consultare l'argomento secondario relativo ai valori di sistema di sicurezza: protocolli SSL (Secure Sockets Layer) nelle informazioni sulla gestione dei sistemi.

Valore protocollo	Protocolli SSL supportati
SSL	SSL versione 2, SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.
SSLv2	SSL versione 2
SSLv3	Protocollo SSL versione 3. Accetterà l'hello SSLv3 incapsulato in un hello formato SSLv2.
TLS	SSL versione 2, SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.
TLSv1	Protocollo TLS versione 1, definito in RFC (Request for Comments) 2246. Accetterà l'hello TLSv1 incapsulato in un hello formato SSLv2.
SSL_TLS	SSL versione 2, SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.

Implementazioni KeyStore System i5 native

Il fornitore System i5 nativo offre due implementazioni della classe KeyStore, IbmISeriesKeyStore o IBMi5OSKeyStore. Entrambe le implementazioni KeyStore forniscono un wapper al supporto DCM (Digital Certificate Manager).

IbmISeriesKeyStore

Il contenuto della memorizzazione chiave si basa su uno specifico identificativo dell'applicazione o file di chiavi, parola d'ordine ed etichetta. JSSE carica le voci memorizzazione chiave dal DCM. Per caricare le voci, JSSE utilizza l'appropriato identificativo dell'applicazione o informazioni del file di chiavi quando l'applicazione effettua il primo tentativo di accesso alle voci o alle informazioni memorizzazione chiave. Non è possibile modificare la memorizzazione chiave ed è necessario effettuare tutte le modifiche alla configurazione utilizzando il DCM (Digital Certificate Manager).

IBMi5OSKeyStore

Il contenuto di questa memorizzazione chiave è basato sul file di memorizzazione certificati i5OS e sulla parola d'ordine per accedere a tale file. Questa classe KeyStore consente la modifica della memorizzazione certificati. È possibile apportare delle modifiche senza utilizzare il Digital Certificate Manager.

L'implementazione di IBMi5OSKeyStore rispetta la specifica Sun Microsystems, Inc. per la API KeyStore Java. È possibile trovare ulteriori informazioni nella pagina relativa alle informazioni javadoc su Keystore di Sun Microsystems, Inc.

Per ulteriori informazioni su come gestire le memorizzazioni chiave tramite DCM, consultare l'argomento relativo al Digital Certificate Manager.

Informazioni correlate

Valori di sistema di sicurezza: protocolli SSL (Secure Sockets Layer)

Informazioni Javadoc sulla classe i5OSLoadStoreParameter:

com.ibm.i5os.keystore

Classe i5OSLoadStoreParameter

```
java.lang.Object
  |
  +--com.ibm.i5os.keystore.i5OSLoadStoreParameter
```

Tutte le interfacce implementate:

java.security.Keystore.LoadStoreParameter

```
public class i5OSLoadStoreParameter
extends java.lang.Object
implements java.security.Keystore.LoadStoreParameter
```

Questa classe crea un oggetto KeyStore.ProtectionParameter che può essere utilizzato per caricare/memorizzare memorizzazioni certificati i5OS. Dopo essere stata creata, questa classe fornisce informazioni sulla memorizzazione certificati cui accedere e la parola d'ordine utilizzata per proteggere tale memorizzazione.

Un utilizzo di esempio di questa classe sarebbe:

```
//inizializzare la memorizzazione chiave
    KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");

 //Caricare una memorizzazione chiave esistente
    File kdbFile = new File("/tmp/certificateStore.kdb");
    i5OSLoadStoreParameter lsp =
   new i5OSLoadStoreParameter (kdbFile, "password".toCharArray());
    ks.load(lsp);

 //Ottenere e aggiungere voci alla memorizzazione certificati
```

```
    ...
 //Salvare la memorizzazione certificati
    Ks.store(lsp);
```

Da: SDK 1.5

Riepilogo del programma di creazione

i5OSLoadStoreParameter(java.io.File ksFile, char[] password)

Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la parola d'ordine da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

i5OSLoadStoreParameter(java.io.File ksFile, java.security.KeyStore.PasswordProtection pwdProtParam)

Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la PasswordProtection da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

Tabella 14. **Riepilogo dei metodi**

java.security.KeyStore. ProtectionParameter	*"getProtectionParameter" a pagina 308*() Restituisce il KeyStore.KeyStoreParameter associato a questo LoadStoreParameter

---

Metodi ereditati dalla classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Dettagli sul programma di creazione

i5OSLoadStoreParameter

```
public i5OSLoadStoreParameter(java.io.File ksFile,
                              char[] password)
                       throws java.lang.IllegalArgumentException
```

Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la parola d'ordine da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

Parametri:

ksFile - l'oggetto File del KeyStore.

Se keystore.load() è stato utilizzato con un i5OSLoadStoreParameter(ksFile = null, password), viene creata una nuova memorizzazione chiave.

Se keystore.store() è stato utilizzato con un i5OSLoadStoreParameter(ksFile = null, password), viene generata una IllegalArgumentException.

password - la parola d'ordine per accedere alla memorizzazione certificati i5OS. Non può essere nulla o vuota.

Genera:

java.lang.IllegalArgumentException - se la parola d'ordine è nulla o vuota

---

i5OSLoadStoreParameter

```
public i5OSLoadStoreParameter(java.io.File ksFile,
                              java.security.KeyStore.PasswordProtection pwdProtParam)
                       throws java.lang.IllegalArgumentException
```

Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la PasswordProtection da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

Se `keystore.load()` è stato utilizzato con un `i5OSLoadStoreParameter(ksFile = null, password)`, viene creata una nuova memorizzazione chiave.

Se `keystore.store()` è stato utilizzato con un `i5OSLoadStoreParameter(ksFile = null, password)`, viene generata una IllegalArgumentException.

Parametri:

`ksFile` - l'oggetto File del KeyStore.

`pwdProtParam` - l'istanza PasswordProtection che verrà utilizzata per acquisire la parola d'ordine. Non può essere nulla.

Genera:

`java.lang.IllegalArgumentException` - se KeyStore.PasswordProtection è nulla o se la parola d'ordine contenuta in pwdProtParam è nulla o vuota.

Dettagli del metodo

---

getProtectionParameter

```
public java.security.KeyStore.ProtectionParameter getProtectionParameter()
```

Restituisce il KeyStore.KeyStoreParameter associato a questo LoadStoreParameter.

Specificato da:

`getProtectionParameter nell'interfaccia java.security.KeyStore.LoadStoreParameter`

Restituisce:

Un'istanza che implementa l'interfaccia KeyStore.KeyStoreParameter

Vedere anche:

`java.security.KeyStore.ProtectionParameter#getProtectionParameter()`

Informazioni Javadoc sulla classe i5OSSystemCertificateStoreFile:

com.ibm.i5os.keystore

Classe i5OSSystemCertificateStoreFile

```
java.lang.Object
   java.io.File
       com.ibm.i5os.keystore.i5OSSystemCertificateStoreFile
```

Tutte le interfacce implementate:

java.io.Serializable, java.lang.Comparable<java.io.File>

```
public class i5OSSystemCertificateStoreFile
extends java.io.File
```

Questa classe fornisce una nuova implementazione File che punta al file di memorizzazione certificati *SYSTEM. Fornisce un meccanismo per consentire all'utente di caricare la memorizzazione certificati *SYSTEM senza dovere necessariamente conoscere il percorso effettivo alla memorizzazione.

Per caricare la memorizzazione certificati *SYSTEM in una memorizzazione chiave, creare prima un i5OSSystemCertificateStoreFile.

Da questo punto, la memorizzazione chiave può essere caricata in due modi:

- Utilizzo di un i5OSLoadStoreParameter:

```
//creare un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();

  //utilizzare questo file per creare un i5OSLoadStoreParameter
     i5OSLoadStoreParameter lsp = new i5OSLoadStoreParameter(starSystemFile, pwd);

  //caricare la memorizzazione certificati in una memorizzazione chiave
     KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
     ks.load(lsp);
```

- Utilizzando un FileInputStream:

```
//creare un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();

  //creare un flusso di immissione dallo starSystemFile
     FileInputStream fis = new FileInputStream(starSystemFile);

  //caricare la memorizzazione certificati in una memorizzazione chiave
     KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
     ks.load(fis, pwd);
```

Da: SDK 1.5

Vedere anche:
Modulo serializzato

Riepilogo campi

Campi ereditati da class java.io.File
pathSeparator, pathSeparatorChar, separator, separatorChar

Riepilogo del programma di creazione

i5OSSystemCertificateStoreFile()

Crea un File() che punta al file di memorizzazione certificati *System.

Riepilogo dei metodi

Metodi ereditati dalla classe java.io.File
canRead, canWrite, compareTo, createNewFile, createTempFile, createTempFile, delete, deleteOnExit, equals, exists, getAbsoluteFile, getAbsolutePath, getCanonicalFile, getCanonicalPath, getName, getParent, getParentFile, getPath, hashCode, isAbsolute, isDirectory, isFile, isHidden, lastModified, length, list, list, listFiles, listFiles, listFiles, listRoots, mkdir, mkdirs, renameTo, setLastModified, setReadOnly, toString, toURI, toURL

Metodi ereditati dalla classe java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Dettagli sul programma di creazione

i5OSSystemCertificateStoreFile

public **i5OSSystemCertificateStoreFile**()

Crea un File() che punta al file di memorizzazione certificati *System.

Informazioni Javadoc su SSLConfiguration per la versione 1.5:

com.ibm.i5os.jsse

Classe SSLConfiguration

```
java.lang.Object
  |
  +--com.ibm.i5os.jsse.SSLConfiguration
```

Tutte le interfacce implementate:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

public final class **SSLConfiguration**
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable

Questa classe si occupa della specifica della configurazione richiesta dall'implementazione System i5 JSSE nativa.

L'implementazione System i5 JSSE nativa funziona in modo ottimale utilizzando un oggetto KeyStore di tipo "IbmISeriesKeyStore". Questo tipo di oggetto KeyStore contiene voci chiavi e voci certificati sicuri basate su un'identificazione dell'applicazione registrata con DCM (Digital Certificate Manager) o su un file di chiavi (memorizzazione certificati digitali). Un oggetto KeyStore di questo tipo può essere utilizzato per inizializzare un oggetto X509KeyManger e un oggetto X509TrustManager dal Provider "IBMi5OSJSSEProvider". Gli oggetti X509KeyManager e X509TrustManager possono essere utilizzati per inizializzare un oggetto SSLContext da "IBMi5OSJSSEProvider". L'oggetto SSLContext fornisce quindi l'accesso all'implementazione System i5 JSSE nativa in base alle informazioni relative alla configurazione specificate per l'oggetto KeyStore. Ogni volta che si esegue un caricamento per un KeyStore "IbmISeriesKeyStore", il KeyStore viene inizializzato in base alla configurazione corrente specificata dall'identificativo dell'applicazione o dal file di chiavi.

Questa classe può essere utilizzata anche per creare un oggetto KeyStore valido di qualsiasi tipo. Il KeyStore viene inizializzato in base alla configurazione corrente specificata dall'identificativo dell'applicazione o dal file di chiavi. Qualsiasi modifica effettuata alla configurazione specificata da un

identificativo di applicazione o da un file di chiavi richiede che l'oggetto KeyStore venga creato nuovamente per poter applicare la modifica. Si tenga presente che una parola d'ordine del file di chiavi deve essere specificata (per la memoria certificati *SYSTEM quando si utilizza un'ID di applicazione) per poter creare in modo corretto un KeyStore di tipo diverso da ″IbmISeriesKeyStore″. La parola d'ordine del file di chiavi deve essere specificata per poter ottenere l'accesso a qualsiasi chiave privata per qualsiasi KeyStore di tipo ″IbmISeriesKeyStore″ creato.

Da: SDK 1.5

Consultare anche:
KeyStore, X509KeyManager, X509TrustManager, SSLContext

Riepilogo del programma di creazione

SSLConfiguration() crea una nuova SSLConfiguration. Per ulteriori informazioni, consultare "Dettagli sul programma di creazione " a pagina 312.

Tabella 15. **Riepilogo dei metodi**

void	"clear" a pagina 315() Cancella tutte le informazioni all'interno dell'oggetto in modo che tutti i metodi ottenuti restituiscano null.
java.lang.Object	"clone" a pagina 316() Crea una nuova copia di questa configurazione SSL.
boolean	"equals" a pagina 316(java.lang.Objectobj) Indica se alcuni oggetti sono ″uguali a″ questo.
protected void	"finalize" a pagina 315() Richiamato dal programma di raccolta di dati su un oggetto quando la raccolta di dati determina che non esistono più riferimenti all'oggetto.
java.lang.String	"getApplicationId" a pagina 314() Restituisce l'ID dell'applicazione.
java.lang.String	"getKeyringLabel" a pagina 314() Restituisce l'etichetta del file di chiavi.
java.lang.String	"getKeyringName" a pagina 314() Restituisce il nome del file di chiavi.
char[]	"getKeyringPassword" a pagina 315() Restituisce la parola d'ordine del file di chiavi.
java.security.KeyStore	"getKeyStore" a pagina 317(char[]password) Restituisce una memorizzazione chiave di tipo ″IbmISeriesKeyStore″ utilizzando la parola d'ordine specifica.
java.security.KeyStore	" getKeyStore" a pagina 317(java.lang.Stringtype, char[]password) Restituisce una memorizzazione chiave di tipo richiesto utilizzando la parola d'ordine specifica.
int	"hashCode" a pagina 316() Restituisce un valore del codice hash per l'oggetto.
staticvoid	(java.lang.String[]args) Esegue le funzioni SSLConfiguration.
void	(java.lang.String[]args, java.io.PrintStreamout) Esegue le funzioni SSLConfiguration.
void	"setApplicationId" a pagina 315(java.lang.StringapplicationId) Imposta l'ID di applicazione.
void	"setApplicationId" a pagina 316(java.lang.StringapplicationId, char[]password) Imposta l'ID di applicazione e la parola d'ordine del file di chiavi.
void	"setKeyring" a pagina 315(java.lang.Stringname,java.lang.Stringlabel, char[]password) Imposta le informazioni del file di chiavi.

Metodi ereditati dalla classe java.lang.Object

```
getClass, notify, notifyAll, toString, wait, wait, wait
```

Dettagli sul programma di creazione

SSLConfiguration

```
public SSLConfiguration()
```

Crea una nuova SSLConfiguration. L'identificazione d'applicazione e le informazioni sul file di chiavi vengono inizializzate su valori predefiniti.

Il valore predefinito per l'identificazione d'applicazione è il valore specificato per la proprietà ″os400.secureApplication″.

I valori predefiniti per le informazioni del file di chiavi sono null se la proprietà ″os400.secureApplication″ viene specificata. Se la proprietà ″os400.secureApplication″ non viene specificata, il valore predefinito per il nome del file di chiavi è il valore specificato per la proprietà ″os400.certificateContainer″. Se la proprietà ″os400.secureApplication″ non viene specificata, l'etichetta del file di chiavi viene inizializzata sul valore della proprietà ″os400.certificateLabel″. Se non viene impostata la proprietà ″os400.secureApplication″ o ″os400.certificateContainer″, il nome del file di chiavi verrà inizializzato su ″*SYSTEM″.

Dettagli del metodo

---

main

```
public static void main(java.lang.String[]args)
```

Esegue le funzioni SSLConfiguration. Esistono quattro comandi che possono essere eseguiti: -help, -create, -display e -update. Il comando deve essere il primo parametro specificato.

Le seguenti opzioni possono essere specificate (in qualsiasi ordine):

-keystore **keystore-file-name**
: Specifica il nome del file memorizzazione chiave da creare, aggiornare o visualizzare. Questa opzione viene richiesta per tutti i comandi.

-storepass **keystore-file-password**
: Specifica la parola d'ordine associata al file di chiavi da creare, aggiornare o visualizzare. Questa opzione viene richiesta per tutti i comandi.

-storetype **keystore-type**
: Specifica il tipo di file di chiavi da creare, aggiornare o visualizzare. Questa opzione può essere specificata per qualsiasi comando. Se questa opzione non viene specificata, viene utilizzato un valore di ″IbmISeriesKeyStore″.

-appid **application-identifier**
: Specifica l'identificazione di applicazione da utilizzare per inizializzare un file di chiavi che si stava creando o aggiornando. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-keyring **keyring-file-name**
: Specifica il nome del file di chiavi da utilizzare per inizializzare un file di chiavi che si stava

creando o aggiornando. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-keyringpass **keyring-file-password**
Specifica la parola d'ordine del file di chiavi da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione può essere specificata per i comandi *-create* e *-update*, viene richiesta quando viene specificato un tipo memorizzazione chiave diverso da "IbmISeriesKeyStore". Se questa opzione non viene specificata, viene utilizzata la parola d'ordine nascosta del file di chiavi.

-keyringlabel **keyring-file-label**
Specifica l'etichetta del file di chiavi da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione può essere specificata solo quando viene specificata anche l'opzione *-keyring*. Se questa opzione non viene specificata quando viene specificata l'opzione *keyring*, viene utilizzata l'etichetta predefinita all'interno del file di chiavi.

-systemdefault
Specifica il valore predefinito del sistema da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-v Specifica che l'emissione verbose deve essere prodotta. Questa opzione può essere specificata per qualsiasi comando.

Il comando di aiuto visualizza le informazioni sull'utilizzo per specificare i parametri in questo metodo. I parametri per richiamare la funzione di aiuto vengono specificati come segue:

```
-help
```

Il comando di creazione crea un nuovo file memorizzazione chiave. Esistono tre variazioni del comando di creazione. La prima variazione per creare una memorizzazione chiave si basa su una specifica identificazione di applicazione, la seconda su un nome, etichetta, e parola d'ordine del file di chiavi e la terza sulla configurazione predefinita del sistema.

Per creare una parola d'ordine basata su un'identificazione di applicazione specifica, deve essere specificata l'opzione *-appid*. I seguenti parametri creano un file memorizzazione chiave di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base all'identificazione di applicazione "APPID":

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -appid APPID
```

Per creare una memorizzazione chiave basata su un file di chiavi specifico, deve essere specificata l'opzione *-keyring*. Devono essere specificate anche le opzioni *-keyringpass* e *keyringlabel*. I seguenti parametri creano un file di chiavi di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base al file di chiavi denominato "keyring.file", alla parola d'ordine del file di chiavi "ringpass" e all'etichetta del file di chiavi "keylabel":

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

Per creare una memorizzazione chiave basata sulla configurazione predefinita del sistema, deve essere specificata l'opzione the *-systemdefault*. I seguenti parametri creano un file memorizzazione chiave di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base alla configurazione predefinita del sistema:

```
-create -keystore keystore.file -storepass keypass -systemdefault
```

Il comando di aggiornamento, aggiorna un file memorizzazione chiave esistente di tipo "IbmISeriesKeyStore". Esistono tre variazioni del comando di aggiornamento identiche alle variazioni del comando di creazione. Le opzioni per il comando di aggiornamento sono identiche alle opzioni utilizzate per il comando di creazione. Il comando di visualizzazione, visualizza la configurazione specificata per

un file memorizzazione chiave esistente. I seguenti parametri visualizzano la configurazione specificata da un file memorizzazione chiave di tipo ″IbmISeriesKeyStore″ denominato ″keystore.file″ con una parola d'ordine di ″keypass″:

```
-display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

Parametri:

`args` - gli argomenti della riga comandi

---

run

```
public void run(java.lang.String[]args,
                java.io.PrintStreamout)
```

Esegue le funzioni SSLConfiguration. I parametri e la funzionalità di questo metodo sono identici al metodo main().

Parametri:

`args` - gli argomenti del comando

`out` - il flusso di emissione i cui risultati devono essere scritti

Vedere anche:`com.ibm.i5os.jsse.SSLConfiguration.main()`

---

getApplicationId

```
public java.lang.String getApplicationId()
```

Restituisce l'ID di applicazione.

Restituisce:

l'ID di applicazione.

---

getKeyringName

```
public java.lang.String getKeyringName()
```

Restituisce il nome del file di chiavi.

Restituisce:

il nome del file di chiavi.

---

getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

Restituisce l'etichetta del file di chiavi.

Restituisce:

l'etichetta del file di chiavi.

---

getKeyringPassword

```
public final char[] getKeyringPassword()
```

Restituisce la parola d'ordine del file di chiavi.

Restituisce:
la parola d'ordine del file di chiavi.

---

finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Richiamato dal programma di raccolta di dati su un oggetto quando la raccolta di dati determina che non esistono più riferimenti all'oggetto.

Sovrascrive:
`finalize` nella classe `java.lang.Object`

Emette:
`java.lang.Throwable` - l'eccezione provocata da questo metodo.

---

clear

```
public void clear()
```

Cancella tutte le informazioni all'interno dell'oggetto in modo che tutti i metodi ottenuti restituiscano null.

---

setKeyring

```
public void setKeyring(java.lang.Stringname,
                       java.lang.Stringlabel,
                       char[]password)
```

Imposta le informazioni sul file di chiavi.

Parametri:
`name` - il nome del file di chiavi

`label` - l'etichetta del file di chiavi o null se deve essere utilizzata la voce del file di chiavi predefinito.

`password` - la parola d'ordine del file di chiavi o null se deve essere utilizzata la parola d'ordine nascosta.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId)
```

Imposta l'ID di applicazione.

Parametri:
`applicationId` - l'ID di applicazione.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId,
                             char[]password)
```

imposta l'ID di applicazione e la parola d'ordine del file di chiavi. La specifica della parola d'ordine del file di chiavi consente a qualsiasi memorizzazione chiave creata di consentire l'accesso alla chiave privata.

Parametri:

`applicationId` - l'ID di applicazione.

`password` - la parola d'ordine del file di chiavi.

---

equals

```
public boolean equals(java.lang.Objectobj)
```

Indica se altri oggetti sono ″uguali a″ questo.

Sovrascrive:

`equals` nella classe `java.lang.Object`

Parametri:

`obj` - oggetto da confrontare

Restituisce:

un indicatore che conferma se gli oggetti specificano le stesse informazioni di configurazione

---

hashCode

```
public int hashCode()
```

Restituisce un valore del codice hash per l'oggetto.

Sovrascrive:

`hashCode` nella classe `java.lang.Object`

Restituisce:

un valore del codice hash per questo oggetto.

---

clone

```
public java.lang.Object clone()
```

Crea una nuova copia di questa configurazione SSL. Le modifiche successive effettuate ai componenti di questa configurazione SSL, non interessano la nuova copia e viceversa.

Sovrascrive:

`clone` nella classe `java.lang.Object`

Restituisce:

una copia di questa configurazione SSL

---

getKeyStore

```
public java.security.KeyStore getKeyStore(char[]password)
                                  throws java.security.KeyStoreException
```

Restituisce una memorizzazione chiave di tipo ″IbmISeriesKeyStore″ utilizzando la parola d'ordine specifica. La memorizzazione chiave viene inizializzata in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto.

Parametri:

`password` - utilizzata per inizializzare la memorizzazione chiave

Restituisce:

KeyStore, inizializzato in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto

Emette:

`java.security.KeyStoreException` - se non è stato possibile creare la memorizzazione chiave

---

getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                       char[]password)
                                  throws java.security.KeyStoreException
```

Restituisce una memorizzazione chiave del tipo richiesto utilizzando la parola d'ordine specifica. La memorizzazione chiave viene inizializzata in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto.

Parametri:

`type` - tipo di memorizzazione chiave da restituire

`password` - utilizzata per inizializzare la memorizzazione chiave

Restituisce:

KeyStore, inizializzato in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto

Emette:

`java.security.KeyStoreException` - se non è stato possibile creare la memorizzazione chiave

Esempi: IBM Java Secure Sockets Extension 1.5:

Gli esempi JSSE mostrano il modo in cui un client e un server possono utilizzare il fornitore System i5 JSSE nativo per creare un contesto che abilita le comunicazioni sicure.

Nota: entrambi gli esempi utilizzano il fornitore JSSE nativo di System i5, indipendentemente dalle proprietà specificate dal file java.security.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

Esempio: client SSL che utilizza un oggetto SSLContext per la versione 1.5:

Questo programma client di esempio utilizza un oggetto SSLContext, da esso inizializzato per utilizzare l'ID applicazione ″MY_CLIENT_APP″. Questo programma utilizzerà l'implementazione System i5 nativa indipendentemente da quanto specificato nel file java.security.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////////
//
// Questo programma client di esempio utilizza un oggetto SSLContext, da esso inizializzato
// per utilizzare l'ID applicazione "MY_CLIENT_APP".
//
// L'esempio utilizza il fornitore System i5 JSSE nativo, indipendentemente dalle
// proprietà specificate dal file java.security.
//
// Sintassi del comando:
//     java -Djava.version=1.5 SslClient
//
// Notare che "-Djava.version=1.5" non è necessario se è stato configurato
// come valore predefinito l'utilizzo di JDK versione 1.5.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
import com.ibm.i5os.jsse.SSLConfiguration;
/**
 * Programma client SSL.
 */
public class SslClient {

   /**
    * Metodo principale SslClient.
    *
    * @param args gli argomenti della riga comandi (non utilizzato)
    */
   public static void main(String args[]) {
       /*
        * Impostare in modo da catturare gli errori inviati.
        */
       try {
           /*
            * Inizializzare un oggetto SSLConfiguration per specificare un ID
            * ID.  "MY_CLIENT_APP" deve essere registrato e configurato
            * correttamente con il DCM (Digital Certificate Manager).
            */
           SSLConfiguration config = new SSLConfiguration();
           config.setApplicationId("MY_CLIENT_APP");
           /*
            * Richiamare un oggetto KeyStore dall'oggetto SSLConfiguration.
            */
           char[] password = "password".toCharArray();
           KeyStore ks = config.getKeyStore(password);
           /*
            * Assegnare ed inizializzare un KeyManagerFactory.
            */
           KeyManagerFactory kmf =
                            KeyManagerFactory.getInstance("IbmISeriesX509");
           kmf.init(ks, password);
           /*
            * Assegnare ed inizializzare un TrustManagerFactory.
            */
           TrustManagerFactory tmf =
                           TrustManagerFactory.getInstance("IbmISeriesX509");
           tmf.init(ks);
           /*
            * Assegnare ed inizializzare un SSLContext.
            */
           SSLContext c =
                      SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
           c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
           /*
            * Richiamare un SSLSocketFactory dal SSLContext.
```

```
            */
           SSLSocketFactory sf = c.getSocketFactory();
           /*
            * Creare un SSLSocket.
            *
            * Modificare l'indirizzo IP codificato con l'indirizzo IP o il nome host
            * del server.
            */
           SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
           /*
            * Inviare un messaggio al server utilizzando la sessione protetta.
            */
           String sent = "Test of java SSL write";
           OutputStream os = s.getOutputStream();
           os.write(sent.getBytes());
           /*
            * Scrivere i risultati sullo schermo.
            */
           System.out.println("Wrote " + sent.length() + " bytes...");
           System.out.println(sent);
           /*
            * Ricevere un messaggio dal server utilizzando la sessione protetta.
            */
           InputStream is = s.getInputStream();
           byte[] buffer = new byte[1024];
           int bytesRead = is.read(buffer);
           if (bytesRead == -1)
               throw new IOException("Unexpected End-of-file Received");
           String received = new String(buffer, 0, bytesRead);
           /*
            * Scrivere i risultati sullo schermo.
            */
           System.out.println("Read " + received.length() + " bytes...");
           System.out.println(received);
       } catch (Exception e) {
           System.out.println("Unexpected exception caught: " +
                              e.getMessage());
           e.printStackTrace();
       }
   }

}
```

Esempio: server SSL che utilizza un oggetto SSLContext per la versione 1.5:

Il seguente programma server utilizza un oggetto SSLContext da esso inizializzato, con un file memorizzazione chiave precedentemente creato.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////////
//
// Il seguente programma server utilizza un oggetto SSLContext da esso
// inizializzato, con un file memorizzazione chiave precedentemente creato.
//
// Il nome e la parola d'ordine memorizzazione chiave del file memorizzazione chiave sono:
//   Nome file: /home/keystore.file
//   Parola d'ordine:  password
//
// È necessario che il programma di esempio disponga del file memorizzazione chiave per poter creare un
// oggetto IbmISeriesKeyStore. L'oggetto KeyStore deve specificare MY_SERVER_APP come
// identificativo dell'applicazione.
//
// Per creare il file memorizzazione chiave, è possibile utilizzare il seguente comando Qshell:
//
```

```
//   java com.ibm.i5os.SSLConfiguration  -create  -keystore /home/keystore.file
//     -storepass password  -appid MY_SERVER_APP
//
// Sintassi del comando:
//     java -Djava.version=1.5 JavaSslServer
//
// Notare che "-Djava.version=1.5" non è necessario se è stato configurato
// come valore predefinito l'utilizzo di JDK versione 1.5.
//
// È anche possibile creare il file memorizzazione chiave immettendo questo comando su una richiesta comandi i5/OS:
//
//  RUNJVA CLASS(com.ibm.i5os.SSLConfiguration) PARM('-create' '-keystore'
//      '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
/**
* Programma server SSL Java che utilizza l'ID applicazione.
*/
public class JavaSslServer {

   /**
    * Metodo principale JavaSslServer.
    *
    * @param args gli argomenti della riga comandi (non utilizzato)
    */
   public static void main(String args[]) {
       /*
        * Impostare in modo da catturare gli errori inviati.
        */
       try {
           /*
            * Assegnare ed inizializzare un oggetto KeyStore.
            */
           char[] password = "password".toCharArray();
           KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
           FileInputStream fis = new FileInputStream("/home/keystore.file");
           ks.load(fis, password);
           /*
            * Assegnare ed inizializzare un KeyManagerFactory.
            */
           KeyManagerFactory kmf =
                             KeyManagerFactory.getInstance("IbmISeriesX509");
           kmf.init(ks, password);
           /*
            * Assegnare ed inizializzare un TrustManagerFactory.
            */
           TrustManagerFactory tmf =
                           TrustManagerFactory.getInstance("IbmISeriesX509");
           tmf.init(ks);
           /*
            * Assegnare ed inizializzare un SSLContext.
            */
           SSLContext c =
                      SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
           c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
           /*
            * Richiamare un SSLServerSocketFactory dal SSLContext.
            */
           SSLServerSocketFactory sf = c.getServerSocketFactory();
           /*
            * Creare un SSLServerSocket.
            */
           SSLServerSocket ss =
```

```
                (SSLServerSocket) sf.createServerSocket(13333);
            /*
             * Eseguire un accept() per creare un SSLSocket.
             */
            SSLSocket s = (SSLSocket) ss.accept();
            /*
             * Ricevere un messaggio dal client utilizzando la sessione protetta.
             */
            InputStream is = s.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead = is.read(buffer);
            if (bytesRead == -1)
                throw new IOException("Unexpected End-of-file Received");
            String received = new String(buffer, 0, bytesRead);
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Read " + received.length() + " bytes...");
            System.out.println(received);
            /*
             * Rimandare il messaggio al client utilizzando la sessione protetta.
             */
            OutputStream os = s.getOutputStream();
            os.write(received.getBytes());
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Wrote " + received.length() + " bytes...");
            System.out.println(received);
        } catch (Exception e) {
            System.out.println("Unexpected exception caught: " +
                               e.getMessage());
            e.printStackTrace();
        }
    }

}
```

Utilizzo di Java Secure Socket Extension 6

Queste informazioni sono valide solo quando si utilizza JSSE sui sistemi che eseguono Java 2 Platform, Standard Edition (J2SE), versione 6 e release successivi. JSSE agisce come una framework che estrae i meccanismi sottostanti sia di SSL che di TLS. Estraendo la complessità e le peculiarità dei protocolli sottostanti, JSSE permette ai programmatori di utilizzare comunicazioni sicure e codificate, riducendo allo stesso tempo la vulnerabilità della sicurezza. JSSE (Java Secure Socket Extension) utilizza entrambi i protocolli SSL (Secure Sockets Layer) e TSL (Transport Layer Security) per fornire comunicazioni sicure e codificate tra i client e i server.

L'implementazione IBM di JSSE viene denominata IBM JSSE. IBM JSSE include un fornitore i5/OS JSSE nativo ed un fornitore IBM Java JSSE puro. Per quanto riguarda inoltre JSSE di Sun Microsystems, Inc., era in precedenza fornito con System i e continuerà ad essere fornito.

Configurazione del server per supportare JSSE 6:

Configurare System i5 per utilizzare implementazioni JSSE differenti. Questo argomento include i requisiti software, le istruzioni su come modificare i fornitori JSSE e le proprietà di sistema e della sicurezza necessarie. La configurazione predefinita utilizza il fornitore JSSE Java puro IBM noto come IBMJSSE2.

Quando si utilizza J2SE (Java 2 Platform, Standard Edition) versione 6 su System i5, JSSE è già configurato. La configurazione predefinita utilizza il fornitore JSSE System i5 nativo.

Modifica dei fornitori JSSE

È possibile configurare JSSE per utilizzare il fornitore JSSE System i5 nativo oppure il JSSE di Sun Microsystems, Inc. invece del fornitore JSSE Java puro IBM. Modificando specifiche proprietà della sicurezza JSSE e proprietà di sistema Java, è possibile passare da un fornitore all'altro.

Gestori della sicurezza

Se l'applicazione JSSE è in esecuzione con un gestore della sicurezza Java abilitato, potrebbe essere necessario impostare le autorizzazioni alla rete disponibile. Per ulteriori informazioni, consultare SSL in Permissions in the Java 2 SDK.

Fornitori JSSE 6:

IBM JSSE include un fornitore System i5 JSSE nativo ed un fornitore IBM Java JSSE puro. Sun Microsystems, Inc. JSSE è fornito anche con System i5. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

Tutti i fornitori rispettano la specifica dell'interfaccia JSSE. Sono in grado di comunicare tra di loro e con qualsiasi altra implementazione SSL o TLS, perfino con implementazioni non Java.

Fornitore JSSE Java puro Sun Microsystems, Inc.

Questa è l'implementazione Sun Java di JSSE. Era inizialmente fornito con System i e continuerà ad essere fornito. Per ulteriori informazioni sul JSEE Sun Microsystems, Inc., consultare il manuale Java Secure Socket Extension (JSSE) Reference Guide di Sun Microsystems, Inc.

Fornitore JSEE Java puro IBM

Il fornitore JSSE Java puro IBM offre le seguenti funzioni:

- Gestisce qualsiasi tipo di oggetto KeyStore per controllare e configurare i certificati digitali (ad esempio, JKS, PKCS12 e così via).
- Permette di utilizzare contemporaneamente qualsiasi combinazione di componenti JSSE provenienti da più implementazioni.

IBMJSSEProvider2 è il nome del fornitore per l'implementazione Java pura. È necessario inoltrare questo nome fornitore, rispettando i caratteri maiuscoli e minuscoli, al metodo java.security.Security.getProvider() o ai vari metodi getInstance() per diverse classi JSSE.

Fornitore JSSE System i5 nativo

Il fornitore System i5 JSSE nativo offre le seguenti funzioni:

- Utilizza il supporto SSL System i5 nativo.
- Permette l'utilizzo di DCM (Digital Certificate Manager) per configurare e controllare i certificati digitali. Ciò viene fornito tramite un tipo System i5 univoco di KeyStore (IbmISeriesKeyStore).
- Permette di utilizzare contemporaneamente qualsiasi combinazione di componenti JSSE provenienti da più implementazioni.

IBMi5OSJSSEProvider è il nome per l'implementazione nativa di System i5. È necessario inoltrare questo nome fornitore, rispettando i caratteri maiuscoli e minuscoli, al metodo java.security.Security.getProvider() o ai vari metodi getInstance() per diverse classi JSSE.

Modifica del fornitore JSSE predefinito

È possibile cambiare il fornitore JSSE predefinito apportando le modifiche appropriate alle proprietà della sicurezza.

Una volta modificato il fornitore JSSE, assicurarsi che le proprietà di sistema specifichino la corretta configurazione per le informazioni sul certificato digitale (memorizzazione chiave) richiesta dal nuovo fornitore.

Per ulteriori informazioni, consultare l'argomento relativo alle proprietà di sicurezza JSSE 6.


JSSE Reference Guide di Sun Microsystems, Inc.

"Proprietà di sicurezza JSSE 6"
Una JVM (Java virtual machine) utilizza diverse importanti proprietà della sicurezza che è possibile impostare modificando il file delle proprietà della sicurezza principale Java.

Proprietà di sicurezza JSSE 6:

Una JVM (Java virtual machine) utilizza diverse importanti proprietà della sicurezza che è possibile impostare modificando il file delle proprietà della sicurezza principale Java.

Questo file, denominato java.security, si trova di solito nell'indirizzario /QIBM/ProdData/Java400/jdk6/lib/security sul server.

L'elenco che segue descrive diverse importanti proprietà della sicurezza per l'utilizzo di JSSE. Utilizzare le descrizioni come guida alla modifica del file java.security.

security.provider.<numero intero>

Il fornitore JSSE che si desidera utilizzare. Registra anche staticamente le classi del fornitore crittografico. Specificare i diversi fornitori JSSE esattamente come nell'esempio che segue:

```
security.provider.5=com.ibm.i5os.jsse.JSSEProvider
security.provider.6=com.ibm.jsse2.IBMJSSEProvider2
security.provider.7=com.sun.net.ssl.internal.ssl.Provider
```

ssl.KeyManagerFactory.algorithm

Specifica l'algoritmo KeyManagerFactory predefinito. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

Per il fornitore Java JSSE puro IBM, utilizzare quanto segue:

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

Per il fornitore Sun Microsystems, Inc. Java JSSE puro, utilizzare quanto segue:

```
ssl.KeyManagerFactory.algorithm=SunX509
```

Per ulteriori informazioni, consultare il Javadoc per javax.net.ssl.KeyManagerFactory.

ssl.TrustManagerFactory.algorithm

Specifica l'algoritmo TrustManagerFactory predefinito. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

Per il fornitore Java JSSE puro IBM, utilizzare quanto segue:

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

Per ulteriori informazioni, consultare Javadoc per javax.net.ssl.TrustManagerFactory.

ssl.SocketFactory.provider

Specifica la produzione socket SSL predefinita. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.SocketFactory.provider=com.ibm.i5os.jsse.JSSESocketFactory
```

Per il fornitore Java JSSE puro IBM, utilizzare quanto segue:

```
ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
```

Per ulteriori informazioni, consultare il Javadoc per javax.net.ssl.SSLSocketFactory.

ssl.ServerSocketFactory.provider

Specifica la produzione socket del server SSL predefinita. Per il fornitore System i5 JSSE nativo, utilizzare quanto segue:

```
ssl.ServerSocketFactory.provider=com.ibm.i5os.jsse.JSSEServerSocketFactory
```

Per il fornitore JSSE nativo di Java, utilizzare:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
```

Per ulteriori informazioni, consultare Javadoc per javax.net.ssl.SSLServerSocketFactory.

Informazioni correlate

- Javadoc javax.net.ssl.KeyManagerFactory
- Javadoc javax.net.ssl.TrustManagerFactory
- Javadoc javax.net.ssl.SSLSocketFactory
- Javadoc javax.net.ssl.SSLServerSocketFactory

Proprietà di sistema Java JSSE 6:

Per utilizzare JSSE nelle applicazioni, è necessario specificare diverse proprietà di sistema necessarie agli oggetti SSLContext predefiniti per fornire la conferma della configurazione. Alcune proprietà si applicano a tutti i fornitori, mentre altre si applicano solo al fornitore System i5 nativo.

Quando si utilizza il fornitore System i5 JSSE nativo, se non si specificano le proprietà, os400.certificateContainer viene impostato sul valore predefinito *SYSTEM, cioè JSSE utilizzerà la voce predefinita nella memoria certificati di sistema.

Proprietà che si applicano al fornitore System i5 JSSE nativo e al fornitore Java JSSE puro IBM

Le proprietà che seguono si applicano ad entrambi i fornitori JSSE. Ciascuna descrizione comprende la proprietà predefinita, se applicabile.

javax.net.ssl.trustStore

Il nome del file che contiene l'oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi. Il valore predefinito è jssecacerts o cacerts (se jssecacerets non esiste).

javax.net.ssl.trustStoreType

Il tipo di oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi. Il valore predefinito è quello restituito dal metodo KeyStore.getDefaultType.

javax.net.ssl.trustStorePassword

La parola d'ordine per l'oggetto KeyStore che si desidera che il TrustManager predefinito utilizzi.

javax.net.ssl.keyStore

Il nome del file che contiene l'oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi. Il valore predefinito è jssecacerts o cacerts (se jssecacerets non esiste).

javax.net.ssl.keyStoreType

Il tipo di oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi. Il valore predefinito è quello restituito dal metodo KeyStore.getDefaultType.

javax.net.ssl.keyStorePassword

La parola d'ordine per l'oggetto KeyStore che si desidera che il KeyManager predefinito utilizzi.

Proprietà valide solo per il fornitore System i5 JSSE nativo

Le seguenti proprietà si applicano solo al fornitore System i5 JSSE nativo.

os400.secureApplication

L'identificativo dell'applicazione. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType

os400.certificateContainer

Il nome del file di chiavi che si desidera utilizzare. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

os400.certificateLabel

L'etichetta del file di chiavi che si desidera utilizzare. JSSE utilizza questa proprietà solo quando non vengono specificate le proprietà che seguono:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

Concetti correlati

"Elenco delle proprietà di sistema Java" a pagina 16
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Informazioni correlate

Proprietà di sistema Sun Microsystems, Inc.

Utilizzo del fornitore System i5 JSSE 6 nativo:

Il fornitore System i5 JSSE nativo offre la serie completa di classi e interfacce JSSE, comprese le implementazioni delle classi SSLConfiguration e Keystore JSSE.

Valori protocollo per il metodo SSLContext.getInstance

La seguente tabella identifica e descrive i valori di protocollo per il metodo SSLContext.getInstance del fornitore System i5 JSSE nativo.

I protocolli SSL supportati possono essere limitati dai valori di sistema impostati sul sistema. Per ulteriori dettagli, consultare l'argomento secondario relativo ai valori di sistema di sicurezza: protocolli SSL (Secure Sockets Layer) nelle informazioni sulla gestione dei sistemi.

Valore protocollo	Protocolli SSL supportati
SSL	SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.
SSLv2	SSL versione 2
SSLv3	Protocollo SSL versione 3. Accetterà l'hello SSLv3 incapsulato in un hello formato SSLv2.
TLS	Protocollo TLS versione 1, definito in RFC (Request for Comments) 2246. Accetterà l'hello TLSv1 incapsulato in un hello formato SSLv2.
TLSv1	Protocollo TLS versione 1, definito in RFC (Request for Comments) 2246. Accetterà l'hello TLSv1 incapsulato in un hello formato SSLv2.
SSL_TLS	SSL versione 3 e TLS versione 1. Accetterà l'hello SSLv3 o TLSv1 incapsulato in un hello formato SSLv2.

Implementazioni KeyStore System i5 native

Il fornitore System i5 nativo offre due implementazioni della classe KeyStore, IbmISeriesKeyStore o IBMi5OSKeyStore. Entrambe le implementazioni KeyStore forniscono un wapper al supporto DCM (Digital Certificate Manager).

IbmISeriesKeyStore

Il contenuto della memorizzazione chiave si basa su uno specifico identificativo dell'applicazione o file di chiavi, parola d'ordine ed etichetta. JSSE carica le voci memorizzazione chiave dal DCM. Per caricare le voci, JSSE utilizza l'appropriato identificativo dell'applicazione o informazioni del file di chiavi quando l'applicazione effettua il primo tentativo di accesso alle voci o alle informazioni memorizzazione chiave. Non è possibile modificare la memorizzazione chiave ed è necessario effettuare tutte le modifiche alla configurazione utilizzando il DCM (Digital Certificate Manager).

IBMi5OSKeyStore

Il contenuto di questa memorizzazione chiave è basato sul file di memorizzazione certificati i5OS e sulla parola d'ordine per accedere a tale file. Questa classe KeyStore consente la modifica della memorizzazione certificati. È possibile apportare delle modifiche senza utilizzare il Digital Certificate Manager.

L'implementazione di IBMi5OSKeyStore rispetta la specifica Sun Microsystems, Inc. per la API KeyStore Java. È possibile trovare ulteriori informazioni nella pagina relativa alle informazioni javadoc su Keystore di Sun Microsystems, Inc.

Per ulteriori informazioni su come gestire le memorizzazioni chiave tramite DCM, consultare l'argomento relativo al Digital Certificate Manager.

Informazioni correlate

Valori di sistema di sicurezza: protocolli SSL (Secure Sockets Layer)

Informazioni Javadoc sulla classe i5OSLoadStoreParameter:

com.ibm.i5os.keystore

Classe i5OSLoadStoreParameter

```
java.lang.Object
  |
  +--com.ibm.i5os.keystore.i5OSLoadStoreParameter
```

Tutte le interfacce implementate:

java.security.Keystore.LoadStoreParameter

```
public class i5OSLoadStoreParameter
extends java.lang.Object
implements java.security.Keystore.LoadStoreParameter
```

Questa classe crea un oggetto KeyStore.ProtectionParameter che può essere utilizzato per caricare/memorizzare memorizzazioni certificati i5OS. Dopo essere stata creata, questa classe fornisce informazioni sulla memorizzazione certificati cui accedere e la parola d'ordine utilizzata per proteggere tale memorizzazione.

Un utilizzo di esempio di questa classe sarebbe:

```
//inizializzare la memorizzazione chiave
    KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");

 //Caricare una memorizzazione chiave esistente
    File kdbFile = new File("/tmp/certificateStore.kdb");
    i5OSLoadStoreParameter lsp =
   new i5OSLoadStoreParameter (kdbFile, "password".toCharArray());
    ks.load(lsp);

 //Ottenere e aggiungere voci alla memorizzazione certificati
```

```
    ...

 //Salvare la memorizzazione certificati
    Ks.store(lsp);
```

Da: SDK 1.5

Riepilogo del programma di creazione

i5OSLoadStoreParameter(java.io.File ksFile, char[] password)

Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la parola d'ordine da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

i5OSLoadStoreParameter(java.io.File ksFile, java.security.KeyStore.PasswordProtection pwdProtParam)

Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la PasswordProtection da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

Tabella 16. **Riepilogo dei metodi**

java.security.KeyStore.ProtectionParameter	*"getProtectionParameter" a pagina 329*() Restituisce il KeyStore.KeyStoreParameter associato a questo LoadStoreParameter

---

Metodi ereditati dalla classe java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Dettagli sul programma di creazione

i5OSLoadStoreParameter

```
public i5OSLoadStoreParameter(java.io.File ksFile,
                              char[] password)
                       throws java.lang.IllegalArgumentException
```

Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la parola d'ordine da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

Parametri:

ksFile - l'oggetto File del KeyStore.

Se keystore.load() è stato utilizzato con un i5OSLoadStoreParameter(ksFile = null, password), viene creata una nuova memorizzazione chiave.

Se keystore.store() è stato utilizzato con un i5OSLoadStoreParameter(ksFile = null, password), viene generata una IllegalArgumentException.

password - la parola d'ordine per accedere alla memorizzazione certificati i5OS. Non può essere nulla o vuota.

Genera:

java.lang.IllegalArgumentException - se la parola d'ordine è nulla o vuota

| ---

| **i5OSLoadStoreParameter**

```
| public i5OSLoadStoreParameter(java.io.File ksFile,
|                               java.security.KeyStore.PasswordProtection pwdProtParam)
|                            throws java.lang.IllegalArgumentException
```

| Crea un'istanza ProtectionParameter dal file memorizzazione chiave e la PasswordProtection da utilizzare per caricare/memorizzare una memorizzazione certificati i5OS.

| Se `keystore.load()` è stato utilizzato con un `i5OSLoadStoreParameter(ksFile = null, password)`, viene creata una nuova memorizzazione chiave.

| Se `keystore.store()` è stato utilizzato con un `i5OSLoadStoreParameter(ksFile = null, password)`, viene generata una IllegalArgumentException.

| **Parametri**:

| `ksFile` - l'oggetto File del KeyStore.

| `pwdProtParam` - l'istanza PasswordProtection che verrà utilizzata per acquisire la parola d'ordine. Non può essere nulla.

| **Genera**:

| `java.lang.IllegalArgumentException` - se KeyStore.PasswordProtection è nulla o se la parola d'ordine contenuta in pwdProtParam è nulla o vuota.

| ---

| **Dettagli del metodo**

| ---

| **getProtectionParameter**

```
| public java.security.KeyStore.ProtectionParameter getProtectionParameter()
```

| Restituisce il KeyStore.KeyStoreParameter associato a questo LoadStoreParameter.

| **Specificato da**:

```
| getProtectionParameter nell'interfaccia java.security.KeyStore.LoadStoreParameter
```

| **Restituisce**:

| Un'istanza che implementa l'interfaccia KeyStore.KeyStoreParameter

| **Vedere anche**:

```
| java.security.KeyStore.ProtectionParameter#getProtectionParameter()
```

| *Informazioni Javadoc sulla classe i5OSSystemCertificateStoreFile:*

| com.ibm.i5os.keystore

| Classe i5OSSystemCertificateStoreFile

```
| java.lang.Object
|    java.io.File
|        com.ibm.i5os.keystore.i5OSSystemCertificateStoreFile
```

| **Tutte le interfacce implementate:**

java.io.Serializable, java.lang.Comparable<java.io.File>

```
public class i5OSSystemCertificateStoreFile
extends java.io.File
```

Questa classe fornisce una nuova implementazione File che punta al file di memorizzazione certificati *SYSTEM. Fornisce un meccanismo per consentire all'utente di caricare la memorizzazione certificati *SYSTEM senza dovere necessariamente conoscere il percorso effettivo alla memorizzazione.

Per caricare la memorizzazione certificati *SYSTEM in una memorizzazione chiave, creare prima un i5OSSystemCertificateStoreFile.

Da questo punto, la memorizzazione chiave può essere caricata in due modi:

- Utilizzo di un i5OSLoadStoreParameter:

```
//creare un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();

  //utilizzare questo file per creare un i5OSLoadStoreParameter
     i5OSLoadStoreParameter lsp = new i5OSLoadStoreParameter(starSystemFile, pwd);

  //caricare la memorizzazione certificati in una memorizzazione chiave
     KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
     ks.load(lsp);
```

- Utilizzando un FileInputStream:

```
//creare un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();

  //creare un flusso di immissione dallo starSystemFile
     FileInputStream fis = new FileInputStream(starSystemFile);

  //caricare la memorizzazione certificati in una memorizzazione chiave
     KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
     ks.load(fis, pwd);
```

Da: SDK 1.5

Vedere anche:
Modulo serializzato

Riepilogo campi

Campi ereditati da class java.io.File
pathSeparator, pathSeparatorChar, separator, separatorChar

Riepilogo del programma di creazione

i5OSSystemCertificateStoreFile()

Crea un File() che punta al file di memorizzazione certificati *System.

Riepilogo dei metodi

Metodi ereditati dalla classe java.io.File
canRead, canWrite, compareTo, createNewFile, createTempFile, createTempFile, delete, deleteOnExit, equals, exists, getAbsoluteFile, getAbsolutePath, getCanonicalFile, getCanonicalPath, getName, getParent, getParentFile, getPath, hashCode, isAbsolute, isDirectory, isFile, isHidden, lastModified, length, list, list, listFiles, listFiles, listFiles, listRoots, mkdir, mkdirs, renameTo, setLastModified, setReadOnly, toString, toURI, toURL

Metodi ereditati dalla classe java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Dettagli sul programma di creazione

i5OSSystemCertificateStoreFile

```
public i5OSSystemCertificateStoreFile()
```

Crea un File() che punta al file di memorizzazione certificati *System.

Informazioni Javadoc su SSLConfiguration per la versione 6:

com.ibm.i5os.jsse

Classe SSLConfiguration

```
java.lang.Object
  |
  +--com.ibm.i5os.jsse.SSLConfiguration
```

Tutte le interfacce implementate:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

public final class **SSLConfiguration**
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable

Questa classe si occupa della specifica della configurazione richiesta dall'implementazione System i5 JSSE nativa.

L'implementazione System i5 JSSE nativa funziona in modo ottimale utilizzando un oggetto `KeyStore` di tipo "IbmISeriesKeyStore". Questo tipo di oggetto `KeyStore` contiene voci chiavi e voci certificati sicuri basate su un'identificazione dell'applicazione registrata con DCM (Digital Certificate Manager) o su un file di chiavi (memorizzazione certificati digitali). Un oggetto `KeyStore` di questo tipo può essere utilizzato per inizializzare un oggetto `X509KeyManger` e un oggetto `X509TrustManager` dal `Provider` "IBMi5OSJSSEProvider". Gli oggetti `X509KeyManager` e `X509TrustManager` possono essere utilizzati per inizializzare un oggetto `SSLContext` da "IBMi5OSJSSEProvider". L'oggetto `SSLContext` fornisce quindi l'accesso all'implementazione System i5 JSSE nativa in base alle informazioni relative alla configurazione specificate per l'oggetto `KeyStore`. Ogni volta che si esegue un caricamento per un `KeyStore` "IbmISeriesKeyStore", il `KeyStore` viene inizializzato in base alla configurazione corrente specificata dall'identificativo dell'applicazione o dal file di chiavi.

Questa classe può essere utilizzata anche per creare un oggetto `KeyStore` valido di qualsiasi tipo. Il `KeyStore` viene inizializzato in base alla configurazione corrente specificata dall'identificativo dell'applicazione o dal file di chiavi. Qualsiasi modifica effettuata alla configurazione specificata da un

identificativo di applicazione o da un file di chiavi richiede che l'oggetto `KeyStore` venga creato nuovamente per poter applicare la modifica. Si tenga presente che una parola d'ordine del file di chiavi deve essere specificata (per la memoria certificati *SYSTEM quando si utilizza un'ID di applicazione) per poter creare in modo corretto un `KeyStore` di tipo diverso da ″IbmISeriesKeyStore″. La parola d'ordine del file di chiavi deve essere specificata per poter ottenere l'accesso a qualsiasi chiave privata per qualsiasi `KeyStore` di tipo ″IbmISeriesKeyStore″ creato.

Da: SDK 1.5

Consultare anche:
KeyStore, X509KeyManager, X509TrustManager, SSLContext

Riepilogo del programma di creazione

`SSLConfiguration()` crea una nuova SSLConfiguration. Per ulteriori informazioni, consultare "Dettagli sul programma di creazione " a pagina 333.

Tabella 17. **Riepilogo dei metodi**

`void`	"clear" a pagina 336() Cancella tutte le informazioni all'interno dell'oggetto in modo che tutti i metodi ottenuti restituiscano null.
`java.lang.Object`	"clone" a pagina 337() Crea una nuova copia di questa configurazione SSL.
`boolean`	"equals" a pagina 337(java.lang.Objectobj) Indica se alcuni oggetti sono ″uguali a″ questo.
`protected void`	"finalize" a pagina 336() Richiamato dal programma di raccolta di dati su un oggetto quando la raccolta di dati determina che non esistono più riferimenti all'oggetto.
`java.lang.String`	"getApplicationId" a pagina 335() Restituisce l'ID dell'applicazione.
`java.lang.String`	"getKeyringLabel" a pagina 335() Restituisce l'etichetta del file di chiavi.
`java.lang.String`	"getKeyringName" a pagina 335() Restituisce il nome del file di chiavi.
`char[]`	"getKeyringPassword" a pagina 336() Restituisce la parola d'ordine del file di chiavi.
`java.security.KeyStore`	"getKeyStore" a pagina 338(char[]password) Restituisce una memorizzazione chiave di tipo ″IbmISeriesKeyStore″ utilizzando la parola d'ordine specifica.
`java.security.KeyStore`	" getKeyStore" a pagina 338(java.lang.Stringtype, char[]password) Restituisce una memorizzazione chiave di tipo richiesto utilizzando la parola d'ordine specifica.
`int`	"hashCode" a pagina 337() Restituisce un valore del codice hash per l'oggetto.
`staticvoid`	(java.lang.String[]args) Esegue le funzioni SSLConfiguration.
`void`	(java.lang.String[]args, java.io.PrintStreamout) Esegue le funzioni SSLConfiguration.
`void`	"setApplicationId" a pagina 336(java.lang.StringapplicationId) Imposta l'ID di applicazione.
`void`	"setApplicationId" a pagina 337(java.lang.StringapplicationId, char[]password) Imposta l'ID di applicazione e la parola d'ordine del file di chiavi.
`void`	"setKeyring" a pagina 336(java.lang.Stringname,java.lang.Stringlabel, char[]password) Imposta le informazioni del file di chiavi.

Metodi ereditati dalla classe java.lang.Object

| getClass, notify, notifyAll, toString, wait, wait, wait

Dettagli sul programma di creazione

SSLConfiguration

```
public SSLConfiguration()
```

Crea una nuova SSLConfiguration. L'identificazione d'applicazione e le informazioni sul file di chiavi vengono inizializzate su valori predefiniti.

Il valore predefinito per l'identificazione d'applicazione è il valore specificato per la proprietà ″os400.secureApplication″.

I valori predefiniti per le informazioni del file di chiavi sono null se la proprietà ″os400.secureApplication″ viene specificata. Se la proprietà ″os400.secureApplication″ non viene specificata, il valore predefinito per il nome del file di chiavi è il valore specificato per la proprietà ″os400.certificateContainer″. Se la proprietà ″os400.secureApplication″ non viene specificata, l'etichetta del file di chiavi viene inizializzata sul valore della proprietà ″os400.certificateLabel″. Se non viene impostata la proprietà ″os400.secureApplication″ o ″os400.certificateContainer″, il nome del file di chiavi verrà inizializzato su ″*SYSTEM″.

Dettagli del metodo

---

main

```
public static void main(java.lang.String[]args)
```

Esegue le funzioni SSLConfiguration. Esistono quattro comandi che possono essere eseguiti: -help, -create, -display e -update. Il comando deve essere il primo parametro specificato.

Le seguenti opzioni possono essere specificate (in qualsiasi ordine):

-keystore **keystore-file-name**
: Specifica il nome del file di chiavi da creare, aggiornare o visualizzare. Questa opzione viene richiesta per tutti i comandi.

-storepass **keystore-file-password**
: Specifica la parola d'ordine associata al file di chiavi da creare, aggiornare o visualizzare. Questa opzione viene richiesta per tutti i comandi.

-storetype **keystore-type**
: Specifica il tipo di file di chiavi da creare, aggiornare o visualizzare. Questa opzione può essere specificata per qualsiasi comando. Se questa opzione non viene specificata, viene utilizzato un valore di ″IbmISeriesKeyStore″.

-appid **application-identifier**
: Specifica l'identificazione di applicazione da utilizzare per inizializzare un file di chiavi che si stava creando o aggiornando. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-keyring **keyring-file-name**
: Specifica il nome del file di chiavi da utilizzare per inizializzare un file di chiavi che si stava

creando o aggiornando. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-keyringpass **keyring-file-password**
Specifica la parola d'ordine del file di chiavi da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione può essere specificata per i comandi *-create* e *-update*, viene richiesta quando viene specificato un tipo memorizzazione chiave diverso da "IbmISeriesKeyStore". Se questa opzione non viene specificata, viene utilizzata la parola d'ordine nascosta del file di chiavi.

-keyringlabel **keyring-file-label**
Specifica l'etichetta del file di chiavi da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione può essere specificata solo quando viene specificata anche l'opzione *-keyring*. Se questa opzione non viene specificata quando viene specificata l'opzione *keyring*, viene utilizzata l'etichetta predefinita all'interno del file di chiavi.

-systemdefault
Specifica il valore predefinito del sistema da utilizzare per inizializzare un file memorizzazione chiave in fase di creazione o aggiornamento. Questa opzione è facoltativa per i comandi *-create* e *-update*. Solo una delle opzioni *-appid*, *keyring* e *-systemdefault* può essere specificata.

-v Specifica che l'emissione verbose deve essere prodotta. Questa opzione può essere specificata per qualsiasi comando.

Il comando di aiuto visualizza le informazioni sull'utilizzo per specificare i parametri in questo metodo. I parametri per richiamare la funzione di aiuto vengono specificati come segue:

```
-help
```

Il comando di creazione crea un nuovo file memorizzazione chiave. Esistono tre variazioni del comando di creazione. La prima variazione per creare una memorizzazione chiave si basa su una specifica identificazione di applicazione, la seconda su un nome, etichetta, e parola d'ordine del file di chiavi e la terza sulla configurazione predefinita del sistema.

Per creare una parola d'ordine basata su un'identificazione di applicazione specifica, deve essere specificata l'opzione *-appid*. I seguenti parametri creano un file memorizzazione chiave di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base all'identificazione di applicazione "APPID":

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -appid APPID
```

Per creare una memorizzazione chiave basata su un file di chiavi specifico, deve essere specificata l'opzione *-keyring*. Devono essere specificate anche le opzioni *-keyringpass* e *keyringlabel*. I seguenti parametri creano un file di chiavi di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base al file di chiavi denominato "keyring.file", alla parola d'ordine del file di chiavi "ringpass" e all'etichetta del file di chiavi "keylabel":

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

Per creare una memorizzazione chiave basata sulla configurazione predefinita del sistema, deve essere specificata l'opzione the *-systemdefault*. I seguenti parametri creano un file memorizzazione chiave di tipo "IbmISeriesKeyStore" denominato "keystore.file" con una parola d'ordine di "keypass" che viene inizializzata in base alla configurazione predefinita del sistema:

```
-create -keystore keystore.file -storepass keypass -systemdefault
```

Il comando di aggiornamento, aggiorna un file memorizzazione chiave esistente di tipo "IbmISeriesKeyStore". Esistono tre variazioni del comando di aggiornamento identiche alle variazioni del comando di creazione. Le opzioni per il comando di aggiornamento sono identiche alle opzioni utilizzate per il comando di creazione. Il comando di visualizzazione, visualizza la configurazione specificata per

un file memorizzazione chiave esistente. I seguenti parametri visualizzano la configurazione specificata da un file memorizzazione chiave di tipo ″IbmISeriesKeyStore″ denominato ″keystore.file″ con una parola d'ordine di ″keypass″:

```
-display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

Parametri:
args - gli argomenti della riga comandi

---

run

```
public void run(java.lang.String[]args,
                java.io.PrintStreamout)
```

Esegue le funzioni SSLConfiguration. I parametri e la funzionalità di questo metodo sono identici al metodo main().

Parametri:
args - gli argomenti del comando

out - il flusso di emissione i cui risultati devono essere scritti

Vedere anche:com.ibm.i5os.jsse.SSLConfiguration.main()

---

getApplicationId

```
public java.lang.String getApplicationId()
```

Restituisce l'ID di applicazione.

Restituisce:
l'ID di applicazione.

---

getKeyringName

```
public java.lang.String getKeyringName()
```

Restituisce il nome del file di chiavi.

Restituisce:
il nome del file di chiavi.

---

getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

Restituisce l'etichetta del file di chiavi.

Restituisce:
l'etichetta del file di chiavi.

---

getKeyringPassword

```
public final char[] getKeyringPassword()
```

Restituisce la parola d'ordine del file di chiavi.

Restituisce:
la parola d'ordine del file di chiavi.

---

finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Richiamato dal programma di raccolta di dati su un oggetto quando la raccolta di dati determina che non esistono più riferimenti all'oggetto.

Sovrascrive:
`finalize` nella classe `java.lang.Object`

Emette:
`java.lang.Throwable` - l'eccezione provocata da questo metodo.

---

clear

```
public void clear()
```

Cancella tutte le informazioni all'interno dell'oggetto in modo che tutti i metodi ottenuti restituiscano null.

---

setKeyring

```
public void setKeyring(java.lang.Stringname,
                       java.lang.Stringlabel,
                       char[]password)
```

Imposta le informazioni sul file di chiavi.

Parametri:
`name` - il nome del file di chiavi

`label` - l'etichetta del file di chiavi o null se deve essere utilizzata la voce del file di chiavi predefinito.

`password` - la parola d'ordine del file di chiavi o null se deve essere utilizzata la parola d'ordine nascosta.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId)
```

Imposta l'ID di applicazione.

Parametri:
`applicationId` - l'ID di applicazione.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId,
                             char[]password)
```

imposta l'ID di applicazione e la parola d'ordine del file di chiavi. La specifica della parola d'ordine del file di chiavi consente a qualsiasi memorizzazione chiave creata di consentire l'accesso alla chiave privata.

Parametri:

`applicationId` - l'ID di applicazione.

`password` - la parola d'ordine del file di chiavi.

---

equals

```
public boolean equals(java.lang.Objectobj)
```

Indica se altri oggetti sono ″uguali a″ questo.

Sovrascrive:

`equals` nella classe `java.lang.Object`

Parametri:

`obj` - oggetto da confrontare

Restituisce:

un indicatore che conferma se gli oggetti specificano le stesse informazioni di configurazione

---

hashCode

```
public int hashCode()
```

Restituisce un valore del codice hash per l'oggetto.

Sovrascrive:

`hashCode` nella classe `java.lang.Object`

Restituisce:

un valore del codice hash per questo oggetto.

---

clone

```
public java.lang.Object clone()
```

Crea una nuova copia di questa configurazione SSL. Le modifiche successive effettuate ai componenti di questa configurazione SSL, non interessano la nuova copia e viceversa.

Sovrascrive:

`clone` nella classe `java.lang.Object`

Restituisce:

una copia di questa configurazione SSL

---

getKeyStore

```
public java.security.KeyStore getKeyStore(char[]password)
                                  throws java.security.KeyStoreException
```

Restituisce una memorizzazione chiave di tipo ″IbmISeriesKeyStore″ utilizzando la parola d'ordine specifica. La memorizzazione chiave viene inizializzata in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto.

Parametri:
`password` - utilizzata per inizializzare la memorizzazione chiave

Restituisce:
KeyStore, inizializzato in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto

Emette:
`java.security.KeyStoreException` - se non è stato possibile creare la memorizzazione chiave

---

getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                           char[]password)
                                  throws java.security.KeyStoreException
```

Restituisce una memorizzazione chiave del tipo richiesto utilizzando la parola d'ordine specifica. La memorizzazione chiave viene inizializzata in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto.

Parametri:
`type` - tipo di memorizzazione chiave da restituire

`password` - utilizzata per inizializzare la memorizzazione chiave

Restituisce:
KeyStore, inizializzato in base alle informazioni attuali sulla configurazione memorizzate all'interno dell'oggetto

Emette:
`java.security.KeyStoreException` - se non è stato possibile creare la memorizzazione chiave

Esempi: IBM Java Secure Sockets Extension 6:

Gli esempi JSSE mostrano il modo in cui un client e un server possono utilizzare il fornitore System i5 JSSE nativo per creare un contesto che abilita le comunicazioni sicure.

Nota: entrambi gli esempi utilizzano il fornitore JSSE nativo di System i5, indipendentemente dalle proprietà specificate dal file java.security.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

Esempio: client SSL che utilizza un oggetto SSLContext per la versione 6:

Questo programma client di esempio utilizza un oggetto SSLContext, da esso inizializzato per utilizzare l'ID applicazione ″MY_CLIENT_APP″. Questo programma utilizzerà l'implementazione System i5 nativa indipendentemente da quanto specificato nel file java.security.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
| ////////////////////////////////////////////////////////////////////////////
| //
| // Questo programma client di esempio utilizza un oggetto SSLContext, da esso inizializzato
| // per utilizzare l'ID applicazione "MY_CLIENT_APP".
| //
| // L'esempio utilizza il fornitore System i5 JSSE nativo, indipendentemente dalle
| // proprietà specificate dal file java.security.
| //
| // Sintassi del comando:
| //     java -Djava.version=1.6 SslClient
| //
| // Notare che "-Djava.version=1.6" non è necessario se è stato configurato
| // come valore predefinito l'utilizzo di JDK versione 6.
| //
| ////////////////////////////////////////////////////////////////////////////
|
| import java.io.*;
| import javax.net.ssl.*;
| import java.security.*;
| import com.ibm.i5os.jsse.SSLConfiguration;
| /**
|  * Programma client SSL.
|  */
| public class SslClient {
|
|    /**
|     * Metodo principale SslClient.
|     *
|     * @param args gli argomenti della riga comandi (non utilizzato)
|     */
|    public static void main(String args[]) {
|        /*
|         * Impostare in modo da catturare gli errori inviati.
|         */
|        try {
|            /*
|             * Inizializzare un oggetto SSLConfiguration per specificare un ID
|             * ID.  "MY_CLIENT_APP" deve essere registrato e configurato
|             * correttamente con il DCM (Digital Certificate Manager).
|             */
|            SSLConfiguration config = new SSLConfiguration();
|            config.setApplicationId("MY_CLIENT_APP");
|            /*
|             * Richiamare un oggetto KeyStore dall'oggetto SSLConfiguration.
|             */
|            char[] password = "password".toCharArray();
|            KeyStore ks = config.getKeyStore(password);
|            /*
|             * Assegnare ed inizializzare un KeyManagerFactory.
|             */
|            KeyManagerFactory kmf =
|                            KeyManagerFactory.getInstance("IbmISeriesX509");
|            kmf.init(ks, password);
|            /*
|             * Assegnare ed inizializzare un TrustManagerFactory.
|             */
|            TrustManagerFactory tmf =
|                           TrustManagerFactory.getInstance("IbmISeriesX509");
|            tmf.init(ks);
|            /*
|             * Assegnare ed inizializzare un SSLContext.
|             */
|            SSLContext c =
|                       SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
|            c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
|            /*
|             * Richiamare un SSLSocketFactory dal SSLContext.
```

```
             */
            SSLSocketFactory sf = c.getSocketFactory();
            /*
             * Creare un SSLSocket.
             *
             * Modificare l'indirizzo IP codificato con l'indirizzo IP o il nome host
             * del server.
             */
            SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
            /*
             * Inviare un messaggio al server utilizzando la sessione protetta.
             */
            String sent = "Test of java SSL write";
            OutputStream os = s.getOutputStream();
            os.write(sent.getBytes());
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Wrote " + sent.length() + " bytes...");
            System.out.println(sent);
            /*
             * Ricevere un messaggio dal server utilizzando la sessione protetta.
             */
            InputStream is = s.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead = is.read(buffer);
            if (bytesRead == -1)
                throw new IOException("Unexpected End-of-file Received");
            String received = new String(buffer, 0, bytesRead);
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Read " + received.length() + " bytes...");
            System.out.println(received);
        } catch (Exception e) {
            System.out.println("Unexpected exception caught: " +
                               e.getMessage());
            e.printStackTrace();
        }
    }

}
```

Esempio: server SSL che utilizza un oggetto SSLContext per la versione 6:

Il seguente programma server utilizza un oggetto SSLContext da esso inizializzato, con un file memorizzazione chiave precedentemente creato.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////////
//
// Il seguente programma server utilizza un oggetto SSLContext da esso
// inizializzato, con un file memorizzazione chiave precedentemente creato.
//
// Il nome e la parola d'ordine memorizzazione chiave del file memorizzazione chiave sono:
//   Nome file: /home/keystore.file
//   Parola d'ordine:  password
//
// È necessario che il programma di esempio disponga del file memorizzazione chiave per poter creare un
// oggetto IbmISeriesKeyStore. L'oggetto KeyStore deve specificare MY_SERVER_APP come
// identificativo dell'applicazione.
//
// Per creare il file memorizzazione chiave, è possibile utilizzare il seguente comando Qshell:
//
```

```
| //   java com.ibm.i5os.SSLConfiguration  -create  -keystore /home/keystore.file
| //     -storepass password  -appid MY_SERVER_APP
| //
| // Sintassi del comando:
| //     java -Djava.version=1.6 JavaSslServer
| //
| // Notare che "-Djava.version=1.6" non è necessario se è stato configurato
| // come valore predefinito l'utilizzo di JDK versione 6.
| //
| // È anche possibile creare il file memorizzazione chiave immettendo questo comando su una richiesta comandi i5/OS:
| //
| //  RUNJVA CLASS(com.ibm.i5os.SSLConfiguration) PARM('-create' '-keystore'
| //      '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
| //
| ////////////////////////////////////////////////////////////////////////////
|
| import java.io.*;
| import javax.net.ssl.*;
| import java.security.*;
| /**
| * Programma server SSL Java che utilizza l'ID applicazione.
| */
| public class JavaSslServer {
|
|    /**
|     * Metodo principale JavaSslServer.
|     *
|     * @param args gli argomenti della riga comandi (non utilizzato)
|     */
|    public static void main(String args[]) {
|        /*
|         * Impostare in modo da catturare gli errori inviati.
|         */
|        try {
|            /*
|             * Assegnare ed inizializzare un oggetto KeyStore.
|             */
|            char[] password = "password".toCharArray();
|            KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
|            FileInputStream fis = new FileInputStream("/home/keystore.file");
|            ks.load(fis, password);
|            /*
|             * Assegnare ed inizializzare un KeyManagerFactory.
|             */
|            KeyManagerFactory kmf =
|                              KeyManagerFactory.getInstance("IbmISeriesX509");
|            kmf.init(ks, password);
|            /*
|             * Assegnare ed inizializzare un TrustManagerFactory.
|             */
|            TrustManagerFactory tmf =
|                             TrustManagerFactory.getInstance("IbmISeriesX509");
|            tmf.init(ks);
|            /*
|             * Assegnare ed inizializzare un SSLContext.
|             */
|            SSLContext c =
|                       SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
|            c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
|            /*
|             * Richiamare un SSLServerSocketFactory dal SSLContext.
|             */
|            SSLServerSocketFactory sf = c.getServerSocketFactory();
|            /*
|             * Creare un SSLServerSocket.
|             */
|            SSLServerSocket ss =
```

```
                (SSLServerSocket) sf.createServerSocket(13333);
            /*
             * Eseguire un accept() per creare un SSLSocket.
             */
            SSLSocket s = (SSLSocket) ss.accept();
            /*
             * Ricevere un messaggio dal client utilizzando la sessione protetta.
             */
            InputStream is = s.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead = is.read(buffer);
            if (bytesRead == -1)
                throw new IOException("Unexpected End-of-file Received");
            String received = new String(buffer, 0, bytesRead);
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Read " + received.length() + " bytes...");
            System.out.println(received);
            /*
             * Rimandare il messaggio al client utilizzando la sessione protetta.
             */
            OutputStream os = s.getOutputStream();
            os.write(received.getBytes());
            /*
             * Scrivere i risultati sullo schermo.
             */
            System.out.println("Wrote " + received.length() + " bytes...");
            System.out.println(received);
        } catch (Exception e) {
            System.out.println("Unexpected exception caught: " +
                               e.getMessage());
            e.printStackTrace();
        }
    }

}
```

JAAS (Java Authentication and Authorization Service)

JAAS (Java Authentication and Authorization Service) è un'estensione standard a J2SE (Java 2 Platform, Standard Edition). J2SE fornisce controlli di accesso basati su dove è stato generato il codice e su chi lo ha firmato (controlli di accesso basati sull'origine del codice). Manca, comunque, la capacità di forzare ulteriori controlli sull'accesso basati sulla persona che esegue il codice. JAAS fornisce una framework che aggiunge questo supporto al modello di sicurezza Java 2.

L'implementazione JAAS su System i5 è compatibile con l'implementazione di Sun Microsystems, Inc. Questa documentazione tratta degli aspetti unici dell'implementazione di System i5. Si presume una familiarità dell'utente con tale materiale generale relativa alle estensioni JAAS. Per rendere più facile per l'utente la gestione di tale documentazione e delle informazioni System i5, vengono forniti i seguenti collegamenti.

Informazioni correlate

Javadoc JAAS specifico per il server System i5

Specifica dell'API JAAS
Contiene informazioni Javadoc su JAAS.

JAAS LoginModule
Tratta in dettaglio gli aspetti correlati all'autenticazione di JAAS.

Preparazione e configurazione di un System i5 per JAAS (Java Authentication and Authorization Service)

È necessario soddisfare i requisiti software e configurare System i5 per utilizzare JAAS (Java Authentication and Authorization Service).

Requisiti software per eseguire JAAS 1.0 su un System i5

Installare i seguenti programmi su licenza:

- J2SDK (Java 2 SDK), versione 1.4 o successive
- È necessario IBM Toolbox per Java (mod 4) programma su licenza (LP - Licensed Program) (5761-JC1) per modificare l'identità del sottoprocesso OS. Esso contiene le classi ProfileTokenCredential necessarie per supportare la modifica dell'identità del sottoprocesso OS di System i5 e le classi di implementazione native.

Configurare il sistema

Per configurare il sistema in modo che utilizzi JAAS, seguire queste fasi:

1. Un file predefinito login.config è fornito in ${java.home}/lib/security e richiama com.ibm.as400.security.auth.login.BasicAuthenticationLoginModule. Tale file login.config collega un semplice ProfileTokenCredential di utilizzo al soggetto autenticato. Se si desidera utilizzare il proprio file login.config con differenti opzioni, è possibile includere la seguente proprietà di sistema quando si richiama la propria applicazione:

   ```
   -Djava.security.auth.login.config=il file login.config
   ```

2. Aggiungere un collegamento simbolico all'indirizzario dell'estensione per il file jt400Native.jar. Ciò consente al programma di caricamento classi di estensioni di caricare questo file.

   Il collegamento simbolico del file jt400Native.jar all'indirizzario /QIBM/ProdData/Java400/jdk14/lib/ext forza tutti gli utenti J2SDK 1.4 sul server all'esecuzione con questa versione di jt400Native.jar. Ciò potrebbe non essere consigliabile se vari utenti richiedono differenti versioni delle classi di IBM Toolbox per Java. Altre opzioni includono la collocazione di jt400Native.jar nell'applicazione CLASSPATH come descritto precedentemente. Un'altra opzione è aggiungere il collegamento simbolico al proprio indirizzario e quindi includere tale indirizzario nel classpath dell'indirizzario dell'estensione, specificando la proprietà di sistema java.ext.dirs quando si richiama l'applicazione.

   Per collegare il file jt400Native.jar all'indirizzario /QIBM/ProdData/Java400/jdk14/lib/ext, eseguire questo comando sulla riga comandi i5/OS per aggiungere il collegamento:

   ```
   ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
      NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')
   ```

   Per collegare il file jt400Native.jar al proprio indirizzario, effettuare quanto segue:

   a. Eseguire questo comando sulla riga comandi i5/OS per aggiungere il collegamento:

      ```
      ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
         NEWLNK('indirizzario estensione/jt400Native.jar')
      ```

   b. Quando si chiama il proprio programma java, utilizzare il seguente modello:

      ```
      java -Djava.ext.dirs=your extension directory:default
      extension directories
      ```

      Nota: consultare IBM Toolbox per Java per informazioni sulle classi di credenziali System i5. Fare clic su **Classi di sicurezza**. Fare clic su **Servizi di autenticazione**. Fare clic sulla classe **ProfileTokenCredential**. Fare clic su **Pacchetto**.

3. Aggiornare i file delle politiche Java 2 per concedere le appropriate autorizzazioni alle reali ubicazioni dei file JAR di IBM Toolbox per Java. Anche se è possibile collegare simbolicamente questi file agli indirizzari dell'estensione e a questi indirizzari è concesso java.security.AllPermission nel file ${java.home}/lib/security/java.policy, l'autorizzazione si basa sulla reale ubicazione dei file JAR.

Per utilizzare con esito positivo le classi credenziali in IBM Toolbox per Java, aggiungere quanto segue al file delle politiche Java 2 della propria applicazione:

```
grant codeBase "file:/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar"
 {
   permission javax.security.auth.AuthPermission "modifyThreadIdentity";
   permission java.lang.RuntimePermission "loadLibrary.*";
   permission java.lang.RuntimePermission "writeFileDescriptor";
   permission java.lang.RuntimePermission "readFileDescriptor";
 }
```

È inoltre necessario aggiungere queste autorizzazioni per il codeBase della propria applicazione dal momento che le operazioni eseguite dai file JAR di IBM Toolbox per Java non vengono eseguite in modalità privilegiata.

Consultare "JAAS (Java Authentication and Authorization Service) 1.0" per informazioni sui file delle politiche Java 2.

4. Assicurarsi che i server host System i5 siano avviati e in esecuzione. Le classi ProfileTokenCredential che si trovano nel Toolbox, ad esempio, jt400Native.jar, vengono utilizzate come credenziali collegate al soggetto autenticato. Le classi credenziali richiedono l'accesso ai server host. È possibile verificare che i server siano avviati e in esecuzione immettendo quanto segue sulla richiesta comandi i5/OS:

   ```
   StrHostSVR *all
   StrTcpSvr *DDM
   ```

   Se i server sono già stati avviati, queste fasi non ottengono alcun risultato. Se i server non sono avviati, vengono avviati con queste fasi.

JAAS (Java Authentication and Authorization Service) 1.0

JAAS (Java Authentication and Authorization Service) è un'estensione standard al J2SDK (Java 2 Software Development Kit), versione 1.3. Attualmente, Java 2 fornisce dei controlli sull'accesso basati sull'origine del codice (ossia dei controlli dell'accesso basati sul *luogo di origine* e sul *firmatario* del codice). Non può, tuttavia, applicare ulteriori controlli sull'accesso basati sull'*esecutore* del codice. JAAS fornisce un framework che aggiunge questo supporto al modello di sicurezza Java 2.

Guida per gli sviluppatori

- **Panoramica**
- **A chi è rivolto questo manuale**
- **Documentazione correlata**
- **Introduzione**
- **Classi principali**
- **Classi comuni**
  - **Soggetto**
  - **Principal**
  - **Credenziali**
- **Classi di autenticazione**
- **LoginContext**
- **LoginModule**
- **CallbackHandler**
- **Callback**
- **Classi di autorizzazione**
- **Politica**
- **AuthPermission**
- **PrivateCredentialPermission**

Riferimenti

- Implementazione
- ″Hello World″, stile JAAS.
- Appendice A: impostazioni JAAS nel file delle proprietà della sicurezza java.security
- Appendice B: file di configurazione del collegamento
- Appendice C: file delle politiche di autorizzazione

Panoramica

JAAS (Java Authentication and Authorization Service) è un'estensione standard al J2SDK (Java 2 Software Development Kit), versione 1.3. Attualmente, Java 2 fornisce dei controlli sull'accesso basati sull'origine del codice (ossia dei controlli dell'accesso basati sul *luogo di origine* e sul *firmatario* del codice). Non può, tuttavia, applicare ulteriori controlli sull'accesso basati sull'*esecutore* del codice. JAAS fornisce un framework che aggiunge questo supporto al modello di sicurezza Java 2.

L'ultimo aggiornamento di questo manuale risale al 17 marzo 2000.

A chi è rivolto questo manuale

Questo manuale è rivolto ai programmatori esperti che desiderano creare delle applicazioni limitate da un modello di sicurezza basato sul soggetto e sull'origine del codice.

Documentazione correlata

Questo manuale presume che l'utente abbia già letto la seguente documentazione:

- Specifica API di Java 2 Software Development Kit
- Specifica dell'API JAAS
- Sicurezza e piattaforma Java

Il manuale LoginModule Developer's Guide fornito da Sun Microsystems, Inc è un supplemento al presente manuale.

Introduzione

L'infrastruttura JAAS può essere divisa in due componenti principali: un componente di **autenticazione** e un componente di **autorizzazione**. Il componente di autenticazione JAAS consente di determinare, in modo affidabile e sicuro, chi sta attualmente elaborando il codice Java, indipendentemente dal fatto che il codice sia in esecuzione come applicazione, applet, bean o servlet. Il componente di autorizzazione JAAS integra il framework di sicurezza di Java 2 fornendo gli strumenti necessari per impedire al codice Java di elaborazione di eseguire delle attività delicate, a seconda della sua origine del codice (come in Java 2) e dell'identità dell'utente autenticato.

L'autenticazione JAAS viene eseguita in modalità *″pluggable″* (ossia collegabile). Questo consente alle applicazioni Java di rimanere indipendenti dalle sottostanti tecnologie di autenticazione. Pertanto, le tecnologie di autenticazione nuove o aggiornate possono essere collegate sotto un'applicazione senza che sia necessario apportare modifiche all'applicazione stessa. Le applicazioni abilitano il processo di autenticazione istanziando un oggetto

`LoginContext`

, che a sua volta fa riferimento ad un oggetto

`Configuration`

per determinare la tecnologia di autenticazione oppure ad un oggetto

`LoginModule`

da utilizzare nell'esecuzione dell'autenticazione. Di norma, gli oggetti LoginModule possono richiedere e verificare un nome utente ed una parola d'ordine. Degli altri possono leggere e verificare un campione vocale o di impronta digitale.

Dopo che l'utente che sta elaborando il codice è stato autenticato, il componente di autorizzazione JAAS lavora insieme al modello di controllo dell'accesso Java 2 esistente per proteggere l'accesso alle risorse sensibili. A differenza di Java 2, dove le decisioni di controllo dell'accesso sono basate esclusivamente sull'ubicazione e sui firmatari del codice (un
`CodeSource`

), in JAAS le decisioni di controllo dell'accesso sono basate sia sul valore
`CodeSource`

che sull'utente che esegue il codice, o il
`Subject`

. Notare che la politica JAAS si limita ad estendere la politica Java 2 con le pertinenti informazioni basate sul Subject (soggetto). Pertanto, le autorizzazioni riconosciute e comprese in Java 2 (
`java.io.FilePermission`

e
`java.net.SocketPermission`

, ad esempio) sono comprese e riconosciute anche da JAAS. Inoltre, anche se la politica di sicurezza JAAS è fisicamente separata dalla politica di sicurezza Java 2 esistente, le due politiche insieme formano una sola politica logica.

Classi principali

Le classi principali JAAS possono essere suddivise in tre categorie: comuni, di autenticazione e di autorizzazione.

- Classi comuni
  - Soggetto, principal, credenziali
- Classi di autenticazione
  - LoginContext, LoginModule, CallbackHandler, Callback
- Classi di autorizzazione
  - Policy, AuthPermission, PrivateCredentialPermission

Classi comuni

Le classi comuni sono condivise sia con i componenti di autorizzazione che con quelli di autenticazione JAAS.

La classe JAAS di chiavi è
`Subject`

, che rappresenta un raggruppamento di informazioni correlate per una singola entità, come ad esempio una persona. Comprende le credenziali private, le credenziali pubbliche e i Principal dell'entità.

Notare che JAAS utilizza l'interfaccia Java 2
`java.security.Principal`

esistente per rappresentare un Principal. Notare inoltre che JAAS non introduce una classe o un'interfaccia di credenziali separata. Una credenziale, come definita da JAAS, può essere qualsiasi oggetto.

Soggetto

Per autorizzare l'accesso alle risorse, le applicazioni devono prima autenticare l'origine della richiesta. Il framework JAAS definisce il termine, Soggetto, per rappresentare l'origine di una richiesta. Un Soggetto può essere qualsiasi entità, come ad esempio una persona o un servizio. Una volta autenticato, un Soggetto viene popolato con le identità associate, o Principal. Un Soggetto può avere molti Principal. Ad esempio, una persona può avere un nome Principal (″John Doe″) e un SSN Principal (″123-45-6789″) che la distingue da altri Soggetti.

Un
`Subject`

(soggetto) può avere anche degli attributi correlati alla sicurezza, indicati come credenziali. Le credenziali sensibili che richiedono una protezione speciale, come ad esempio le chiavi crittografiche private, sono memorizzate in un
`Set`

di credenziali private. Le credenziali di cui è prevista la condivisione, come ad esempio i certificati di chiavi pubbliche o i ticket Kerberos, sono memorizzati in un
`Set`

di credenziali pubbliche . Per accedere e modificare i vari set di credenziali sono richieste della autorizzazioni differenti.

I soggetti sono creati utilizzando questi programmi di creazione:

```
public Subject();

public Subject(boolean readOnly, Set principals,
               Set pubCredentials, Set privCredentials);
```

Il primo programma di creazione crea un Soggetto con dei set di Principal e credenziali vuoti (non nulli). Il secondo programma di creazione crea un Soggetto con i set di Principal e credenziali specificati. Ha anche un argomento booleano che può creare un Soggetto di sola lettura (set di Principale e di credenziali immutabili).

Un metodo alternativo per ottenere un riferimento ad un Soggetto autenticato senza utilizzare questi programmi di creazione verrà mostrato nella sezione LoginContext.

Se un Soggetto non è stato istanziato per essere in uno stato di sola lettura, è possibile impostarlo su uno stato di sola lettura richiamando questo metodo:

```
public void setReadOnly();
```

Un
`AuthPermission("setReadOnly")`

è richiesto per richiamare questo metodo. Una volta eseguita l'impostazione su uno stato di sola lettura, eventuali tentativi di aggiungere o rimuovere dei Principal o delle credenziali determineranno la generazione di una
`IllegalStateException`

.

Questo metodo può essere richiamato per testare lo stato di sola lettura di un Soggetto:

```
public boolean isReadOnly();
```

Per richiamare i Principal associati ad un Soggetto, sono disponibili due metodi:

```
public Set getPrincipals();
public Set getPrincipals(Class c);
```

Il primo metodo restituisce tutti i Principal contenuti nel Soggetto, mentre il secondo metodo restituisce solo i Principal che sono un'istanza della Classe c specificata oppure un'istanza di una sottoclasse della Classe c. Se al Soggetto non è associato alcun Principal, verrà restituito un set vuoto.

Per richiamare le credenziali pubbliche associate ad un Soggetto, sono disponibili i seguenti metodi:

```
public Set getPublicCredentials();
public Set getPublicCredentials(Class c);
```

Il comportamento osservato di questi metodi è identico a quello per il metodo
`getPrincipals`
.

Per accedere alle credenziali private associate ad un Soggetto, sono disponibili i seguenti metodi:

```
public Set getPrivateCredentials();
public Set getPrivateCredentials(Class c);
```

Il comportamento osservato di questi metodi è identico a quello per i metodi
`getPrincipals`
e
`getPublicCredentials`
.

Per modificare o agire su un set di Principal, un set di credenziali pubbliche o un set di credenziali private del Soggetto, i chiamanti utilizzano i metodi definiti nella classe
`java.util.Set`
. Il seguente esempio dimostra questo:

```
Subject subject;
Principal principal;
Object credential;

// aggiungere un Principal e delle credenziali al Soggetto
subject.getPrincipals().add(principal);
subject.getPublicCredentials().add(credential);
```

Notare che un
`AuthPermission("modifyPrincipals")`
,
`AuthPermission("modifyPublicCredentials")`
o
`AuthPermission("modifyPrivateCredentials")`
è richiesto per modificare i rispettivi set. Notare inoltre che solo i set restituiti tramite i metodi

getPrincipals

,
getPublicCredentials

e
getPrivateCredentials

sono supportati dai rispettivi set interni del Soggetto. Pertanto, qualsiasi modifica al set restituito influenza anche i set interni. I set restituiti tramite i metodi
getPrincipals(Class c)

,
getPublicCredentials(Class c)

e
getPrivateCredentials(Class c)

non sono supportati dai rispettivi set interni del Soggetto. Un nuovo set viene creato e restituito per ogni richiamo di metodo. Eventuali modifiche a questi set non influenzeranno i set interni del Soggetto. Il seguente metodo restituisce il Soggetto associato allo specifico
AccessControlContext

oppure ″null″ se nessun Soggetto è associato al
AccessControlContext

.
```
    public static Subject getSubject(final AccessControlContext acc);
```

Un
AuthPermission("getSubject")

è richiesto per richiamare
Subject.getSubject

.

La classe Subject (Soggetto) include anche questi metodi ereditati da
java.lang.Object

:
```
    public boolean equals(Object o);
    public String toString();
    public int hashCode();
```

I seguenti metodi statici possono essere richiamati per eseguire del lavoro come uno specifico Soggetto:
```
    public static Object doAs(final Subject subject,
                        final java.security.PrivilegedAction action);

    public static Object doAs(final Subject subject,
                        final java.security.PrivilegedExceptionAction action)
                        throws java.security.PrivilegedActionException;
```

Entrambi i metodi associano prima il ***soggetto*** specificato al
AccessControlContext

del sottoprocesso corrente ed elaborano quindi l'***azione***. In questo modo, l'***azione*** viene eseguita come il ***soggetto***. Il primo metodo può generare delle eccezioni di esecuzione ma, con una normale elaborazione, restituisce un oggetto dal metodo run() del suo argomento di azione. Il secondo metodo si comporta in modo analogo, con l'eccezione che può generare un'eccezione controllata (o checked exception) dal suo metodo

`PrivilegedExceptionAction`

run(). Un

`AuthPermission("doAs")`

è richiesto per richiamare i metodi

`doAs`

.

Sono qui riportati due esempi di utilizzo del primo metodo

`doAs`

. Si presuma che un

`Subject`

con un Principal di classe

`com.ibm.security.Principal`

denominato ″BOB″ sia stato autenticato da un

`LoginContext`

″lc″. Si presuma inoltre che sia stato installato un SecurityManager e che nella politica per il controllo accessi JAAS esista quanto segue (vedere la sezione relativa alle politiche per ulteriori dettagli sul file politiche JAAS):

```
   // Concedere a "BOB" l'autorizzazione a leggere il file "foo.txt"
   grant Principal com.ibm.security.Principal "BOB" {
      permission java.io.FilePermission "foo.txt", "read";
};
```

Subject.doAs Example 1

```
    class ExampleAction implements java.security.PrivilegedAction {
        public Object run() {
            java.io.File f = new java.io.File("foo.txt");

            // exists() richiama un controllo di sicurezza
            if (f.exists()) {
                System.out.println("Il file foo.txt esiste.");
            }
            return null;
        }
    }

    public class Example1 {
        public static void main(String[] args) {

            // Autenticare il soggetto, "BOB".
            // Questo processo è descritto nella
            // sezione LoginContext.

            Subject bob;
            ...
```

```
            // eseguire "ExampleAction" come "BOB":
            Subject.doAs(bob, new ExampleAction());
        }
    }
```

Durante l'elaborazione,
`ExampleAction`

rileverà un controllo di sicurezza quando esegue una chiamata a
`f.exists()`

. Tuttavia, poiché
`ExampleAction`

è in esecuzione come ″BOB″, e poiché la politica JAAS (sopra) concede la necessaria
`FilePermission`

a ″BOB″, la
`ExampleAction`

passerà il controllo di sicurezza.

L'esempio 2 ha lo stesso scenario dell'esempio 1.

Subject.doAs Example 2

```
    public class Example2 {
        // Esempio di utilizzo di una classe di azione anonima.
        public static void main(String[] args) {
            // Autenticare il soggetto, "BOB".
            // Questo processo è descritto nella
            // sezione LoginContext.

            Subject bob;
            ...

            // eseguire "ExampleAction" come "BOB":
            Subject.doAs(bob, new ExampleAction() {
                public Object run() {
                    java.io.File f = new java.io.File("foo.txt");
                    if (f.exists()) {
                        System.out.println("Il file foo.txt esiste.");
                    }
                    return null;
                }
            });
        }
    }
```

Entrambi gli esempi generano una
`SecurityException`

se l'istruzione di concessione (grant) dell'autorizzazione di esempio è modificata in modo non corretto, come ad esempio aggiungendo un CodeBase non corretto o modificando il Principal in ″MOE″. La rimozione del campo Principal dal blocco ″grant″ ed il suo successivo spostamento nel file di politica Java 2 non causerà la generazione di una
`SecurityException`

perché l'autorizzazione è ora più generica (disponibile a tutti i Principal).

Poiché entrambi gli esempi eseguono la stessa funzione, ci dev'essere una ragione per scrivere il codice in un modo piuttosto che nell'altro. L'esempio 1 può essere più facile da leggere per i programmatori che non hanno dimestichezza con le classi anonime. Inoltre, la classe di ***azione*** può essere inserita in un file separato con un CodeBase univoco e queste informazioni possono essere quindi utilizzate dall'operazione di concessione (grant) delle autorizzazioni. L'esempio 2 è più compatto e l'***azione*** da eseguire è più facile da trovare poiché si trova giusto nella chiamata
`doAs`

.

I seguenti metodi eseguono anch'essi del lavoro su uno specifico Soggetto. Tuttavia, i metodi
`doAsPrivileged`

avranno dei controlli di sicurezza basati sull'***azione*** e sul ***Soggetto*** forniti. Il contesto fornito verrà collegato al ***Soggetto*** ed all'***azione*** specificati. Un oggetto di contesto nullo ignorerà del tutto l'
`AccessControlContext`

corrente.

```
    public static Object doAsPrivileged(final Subject subject,
                         final java.security.PrivilegedAction action,
                         final java.security.AccessControlContext acc);

    public static Object doAsPrivileged(final Subject subject,
                         final java.security.PrivilegedExceptionAction action,
                         final java.security.AccessControlContext acc)
                         throws java.security.PrivilegedActionException;
```

I metodi
`doAsPrivileged`

si comportano in modo analogo ai metodi
`doAs`

: il ***Soggetto*** è associato al contesto ***acc***, viene eseguita l'***azione*** e possono essere generate delle eccezioni di esecuzione o delle eccezioni controllate (o checked exception). Tuttavia, i metodi
`doAsPrivileged`

svuotano l'
`AccessControlContext`

del sottoprocesso esistente prima di associare il ***Soggetto*** al contesto fornito e prima di richiamare l'***azione***. In caso di argomento ***acc*** nullo, le decisioni di controllo dell'accesso (richiamante durante i processi dell'***azione***) sono basati esclusivamente sul ***Soggetto*** e sull'***azione***. Un
`AuthPermission("doAsPrivileged")`

è richiesto quando si richiamano i metodi
`doAsPrivileged`

.

Principal

Come precedentemente menzionato, i Principal possono essere associati ad un Soggetto. I Principal rappresentano le identità del Soggetto e devono implementare le interfacce
`java.security.Principal`

e
`java.io.Serializable`

. La sezione relativa al Soggetto descrive i modi per aggiornare i Principal associati ad un Soggetto.

Credenziali

le classi di credenziali pubbliche e private non fanno parte della libreria di classi JAAS principale. Qualsiasi classe Java può pertanto rappresentare una credenziale. Tuttavia, gli sviluppatori possono scegliere che le loro classi di credenziali implementino due interfacce correlate alle credenziali: Refreshable e Destroyable.

Refreshable

Questa **interfaccia** consente ad una credenziale di aggiornarsi automaticamente. Ad esempio, una credenziale con una specifica durata con limitazione temporale può implementare quest'interfaccia per consentire ai chiamanti di aggiornare il periodo di tempo per cui è valida. L'interfaccia ha due metodi astratti:

```
boolean isCurrent();
```

Determina se la credenziale è corrente o valida.

```
void refresh() throws RefreshFailedException;
```

Aggiorna o estende la validità della credenziale. Questa implementazione di metodi esegue un controllo di sicurezza
`AuthPermission("refreshCredential")`

per assicurare che il chiamante disponga dell'autorizzazione necessaria per aggiornare la credenziale.

Destroyable

Questa **interfaccia** consente di eliminare il contenuto all'interno di una credenziale. L'interfaccia ha due metodi astratti:

```
boolean isDestroyed();
```

Determina se la credenziale è stata eliminata.

```
void destroy() throws DestroyFailedException;
```

Elimina e scarta le informazioni associate a questa credenziale. Le chiamate successive ad alcuni metodi di questa credenziale causeranno la generazione di una
`IllegalStateException`

. Questa implementazione di metodi esegue un controllo di sicurezza
`AuthPermission("destroyCredential")`

per assicurare che il chiamante disponga dell'autorizzazione necessaria per eliminare la credenziale.

Classi di autenticazione

Per autenticare un
`Subject`

, vengono eseguiti i seguenti passi:

1. Un'applicazione istanzia un

`LoginContext`

.

2. Il
   `LoginContext`

   consulta una configurazione per caricare tutti i LoginModule configurati per tale applicazione.
3. L'applicazione richiama il metodo di *login* del LoginContext.
4. Il metodo di *login* richiama tutti i LoginModule caricati. Ciascun
   `LoginModule`

   prova ad autenticare il
   `Subject`

   . In caso di esito positivo, i LoginModule associano i Principal e le credenziali pertinenti al
   `Subject`

   .
5. Il
   `LoginContext`

   restituisce lo stato dell'autenticazione all'applicazione.
6. Se l'autenticazione ha avuto esito positivo, l'applicazione richiama il
   `Subject`

   autenticato dal
   `LoginContext`

   .

LoginContext

La classe
`LoginContext`

fornisce i metodi di base utilizzati per autenticare i Soggetti e fornisce un modo per sviluppare un'applicazione indipendente dalla tecnologia di autenticazione sottostante. Il
`LoginContext`

consulta una configurazione
`Configuration`

per determinare i servizi di autenticazione, o LoginModule, configurati per una specifica applicazione. Pertanto, dei LoginModule differenti possono essere collegati sotto un'applicazione senza richiedere alcuna modifica all'applicazione stessa.
`LoginContext`

offre quattro programmi di creazione da cui scegliere:

```
public LoginContext(String name) throws LoginException;

public LoginContext(String name, Subject subject) throws LoginException;

public LoginContext(String name, CallbackHandler callbackHandler)
       throws LoginException
```

```
public LoginContext(String name, Subject subject,
       CallbackHandler callbackHandler) throws LoginException
```

Tutti i programmi di creazione condividono un parametro comune: ***name***. Questo argomento è utilizzato dal

`LoginContext`

per indicizzare la configurazione di collegamento. I programmi di creazione che non prendono un

`Subject`

come parametro di immissione istanziano un nuovo

`Subject`

. Le immissioni nulle non sono consentite per tutti i programmi di creazione. I chiamanti richiedono una

`AuthPermission("createLoginContext")`

per istanziare un

`LoginContext`

.

L'autenticazione effettiva si verifica con una chiamata al seguente metodo:

```
public void login() throws LoginException;
```

Quando viene richiamato il *login*, tutti i metodi di *login* rispettivi dei LoginModule configurati sono richiamati per eseguire l'autenticazione. Se l'autenticazione è stata eseguita correttamente, il

`Subject`

autenticato (che può ora detenere dei Principal, delle credenziali pubbliche e delle credenziali private) può essere richiamato utilizzando il seguente metodo:

```
public Subject getSubject();
```

Per scollegare un

`Subject`

e rimuovere i suoi Principal e credenziali autenticati, viene fornito il seguente metodo:

```
public void logout() throws LoginException;
```

Il seguente frammento di codice in un'applicazione autenticherà un Soggetto denominato ″bob″ dopo l'accesso ad un file di configurazione con una voce di configurazione denominata ″moduleFoo″:

```
Subject bob = new Subject();
LoginContext lc = new LoginContext("moduleFoo", bob);
try {
    lc.login();
    System.out.println("autenticazione riuscita");
} catch (LoginException le) {
    System.out.println("autenticazione non riuscita"+le.printStackTrace());
}
```

Questo frammento di codice in un'applicazione autenticherà un Soggetto ″senza nome″ ed utilizzerà quindi il metodo getSubject per richiamarlo:

```
LoginContext lc = new LoginContext("moduleFoo");
try {
    lc.login();
    System.out.println("autenticazione riuscita");
```

```
    } catch (LoginException le) {
        System.out.println("autenticazione non riuscita"+le.printStackTrace());
    }
    Subject subject = lc.getSubject();
```

Se l'autenticazione ha avuto esito negativo, *getSubject* restituisce un valore nullo. Inoltre, non c'è una
`AuthPermission("getSubject")`

richiesta per eseguire questa operazione come nel caso di
`Subject.getSubject`

LoginModule

L'**interfaccia** LoginModule consente agli sviluppatori di implementare diversi tipi di tecnologie di autenticazione che possono essere collegati sotto un'applicazione. Ad esempio, un tipo di
`LoginModule`

può eseguire una forma di autenticazione basata su nome utente/parola d'ordine.

Il manuale LoginModule Developer's Guide è un documento dettagliato che fornisce agli sviluppatori delle istruzioni dettagliate e progressive per implementare i LoginModule.

Per istanziare un
`LoginModule`

, un
`LoginContext`

prevede che ciascun
`LoginModule`

fornisca un programma di creazione pubblico che non assume alcun argomento. Quindi, per inizializzare un
`LoginModule`

con le informazioni pertinenti, un
`LoginContext`

richiama il metodo
`initialize`

del LoginModule. Il *Soggetto* fornito è sicuramente non nullo.

```
   void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options);
```

Il seguente metodo avvia il processo di autenticazione:

```
    boolean login() throws LoginException;
```

Un'implementazione di metodo di esempio può richiedere all'utente un nome utente ed una parola d'ordine e verificare quindi le informazioni confrontandole con i dati memorizzati in un servizio di denominazione come NIS o LDAP. Delle implementazioni alternative possono interfacciarsi con delle smart card e con dei dispositivi biometrici oppure possono semplicemente estrarre le informazioni sull'utente dal sottostante sistema operativo. Questa è considerata la ***fase 1*** del processo di autenticazione JAAS.

Il seguente metodo completa e finalizza il processo di autenticazione:

```
boolean commit() throws LoginException;
```

Se la ***fase 1*** del processo di autenticazione è stata eseguita correttamente, questo metodo continua con la ***fase 2***: associazione di Principal, credenziali pubbliche e credenziali private con il Soggetto. Se la ***fase 1*** ha avuto esito negativo, il metodo *commit* rimuove qualsiasi eventuale stato di autenticazione precedentemente memorizzato, come ad esempio dei nomi utente e delle parole d'ordine.

Il seguente metodo arresta il processo di autenticazione se la ***fase 1*** ha avuto esito negativo:

```
boolean abort() throws LoginException;
```

Le tipiche implementazioni di questo metodo eliminano l'eventuale stato di autenticazione memorizzato in precedenza, come ad esempio i nomi utente o le parole d'ordine. Il seguente metodo scollega un Soggetto:

```
boolean logout() throws LoginException;
```

Questo metodo rimuove i Principal e le credenziali originariamente associati al
`Subject`

durante l'operazione di
`commit`

. Le credenziali vengono eliminate all'atto della rimozione.

CallbackHandler

In alcuni casi, un LoginModule deve comunicare con l'utente per ottenere le informazioni di autenticazione. Per questo scopo, i LoginModule utilizzano un CallbackHandler. Le applicazioni implementano l'**interfaccia** CallbackHandler e la passano al LoginContext, che la inoltra direttamente ai LoginModule sottostanti. I LoginModule utilizzano il CallbackHandler sia per raccogliere l'immissione dagli utenti (come ad esempio una parola d'ordine o un numero pin di una smart card) oppure per fornire delle informazioni agli utenti (come ad esempio delle informazioni sullo stato). Consentendo all'applicazione di specificare il CallbackHandler, i LoginModule sottostanti possono restare indipendenti dai vari modi in cui le applicazioni interagiscono con gli utenti. Ad esempio, l'implementazione di un CallbackHandler per un'applicazione GUI può visualizzare una finestra per sollecitare l'immissione da un utente. L'implementazione di un CallbackHandler per uno strumento non GUI può richiedere all'utente di eseguire l'immissione direttamente dalla riga comandi.
`CallbackHandler`

è un'**interfaccia** con un metodo per implementare:

```
void handle(Callback[] callbacks)
throws java.io.IOException, UnsupportedCallbackException;
```

Callback

Il pacchetto javax.security.auth.callback contiene l'**interfaccia** Callback e varie implementazioni. I LoginModule possono passare una schiera di Callback direttamente al metodo *handle* di un CallbackHandler.

Consultare le varie API Callback per ulteriori informazioni sul loro utilizzo.

Classi di autorizzazione

In caso di corretta autenticazione di un
`Subject`

, è possibile applicare i controlli dell'accesso dettagliati su tale
`Subject`

richiamando il metodo Subject.doAs o il metodo Subject.doAsPrivileged. Le autorizzazioni concesse a tale
`Subject`

sono configurate in una
`Policy`

JAAS.

Politica

Questa è una classe **astratta** per rappresentare il controllo dell'accesso JAAS a livello del sistema. Per impostazione predefinita, JAAS fornisce un'implementazione di classe secondaria basata sui file, PolicyFile. Ciascuna classe secondaria
`Policy`

deve implementare i seguenti metodi:

```
public abstract java.security.PermissionCollection getPermissions
                (Subject subject,
                java.security.CodeSource cs);
public abstract void refresh();
```

Il metodo
`getPermissions`

restituisce le autorizzazioni concesse agli specifici
`Subject`

e
`CodeSource`

. Il metodo
`refresh`

aggiorna la
`Policy`

di tempo di esecuzione con le eventuali modifiche apportate dall'ultima volta che è stata caricata dal suo archivio permanente (un file o un database, ad esempio). Il metodo
`refresh`

richiede un
`AuthPermission("refreshPolicy")`

.

Il seguente metodo richiama l'oggetto
`Policy`

di tempo di esecuzione corrente ed è protetto con un controllo di sicurezza che richiede che il chiamante abbia una
`AuthPermission("getPolicy")`

.

```
    public static Policy getPolicy();
```

Il seguente codice di esempio dimostra come un oggetto
`Policy`

può essere interrogato per il set di autorizzazioni concesso agli specifici
`Subject`

e
`CodeSource`

:

```
    policy = Policy.getPolicy();
    PermissionCollection perms = policy.getPermissions(subject, codeSource);
```

Per impostare un nuovo oggetto
`Policy`

per il tempo di esecuzione Java, è possibile utilizzare il metodo
`Policy.setPolicy`

. Questo metodo richiede che il chiamante abbia una
`AuthPermission("setPolicy")`

.

```
    public static void setPolicy(Policy policy);
```

Voci di esempio dei file di politiche:

Questi esempi sono pertinenti solo per l'implementazione PolicyFile predefinita.

Ciascuna voce nella
`Policy`

è rappresentata come una voce ***grant***. Ciascuna voce ***grant*** specifica una terzina codebase/firmatari-codice/Principal e le autorizzazioni ad essa concesse. In modo specifico, le autorizzazioni verranno concesse a qualsiasi codice scaricato dal *codebase* specificato e firmato dai *firmatari del codice* specificati, a condizione che il
`Subject`

che esegue tale codice abbia tutti i *Principal* specificati nel suo set di
`Principal`

. Fare riferimento agli esempi Subject.doAs per vedere come un
`Subject`

diventa associato al codice in esecuzione.

```
    grant CodeBase ["URL"],
          Signedby ["signers"],
          Principal [Principal_Class] "Principal_Name",
          Principal ... {
        permission Permission_Class ["Target_Name"]
                                    [, "Permission_Actions"]
                                    [, signedBy "SignerName"];
```

```
};

// voce grant di esempio
grant CodeBase "http://griffin.ibm.com", Signedby "davis",
      Principal com.ibm.security.auth.NTUserPrincipal "kent" {
    permission java.io.FilePermission "c:/kent/files/*", "read, write";
};
```

Se non viene specificata alcuna informazione sul *Principal* nella voce ″grant″ della
`Policy`

JAAS, verrà generata un'eccezione di analisi. Tuttavia, le voci ″grant″ che già esistono nel file di politiche basato sull'origine del codice Java 2 regolare (e pertanto senza informazioni sul *Principal*) sono ancora valide. In questi casi, viene considerato implicito che le informazioni sul *Principal* siano '*' (le voci ″grant″ (concessione) sono valide per tutti i Principal).

I componenti CodeBase e Signedby della voce ″grant″ (concessione) sono facoltativi nella
`Policy`

JAAS. Se non sono presenti, corrisponderanno tutti i codebase e tutti i firmatari (compreso il codice non firmato).

Nell'esempio precedente, la voce ***grant*** specifica che il codice scaricato da ″http://griffin.ibm.com″, firmato da ″davis″ ed in esecuzione come l'utente NT ″kent″, ha una
`Permission`

. Questa
`Permission`

consente al codice di elaborazione di leggere e scrivere file nell'indirizzario ″c:\kent\files″.

In una singola voce ***grant*** possono essere elencati più Principal. Il
`Subject`

corrente che esegue il codice deve avere tutti i Principal specificati nel suo set di
`Principal`

per avere la concessione delle autorizzazioni della voce.

```
grant Principal com.ibm.security.auth.NTUserPrincipal "kent",
      Principal com.ibm.security.auth.NTSidGroupPrincipal "S-1-1-0" {
    permission java.io.FilePermission "c:/user/kent/", "read, write";
    permission java.net.SocketPermission "griffin.ibm.com", "connect";
};
```

Questa voce concede all'eventuale codice in esecuzione come utente NT ″kent″ con il numero di identificazione di gruppo NT ″S-1-1-0″ l'autorizzazione a leggere e scrivere file in ″c:\user\kent″ e l'autorizzazione a stabilire collegamenti socket a ″griffin.ibm.com″.

AuthPermission

Questa classe incapsula le autorizzazioni di base richieste per JAAS. Una AuthPermission contiene un nome (indicato anche come ″nome di destinazione″) ma nessun elenco di azioni, quindi o si dispone dell'autorizzazione indicata oppure no. Oltre ai metodi ereditati (dalla classe
`Permission`

), una

`AuthPermission`

ha due programmi di creazione pubblici:

```
public AuthPermission(String name);
public AuthPermission(String name, String actions);
```

Il primo programma di creazione crea una nuova AuthPermission con il nome specificato. Il secondo programma di creazione crea anche un nuovo oggetto AuthPermission con il nome specificato ma ha un argomento *actions* aggiuntivo che è attualmente non utilizzato e sono nulli. Questo programma di creazione esiste esclusivamente per consentire all'oggetto

`Policy`

di istanziare dei nuovi oggetti Permission. Per la maggior parte del codice, il primo programma di creazione è appropriato.

L'oggetto AuthPermission è utilizzato per proteggere l'accesso agli oggetti Policy, Subject, LoginContext e Configuration. Consultare il Javadoc di AuthPermission per l'elenco di nomi validi supportati.

PrivateCredentialPermission

Questa classe protegge l'accesso alle credenziali private di un Soggetto e fornisce un programma di creazione pubblico:

```
public PrivateCredentialPermission(String name, String actions);
```

Consultare il Javadoc di PrivateCredentialPermission per informazioni più dettagliate su questa classe.

Implementazione

Nota: l'Appendice A contiene un file **java.security** di esempio che include le proprietà statiche qui citate.

Poiché esistono dei valori predefiniti per i file di politiche ed i fornitori JAAS, gli utenti non devono elencare né staticamente (nel file java.security file) né dinamicamente (opzione della riga comandi ***-D***) i loro valori per implementare JAAS. Inoltre, i fornitori di file di politiche e di configurazione predefiniti possono essere sostituiti da un fornitore sviluppato dall'utente. Questa sezione, pertanto, è un tentativo di spiegare i file di politiche e i fornitori predefiniti JAAS e le proprietà che abilitano i fornitori alternativi.

Per altre informazioni, oltre a quelle qui riepilogate, leggere la API dei file di configurazione e di politiche predefiniti.

Fornitore di autenticazione

Il fornitore di autenticazione, o classe di configurazione, è impostato staticamente con

`login.configuration.provider=[class]`

nel file *java.security*. Questo fornitore crea l'oggetto

`Configuration`

.

Ad esempio:

```
login.configuration.provider=com.foo.Config
```

Se la proprietà di sicurezza

`login.configuration.provider`

non viene trovata in java.security, JAAS la imposterà sul valore predefinito:
`com.ibm.security.auth.login.ConfigFile`

.

Se un gestore della sicurezza è impostato prima che venga creato l'oggetto
`Configuration`

, sarà richiesta la concessione di una
`AuthPermission("getLoginConfiguration")`

.

Non è possibile impostare dinamicamente il fornitore di configurazione sulla riga comandi.

File di configurazione dell'autenticazione

I file di configurazione dell'autenticazione possono essere impostati staticamente in *java.security* con
`login.config.url.n=[URL]`

, dove *n* è un valore numerico intero consecutivo che inizia con 1. Il formato è identico al formato per i file di politiche di sicurezza Java (policy.url.n=[URL]).

Se la proprietà di sicurezza
`policy.allowSystemProperty`

è impostata su ″true″ in java.security, gli utenti possono impostare dinamicamente i file di politiche sulla riga comandi utilizzando l'opzione ***-D*** con questa proprietà:
`java.security.auth.login.config`

. Il valore può essere un percorso oppure un URL. Ad esempio (su NT):

```
... -Djava.security.auth.login.config=c:\config_policy\login.config ...
o
... -Djava.security.auth.login.config=file:c:/config_policy/login.config ...
```

Nota: l'utilizzo di segni di uguale doppi (==) sulla riga comandi consente all'utente di sostituire tutti gli altri file di politiche trovati.

Se non è possibile trovare alcun file di configurazione staticamente o dinamicamente, JAAS proverà a caricare il file di configurazione da questa ubicazione predefinita:
`${user.home}\.java.login.config`

dove *${user.home}* è un'ubicazione che dipende dal sistema.

Fornitore di autorizzazione

Il fornitore di autorizzazione, o classe di politica JAAS, è impostato staticamente con
`auth.policy.provider=[class]`

nel file *java.security*. Questo fornitore crea l'oggetto
`Policy`

basato sul Soggetto di JAAS.

Ad esempio:

```
auth.policy.provider=com.foo.Policy
```

Se la proprietà di sicurezza
`auth.policy.provider`

non viene trovata in java.security, JAAS la imposterà sul valore predefinito:
`com.ibm.security.auth.PolicyFile`

.

Se un gestore della sicurezza è impostato prima che venga creato l'oggetto
`Configuration`

, sarà richiesta la concessione di una
`AuthPermission("getPolicy")`

.

Non è possibile impostare dinamicamente il fornitore di autorizzazione sulla riga comandi.

File di politiche di autorizzazione

I file di politiche di autorizzazione possono essere impostati staticamente in *java.security* con
`auth.policy.url.n=[URL]`

, dove *n* è un valore numerico intero consecutivo che inizia con 1. Il formato è identico al formato per i file di politiche di sicurezza Java (policy.url.n=[URL]).

Se la proprietà di sicurezza
`policy.allowSystemProperty`

è impostata su ″true″ in java.security, gli utenti possono impostare dinamicamente i file di politiche sulla riga comandi utilizzando l'opzione ***-D*** con questa proprietà:
`java.security.auth.policy`

. Il valore può essere un percorso oppure un URL. Ad esempio (su NT):

```
... -Djava.security.auth.policy=c:\auth_policy\java.auth.policy ...
o
... -Djava.security.auth.policy=file:c:/auth_policy/java.auth.policy ...
```

Nota: l'utilizzo di segni di uguale doppi (==) sulla riga comandi consente all'utente di sostituire tutti gli altri file di politiche trovati.

Non esiste un'ubicazione predefinita da cui caricare una politica di autorizzazione.

″Hello World″, stile JAAS.

È ora disponibile un altro programma ″Hello World″ **JAAS**. In questa sezione, sarà reso disponibile un programma per verificare l'installazione JAAS.

Installazione Si presume che JAAS sia stato installato. Ad esempio, i file JAR JAAS sono stati copiati nell'indirizzario delle estensioni del Development Kit.

Richiamare i file Scaricare theHelloWorld.tar nell'indirizzario di verifica. Espanderlo utilizzando ″jar xvf HelloWorld.tar″.

Verificare il contenuto dell'indirizzario di verifica.

file di origine:
- HWLoginModule.java
- HWPrincipal.java
- HelloWorld.java

file di classe
- I file di origine sono stati precompilati per conto dell'utente nell'indirizzario delle classi.

file di politiche
- jaas.config
- java2.policy
- jaas.policy

Compilare i file di origine I tre file di origine, ***HWLoginModule.java***, ***HWPrincipal.java*** e ***HelloWorld.java*** sono già compilati e pertanto non richiedono alcuna compilazione.

Se qualche file di origine viene modificato, passare all'indirizzario di verifica dove sono stati salvati ed immettere:

```
javac -d .\classes *.java
```

Al classpath deve essere aggiunto l'indirizzario delle classi (.\classes) per poter compilare le classi.

Nota:

```
HWLoginModule
```

e

```
HWPrincipal
```

si trovano nel pacchetto

```
com.ibm.security
```

e saranno creati nell'appropriato indirizzario durante la compilazione (>indirizzario_verifica<\classes\com\ibm\security).

Esaminare i file di politiche Il file di configurazione, ***jaas.config***, contiene una voce:

```
helloWorld {
   com.ibm.security.HWLoginModule required debug=true;
};
```

Solo un

```
LoginModule
```

è fornito con il fascicolo di verifica. Quando si elabora l'applicazione ***HelloWorld***, fare delle prove modificando il

```
LoginModuleControlFlag
```

(required, requisite, sufficient, optional, ossia richiesto, requisito, sufficiente e facoltativo) e cancellando l'indicatore di debug. Se sono disponibili più LoginModule per la verifica, modificare questa configurazione e fare delle prove con più LoginModule.

```
HWLoginModule
```

verrà trattato brevemente.

Il file delle politiche Java 2, ***java2.policy***, contiene un blocco sull'autorizzazione:

```
grant {
   permission javax.security.auth.AuthPermission "createLoginContext";
   permission javax.security.auth.AuthPermission "modifyPrincipals";
   permission javax.security.auth.AuthPermission "doAsPrivileged";
};
```

Le tre autorizzazioni sono richieste perché l'applicazione ***HelloWorld*** (1) crea un oggetto LoginContext, (2) modifica i Principal del

`Subject`

autenticato e (3) richiama il metodo doAsPrivileged della classe

`Subject`

.

Anche il file delle politiche JAAS, ***jaas.policy***, contiene un blocco sull'autorizzazione:

```
grant Principal com.ibm.security.HWPrincipal "bob" {
   permission java.util.PropertyPermission "java.home", "read";
   permission java.util.PropertyPermission "user.home", "read";
   permission java.io.FilePermission "foo.txt", "read";
};
```

Le tre autorizzazioni sono inizialmente concesse ad un

`HWPrincipal`

denominato ***bob***. Il Principal effettivo aggiunto al

`Subject`

autenticato è il nome utente utilizzato durante il processo di collegamento (altre informazioni saranno fornite più avanti).

Questo è il codice di operazione da ***HelloWorld*** con le tre chiamate di sistema (il motivo per le autorizzazioni richieste) in **grassetto**:

```
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("\nYour java.home property: "
                           +System.getProperty("java.home"));

        System.out.println("\nYour user.home property: "
                           +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("\nfoo.txt does ");
        if (!f.exists()) System.out.print("not ");
        System.out.println("exist in your current directory");

        System.out.println("\nOh, by the way ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignorare
        }
        System.out.println("\n\nHello World!\n");
        return null;
    }
}, null);
```

Quando si esegue il programma ***HelloWorld***, utilizzare vari nomi utente e modificare ***jaas.policy*** di conseguenza. Non è necessario modificare ***java2.policy***. Creare inoltre un file denominato ***foo.txt*** nell'indirizzario di verifica per verificare l'ultima chiamata al sistema.

Esaminare i file di origine il LoginModule,
`HWLoginModule`

, autentica gli utenti che immettono la parola d'ordine corretta (sensibile al maiuscolo/minuscolo): **Go JAAS**.

L'applicazione ***HelloWorld*** concede agli utenti tre tentativi per effettuare tale operazione. Quando **Go JAAS** è immesso correttamente, un
`HWPrincipal`

con un nome uguale al nome utente viene aggiunto al
`Subject`

autenticato.

La classe Principal,
`HWPrincipal`

, rappresenta un Principal basato sul nome utente immesso. Questo nome è importante quando si concedono le autorizzazioni ai Soggetti autenticati.

L'applicazione principale,
`HelloWorld`

, crea prima un
`LoginContext`

basato su una voce di configurazione con il nome **helloWorld**. Il file di configurazione è già stato trattato. I Callback vengono utilizzati per richiamare l'immissione dell'utente. Controllare la classe
`MyCallbackHandler`

che si trova nel file ***HelloWorld.java*** per vedere questo processo.

```
LoginContext lc = null;
try {
    lc = new LoginContext("helloWorld", new MyCallbackHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}
```

L'utente immette una combinazione nome utente/parola d'ordine (fino a tre volte) e se **Go JAAS** viene immesso come parola d'ordine, il Soggetto viene autenticato (
`HWLoginModule`

aggiunge un
`HWPrincipal`

al Soggetto).

Come indicato in precedenza, il lavoro viene quindi eseguito come Soggetto autenticato.

Eseguire il test HelloWorld

Per eseguire il programma ***HelloWorld***, passare prima all'indirizzario di verifica. I file di configurazione e di politica dovranno essere caricati. Consultare Implementazione per le proprietà corrette da impostare in *java.security* oppure sulla riga comandi. Quest'ultimo metodo verrà qui trattato.

Il seguente comando è stato suddiviso in varie righe per una maggiore chiarezza. Immetterlo però come un singolo comando continuo.

```
java -Djava.security.manager=
     -Djava.security.auth.login.config=.\jaas.config
     -Djava.security.policy=.\java2.policy
     -Djava.security.auth.policy=.\jaas.policy
     HelloWorld
```

Nota: l'utilizzo di ″.\filename″ per i file delle politiche è necessario perché il percorso canonico dell'indirizzario di verifica di ciascun utente varierà. Volendo, sostituire a ″.″ il percorso all'indirizzario di verifica. Ad esempio, se l'indirizzario di verifica è ″c:\test\hello″, il primo file viene modificato in:

```
-Djava.security.auth.login.config=c:\test\hello\jaas.config
```

Se i file delle politiche non vengono trovati, verrà generata una

`SecurityException`

. Altrimenti, verranno visualizzate le informazioni relative alle proprietà ***java.home*** e ***user.home***. Inoltre, verrà verificata l'esistenza di un file denominato ***foo.txt*** nell'indirizzario di verifica. Infine, viene visualizzato il messaggio ricorrente ″Hello World″.

Divertirsi con HelloWorld

Rieseguire ***HelloWorld*** tutte le volte che si desidera. È stato già consigliato di variare le combinazioni di nome utente/parola d'ordine immesse, di modificare le voci del file di configurazione, di modificare le autorizzazioni al file delle politiche e anche di aggiungere ulteriori LoginModule (stack) alla voce di configurazione ***helloWorld***. È anche possibile aggiungere dei campi di base di codice ai file delle politiche.

Provare infine ad eseguire il programma senza un SecurityManager per vedere come funziona se si rilevano dei problemi.

Appendice A: impostazioni JAAS nel file delle proprietà della sicurezza java.security

Qui di seguito è riportata una copia del file

`java.security`

che compare in ogni installazione di Java 2. Questo file compare nell'indirizzario

`lib/security`

(

`lib\security`

su Windows) del tempo di esecuzione Java 2. Pertanto, se il tempo di esecuzione Java 2 è installato in un indirizzario denominato

`jdk1.3`

, il file è

- `jdk1.3/lib/security/java.security`

(Unix)

- `jdk1.3\lib\security\java.security`

  (Windows)

JAAS aggiunge quattro nuove proprietà a `java.security`:

- Proprietà di autenticazione
  - `login.configuration.provider`
  - `login.policy.url.n`
- Proprietà di autorizzazione
  - `auth.policy.provider`
  - `auth.policy.url.n`

Le nuove proprietà JAAS si trovano alla fine di questo file:

```
#
# This is the "master security properties file".
#
# In this file, various security properties are set for use by
# java.security classes. This is where users can statically register
# Cryptography Package Providers ("providers" for short). The term
# "provider" refers to a package or set of packages that supply a
# concrete implementation of a subset of the cryptography aspects of
# the Java Security API. A provider may, for example, implement one or
# more digital signature algorithms or message digest algorithms.
#
# Each provider must implement a subclass of the Provider class.
# To register a provider in this master security properties file,
# specify the Provider subclass name and priority in the format
#
#    security.provider.n=className
#
# This declares a provider, and specifies its preference
# order n. The preference order is the order in which providers are
# searched for requested algorithms (when no specific provider is
# requested). The order is 1-based; 1 is the most preferred, followed
# by 2, and so on.
#
# className must specify the subclass of the Provider class whose
# constructor sets the values of various properties that are required
# for the Java Security API to look up the algorithms or other
# facilities implemented by the provider.
#
# There must be at least one provider specification in java.security.
# There is a default provider that comes standard with the JDK. It
# is called the "SUN" provider, and its Provider subclass
# named Sun appears in the sun.security.provider package. Thus, the
# "SUN" provider is registered via the following:
#
#    security.provider.1=sun.security.provider.Sun
#
# (The number 1 is used for the default provider.)
```

```
#
# Note: Statically registered Provider subclasses are instantiated
# when the system is initialized. Providers can be dynamically
# registered instead by calls to either the addProvider or
# insertProviderAt method in the Security class.

#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun

#
# Class to instantiate as the system Policy. This is the name of the class
# that will be used as the Policy object.
#
policy.provider=sun.security.provider.PolicyFile

# The default is to have a single system-wide policy file,
# and a policy file in the user's home directory.
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy

# whether or not we expand properties in the policy file
# if this is set to false, properties (${...}) will not be expanded in policy
# files.
policy.expandProperties=true

# whether or not we allow an extra policy to be passed on the command line
# with -Djava.security.policy=somefile. Comment out this line to disable
# this feature.
policy.allowSystemProperty=true

# whether or not we look into the IdentityScope for trusted Identities
# when encountering a 1.1 signed JAR file. If the identity is found
# and is trusted, we grant it AllPermission.
policy.ignoreIdentityScope=false

#
# Default keystore type.
#
keystore.type=jks

#
# Class to instantiate as the system scope:
#
system.scope=sun.security.provider.IdentityDatabase

############################################################################
#
# Java Authentication and Authorization Service (JAAS)
# properties and policy files:
#

# Class to instantiate as the system Configuration for authentication.
# This is the name of the class that will be used as the Authentication
# Configuration object.
#
login.configuration.provider=com.ibm.security.auth.login.ConfigFile

# The default is to have a system-wide login configuration file found in
# the user's home directory.  For multiple files, the format is similar to
# that of CodeSource-base policy files above, that is policy.url.n
login.config.url.1=file:${user.home}/.java.login.config

# Class to instantiate as the system Principal-based Authorization Policy.
# This is the name of the class that will be used as the Authorization
# Policy object.
```

```
#
auth.policy.provider=com.ibm.security.auth.PolicyFile

# The default is to have a system-wide Principal-based policy file found in
# the user's home directory.  For multiple files, the format is similar to
# that of CodeSource-base policy files above, that is policy.url.n and
# auth.policy.url.n
auth.policy.url.1=file:${user.home}/.java.auth.policy
```

Appendice B: file di configurazione del collegamento

Un file di configurazione del collegamento contiene uno o più nomi applicazione
`LoginContext`

che hanno il seguente formato:

```
Application {
    LoginModule Flag ModuleOptions;
    > altre voci LoginModule <
    LoginModule Flag ModuleOptions;
};
```

I file di configurazione del collegamento sono individuati utilizzando la proprietà di sicurezza
`login.config.url.n`

che si trova nel file
`java.security`

.Per ulteriori informazioni su questa proprietà e sull'ubicazione del file
`java.security`

, consultare l'Appendice A.

Il valore *Flag* controlla il comportamento generale mentre l'autenticazione procede in ordine discendente lungo lo stack. I dati qui di seguito riportati rappresentano una descrizione dei valori validi per *Flag* e la loro rispettiva semantica:

1. **Richiesto** Il
   `LoginModule`

   è richiesto per una corretta esecuzione. Sia in caso di corretta esecuzione che di errore, l'autenticazione continua rispettando fino in fondo la sequenza dell'elenco
   `LoginModule`

   .
2. **Requisito** Il
   `LoginModule`

   è richiesto per una corretta esecuzione. In caso di corretta esecuzione, l'autenticazione continua rispettando fino in fondo la sequenza dell'elenco
   `LoginModule`

   . In caso di errore, il controllo viene restituito immediatamente all'applicazione (l'applicazione non continua rispettando fino in fondo la sequenza dell'elenco
   `LoginModule`

).

3. **Sufficiente** Il
   `LoginModule`

   non è richiesto per una corretta esecuzione. In caso di corretta esecuzione, il controllo viene immediatamente restituito all'applicazione (l'autenticazione non procede rispettando fino in fondo la sequenza dell'elenco
   `LoginModule`

). In caso di errore, l'autenticazione continua rispettando fino in fondo la sequenza dell'elenco
   `LoginModule`

   .
4. **Facoltativo** Il
   `LoginModule`

   non è richiesto per una corretta esecuzione. Sia in caso di corretta esecuzione che di errore, l'autenticazione continua rispettando fino in fondo la sequenza dell'elenco
   `LoginModule`

   .

L'autenticazione generale riesce solo se tutti i LoginModule *richiesti* e *requisiti* hanno avuto esito positivo. Se un
`LoginModule`

sufficiente è configurato e ha esito positivo, solo i
`LoginModule`

richiesti e *requisiti* che precedono quelli *sufficienti* devono aver avuto esito positivo perché l'autenticazione generale abbia esito positivo. Se non sono configurati LoginModule *richiesti* o *requisiti* per un'applicazione, deve avere esito positivo almeno un
`LoginModule`

sufficiente o *facoltativo.*

File di configurazione di esempio:

```
/*  File di configurazione di esempio  */

Login1 {
   com.ibm.security.auth.module.SampleLoginModule required debug=true;
};

Login2 {
   com.ibm.security.auth.module.SampleLoginModule required;
   com.ibm.security.auth.module.NTLoginModule sufficient;
   ibm.loginModules.SmartCard requisite debug=true;
   ibm.loginModules.Kerberos optional debug=true;
};
```

Nota: gli indicatori non sono sensibili al maiuscolo/minuscolo. *REQUISITE = requisite = Requisite.*

Login1 ha solo un LoginModule che è un'istanza della classe
`com.ibm.security.auth.module.SampleLoginModule`

. Pertanto, un'autenticazione
`LoginContext`

associata a **Login1** avrà un'autenticazione eseguita correttamente solo nel caso in cui l'autenticazione del suo singolo modulo venga eseguita correttamente. L'indicatore *richiesto* è di minima entità in quest'esempio; i valori di indicatore hanno un effetto rilevante sull'autenticazione quando sono presenti due o più moduli.

Login2 è più facile da spiegare con una tabella.

Stato autenticazione Login2									
Modulo di collegam. di esempio	richiesto	riuscito	riuscito	riuscito	riuscito	fallito	fallito	fallito	fallito
Modulo di collegam. NT	sufficiente	riuscito	fallito	fallito	fallito	riuscito	fallito	fallito	fallito
Smart Card	requisito	*	riuscito	riuscito	fallito	*	riuscito	riuscito	fallito
Kerberos	facoltativo	*	riuscito	fallito	*	*	riuscito	fallito	*
Autenticazione globale		riuscito	riuscito	riuscito	fallito	fallito	fallito	fallito	fallito

* = valore semplice dovuto al controllo che ritorna all'applicazione perché un modulo *REQUISITE* (requisito) precedente non è riuscito oppure perché un modulo *SUFFICIENT* (sufficiente) precedente ha avuto esito positivo.

Appendice C: file delle politiche di autorizzazione

Se i precedenti esempi di blocchi di concessione di politiche JAAS basati sui Principal non erano sufficienti, ne vengono qua proposti degli altri.

```
// SAMPLE JAAS POLICY FILE:  java.auth.policy

// Le seguenti autorizzazioni sono concesse al Principal 'Pooh' e a tutta l'origine di codice:

grant Principal com.ibm.security.Principal "Pooh" {
   permission javax.security.auth.AuthPermission "setPolicy";
   permission java.util.PropertyPermission "java.home", "read";
   permission java.util.PropertyPermission "user.home", "read";
   permission java.io.FilePermission "c:/foo/jaas.txt", "read";
};

// Le seguenti autorizzazioni sono concesse al Principal 'Pooh' E a 'Eyeore'
// e all'origine di codice firmata da "DrSecure":

grant signedBy "DrSecure"
      Principal com.ibm.security.Principal "Pooh",
      Principal com.ibm.security.Principal "Eyeore" {
   permission javax.security.auth.AuthPermission "modifyPublicCredentials";
   permission javax.security.auth.AuthPermission "modifyPrivateCredentials";
   permission java.net.SocketPermission "us.ibm.com", "connect,accept,resolve";
   permission java.net.SocketPermission "griffin.ibm.com", "accept";
};

// Le seguenti autorizzazioni son concesse al Principal 'Pooh' E a 'Eyeore' e a
// 'Piglet' e all'origine di codice dall'indirizzario c:\jaas firmato da "kent" e "bruce":

grant codeBase "file:c:/jaas/*",
      signedBy "kent, bruce",
```

```
      Principal com.ibm.security.Principal "Pooh",
      Principal com.ibm.security.Principal "Eyeore",
      Principal com.ibm.security.Principal "Piglet"  {
   permission javax.security.auth.AuthPermission "getSubject";
   permission java.security.SecurityPermission "printIdentity";
   permission java.net.SocketPermission "guapo.ibm.com", "accept";
};
```

Esempi di JAAS (Java Authentication and Authorization Service)

Quest'argomento contiene degli esempi di JAAS (Java Authentication and Authorization Service) su un System i5.

Esistono due esempi JAAS, HelloWorld e SampleThreadSubjectLogin. Fare clic su questi collegamenti per il codice sorgente e le istruzioni.

Compilazione ed esecuzione di HelloWorld con JAAS (Java Authentication and Authorization Service) su un System i5:

Queste informazioni esaminano come viene compilato ed eseguito **HelloWorld** per JAAS (Java Authentication and Authorization Service) su System i5.

Queste informazioni devono essere considerate una sostituzione per la sezione **HelloWorld** dell'argomento "JAAS (Java Authentication and Authorization Service) 1.0" a pagina 344. I file di configurazione, politica e codice sorgente sono gli stessi di quelli contenuti nel manuale API Developers Guide. Ci sono, tuttavia, alcuni aspetti che sono univoci per System i5.

1. Sarebbe opportuno inserire i seguenti file di origine nel proprio indirizzario di verifica:
   - HWLoginModule.java
   - HWPrincipal.java
   - HelloWorld.java

   È necessario compilare tali file di origine nel proprio indirizzario ./classes.

   Per esaminare il codice sorgente per questi file formattati per il browser HTML, consultare "Esempi: HelloWorld JAAS" a pagina 504.
2. È necessario compilare i tre file di origine, HWLoginModule.java, HWPrincipal.java e HelloWorld.java. Eseguire i seguenti comandi (ciascuno su una riga) su una riga comandi i5/OS:
   a. strqsh
   b. cd *yourTestDir*
   c.
   ```
   javac -J-Djava.version=1.3
         -classpath /qibm/proddata/os400/java400/ext/jaas13.jar:.
         -d ./classes *.java
   ```

   Dove *yourTestDir* è l'indirizzario creato per congelare i file di esempio. È necessario aggiungere l'indirizzario delle classi (.\classes) al classpath affinché esso compili le classi.

   Nota: HWLoginModule e HWPrincipal si trovano nel pacchetto com.ibm.security e vengono create nell'indirizzario appropriato durante la compilazione (\classes\com\ibm\security).
3. Eseguire i seguenti comandi (ciascuno su una riga) sulla riga comandi i5/OS:
   a. strqsh

b. cd *yourTestDir*

   Dove *yourTestDir* è l'indirizzario creato per congelare i file di esempio. È necessario aggiungere l'indirizzario delle classi (.\classes) al classpath affinché esso compili le classi.

c. Sarebbe opportuno inserire i seguenti file di origine nel proprio indirizzario di verifica:
   - jaas.config
   - java2.policy
   - jaas.policy

d.
```
java -Djava.security.manager=
     -Djava.security.auth.login.config=./jaas.config
     -Djava.security.policy=./java2.policy
     -Djava.security.auth.policy=./jaas.policy
     -Djava.version=1.3
     -classpath ./classes
     HelloWorld
```

e. Quando viene richiesto il nome utente, immettere **bob**. Se è in esecuzione con un responsabile della riservatezza, è necessario immettere l'utente**bob** affinché abbiano esito positivo tutte le autorizzazioni di accesso. Quando viene richiesta una parola d'ordine, immettere **Go JAAS**, sensibile al maiuscolo e al minuscolo, con uno spazio.

Dettagli: come funziona HelloWorld per JAAS (Java Authentication and Authorization Service):

Questo documento esamina da vicino come funziona **HelloWorld** per JAAS (Java Authentication and Authorization Service).

Sarebbe opportuno considerare tali informazioni come sostituzione per la sezione **HelloWorld** del manuale API Developers Guide. I file di configurazione, politica e codice sorgente sono gli stessi di quelli contenuti nel manuale API Developers Guide. Esistono, tuttavia, alcuni aspetti univoci per la piattaforma System i5.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

File di configurazione e delle politiche

Il file di configurazione, **jaas.config**, contiene una voce:
```
helloWorld {
    com.ibm.security.HWLoginModule required debug=true;
};
```

Il fascicolo di verifica include soltanto un LoginModule. Quando si esegue l'applicazione HelloWorld, è possibile effettuare una prova modificando LoginModuleControlFlag (necessario,requisito, sufficiente, facoltativo) e cancellando l'indicatore del debug. Se sono disponibili più LoginModule per la modifica, è possibile modificare questa configurazione e fare delle prove con più LoginModule.

Il file delle politiche Java 2, **java2.policy**, contiene un blocco sull'autorizzazione:
```
grant {
    permission javax.security.auth.AuthPermission "createLoginContext";
    permission javax.security.auth.AuthPermission "modifyPrincipals";
    permission javax.security.auth.AuthPermission "doAsPrivileged";
};
```

Le tre autorizzazioni sono richieste in quanto l'applicazione HelloWorld effettua quanto segue:

1. Crea un oggetto LoginContext.
2. Modifica i Principal del Soggetto autenticato.

3. Chiama il metodo doAsPrivileged della classe Subject.

Anche il file delle politiche JAAS, **jaas.policy**, contiene un blocco sull'autorizzazione:

```
grant Principal com.ibm.security.HWPrincipal "bob" {
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "foo.txt", "read";
};
```

Le tre autorizzazioni vengono inizialmente concesse ad un HWPrincipal denominato ″bob″. Il Principal effettivo aggiunto al Soggetto autenticato è il nome utente utilizzato durante il processo di collegamento.

Segue un codice di operazione da HelloWorld con le tre chiamate del sistema (il motivo per le autorizzazioni richieste) in grassetto:

```
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("\nYour java.home property: "
                           +System.getProperty("java.home"));

        System.out.println("\nYour user.home property: "
                           +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("\nfoo.txt does ");
        if (!f.exists()) System.out.print("not ");
        System.out.println("exist in your current directory");

        System.out.println("\nOh, by the way ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignorare
        }
        System.out.println("\n\nHello World!\n");
        return null;
    }
}, null);
```

Quando si esegue il programma HelloWorld, utilizzare vari nomi utente e modificare jaas.policy di conseguenza. Non dovrebbe essere necessario modificare java2.policy. Inoltre, creare un file denominato foo.txt nell'indirizzario di verifica per controllare l'ultima chiamata al sistema e confermare che viene concesso il corretto livello di accesso a quel file.

Esaminare i file di origine HelloWorld

La classe LoginModule, HWLoginModule autentica semplicemente ogni utente che immette la parola d'ordine corretta (sensibile al maiuscolo e al minuscolo, con spazio):

- **Go JAAS**

Se è in esecuzione con un responsabile della riservatezza, è necessario immettere 'bob' affinché abbiano esito positivo tutte le autorizzazioni di accesso.

L'applicazione HelloWorld concede agli utenti tre tentativi per effettuare ciò. Quando Go JAAS viene immesso correttamente, viene aggiunto un oggetto HWPrincipal con un nome uguale al nome utente aggiunto al soggetto autenticato.

La classe del Principal, HWPrincipal, rappresenta un Principal basato sul nome utente che viene immesso. Questo nome è importante quando si concedono le autorizzazioni ai Soggetti autenticati.

L'applicazione, HelloWorld, prima crea un LoginContext basato su una voce della configurazione con il nome helloWorld. I Callback vengono utilizzati per richiamare l'immissione dell'utente. Esaminare la classe MyCallbackHandler ubicata nel file HelloWorld.java per vedere questo processo. Segue un estratto dal codice sorgente:

```
LoginContext lc = null;
try {
    lc = new LoginContext("helloWorld", new MyCallbackHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}
```

L'utente immette un nome utente e una parola d'ordine (fino a tre volte) e se Go JAAS viene immesso come parola d'ordine, il soggetto viene autenticato (HWLoginModule aggiunge un HWPrincipal al soggetto). Il lavoro viene quindi eseguito come Soggetto autenticato.

Se non si trovano i file delle politiche, viene emessa una **SecurityException**. Altrimenti, vengono visualizzate le informazioni relative alle proprietà java.home e user.home. Inoltre, viene controllata l'esistenza di un file denominato foo.txt nel proprio indirizzario di verifica. Infine, viene visualizzato il messaggio ricorrente "Hello World".

Buon divertimento con HelloWorld

Rieseguire HelloWorld tutte le volte che si desidera. Segue una lista di alcune delle operazioni che è possibile provare:

- Modificare il nome utente e le parole d'ordine immessi
- Modificare le voci del file di configurazione
- Modificare le autorizzazioni file delle politiche
- Aggiungere i LoginModule aggiuntivi alla voce di configurazione helloWorld
- Aggiungere i campi di base del codice ai file delle politiche
- Eseguire il programma senza un SecurityManager per vedere come funziona se l'utente incontra dei problemi

Istruzioni SampleThreadSubjectLogin di JAAS (Java Authentication and Authorization Service):

Attenersi alle seguenti istruzioni per utilizzare l'esempio SampleThreadSubjectLogin.java.

File di origine

Collocare il file di origineSampleThreadSubjectLogin.java nel proprio indirizzario di verifica: è necessario compilare tale file nel proprio indirizzario ./classes.

Per consultare il codice sorgente in questo file che è stato formattato per il proprio browser HTML, esaminare Esempio: JAAS SampleThreadSubjectLogin.

File delle politiche

Esaminare il manuale API Developers Guide per informazioni generali sui file delle politiche JAAS. Seguono i file delle politiche specifici nell'esempio SampleThreadSubjectLogin:

- threadLogin.config
- threadJava2.policy
- threadJaas.policy

Esaminare i commenti all'inizio di SampleThreadSubjectLogin.java per informazioni sulla compilazione ed esecuzione di questo esempio.

IBM JGSS (Java Generic Security Service)

JGSS (Java Generic Security Service) fornisce un'interfaccia generica per l'autenticazione e la protezione dei messaggi. A questa interfaccia è possibile collegare una varietà di meccanismi di sicurezza basati su chiavi secrete, chiavi pubbliche o altre tecnologie di sicurezza.

Estraendo la complessità e le peculiarità dei meccanismi di sicurezza sottostanti in un'interfaccia standardizzata, JGSS fornisce i seguenti benefici per lo sviluppo di applicazioni di rete sicure:

- È possibile sviluppare l'applicazione per una singola interfaccia estratta
- È possibile utilizzare l'applicazione con meccanismi di sicurezza differenti senza dover effettuare modifiche

JGSS definisce i collegamenti Java per GSS-API (Generic Security Service Application Programming Interface), un'API crittografica standardizzata dalla IETF (Internet Engineering Task Force) ed adottata dalla X/Open Group.

L'implementazione IBM di JGSS viene denominata IBM JGSS. IBM JGSS è un'implementazione della framework GSS-API che utilizza Kerberos V5 come sistema di sicurezza sottostante predefinito. Fornisce anche una funzione costituita da un modulo di collegamento al JAAS(Java Authentication and Authorization Service) per la creazione e l'utilizzo delle credenziali Kerberos. È anche possibile fare in modo che JGSS effettui i controlli di autorizzazione JAAS quando si utilizzano quelle credenziali.

JGSS IBM include un fornitore JGSS System i5 nativo, un fornitore JGSS Java e delle versioni Java degli strumenti di gestione delle credenziali Kerberos (kinit, ktab e klist).

Nota: Il fornitore JGSS System i5 nativo utilizza la libreria NAS (Network Authentication Service) System i5 nativa. Quando si utilizza il fornitore nativo, è necessario utilizzare i programmi di utilità Kerberos System i5 nativi. Per ulteriori informazioni, consultare Fornitori JGSS.

Potenziamento della sicurezza da Sun Microsystems, Inc.

Internet Engineering Task Force (IETF) RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1

IETF RFC 2853 Generic Security Service API Version 2: Java Bindings

GSS-API Extensions for DCE della X/Open Group

Concetti su JGSS

Le operazioni JGSS sono composte da quattro livelli distinti, secondo gli standard di GSS-API (Generic Security Service Application Programming Interface).

I livelli vengono riportati di seguito:

1. Raccolta delle credenziali dei principal.
2. Creazione e stabilità del contesto di sicurezza tra i principal dei peer comunicanti
3. Scambio di messaggi sicuri tra i peer
4. Ripulitura e rilascio delle risorse

Inoltre JGSS livella JCA (Java Cryptographic Architecture) per offrire le funzioni di collegamento diretto a diversi meccanismi di sicurezza.

Utilizzare i seguenti collegamenti al fine di leggere descrizioni di alto livello di questi importanti concetti JGSS.

Principal e credenziali JGSS:

L'identità sotto la quale un'applicazione stabilisce comunicazioni sicure JGSS con un peer è detta principal. Un principal può essere un utente reale oppure un servizio non presieduto. Un principal acquisisce le credenziali specifiche per il meccanismo di sicurezza come prova dell'identità in base a tale meccanismo.

Ad esempio, quando si utilizza il meccanismo Kerberos, le credenziali di un principal si presentano come un TGT (ticket-granting ticket) emesso da un KDC (key distribution center) Kerberos. In un ambiente a più meccanismi, delle credenziali GSS-API possono contenere più elementi di credenziali e ciascun elemento rappresenta le credenziali di uno specifico meccanismo sottostante.

Lo standard GSS-API non prescrive il modo in cui un principal acquisisce le credenziali e le implementazioni GSS-API di norma non forniscono un mezzo per l'acquisizione delle credenziali. Un principal ottiene le credenziali prima di utilizzare GSS-API; GSS-API si limita a interrogare il meccanismo di sicurezza per le credenziali per conto del principal.

IBM JGSS include le versioni Java degli strumenti di gestione delle credenziali Kerberos com.ibm.security.krb5.internal.tools Class Kinit, com.ibm.security.krb5.internal.tools Class Ktab e com.ibm.security.krb5.internal.tools Class Klist. Inoltre, IBM JGSS potenzia il GSS-API standard fornendo un'interfaccia per il collegamento Kerberos facoltativa che utilizza JAAS. L'interfaccia per il collegamento facoltativa è supportata dal fornitore Java JGSS puro ma non è supportata dal fornitore i5/OS nativo.

Concetti correlati

"Acquisizione delle credenziali kerberos e creazione di chiavi segrete" a pagina 389
GSS-API non definisce una modalità per ottenere le credenziali. Per questo motivo, il meccanismo IBM JGSS Kerberos richiede che l'utente utilizzi uno dei seguenti metodi, per ottenere le credenziali Kerberos: questo argomento fornisce istruzioni su come ottenere le credenziali Kerberos, creare chiavi segrete, utilizzare il JASS per eseguire i collegamenti a Kerberos e i controlli delle autorizzazioni; fornisce un elenco delle autorizzazioni JAAS richieste dalla JVM (Java virtual machine).

"Fornitore JGSS" a pagina 386
IBM JGSS include un fornitore System i5 JGSS nativo ed un fornitore Java JGSS puro. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

com.ibm.security.krb5.internal.tools Class Klist:

Questa classe può essere eseguita come uno strumento di riga comandi per elencare le voci nella cache delle credenziali e nel keytab.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Klist
```

public class **Klist**
extends java.lang.Object

Questa classe può essere eseguita come uno strumento di riga comandi per elencare le voci nella cache delle credenziali e nel keytab.

Riepilogo del programma di creazione

```
Klist()
```

Riepilogo del metodo

static void	`main(java.lang.String[] args)` Il programma principale che può essere richiamato sulla riga comandi.

Metodi ereditati dalla classe java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Dettagli sul programma di creazione

Klist

```
public Klist()
```

Dettagli del metodo

main

```
public static void main(java.lang.String[] args)
```

Il programma principale che può essere richiamato sulla riga comandi.

Utilizzo: java com.ibm.security.krb5.tools.Klist [[-c] [-f] [-e] [-a]] [-k [-t] [-K]] [name]

Opzioni disponibili per le cache delle credenziali:
- **-f** mostra gli indicatori di credenziali
- **-e** mostra il tipo di crittografia
- **-a** visualizza l'elenco indirizzi

Opzioni disponibili per i keytab:
- **-t** mostra le date/ore delle voci dei keytab
- **-K** mostra le chiavi DES delle voci dei keytab

com.ibm.security.krb5.internal.tools Class Kinit:

Strumento Kinit per ottenere i ticket Kerberos v5.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Kinit
```

public class **Kinit**
extends java.lang.Object

Strumento Kinit per ottenere i ticket Kerberos v5.

Riepilogo del programma di creazione

`Kinit(java.lang.String[] args)`
Crea un nuovo oggetto Kinit.

Riepilogo del metodo

`static void`	`main(java.lang.String[] args)` Il metodo principale viene utilizzato per accettare l'immissione dalla riga comandi per la richiesta di ticket.

Metodi ereditati dalla classe java.lang.Object

```
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Dettagli sul programma di creazione

Kinit

```
public Kinit(java.lang.String[] args)
      throws java.io.IOException,
             RealmException,
             KrbException
```

Crea un nuovo oggetto Kinit.

Parametri:

`args` - schiera di opzioni di richiesta di ticket. Le opzioni disponibili sono: -f, -F, -p, -P, -c, -k, principal, password.

Genera:

java.io.IOException - se si verifica un errore I/E.
RealmException - se non è stato possibile istanziare l'ambito.
KrbException - se si verifica un errore durante l'operazione Kerberos.

Dettagli del metodo

main

```
public static void main(java.lang.String[] args)
```

Il metodo principale viene utilizzato per accettare l'immissione dalla riga comandi per la richiesta di ticket.

Utilizzo: java com.ibm.security.krb5.tools.Kinit [-f] [-F] [-p] [-P] [-k] [-c nome cache] [principal] [parola d'ordine]

- **-f** inoltrabile
- **-F** non inoltrabile
- **-p** inoltrabile via proxy
- **-P** non inoltrabile via proxy
- **-c** nome chace (ad es. FILE:d:\temp\mykrb5cc)
- **-k** utilizza keytab
- **-t** nome file keytab
- principal il nome di principal (ad es. qwedf qwedf@IBM.COM)
- password la parola d'ordine Kerberos del principal

Utilizzare java com.ibm.security.krb5.tools.Kinit -help per richiamare il menu dell'aiuto.

Attualmente è supportata solo la cache delle credenziali basata sui file. Per impostazione predefinita, un file di cache denominato krb5cc_{user.name} viene generato nell'indirizzario {user.home} per memorizzare il certificato ottenuto da KDC. Ad esempio, su Windows NT, può essere c:\winnt\profiles\qwedf\ krb5cc_qwedf, in cui qwedf è lo {user.name}, e c:\winnt\profile\qwedf è la {user.home}. La {user.home} viene ottenuta da Kerberos dalla proprietà di sistema Java ″user.home″. Se in qualche (raro) caso il valore {user.home} è nullo, il file di cache viene memorizzato nell'indirizzario corrente da cui è in esecuzione il

programma. {user.name} è il nome utente di accesso del sistema operativo. Può essere diverso dal nome di principal dell'utente. Un utente può avere più nomi principal, ma il principal principale della cache delle credenziali può essere soltanto uno, il che significa che un file di cache può memorizzare solo i ticket per uno specifico principal utente. Se l'utente passa il nome principal al Kinit successivo, il file di cache generato per il nuovo ticket sovrascrive il file di cache esistente per impostazione predefinita. Per evitare la sovrascrittura, è necessario specificare un indirizzario differente oppure un nome di file di cache differente quando si richiede un nuovo ticket.

Ubicazione del file di cache

Esistono vari modi per definire un nome ed un'ubicazione di file di cache specifico per l'utente; essi sono qui di seguito elencati nell'ordine in cui Kerberos effettua la ricerca:

1. opzione **-c**. Utilizzare java com.ibm.security.krb5.tools.Kinit -c FILE:<nome indirizzario e nome file specifici per l'utente>. ″FILE:″ è il prefisso per identificare il tipo di cache delle credenziali. Il valore predefinito è il tipo basato sui file.
2. Impostare la proprietà di sistema Java ″KRB5CCNAME″ utilizzando -DKRB5CCNAME=FILE:<nome indirizzario e nome file specifici per l'utente> durante l'esecuzione.
3. Impostare la variabile di ambiente ″KRB5CCNAME″ su una richiesta comandi prima dell'esecuzione. La modalità di impostazione delle variabili di ambiente varia a seconda del sistema operativo. Ad esempio, Windows utilizza set KRB5CCNAME=FILE:<nome indirizzario e nome file specifici per l'utente >, mentre UNIX utilizza export KRB5CCNAME=FILE:<nome indirizzario e nome file specifici per l'utente>. Notare che Kerberos si avvale del comando specifico per il sistema per richiamare la variabile di ambiente. Il comando utilizzato su UNIX è ″/usr/bin/env″.

KRB5CCNAME è sensibile al maiuscolo/minuscolo ed è tutto in maiuscolo.

Se KRB5CCNAME non è impostato come sopra descritto, viene utilizzato un file di cache predefinito. La cache predefinita viene individuata nel seguente ordine:

1. /tmp/krb5cc_<uid> sulle piattaforme Unix, dove <uid> è l'ID utente dell'utente che esegue la JVM Kinit
2. <user.home>/krb5cc_<user.name>, dove <user.home> e <user.name> sono rispettivamente le proprietà Java user.home e user.name
3. <user.home>/krb5cc (se non è possibile ottenere <user.name> dalla JVM)

Superotempo comunicazioni KDC

Kinit comunica con il KDC (Key Distribution Center) per acquisire un TGT (ticket-granting ticket), ossia delle credenziali. Queste comunicazioni possono essere impostate in modo che vadano in superotempo se il KDC non risponde entro un certo lasso di tempo. Il periodo di superotempo può essere impostato (in millisecondi) nel file di configurazione Kerberos nella stanza libdefaults (per essere applicabile a tutti i KDC) oppure nelle singole stanze KDC. Il valore di superotempo predefinito è 30 secondi.

com.ibm.security.krb5.internal.tools Class Ktab:

Questa classe può essere eseguita come uno strumento di riga comandi per aiutare l'utente a gestire le voci nella tabella chiavi. Le funzioni disponibili includono i tasti di elencazione/aggiunta/ aggiornamento/cancellazione di servizi.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Ktab
```

public class **Ktab**
extends java.lang.Object

Questa classe può essere eseguita come uno strumento di riga comandi per aiutare l'utente a gestire le voci nella tabella chiavi. Le funzioni disponibili includono i tasti di elencazione/aggiunta/ aggiornamento/cancellazione di servizi.

Riepilogo del programma di creazione

`Ktab()`

Riepilogo del metodo

`static void`	`main(java.lang.String[] args)` Il programma principale che può essere richiamato sulla riga comandi.

Metodi ereditati dalla classe java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Dettagli sul programma di creazione

Ktab

```
public Ktab()
```

Dettagli del metodo

main

```
public static void main(java.lang.String[] args)
```

Il programma principale che può essere richiamato sulla riga comandi.

Utilizzo: java com.ibm.security.krb5.tools.Ktab <opzioni>

Opzioni disponibili per Ktab:
- **-l** elencare le voci ed il nome keytab
- **-a** <*nome principal*>(<*parola d'ordine*>) aggiungere una voce al keytab
- **-d** <*nome principal*> cancellare una voce dal keytab
- **-k** <*nome keytab*> specificare il nome ed il percorso del keytab con il prefisso FILE:
- **-help** visualizzare le istruzioni

Stabilimento del contesto JGSS:

Avendo acquisito delle credenziali di sicurezza, i due peer comunicanti stabiliscono un contesto di sicurezza utilizzando le loro credenziali. Anche se i peer stabiliscono un singolo contesto congiunto, ciascun peer mantiene una propria copia locale del contesto. Lo stabilimento del contesto comporta che il peer iniziatore autentichi la propria identità presso il peer accettante. L'iniziatore può richiedere facoltativamente l'autenticazione reciproca, nel qual caso l'accettatore autentica se stesso presso l'iniziatore.

Quando lo stabilimento del contesto è completo, il contesto stabilito include le informazioni sullo stato (come le chiavi crittografiche condivise) che abilitano il successivo scambio di messaggi protetti tra i due peer.

Protezione e scambio di messaggi JGSS:

Dopo aver stabilito un contesto, i due peer sono pronti ad intraprendere lo scambio di messaggi sicuri. L'emittente del messaggio richiama la sua implementazione GSS-API locale per codificare il messaggio, la quale garantisce l'integrità del messaggio e facoltativamente la sua riservatezza. L'applicazione quindi trasporta il token risultante nel peer.

L'implementazione GSS-API locale del peer utilizza le informazioni del contesto stabilito nei seguenti modi:

- Verifica l'integrità del messaggio
- Decodifica il messaggio, se il messaggio è stato codificato

Rilascio e ripulitura di risorse:

Per poter liberare le risorse, un'applicazione JGSS cancella un contesto non più necessario. Sebbene un'applicazione JGSS possa accedere a un contesto cancellato, qualsiasi tentativo di utilizzarlo per lo scambio dei messaggi risulta in un errore.

Meccanismi di sicurezza:

GSS-API è composto da una framework astratta di uno o più meccanismi di sicurezza sottostanti. La modalità con la quale la framework interagisce con i meccanismi di sicurezza sottostanti è specifica dell'implementazione.

Le implementazioni sono composte da due categorie:

- Ad un estremo, un'implementazione monolitica collega fermamente la framework ad un singolo meccanismo. Questo tipo di implementazione preclude l'utilizzo di altri meccanismi o anche implementazioni differenti dello stesso meccanismo.
- All'altro estremo, un'implementazione ampiamente modulare offre una facilità di utilizzo e flessibilità. Questo tipo di implementazione offre la capacità di collegare più agevolmente e direttamente diversi meccanismi di sicurezza e le loro implementazioni nella framework.

IBM JGSS rientra nella seconda categoria. Come implementazione modulare, IBM JGSS livella la framework del fornitore definita da JCA (Java Cryptographic Architecture) e considera ogni meccanismo sottostante come un fornitore (JCA). Un fornitore JGSS fornisce un'implementazione concreta di un meccanismo di sicurezza JGSS. Un'applicazione può creare istanze ed utilizzare più meccanismi.

Un fornitore può supportare più meccanismi e JGSS facilita l'utilizzo di differenti meccanismi di sicurezza. Tuttavia, GSS-API non fornisce un'indicazione, ai due peer comunicanti tra loro, per la scelta riguardo a quale meccanismo utilizzare quando ve ne è più di uno a disposizione. Un modo per scegliere un meccanismo è quello di utilizzare SPNEGO (Simple And Protected GSS-API Negotiating Mechanism), che è uno pseudo meccanismo che negozia il meccanismo reale tra i due peer. IBM JGSS non comprende un meccanismo SPNEGO.

Per ulteriori informazioni relative a SPNEGO, consultare IETF (Internet Engineering Task Force) RFC 2478 The Simple and Protected GSS-API Negotiation Mechanism

Configurazione del server per utilizzare IBM JGSS

La modalità di configurazione del server per l'utilizzo di JGSS dipende da quale versione di J2SE (Java 2 Platform, Standard Edition) viene eseguita sul sistema.

Configurazione di System i5 per utilizzare JGSS con J2SDK, versione 1.3:

Quando si utilizza J2SDK (Java 2 Software Development Kit), versione 1.3 sul server, è necessario preparare e configurare il server per utilizzare JGSS. La configurazione predefinita utilizza il fornitore JGSS Java puro.

Requisiti software

Per utilizzare JGSS con J2SDK, versione 1.3, il server deve disporre di JAAS (Java Authentication and Authorization Service) versione 1.3 installato.

Configurazione del server per utilizzare JGSS

Per configurare il proprio server per l'utilizzo di JGSS con J2SDK, versione 1.3, aggiungere un collegamento simbolico nell'indirizzario di estensione del file ibmjgssprovider.jar. Il file ibmjgssprovider.jar contiene le classi JGSS e il fornitore JGSS Java puro. L'aggiunta di un collegamento simbolico consente al programma di caricamento della classe di estensione di caricare il file ibmjgssprovider.jar.

Aggiungere collegamento simbolico

Per aggiungere il collegamento simbolico, sulla riga comandi i5/OS immettere il seguente comando (su una sola riga) e premere **INVIO**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssprovider.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/ibmjgssprovider.jar')
```

Nota: La politica Java 1.3 predefinita sul server concede le appropriate autorizzazioni a JGSS. Se si intende creare un proprio file java.policy, consultare l'argomento relativo alle autorizzazioni JVM per avere un elenco di autorizzazioni da concedere a ibmjgssprovider.jar.

Modifica dei fornitori JGSS

Dopo aver configurato il server per utilizzare JGSS, che utilizza il fornitore Java puro come quello predefinito, è possibile configurare JGSS per utilizzare il fornitore JGSS System i5 nativo. Quindi, dopo aver configurato JGSS per l'utilizzo del fornitore nativo, è possibile passare da un fornitore all'altro.

Gestori della sicurezza

Se si sta eseguendo l'applicazione JGSS IBM con un gestore della sicurezza Java abilitato, consultare l'argomento Utilizzo di un gestore della sicurezza.

Concetti correlati

"Autorizzazioni JVM" a pagina 387
In aggiunta ai controlli del controllo accesso eseguiti da JGSS, la JVM (Java virtual machine) esegue controlli dell'autorizzazione durante l'accesso a una serie di risorse, inclusi file,proprietà Java pacchetti e socket.

"Fornitore JGSS" a pagina 386
IBM JGSS include un fornitore System i5 JGSS nativo ed un fornitore Java JGSS puro. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

"Utilizzo di un gestore della sicurezza" a pagina 387
Se si sta eseguendo l'applicazione JGSS con il gestore della sicurezza Java abilitato, è necessario accertarsi che l'applicazione e JGSS abbiano le autorizzazioni necessarie.

Configurazione di JGSS per utilizzare il fornitore JGSS System i5 nativo:

IBM JGSS utilizza il fornitore Java puro come predefinito. Si dispone anche dell'opzione di utilizzare il fornitore JGSS System i5 nativo.

Requisiti software

Il fornitore JGSS System i5 nativo deve poter accedere alle classi in IBM Toolbox per Java. Per istruzioni relative a come accedere a IBM Toolbox per Java, consultare il seguente argomento secondario.

Accertarsi di aver configurato il servizio di autenticazione della rete.

Nota: nelle seguenti istruzioni, ${java.home} indica il percorso all'ubicazione della versione di Java che si sta utilizzando sul server. Ad esempio, se si sta utilizzando la versione 1.4 di J2SDK, ${java.home} è /QIBM/ProdData/Java400/jdk14. Ricordarsi di sostituire ${java.home} nei comandi con il percorso reale dell'indirizzario principale Java.

Per configurare JGSS per utilizzare il fornitore JGSS System i5 nativo, completare le seguenti attività:

Aggiungere un collegamento simbolico

Per aggiungere un collegamento simbolico all'indirizzario di estensione per il file ibmjgssiseriesprovider.jar, su una riga comandi i5/OS immettere il seguente comando (su una singola riga) e premere **INVIO**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssiseriesprovider.jar')
NEWLNK('${java.home}/lib/ext/ibmjgssiseriesprovider.jar')
```

Dopo aver aggiunto un collegamento simbolico all'indirizzario di estensione del file ibmjgssiseriesprovider.jar, il programma di caricamento della classe di estensione caricherà il file JAR.

Aggiungere il fornitore nell'elenco dei fornitori della sicurezza

Aggiungere il fornitore nativo all'elenco dei fornitori della sicurezza nel file java.security.

1. Aprire il file ${java.home}/lib/security/java.security per la modifica.
2. Reperire l'elenco dei fornitori della sicurezza. Tale elenco dovrebbe trovarsi nella parte alta del file java.security e apparire approssimativamente come segue:

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=com.sun.rsajca.Provider
   security.provider.3=com.ibm.crypto.provider.IBMJCE
   security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
   ```

3. Aggiungere il fornitore JGSS System i5 nativo all'elenco dei fornitori della sicurezza prima del fornitore Java originale. In altre parole, aggiungere com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider nell'elenco con un numero inferiore rispetto al valore dato in com.ibm.jgss.IBMJGSSProvider, quindi aggiornare la posizione di IBMJGSSProvider. Ad esempio:

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=com.sun.rsajca.Provider
   security.provider.3=com.ibm.crypto.provider.IBMJCE
   security.provider.4=com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider
   security.provider.5=com.ibm.security.jgss.IBMJGSSProvider
   ```

   Notare che IBMJGSSiSeriesProvider diventa la quarta voce nell'elenco e IBMJGSSProvider la quinta voce. Inoltre, verificare che i numeri delle voci nell'elenco dei fornitori della sicurezza siano sequenziali e che il valore di ogni voce venga incrementata di un solo numero alla volta.
4. Salvare e chiudere il file java.security.

Concetti correlati

"Fornitore JGSS" a pagina 386
IBM JGSS include un fornitore System i5 JGSS nativo ed un fornitore Java JGSS puro. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

"Configurazione di System i5 per utilizzare JGSS con J2SDK, versione 1.3" a pagina 383
Quando si utilizza J2SDK (Java 2 Software Development Kit), versione 1.3 sul server, è necessario preparare e configurare il server per utilizzare JGSS. La configurazione predefinita utilizza il fornitore JGSS Java puro.

Informazioni correlate

Servizio di autenticazione di rete

Abilitazione del fornitore JGSS System i5 nativo per accedere a IBM Toolbox per Java:

Il fornitore JGSS System i5 nativo deve poter accedere alle classi in IBM Toolbox per Java.

Utilizzare una delle seguenti opzioni per consentire ad IBM JGSS di accedere al file jt400Native.jar di Toolbox per Java:

Nota: Nelle seguenti istruzioni, *${java.home}* indica il percorso all'ubicazione della versione di Java che si sta utilizzando sul server. Ad esempio, se si sta utilizzando J2SE, versione 1.5, *${java.home}* è /QIBM/ProdData/Java400/jdk15. Ricordarsi di sostituire *${java.home}* nei comandi con il percorso effettivo all'indirizzario principale Java.

Opzione 1: inserire un collegamento simbolico nell'indirizzario di estensione Java

Per inserire un collegamento nel file jt400Native.jar nell'indirizzario di estensione Java, su una riga comandi i5/OS immettere il seguente comando (su una singola riga) e premere **INVIO**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('${java.home}/lib/ext/jt400Native.jar')
```

Nota: Inserendo un collegamento simbolico al file jt400Native.jar nell'indirizzario di estensione Java, tutti gli utenti di J2SE saranno obbligati ad utilizzare tale versione di jt400Native.jar. Ciò potrebbe non essere consigliabile se vari utenti richiedono differenti versioni delle classi di IBM Toolbox per Java.

Opzione 2: inserire un collegamento simbolico nel proprio indirizzario di estensione

Per collegare il file jt400Native.jar al proprio indirizzario, su una riga comandi i5/OS immettere il seguente comando (su una singola riga) e premere **INVIO**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('<indirizzario estensione>/jt400Native.jar')
```

Quando si richiama il programma, utilizzare il seguente formato per il comando Java:

```
java -Djava.ext.dirs=<indirizzario estensione>:${java.home}/lib/ext:/QIBM/UserData/Java400/ext
```

Configurazione di System i5 per utilizzare JGSS con J2SDK, versione 1.4 o una versione successiva:

Quando si utilizza J2SDK (Java 2 Software Development Kit), versione 1.4 o successive sul server, JGSS è già configurato. La configurazione predefinita utilizza il fornitore JGSS Java puro.

Modifica dei fornitori JGSS

È possibile configurare JGSS per utilizzare il fornitore System i5 JGSS nativo invece del fornitore Java JGSS puro. Quindi, dopo aver configurato JGSS per l'utilizzo del fornitore nativo, è possibile passare da un fornitore all'altro. Per ulteriori informazioni, consultare i seguenti argomenti:

- "Fornitore JGSS"
- "Configurazione di JGSS per utilizzare il fornitore JGSS System i5 nativo" a pagina 384

Gestori della sicurezza

Se si sta eseguendo l'applicazione JGSS con il gestore della sicurezza Java abilitato, consultare Utilizzo di un gestore della sicurezza.

Fornitore JGSS:

IBM JGSS include un fornitore System i5 JGSS nativo ed un fornitore Java JGSS puro. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

Il fornitore JGSS Java puro offre le seguenti caratteristiche:

- Assicura la massima trasferibilità della propria applicazione.
- Gestisce le interfacce di collegamento JAAS Kerberos facoltative.
- Offre la compatibilità con gli strumenti di gestione delle credenziali Kerberos Java.

Il fornitore System i5 JGSS nativo offre le seguenti funzioni:
- Utilizza le librerie System i5 Kerberos native.
- Offre la compatibilità con gli strumenti di gestione delle credenziali Kerberos Qshell.
- Una esecuzione più rapida delle applicazioni JGSS.

Nota: entrambi i fornitori JGSS aderiscono alla specifica GSS-API diventando così reciprocamente compatibili. In altre parole, un'applicazione che utilizza il fornitore Java JGSS puro può interoperare con un'applicazione che utilizza il fornitore System i5 JGSS nativo.

Modifica del fornitore JGSS

È possibile modificare facilmente il fornitore JGSS utilizzando una delle seguenti opzioni:
- modifica dell'elenco dei fornitori per la sicurezza in `${java.home}/lib/security/java.security`

  Nota: ${java.home} indica il percorso all'ubicazione della versione di Java che si sta utilizzando sul server. Ad esempio, se si sta utilizzando la versione 1.5 di J2SE, ${java.home} è /QIBM/ProdData/Java400/jdk15.
- Specificare il nome del fornitore nell'applicazione JGSS utilizzando GSSManager.addProviderAtFront() o GSSManager.addProviderAtEnd(). Per ulteriori informazioni consultare GSSManager javadoc.

Utilizzo di un gestore della sicurezza:

Se si sta eseguendo l'applicazione JGSS con il gestore della sicurezza Java abilitato, è necessario accertarsi che l'applicazione e JGSS abbiano le autorizzazioni necessarie.

Autorizzazioni JVM:

In aggiunta ai controlli del controllo accesso eseguiti da JGSS, la JVM (Java virtual machine) esegue controlli dell'autorizzazione durante l'accesso a una serie di risorse, inclusi file,proprietà Java pacchetti e socket.

L'elenco che segue identifica le autorizzazioni richieste quando si utilizzano le funzioni JAAS di JGSS o si utilizza JGSS con un gestore della sicurezza:
- javax.security.auth.AuthPermission ″modifyPrincipals″
- javax.security.auth.AuthPermission ″modifyPrivateCredentials″
- javax.security.auth.AuthPermission ″getSubject″
- javax.security.auth.PrivateCredentialPermission ″javax.security.auth.kerberos.KerberosKey javax.security.auth.kerberos.KerberosPrincipal \″*\″″, ″read″
- javax.security.auth.PrivateCredentialPermission ″javax.security.auth.kerberos.KerberosTicket javax.security.auth.kerberos.KerberosPrincipal \″*\″″, ″read″
- java.util.PropertyPermission ″com.ibm.security.jgss.debug″, ″read″
- java.util.PropertyPermission ″DEBUG″, ″read″
- java.util.PropertyPermission ″java.home″, ″read″
- java.util.PropertyPermission ″java.security.krb5.conf″, ″read″
- java.util.PropertyPermission ″java.security.krb5.kdc″, ″read″
- java.util.PropertyPermission ″java.security.krb5.realm″, ″read″
- java.util.PropertyPermission ″javax.security.auth.useSubjectCredsOnly″, ″read″

- java.util.PropertyPermission ″user.dir″, ″read″
- java.util.PropertyPermission ″user.home″, ″read″
- java.lang.RuntimePermission ″accessClassInPackage.sun.security.action″
- java.security.SecurityPermission ″putProviderProperty.IBMJGSSProvider″

Informazioni correlate

Autorizzazioni in Java 2 SDK di Sun Microsystems, Inc.

Controlli autorizzazioni JAAS:

IBM JGSS effettua i controlli delle autorizzazioni al tempo di esecuzione nel momento in cui il programma abilitato a JAAS utilizza le credenziali ed accede ai servizi. È possibile disabilitare questa funzione JAAS facoltativa impostando la proprietà Java javax.security.auth.useSubjectCredsOnly su false. Inoltre, JGSS effettua i controlli delle autorizzazioni solo quando l'applicazione utilizza un gestore della sicurezza.

JGSS effettua i controlli delle autorizzazioni confrontandole con la politica Java in vigore per il contesto del controllo di accesso corrente. JGSS effettua i seguenti controlli specifici sulle autorizzazioni:

- javax.security.auth.kerberos.DelegationPermission
- javax.security.auth.kerberos.ServicePermission

Controllo DelegationPermission

DelegationPermission consente alla politica di sicurezza di controllare l'uso delle funzioni di Kerberos relative all'inoltro e all'inoltro tramite proxy dei certificati. Utilizzando queste funzioni, un client può consentire ad un servizio di agire al posto del client.

DelegationPermission richiede due argomenti, nel seguente ordine:

1. Il principal subordinato, che è il nome del principal del servizio che agisce al posto del client e con la sua autorizzazione.
2. Il nome del servizio che il client desidera che il principal subordinato utilizzi.

Esempio: utilizzo del controllo DelegationPermission

Nell'esempio che segue, superSecureServer è il principal subordinato e krbtgt/REALM.IBM.COM@REALM.IBM.COM è il servizio che si desidera che superSecureServer utilizzi al posto del client. In questo caso, il servizio è il certificato di concessione certificati per il client, cioè, superSecureServer può ottenere, al posto del client, un certificato per qualsiasi servizio.

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
```

Nell'esempio precedente DelegationPermission concede al client l'autorizzazione per ottenere un nuovo certificato di concessione certificati dal KDC (Key Distribution Center-Centro distribuzione chiavi) che solo superSecureServer potrà utilizzare. Dopo che il client ha inviato il nuovo certificato di concessione certificati a superSecureServer, superSecureServer ha la facoltà di agire al posto del client.

L'esempio che segue abilita il client ad ottenere un nuovo certificato che consente a superSecureServer di accedere, al posto del client, solo al servizio ftp.

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"ftp/ftp.ibm.com@REALM.IBM.COM\"";
```

Controllo ServicePermission

I controlli ServicePermission limitano l'utilizzo delle credenziali per iniziare ed accettare il contesto. L'iniziatore di un contesto deve disporre dell'autorizzazione per iniziare un contesto. Allo stesso modo, l'accettante di un contesto deve disporre dell'autorizzazione per accettare un contesto.

Esempio: utilizzo del controllo ServicePermission

L'esempio che segue consente alla parte client di iniziare un contesto con il servizio ftp tramite la concessione dell'autorizzazione al client:

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "initiate";
```

L'esempio che segue consente alla parte server di accedere ed utilizzare la chiave segreta per il servizio ftp tramite la concessione dell'autorizzazione al server:

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "accept";
```

Informazioni correlate

Documentazione di Sun Microsystems, Inc.

Esecuzione delle applicazioni IBM JGSS

L'API di IBM JGSS (Java Generic Security Service) 1.0 scherma le applicazioni sicure dalla complessità e peculiarità dei diversi meccanismi di sicurezza sottostanti. JGSS utilizza le funzioni fornite da JAAS (Java Authentication and Authorization Service) e IBM JCE (Java Cryptography Extension).

Le funzioni JGSS comprendono:

- L'autenticazione dell'identità
- L'integrità e la riservatezza del messaggio
- Un'interfaccia di collegamento a JAAS Kerberos facoltativa e controlli dell'autorizzazione

Acquisizione delle credenziali kerberos e creazione di chiavi segrete:

GSS-API non definisce una modalità per ottenere le credenziali. Per questo motivo, il meccanismo IBM JGSS Kerberos richiede che l'utente utilizzi uno dei seguenti metodi, per ottenere le credenziali Kerberos: questo argomento fornisce istruzioni su come ottenere le credenziali Kerberos, creare chiavi segrete, utilizzare il JASS per eseguire i collegamenti a Kerberos e i controlli delle autorizzazioni; fornisce un elenco delle autorizzazioni JAAS richieste dalla JVM (Java virtual machine).

È possibile ottenere le credenziali utilizzando uno dei seguenti metodi:

- Strumenti Kinit e Ktab
- Interfaccia di collegamento a JAAS Kerberos facoltativa

Strumenti Kinit e Ktab:

La scelta del fornitore JGSS determina il tipo di strumento da utilizzare per ottenere le credenziali Kerberos e le chiavi segrete.

Utilizzo del fornitore JGSS Java puro

Se si utilizza il fornitore JGSS Java puro, utilizzare gli strumenti IBM JGSS Kinit e Ktab per ottenere le credenziali e le chiavi segrete. Gli strumenti Kinit e Ktab utilizzano le interfacce della riga comandi e forniscono opzioni simili a quelle offerte da altre versioni.

- È possibile ottenere le credenziali Kerberos utilizzando lo strumento Kinit. Questo strumento contatta il KDC (Kerberos Distribution Center-Centro di distribuzione Kerberos) ed ottiene un TGT

(ticket-granting ticket-certificato di concessione certificati). Il TGT permette di accedere ad altri servizi abilitati a Kerberos, compresi quelli che utilizzano GSS-API.

- Un server può ottenere una chiave segreta utilizzando lo strumento Ktab. JGSS memorizza la chiave segreta nel file della tabella di chiavi sul server. Per ulteriori informazioni, consultare la documentazione Java per il Ktab.

In alternativa, l'applicazione può utilizzare l'interfaccia di collegamento a JAAS per ottenere i TGT e le chiavi segrete.

Utilizzo del fornitore System i5 JGSS nativo

Se si sta utilizzando il fornitore System i5 JGSS nativo, utilizzare i programmi di utilità klist e kinit Qshell.

Concetti correlati

"Interfaccia di collegamento Kerberos JAAS"
IBM JGSS fornisce un'interfaccia di collegamento a JAAS (Java Authentication and Authorizaiton Service) Kerberos. È possibile disabilitare questa funzione impostando la proprietà Java javax.security.auth.useSubjectCredsOnly su false.

Riferimenti correlati

"com.ibm.security.krb5.internal.tools Class Kinit" a pagina 379
Strumento Kinit per ottenere i ticket Kerberos v5.

"com.ibm.security.krb5.internal.tools Class Ktab" a pagina 381
Questa classe può essere eseguita come uno strumento di riga comandi per aiutare l'utente a gestire le voci nella tabella chiavi. Le funzioni disponibili includono i tasti di elencazione/aggiunta/aggiornamento/cancellazione di servizi.

Interfaccia di collegamento Kerberos JAAS:

IBM JGSS fornisce un'interfaccia di collegamento a JAAS (Java Authentication and Authorizaiton Service) Kerberos. È possibile disabilitare questa funzione impostando la proprietà Java javax.security.auth.useSubjectCredsOnly su false.

Nota: Mentre il fornitore Java JGSS puro può utilizzare l'interfaccia di collegamento, il fornitore System i5 JGSS nativo non ha questa capacità.

Per ulteriori informazioni su JAAS, consultare JASS (Java Authentication and Authorization Service).

Autorizzazioni a JVM e JAAS

Se si sta utilizzando un gestore della sicurezza, è necessario assicurarsi che l'applicazione e JGSS dispongano delle necessarie autorizzazioni a JVM e JAAS. Per ulteriori informazioni, consultare Utilizzare un gestore sicurezza.

Opzioni del file di configurazione JAAS

L'interfaccia di collegamento richiede un file di configurazione JAAS che specifichi com.ibm.security.auth.module.Krb5LoginModule come modulo di collegamento da utilizzare. La tabella che segue elenca le opzioni supportate da Krb5LoginModule. Tenere presente che le opzioni non sono sensibili al maiuscolo/minuscolo.

Nome opzione	Valore	Valore predefinito	Spiegazione
principal	<string>	Nessuno; viene richiesto all'utente.	Nome principal Kerberos

Nome opzione	Valore	Valore predefinito	Spiegazione
credsType	initiator | acceptor | both	initiator	Tipo di credenziale JGSS
forwardable	true | false	false	Acquisisce o meno un TGT inoltrabile
proxiable	true | false	false	Acquisisce o meno un TGT inoltrabile via proxy
useCcache	<URL>	Non utilizzare ccache	Richiama il TGT dalla cache delle credenziali specificata
useKeytab	<URL>	Non utilizzare tabella chiavi	Richiama la chiave segreta dalla tabella di chiavi specificata
useDefaultCcache	true | false	Non utilizzare ccache predefinita	Richiama il TGT dalla cache delle credenziali predefinita
useDefaultKeytab	true | false	Non utilizzare tabella chiavi predefinita	Richiama la chiave segreta dalla tabella di chiavi specificata

Per un facile esempio sull'utilizzo di Krb5LoginModule, consultare il file di configurazione del collegamento a JAAS di esempio.

Incompatibilità tra opzioni

Alcune opzioni Krb5LoginModule, tranne il nome principal, sono incompatibili tra loro, non è quindi possibile specificarle insieme. La tabella che segue illustra le opzioni del modulo di collegamento compatibili e quelle incompatibili.

Gli indicatori nella tabella descrivono le relazioni tra le due opzioni associate:

- X = Incompatibile
- N/A = Combinazione Non Applicabile
- Spazio = Compatibile

Opzione Krb5LoginModule	credsType initiator	credsType acceptor	credsType both	forward	proxy	use Ccache	use Keytab	useDefault Ccache	useDefault Keytab
credsType=initiator		N/A	N/A				X		X
credsType=acceptor	N/A		N/A	X	X	X		X	
credsType=both	N/A	N/A							
forwardable		X				X	X	X	X
proxiable		X				X	X	X	X
useCcache		X		X	X		X	X	X
useKeytab	X			X	X	X		X	X
useDefaultCcache		X		X	X	X	X		X
useDefaultKeytab	X			X	X	X	X	X	

Opzioni per il nome principal

È possibile specificare un nome principal in combinazione con un'altra opzione. Se non si specifica un nome principal, Krb5LoginModule potrebbe richiederlo all'utente. Krb5LoginModule richiederà o meno all'utente il nome principal a seconda dell'altra opzione specificata.

Formato del nome principal del servizio

È necessario utilizzare uno dei seguenti formati per specificare un nome principal del servizio:

- <nome_servizio> (ad esempio, superSecureServer)
- <nome_servizio>@<host> (ad esempio, superSecureServer@myhost)

Nell'ultimo formato, <host> è il nome host della macchina su cui risiede il servizio. È possibile (ma non è necessario) utilizzare un nome host completo.

Nota: JAAS riconosce alcuni caratteri come delimitatori. Se si utilizza uno dei caratteri che seguono in una stringa JAAS (ad esempio, un nome principal), racchiuderlo tra apici:

```
_  (sottolineatura)
:  (due punti)
/  (barra)
\  (barra retroversa)
```

Richiesta del nome principal e parola d'ordine

Le opzioni che vengono specificate nel file di configurazione JAAS determinano se il collegamento a Krb5LoginModule è interattivo o meno.

- Un collegamento non interattivo non richiede informazioni di alcun tipo
- Un collegamento interattivo richiede il nome principal, la parola d'ordine o entrambi

Collegamenti non interattivi

Il collegamento procede in maniera non interattiva quando si specifica initiator (iniziatore) come tipo di credenziale (`credsType=initiator`) e si intraprende una delle seguenti azioni:

- Si specifica l'opzione useCcache
- Si imposta l'opzione useDefaultCcache su true

Il collegamento procede in maniera non interattiva anche quando si specifica acceptor (accettante) o both (entrambi) come tipo di credenziale (`credsType=acceptor` o `credsType=both`) e si intraprende una delle seguenti azioni:

- Si specifica l'opzione useKeytab
- Si imposta l'opzione useDefaultKeytab su true

Collegamenti interattivi

Se si utilizzano altre configurazioni, il modulo di collegamento richiederà un nome principal e una parola d'ordine per poter ottenere un TGT da un KDC Kerberos. Il modulo di collegamento richiede solo la parola d'ordine se si specifica l'opzione principal.

I collegamenti interattivi richiedono che l'applicazione specifichi com.ibm.security.auth.callback.Krb5CallbackHandler come handler della callback, quando si crea il contesto del collegamento. L'handler della callback è il responsabile della richiesta di immissione.

Opzioni per il tipo di credenziale

Quando si richiede che il tipo di credenziale sia both (entrambi, iniziatore e accettante) (`credsType=both`), Krb5LoginModule ottiene sia un TGT che una chiave segreta. Il modulo di collegamento utilizza il TGT per iniziare i contesti e la chiave segreta per accettarli. È necessario che il file di configurazione JAAS contenga informazioni sufficienti per permettere al modulo di collegamento di acquisire i due tipi di credenziali.

Per i tipi di credenziali acceptor (accettante) e both (entrambi), il modulo di collegamento presume un principal del servizio.

File delle politiche e di configurazione:

JGSS e JAAS dipendono da una serie di file di configurazione e delle politiche. È necessario modificare questi file per renderli conformi al proprio ambiente e alle proprie applicazioni. Se non si utilizza JAAS con JGSS, è possibile ignorare tranquillamente i file di configurazione e delle politiche JAAS.

Nota: nelle seguenti istruzioni, ${java.home} indica il percorso all'ubicazione della versione di Java che si sta utilizzando sul server. Ad esempio, se si sta utilizzando la versione 1.5 di J2SE, ${java.home} è /QIBM/ProdData/Java400/jdk15. Ricordarsi di sostituire ${java.home} nelle impostazioni delle proprietà con il percorso reale dell'indirizzario principaleJava.

File di configurazione Kerberos

IBM JGSS richiede un file di configurazione Kerberos. Il nome predefinito e l'ubicazione del file di configurazione Kerberos varia in base al sistema operativo utilizzato. JGSS utilizza il seguente ordine di ricerca del file di configurazione predefinito:

1. Il file fa riferimento alla proprietà Java java.security.krb5.conf
2. ${java.home}/lib/security/krb5.conf
3. c:\winnt\krb5.ini su piattaforme Microsoft Windows
4. /etc/krb5/krb5.conf su piattaforme Solaris
5. /etc/krb5.conf su altre piattaforme Unix

File di configurazione JAAS

L'utilizzo della funzione di collegamento JAAS richiede un file di configurazione JAAS. È possibile specificare il file di configurazione JAAS impostando una delle seguenti proprietà:

- La proprietà di sistema Java java.security.auth.login.config
- La proprietà della sicurezza login.config.url.<numero intero> nel file ${java.home}/lib/security/java.security

Per ulteriori informazioni, consultare il sito Web di Sun Java JAAS (Java Authentication and Authorization Service) .

File politica JAAS

Quando si utilizza l'implementazione della politica predefinita, JGSS concede le autorizzazioni JAAS alle entità mediante registrazione delle autorizzazioni in un file delle politiche. È possibile specificare il file delle politiche JAAS mediante impostazione di una delle seguenti proprietà:

- La proprietà di sistema Java java.security.policy
- La proprietà della sicurezza policy.url.<numero intero> nel file ${java.home}/lib/security/java.security

Se si sta utilizzando J2SDK, versione 1.4 o un release successivo, è facoltativo specificare un file delle politiche JAAS separato. Il fornitore della politica predefinito in J2SDK, versione 1.4 e precedenti supporta le voci del file delle politiche richieste da JAAS.

Per ulteriori informazioni, consultare il sito Web di Sun Java JASS (Java Authentication and Authorization Service) .

File delle proprietà di sicurezza principale Java

Una JVM (Java virtual machine) utilizza diverse importanti proprietà della sicurezza che è possibile impostare modificando il file delle proprietà della sicurezza principale Java. Questo file, denominato java.security, in genere risiede nell'indirizzario ${java.home}/lib/security sul server.

L'elenco che segue descrive diverse proprietà di sicurezza rilevanti per l'utilizzo di JGSS. Utilizzare le descrizioni come una guida per modificare il file java.security.

Nota: quando possibile, le descrizioni comprendono i valori appropriati necessari per l'esecuzione degli esempi JGSS.

security.provider.<numero intero>: fornitore JGSS che si desidera utilizzare. Registra anche staticamente le classi del fornitore crittografico. IBM JGSS utilizza i servizi crittografici e altri servizi di sicurezza forniti da IBM JCE Provider. Specificare i pacchetti sun.security.provider.Sun e com.ibm.crypto.provider.IBMJCE esattamente come specificato di seguito:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

policy.provider: classe handler politiche di sistema. Ad esempio:

```
policy.provider=sun.security.provider.PolicyFile
```

policy.url.<numero intero>: URL dei file delle politiche. Per utilizzare il file delle politiche di esempio, includere una voce come indicato di seguito:

```
policy.url.1=file:/home/user/jgss/config/java.policy
```

login.configuration.provider: classe handler configurazione collegamento JAAS, ad esempio:

```
login.configuration.provider=com.ibm.security.auth.login.ConfigFile
```

auth.policy.provider: classe handler politiche per il controllo accessi basati sui principal JAAS, ad esempio:

```
auth.policy.provider=com.ibm.security.auth.PolicyFile
```

login.config.url.<numero intero>: URL per file di configurazione collegamento JAAS. Per utilizzare l'esempio del file di configurazione, includere una voce simile a:

```
login.config.url.1=file:/home/user/jgss/config/jaas.conf
```

auth.policy.url.<numero intero>: URL per i file delle politiche JAAS. È possibile includere sia strutture basate su CodeSource che quelle basate sui principal, nel file delle politiche JAAS. Per utilizzare il file delle politiche di esempio, includere una voce come indicato di seguito:

```
auth.policy.url.1=file:/home/user/jgss/config/jaas.policy
```

Tabella chiavi del server e cache delle credenziali

Un principal dell'utente conserva le sue credenziali kerberos in una cache delle credenziali. Un principal del servizio conserva la sua chiave segreta nella tabella delle chiavi. Durante il tempo di esecuzione, IBM JGSS localizza queste cache nei modi di seguito indicati:

Cache delle credenziali utente

Per localizzare la cache delle credenziali utente, JGSS utilizza il seguente ordine:

1. Il file al quale si fa riferimento nella proprietà Java KRB5CCNAME
2. Il file al quale si fa riferimento nella variabile di ambiente KRB5CCNAME
3. /tmp/krb5cc_<uid> in sistemi Unix
4. ${user.home}/krb5cc_${user.name}
5. ${user.home}/krb5cc (if ${user.name} non è reperibile)

Tabella chiavi del server

Per localizzare il file della tabella chiavi del server, JGSS utilizza il seguente ordine:

1. Il valore della proprietà Java KRB5_KTNAME
2. La voce default_keytab_name nella stanza libdefaults del file di configurazione Kerberos

3. ${user.home}/krb5_keytab

Sviluppo delle applicazioni IBM JGSS

Utilizzare JGSS per sviluppare applicazioni sicure. È possibile trovare indicazioni su come generare token di trasporto, creare oggetti JGSS, stabilire il contesto ed altro ancora.

Per sviluppare le applicazioni JGSS, è necessario conoscere bene la specifica GSS-API ad alto livello e la specifica dei collegamenti Java. IBM JGSS 1.0 si basa principalmente su tali specifiche ed è pienamente conforme ad esse. Consultare i seguenti collegamenti per reperire ulteriori informazioni al riguardo.

- RFC 2743: Generic Security Service Application Programming Interface Version 2, Update 1
- RFC 2853: Generic Security Service API Version 2: Java Bindings

Fasi di programmazione dell'applicazione IBM JGSS:

Esistono più fasi necessarie allo sviluppo di un'applicazione JGSS, includendo l'utilizzo dei token di trasporto, la creazione di oggetti JGSS necessari, l'istituzione e la cancellazione del contesto e l'utilizzo dei servizi tramite messaggio.

Le operazioni svolte in un'applicazione JGSS seguono il modello operativo GSS-API (Generic Security Service Application Programming Interface). Per ulteriori informazioni relative ai concetti importanti sulle operazioni JGSS, consultare Concetti su JGSS.

Token di trasporto JGSS

Alcune operazioni JGSS importanti generano token sotto forma di schiere di byte Java. Compete all'applicazione inviare i token da un peer JGSS ad un altro. JGSS non impone in alcun modo l'utilizzo di un protocollo specifico che utilizza per trasportare i token. Le applicazioni possono trasportare i token JGSS insieme di altri dati delle applicazioni (cioè, dati non JGSS). Tuttavia, le operazioni JGSS accettano e utilizzano soltanto token JGSS specifici.

Sequenza delle operazioni in un'applicazione JGSS

Le operazioni JGSS richiedono alcune strutture di programmazione che l'utente deve utilizzare nell'ordine di seguito elencato. Ogni fase è valida sia per l'iniziatore che per l'accettante.

Nota: nelle informazioni vengono inclusi frammenti di codice di esempio che illustrano l'utilizzo delle API JGSS ad alto livello e presumono l'importazione da parte dell'applicazione del pacchetto org.ietf.jgss. Sebbene molte delle API ad alto livello vengono sovraccaricate, nei frammenti vengono visualizzati soltanto i formati più comunemente utilizzati di questi metodi. Utilizzare i metodi API che più si adattano alle proprie esigenze.

Creazione di un GSSManager:

Con la classe astratta GSSManager è possibile creare i seguenti oggetti JGSS.

La classe astratta GSSManager crea quanto segue:

- GSSName
- GSSCredential
- GSSContext

Inoltre GSSManager dispone di metodi per la determinazione dei meccanismi di sicurezza supportati e i tipi di nome e per la specifica dei fornitori JGSS. Utilizzare il metodo statico getInstance GSSManager per creare un'istanza di GSSManager predefinita:

```
GSSManager manager = GSSManager.getInstance();
```

Creazione di un GSSName:

GSSName rappresenta l'identità di un principal GSS-API. GSSName può contenere molte rappresentazioni del principal, una per ogni meccanismo sottostante supportato. Un GSSName che contiene solo una rappresentazione del nome è denominato MN (Mechanism Name-Nome meccanismo).

GSSManager dispone di diversi metodi caricati per la creazione di un GSSName da una stringa o una schiera contigua di byte. I metodi interpretano la stringa o la schiera di byte secondo il tipo di nome specificato. In genere, vengono utilizzati i metodi a schiere di byte GSSName per ricostituire un nome esportato. Il nome esportato è generalmente un nome del meccanismo di tipo GSSName.NT_EXPORT_NAME. Alcuni di questi metodi consentono di specificare un meccanismo di sicurezza con il quale creare il nome.

Esempio: utilizzo di GSSName

Il seguente frammenti di codice di base consente di visualizzare le modalità di utilizzo di GSSName.

Nota: specificare le stringhe del nome del servizio Kerberos come <service> o <service@host> dove <service> è il nome del servizio e <host> è il nome host della macchina sulla quale si esegue il servizio. È possibile (ma non è necessario) utilizzare un nome host completo. Quando si omette la parte @<host> della stringa, GSSName utilizza il nome host locale.

```
// Creare GSSName per l'utente foo.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);

// Creare il nome del meccanismo Kerberos V5 per l'utente foo.
Oid krb5Mech = new Oid("1.2.840.113554.1.2.2");
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME, krb5Mech);

// Creare un nome meccanismo da un nome non meccanismo mediante utilizzo del
// metodo canonicalize GSSName.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);
GSSName fooKrb5Name = fooName.canonicalize(krb5Mech);
```

Creazione di un GSSCredential:

GSSCredential contiene tutte le informazioni crittografiche necessarie per creare un contesto per conto di un principal e può contenere le informazioni relative alla credenziale più meccanismi.

GSSManager dispone di tre metodi di creazione delle credenziali. Due di questi metodi hanno come parametri un GSSName, la durata della credenziale, uno o più meccanismi dal quale reperire le credenziali e il tipo di utilizzo delle stesse. Il terzo metodo dispone soltanto di un tipo di utilizzo ed utilizza i valori predefiniti per altri parametri. Anche se si specifica un valore null per il meccanismo verrà utilizzato il meccanismo predefinito. La specifica di un valore null per la schiera di meccanismi provoca la restituzione da parte del metodo delle credenziali per il gruppo predefinito di meccanismi.

Nota: IBM JGSS supporta solo il meccanismo Kerberos V5, che è quello predefinito.

L'applicazione può creare soltanto uno dei tre tipi di credenziali (*initiate, accept*, o *initiate and accept*) alla volta.

- L'iniziatore del contesto crea le credenziali *initiate*
- L'accettante crea le credenziali di tipo *accept*
- Un accettante che è anche iniziatore crea credenziali *initiate and accept*.

Esempio: acquisizione di credenziali

Nel seguente esempio viene descritto come ottenere le credenziali predefinite di un iniziatore:

```
GSSCredentials fooCreds = manager.createCredentials(GSSCredential.INITIATE)
```

Nel seguente esempio viene descritto come ottenere le credenziali Kerberos V5 di un iniziatore *foo* che dispone di un periodo di validità predefinito:

```
GSSCredential fooCreds = manager.createCredential(fooName, GSSCredential.DEFAULT_LIFETIME,
                                                  krb5Mech,GSSCredential.INITIATE);
```

Nel seguente esempio viene descritto come ottenere tutte le credenziali dell'accettante predefinite:

```
GSSCredential serverCreds = manager.createCredential(null, GSSCredential.DEFAULT_LIFETIME,
                                                     (Oid)null, GSSCredential.ACCEPT);
```

Creazione di GSSContext:

IBM JGSS supporta due metodi forniti da GSSManager per la creazione di un contesto. Questi metodi sono un metodo utilizzato dall'iniziatore del contesto ed un metodo utilizzato dall'accettante.

Nota: GSSManager fornisce un terzo metodo per la creazione di un contesto che implica una nuova creazione di contesti precedentemente esportati. Tuttavia, poiché il meccanismo IBM JGSS Kerberos V5 non supporta l'utilizzo dei contesti esportati, IBM JGSS non supporta questo metodo.

L'applicazione non può utilizzare un contesto iniziatore per l'accettazione del contesto, né utilizzare un contesto accettante come iniziatore del contesto. Entrambi i metodi supportati per la creazione di un contesto richiedono l'immissione di una credenziale. Quando il valore della credenziale è null, JGSS utilizza la credenziale predefinita.

Esempi: utilizzo di GSSContext

Il seguente esempio crea un contesto con il quale il principal (foo) può iniziare un contesto con il peer (superSecureServer) sull'host (securityCentral). L'esempio specifica il peer come superSecureServer@securityCentral. Il contesto creato è valido per il periodo predefinito:

```
GSSName serverName = manager.createName("superSecureServer@securityCentral",
                                        GSSName.NT_HOSTBASED_SERVICE, krb5Mech);
GSSContext fooContext = manager.createContext(serverName, krb5Mech, fooCreds,
                                              GSSCredential.DEFAULT_LIFETIME);
```

Il seguente esempio crea un contesto per superSecureServer al fine di accettare i contesti iniziali da qualsiasi peer:

```
GSSContext serverAcceptorContext = manager.createContext(serverCreds);
```

Notare che l'applicazione può creare e utilizzare simultaneamente entrambi i tipi di contesto.

Richiesta di servizi di sicurezza JGSS facoltativi:

La propria applicazione può richiedere una serie di servizi di sicurezza facoltativi. IBM JGSS supporta più servizi.

I servizi facoltativi supportati sono:

- Delega
- Reciproca autenticazione
- Ripetizione rilevamento
- Rilevamento fuori sequenza
- Riservatezza disponibile per messaggio
- Integrità disponibile per messaggio

Per richiedere un servizio facoltativo, la propria applicazione deve richiederlo esplicitamente, utilizzando il metodo di richiesta appropriato sul contesto. Solo un iniziatore può richiedere questi servizi facoltativi. Tale iniziatore deve farne richiesta prima di iniziare a stabilire il contesto.

Per ulteriori informazioni sui servizi facoltativi, consultare Optional Service Support in Internet Engineering Task Force (IETF) RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1 .

Esempio: richiesta di servizi facoltativi

Nel seguente esempio, un contesto (fooContext) effettua delle richieste di abilitazione dei servizi di delega e di reciproca autenticazione:

```
fooContext.requestMutualAuth(true);
fooContext.requestCredDeleg(true);
```

Stabilimento del contesto JGSS:

I due peer comunicanti devono stabilire un contesto di sicurezza entro il quale essi potranno utilizzare i servizi tramite messaggio.

L'iniziatore richiama initSecContext() sul suo contesto, il quale restituisce un token all'applicazione iniziatore. L'applicazione iniziatore trasporta il token del contesto nell'applicazione accettante. L'accettante richiama acceptSecContext() sul suo contesto, specificando il token del contesto ricevuto dall'iniziatore. In base a quale meccanismo sottostante e ai servizi facoltativi selezionati dall'iniziatore, acceptSecContext() potrebbe produrre un token che l'applicazione accettante deve inviare all'applicazione iniziatore. Quest'ultima utilizza quindi il token ricevuto per richiamare initSecContext() ancora una volta.

Un'applicazione può effettuare più chiamate a GSSContext.initSecContext() e GSSContext.acceptSecContext(). Un'applicazione può inoltre scambiare più token con un peer durante la fase in cui si stabilisce un contesto. Quindi, il metodo tipico per stabilire un contesto consiste nell'utilizzare un loop per chiamare GSSContext.initSecContext() o GSSContext.acceptSecContext() fino a che le applicazioni stabiliscono il contesto.

Esempio: come stabilire il contesto

Il seguente esempio illustra come viene stabilito un contesto dal lato dell'iniziatore (foo):

```
byte array[] inToken = null; // Il token di immissione è null per la prima chiamata
int inTokenLen = 0;

do {
    byte[] outToken = fooContext.initSecContext(inToken, 0, inTokenLen);

    if (outToken != null) {
        send(outToken); // trasportare token all'accettante
    }

    if( !fooContext.isEstablished()) {
        inToken = receive(); // ricevere token dall'accettante
          inTokenLen = inToken.length;
    }
} while (!fooContext.isEstablished());
```

Il seguente esempio illustra come viene stabilito un contesto dal lato dell'accettante:

```
// Il codice dell'accettante per stabilire il contesto può essere il seguente:
do {
    byte[] inToken = receive(); // ricevere token dall'iniziatore
    byte[] outToken =
        serverAcceptorContext.acceptSecContext(inToken, 0, inToken.length);

    if (outToken != null) {
        send(outToken); // trasportare token all'iniziatore
    }
} while (!serverAcceptorContext.isEstablished());
```

Utilizzo di servizi per messaggi JGSS:

Dopo aver stabilito il contesto di sicurezza i due peer comunicanti, possono scambiare i messaggi sicuri nel contesto stabilito.

Durante la fase in cui si stabilisce un contesto, ognuno dei due peer, sia iniziatore che accettante, può originare un messaggio sicuro. Per rendere sicuro un messaggio, IBM JGSS elabora un MIC (message integrity code) codificato per il messaggio. Facoltativamente, IBM JGSS può disporre del meccanismo Kerberos V5 che codifica il messaggio al fine di garantire la privacy.

Invio di messaggi

IBM JGSS fornisce due gruppi di metodi per rendere sicuri i messaggi: wrap() e getMIC(). g

Utilizzo del wrap()

Il metodo wrap effettua le seguenti operazioni:
- Elabora un MIC
- Codifica il messaggio (facoltativo)
- Restituisce un token

L'applicazione chiamante utilizza la classe MessageProp assieme a GSSContext per specificare se applicare la codifica al messaggio.

Il token restituito contiene sia MIC che il testo del messaggio. Il testo del messaggio è o un testo cifrato (per un messaggio codificato) o un testo in chiaro (per i messaggi non codificati).

Utilizzo del getMIC()

Il metodo getMIC effettua le seguenti operazioni ma non è in grado di codificare il messaggio:
- Elabora un MIC
- Restituisce un token

Il token restituito contiene soltanto il MIC elaborato e non comprende il messaggio originale. Quindi oltre a trasportare il token MIC al peer, quest'ultimo deve in qualche modo essere al corrente del messaggio originale al fine di poter verificare il MIC.

Esempio: utilizzo di servizi tramite messaggio per inviare un messaggio

Il seguente esempio consente di visualizzare la modalità in cui un peer (foo) può codificare un messaggio per la consegna ad un altro peer (superSecureServer):

```
byte[] message = "Ready to roll!".getBytes();
MessageProp mprop = new MessageProp(true); // foo richiede il messaggio codificato
byte[] wrappedMessage =
    fooContext.wrap(message, 0, message.length, mprop);
send(wrappedMessage); // trasferire il messaggio codificato a superSecureServer

// Questo è il modo in cui superSecureServer può ottenere un MIC da consegnare a foo:
byte[] message = "You bet!".getBytes();
MessageProp mprop = null;  // superSecureServer è soddisfatto della
                           // qualità predefinita di protezione

byte[] mic =
    serverAcceptorContext.getMIC(message, 0, message.length, mprop);
send(mic);
// inviare MIC a foo. foo necessita inoltre del messaggio originale per verificare MIC
```

Ricezione di messaggi

Il destinatario del messaggio codificato utilizza il metodo unwrap() per decodificare il messaggio. Il metodo unwrap esegue le seguenti operazioni:

- Verifica il MIC crittografico incorporato nel messaggio
- Restituisce il messaggio originale sul quale il mittente ha elaborato il MIC

Se il mittente ha codificato il messaggio, il metodo unwrap() decodifica il messaggio prima di verificare il MIC e quindi restituisce il messaggio in testo in chiaro originale. Il destinatario di un token MIC utilizza verifyMIC() per verificare il MIC su un messaggio fornito.

Le applicazioni peer utilizzano il loro protocollo per consegnare reciprocamente il contesto JGSS e i token messaggi reciproci. Le applicazioni peer inoltre necessitano della definizione di un protocollo per determinare se il token sia un MIC o un messaggio codificato. Ad esempio, parte di questo protocollo può essere tanto semplice (e rigido) quanto quello utilizzato dalle applicazioni SASL (Simple Authentication and Security Layer). Il protocollo SASL specifica che l'accettante del contesto sia sempre il primo peer ad inviare un token (codificato) tramite messaggio dopo aver stabilito il contesto.

Per ulteriori informazioni, consultare SASL (Simple Authentication and Security Layer).

Esempio: utilizzo di servizi tramite messaggio per ricevere un messaggio

I seguenti esempi consentono di visualizzare il modo in cui un peer (superSecureServer) decodifica un token codificato che esso riceve da un altro peer (foo):

```
MessageProp mprop = new MessageProp(false);

byte[] plaintextFromFoo =
    serverAcceptorContext.unwrap(wrappedTokenFromFoo, 0,
                                 wrappedTokenFromFoo.length, mprop);

// superSecureServer può ora esaminare mprop per determinare le proprietà del messaggio
// (ad es. se il messaggio è stato codificato) applicate da foo.

// foo verifica il MIC ricevuto da superSecureServer:

MessageProp mprop = new MessageProp(false);
fooContext.verifyMIC(micFromFoo, 0, micFromFoo.length, messageFromFoo, 0,
                     messageFromFoo.length, mprop);

// foo può ora esaminare mprop per determinare le proprietà dei messaggi applicate da

// superSecureServer. In particolare, può asserire che il messaggio non è stato

// codificato dal momento che getMIC non codifica i messaggi.
```

Cancellazione del contesto JGSS:

Un peer cancella un contesto quando quest'ultimo non è più necessario. Nelle operazioni JGSS, ogni peer decide unilateralmente quando cancellare un contesto e di ciò non ne deve informare il suo peer.

JGSS non definisce un token di contesto di cancellazione. Per cancellare un contesto, il peer richiama il metodo dispose dell'oggetto GSSContext per liberare qualsiasi risorsa utilizzata dal contesto. Un oggetto GSSContext eliminato è ancora accessibile, a meno che l'applicazione non imposti l'oggetto su null. Tuttavia, qualsiasi tentativo di utilizzare un contesto eliminato (ma ancora accessibile) restituisce una eccezione.

Utilizzo di JAAS con l'applicazione JGSS:

IBM JGSS include una funzione di collegamento JAAS facoltativa che consente all'applicazione di utilizzare JAAS per ottenere le credenziali. Dopo che tale funzione salva le credenziali principal e le chiavi segrete nell'oggetto soggetto del contesto di collegamento JAAS, JGSS può richiamare le credenziali da tale oggetto.

Il comportamento predefinito di JGSS è quello di richiamare le credenziali e le chiavi segrete dal soggetto. È possibile disabilitare questa funzione impostando la proprietà Java javax.security.auth.useSubjectCredsOnly su false.

Nota: Mentre il fornitore Java JGSS puro può utilizzare l'interfaccia di collegamento, il fornitore System i5 JGSS nativo non ha questa capacità.

Per ulteriori informazioni sulle funzioni JAAS, consultare "Acquisizione delle credenziali kerberos e creazione di chiavi segrete" a pagina 389.

Per utilizzare la funzione di collegamento JAAS, l'applicazione deve seguire il modello di programmazione JAAS nei modi descritti di seguito:

- Creare un contesto di collegamento JAAS
- Operare entro i confini della struttura Subject.doAs di JAAS

Il seguente frammento di codice illustra il concetto di operare entro i confini della struttura Subject.doAs di JAAS:

```
static class JGSSOperations implements PrivilegedExceptionAction {
    public JGSSOperations() {}
    public Object run () throws GSSException {
        // Il codice dell'applicazione JGSS viene inserito/eseguito qui
    }
}

public static void main(String args[]) throws Exception {
    // Creare un contesto collegamento che verrà utilizzato dall'handler
    // di callback Kerberos
    // com.ibm.security.auth.callback.Krb5CallbackHandler

    // Deve esistere una configurazione JAAS per "JGSSClient"
    LoginContext loginContext =
        new LoginContext("JGSSClient", new Krb5CallabackHandler());
        loginContext.login();

    // Eseguire l'intera applicazione JGSS nella modalità privilegiata JAAS
    Subject.doAsPrivileged(loginContext.getSubject(),
                           new JGSSOperations(), null);
}
```

Debug di JGSS

Quando si tenta di identificare i problemi JGSS, utilizzare la capacità della funzione di debug di JGSS per produrre utili messaggi categorizzati.

È possibile attivare una o più categorie impostando i valori appropriati della proprietàJava com.ibm.security.jgss.debug. È possibile attivare più categorie utilizzando una virgola per separare i nomi delle categorie.

Le categorie di debug sono le seguenti:

Categoria	Descrizione
help	Elenca le categorie di debug
all	Attiva l'esecuzione del debug di tutte le categorie
off	Disattiva completamente l'esecuzione del debug

Categoria	Descrizione
app	Esegue il debug dell'applicazione (predefinita)
ctx	Esegue il debug delle operazioni di contesto
cred	Operazioni (compresi i nomi) delle credenziali
marsh	Esegue il marshal dei token
mic	Operazioni MIC
prov	Operazioni fornitore
qop	Operazioni QOP
unmarsh	Disabilita l'esecuzione marshal dei token
unwrap	Operazioni unwrap
wrap	Operazioni wrap

Classe di debug JGSS

Per eseguire in modo programmato un'applicazione JGSS, utilizzare la classe debug nella framework di IBM JGSS. L'applicazione può utilizzare la classe debug per attivare e disattivare le categorie di debug e visualizzare le informazioni di debug delle categorie attive.

Il programma di creazione di debug predefinito legge la proprietà Java com.ibm.security.jgss.debug per determinare quale categoria attivare.

Esempio: categoria di debug delle applicazioni

Il seguente esempio consente di visualizzare la modalità di richiesta delle informazioni di debug per la categoria applicazione (app):

```
import com.ibm.security.jgss.debug;

Debug debug = new Debug(); // richiama le categorie dalla proprietà Java

// Per impostare someBuffer, è richiesto molto lavoro. Verificare che la
// categoria sia attiva prima di impostare il debug.

if (debug.on(Debug.OPTS_CAT_APPLICATION)) {
    // Inserire dati in someBuffer.
 debug.out(Debug.OPTS_CAT_APPLICATION, someBuffer);
    // someBuffer potrebbe essere una schiera di byte o una stringa.
```

Esempi: IBM JGSS (Java Generic Security Service)

I file di esempio IBM JGSS (Java Generic Security Service) includono programmi client e server, file di configurazione, file di politiche e informazioni di riferimento Javadoc. Utilizzare i programmi di esempio per effettuare controlli e verifiche della configurazione di JGSS.

È possibile visualizzare le versioni HTML degli esempi o scaricare le informazioni Javadoc ed il codice sorgente per i programmi di esempio. Scaricando gli esempi è possibile visualizzare le informazioni di riferimento Javadoc, esaminare il codice, modificare i file di configurazione e delle politiche e compilare ed eseguire i programmi di esempio.

Descrizione dei programmi di esempio

Gli esempi JGSS includono quattro programmi:

- server non-JAAS
- client non-JAAS

- server abilitato a JAAS
- client abilitato a JAAS

Le versioni abilitate a JAAS sono completamente interfacciabili con le relative controparti non-JAAS. È possibile, quindi, eseguire un client abilitato a JAAS su un server non-JAAS oppure un client non-JAAS su un server abilitato a JAAS.

Nota: quando si esegue un esempio, è possibile specificare una o più proprietà Java facoltative, così come i nomi dei file di configurazione e delle politiche, le opzioni di debug di JGSS e il gestore della sicurezza. È anche possibile attivare e disattivare le funzioni JAAS.

È possibile eseguire gli esempi, in una configurazione ad un server o a due server. La configurazione ad un server consiste nel disporre di un client che comunica con un server primario. La configurazione a due server consiste nel disporre di un server primario ed uno secondario, il server primario agisce come iniziatore, o client, per il server secondario.

Quando si utilizza una configurazione a due server, è il client ad iniziare per primo un contesto e a scambiare messaggi protetti con il server primario. Successivamente, il client delega le proprie credenziali al server primario. Ora è il server primario che utilizza, al posto del client, queste credenziali per iniziare un contesto e scambiare messaggi con il server secondario. È anche possibile utilizzare una configurazione a due server in cui il server primario agisca come un client. In tal caso, il server primario utilizza le proprie credenziali per iniziare un contesto e scambiare messaggi protetti con il server secondario.

Può essere in esecuzione simultaneamente sul server primario, un numero illimitato di client. Sebbene sia possibile avere un client in esecuzione direttamente sul server secondario, quest'ultimo non può utilizzare le credenziali delegate o essere in esecuzione come iniziatore utilizzando le proprie credenziali.

Visualizzazione degli esempi di IBM JGSS:

I file di esempio IBM JGSS (Java Generic Security Service) includono programmi client e server, file di configurazione, file di politiche e informazioni di riferimento Javadoc. Utilizzare i collegamenti che seguono per visualizzare le versioni HTML degli esempi JGSS.

Concetti correlati

"Esempi: IBM JGSS (Java Generic Security Service)" a pagina 402
I file di esempio IBM JGSS (Java Generic Security Service) includono programmi client e server, file di configurazione, file di politiche e informazioni di riferimento Javadoc. Utilizzare i programmi di esempio per effettuare controlli e verifiche della configurazione di JGSS.

Attività correlate

"Esempi: scaricamento ed esecuzione di programmi JGSS di esempio" a pagina 408
Questo argomento contiene istruzioni per scaricare ed eseguire le informazioni di esempio Javadoc.

Riferimenti correlati

"Esempio: programma client IBM JGSS non JAAS" a pagina 524
Utilizzare questo client di esempio JGSS insieme al server di esempio JGSS.

"Esempio: programma server IBM JGSS non JAAS" a pagina 532
Quest'esempio contiene un server di esempio JGSS utilizzato insieme ad un client di esempio JGSS.

"Esempio: programma client IBM JGSS abilitato a JAAS" a pagina 543
Questo programma di esempio effettua un collegamento a JAAS e opera all'interno del contesto di collegamento a JAAS. Non imposta la variabile javax.security.auth.useSubjectCredsOnly, lasciando che la variabile assuma il valore predefinito di "true" in modo che GSS Java acquisisca le credenziali dal Soggetto JAAS associato al contesto di accesso creato dal client.

"Esempio: programma server IBM JGSS abilitato" a pagina 545
Questo programma di esempio effettua un collegamento a JAAS e opera all'interno del contesto di collegamento a JAAS.

Esempio: file di configurazione Kerberos:

Quest'argomento contiene il file di configurazione Kerberos per l'esecuzione delle applicazioni di esempio JGSS.

Per ulteriori informazioni sull'utilizzo del file di configurazione di esempio, consultare Scaricamento ed esecuzione degli esempi IBM JGSS.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
# --------------------------------------------------------------------------------
# File di configurazione Kerberos per l'esecuzione delle applicazioni di esempio JGSS.
# Modificare le voci per adattarle al proprio ambiente.
#--------------------------------------------------------------------------------

[libdefaults]
 default_keytab_name   = /QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab
 default_realm         = REALM.IBM.COM
 default_tkt_enctypes  = des-cbc-crc
 default_tgs_enctypes  = des-cbc-crc
 default_checksum      = rsa-md5
 kdc_timesync          = 0
 kdc_default_options   = 0x40000010
 clockskew             = 300
 check_delegate        = 1
 ccache_type           = 3
 kdc_timeout           = 60000


[realms]
 REALM.IBM.COM = {
         kdc = kdc.ibm.com:88
 }

[domain_realm]
 .ibm.com = REALM.IBM.COM
```

Esempio: file di configurazione collegamento JAAS:

Quest'argomento contiene la configurazione di collegamento JAAS per gli esempi JGSS.

Per ulteriori informazioni sull'utilizzo del file di configurazione di esempio, consultare Scaricamento ed esecuzione degli esempi IBM JGSS.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/**
 * --------------------------------------------------------------------------------
 * Configurazione di collegamento JAAS per esempi di JGSS.
 * --------------------------------------------------------------------------------
 *
 * Esonero di responsabilità per gli esempi di codice
 * IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione
 * da cui creare funzioni simili personalizzate, in base a
 * richieste specifiche.
 * Tutto il codice di esempio viene fornito da IBM solo a scopo illustrativo.
 * Questi esempi non sono stati interamente testati in tutte le condizioni.
 * IBM, quindi, non fornisce nessun tipo di garanzia o affidabilità
 * implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
 * Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO" senza
 * garanzie di alcun tipo.
 * Non viene riconosciuta alcuna garanzia implicita di non contraffazione, commerciabilità e
 * e adeguatezza.
```

```
 *
 *
 * Opzioni supportate:
 *     principal=<string>
 *     credsType=initiator|acceptor|both (default=initiator)
 *     forwardable=true|false (default=false)
 *     proxiable=true|false (default=false)
 *     useCcache=<URL_string>
 *     useKeytab=<URL_string>
 *     useDefaultCcache=true|false (default=false)
 *     useDefaultKeytab=true|false (default=false)
 *     noAddress=true|false (default=false)
 *
 * Dominio predefinito (ottenuto da un file di configurazione Kerberos) viene
 * utilizzato se il principal specificato non comprende un componente del dominio.
 */

JAASClient {
 com.ibm.security.auth.module.Krb5LoginModule required
  useDefaultCcache=true;
};

JAASServer {
 com.ibm.security.auth.module.Krb5LoginModule required
  credsType=acceptor useDefaultKeytab=true
  principal=gss_service/myhost.ibm.com@REALM.IBM.COM;
};
```

Esempio: file delle politiche JAAS:

Quest'argomento contiene il file delle politiche JAAS per l'esecuzione delle applicazioni di esempio JGSS.

Per ulteriori informazioni sull'utilizzo del file delle politiche di esempio, consultare Scaricamento ed esecuzione degli esempi IBM JGSS.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// --------------------------------------------------------------------------
// File delle politiche JAAS per l'esecuzione delle applicazioni di esempio di JGSS.
// Modificare queste autorizzazioni per adattarle al proprio ambiente.
// Non si consiglia di utilizzarlo per scopi diversi da quello sopra indicato.
// In particolare, non utilizzare questo file di politiche o il suo
// contenuto per proteggere risorse in un ambiente di produzione.
//
// Esonero di responsabilità per gli esempi di codice
// IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice
// di programmazione da cui creare una funzione simile personalizzata in base alle proprie
// esigenze specifiche.
// Tutto il codice di esempio viene fornito da IBM solo a scopo illustrativo.
// Questi esempi non sono stati interamente testati in tutte le condizioni.
// IBM, quindi, non fornisce alcuna garanzia sull'affidabilità e la funzionalità
// di questi programmi.
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO" senza
// garanzie di alcun tipo.
// Non viene riconosciuta alcuna garanzia implicita di non contraffazione, commerciabilità e
// adeguatezza a scopi specifici.
//
// --------------------------------------------------------------------------

//--------------------------------------------------------------------------
// Autorizzazioni solo per il client
//--------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
```

```
        "bob@REALM.IBM.COM"
{
    // foo deve poter iniziare un contesto con il server
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "initiate";

    // In questo modo foo può delegare le sue credenziali al server
    permission javax.security.auth.kerberos.DelegationPermission
        "\"gss_service/myhost.ibm.com@REALM.IBM.COM\" \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
};

//--------------------------------------------------------------------------
// Autorizzazioni solo per il server
//--------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service/myhost.ibm.com@REALM.IBM.COM"
{
    // Autorizzazione per il server ad accettare collegamenti di rete sul proprio host
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // Autorizzazione per il server ad accettare i contesti JGSS
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "accept";

    // Il server agisce come un client quando comunica con il server secondario (di riserva)
    // Questa autorizzazione permette al server di iniziare un contesto con il server secondario
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "initiate";
};

//--------------------------------------------------------------------------
// Autorizzazioni per il server secondario
//--------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service2/myhost.ibm.com@REALM.IBM.COM"
{
    // Autorizzazione per il server secondario ad accettare collegamenti di rete sul proprio host
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // Autorizzazione per il server ad accettare i contesti JGSS
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "accept";
};
```

Esempio: file delle politiche Java:

Quest'argomento contiene il file delle politiche Java per l'esecuzione delle applicazioni di esempio JGSS sul server.

Per ulteriori informazioni sull'utilizzo del file delle politiche di esempio, consultare Scaricamento ed esecuzione degli esempi IBM JGSS.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// --------------------------------------------------------------
// File delle politiche Java per l'esecuzione delle applicazioni di esempio di JGSS sul
// server.
// Modificare queste autorizzazioni per adattarle al proprio ambiente.
// Non si consiglia di utilizzarlo per scopi diversi da quello sopra indicato.
// In particolare, non utilizzare questo file di politiche o il suo
// contenuto per proteggere risorse in un ambiente di produzione.
```

```
//
// Esonero di responsabilità per gli esempi di codice
// IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice
// di programmazione da cui creare una funzione simile personalizzata in base alle proprie
// esigenze specifiche.
// Tutto il codice di esempio viene fornito da IBM solo a scopo illustrativo.
// Questi esempi non sono stati interamente testati in tutte le condizioni.
// IBM, quindi, non fornisce alcuna garanzia sull'affidabilità e la funzionalità
// di questi programmi.
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO" senza
// garanzie di alcun tipo.
// Non viene riconosciuta alcuna garanzia implicita di non contraffazione, commerciabilità e
// adeguatezza a scopi specifici.
//
//-------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar" {

    // Per Java 1.4
    permission javax.security.auth.AuthPermission "createLoginContext.JAASClient";
    permission javax.security.auth.AuthPermission "createLoginContext.JAASServer";

    permission javax.security.auth.AuthPermission "doAsPrivileged";

    // Autorizzazione per richiedere un certificato dal KDC
    permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/REALM.IBM.COM@REALM.IBM.COM", "initiate";

    // Autorizzazione per accedere alle classi sun.security.action
    permission java.lang.RuntimePermission "accessClassInPackage.sun.security.action";

    // È possibile accedere ad un ampio gruppo di proprietà Java
    permission java.util.PropertyPermission "java.net.preferIPv4Stack", "read";
    permission java.util.PropertyPermission "java.version", "read";
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.util.PropertyPermission "DEBUG", "read";
    permission java.util.PropertyPermission "com.ibm.security.jgss.debug", "read";
    permission java.util.PropertyPermission "java.security.krb5.kdc", "read";
    permission java.util.PropertyPermission "java.security.krb5.realm", "read";
    permission java.util.PropertyPermission "java.security.krb5.conf", "read";
    permission java.util.PropertyPermission "javax.security.auth.useSubjectCredsOnly",
    "read,write";

    // Autorizzazione per comunicare con l'host del KDC Kerberos
    permission java.net.SocketPermission "kdc.ibm.com", "connect,accept,resolve";

    // Gli esempi vengono eseguiti dall'host locale
    permission java.net.SocketPermission "myhost.ibm.com", "accept,connect,resolve";
    permission java.net.SocketPermission "localhost", "listen,accept,connect,resolve";

    // Accedere ad alcune possibili ubicazioni di configurazione Kerberos
    // Modificare i percorsi file per adattarli al proprio ambiente
    permission java.io.FilePermission "${user.home}/krb5.ini", "read";
    permission java.io.FilePermission "${java.home}/lib/security/krb5.conf", "read";

    // Accedere alla tabella di chiavi Kerberos in modo da poter ottenere la chiave del server.
    permission java.io.FilePermission
    "/QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab", "read";

    // Accedere alla cache delle credenziali Kerberos dell'utente.
    permission java.io.FilePermission "${user.home}/krb5cc_${user.name}",
    "read";
};
```

Esempi: scaricamento e visualizzazione delle informazioni Javadoc per gli esempi di IBM JGSS:

Per scaricare e visualizzare la documentazione dei programmi di esempio di IBM JGSS, completare le seguenti fasi.

1. Scegliere un'indirizzario esistente (o crearne uno nuovo) dove si desidera memorizzare le informazioni Javadoc.
2. Scaricare le informazioni Javadoc (jgsssampledoc.zip) nell'indirizzario.
3. Estrarre i file da jgsssampledoc.zip nell'indirizzario.
4. Utilizzare il browser per accedere al file index.htm.

Esempi: scaricamento ed esecuzione di programmi JGSS di esempio:

Questo argomento contiene istruzioni per scaricare ed eseguire le informazioni di esempio Javadoc.

Prima di modificare o eseguire gli esempi, leggere la descrizione dei programmi di esempio nell'articolo Esempi: IBM JGSS (Java Generic Security Service).

Per eseguire i programmi di esempio, effettuare quanto segue:

1. Scaricare i file di esempio sul server
2. Preparare l'esecuzione dei file di esempio
3. Eseguire i programmi di esempio

Concetti correlati

"Esempi: IBM JGSS (Java Generic Security Service)" a pagina 402
I file di esempio IBM JGSS (Java Generic Security Service) includono programmi client e server, file di configurazione, file di politiche e informazioni di riferimento Javadoc. Utilizzare i programmi di esempio per effettuare controlli e verifiche della configurazione di JGSS.

Attività correlate

"Esempi: scaricamento ed esecuzione di programmi JGSS di esempio"
Questo argomento contiene istruzioni per scaricare ed eseguire le informazioni di esempio Javadoc.

Esempi: scaricamento degli esempi IBM JGSS:

Quest'argomento contiene le istruzioni per scaricare le informazioni Javadoc JGSS di esempio sul sistema.

Prima di modificare o eseguire gli esempi, leggere la descrizione dei programmi di esempio.

Per scaricare i file di esempio e memorizzarli sul server, completare la seguente procedura:

1. Sul server, scegliere un indirizzario esistente (o crearne uno nuovo) dove si desidera memorizzare i programmi di esempio, i file di configurazione e i file delle politiche.
2. Scaricare i programmi di esempio (ibmjgsssample.zip).
3. Estrarre i file da ibmjgsssample.zip nell'indirizzario sul server.

Estrarre il contenuto del file ibmjgsssample.jar effettua quanto segue:

- Inserire il file ibmgjsssample.jar, che contiene il file .class di esempio, nell'indirizzario selezionato
- creare un sottoindirizzario (denominato config) che contiene i file di configurazione e delle politiche.
- creare un sottoindirizzario (denominato src) che contiene un esempio dei file origine .java.

Informazioni correlate

È possibile leggere le attività correlate o consultare un esempio:

- "Esempi: preparazione dell'esecuzione dei programmi di esempio JGSS" a pagina 409
- "Esempi: esecuzione dei programmi di esempio JGSS" a pagina 409
- "Esempio: esecuzione dell'esempio non JAAS" a pagina 410

Esempi: preparazione dell'esecuzione dei programmi di esempio JGSS:

Dopo aver scaricato il codice sorgente, è necessario eseguire alcune attività di preparazione prima di poter eseguire i programmi di esempio.

Prima di modificare o eseguire gli esempi, consultare "Esempi: IBM JGSS (Java Generic Security Service)" a pagina 402.

Dopo aver scaricato il codice sorgente, è necessario eseguire le seguenti attività prima di poter eseguire i programmi di esempio:

- Modificare i file di configurazione e delle politiche al fine di adattarli al proprio ambiente. Per ulteriori informazioni, fare riferimento ai commenti presenti in ogni file di configurazione e delle politiche.
- Assicurarsi che il file java.security contenga le impostazioni corrette per System i5. Per ulteriori informazioni, consultare "File delle politiche e di configurazione" a pagina 392.
- Inserire il file di configurazione Kerberos modificato (krb5.conf) nell'indirizzario sul server appropriato per la versione di J2SDK che si sta utilizzando:
  - per la versione 1.4 di J2SDK: /QIBM/ProdData/Java400/jdk14/lib/security
  - per la versione 1.5 di J2SE: /QIBM/ProdData/Java400/jdk15/lib/security

  Attività correlate

  "Esempi: scaricamento degli esempi IBM JGSS" a pagina 408
  Quest'argomento contiene le istruzioni per scaricare le informazioni Javadoc JGSS di esempio sul sistema.

  "Esempi: esecuzione dei programmi di esempio JGSS"
  Dopo aver scaricato e modificato il codice sorgente, è possibile eseguire uno degli esempi.

  Riferimenti correlati

  "Esempio: esecuzione dell'esempio non JAAS" a pagina 410
  Per eseguire un esempio, è necessario scaricare e modificare il codice sorgente dell'esempio.

Esempi: esecuzione dei programmi di esempio JGSS:

Dopo aver scaricato e modificato il codice sorgente, è possibile eseguire uno degli esempi.

Prima di modificare o eseguire gli esempi, leggere la descrizione dei programmi di esempio.

Per eseguire un esempio, è necessario avviare prima il programma del server. Il programma del server deve essere attivo e pronto a ricevere i collegamenti prima che venga avviato il programma del client. Il server è pronto a ricevere i collegamenti quando l'utente visualizza `in ascolto sulla porta <server_port>`. Ricordarsi o annotare il valore relativo a <server_port >, ovvero il numero di porta che deve essere specificato all'avvio del client.

Utilizzare il seguente comando per avviare un programma di esempio:

```
java [-Dproperty1=value1 ... -DpropertyN=valueN] com.ibm.security.jgss.test.<program> [options]
```

dove

- [-DpropertyN=valueN] è una o più proprietà Java facoltative, compresi i nomi dei file di configurazione e delle politiche, le opzioni di debug JGSS e il gestore della sicurezza. Per ulteriori informazioni, vedere il seguente esempio ed Eseguire le applicazioni JGSS.
- <program> è un parametro richiesto che specifica il programma di esempio che si desidera eseguire (`Client`, `Server`, `JAASClient`, oppure `JAASServer`).
- [options] è un parametro facoltativo del programma di esempio che si desidera eseguire. Per visualizzare un elenco delle opzioni supportate, utilizzare il seguente comando:

  ```
  java com.ibm.security.jgss.test.<program> -?
  ```

Nota: disattivare le funzioni JAAS nell'esempio abilitato a JGSS impostando la proprietà Java javax.security.auth.useSubjectCredsOnly su false. Naturalmente, il valore predefinito degli esempi abilitati da JAAS deve attivare JAAS, ciò significa che il valore della proprietà è impostato su true. Nei programmi client e server non JAAS la proprietà viene impostata su false, a meno che non sia stato esplicitamente impostato il valore della proprietà.

Informazioni correlate

È possibile leggere le attività correlate o consultare un esempio:

- "Esempi: preparazione dell'esecuzione dei programmi di esempio JGSS" a pagina 409
- "Esempi: scaricamento degli esempi IBM JGSS" a pagina 408
- "Esempio: esecuzione dell'esempio non JAAS"

Esempio: esecuzione dell'esempio non JAAS:

Per eseguire un esempio, è necessario scaricare e modificare il codice sorgente dell'esempio.

Per ulteriori informazioni, consultare l'argomento Scaricare ed eseguire i programmi di esempio.

Avvio del server primario

Utilizzare il seguente comando per avviare il server non JAAS in ascolto sulla porta 4444. Il server funziona da principal (superSecureServer) ed utilizza un server secondario (backupServer). Nel server inoltre vengono visualizzate le informazioni di debug delle credenziali e delle applicazioni.

```
java -classpath ibmjgsssample.jar
     -Dcom.ibm.security.jgss.debug="app, cred"
     com.ibm.security.jgss.test.Server -p 4444
     -n superSecureServer -s backupServer
```

Una volta eseguito con esito positivo questo esempio viene visualizzato il seguente messaggio:

```
in ascolto sulla porta 4444
```

Avvio del server secondario

Utilizzare il seguente comando per avviare un server secondario non JAAS in ascolto sulla porta 3333 e che funzioni come principal backupServer:

```
java -classpath ibmjgsssample.jar
     com.ibm.security.jgss.test.Server -p 3333
     -n backupServer
```

Avvio del client

Utilizzare il seguente comando (immetterlo su un'unica riga) per eseguire un client abilitato a JAAS (myClient). Il client comunica con il server primario sull'host (securityCentral). Il client che viene eseguito con il gestore della sicurezza predefinito abilitato, utilizza i file di configurazione e delle politiche JAAS e il file delle politiche Java dell'indirizzario config. Per ulteriori informazioni relative all'indirizzario config, consultare l'argomento Scaricamento degli esempi IBM JGSS.

```
java -classpath ibmjgsssample.jar
     -Djava.security.manager
     -Djava.security.auth.login.config=config/jaas.conf
     -Djava.security.policy=config/java.policy
     -Djava.security.auth.policy=config/jaas.policy
     com.ibm.security.jgss.test.JAASClient -n myClient
     -s superSecureServer -h securityCentral:4444
```

Informazioni di riferimento Javadoc IBM JGSS

Le informazioni di riferimento Javadoc per IBM JGSS includono le classi e i metodi presenti nel pacchetto di api org.ietf.jgss e le versioni Java di alcuni strumenti di gestione delle credenziali Kerberos.

Sebbene JGSS includa diversi pacchetti accessibili al pubblico (ad esempio, com.ibm.security.jgss e com.ibm.security.jgss.spi), è opportuno utilizzare solo le API del pacchetto org.ietf.jgss standard. Utilizzando solo questo pacchetto l'applicazione viene uniformata alle specifiche di GSS-API garantendo una interoperabilità e una adattabilità ottimali.

- org.ietf.jgss
- "com.ibm.security.krb5.internal.tools Class Kinit" a pagina 379
- "com.ibm.security.krb5.internal.tools Class Ktab" a pagina 381
- "com.ibm.security.krb5.internal.tools Class Klist" a pagina 378

Ottimizzazione delle prestazioni dei programmi Java con IBM Developer Kit per Java

Prendere in considerazione vari aspetti delle prestazioni dell'applicazione Java quando si crea un'applicazione Java.

Sono qui di seguito elencate alcune operazioni che è possibile eseguire per ottenere delle prestazioni migliori:

- Migliorare le prestazioni del codice Java utilizzando il compilatore JIT (Just-In-Time) oppure la cache per i programmi di caricamento classe utente.
- Impostare con attenzione i propri valori per ottenere le prestazioni ottimali di raccolta di dati inutili.
- Utilizzare i metodi nativi solo per avviare le funzioni del sistema relativamente lunghe da eseguire e che non sono disponibili direttamente in Java.
- Utilizzare le eccezioni Java in casi che non rientrano nel regolare flusso dell'applicazione.

Utilizzare questi strumenti con PEX (Performance Explorer) per localizzare i problemi delle prestazioni nei propri programmi Java:

- È possibile raccogliere le informazioni sulle prestazioni della traccia di eventi Java utilizzando la JVM (i5/OS Java virtual machine).
- Per determinare il tempo impiegato in ciascun metodo Java, utilizzare le tracce di chiamata Java.
- Individuare la quantità relativa di tempo CPU impiegato in ciascun metodo Java e tutte le funzioni di sistema utilizzate dal programma Java con la creazione di profili Java.
- Utilizzare Java PDC (Performance Data Collector) per fornire le informazioni di profilo sui programmi eseguiti sul server.

Qualsiasi sessione di lavoro può avviare e arrestare PEX. Normalmente, i dati raccolti sono estesi al sistema e appartengono a tutti i lavori sul sistema, inclusi i propri programmi Java. A volte, potrebbe essere necessario avviare e arrestare la raccolta delle prestazioni dall'interno di un'applicazioneJava. Ciò riduce il tempo di raccolta e può ridurre l'ampio volume di dati normalmente prodotti da una chiamata o traccia di ritorno. PEX non può essere eseguito in un sottoprocesso Java. Per avviare e arrestare una raccolta, è necessario scrivere un metodo nativo che comunichi ad un lavoro indipendente tramite una coda o una memoria condivisa. Quindi, il secondo lavoro avvia e arresta la raccolta al momento appropriato.

Oltre ai dati sulle prestazioni a livello delle applicazioni, è possibile utilizzare gli strumenti delle prestazioni a livello di sistema System i esistenti. Tali strumenti riportano le statistiche per un sottoprocesso Java.

Concetti correlati

"Considerazioni sulle prestazioni della compilazione statica di Java" a pagina 415
A partire dalla V6R1, la velocità di trasformazione non sarà influenzata dal livello di ottimizzazione impostato.

"Strumenti delle prestazioni per la creazione profiliJava" a pagina 417
Il profilo CPU (central processing unit) dell'intero sistema calcola la relativa quantità di tempo della CPU impiegato in ogni metodo Java e in tutte le funzioni di sistema utilizzate dal programma Java.

Informazioni correlate

 Performance Tools for iSeries, SC41-5340

 Questo manuale contiene esempi sui prospetti PEX.

Strumenti delle prestazioni della traccia eventi Java

La i5/OS JVM (Java virtual machine) abilita la traccia di alcuni eventi Java.

È possibile raccogliere tali eventi senza alcuna strumentazione nel codice Java. Questi eventi includono attività come raccolta di dati inutili, creazione di un sottoprocesso, caricamento di una classe e vincolo. Il comando RUNJVA (Esecuzione Java) non specifica tali eventi. Invece si crea una definizione PEX (Performance Explorer) e si utilizza il comando STRPEX (Avvio PEX) per raccogliere questi eventi. Ogni evento contiene utili informazioni sulle prestazioni, come registrazione data/ora e cicli CPU (central processing unit). È possibile tenere traccia sia degli eventi Java che di altre attività del sistema come immissione ed emissione disco, con la stessa definizione di traccia.

Per una descrizione completa degli eventi Java, consultare Performance Tools for iSeries, SC41-5340.

Considerazioni sulle prestazioni Java

La conoscenza delle seguenti considerazioni può essere di ausilio nel migliorare le prestazioni delle applicazioni Java.

Creazione di programmi Java ottimizzati

Il parametro OPTIMIZE viene utilizzato solo quando si creano i programmi Java per un release di destinazione precedente alla V6R1. Per un release di destinazione V6R1 e successive, il valore viene ignorato e viene utilizzato OPTIMIZE(*INTERPRET). Questo significa che il programma Java è interpretato dai codici byte o eseguito con il compilatore JIT (Just-in-Time) quando viene avviato. Le variabili possono essere visualizzate e modificate durante l'esecuzione del debug.

Utilizzo del compilatore JIT (Just-In-Time)

L'utilizzo del compilatore JIT (Just-In-Time) in combinazione con l'MMI (Mixed-Mode Interpreter) risulta in prestazioni di avvio quasi identiche a quelle del codice compilato. MMI interpreta il codice Java fino a raggiungere la soglia specificata dalla proprietà di sistema Java os400.jit.mmi.threshold. Una volta raggiunta la soglia, i5/OS impiega il tempo e le risorse necessarie ad utilizzare il compilatore JIT per compilare un metodo sui metodi utilizzati più frequentemente. L'utilizzo del compilatore JIT fornisce un codice altamente ottimizzato che migliora le prestazioni al tempo di esecuzione rispetto al codice precompilato.

Per ulteriori informazioni, consultare il Comando CL (Control Language) DSPJVAPGM (Visualizzazione programma Java).

Utilizzo delle cache per i programmi di caricamento classe utente

L'Utilizzo della cache JVM (i5/OS Java virtual machine) per i programmi di caricamento classe utente migliora le prestazioni di avvio per le classi caricate da un programma di caricamento classe utente. La cache memorizza gli oggetti programmaJava ottimizzati permettendo, così, alla JVM di riutilizzarli. Il

riutilizzo dei programmi Java memorizzati, migliora le prestazioni in quanto, evita di ricreare gli oggetti programma Java memorizzati in cache e di verificarne il bytecode.

Utilizzare le seguenti proprietà per controllare le cache per i programmi di caricamento classe utente:

os400.define.class.cache.file
Il valore di questa proprietà specifica il nome (con il percorso completo) di un file JAR (Java ARchive) valido. Il file JAR specificato deve contenere almeno un indirizzario JAR valido (come creato dal comando QSH `jar`) e il singolo membro necessario perché il comando `jar` funzioni. Non inserire il file JAR specificato in un CLASSPATH Java. Il valore predefinito di questa proprietà è /QIBM/ProdData/Java400/QDefineClassCache.jar. Per disabilitare la memorizzazione in cache, specificare questa proprietà senza valore.

os400.define.class.cache.hours
Il valore di questa proprietà specifica il periodo di tempo (in ore) di permanenza di un oggetto programma Java nella cache. Se la JVM non utilizza un oggetto programma Java memorizzato in cache durante il periodo di tempo specificato, i5/OS elimina l'oggetto programma Java dalla cache. Il valore predefinito di questa proprietà è 768 ore (33 giorni). Il valore massimo è 9999 (59 settimane circa). Se si specifica un valore che i5/OS non riconosce come numero decimale valido, i5/OS utilizzerà il valore predefinito.

os400.define.class.cache.maxpgms
Il valore di questa proprietà specifica il numero massimo di oggetti programma Java che la cache può contenere. Quando la cache supera questo limite, i5/OS rimuove l'oggetto programma Java meno recente dalla cache. i5/OS determina qual è il programma memorizzato in cache meno recente confrontando gli orari in cui la JVM ha fatto riferimento l'ultima volta agli oggetti programma Java. Il valore predefinito è 5000 e il valore massimo è 40000. Se si specifica un valore che i5/OS non riconosce come numero decimale valido, i5/OS utilizzerà il valore predefinito.

Utilizzare DSPJVAPGM sul file JAR, specificato nella proprietà os400.define.class.cache.file, per determinare il numero di oggetti programma Java memorizzati in cache.

- Il campo **programmi Java** del pannello DSPJVAPGM indica il numero di oggetti programma Java memorizzati in cache.
- Il campo **Dimensione programma Java** indica la quantità di memoria utilizzata dagli oggetti programma Java memorizzati in cache.
- Gli altri campi del pannello DSPJVAPGM non sono importanti quando si utilizza il comando su un file JAR utilizzato per la memorizzazione in cache.

Prestazioni della cache

L'esecuzione di alcune applicazioni Java può memorizzare in cache un ampio numero di oggetti programma Java. Utilizzare DSPJVAPGM per determinare se il numero di programmi Java memorizzati in cache si avvicina al valore massimo consentito prima della fine dell'elaborazione dell'applicazione. Le prestazioni dell'applicazione potrebbero non essere ottimali se la cache si riempie, poiché i5/OS potrebbe eliminare dalla cache alcuni programmi necessari all'applicazione.

È possibile, comunque, evitare i problemi di prestazione che si presentano quando la cache si riempie. Ad esempio, è possibile configurare le applicazioni in modo che utilizzino cache separate per applicazioni che vengono eseguite frequentemente, ma caricano programmi differenti nella cache. L'utilizzo di cache separate evita che la cache si riempia e, di conseguenza, che i5/OS elimini dalla cache i programmi Java. In alternativa, è possibile aumentare il numero specificato per la proprietà os400.define.class.cache.maxpgms.

È possibile utilizzare il comando CL Modifica programmaJava (CHGJVAPGM) sul file JAR per modificare l'ottimizzazione delle classi nella cache. CHGJVAPGM influenza solo i programmi correntemente contenuti nella cache. Una volta apportate le modifiche ai livelli di ottimizzazione, la proprietà os400.defineClass.optLevel specifica come ottimizzare le classi aggiunte alla cache.

Ad esempio, per utilizzare il JAR cache inviato con un numero massimo di 20000 oggetti programma Java, dove ciascun programma Java ha una durata massima di 1 anno, impostare i seguenti valori per le proprietà della cache:

```
os400.define.class.cache.file    /QIBM/ProdData/Java400/QDefineClassCache.jar
os400.define.class.cache.hours   8760
os400.define.class.cache.maxpgms 20000
```

Selezione della modalità da utilizzare quando si esegue un programma Java

Quando si esegue un programma Java, è possibile selezionare la modalità che si desidera utilizzare. Tutte le modalità verificano il codice e creano un oggetto del programma Java per conservare il formato verificato in precedenza del programma.

È possibile utilizzare una delle seguenti modalità:

- Interpretato
- Compilazione JIT (Just-In-Time)

Modalità di selezione	Dettagli
Interpretato	Ciascun bytecode viene interpretato nel tempo di esecuzione. Per informazioni sull'esecuzione del programma Java in modalità interpretato, consultare il comando RUNJVA (Esecuzione Java) .

Compilazione JIT (Just-In-Time)	i5/OS interpreta i metodi Java fino a raggiungere la soglia specificata dalla proprietà di sistema Java os400.jit.mmi.threshold. Una volta raggiunta la soglia, i5/OS utilizza il compilatore JIT per compilare i metodi in istruzioni native di sistema.
	
	Per utilizzare il compilatore Just-In-Time, è necessario impostare il valore del compilatore su `jitc`. È possibile impostare il valore aggiungendo una variabile di ambiente o impostando la proprietà di sistema java.compiler. Selezionare un metodo dall'elenco riportato di seguito per impostare il valore del compilatore:
	• Da una riga comandi i5/OS, aggiungere la variabile di ambiente utilizzando il comando ADDENVVAR (Aggiunta variabile ambiente). Successivamente eseguire il programma Java utilizzando il comando RUNJVA (Esecuzione Java) o il comando JAVA. Ad esempio, utilizzare:
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc)`
	`JAVA CLASS(Test)`
	• Impostare la proprietà di sistema java.compiler sulla riga comandi i5/OS. Ad esempio, immettere `JAVA CLASS(Test) PROP((java.compiler jitc))`
	• Impostare la proprietà di sistema java.compiler sulla riga comandi di Qshell Interpreter. Ad esempio, immettere `java -Djava.compiler=jitc Test`

Esistono tre modi in cui è possibile eseguire un programma Java (CL, QSH e JNI). Ciascuno di essi specifica la modalità diversamente. Questa tabella indica come si effettua tale operazione.

Modalità	Comando CL	Comando QShell	API di richiamo JNI
Interpret	INTERPRET(*YES)	-Djava.compiler=NONE -interpret	os400.run.mode=interpret
JIT	INTERPRET(*JIT)	-Djava.compiler=jitc	os400.run.mode=jitc

Interprete Java

L'interpreter Java fa parte della JVM (Java virtual machine) che interpreta i file di classe Java per una specifica piattaforma hardware. L'interpreter Java decodifica ogni bytecode ed esegue una serie di istruzioni macchina per quel bytecode.

JVM (Java virtual machine)

Compilazione statica di Java

Nella V6R1, l'elaborazione diretta non è più supportata. Questo significa che il livello di ottimizzazione per i programmi Java è ignorato e OPTIMIZE(*INTERPRET) viene utilizzato quando si crea un programma Java su V6R1 o un release successivo.

Come nei release precedenti, un programma Java contiene una versione precedentemente verificata di uno o più file di classe Java. Tuttavia, in V6R1, un programma Java non contiene istruzioni macchina. Durante il tempo di esecuzione, il programma Java viene interpretato dai codici byte o altrimenti eseguito con il compilatore JIT (Just-In-Time).

Per i release di destinazione precedenti alla V6R1, il parametro OPTIMIZE specifica il livello di ottimizzazione del programma Java. Quando si crea un programma Java per un release di destinazione precedente alla V6R1, il valore del parametro OPTIMIZE è incapsulato nel programma Java, ma non viene generata alcuna istruzione macchina. Il valore incapsulato del parametro OPTIMIZE viene utilizzato durante la riconversione del programma Java sul release di destinazione del sistema operativo.

Considerazioni sulle prestazioni della compilazione statica di Java:

A partire dalla V6R1, la velocità di trasformazione non sarà influenzata dal livello di ottimizzazione impostato.

Il parametro OPTIMIZE viene utilizzato solo quando si creano i programmi Java per un release di destinazione precedente alla V6R1. Per un release di destinazione V6R1 e successive, il valore viene ignorato e viene utilizzato OPTIMIZE(*INTERPRET). Il programma Java viene interpretato dai codici byte o eseguito con il compilatore JIT (Just-in-Time) quando è avviato. Se i codici byte sono in fase di interpretazione, le variabili possono essere visualizzate e modificate durante l'esecuzione del debug. Se il metodo è stato compilato dal compilatore JIT, non sono disponibili informazioni di debug per System i5 Debugger.

Per i release di destinazione precedenti alla V6R1, il parametro OPTIMIZE specifica il livello di ottimizzazione del programma Java. Quando si crea un programma Java per un release di destinazione precedente alla V6R1, il valore del parametro OPTIMIZE è incapsulato nel programma Java, ma non viene generata alcuna istruzione macchina. Il valore incapsulato del parametro OPTIMIZE viene utilizzato durante la riconversione del programma Java sul release di destinazione del sistema operativo.

Compilatore JIT (Just-In-Time)

Un compilatore JIT (Just-In-Time) è un compilatore specifico della piattaforma che genera istruzioni di sistema per ogni metodo come necessario.

Nota: l'impostazione predefinita per i5/OS consiste nell'interpretare (non compilare) i metodi Java utilizzando l'MMI (Mixed-Mode Interpreter). L'MMI crea un profilo di ciascun metodo Java man

mano che lo interpreta. Una volta raggiunta la soglia specificata dalla proprietà os400.jit.mmi.threshold, l'MMI specifica che i5/OS utilizza il compilatore JIT per compilare il metodo.

Per ulteriori informazioni, consultare le voci per le proprietà java.compiler e os400.jit.mmi.threshold nell'argomento Elenco delle proprietà di sistema Java.

Concetti correlati

"Elenco delle proprietà di sistema Java" a pagina 16
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Raccolta di dati inutili Java

La raccolta di dati inutili è il processo di liberare la memoria che viene utilizzata da oggetti cui non fa più riferimento un programma. Con tale raccolta, non è più necessario che i programmatori scrivano il codice incline all'errore per ″liberare″ o ″cancellare″ i propri oggetti. Questo codice dà frequentemente come risultato degli errori di programma ″perdite di memoria″. Il programma di raccolta di dati inutili individua automaticamente un oggetto o gruppo di oggetti che il programma dell'utente non può più raggiungere. Questo avviene in quanto non esiste alcun riferimento a quell'oggetto in nessuna struttura del programma. Una volta raccolto l'oggetto, è possibile assegnare lo spazio per altri utilizzi.

Java runtime environment include un programma di raccolta di dati inutili che libera la memoria non più in uso. Tale programma di raccolta si attiva automaticamente, quando richiesto.

È inoltre possibile avviare il programma di raccolta di dati inutili esplicitamente sotto il controllo del programma Java utilizzando il metodo java.lang.Runtime.gc().

Concetti correlati

"Utilizzo del comando Gestione lavori JVM" a pagina 419
Se si sta utilizzando IBM Technology for Java Virtual Machine, è possibile utilizzare il comando CL WRKJVMJOB (Gestione lavori JVM) per raccogliere dati sulle prestazioni.

Raccolta di dati inutili avanzata di IBM Developer Kit per Java

IBM Developer Kit per Java implementa un algoritmo del programma di raccolta di dati inutili avanzato. Tale algoritmo permette di trovare e raccogliere oggetti irraggiungibili senza pause significative nell'operazione del programma Java. Un programma di raccolta simultaneo rileva contemporaneamente i riferimenti agli oggetti all'interno dei sottoprocessi in esecuzione, invece che in un singolo sottoprocesso.

Molti programmi di raccolta di dati inutili sono di tipo ″stop-the-world″. Ciò significa che nel momento in cui si verifica il ciclo di raccolta, tutti i sottoprocessi, ad eccezione di quello per la raccolta dei dati inutili, si arrestano mentre il programma di raccolta di dati inutili esegue il suo lavoro. Quando si verifica questo, i programmi Java effettuano una pausa e si logora ogni capacità del processore multiplo della piattaforma relativa a Java, mentre il programma di raccolta di dati inutili è al lavoro. L'algoritmo System i non arresta tutti i sottoprocessi del programma simultaneamente. Esso consente a quei sottoprocessi di continuare l'operazione mentre il programma di raccolta di dati inutili completa la sua attività. Ciò impedisce le pause e consente l'utilizzo di tutti i processori durante la raccolta di dati inutili.

La raccolta di dati inutili si verifica automaticamente in base ai parametri che si specificano quando si avvia la Java virtual machine. È inoltre possibile avviare tale raccolta esplicitamente sotto il controllo del programma Java utilizzando il metodo java.lang.Runtime.gc().

Per una definizione di base, consultare la Raccolta di dati inutili Java .

Per le opzioni di raccolta di dati inutili disponibili con IBM Technology for Java, consultare il capitolo Garbage collector diagnostics nel manuale Java Diagnostics Guide 6.

Informazioni correlate

Tecnologia Java, stile IBM: politiche di raccolta dei dati inutili

Considerazioni sulle prestazioni della raccolta di dati inutili Java

La raccolta di dati inutili sulla i5/OS JVM (Java virtual machine) opera in modalità asincrona continua. È possibile che il parametro GCHINL (Dimensione iniziale della raccolta di dati inutili) sul comando RUNJVA (Esecuzione Java) influenzi le prestazioni dell'applicazione.

Il parametro GCHINL specifica la quantità del nuovo spazio dell'oggetto consentito tra le raccolte di dati inutili. Un valore piccolo può causare un sovraccarico della raccolta di dati inutili. Un valore grande può limitare tale raccolta e causare errori dovuti ad un esaurimento di memoria. Tuttavia, per la maggior parte delle applicazioni, i valori predefiniti dovrebbero essere corretti.

La raccolta di dati inutili stabilisce che un oggetto non è più necessario valutando se esistono riferimenti validi ad esso.

Considerazioni sulle prestazioni di richiamo del metodo nativo Java

L'esecuzione del richiamo del metodo nativo sulla piattaforma System i non è uguale all'esecuzione di tale richiamo su altre piattaforme.

Per la JVM Classic, Java su System i5 è stato ottimizzato spostando la JVM (Java virtual machine) sotto la MI (machine interface). Il richiamo del metodo nativo richiede una chiamata all'interno del codice MI e può richiedere chiamate JNI (Java Native Interface) dispendiose retroattive nella JVM (Java virtual machine). La JVM IBM Technology for Java viene eseguita sopra la MI. Sia per Classic che per IBM Technology for Java, i metodi nativi non devono eseguire routine di piccole dimensioni che è possibile scrivere facilmente in Java. Utilizzare i metodi nativi solo per avviare le funzioni del sistema relativamente lunghe da eseguire e che non sono disponibili direttamente in Java.

Considerazioni sulle prestazioni dell'eccezione Java

L'architettura delle eccezioni System i consente capacità versatili di interruzione e ripetizione. Essa consente anche l'interazione di un linguaggio misto. La generazione di eccezioni Java sulla piattaforma System i può essere più dispendiosa che su altre piattaforme. Ciò non dovrebbe influenzare le prestazioni globali dell'applicazione a meno che non vengano utilizzate abitualmente le eccezioni Java nel normale percorso di applicazione.

Strumenti delle prestazioni delle tracce di chiamata Java

Le tracce di chiamata del metodo Java forniscono informazioni significative sulle prestazioni riguardo al tempo impiegato in ogni metodo Java.

L'emissione della traccia di restituzione/chiamata prodotta con il comando PRTPEXRPT (Stampa prospetto Performance Explorer) mostra il tempo CPU (central processing unit) per ogni chiamata ad ogni metodo Java tracciato. In alcuni casi, non è possibile abilitare tutti i file di classe per la traccia della restituzione di chiamata. O è possibile che si chiamino funzioni del sistema e metodi nativi non abilitati per la traccia. In questo caso, tutto il tempo CPU utilizzato in questi metodi o funzioni di sistema si accumula. Quindi viene riportato all'ultimo metodo Java che viene chiamato ed è stato abilitato.

Strumenti delle prestazioni per la creazione profiliJava

Il profilo CPU (central processing unit) dell'intero sistema calcola la relativa quantità di tempo della CPU impiegato in ogni metodo Java e in tutte le funzioni di sistema utilizzate dal programma Java.

Utilizzare una definizione PEX (Performance Explorer) che tenga traccia degli eventi del ciclo di esecuzione *PMCO (performance monitor counter overflow). Gli esempi sono in genere specificati in intervalli di un millisecondo. Per raccogliere un profilo di traccia valido, è necessario eseguire l'applicazione Java finché non vengono accumulati da due a tre minuti di tempo della CPU. Ciò potrebbe

produrre più di 100,000 esempi. Il comando PRTPEXRPT (Stampa prospetto PEX) produce un istogramma del tempo della CPU impiegato durante l'intera applicazione. Questo include ciascun metodo Java e tutte le attività a livello del sistema.

Nota: il profilo CPU non indica l'utilizzo relativo della CPU per i programmi Java interpretati.

Java Virtual Machine Tool Interface

La JVMTI (Java Virtual Machine Tool Interface) è un'interfaccia per analizzare la JVM (Java virtual machine).

JVM TI sostituisce JVMPI (Java Virtual Machine Profiler Interface) e JVMDI (Java Virtual Machine Debugger Interface). JVMTI contiene tutta la funzionalità di JVMDI e di JVMPI, oltre alle nuove funzioni. La JVMTI è stata aggiunta come parte di J2SE 5.0. In JDK 6, le interfacce JVMDI e JVMPI non sono più offerte e JVMTI è la sola opzione disponibile.

Il supporto JVM TI posiziona gli hook nella JVM e nel compilatore JIT (Just-In-Time) che, quando attivati, forniscono informazioni sull'evento a un agent di creazione del profilo. L'agent di creazione del profilo è implementato come un programma di servizio ILE (integrated language environment). Il programma per la creazione profili invia informazioni di controllo alla JVM per abilitare e disabilitare eventi JVMTI. Ad esempio, è possibile che il programma per la creazione profili non sia interessato agli hook di accesso e uscita dal metodo e potrebbe comunicare alla JVM di non volere ricevere queste notifiche di eventi. La JVM e JIT hanno hook di eventi JVMTI incorporati che inviano notifiche di eventi all'agent di creazione profilo se l'evento è abilitato. Il programma per la creazione profili comunica alla JVM a quali eventi è interessato e la JVM invia notifiche degli eventi al programma di creazione profilo quando questi si verificano.

Un programma di servizio, denominato QJVAJVMTI che si trova nella libreria QSYS, supporta le funzioni JVMTI.

Informazioni correlate

JVMTI (Java Virtual Machine Tool Interface) di Sun Microsystems, Inc.

Raccolta di dati sulle prestazioni Java

Per raccogliere i dati sulle prestazioni Java sul server, attenersi alla seguente procedura.

1. Creare una definizione PEX (Performance Explorer) che specifichi:
   - Un nome definito dall'utente
   - Tipo di raccolta dati
   - Nome lavoro
   - Serie di eventi di sistema su cui si ha intenzione di raccogliere informazioni del sistema

   Nota: una definizione PEX di *STATS è preferibile ad una definizione *TRACE se l'emissione che si desidera è di tipo `java_g -prof` e si conosce il nome lavoro specifico del programma Java.

   Segue un esempio di una definizione *STATS:

   ```
   ADDPEXDFN DFN(YOURDFN) JOB(*ALL/YOURID/QJVACMDSRV) DTAORG(*HIER)
   TEXT('la definizione stats')
   ```

   Questa definizione *STATS non richiama tutti gli eventi Java in esecuzione. Si crea un profilo solo degli eventi Java che si trovano nella propria sessione Java. Questa modalità di operazione può aumentare il tempo di esecuzione del programma Java.

   Segue un esempio di una definizione *TRACE:

   ```
   ADDPEXDFN DFN(YOURDFN) TYPE(*TRACE) JOB(*ALL) TRCTYPE(*SLTEVT)
   SLTEVT(*YES) PGMEVT(*JVAENTRY *JVAEXIT)
   ```

Questa definizione *TRACE raccoglie i dati da eventi di accesso e di uscita dai programmi Java nel sistema. Il compilatore JIT (Just-In-Time) genera il codice con gli hook di accesso e di uscita quando si abilita *JVAENTRY e *JVAEXIT sulla definizione PEX utilizzando la proprietà di sistema os400.enbpfrcol. L'analisi di questo tipo di raccolta può essere più lento di una traccia *STATS, a seconda di quanti sono gli eventi di programma Java di cui si dispone e della durata della raccolta di dati PEX.

2. Abilitare *JVAENTRY e *JVAEXIT sulla definizione PEX utilizzando la proprietà di sistema os400.enbpfrcol. Impostare il valore di os400.enbpfrcol su **1**.
3. Avviare la raccolta dati PEX utilizzando il comando STRPEX (Avvio Performance Explorer).
4. Eseguire il programma che si intende analizzare.

   Sarebbe opportuno che questo programma non fosse un ambiente di sviluppo. Esso genera un'ingente quantità di dati in breve tempo. Sarebbe opportuno limitare la raccolta dati a cinque minuti. Un programma Java in esecuzione durante questo arco tempo genera molti dati di sistema PEX. Se vengono raccolti troppi dati, è necessario troppo tempo per elaborarli.
5. Arrestare la raccolta dati PEX utilizzando il comando ENDPEX (Fine Performance Explorer).

   Nota: se la raccolta dati PEX è già stata arrestata altre volte, è necessario specificare un file di sostituzione di *YES o i propri dati non verranno salvati.
6. Collegare l'indirizzario dell'IFS (Integrated File System) al sistema con il programma di visualizzazione scelto: programma di visualizzazione java_g -prof o Jinsight.

   È possibile copiare questo file dal server ed utilizzarlo come immissione per qualsiasi strumento per la creazione di profili adatto.

Concetti correlati

"Proprietà di sistema Java" a pagina 15
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Informazioni correlate

Comando CL CRTJVAPGM (Creazione programma Java)

Utilizzo del comando Gestione lavori JVM

Se si sta utilizzando IBM Technology for Java Virtual Machine, è possibile utilizzare il comando CL WRKJVMJOB (Gestione lavori JVM) per raccogliere dati sulle prestazioni.

È possibile accedere alle informazioni disponibili dal comando WRKJVMJOB sia dallo schermo di WRKJOB (Gestione lavoro) che immettendo il comando WRKJVMJOB. Le informazioni verranno visualizzate solo per i lavori IBM Technology for Java Virtual Machine.

Quando si utilizza WRKJVMJOB sono disponibili le seguenti informazioni o funzioni:

- Gli argomenti e le opzioni con cui è stata avviata la JVM.
- Le variabili di ambiente sia per ILE che per PASE.
- Le richieste di blocco Java in sospeso per il lavoro JVM.
- Le informazioni di raccolta dei dati inutili.
- Le proprietà di sistema Java.
- L'elenco dei sottoprocessi associati alla JVM.
- La registrazione dei lavori completati parzialmente per il lavoro JVM.
- La capacità di gestire i file di immissione e di emissione di spool per il lavoro JVM.
- La capacità di generare i dump JVM (sistema, heap, Java) da un'opzione di pannello. Queste capacità sono disponibili anche dal comando GENJVMDMP (Generazione dump JVM).
- La capacità di abilitare e disabilitare la raccolta dei dati inutili dettagliata da un'opzione di pannello.

Informazioni correlate

Comandi e strumenti per il IBM Developer Kit per Java

Quando si utilizza il IBM Developer Kit per Java, è possibile utilizzare gli strumenti Java con Qshell Interpreter o i comandi CL.

Se si ha esperienza precedente di programmazione Java, potrebbe essere più comodo utilizzare gli strumenti Java Qshell Interpreter perché sono simili agli strumenti che si utilizzerebbero con Sun Microsystems, Inc. Java Development Kit. Consultare l'argomento relativo alla Qshell per informazioni sull'utilizzo dell'ambiente Qshell.

Se si è un programmatore i5/OS, utilizzare i comandi CL per Java tipici dell'ambiente System i. Continuare a leggere per ulteriori informazioni sull'utilizzo dei comandi CL e dei comandi System i Navigator.

Informazioni correlate

Qshell Interpreter

Strumenti Java supportati da IBM Developer Kit per Java

L'ambiente Qshell include gli strumenti di sviluppo Java normalmente richiesti per lo sviluppo di programmi.

Tranne poche eccezioni, gli strumenti Java supportano la sintassi e le opzioni che sono documentate da Sun Microsystems, Inc. Essi devono essere tutti eseguiti utilizzando il Qshell Interpreter.

È possibile avviare Qshell Interpreter utilizzando il comando STRQSH o QSH (Avvio Qshell). Quando Qshell Interpreter è in esecuzione, compare un pannello Immissione comando QSH. Tutte le emissioni e i messaggi derivati dagli strumenti e dai programmi Java che sono in esecuzione sotto Qshell compaiono in questo pannello. Qualsiasi immissione in un programma Java è inoltre letta da questo pannello.

Nota: Le funzioni dell'immissione di comando i5/OS non sono disponibili direttamente dall'interno della Qshell. Per richiamare una riga comando, premere F21 (Immissione comando CL).

Concetti correlati

"NAWT (Native Abstract Windowing Toolkit)" a pagina 247
NAWT (Native Abstract Windowing Toolkit), più che un toolkit, è un termine creato per indicare il supporto i5/OS nativo che fornisce ai servlet e alle applicazioni Java la capacità di utilizzare le funzioni grafiche AWT (Abstract Windowing Toolkit) offerte da J2SE (Java 2 Platform, Standard Edition).

Strumenti Java

Consultare gli argomenti riportati di seguito per la descrizione degli strumenti Java.

Strumento appletviewer Java:

Lo strumento `appletviewer` Java consente di eseguire applet senza un browser web. Esso è compatibile con lo strumento `appletviewer` fornito da Sun Microsystems, Inc.

Per eseguire lo strumento `appletviewer`, è necessario utilizzare NAWT (Native Abstract Window Toolkit) e utilizzare la classe `sun.applet.AppletViewer` o eseguire lo strumento appletviewer nel Qshell Interpreter.

Il seguente esempio illustra come utilizzare la classe`sun.applet.AppletViewer` e come eseguire l'esempio demo TicTacToe. Per informazioni sul caricamento degli esempi demo, consultare le istruzioni Come estrarre file di esempio.

Dalla riga dei comando immettere:

```
cd '/home/MyUserID/demo/applets/TicTacToe'
```

Per JDK 1.4, immettere il comando:

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.4))
```

Per JDK 1.5, immettere il comando:

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.5))
```

| Il seguente esempio illustra come utilizzare lo strumento `appletviewer`nel Qshell interpreter e come eseguire l'esempio demo TicTacToe. Per informazioni sul caricamento degli esempi demo, consultare le istruzioni Come estrarre file di esempio.

Il comando corrispondente è:

```
cd /home/MyUserID/demo/applets/TicTacToe
```

Per JDK 1.4, immettere il comando:

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.4 example1.html
```

Per JDK 1.5, immettere il comando:

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.5 example1.html
```

Nota: -J sono indicatori del tempo di esecuzione per Appletviewer. -D sono proprietà.

Concetti correlati

"NAWT (Native Abstract Windowing Toolkit)" a pagina 247
NAWT (Native Abstract Windowing Toolkit), più che un toolkit, è un termine creato per indicare il supporto i5/OS nativo che fornisce ai servlet e alle applicazioni Java la capacità di utilizzare le funzioni grafiche AWT (Abstract Windowing Toolkit) offerte da J2SE (Java 2 Platform, Standard Edition).

Informazioni correlate

Strumento appletviewer di Sun Microsystems, Inc.

Come estrarre i file di esempio:

La procedura che segue descrive una delle modalità di estrazione dei file di esempio prima di eseguire lo strumento appletviewer Java. Questa procedura assume che si desideri estrarre i file di esempio nell'indirizzario principale.

1. Immettere il comando Avvio Qshell (QSH) sulla riga comandi.
2. Creare, se non è già presente, un indirizzario IFS (integrated file system) di livello principale per il proprio ID utente:
   ```
   mkdir /home/MyUserID
   ```
3. Creare un indirizzario demo all'interno dell'indirizzario IFS:
   ```
   mkdir /home/MyUserID/demo
   ```
4. Passare all'indirizzario demo:
   ```
   cd /home/myUserId/demo
   ```
5. Per JDK 1.4, utilizzare questo comando:
   ```
   jar xf /QIBM/ProdData/Java400/jdk14/demo.jar
   ```
   Per JDK 1.5, utilizzare questo comando:
   ```
   jar xf /QIBM/ProdData/Java400/jdk15/demo.jar
   ```

Strumento apt Java:

Lo strumento `apt` Java elabora annotazioni di programma.

Lo strumento `apt` è disponibile solo con JDK 1.5 e le versioni successive. Lo strumento `apt` è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni sullo strumento `apt`, consultare Annotation Processing Tool (apt) di Sun Microsystems, Inc.

Strumento extcheckJava:

In J2SE (Java 2 Platform, Standard Edition), lo strumento `extcheck` rileva i conflitti di versione tra un file JAR di destinazione e i file JAR di estensione attualmente installati. Esso è compatibile con il `keytool` fornito da Sun Microsystems, Inc.

Lo strumento `extcheck` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

Strumento extcheck di Sun Microsystems, Inc.

Applicazione hwkeytool Java:

L'applicazione `hwkeytool` consente di utilizzare le capacità di crittografia del coprocessore crittografico modello 4764 con JCE (Java Cryptography Extension) e JCA (Java Cryptography Architecture).

L'applicazione `hwkeytool` per l'hardware utilizza la stessa sintassi e gli stessi comandi dell'applicazione `keytool`, fatta eccezione per due comandi e la memorizzazione chiave predefinita. L'applicazione keytool per l'hardware fornisce dei parametri aggiuntivi ai comandi `-genkey` e `delete`.

Sul comando `-genkey`, sono disponibili i seguenti parametri aggiuntivi:

-KeyLabel
Consente di impostare un'etichetta specifica per la chiave hardware.

-hardwaretype
Determinare il tipo di coppia di chiavi: PKDS (public key data set) o RETAINED.

-hardwareusage
Impostare l'utilizzo della coppia di chiavi generata, che può essere una chiave di sola firma oppure una chiave di firma e di gestione chiavi.

Sul comando `delete`, è disponibile un parametro aggiuntivo **-hardwarekey**, che cancella la coppia di chiavi dalla memorizzazione chiave e dall'hardware.

Il nome della memorizzazione chiave predefinita è `.HWkeystore`. È possibile modificare tale nome utilizzando il parametro **-keystore**.

Coprocessore crittografico 4764

"Keytool Java" a pagina 424
In J2SE (Java 2 Platform, Standard Edition), il `keytool` crea coppie di chiavi pubbliche e private e certificazioni autofirmate e gestisce le memorizzazioni chiave. In J2SDK, gli strumenti `jarsigner` e `keytool` sostituiscono lo strumento `javakey`. Esso è compatibile con il `keytool` fornito da Sun Microsystems, Inc.

Strumento idlj Java:

Lo strumento `idlj` genera collegamenti Java da uno specifico file IDL (Interface Definition Language).

Lo strumento `idlj` è inoltre conosciuto come compilatore da IDL a Java. Esso è compatibile con lo strumento `idlj` fornito da Sun Microsystems, Inc.

Informazioni correlate

Strumento idlj di Sun Microsystems, Inc.

Strumento jar Java:

Lo strumento `jar` combina più file in un singolo file JAR (Java ARchive). Esso è compatibile con lo strumento `jar` fornito da Sun Microsystems, Inc

Lo strumento `jar` è disponibile utilizzando Qshell Interpreter.

IFS (integrated file system

jar - lo strumento Java Archive di Sun Microsystems, Inc.

Strumento jarsigner Java:

In J2SE (Java 2 Platform, Standard Edition), la funzione `jarsigner` firma i file JAR e verifica le firme sui file JAR firmati.

Lo strumento `jarsigner` accede alla memorizzazione chiave, che il `keytool` crea e gestisce, quando è necessario che trovi la chiave privata per firmare un file JAR. In J2SDK, gli strumenti `jarsigner` e `keytool` sostituiscono lo strumento `javakey`. Esso è compatibile con lo strumento `jarsigner` fornito da Sun Microsystems, Inc.

Lo strumento `jarsigner` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

jarsigner - JAR Signing and Verification Tool di Sun Microsystems, Inc.

Strumento javac Java:

Lo strumento `javac` compila i programmi Java. Esso è compatibile con lo strumento `javac` fornito da Sun Microsystems, Inc. con un'eccezione.

-classpath
Non sovrascrive il classpath predefinito. Al contrario, viene accodato al classpath predefinito di sistema. L'opzione `-classpath` sostituisce la variabile di ambiente CLASSPATH.

Lo strumento `javac` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

javac - programma di compilazione del linguaggio di programmazione Java di Sun Microsystems, Inc.

Strumento javadoc Java:

Lo strumento `javadoc` genera la documentazione API. Esso è compatibile con lo strumento `javadoc` fornito da Sun Microsystems, Inc.

Lo strumento `javadoc` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

Strumento javadoc di Sun Microsystems, Inc.

Strumento javah Java:

Lo strumento `javah` facilita l'implementazione dei metodi nativi di Java. Esso è compatibile con lo strumento `javah` fornito da Sun Microsystems, Inc. con alcune eccezioni.

Nota: scrivere i metodi nativi significa che la propria applicazione non è Java puro al 100%. Ciò significa inoltre che la propria applicazione non è direttamente trasferibile su più piatteforme. I metodi nativi sono, per natura, specifici del sistema o della piattaforma. L'utilizzo di tali metodi può aumentare i costi di sviluppo e manutenzione per le proprie applicazioni.

Lo strumento `javah` è disponibile utilizzando Qshell Interpreter. Esso legge un file di classe Java e crea un file di intestazione in linguaggio C nell'indirizzario di lavoro corrente. Il file di intestazione scritto è un STMF (Stream File) System i5. Esso deve essere copiato in un membro di file prima di potere essere incluso in un programma C System i5

Lo strumento `javah` è compatibile con lo strumento fornito da Sun Microsystems, Inc. Se vengono specificate queste opzioni, tuttavia, il server le ignora.

-td Lo strumento `javah` su System i5 non richiede un indirizzario temporaneo.

-stubs

Java su System i5 supporta solo il formato JNI (Java Native Interface) dei metodi nativi. Gli stub venivano richiesti solo per il formato precedente a JNI dei metodi nativi.

-trace È collegato all'emissione del file stub .c, che Java su System i5 non supporta.

-v Non supportato.

Nota: è sempre necessario specificare l'opzione -jni. System i5 non supporta le implementazioni del metodo nativo precedenti a JNI.

Informazioni correlate

Strumento javah di Sun Microsystems, Inc.

Strumento javap Java:

Lo strumento `javap` disassembla i file Java compilati e stampa una rappresentazione del programma Java. Ciò può essere utile quando il codice sorgente originale non è più disponibile su un sistema.

-b Questa opzione viene ignorata. La compatibilità con le versioni precedenti non è richiesta perché Java su System i5 supporta solo le versioni recenti di JDK (Java Development Kit).

-p Su System i5, -p non è un'opzione valida. È necessario specificare -private.

-verify

Questa opzione viene ignorata. Lo strumento `javap` non effettua una verifica su System i5.

Lo strumento `javap` è disponibile utilizzando Qshell Interpreter.

Nota: possibile che l'utilizzo dello strumento `javap` per disassemblare le classi violi l'accordo di licenza per tali classi. Consultare tale accordo per le classi prima di utilizzare lo strumento `javap`.

Informazioni correlate

Strumento javap di Sun Microsystems, Inc.

Keytool Java:

In J2SE (Java 2 Platform, Standard Edition), il `keytool` crea coppie di chiavi pubbliche e private e certificazioni autofirmate e gestisce le memorizzazioni chiave. In J2SDK, gli strumenti `jarsigner` e `keytool` sostituiscono lo strumento `javakey`. Esso è compatibile con il `keytool` fornito da Sun Microsystems, Inc.

Il `keytool` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

keytool di Sun Microsystems, Inc.

Strumento native2ascii Java:

Lo strumento `native2ascii` converte un file con caratteri codificati nativi (caratteri diversi da Latino 1 e Unicode) in uno con caratteri codificati Unicode. Esso è compatibile con lo strumento `native2ascii` fornito da Sun Microsystems, Inc.

Lo strumento `native2ascii` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

Strumento native2ascii di Sun Microsystems, Inc.

Strumento orbd Java:

Lo strumento`orbd` consente di localizzare con esattezza e di richiamare oggetti permanenti sui server nell'ambiente CORBA.

ORBD viene utilizzato al posto di Transient Naming Service (tnameserv), perché include Transient Naming Service e Persistent Naming Service. Lo strumento `orbd` incorpora le funzioni di un Server Manager, di un Servizio di denominazione interfacciabile e del server nomi Bootstrap. Quando utilizzato insieme al `servertool`, il Server Manager localizza, registra e attiva un server quando il cliente desidera attivare il server.

Informazioni correlate

Strumento orbd di Sun Microsystems, Inc.

Strumento pack200 Java:

Lo strumento `pack200` è un'applicazione Java che comprime un file JAR in un file `pack200`.

Lo strumento `pack200` è disponibile solo con JDK 1.5 e versioni successive. Lo strumento `pack200` è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni, esaminare lo strumento pack200 di Sun Microsystems, Inc.

Concetti correlati

"Strumento unpack200 Java" a pagina 427
Lo strumento `unpack200` Java decomprime un file `pack200` in un file JAR.

Policytool Java:

In J2SDK (Java 2 SDK), Edizione Standard, il `policytool` crea e modifica i file di configurazione politica esterni che definiscono la politica di sicurezza Java della propria installazione. Esso è compatibile con il `policytool` fornito da Sun Microsystems, Inc.

Il `policytool` è uno strumento GUI disponibile con Qshell Interpreter e NAWT (Native Abstract Window Toolkit). Consultare IBM Developer Kit per Java Native Abstract Window Toolkit per ulteriori informazioni.

Informazioni correlate

policytool di Sun Microsystems, Inc.

Strumento rmic Java:

Lo strumento `rmic` genera file stub e di classe per gli oggetti Java. Esso è compatibile con lo strumento `rmic` fornito da Sun Microsystems, Inc.

Lo strumento `rmic` è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni sullo strumento `rmic`, esaminare lo strumento rmic di Sun Microsystems, Inc.

Strumento rmid Java:

In J2SE (Java 2 Platform, Standard Edition), lo strumento `rmid` avvia il daemon del sistema di attivazione in modo che sia possibile registrare e attivare gli oggetti in una JVM (Java virtual machine). Esso è compatibile con lo strumento `rmid` fornito da Sun Microsystems, Inc.

Lo strumento `rmid` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

Strumento rmid di Sun Microsystems, Inc.

Strumento rmiregistry Java:

Lo strumento `rmiregistry` avvia un registro oggetto remoto su una porta specificata. Esso è compatibile con lo strumento `rmiregistry` fornito da Sun Microsystems, Inc.

Lo strumento `rmiregistry` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

Strumento rmiregistry di Sun Microsystems, Inc.

Strumento serialver Java:

Lo strumento `serialver` restituisce il numero della versione o l'identificativo univoco di serializzazione per una o più classi. Esso è compatibile con lo strumento `serialver` fornito da Sun Microsystems, Inc.

Lo strumento `serialver` è disponibile utilizzando Qshell Interpreter.

Informazioni correlate

Strumento serialver di Sun Microsystems, Inc.

Servertool Java:

Il`servertool` fornisce un'interfaccia della riga comandi attraverso cui i programmatori delle applicazioni possono registrarsi, annullare la registrazione, avviare e chiudere un server permanente.

Informazioni correlate

servertool di Sun Microsystems, Inc.

Strumento tnameserv Java:

In J2SE (Java 2 Platform, Standard Edition), lo strumento `tnameserv` (Transient Naming Service) fornisce accesso al servizio di denominazione. Esso è compatibile con lo strumento `tnameserv` fornito da Sun Microsystems, Inc.

Lo strumento `tnameserv` è disponibile utilizzando Qshell Interpreter.

Strumento unpack200 Java:

Lo strumento `unpack200` Java decomprime un file `pack200` in un file JAR.

Lo strumento `unpack200` è disponibile solo con JDK 1.5 e versioni successive. Lo strumento `unpack200` è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni, consultare lo strumento unpack200 di Sun Microsystems, Inc. 

Concetti correlati

"Strumento pack200 Java" a pagina 425
Lo strumento `pack200` è un'applicazione Java che comprime un file JAR in un file `pack200`.

Comando Java in Qshell

Il comando `java` in Qshell esegue i programmi Java. Esso risulta compatibile con lo strumento `java` fornito da Sun Microsystems, Inc. con poche eccezioni.

IBM Developer Kit per Java ignora queste opzioni del comando `java` in Qshell.

Opzione	Descrizione
-cs	Questa opzione non è supportata.
-checksource	Questa opzione non è supportata.
-debug	Questa opzione è supportata dal programma di debug interno di System i5.
-noasyncgc	La raccolta di dati inutili è sempre in esecuzione con IBM Developer Kit per Java.
-prof	System i5 dispone di propri strumenti delle prestazioni.
-ss	Questa opzione non è applicabile sulla piattaforma System i5.
-oss	Questa opzione non è applicabile sulla piattaforma System i5.
-t	System i5 utilizza la propria funzione di traccia.
-verify	Verificare sempre sulla piattaforma System i5.
-verifyremote	Verificare sempre sulla piattaforma System i5.
-noverify	Verificare sempre sulla piattaforma System i5.

Sulla piattaforma System i5, l'opzione `-classpath` non sovrascrive il classpath predefinito. Al contrario, viene accodato al classpath predefinito di sistema. L'opzione `-classpath` sostituisce la variabile di ambiente CLASSPATH.

Il comando `java` nella Qshell supporta le opzioni per la piattaforma System i5. Sono qui di seguito indicate le opzioni supportate.

Opzione	Descrizione
-chkpath	Questa opzione controlla l'accesso pubblico alla scrittura agli indirizzari nel CLASSPATH.

Opzione	Descrizione
-Xrun[:]	Viene visualizzato un messaggio indicante la presenza di un programma di servizio e di una stringa di parametri facoltativi per la funzione JVM_OnLoad durante l'avvio della JVM. Quest'opzione è stata dichiarata obsoleta. Utilizzare invece -agentlib: o -agentpath:.
-agentlib:	Indica un programma di servizio i5/OS contenente un agent VM. La VM tenta di caricare il programma di servizio da una libreria i5/OS inclusa nell'elenco librerie durante l'avvio.
-agentpath:	Caricare la libreria dal percorso assoluto che segue questa opzione. Non si verifica l'espansione del nome libreria e le opzioni vengono inoltrate all'agent nel momento dell'avvio.
-javaagent:<jarpath>[=<opzioni>]	Carica agent per il linguaggio di programmazione Java per l'utilizzo con il pacchetto java.lang.instrument. *jarpath* è il percorso per il file JAR agent. *opzioni* sono le opzioni per l'agent. È possibile utilizzare -javaagent:<jarpath>[=<opzioni>] più di una volta sulla stessa riga comandi per creare più agent. Più di un agent può utilizzare lo stesso *jarpath*.

Il comando RUNJVA (Esecuzione Java) nelle informazioni di riferimento sul comando CL descrive dettagliatamente queste nuove opzioni. Le informazioni di riferimento sul comando CL per il comando CRTJVAPGM (Creazione programma Java), il comando DLTJVAPGM (Cancellazione programma Java) e il comando DSPJVAPGM (Visualizzazione programma Java) contengono informazioni sulla gestione dei programmi Java.

Il comando `java` in Qshell è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni sul comando `java` in Qshell, consultare lo strumento java di Sun Microsystems, Inc.

Comandi CL supportati da Java

L'ambiente CL contiene i comandi CL per l'ottimizzazione e la gestione dei programmi Java.

- Il comando ANZJVAPGM (Analisi programma Java) analizza un programma Java, ne elenca le classi e mostra lo stato corrente di ogni classe.
- Il comando ANZJVM (Analisi Java Virtual Machine) richiama e imposta le informazioni in una JVM (Java virtual machine). Tale comando fornisce assistenza nel sottoporre a debug i programmi Java restituendo informazioni sulle classi attive.
- Il comando CHGJVAPGM (Modifica programma Java) modifica gli attributi di un programma Java.
- Il comando CRTJVAPGM (Creazione programma Java) crea un programma Java su un System i da un file di classe Java, un file ZIP o un file JAR.
- Il comando DLTJVAPGM (Cancellazione programma Java) cancella un programma Java System i associato a un file di classe Java, un file ZIP o un file JAR.
- Il comando DSPJVAPGM (Visualizzazione programma Java) visualizza le informazioni su un programma Java.
- Il comando DSPJVMJOB (Visualizzazione lavori Java Virtual Machine) visualizza le informazioni sui lavori JVM attivi per fornire assistenza nella gestione dell'applicazione di PTF (program temporary fix). È possibile trovare maggiori dettagli sul comando DSPJVMJOB nell'argomento "Applicazione delle PTF (program temporary fix)" a pagina 577.

- Il comando DMPJVM (Esecuzione del dump di Java Virtual Machine) esegue il dump su informazioni relative alla JVM (Java virtual machine) per un lavoro specificato su un file di stampa di spool.
- Il comando GENJVMDMP (Generazione dump JVM) genera i dump JVM (Java Virtual Machine) su richiesta.
- Il comando PRTJVMJOB (Stampa lavoro JVM) consente di stampare le JVM (Java Virtual Machine) in esecuzione nei lavori attivi.
- Il comando JAVA e il comando RUNJVA (Esecuzione Java) eseguono i programmi System i Java.
- WRKJVMJOB (Gestione lavori JVM) visualizza le informazioni sui lavori in esecuzione in IBM Technology per Java Virtual Machine.

  Stringhe del parametro LICOPT (Licensed Internal Code Option)

  API del comando CL e del programma

Considerazioni per l'utilizzo del comando ANZJVM

A causa della durata dell'esecuzione del comando ANZJVM, è molto probabile che la JVM venga chiusa prima che ANZJVM possa finire. Nel caso in cui JVM venga chiusa, ANZJVM restituisce il messaggio JVAB606 (cioè JVM è stata chiusa durante l'elaborazione di ANZJVM) insieme ai dati che ha potuto ottenere.

Non esistono limiti superiori sul numero di classi che JVM può gestire. Se esistono più classi che è possibile gestire, è opportuno che ANZJVM restituisca i dati gestibili con un messaggio in cui si evidenzia che esistono informazioni aggiuntive non notificate. Quando è necessario troncare i dati, ANZJVM restituisce il maggior numero possibile di informazioni.

La lunghezza del parametro interno è limitata a 3600 secondi (un'ora). Il numero di classi su cui ANZJVM può restituire informazioni è limitato dalla quantità di memoria sul proprio sistema.

Comandi System i Navigator supportati da Java

System i Navigator è un'interfaccia grafica per il desktop Windows. È parte di System i Access per Windows e copre molte funzioni di i5/OS di cui i responsabili o gli utenti hanno bisogno per svolgere il loro lavoro giornaliero. È possibile utilizzare i comandi System i Navigator per creare ed eseguire i programmi Java.

System i Navigator supporta Java come un modulo aggiuntivo contenuto nell'opzione File Systems di System i Access per Windows. Per utilizzare il modulo aggiuntivo System i Navigator Java, è necessario installare IBM Developer Kit per Java sul server. Successivamente, per installare il modulo aggiuntivo Java sul PC (personal computer), selezionare File Systems tramite Installazione selettiva nella cartella Client Access.

I file della classe, JAR, ZIP e Java si trovano nell'integrated file system. System i Navigator consente di visualizzare questi file nel pannello di destra. Fare clic con il tasto destro del mouse sul file di classe, JAR, ZIP o java che si desidera utilizzare. Ciò attiva un menu di contesto.

Selezionando **Programma Java associato --> Nuovo...** dal menu di contesto, si avvia il programma di conversione Java, che crea i programmi System i Java associati al file di classe, JAR o ZIP. Una finestra di dialogo permette di specificare i dettagli su come creare il programma. È possibile creare i programmi per la conversione Java o per l'interpretazione Java.

Nota: se si seleziona la conversione, i bytecode nel file di classe si trasformano in istruzioni RISC che determinano migliori prestazioni rispetto a quelle ottenute se si utilizza l'interpretazione.

Selezionando **Programma Java associato --> Modifica...** dal menu di contesto, è possibile modificare gli attributi dei programmi Java collegati ai file di classe Java, file ZIP oppure file JAR.

Selezionando **Programma Java associato --> Esegui...** dal menu di contesto, viene eseguito il file di classe sul server. È possibile inoltre selezionare un file JAR o ZIP ed eseguire un file di classe ubicato all'interno di quel file JAR o ZIP. Una finestra di dialogo viene visualizzata per permettere all'utente di specificare i dettagli su come eseguire il programma. Se si è già selezionato **Programma Java associato --> Nuovo...**, durante l'esecuzione del programma viene utilizzato il programma System i Java associato al file di classe. Se un programma System i Java non è già associato al file di classe, il programma System i Java viene creato prima dell'esecuzione del programma.

Selezionando **Programma Java associato --> Cancella...** dal menu di contesto, vengono cancellati i programmi System i Java associati al file di classe, JAR o ZIP.

Selezionando **Proprietà** dal menu di contesto, viene visualizzata una casella di dialogo delle proprietà che contiene i separatori **Programmi Java** e **Opzioni Java** . Questi separatori consentono di visualizzare i dettagli sul modo in cui i programmi System i Java associati sono stati creati per il file di classe, JAR o ZIP.

Nota: questi pannelli rappresentano le informazioni sulla visualizzazione dei programmi Java.

Selezionando **Compila file Java** dal menu di contesto, tutti i file java selezionati vengono convertiti nei relativi bytecode del file di classe.

Consultare le informazioni di aiuto incluse in System i Navigator per i parametri e le opzioni delle finestre di dialogo System i Navigator **Nuovo programma Java**, **Modifica programma Java**, **Esegui programma Java**, **Programmi Java**, **Opzioni Java**, **Compila file Java** e **Cancella il programma Java**.

Esecuzione del debug dei programmi Java su i5/OS

Sono disponibili varie opzioni per eseguire il debug e risolvere i problemi dei programmi Java eseguiti sul sistema. compresi IBM System i5 Debugger, il pannello interattivo del sistema, i programmi di debug abilitati a Java Debug Wire Protocol e Heap Analysis Tools for Java.

Le seguenti informazioni non forniscono un quadro esaustivo di tutte le possibilità ma elencano varie opzioni.

Uno dei modi più facili per eseguire il debug dei programmi Java eseguiti sul sistema consiste nell'utilizzare IBM System i5 Debugger. IBM System i5 Debugger fornisce una GUI (graphical user interface) che consente di utilizzare più facilmente le funzionalità di debug del server. È possibile utilizzare il pannello interattivo del server per eseguire il debug dei programmi Java, anche se System i5 Debugger fornisce una GUI di più facile utilizzo che consente di eseguire le stesse funzioni.

Inoltre, JVM (i5/OS Java virtual machine) supporta JDWP (Java Debug Wire Protocol), che fa parte di JPDA (Java Platform Debugger Architecture). I programmi di debug abilitati JDWP consentono di effettuare il debug remoto da client in esecuzione su diversi sistemi operativi. (IBM System i5 Debugger consente anche di eseguire il debug remoto in modo analogo, anche se non utilizza JDWP). Un programma abilitato a JDWP di questo tipo è il programma di debug Java nella piattaforma strumento universale di progetto Eclipse.

Se le prestazioni del programma peggiorano dopo che questo è stato in esecuzione più a lungo, è possibile aver codificato inavvertitamente una perdita di memoria. È possibile utilizzare Heap Analysis Tools for Java come ausilio nell'esecuzione del debug del programma e nell'individuazione delle perdite di memoria eseguendo l'analisi dell'heap dell'applicazione Java e il profilo di creazione oggetto nel tempo.

IBM System i5 Debugger

"JPDA (Java Platform Debugger Architecture)" a pagina 441
JPDA (Java Platform Debugger Architecture) consiste nella JVM Debug Interface/JVM Tool Interface,

nel Java Debug Wire Protocol e nella Java Debug Interface. Tutte queste parti di JPDA abilitano qualsiasi interfaccia di un programma di debug che utilizza il JDWP per eseguire le operazioni di debug. La suddetta interfaccia può essere eseguita in modalità remota o come applicazione System i5.

JDT (Java development tooling) debug

Eclipse project Web site

Heap Analysis Tools for Java

Esecuzione del debug dei programmi Java utilizzando System i5 Debugger

Il modo più facile per eseguire i debug dei programmi Java in esecuzione sul sistema consiste nell'utilizzare IBM System i5 Debugger. IBM System i5 Debugger fornisce una GUI che consente di utilizzare più facilmente le funzionalità di debug del sistema.

Per ulteriori informazioni sull'utilizzo di System i5 Debugger per eseguire il debug e per testare i programmi Java in esecuzione sul server, consultare IBM System i5 Debugger.

Debug di sistema per IBM Technology for Java

Queste istruzioni presentano varie opzioni per l'esecuzione del debug delle JVM IBM Technology for Java.

Debug interattivo dalla riga comandi CL

Il modo più semplice per avviare il programma di debug del sistema consiste nell'utilizzare il parametro OPTION(*DEBUG) del comando CL JAVA. Ad esempio:

```
> JAVA CLASS(Hello) OPTION(*DEBUG)
```

Abilitazione del debug per la JVM IBM Technology for Java

Per eseguire il debug di un lavoro JVM IBM Technology for Java da un altro lavoro, la JVM deve essere avviata con il debug abilitato. Il debug di Java è gestito da un agent di debug. Avviare questo agent durante l'avvio della JVM è la chiave per eseguire correttamente il debug del codice Java. Dopo che la JVM è stata correttamente avviata con l'agent di debug, è possibile eseguire il debug della JVM utilizzando il comando STRSRVJOB (Avvio lavoro di servizio) ed il comando STRDBG (Avvio debug) oppure dalla GUI Debugger System i5. Le seguenti sezioni descrivono vari modi per avviare l'agent di debug. In ogni caso, lo scopo di un parametro o di una variabile di ambiente è quello di indicare che l'agent di debug Java deve essere avviato. Queste descrizioni iniziano con le situazioni più semplici e avanzano a situazioni più complicate.

Nota:

- L'agent di debug è necessario solo per la JVM IBM Technology for Java. Non deve essere avviato per la JVM Classic.
- L'agent di debug non sospende la JVM prima di accedere al metodo principale. Per i programmi a breve esecuzione oppure per eseguire il debug del metodo principale, potrebbe essere necessario del codice Java aggiuntivo per arrestare la JVM. Un modo per eseguire tale operazione consiste nell'utilizzare un loop di attesa a tempo. Un altro modo consiste nel leggere dall'immissione standard.
- Se il debug viene tentato su una JVM che non ha l'agent di debug abilitato, alla registrazione lavori viene inviato un messaggio di diagnostica JVAB307 sia del lavoro JVM che del lavoro di servizio. Il testo del messaggio identifica il lavoro JVM che non ha il debug abilitato. Questo messaggio indica che la JVM deve essere riavviata perché sia possibile eseguirne il debug correttamente. Non è possibile abilitare il debug dopo che la JVM è stata avviata.

Abilitazione del debug Java dalla CL

Per abilitare il debug Java dalla CL, aggiungere il parametro `AGTPGM(D9TI)` al comando CL JAVA. Ad esempio:

```
> JAVA CLASS(Hello) AGTPGM(D9TI)
```

Abilitazione del debug Java dalla Qshell o dal terminale PASE

Per abilitare il debug Java dalla Qshell (QSH) o dal terminale PASE (QP2TERM), aggiungere il parametro `-debug` nel richiamo `java`. Ad esempio:

```
> java -debug Hello
```

Utilizzare il parametro `-debug` è il modo più semplice per avviare l'agent di debug. Equivale ad aggiungere il parametro `-agentlib:d9ti`. L'agent di debug verrà avviato anche da:

```
> java -agentlib:d9ti Hello
```

Abilitazione del debug Java per una JVM di lavoro batch

Se la JVM di lavoro batch viene avviata con il comando CL SBMJOB (Inoltro lavoro), il parametro `AGTPGM(D9TI)` può essere aggiunto al comando CL JAVA. Ad esempio, quanto segue avvierà la JVM di lavoro batch con un agent di debug:

```
> SBMJOB CMD(JAVA CLASS(HELLO) AGTPGM(D9TI))
         CPYENVVAR(*YES) ALWMLTTHD(*YES)
```

Se il lavoro batch viene avviato in qualche altro modo, è possibile utilizzare la variabile di ambiente JAVA_TOOL_OPTIONS per avviare l'agent di debug. La variabile di ambiente JAVA_TOOL_OPTIONS viene interrogata automaticamente dalla JVM durante l'avvio. Se è impostata su *-debug* o *-agentlib:d9ti*, l'agent di debug verrà avviato per la JVM. Ad esempio, una delle seguenti può essere utilizzata per impostare la variabile di ambiente:

```
> ADDENVVAR ENVVAR(JAVA_TOOL_OPTIONS) VALUE('-debug')
> ADDENVVAR ENVVAR(JAVA_TOOL_OPTIONS) VALUE('-agentlib:d9ti')
```

Se il lavoro batch non eredita automaticamente tutte le variabili di ambiente, la variabile di ambiente JAVA_TOOL_OPTIONS dovrà essere impostata per tutto il sistema. Ad esempio:

```
> ADDENVVAR ENVVAR(JAVA_TOOL_OPTIONS) VALUE('-debug') LEVEL(*SYS)
```

Nota: quando si imposta la variabile di ambiente JAVA_TOOL_OPTIONS per tutto il sistema, tutte le JVM IBM Technology for Java avviate sul sistema vengono avviate con il debug abilitato. **Questo può causare una notevole riduzione delle prestazioni**.

Abilitazione del debug Java per una JVM creata con la API di richiamo Java

Quando si utilizza la API C/C++ JNI_CreateJavaVM per creare una JVM, è possibile abilitare il debug utilizzando uno qualsiasi dei seguenti metodi:

- Impostare la variabile di ambiente JAVA_TOOL_OPTIONS su *-debug*.
- Impostare la variabile di ambiente JAVA_TOOL_OPTIONS su *-agentlib:d9ti*.
- Aggiungere il parametro `-debug` all'elenco dei parametri delle opzioni passato alla API JNI_CreateJavaVM C/C++.
- Aggiungere il parametro `-agentlib:d9ti` all'elenco dei parametri delle opzioni passato alla API C/C++ JNI_CreateJavaVM.

Inoltre, per vedere il codice sorgente Java per le classi oggetto del debug, potrebbe essere necessario impostare la variabile di ambiente DEBUGSOURCEPATH sull'ubicazione di indirizzario di base del codice sorgente Java.

Avvio della JVM IBM Technology for Java dalla GUI (graphical user interface) System i5 Debugger

Per avviare una JVM IBM Technology for Java dalla GUI (graphical user interface) System i5 Debugger, è necessario impostare la variabile di ambiente JAVA_HOME quando si avvia il lavoro JVM. È possibile impostare questa variabile di ambiente utilizzando il pannello **Comando di inizializzazione** quando si avvia la JVM. Questo pannello si trova nella finestra Avvia debug dell'interfaccia System i5 Debugger.

Ad esempio, per avviare una JVM JDK 5.0 a 32 bit, aggiungere quanto segue al pannello **Comando di inizializzazione**:

```
ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit')
```

Nota: I punti di osservazione per le variabili locali non sono supportati nella JVM IBM Technology for Java. L'implementazione Debug sistema per la JVM IBM Technology for Java utilizza JVMTI, che non fornisce la funzionalità di punti di osservazione per le variabili locali.

Informazioni correlate

System i5 Debugger

Debug di sistema per le JVM Classic

Queste istruzioni presentano varie opzioni per il debug delle JVM Classic (Java Virtual Machine).

Debug interattivo dalla riga comandi CL

Il modo più semplice per avviare il programma di debug del sistema consiste nell'utilizzare il parametro OPTION(*DEBUG) del comando CL JAVA. Ad esempio:

```
 > JAVA CLASS(Hello) OPTION(*DEBUG)
```

Abilitazione del debug per la JVM Classic

Per eseguire il debug di un lavoro di una JVM Classic da un altro lavoro, la JVM deve essere avviata con il compilatore JIT (Just-in-Time) disabilitato. Dopo che la JVM è stata correttamente avviata con il compilatore JIT disabilitato, è possibile eseguire il debug della JVM utilizzando il comando STRSRVJOB (Avvio lavoro di servizio) ed il comando STRDBG (Avvio debug) oppure dalla GUI Debugger System i5. Le seguenti sezioni descrivono vari modi per disabilitare il compilatore JIT.

Nota:

- La disabilitazione del compilatore JIT è necessaria solo per la JVM Classic.
- Queste istruzioni non sospendono la JVM prima dell'accesso al metodo principale. Per i programmi a breve esecuzione oppure per eseguire il debug del metodo principale, potrebbe essere necessario del codice Java aggiuntivo per arrestare la JVM. Un modo per eseguire tale operazione consiste nell'utilizzare un loop di attesa a tempo. Un altro modo consiste nel leggere dall'immissione standard.
- Se viene tentato il debug su una JVM che sta utilizzando il compilatore JIT, le funzioni di debug non funzioneranno come previsto. Non è possibile disabilitare il compilatore JIT dopo che la JVM è stata avviata.

Abilitazione del debug Java dalla CL

Per abilitare il debug Java dalla CL, aggiungere il parametro `PROP((java.compiler NONE))` al comando CL JAVA. Ad esempio:

```
>  JAVA CLASS(Hello) PROP((java.compiler NONE))
```

Abilitazione del debug Java dalla Qshell o dal terminale PASE

| Per abilitare il debug Java dalla Qshell (QSH) o dal terminale PASE (QP2TERM), aggiungere il parametro
| `-Djava.compiler=NONE` nel richiamo `java`. Ad esempio:

```
| > java -Djava.compiler=NONE Hello
```

| Abilitazione del debug Java per una JVM di lavoro batch

| Se la JVM di lavoro batch viene avviata dal comando CL SBMJOB (Inoltro lavoro), il parametro
| `PROP((java.compiler NONE))` può essere aggiunto al comando CL JAVA. Ad esempio, quanto segue
| avvierà la JVM di lavoro batch con un agent di debug:

```
| > SBMJOB CMD(JAVA CLASS(HELLO) PROP((java.compiler NONE)))
|         CPYENVVAR(*YES) ALWMLTTHD(*YES)
```

| Abilitazione del debug Java per una JVM creata con la API di richiamo Java

| Quando si utilizza la API C/C++ JNI_CreateJavaVM per creare una JVM, è possibile disabilitare il JIT
| utilizzando un file SystemDefault.properties per impostare `java.compiler=NONE`. Inoltre, per vedere il
| codice sorgente Java per le classi oggetto del debug, potrebbe essere necessario impostare la variabile di
| ambiente DEBUGSOURCEPATH sull'ubicazione di indirizzario di base del codice sorgente Java.

| Avvio della JVM Classic dalla GUI (graphical user interface) System i5 Debugger

| La JVM Classic viene utilizzata per impostazione predefinita quando si avvia Java da System i5
| Debugger. Non impostare la variabile di ambiente JAVA_HOME quando si avvia una JVM Classic.

| **Concetti correlati**

| "File SystemDefault.properties" a pagina 15
| Il file SystemDefault.properties è un file delle proprietà Java standard che consente di specificare le
| proprietà predefinite dell'ambiente Java.

| "Elenco delle proprietà di sistema Java" a pagina 16
| Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste
| sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

| **Informazioni correlate**

| System i5 Debugger

Operazioni di debug

È possibile utilizzare il pannello interattivo del server per utilizzare l'opzione `*DEBUG` per visualizzare il codice sorgente prima di eseguire il programma. Quindi, è possibile impostare i punti di interruzione o oltrepassare o entrare in un programma per analizzare gli errori mentre il programma è in esecuzione.

Esecuzione del debug dei programmi Java utilizzando l'opzione *DEBUG

Per eseguire il debug dei programmi Java utilizzando l'opzione *DEBUG, attenersi alla seguente procedura:

1. Compilare il programma Java utilizzando l'opzione DEBUG, che è l'opzione -g sullo strumento `javac`.
2. Inserire il file di classe (.class) e il file di origine (.java) nello stesso indirizzario sul proprio server.
3. Eseguire il programma Java utilizzando il comando RUNJVA (Esecuzione Java) sulla riga comandi i5/OS. Specificare OPTION(*DEBUG) sul comando RUNJVA (Esecuzione Java). Ad esempio: `RUNJVA CLASS(classname) OPTION(*DEBUG)`

   È possibile effettuare il debug soltanto di una classe. Se viene immesso un nome file JAR per la parola chiave CLASS, OPTION(*DEBUG) non è supportato.
4. Viene visualizzato il sorgente programma Java.
5. Premere F6 (Aggiunta/Eliminazione punto di interruzione) per impostare i punti di interruzione o premere F10 (Avanzamento) per avanzare nel programma.

Nota:

- Durante l'utilizzo dei punti di interruzione e degli avanzamenti, controllare il flusso logico del programma Java, quindi visualizzare e modificare le variabili, come necessario.
- L'utilizzo di OPTION(*DEBUG) sul comando RUNJVA disabilita il compilatore JIT (Just-In-Time).
- se l'utente non è autorizzato a utilizzare il comando STRSRVJOB (Avvio lavoro di servizio), OPTION(*DEBUG) viene ignorato.

Esecuzione del debug dei programmi Java da un altro pannello

Quando si effettua il debug di un programma Java con il pannello interattivo del server, il sorgente programma viene visualizzato ogni qualvolta incontri un punto di interruzione. Ciò può interferire con l'emissione del pannello del programma Java. Per evitare ciò, effettuare il debug del programma Java da un altro pannello. L'emissione dal programma Java viene visualizzata dove è in esecuzione il comando Java e il sorgente programma viene visualizzato sull'altro pannello.

| È anche possibile eseguire il debug di un programma Java già in esecuzione in questo modo a condizione
| che sia stato avviato con il debug abilitato.

| **Nota:** È possibile abilitare il debug Java nei seguenti modi:
| - Se si sta utilizzando la JVM Classic:
| - Specificare la proprietà `java.compiler=NONE` quando si avvia la JVM (Java virtual machine)
| - Specificare INTERPRET(*YES) sul comando RUNJVA (Esecuzione Java)
| - Se si sta utilizzando IBM Technology for Java, l'opzione AGTPGM(D9TI) deve essere aggiunta al comando JAVA/RUNJVA per utilizzare System i5 Debugger con la JVM. AGTPGM(D9TI) non è necessaria quando si utilizza OPTION(*DEBUG).

  Non è possibile specificare l'opzione AGTPGM(D9TI) per la JVM Classic.

| Per effettuare il debug di Java da un altro pannello, eseguire queste operazioni:

1. È necessario congelare il programma Java mentre si avvia l'impostazione per effettuare il debug.

   È possibile congelare il programma Java operando affinché il programma effettui quanto segue:
   - Attendere l'immissione dalla tastiera.
   - Attendere un intervallo di tempo.
   - Eseguire il loop per verificare una variabile, che richiede all'utente di impostare un valore per collocare successivamente il programma Java fuori dal loop.
2. Una volta congelato il programma Java, andare su un altro pannello per eseguire queste fasi:
   a. Immettere il comando WRKACTJOB (Gestione lavori attivi) sulla riga comandi.
   b. Trovare il lavoro BCI (batch immediato) dove è in esecuzione il proprio programma Java. Cercare QJVACMDSRV nell'elenco Sottosistema/Lavoro. Cercare nell'elenco Utente il proprio ID utente. Cercare BCI nell'elenco Tipo.
   c. Immettere l'opzione 5 per gestire tale lavoro.
   d. In alto al pannello di Gestione lavoro, sono visualizzati il Lavoro, l'Utente e il Numero. Immettere `STRSRVJOB Number/User/Job`
   e. Immettere `STRDBG CLASS(classname)`. Classname è il nome della classe Java su cui si desidera effettuare il debug. Può essere sia il nome classe specificato sul comando Java o un'altra classe.
   f. L'origine per tale classe viene visualizzata nel pannello Visualizzazione origine modulo.
   g. Impostare i punti di interruzione, premendo F6 (Aggiunta/Cancellazione punto di interruzione), ogni qualvolta si desidera arrestare tale classe Java. Premere F14 per aggiungere altre classi, programmi o programmi di servizio su cui effettuare il debug.
   h. Premere F12 (Ripresa) per continuare ad eseguire il programma.

3. Arrestare il congelamento del proprio programma Java originale. Quando si attivano i punti di interruzione, viene visualizzato il pannello di Visualizzazione origine modulo sul pannello dove sono stati immessi i comandi STRSRVJOB (Avvio lavoro di servizio) e STRDBG (Avvio debug). Quando il programma Java termina, viene visualizzato un messaggio `Job being serviced ended`.
4. Immettere il comando ENDDBG (Fine debug).
5. Immettere il comando ENDSRVJOB (Fine lavoro di servizio).

Quando si effettua il debug di un programma Java, il proprio programma Java è in effetti in esecuzione nella JVM (Java virtual machine) in un lavoro BCI (batch immediato). Il codice sorgente viene visualizzato nel pannello interattivo, ma il programma Java non è in esecuzione in quell'ambito. Esso è in esecuzione in un altro lavoro, che è servito. Consultare l'argomento relativo alla variabile di ambiente QIBM_CHILD_JOB_SNDINQMSG per ulteriori informazioni su questa variabile che controlla se il lavoro BCI attende prima di chiamare la JVM (Java virtual machine).

Concetti correlati

"Compilatore JIT (Just-In-Time)" a pagina 415
Un compilatore JIT (Just-In-Time) è un compilatore specifico della piattaforma che genera istruzioni di sistema per ogni metodo come necessario.

Informazioni correlate

System i5 Debugger

Visualizzazioni di debug iniziali per i programmi Java:

Quando si effettua il debug dei propri programmi Java seguire queste visualizzazioni di esempi per i propri programmi. Esse mostrano un programma di esempio, denominato Hellod.

- Immettere `ADDENVVAR ENVVAR(CLASSPATH) VALUE ('/MYDIR')`.
- Immettere questo comando: `RUNJVA CLASS(HELLOD) OPTION(*DEBUG)`. Inserire il nome del proprio programma Java al posto di HELLOD.
- Attendere la visualizzazione del pannello Visualizzazione origine modulo. Questa è l'origine per il programma Java HELLOD.

```
+------------------------------------------------------------------------------+
|                           Visualizzazione origine modulo                     |
|                                                                              |
|  Nome file di classe:   HELLOD                                               |
|       1  import java.lang.*;                                                 |
|       2                                                                      |
|       3  public class Hellod extends Object                                  |
|       4  {                                                                   |
|       5  int k;                                                              |
|       6  int l;                                                              |
|       7  int m;                                                              |
|       8  int n;                                                              |
|       9  int o;                                                              |
|      10  int p;                                                              |
|      11  String myString;                                                    |
|      12  Hellod myHellod;                                                    |
|      13  int myArray[];                                                      |
|      14                                                                      |
|      15  public Hellod()                                                     |
|                                                                  Segue...    |
|  Debug . . .                                                                 |
|                                                                              |
|  F3=Fine progr.   F6=Agg./Elim. punti int.  F10=Step   F11=Visual. variabile |
|  F12=Ripresa      F17=Vis. variabile   F18=Gestione visual.  F24=Altri tasti |
|                                                                              |
+------------------------------------------------------------------------------+
```

- Premere F14 (Gestione lista moduli).

- Viene visualizzato il pannello di Gestione lista moduli. È possibile aggiungere altre classi e programmi per effettuare il debug immettendo l'opzione 1 (Aggiunta programma). Visualizzare la relativa origine con l'opzione 5 (Visualizzazione origine modulo).

```
+-------------------------------------------------------------------------------+
|                              Gestione lista moduli                            |
|                                                  Sistema:   AS400             |
|  Immettere le opzioni e premere Invio.                                        |
|    1=Agg. program. 4=Elimin. program. 5=Visualizzaz. origine modulo           |
|    8=Gestione punti di interruzione modulo                                    |
|                                                                               |
|  Opz   Programma/modulo     Libreria         Tipo                             |
|                             *LIBL            *SRVPGM                          |
|          HELLOD                              *CLASS      Selezionato          |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                    Fine       |
|  Comando                                                                      |
|  ===>                                                                         |
|  F3=Fine   F4=Richiesta  F5=Rivisual. F9=Duplicaz.  F12=Annullamento          |
|  F22=Visual. nome file classe                                                 |
|                                                                               |
+-------------------------------------------------------------------------------+
```

- Quando si aggiunge una classe di cui effettuare il debug, è possibile che sia necessario immettere il nome classe qualificato del pacchetto che è più lungo del campo di immissione Programma/modulo. Per immettere un nome più lungo, seguire queste fasi:
  1. Immettere l'opzione 1 (Aggiunta programma).
  2. Lasciare vuoto il campo Programma/modulo.
  3. Lasciare il campo libreria come *LIBL.
  4. Immettere *CLASS per il Tipo.
  5. Premere Invio.
  6. Viene visualizzata una finestra a comparsa dove si dispone di maggiore spazio per immettere il nome file di classe qualificato del pacchetto. Ad esempio: `pkgname1.pkgname2.classname`

Impostazioni dei punti di interruzione:

È possibile controllare l'esecuzione di un programma con i punti di interruzione. I punti di interruzione arrestano un programma in esecuzione a un'istruzione specifica.

Per impostare i punti di interruzione, effettuare quanto segue:

1. Posizionare il cursore sulla riga del codice in cui si desidera impostare un punto di interruzione.
2. Premere F6 (Aggiunta/Eliminazione punti di interruzione) per impostare il punto di interruzione.
3. Premere F12 (Ripresa) per eseguire il programma.

Nota: subito prima che la riga del codice sia in esecuzione, dove è impostato il punto di interruzione, si visualizza il sorgente programma che indica l'avvenuta corrispondenza con il punto di interruzione.

```
+-------------------------------------------------------------------------------+
|                           Visualizzazione origine modulo                      |
|                                                                               |
|Sottop. corrente: 00000019     Sottop. arrestato:00000019                      |
|Nome file classe:  Hellod                                                      |
```

```
35  public static void main(String[] args)
36      {
37    int i,j,h,B[],D[][];
38      Hellod A=new Hellod();
39      A.myHellod = A;
40      Hellod C[];
41     C = new Hellod[5];
42      for (int counter=0; counter<2; counter++) {
43        C[counter] = new Hellod();
44        C[counter].myHellod = C[counter];
45      }
46      C[2] = A;
47      C[0].myString = null;
48      C[0].myHellod = null;

49      A.method1();
Debug . . .

F3=Fine progr.   F6=Agg./Elim. punti int.  F10=Fase   F11=Visual. variabile
F12=Ripresa  F17=Visual. variab.  F18=Gestione visual.  F24=Altri tasti
Punto di interruzione aggiunto alla riga 41
```

Dopo il rilevamento di un punto di interruzione o il completamento di un passo, è possibile utilizzare il comando TBREAK per impostare un punto di interruzione che è valido solo per il sottoprocesso corrente.

Esecuzione delle istruzioni dei programmi Java:

È possibile eseguire le istruzioni del programma durante l'esecuzione del debug. È possibile ignorare o eseguire dettagliatamente altre funzioni. I programmi Java e i metodi nativi possono utilizzare la funzione di esecuzione istruzione.

Quando il sorgente programma viene visualizzato per la prima volta è possibile avviare la funzione di esecuzione istruzione. Il programma si arresta prima di eseguire la prima istruzione. Premere F10 (Fase). Continuare a premere F10 (Fase) per eseguire le istruzioni del programma. Premere F22 (Esecuzione dettagliata) per eseguire dettagliatamente qualsiasi funzione chiamata dal programma. È possibile inoltre avviare la funzione di esecuzione istruzione ogni volta che viene raggiunto un punto di interruzione. Per informazioni sull'impostazione di punti di interruzione, consultare l'argomento Impostare punti di interruzione.

```
                          Visualizzazione origine modulo

Sottop. corrente: 00000019     Sottop. arrestato:00000019
Nome file classe:  Hellod
35  public static void main(String[] args)
36      {
37    int i,j,h,B[],D[][];
38      Hellod A=new Hellod();
39      A.myHellod = A;
40      Hellod C[];
41     C = new Hellod[5];
42      for (int counter=0; counter<2; counter++) {
43        C[counter] = new Hellod();
44        C[counter].myHellod = C[counter];
45      }
46      C[2] = A;
47      C[0].myString = null;
48      C[0].myHellod = null;
49      A.method1();
Debug . . .

F3=Fine progr.   F6=Agg./Elim. punti int.  F10=Fase   F11=Visual. variabile
```

```
|F12=Ripresa  F17=Visual. variab.  F18=Gestione visual.  F24=Altri tasti          |
|Fase completata alla riga 42 nel sottoprocesso 00000019                         |
+--------------------------------------------------------------------------------+
```

Per continuare ad eseguire il programma, premere F12 (Ripresa).

Valutazione di variabili nei programmi Java:

Esistono due modi per determinare il valore di una variabile quando un programma arresta l'esecuzione in un punto di interruzione o in una fase.

- Opzione 1: immettere `EVAL VariableName` sulla riga comandi del debug.
- Opzione 2: posizionare il cursore sul nome della variabile nel codice sorgente visualizzato e premere F11 (Visualizzazione variabile).

Nota: è inoltre possibile modificare il contenuto di una variabile utilizzando il comando EVAL. Per ulteriori informazioni sulle variazioni del comando EVAL, consultare WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712 e le informazioni della guida in linea.

Quando si esaminano le variabili in un programma Java, si noti quanto segue:

- Se si determina il valore di una variabile che è un'istanza di una classe Java la prima riga del pannello visualizza il tipo di oggetto. Essa visualizza inoltre un identificativo per l'oggetto. Seguendo la prima riga del pannello, viene visualizzato il contenuto per ogni campo nell'oggetto. Se la variabile è null, la prima riga del pannello indica tale valore. Gli asterischi mostrano il contenuto di ogni campo (di un oggetto null).
- Se si determina il valore di una variabile che è un oggetto stringa Java viene visualizzato il contenuto di tale stringa. Se la stringa è null, viene visualizzato null.
- Non è possibile modificare una variabile che è una stringa o un oggetto.
- Se si determina il valore di una variabile che è una schiera, viene visualizzato 'ARR' seguito da un identificativo per tale schiera. È possibile determinare il valore degli elementi di una schiera utilizzando un subscript del nome della variabile. Se la schiera è null, viene visualizzato null.
- Non è possibile modificare una variabile che è una schiera. È possibile modificare un elemento di una schiera se questa non è una schiera di stringhe o oggetti.
- Per le variabili che sono schiere, è possibile specificare arrayname.length per vedere di quanti elementi si compone la schiera.
- Se si desidera vedere il contenuto di una variabile che è un campo di una classe, è possibile specificare classvariable.fieldname.
- Se si tenta di determinare il valore di una variabile prima che sia stata inizializzata, può verificarsi una delle seguenti situazioni. Viene visualizzato un messaggio `Variabile non disponibile da visualizzare` o viene visualizzato il contenuto non inizializzato della variabile, che potrebbe essere un valore anomalo.

Debug dei programmi Java e del metodo nativo:

È possibile effettuare il debug dei programmi Java e dei programmi del metodo nativo contemporaneamente. Durante il debug della propria origine sul pannello interattivo, è possibile effettuare il debug di un metodo nativo programmato in C, contenuto all'interno di un programma di servizio (*SRVPGM). È necessario compilare *SRVPGM e crearlo con i dati di debug.

Per utilizzare il pannello interattivo del server per effettuare il debug dei programmi Java e dei programmi del metodo nativo contemporaneamente, procedere come segue:

1. Premere F14 (Gestione elenco moduli) quando il sorgente programma Java viene visualizzato per mostrare il pannello WRKMODLST (Gestione elenco moduli).
2. Selezionare l'opzione 1 (Aggiunta programma) per aggiungere il proprio programma di servizio.

3. Selezionare l'opzione 5 (Visualizzazione origine modulo) per visualizzare il *MODULE che si intende sottoporre a debug e l'origine.
4. Premere F6 (Aggiunta/Eliminazione punto di interruzione) per impostare i punti di interruzione nel programma di servizio. Per ulteriori informazioni sull'impostazione dei punti di interruzione, consultare "Impostazioni dei punti di interruzione" a pagina 437.
5. Premere F12 (Ripresa) per eseguire il programma.

Nota: quando viene attivato il punto di interruzione nel proprio programma di servizio, il programma arresta l'esecuzione e viene visualizzata l'origine per il programma di servizio.

Utilizzo della variabile di ambiente QIBM_CHILD_JOB_SNDINQMSG per il debug

La variabile di ambiente QIBM_CHILD_JOB_SNDINQMSG controlla se il lavoro BCI (batch immediato), nel quale è in esecuzione la Java virtual machine, è in attesa prima dell'avvio della Java virtual machine.

Se si imposta la variabile di ambiente su un valore di 1 quando il comando RUNJVA (Esecuzione Java) è in esecuzione, viene inviato un messaggio alla coda messaggi dell'utente. Il messaggio è inviato prima che la Java virtual machine venga avviata nel lavoro BCI. Il messaggio risulta in questo modo:

```
Spawned (child) process 023173/JOB/QJVACMDSRV is stopped (G C)
```

Per visualizzare questo messaggio, immettere SYSREQ e selezionare l'opzione 4.

Il lavoro BCI attende finché non viene immessa una risposta a questo messaggio. Una risposta con (G) avvia la Java virtual machine.

È possibile impostare i punti di interruzione in un *SRVPGM o *PGM, che il lavoro BCI chiama, prima di rispondere al messaggio.

Nota: non è possibile impostare punti di interruzione in una classe Java, perché in quel punto, la Java virtual machine non è stata avviata.

Esecuzione del debug delle classi Java caricate tramite un programma di caricamento delle classi personalizzato

Per utilizzare il pannello interattivo del server per eseguire il debug di una classe caricata tramite un programma di caricamento delle classi personalizzato, completare la seguente procedura.

1. Impostare la variabile di ambiente DEBUGSOURCEPATH nell'indirizzario contenente il codice sorgente o, nel caso di una classe qualificata di pacchetto, l'indirizzario iniziale dei nomi pacchetto.

   Ad esempio, se il programma di caricamento classi personalizzato carica delle classi individuate nell'indirizzario /MYDIR, effettuare quanto segue:

   ```
   ADDENVVAR ENVVAR(DEBUGSOURCEPATH) VALUE('/MYDIR')
   ```

2. Aggiungere la classe alla vista di debug dallo schermo Visualizzazione origine modulo.

   Se la classe è già stata caricata nella JVM (Java virtual machine), aggiungere normalmente solo *CLASS e visualizzare il codice sorgente su cui effettuare il debug.

   Ad esempio, per visualizzare l'origine per pkg1/test14.class, immettere quanto segue:

   ```
   Opt     Program/module     Library        Type
   1       pkg1.test14_       *LIBL          *CLASS
   ```

   Se la classe non è stata caricata nella JVM, eseguire le stesse fasi per aggiungere *CLASS come precedentemente indicato. Viene quindi visualizzato il messaggio **File di classe Java non disponibile**. A questo punto, è possibile riprendere l'elaborazione del programma. La JVM viene arrestata automaticamente quando viene immesso ogni metodo della classe corrispondente al nome fornito. Il codice sorgente per la classe viene visualizzato ed è possibile effettuare il debug.

Esecuzione del debug dei servlet

Il debug dei servlet è un tipo speciale del debug delle classi caricate tramite un programma di caricamento classi personalizzato. I servlet vengono eseguiti nel tempo di esecuzione Java del server HTTP IBM. Il debug dei servlet può essere effettuato in vari modi.

È possibile eseguire il debug dei servlet seguendo le istruzioni per le classi caricate tramite il programma di caricamento classi personalizzato.

Per eseguire il debug del servlet, è inoltre possibile utilizzare il pannello interattivo del server seguendo le istruzioni di seguito riportate:

1. Utilizzare il comando `javac -g` in Qshell Interpreter per compilare il proprio servlet.
2. Copiare il codice sorgente (file .java) e il codice compilato (file .class) in /QIBM/ProdData/Java400.
3. Eseguire il comando CRTJVAPGM (Creazione programma Java) rispetto al file .class file utilizzando il livello di ottimizzazione 10, OPTIMIZE(10).
4. Avviare il server.
5. Eseguire il comando STRSRVJOB (Avvio lavoro di servizio) sul lavoro dove viene eseguito il servlet.
6. Immettere STRDBG CLASS(myServlet), dove myServlet è il nome del proprio servlet. Sarebbe opportuno visualizzare l'origine.
7. Impostare un punto di interruzione nel servlet e premere F12.
8. Eseguire il proprio servlet. Quando il servlet attiva il punto di interruzione, è possibile continuare il debug.

Un altro modo per eseguire il debug dei programmi e dei servlet Java eseguiti sul sistema consiste nell'utilizzare IBM System i5 Debugger. System i5 Debugger fornisce una GUI (graphical user interface) che abilita l'utente ad utilizzare più facilmente le funzionalità di debug del sistema.

Informazioni correlate

System i5 Debugger

JPDA (Java Platform Debugger Architecture)

JPDA (Java Platform Debugger Architecture) consiste nella JVM Debug Interface/JVM Tool Interface, nel Java Debug Wire Protocol e nella Java Debug Interface. Tutte queste parti di JPDA abilitano qualsiasi interfaccia di un programma di debug che utilizza il JDWP per eseguire le operazioni di debug. La suddetta interfaccia può essere eseguita in modalità remota o come applicazione System i5.

JVMDI (Java Virtual Machine Tool Interface)

La JVMTI sostituisce la JVMDI (Java Virtual Machine Debug Interface) e la JVMPI (Java Virtual Machine Profiler Interface). JVMTI contiene tutta la funzionalità di JVMDI e di JVMPI, oltre alle nuove funzioni. La JVMTI è stata aggiunta come parte di J2SE 5.0. In JDK 6, le interfacce JVMDI e JVMPI non sono più offerte e JVMTI è la sola opzione disponibile.

Per ulteriori informazioni sull'utilizzo della JVMTI, consultare la pagina di guida di riferimento alla JVMTI sul sito web di Sun Microsystems, Inc.

JVMDI (Java) (solo per JDK 1.4)

In J2SDK (Java 2 SDK), Standard Edition, la JVMDI fa parte delle API (application program interface) della piattaforma Sun Microsystems, Inc.. La JVMDI consente a chiunque di scrivere un programma di debug Java per un System i5 in codice C. Non è necessario che il programma di debug conosca la struttura interna della JVM (Java virtual machine) dato che utilizza interfacce JVMDI. JVMDI è l'interfaccia di livello più basso in JPDA più vicina alla JVM (Java virtual machine).

JDWP (Java Debug Wire Protocol)

Il JDWP (Java Debug Wire Protocol) è un protocollo di comunicazione definito tra un processo del programma di debug e la JVMDI/JVMTI. È possibile utilizzare JDWP da un sistema remoto o su un socket locale. Esso è un livello eliminato dalla JVMDI/JVMTI.

Avvio di JDWP in QShell

Per avviare JDWP ed eseguire la classe Java SomeClass, immettere il seguente comando in QShell:

```
java -interpret -agentlib:jdwp=transport=dt_socket,
address=8000,server=y,suspend=n SomeClass
```

In questo esempio, JDWP è in ascolto per collegamenti dai programmi di debug remoti sulla porta 8000 TCP/IP, ma è possibile utilizzare qualsiasi numero di porta si desideri; dt_socket è il nome del SRVPGM che gestisce il trasferimento JDWP e non cambia.

Per opzioni aggiuntive che è possibile utilizzare con -agentlib, consultare la pagina relativa alle opzioni di richiamo della VM Sun di Sun Microsystems, Inc. Queste opzioni sono disponibili sia per JDK 1.4 che per 1.5 su i5/OS.

Avvio di JDWP da una riga comandi CL

Per avviare JDWP con il comando CL, sono fornite due nuove opzioni: AGTPGM e AGTOPTIONS.

Il valore di AGTPGM è JDWP e si può definire il valore di AGTOPTIONS in modo che sia la stessa stringa che si sarebbe utilizzata sulla riga comandi QShell.

Per avviare JDWP ed eseguire la classe Java SomeClass, immettere il seguente comando:

```
JAVA CLASS(SomeClass) INTERPRET(*YES) AGTPGM(JDWP)
AGTOPTIONS('transport=dt_socket,address=8000,server=y,suspend=n')
```

JDI (Java Debug Interface)

JDI (Java Debug Interface) è un'interfaccia di linguaggio Java ad alto livello fornita per lo sviluppo di strumenti. JDI nasconde la complessità di JVMDI/JVMTI e JDWP dietro alcune definizioni della classe Java. JDI è inclusa nel file rt.jar, quindi l'interfaccia del programma di debug esiste su qualsiasi piattaforma che dispone di Java installato.

Se si intende scrivere i programmi di debug per Java, sarebbe opportuno utilizzare JDI in quanto è l'interfaccia più semplice e il proprio codice è indipendente dalla piattaforma.

Per ulteriori informazioni su JDPA, consultare Java Platform Debugger Architecture Overview di Sun Microsystems, Inc.

Debug a velocità massima della JVM:

La JVM (i5/OS Java Virtual Machine) supporta il debug a velocità massima.

Il debug a velocità massima permette di eseguire l'applicazione in modo che le prestazioni sfruttino tutti i vantaggi del codice compilato da JIT senza perdere la capacità di eseguire alcune delle più comuni attività di debug, come l'impostazione dei punti di interruzione, l'avanzamento nel codice e la visualizzazione delle variabili locali.

Poiché il debug a velocità massima permette ai metodi di essere compilati da JIT, esistono alcune restrizioni per l'esecuzione del debug:

- Le operazioni di avanzamento sulle istruzioni restituite non funzionano se il chiamante è un codice compilato.
- I punti di osservazione vengono attivati solo nei metodi non compilati che modificano il campo osservato.
- Le variabili locali non possono essere visualizzate per i metodi compilati da JIT.
- La ridefinizione della classe non è consentita.

Nota: questa funzione è supportata solo per i programmi di debug che utilizzano JDWP (Java Debug Wire Protocol) per effettuare le operazioni di debug. Il programma di debug del sistema corrente non supporta il debug a velocità massima.

Ricerca delle perdite di memoria

Se le prestazioni del programma peggiorano dopo che questo è stato in esecuzione più a lungo è possibile aver codificato per errore una perdita di memoria. È possibile utilizzare Heap Analysis Tools for Java come ausilio nell'esecuzione del debug del programma e nell'individuazione delle perdite di memoria eseguendo l'analisi dell'heap dell'applicazione Java e il profilo di creazione oggetto nel tempo.

Per ulteriori dettagli, consultare Heap Analysis Tools for Java.

È inoltre possibile utilizzare il comando CL ANZJVM (Analisi Java Virtual Machine) per ricercare perdite di oggetti. ANZJVM rileva perdite di oggetti acquisendo due copie dell'heap di raccolta di dati inutili separati da un intervallo di tempo specificato. Per rilevare perdite di oggetti, è necessario consultare il numero di istanze di ogni classe nell'heap. È probabile che le classi con un numero di istanze insolitamente alto perdano oggetti.

Sarebbe inoltre opportuno notare il cambio in numero di istanze di ogni classe tra le due copie dell'heap di raccolta di dati inutili. Se il numero di istanze di una classe aumenta continuamente, sarebbe opportuno notare che tale classe possa perdere oggetti. Maggiore è l'intervallo di tempo tra le due copie, maggiore è la certezza che gli oggetti perdano effettivamente. Eseguendo più volte ANZJVM con un intervallo di tempo maggiore, dovrebbe essere possibile individuare in modo abbastanza esatto da quale oggetto provenga la perdita.

Utilizzo del comando Generazione dump JVM

Se si sta utilizzando IBM Technology for Java Virtual Machine, è possibile utilizzare il comando CL GENJVMDMP (Generazione dump JVM) per generare i dump di heap, sistema e Java.

Il comando GENJVMDMP genera i dump JVM (Java Virtual Machine) su richiesta. Le informazioni verranno visualizzate solo per i lavori IBM Technology for Java Virtual Machine. È possibile generare i seguenti tipi di dump:

***JAVA** Genera più file che contengono informazioni di diagnostica per la JVM e le applicazioni Java in esecuzione nella JVM.

***SYSTEM**
Genera un'immagine di memoria grezza in formato binario del lavoro che era in esecuzione quando è stato iniziato il dump.

***HEAP**
Genera un dump di tutte le allocazioni di spazio heap che non sono state ancora rilasciate.

Informazioni correlate

Comando CL GENJVMDMP (Generazione dump JVM)

Esempi di codice per IBM Developer Kit per Java

Segue un elenco di esempi di codice per IBM Developer Kit per Java.

Internazionalizzazione

- "Esempio: internazionalizzazione delle date utilizzando la classe java.util.DateFormat" a pagina 446
- "Esempio: internazionalizzazione del pannello numerico utilizzando la classe java.util.NumberFormat" a pagina 447
- "Esempio: internazionalizzazione di dati specifici della locale utilizzando la classe java.util.ResourceBundle" a pagina 447

JDBC

- "Esempio: proprietà Access" a pagina 448
- "Esempio: BLOB" a pagina 144
- "Esempio: interfaccia CallableStatement per IBM Developer Kit per Java" a pagina 452
- "Esempio: eliminazione di valori da una tabella tramite il cursore di un'altra istruzione" a pagina 125
- "Esempio: CLOB" a pagina 148
- "Esempio: creazione di UDBDataSource e suo collegamento alla JNDI" a pagina 59
- "Esempio: creazione di un UDBDataSource e acquisizione di un ID utente e una parola d'ordine" a pagina 61
- "Esempio: creazione di UDBDataSourceBind ed impostazione delle proprietà DataSource" a pagina 59
- "Esempio: restituzione di un elenco di tabelle utilizzando l'interfaccia DatabaseMetaData IBM Developer Kit per Java" a pagina 70
- "Esempio: creazione di UDBDataSource e suo collegamento alla JNDI" a pagina 59
- "Esempio: Datalink" a pagina 152
- "Esempio: tipi distinti" a pagina 153
- "Esempio: inserimento di istruzioni SQL nell'applicazione Java" a pagina 187
- "Esempio: fine di una transazione" a pagina 93
- "Esempio: ID utente e parola d'ordine non validi" a pagina 44
- "Esempio: JDBC" a pagina 37
- "Esempio: più collegamenti che operano su una transazione" a pagina 87
- "Esempio: acquisizione di un contesto iniziale prima di collegare UDBDataSource" a pagina 60
- "Esempio: ParameterMetaData" a pagina 102
- "Esempio: modifica di valori con un'istruzione tramite il cursore di un'altra istruzione" a pagina 128
- "Esempio: interfaccia ResultSet per IBM Developer Kit per Java" a pagina 131
- "Esempio: sensibilità del ResultSet" a pagina 119
- "Esempio: ResultSet sensibili e non sensibili" a pagina 117
- "Esempio: impostazione del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource" a pagina 133
- "Esempio: SQLException" a pagina 73
- "Esempio: sospensione e ripristino di una transazione" a pagina 95
- "Esempio: ResultSet sospesi" a pagina 91
- "Esempio: esecuzione della verifica sulle prestazioni del lotto di collegamenti" a pagina 134
- "Esempio: verifica delle prestazioni di due DataSource" a pagina 136
- "Esempio: aggiornamento di BLOB" a pagina 146
- "Esempio: aggiornamento dei CLOB" a pagina 150
- "Esempio: utilizzo di un collegamento con più transazioni" a pagina 89

- “Esempio: utilizzo di BLOB” a pagina 147
- “Esempio: utilizzo di CLOB” a pagina 151
- “Creazione e popolamento di un DB2CachedRowSet” a pagina 156
- “Creazione e popolamento di un DB2CachedRowSet” a pagina 156
- “Esempio: utilizzo di JTA per gestire una transazione” a pagina 85
- “Esempio: utilizzo dei ResultSet di metadati che hanno più di una colonna” a pagina 70
- “Esempio: utilizzo simultaneo di JDBC e JDBC di IBM Toolbox per Java” a pagina 499
- “Esempio: utilizzo di PreparedStatement per ottenere un ResultSet” a pagina 104
- “Creazione e popolamento di un DB2CachedRowSet” a pagina 156
- “Creazione e popolamento di un DB2CachedRowSet” a pagina 156
- “Creazione e popolamento di un DB2CachedRowSet” a pagina 156
- “Creazione e popolamento di un DB2CachedRowSet” a pagina 156
- “Esempio: utilizzo del metodo executeUpdate dell’oggetto Statement” a pagina 99

Java Authentication and Authorization Service
- “Esempi: HelloWorld JAAS” a pagina 504
- “Esempio: JAAS SampleThreadSubjectLogin” a pagina 514

Java Generic Security Service
- “Esempio: programma client IBM JGSS non JAAS” a pagina 524
- “Esempio: programma server IBM JGSS non JAAS” a pagina 532
- “Esempio: programma client IBM JGSS abilitato a JAAS” a pagina 543
- “Esempio: programma server IBM JGSS abilitato” a pagina 545

Java Secure Sockets Extension
- “Esempi: IBM Java Secure Sockets Extension 1.4” a pagina 296

Java con altri linguaggi di programmazione
- “Esempio: richiamo di un programma CL con java.lang.Runtime.exec()” a pagina 231
- “Esempio: richiamo di un comando CL con java.lang.Runtime.exec()” a pagina 232
- “Esempio: richiamo di un altro programma Java con java.lang.Runtime.exec()” a pagina 231
- “Esempio: richiamo di Java da C” a pagina 238
- “Esempio: richiamo di Java da RPG” a pagina 238
- “Esempio: utilizzo dei flussi di immissione ed emissione per la comunicazione tra processi” a pagina 237
- “Esempio: API di richiamo Java” a pagina 215
- “Esempio: metodo nativo IBM i5/OS PASE per Java” a pagina 226
- Socket
- “Esempi: utilizzo della JNI (Java Native Interface) per i metodi nativi” a pagina 558

SQLJ
- “Esempio: inserimento di istruzioni SQL nell’applicazione Java” a pagina 187

SSL (Secure socket layer)
- “Esempi: modifica del codice Java per l’utilizzo delle produzioni socket del client” a pagina 277
- “Esempi: modifica del codice Java per l’utilizzo delle produzioni socket del server” a pagina 275
- “Esempi: modifica del client Java per l’utilizzo di SSL (secure sockets layer)” a pagina 280

- "Esempi: modifica del server Java per l'utilizzo di SSL (secure sockets layer)" a pagina 279

IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

FATTE SALVE LE GARANZIE INDEROGABILI DI LEGGE, IBM, I SUOI SVILUPPATORI DI PROGRAMMI E FORNITORI NON FORNISCONO GARANZIE O DICHIARAZIONI DI ALCUN TIPO, ESPRESSE O IMPLICITE, INCLUSE, A TITOLO ESEMPLIFICATIVO, GARANZIE O CONDIZIONI IMPLICITE DI COMMERCIABILITÀ O IDONEITÀ PER UNO SCOPO PARTICOLARE, INCLUSE LE GARANZIE DI FUNZIONAMENTO ININTERROTTO, RELATIVE AL CODICE O AL SUPPORTO TECNICO, SE ESISTENTE.

IN NESSUN CASO IBM, I SUOI SVILUPPATORI DI PROGRAMMI O FORNITORI SONO RESPONSABILI PER QUANTO SEGUE ANCHE SE INFORMATI DELLA POSSIBILITÀ DEL VERIFICARSI DI TALI DANNI:

1. PERDITA O DANNEGGIAMENTO DI DATI;
2. DANNI DIRETTI, SPECIALI, INCIDENTALI O INDIRETTI O QUALSIASI DANNO ECONOMICO CONSEGUENTE; O
3. PERDITE DI PROFITTI, AFFARI, ENTRATE O SPESE ANTICIPATE.

ALCUNE GIURISDIZIONI NON PERMETTONO L'ESCLUSIONE O LA LIMITAZIONE DI DANNI INCIDENTALI O CONSEQUENZIALI, PER TALE MOTIVO ALCUNE O TUTTE LE PRECEDENTI LIMITAZIONI O ESCLUSIONI POTREBBERO NON ESSERE VALIDE PER TUTTI GLI UTENTI.

Esempio: internazionalizzazione delle date utilizzando la classe java.util.DateFormat

Il seguente esempio mostra come è possibile utilizzare le locali per formattare le date.

Esempio 1: dimostra l'utilizzo della classe java.util.DateFormat per l'internazionalizzazione delle date

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//************************
// File: DateExample.java
//************************

import java.text.*;
import java.util.*;
import java.util.Date;

public class DateExample {

   public static void main(String args[]) {

     // Richiamare la data
     Date now = new Date();

     // Richiamare i progr. di formattazione data per valore predefinito, locali tedesca e francese
     DateFormat theDate = DateFormat.getDateInstance(DateFormat.LONG);
     DateFormat germanDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.GERMANY);
     DateFormat frenchDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);

     // Formattare e stampare le date
     System.out.println("Date in the default locale: " + theDate.aformat(now));
     System.out.println("Date in the German locale : " + germanDate.format(now));
     System.out.println("Date in the French locale : " + frenchDate.format(now));
   }
}
```

"Esempi: creazione di un programma Java internazionalizzato" a pagina 32
Se è necessario personalizzare un programma Java per una specifica regione del mondo, è possibile creare un programma Java internazionalizzato con le locali Java.

Esempio: internazionalizzazione del pannello numerico utilizzando la classe java.util.NumberFormat

Il seguente esempio mostra come è possibile utilizzare le locali per formattare i numeri.

Esempio 1: dimostra l'utilizzo della classe java.util.NumberFormat per l'internazionalizzazione delle emissioni numeriche

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//*************************
// File: NumberExample.java
//*************************

 import java.lang.*;
 import java.text.*;
 import java.util.*;

 public class NumberExample {

     public static void main(String args[]) throws NumberFormatException {

       // Il numero da formattare
       double number = 12345.678;

       // Richiamare i progr. di formattazione data per valore predefinito, locali spagnola e giapponese
       NumberFormat defaultFormat = NumberFormat.getInstance();
       NumberFormat spanishFormat = NumberFormat.getInstance(new
 Locale("es", "ES"));
       NumberFormat japaneseFormat = NumberFormat.getInstance(Locale.JAPAN);

       // Stampare il numero nei formati predefinito, spagnolo e giapponese
       // (Nota: NumberFormat non è necessariamente per il formato predefinito)
       System.out.println("The number formatted for the default locale; " +
                          defaultFormat.format(number));
       System.out.println("The number formatted for the Spanish locale; " +
                          spanishFormat.format(number));
       System.out.println("The number formatted for the Japanese locale; " +
                          japaneseFormat.format(number));
     }
 }
```

"Esempi: creazione di un programma Java internazionalizzato" a pagina 32
Se è necessario personalizzare un programma Java per una specifica regione del mondo, è possibile creare un programma Java internazionalizzato con le locali Java.

Esempio: internazionalizzazione di dati specifici della locale utilizzando la classe java.util.ResourceBundle

Questo esempio mostra come è possibile utilizzare le locali con i bundle della risorsa per internazionalizzare le stringhe del programma.

Questi file di proprietà sono richiesti affinché il programma ResourceBundleExample funzioni come previsto:

Contenuto di RBExample.properties

```
Hello.text=Hello
```

Contenuto di RBExample_de.properties

```
Hello.text=Guten Tag
```

Contenuto di RBExample_fr_FR.properties

```
Hello.text=Bonjour
```

Esempio 1: dimostra l'utilizzo della classe java.util.ResourceBundle per l'internazionalizzazione dei dati specifici della locale

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//*********************************
// File: ResourceBundleExample.java
//*********************************

import java.util.*;

public class ResourceBundleExample {
   public static void main(String args[]) throws MissingResourceException {

     String resourceName = "RBExample";
     ResourceBundle rb;

     // Locale predefinita
     rb = ResourceBundle.getBundle(resourceName);
     System.out.println("Default : " + rb.getString("Hello" + ".text"));

     // Richiedere un bundle di risorse con la locale specificata esplicitamente
     rb = ResourceBundle.getBundle(resourceName, Locale.GERMANY);
     System.out.println("German : " + rb.getString("Hello" + ".text"));

     // Nessun file proprietà per la Cina in questo esempio... usare valore predefinito
     rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);
     System.out.println("Chinese : " + rb.getString("Hello" + ".text"));

     // Di seguito viene riportato un effettuare ciò...
     Locale.setDefault(Locale.FRANCE);
     rb = ResourceBundle.getBundle(resourceName);
     System.out.println("French : " + rb.getString("Hello" + ".text"));

     // Nessun file proprietà file per la Cina in questo esempio...
          // usare valore predefinito, il quale è ora fr_FR.
     rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);
     System.out.println("Chinese : " + rb.getString("Hello" + ".text"));
   }
}
```

"Esempi: creazione di un programma Java internazionalizzato" a pagina 32
Se è necessario personalizzare un programma Java per una specifica regione del mondo, è possibile creare un programma Java internazionalizzato con le locali Java.

Esempio: proprietà Access

Questo è un esempio di come utilizzare la proprietà Access di Java.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Questo programma presume che l'indirizzario cujosql esista.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class AccessPropertyTest {
    public String url = "jdbc:db2:*local";
```

```
    public Connection connection = null;

    public static void main(java.lang.String[] args)
    throws Exception
    {
        AccessPropertyTest test = new AccessPropertyTest();

        test.setup();

        test.run();
        test.cleanup();
    }


/**
Impostare il DataSource utilizzato nella verifica.
**/
    public void setup()
    throws Exception
    {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        connection = DriverManager.getConnection(url);
        Statement s = connection.createStatement();
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.TEMP");
        } catch (SQLException e) { // Ignorarlo - non esiste
        }

        try {
            String sql = "CREATE PROCEDURE CUJOSQL.TEMP "
                  + " LANGUAGE SQL SPECIFIC CUJOSQL.TEMP "
                  + " MYPROC: BEGIN"
                  + "   RETURN 11;"
                  + " END MYPROC";
            s.executeUpdate(sql);
        } catch (SQLException e) {
            // Ignorarlo - già presente.
        }
        s.executeUpdate("create table cujosql.temp (col1 char(10))");
        s.executeUpdate("insert into cujosql.temp values ('compare')");
        s.close();
    }


    public void resetConnection(String property)
    throws SQLException
    {
        if (connection != null)
            connection.close();

        connection = DriverManager.getConnection(url + ";access=" + property);
    }


    public boolean canQuery() {
        Statement s = null;
        try {
            s = connection.createStatement();
            ResultSet rs = s.executeQuery("SELECT * FROM cujosql.temp");
            if (rs == null)
                return false;

            rs.next();

            if (rs.getString(1).equals("compare   "))
                return true;
```

```
        return false;

    } catch (SQLException e) {
        // System.out.println("Exception: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignorarlo.
            }
        }
    }
}


public boolean canUpdate() {
    Statement s = null;
    try {
        s = connection.createStatement();
        int count = s.executeUpdate("INSERT INTO CUJOSQL.TEMP VALUES('x')");
        if (count != 1)
            return false;

        return true;

    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignorarlo.
            }
        }
    }
}


public boolean canCall() {
    CallableStatement s = null;
    try {
        s = connection.prepareCall("? = CALL CUJOSQL.TEMP()");
        s.registerOutParameter(1, Types.INTEGER);
        s.execute();
        if (s.getInt(1) != 11)
            return false;

        return true;

    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignorarlo.
            }
```

```
            }
        }
    }


    public void run()
    throws SQLException
    {
        System.out.println("Set the connection access property to read only");
        resetConnection("read only");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

        System.out.println("Set the connection access property to read call");
        resetConnection("read call");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

        System.out.println("Set the connection access property to all");
        resetConnection("all");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

    }


    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            // Ignorarlo.
        }
    }
}
```

Esempio: BLOB

Questo è un esempio del modo in cui è possibile inserire un BLOB nel database o richiamarlo dal database.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// PutGetBlobs è un'applicazione di esempio
// che mostra come gestire l'API JDBC
// per ottenere e inserire i BLOB in e dalle
// colonne database.
//
// I risultati dell'esecuzione di questo programma
// consistono in due valori BLOB in una
// nuova tabella. Sono identici e
// contengono 500k di dati byte
// casuali.
////////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
   public static void main(String[] args)
   throws SQLException
```

```
  {
      // Registrare il programma di controllo JDBC nativo.
      try {
         Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
     } catch (Exception e) {
         System.exit(1);  // Errore di impostazione.
     }

     // Stabilire una Connection e una Statement con cui operare.
     Connection c = DriverManager.getConnection("jdbc:db2:*local");
     Statement s = c.createStatement();

     // Ripulire l'esecuzione precedente di questa applicazione.
     try {
         s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
     } catch (SQLException e) {
         // Ignorarlo - presupporre che la tabella non esista.
     }

     // Creare una tabella con una colonna BLOB. La dimensione predefinita della
     // colonna BLOB è 1 MB.
     s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

     // Creare un oggetto PreparedStatement che consenta di inserire un nuovo
     // oggetto Blob nel database.
     PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

     // Creare un valore BLOB grande...
     Random random = new Random ();
     byte [] inByteArray = new byte[500000];
     random.nextBytes (inByteArray);

     // Impostare il parametro PreparedStatement. Nota: non è trasferibile
     // in tutti i programmi di controllo JDBC. Questi ultimi non dispongono di
     // supporto quando si usa setBytes per le colonne BLOB. Viene utilizzato
     // per consentire la creazione di nuovi BLOB. Consente inoltre ai programmi di
     // controllo JDBC 1.0 di gestire le colonne contenenti i dati BLOB.
     ps.setBytes(1, inByteArray);

     // Elaborare l'istruzione, inserendo il BLOB nel database.
     ps.executeUpdate();

     // Elaborare una query e ottenere il BLOB inserito fuori dal
     // database come un oggetto Blob.
     ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
     rs.next();
     Blob blob = rs.getBlob(1);


     // Inserire di nuovo Blob nel database tramite
     // PreparedStatement.
     ps.setBlob(1, blob);
     ps.execute();

     c.close(); // La chiusura del collegamento chiude anche stmt e rs.
  }
}
```

Esempio: interfaccia CallableStatement per IBM Developer Kit per Java

Questo è un esempio della modalità di utilizzo dell'interfaccia CallableStatement.

Esempio: interfaccia CallableStatement

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Collegarsi al server.
Connection c = DriverManager.getConnection("jdbc:db2://mySystem");

// Creare l'oggetto CallableStatement.
// Esso precompila la chiamata specificata in una procedura memorizzata.
// I punti interrogativi indicano dove devono essere impostati i parametri di immissione
// e dove possono essere richiamati i parametri di emissione.
// I primi due parametri sono di immissione e il terzo è di emissione.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

// Impostare i parametri di immissione.
cs.setInt (1, 123);
cs.setInt (2, 234);

// Registrare il tipo di parametro di emissione.
cs.registerOutParameter (3, Types.INTEGER);

// Eseguire la procedura memorizzata.
cs.execute ();

// Richiamare il valore del parametro di emissione.
int sum = cs.getInt (3);

// Chiudere CallableStatement e Connection.
cs.close();
c.close();
```

"CallableStatement" a pagina 106
L'interfaccia CallableStatement JDBC estende PreparedStatement e fornisce il supporto per i parametri di immissione/emissione e di emissione. L'interfaccia CallableStatement dispone inoltre del supporto per i parametri di immissione fornito dall'interfaccia PreparedStatement.

Esempio: eliminazione di valori da una tabella tramite il cursore di un'altra istruzione

Quest'esempio Java mostra come eliminare i valori da una tabella tramite il cursore di un'altra istruzione.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class UsingPositionedDelete {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {
        UsingPositionedDelete test = new UsingPositionedDelete();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }



/**
Gestire quanto necessario per il lavoro di impostazione necessario.
**/
    public void setup() {
        try {
            // Registrare il programma di controllo JDBC.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

```
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
            } catch (SQLException e) {
                // Ignorare i problemi.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
In questa sezione, deve essere aggiunto il codice necessario
per eseguire la verifica. se è necessaria solo un collegamento al database,
può essere utilizzata la variabile globale 'connection'.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Aggiornare ogni valore utilizzando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement
                                      ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
                                      rs.getCursorName ());


            // Eseguire il loop del ResultSet e aggiornare ogni voce.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Ripulire le risorse dopo averle utilizzate.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }



/**
```

```
In questa sezione, inserire tutti i lavori di ripulitura per la verifica.
**/
    public void cleanup() {
        try {
            // Chiudere il collegamento globale aperto in setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }



/**
Visualizzare il contenuto della tabella.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Esempio: CLOB

Questo è un esempio del modo in cui è possibile inserire un CLOB in un database o richiamarlo da un database.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// PutGetClobs è un esempio di applicazione
// che mostra come gestire l'API JDBC
// per ottenere e inserire i CLOB in e dalle
// colonne database.
//
// I risultati dell'esecuzione di questo programma
// consistono in due valori CLOB in una
// nuova tabella. Sono identici e
// contengono 500k di dati di testo
// ripetitivo.
////////////////////////////////////////
import java.sql.*;

public class PutGetClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

```
        } catch (Exception e) {
            System.exit(1);  // Errore di impostazione.
        }

        // Stabilire una Connection e una Statement con cui operare.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Ripulire l'esecuzione precedente di questa applicazione.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
        } catch (SQLException e) {
            // Ignorarlo - presupporre che la tabella non esista.
        }

        // Creare una tabella con una colonna CLOB. La dimensione predefinita della
        // colonna BLOB è 1 MB.
        s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

        // Creare un oggetto PreparedStatement che consente di inserire un nuovo
        // oggetto Clob nel database.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

        // Creare un valore CLOB grande...
        StringBuffer buffer = new StringBuffer(500000);
        while (buffer.length() < 500000) {
            buffer.append("All work and no play makes Cujo a dull boy.");
        }
        String clobValue = buffer.toString();

        // Impostare il parametro PreparedStatement. Non è trasferibile
        // in tutti i programmi di controllo JDBC. Questi ultimi non dispongono di
        // supporto setBytes per le colonne CLOB. Ciò viene effettuato per
        // consentire all'utente di creare nuovi CLOB. Consente inoltre alle unità di
        // controllo JDBC 1.0 un modo per gestire le colonne contenenti
        // dati Clob.
        ps.setString(1, clobValue);

        // Elaborare l'istruzione, inserendo il clob nel database.
        ps.executeUpdate();

        // Elaborare una query e ottenere il CLOB inserito nel
        // database come oggetto Clob.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
        rs.next();
        Clob clob = rs.getClob(1);


        // Inserire di nuovo Clob nel database tramite
        // PreparedStatement.
        ps.setClob(1, clob);
        ps.execute();

        c.close(); // La chiusura del collegamento chiude anche stmt e rs.
    }
}
```

Esempio: creazione di UDBDataSource e suo collegamento alla JNDI

Questo è un esempio di come creare un UDBDataSource e collegarlo alla JNDI.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Importare i pacchetti richiesti. Al momento dello sviluppo,
// la classe specifica del programma di controllo JDBC che implementa
// DataSource deve essere importata.
```

```
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Creare un nuovo oggetto UDBDataSource e fornirgli
        // una descrizione.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource");

        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Collegare l'oggetto UDBDataSource appena creato al
        // servizio indirizzario JNDI, fornendogli un nome
        // utilizzabile per ricercare di nuovo questo oggetto
        // successivamente.
        ctx.rebind("SimpleDS", ds);
    }
}
```

Esempio: creazione di un UDBDataSource e acquisizione di un ID utente e una parola d'ordine

Questo è un esempio di come creare UDBDataSource e utilizzare il metodo getConnection per ottenere un ID utente e una parola d'ordine al tempo di esecuzione.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/// Importare i pacchetti richiesti. Non esiste codice
// specifico del programma di controllo necessario nelle applicazioni
// del tempo di esecuzione.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Richiamare l'oggetto UDBDataSource collegato usando il nome
        // con cui è stato precedentemente collegato. Nel tempo di esecuzione
        // viene usata solo l'interfaccia DataSource, quindi non c'è bisogno
        // di convertire l'oggetto nella classe di implementazione UDBDataSource.
        // (Non è necessario conoscere qual è la classe di
        // implementazione. Solo il nome JNDI logico è
        // necessario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una volta ottenuto il DataSource, può essere usato per stabilire
        // un collegamento. Il profilo utente cujo e la parola d'ordine newtiger
        // vengono utilizzati per creare il collegamento al posto dell'ID utente e
```

```
        // della parola d'ordine predefiniti per il DataSource.
        Connection connection = ds.getConnection("cujo", "newtiger");

        // Il collegamento può essere usato per creare oggetti Statement e
        // aggiornare il database o elaborare le query come segue.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // Il collegamento viene chiuso prima del termine dell'applicazione.
        connection.close();
    }
}
```

Esempio: creazione di UDBDataSourceBind ed impostazione delle proprietà DataSource

Questo è un esempio di come creare UDBDataSource e impostare l'ID utente e la parola d'ordine come proprietà DataSource.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Importare i pacchetti richiesti. Al momento dello sviluppo,
// la classe specifica del programma di controllo JDBC che implementa
// DataSource deve essere importata.
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Creare un nuovo oggetto UDBDataSource e fornirgli
        // una descrizione.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource "  +
                          "with cujo as the default " +
                          "profile to connect with.");

        // Fornire un ID utente e una parola d'ordine da utilizzare
        // per le richieste di collegamento.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Collegare l'oggetto UDBDataSource appena creato al
        // servizio indirizzario JNDI, fornendogli un nome
        // utilizzabile per ricercare di nuovo questo oggetto
        // successivamente.
        ctx.rebind("SimpleDS2", ds);
    }
}
```

Esempio: restituzione di un elenco di tabelle utilizzando l'interfaccia DatabaseMetaData IBM Developer Kit per Java

Questo esempio mostra come restituire una lista di tabelle.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Collegarsi al server.
Connection c = DriverManager.getConnection("jdbc:db2:mySystem");

// Richiamare i meta dati database dal collegamento.
DatabaseMetaData dbMeta = c.getMetaData();

// Richiamare un elenco di tabelle che corrispondono a questi criteri.
String catalog = "myCatalog";
String schema = "mySchema";
String table  = "myTable%"; // % indica il modello di ricerca
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"}:
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterare attraverso il ResultSet per richiamare i valori.

// Chiudere il collegamento.
c.close():
```

Esempio: Datalink

Quest'applicazione di esempio mostra come utilizzare la API JDBC per gestire le colonne database del datalink.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// PutGetDatalinks è un'applicazione di esempio
// che mostra come utilizzare l'API JDBC
// per gestire le colonne database del datalink.
////////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrare il programma di controllo JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Errore di impostazione.
        }

        // Stabilire una Connection e una Statement con cui operare.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Ripulire l'esecuzione precedente di questa applicazione.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
        } catch (SQLException e) {
            // Ignorarlo - presupporre che la tabella non esista.
        }

        // Creare una tabella con una colonna datalink.
```

```
        s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

        // Creare un oggetto PreparedStatement che consente di aggiungere un nuovo
        // datalink al database. Poiché la conversione in un datalink non può
        // essere effettuata direttamente nel database, è possibile codificare
        // l'istruzione SQL per eseguire la conversione esplicita.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
                                                  VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

        // Impostare il datalink. Questo URL punta ad un argomento relativo alle
        // nuove funzioni di JDBC 3.0.
        ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

        // Elaborare l'istruzione, inserendo il CLOB nel database.
        ps.executeUpdate();

        // Elaborare una query e ottenere il CLOB appena inserito nel
        // database come oggetto Clob.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        String datalink = rs.getString(1);


        // Inserire tale valore di datalink nel database tramite
        // PreparedStatement. Nota: questa funzione richiede JDBC 3.0
        // supporto di JDBC 2.0.
        /*
        try {
            URL url = new URL(datalink);
            ps.setURL(1, url);
            ps.execute();
        } catch (MalformedURLException mue) {
            // Gestire in questo punto l'immissione.
        }

        rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        URL url = rs.getURL(1);
        System.out.println("URL value is " + url);
        */

        c.close(); // La chiusura del collegamento chiude anche stmt e rs.
    }
}
```

Esempio: tipi distinti

Questo esempio illustra come utilizzare tipi distinti.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// Questo esempio di programma mostra esempi
// di diverse attività comuni che possono
// essere eseguite con tipi distinti.
////////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrare il programma di controllo JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
```

```
            System.exit(1);  // Errore di impostazione.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Ripulire le vecchie esecuzioni.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignorarlo e presupporre che la tabella non esista.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
        } catch (SQLException e) {
            // Ignorarlo e presupporre che la tabella non esista.
        }

        // Creare il tipo, creare la tabella e inserire un valore.
        s.executeUpdate("CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)");
        s.executeUpdate("CREATE TABLE CUJOSQL.SERIALNOS (COL1 CUJOSQL.SSN)");

        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.SERIALNOS VALUES(?)");
        ps.setString(1, "399924563");
        ps.executeUpdate();
        ps.close();

        // È possibile ottenere dettagli sui tipi disponibili con i nuovi metadati in
        // JDBC 2.0
        DatabaseMetaData dmd = c.getMetaData();

        int types[] = new int[1];
        types[0] = java.sql.Types.DISTINCT;

        ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN", types);
        rs.next();
        System.out.println("Type name " + rs.getString(3) +
                           " has type " + rs.getString(4));


        // Accedere ai dati inseriti.
        rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
        rs.next();
        System.out.println("The SSN is " + rs.getString(1));

        c.close(); // La chiusura del collegamento chiude anche stmt e rs.
    }
}
```

Esempio: inserimento di istruzioni SQL nell'applicazione Java

La seguente applicazione SQLJ di esempio, App.sqlj, utilizza SQL statico per richiamare e aggiornare i dati dalla tabella EMPLOYEE del database di esempio DB2.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ; // [1]
#sql iterator App_Cursor2 (String) ;

class App
{
```

```
 /**********************
  ** Registrare unità **
  **********************/

static
{
  try
  {
    Class.forName("com.ibm.db2.jdbc.app.DB2Driver").newInstance();
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
}

 /********************
  **      Main      **
  ********************/

public static void main(String argv[])
{
  try
  {
    App_Cursor1 cursor1;
    App_Cursor2 cursor2;

    String str1 = null;
    String str2 = null;
    long   count1;

    // L'URL è jdbc:db2:dbname
    String url = "jdbc:db2:sample";

    DefaultContext ctx = DefaultContext.getDefaultContext();
    if (ctx == null)
    {
      try
      {
        // collegarsi con id/parole d'ordine predefiniti
        Connection con = DriverManager.getConnection(url);
        con.setAutoCommit(false);
        ctx = new DefaultContext(con);
      }
      catch (SQLException e)
      {
        System.out.println("Error: could not get a default context");
        System.err.println(e) ;
        System.exit(1);
      }
      DefaultContext.setDefaultContext(ctx);
    }

    // richiamare i dati dal database
    System.out.println("Retrieve some data from the database.");
    #sql cursor1 = {SELECT empno, firstnme FROM employee}; // [2]

    // visualizzare la serie di risultati
    // cursor1.next() restituisce false quando non ci sono più righe
    System.out.println("Received results:");
    while (cursor1.next()) // [3]
    {
      str1 = cursor1.empno(); // [4]
      str2 = cursor1.firstnme();

      System.out.print (" empno= " + str1);
```

```
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor1.close(); // 9

    // richiamare il numero di impiegati dal database
    #sql { SELECT count(*) into :count1 FROM employee }; // 5
    if (1 == count1)
      System.out.println ("There is 1 row in employee table");
    else
      System.out.println ("There are " + count1
                          + " rows in employee table");

    // aggiornare il database
    System.out.println("Update the database.");
    #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

    // richiamare i dati aggiornati dal database
    System.out.println("Retrieve the updated data from the database.");
    str1 = "000010";
    #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

    // visualizzare la serie di risultati
    // cursor2.next() restituisce false quando non ci sono più righe
    System.out.println("Received results:");
    while (true)
    {
      #sql { FETCH :cursor2 INTO :str2 }; // 7
      if (cursor2.endFetch()) break; // 8

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor2.close(); // 9

    // rollback dell'aggiornamento
    System.out.println("Rollback the update.");
    #sql { ROLLBACK work };
    System.out.println("Rollback done.");
  }
  catch( Exception e )
  {
    e.printStackTrace();
  }
 }
}
```

[1]Dichiarazione di iteratori. Questa sezione dichiara due tipi di iteratori:

- App_Cursor1: dichiara i tipi e i nomi dei dati di colonna e restituisce i valori delle colonne a seconda del nome di colonna (collegamento denominato a colonne).
- App_Cursor2: dichiara i tipi dei dati di colonna e restituisce i valori delle colonne tramite la posizione della colonna (collegamento di posizione alle colonne).

[2]Inizializzazione dell'iteratore. L'oggetto iteratore cursor1 viene inizializzato utilizzando il risultato di una query. La query memorizza il risultato in cursor1.

[3]Avanzamento dell'iteratore alla riga successiva. Il metodo cursor1.next() restituisce il valore Booleano `false` se non esistono più righe da richiamare.

[4]Spostamento di dati. Il metodo del programma di accesso denominato empno() restituisce il valore della colonna denominata empno sulla riga corrente. Il metodo di accesso denominato firstnme() restituisce il valore della colonna denominata firstnme sulla riga corrente.

[5]Dati SELECT in una variabile host. L'istruzione SELECT trasferisce il numero di righe presente in una tabella in una variabile host count1.

[6] Inizializzazione dell'iteratore. L'oggetto iteratore cursor2 viene inizializzato utilizzando il risultato di una query. La query memorizza il risultato in cursor2.

[7]Richiamo di dati. L'istruzione FETCH restituisce il valore corrente della prima colonna dichiarata nel cursore ByPos dalla tabella dei risultati nella variabile host str2.

[8]Controllo della riuscita di un'istruzione FETCH.INTO. Il metodo endFetch() restituisce un valore Booleano true se l'iteratore non è posizionato su una riga, cioè se l'ultimo tentativo di selezionare una riga ha avuto esito negativo. Il metodo endFetch() restituisce false se l'ultimo tentativo di selezionare una riga ha avuto esito positivo. DB2 tenta di selezionare una riga quando viene richiamato il metodo next(). Un'istruzione FETCH...INTO chiama implicitamente il metodo next().

[9]Chiusura degli iteratori. Il metodo close() rilascia qualsiasi risorsa mantenuta dagli iteratori. È necessario chiudere in modo esplicito gli iteratori per assicurarsi che le risorse di sistema vengano rilasciate in modo tempestivo.

Esempio: fine di una transazione

Questo è un esempio di chiusura di una transazione nell'applicazione.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEnd {

    public static void main(java.lang.String[] args) {
        JTATxEnd test = new JTATxEnd();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
```

```
        s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.)");

        s.close();
    } finally {
        if (c != null) {
            c.close();
        }
    }
}


/**
 * Questa verifica utilizza il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Per transazioni XA, è necessario l'identificativo transazione.
        // Un'implementazione dell'interfaccia XID non è inclusa all'unità
        // di controllo JDBC. Vedere Transazioni con JTA per una
        // descrizione di questa interfaccia per creare la classe relativa ad essa.
        Xid xid = new XidImpl();

        // Il collegamento da XAResource può essere utilizzato come qualsiasi
        // collegamento JDBC.
        Statement     stmt   = c.createStatement();

        // La risorsa XA deve essere notificata prima di avviare qualsiasi
        // elaborazione di transazioni.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Creare un ResultSet durante l'elaborazione JDBC e selezionare una riga.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Quando viene chiamato il metodo end, tutti i cursori ResultSet vengono chiusi.
        // L'accesso al ResultSet dopo questo punto fa in modo che
        // venga emessa un'eccezione.
        xaRes.end(xid, XAResource.TMNOFLAGS);

        try {
            String value = rs.getString(1);
            System.out.println("Something failed if you receive this message.");
        } catch (SQLException e) {
            System.out.println("The expected exception was thrown.");
        }

        // Eseguire il commit della transazione per assicurarsi che tutti i vincoli vengano
        // rilasciati.
        int rc = xaRes.prepare(xid);
        xaRes.commit(xid, false);


    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
```

```
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

"Transazioni distribuite JDBC" a pagina 82
In genere le transazioni in JDBC (Java Database Connectivity) sono locali. Ciò significa che un singolo collegamento esegue tutto il lavoro della transazione e che il collegamento può lavorare solo su una transazione alla volta.

Esempio: ID utente e parola d'ordine non validi

Questo esempio illustra come utilizzare la proprietà Connection in modalità di denominazione SQL.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////////////////////////////////////////////
//
// Esempio InvalidConnect.
//
// Questo programma utilizza la proprietà Connection nella modalità di denominazione SQL.
//
//////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di programma di controllo JDBC IBM Developer per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////
import java.sql.*;
import java.util.*;

public class InvalidConnect {

    public static void main(java.lang.String[] args)
    {
        // Registrare il programma di controllo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
```

```
            System.out.println("ERROR: JDBC driver did not load.");
            System.exit(0);
        }

        // Tentare di ottenere un collegamento senza specificare un utente o una
        // parola d'ordine. Il tentativo ha esito positivo e il collegamento usa
        // lo stesso profilo utente tramite il quale è in esecuzione il lavoro.
        try {
            Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
            c1.close();
        } catch (SQLException e) {
            System.out.println("This test should not get into this exception path.");
            e.printStackTrace();
            System.exit(1);
        }

        try {
            Connection c2 = DriverManager.getConnection("jdbc:db2:*local",
                                                        "notvalid", "notvalid");
        } catch (SQLException e) {
            System.out.println("This is an expected error.");
            System.out.println("Message is " + e.getMessage());
            System.out.println("SQLSTATE is " + e.getSQLState());
        }

    }
}
```

Esempio: JDBC

Questo è un esempio della modalità di utilizzo del programma BasicJDBC. Questo programma utilizza il programma di controllo JDBC nativo per IBM Developer Kit per Java per creare una semplice tabella ed elaborare una query che visualizza i dati in tale tabella.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////////
//
// Esempio BasicJDBC. Questo programma utilizza il programma di controllo JDBC nativo
// affinché Developer Kit per Java crei una tabella semplice ed elabori una query
// che visualizzi i dati in tale tabella.
//
// Sintassi del comando:
//    BasicJDBC
//
////////////////////////////////////////////////////////////////////////////
//
// Questo codice sorgente è un esempio del programma di controllo JDBC IBM Developer
Kit per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
```

```
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////////////////////////////////////////////

// Includere le classi Java da utilizzare. In questa applicazione, molte classi
// del pacchetto java.sql vengono utilizzate e anche la classe
// java.util.Properties viene utilizzata come parte per stabilire
// un collegamento al database.
import java.sql.*;
import java.util.Properties;

// Creare una classe public che includa il programma.
public class BasicJDBC {

    // Il collegamento è una variabile privata dell'oggetto.
    private Connection connection = null;

    // Qualsiasi classe che deve essere un 'punto di immissione' per eseguire
    // un programma deve disporre di un metodo principale. Quest'ultimo costituisce
    // il punto di partenza dell'elaborazione nel momento in cui viene chiamato il programma.
    public static void main(java.lang.String[] args) {

        // Creare un oggetto di tipo BasicJDBC. Esso si rivela
        // fondamentale nella programmazione orientata agli oggetti. Una volta
        // creato un oggetto, chiamare diversi metodi su tale
        // oggetto per eseguire il lavoro.
        // In questo caso, chiamando il programma di creazione dell'oggetto
        // viene creata un collegamento al database che viene utilizzato
        // dagli altri metodi per eseguire un lavoro sul database.
        BasicJDBC test = new BasicJDBC();

        // Chiamare il metodo rebuildTable. Questo metodo assicura che la
        // tabella utilizzata nel programma esiste e viene visualizzata
        // correttamente. Il valore di ritorno è di tipo booleano per
        // indicare se la nuova creazione della tabella è stata completata
        // con esito positivo o meno. In caso non lo sia, viene visualizzato
        // un messaggio e si esce dal programma.
        if (!test.rebuildTable()) {
            System.out.println("Failure occurred while setting up " +
                               " for running the test.");
            System.out.println("Test will not continue.");
            System.exit(0);
        }

        // Il metodo run query viene chiamato successivamente. Questo metodo
        // elabora un'istruzione di selezione SQL sulla tabella che è stata
        // creata nel metodo rebuildTable. L'emissione di tale query
        // viene inviato all'emissione standard affinché possa essere esaminata.
        test.runQuery();

        // Infine, viene chiamato il metodo cleanup. Questo metodo assicura
        // che il collegamento al database cui è collegato l'oggetto
        // viene chiuso.
        test.cleanup();
    }


    /**
    Questo è il programma di creazione per la verifica JDBC. Esso crea un collegamento
    database che viene memorizzato in una variabile di istanza da utilizzare in
    successive chiamate al metodo.
    **/
    public BasicJDBC() {

        // Un modo per creare un collegamento database consiste nel passare un URL
```

```
    // e un oggetto di proprietà java in DriverManager. Il codice riportato
    // di seguito crea un oggetto di proprietà che dispone di un proprio ID utente
    // e parola d'ordine. Queste parti di informazioni vengono utilizzate per
    // collegarsi al database.
    Properties properties = new Properties ();
    properties.put("user", "cujo");
    properties.put("user", "newtiger");

    // Utilizzare un blocco try/catch per acquisire tutte le eccezioni che
    // derivano dal seguente codice.
    try {
        // DriverManager deve sapere che esiste un programma di controllo JDBC disponibile
        // per gestire una richiesta di collegamento utente. La seguente riga
        // fa in modo che il programma di controllo JDBC venga caricato e che DriveManager venga registrato.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        // Creare l'oggetto DatabaseConnection utilizzato da questo programma in
        // tutte le altre chiamate del metodo effettuate. Il seguente codice
        // specifica che deve essere stabilito un collegamento al database locale
        // e che tale collegamento deve essere conforme alle proprietà impostate
        // precedentemente (cioè, deve utilizzare l'ID utente e la parola d'ordine specificati).
        connection = DriverManager.getConnection("jdbc:db2:*local", properties);

    } catch (Exception e) {
        // Se una qualsiasi delle righe del blocco try/catch ha esito negativo,
        // controllare i trasferimenti alla seguente riga di codice. Un'applicazione
        // affidabile tenta di gestire il problema o di fornire ulteriori dettagli.
        // In questo programma viene visualizzato il messaggio di errore dell'eccezione
        // e l'applicazione consente al programma di eseguire una restituzione.
        System.out.println("Caught exception: " + e.getMessage());
    }
}


/**
Assicurarsi che la tabella qgpl.basicjdbc venga visualizzata correttamente all'inizio
della verifica.

@returns boolean    Restituisce true se la tabella è stata ricreata con esito positivo;
                    restituisce false se si è verificato un errore.
**/
public boolean rebuildTable() {
    // Raggruppare tutte le funzionalità in un blocco try/catch in modo che si
    // tenti di gestire gli errori quando si verificano all'interno di questo metodo.
    try {

        // Gli oggetti Statement vengono usati per elaborare istruzioni SQL sul
        // database. L'oggetto Connection è utilizzato per creare uno Statement
        // Statement.
        Statement s = connection.createStatement();

        try {
            // Creare la tabella di verifica da zero. Elaborare un'istruzione di
            // aggiornamento che tenti di cancellare la tabella se attualmente esiste.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // Non eseguire nulla se si verifica un'eccezione. Si presuppone che
            // il problema consiste nel fatto che la tabella interrotta non
            // esiste e che può essere creata successivamente.
        }

        // Utilizzare l'oggetto statement per creare la tabella.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Utilizzare l'oggetto statement per popolare la tabella con alcuni dati.
        s.executeUpdate("insert into qgpl.basicjdbc values(1, 'Frank Johnson')");
        s.executeUpdate("insert into qgpl.basicjdbc values(2, 'Neil Schwartz')");
```

```
        s.executeUpdate("insert into qgpl.basicjdbc values(3, 'Ben Rodman')");
        s.executeUpdate("insert into qgpl.basicjdbc values(4, 'Dan Gloore')");

        // Chiudere l'istruzione SQL per indicare al database che non è più
        // necessario.
        s.close();

        // Se l'intero metodo viene elaborato con esito positivo, viene restituito true.
        // A questo punto, la tabella è stata creata o aggiornata correttamente.
        return true;

    } catch (SQLException sqle) {
        // Se una delle istruzioni SQL ha esito negativo (diverso dall'interruzione
        // della tabella gestita nel blocco try/catch interno), viene visualizzato il
        // messaggio di errore e viene restituito false al chiamante,
        // indicante che potrebbe non essere completa.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}



/**
Esegue una query sulla tabella dimostrativa e i risultati vengono visualizzati
nell'emissione standard.
**/
public void runQuery() {
    // Raggruppare tutte le funzionalità in un blocco try/catch in modo che si
    // tenti di gestire gli errori quando si verificano all'interno di questo
    // metodo.
    try {
        // Creare un oggetto Statement.
        Statement s = connection.createStatement();

        // Utilizzare l'oggetto statement per eseguire una query SQL. Le
        // query restituiscono oggetti ResultSet usati per consultare i
        // dati forniti dalla query.
        ResultSet rs = s.executeQuery("select * from qgpl.basicjdbc");

        // Visualizzare la parte superiore della 'tabella' e iniziare il conteggio
        // del numero di righe restituite.
        System.out.println("--------------------");
        int i = 0;

        // Il successivo metodo ResultSet viene usato per elaborare le righe di un
        // ResultSet. Il successivo metodo deve essere chiamato una volta prima che i
        // primi dati siano disponibili per la visualizzazione. Purché venga restituito
        // true, esiste un'altra riga di dati che può essere utilizzata.
        while (rs.next()) {

            // Ottenere entrambe le colonne nella tabella per ogni riga e scrivere una
            // riga nella tabella sul pannello con i dati. Quindi, aumentare il
            // conteggio delle righe elaborate.
            System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
            i++;
        }

        // Inserire un bordo alla fine della tabella e visualizzare il numero
        // delle righe come emissione.
        System.out.println("--------------------");
        System.out.println("There were " + i + " rows returned.");
        System.out.println("Output is complete.");

    } catch (SQLException e) {
        // Visualizzare le informazioni aggiuntive sulle eccezioni SQL
        // che vengono generate come emissione.
```

```
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            e.printStackTrace();
        }
    }


    /**
    Il seguente metodo assicura che le risorse JDBC ancora assegnate
    sono libere.
    **/
    public void cleanup() {
        try {
            if (connection != null)
                connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Esempio: più collegamenti che operano su una transazione

Questo è un esempio del modo in cui utilizzare più collegamenti che operano su una singola transazione.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;
public class JTAMultiConn {
   public static void main(java.lang.String[] args) {
      JTAMultiConn test = new JTAMultiConn();
      test.setup();
      test.run();
   }
/**
* Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
*/
   public void setup() {
      Connection c = null;
      Statement s = null;
      try {
         Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
         c = DriverManager.getConnection("jdbc:db2:*local");
         s = c.createStatement();
         try {
            s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
         }
         catch (SQLException e) {
         // Ignorare... non esiste
         }
         s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
                         (50))");
               s.close();
      }
      finally {
         if (c != null) {
            c.close();
         }
```

```
      }
   }
/**
* Questa verifica usa il supporto JTA per gestire le transazioni.
*/
   public void run() {
      Connection c1 = null;
      Connection c2 = null;
      Connection c3 = null;
      try {
         Context ctx = new InitialContext();
         // Presupporre che il sorgente dati venga riportato da un UDBXADataSource.
         UDBXADataSource ds = (UDBXADataSource)
                              ctx.lookup("XADataSource");
         // Dal DataSource, ottenere un oggetto XAConnection che
         // contenga un XAResource e un oggetto Connection.
         XAConnection xaConn1 = ds.getXAConnection();
         XAConnection xaConn2 = ds.getXAConnection();
         XAConnection xaConn3 = ds.getXAConnection();
         XAResource xaRes1 = xaConn1.getXAResource();
         XAResource xaRes2 = xaConn2.getXAResource();
         XAResource xaRes3 = xaConn3.getXAResource();
         c1 = xaConn1.getConnection();
         c2 = xaConn2.getConnection();
         c3 = xaConn3.getConnection();
         Statement stmt1 = c1.createStatement();
         Statement stmt2 = c2.createStatement();
         Statement stmt3 = c3.createStatement();
         // Per transazioni XA, è necessario un identificativo transazione.
         // Il supporto per la creazione di XID viene lasciato di nuovo al
         // programma applicativo.
         Xid xid = JDXATest.xidFactory();
         // Eseguire l'elaborazione di alcune transazioni tramite cui sono stati
         // creati i tre collegamenti.
         xaRes1.start(xid, XAResource.TMNOFLAGS);
         int count1 = stmt1.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-A')");
         xaRes1.end(xid, XAResource.TMNOFLAGS);

         xaRes2.start(xid, XAResource.TMJOIN);
         int count2 = stmt2.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-B')");
         xaRes2.end(xid, XAResource.TMNOFLAGS);

         xaRes3.start(xid, XAResource.TMJOIN);
         int count3 = stmt3.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-C')");
         xaRes3.end(xid, XAResource.TMSUCCESS);
         // Una volta terminato, eseguire il commit della transazione come una singola unità
         // È richiesto un prepare() e commit() o un commit() a 1 fase per ogni
         // database separato (XAResource) che ha preso parte alla
         // transazione. Poiché tutte le risorse cui si ha avuto accesso (xaRes1, xaRes2
         // e xaRes3) si riferiscono allo stesso database, è necessario solo un prepare o commit.
         int rc = xaRes.prepare(xid);
         xaRes.commit(xid, false);
      }
      catch (Exception e) {
         System.out.println("Something has gone wrong.");
         e.printStackTrace();
      }
      finally {
         try {
            if (c1 != null) {
               c1.close();
            }
         }
         catch (SQLException e) {
            System.out.println("Note: Cleaup exception " +
                               e.getMessage());
         }
```

```
         try {
            if (c2 != null) {
               c2.close();
            }
         }
         catch (SQLException e) {
            System.out.println("Note: Cleaup exception " +
                               e.getMessage());
         }
         try {
            if (c3 != null) {
               c3.close();
            }
         }
         catch (SQLException e) {
            System.out.println("Note: Cleaup exception " +
                               e.getMessage());
         }
      }
   }
}
```

Esempio: acquisizione di un contesto iniziale prima di collegare UDBDataSource

Il seguente esempio ottiene un contesto iniziale prima di collegare UDBDataSource. Il metodo di ricerca viene quindi utilizzato su quel contesto per restituire un oggetto di tipo DataSource affinché l'applicazione lo utilizzi.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Importare i pacchetti richiesti. Non esistono codici
// specifici del programma di controllo necessari nelle applicazioni
// del tempo di esecuzione.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Richiamare un contesto JNDI. Il contesto serve come
        // root per l'ubicazione cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Richiamare l'oggetto UDBDataSource collegato usando il nome
        // con cui è stato precedentemente collegato. Nel tempo di esecuzione
        // viene usata solo l'interfaccia DataSource, quindi non c'è bisogno
        // di convertire l'oggetto nella classe di implementazione UDBDataSource.
        // (Non è necessario conoscere qual è la classe di
        // implementazione. Solo il nome JNDI logico è
        // necessario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una volta ottenuto il DataSource, può essere usato per stabilire
        // un collegamento. Questo oggetto Connection è dello stesso tipo di
        // quello restituito se viene utilizzato l'approccio DriverManager
        // per stabilire il collegamento. Quindi, quanto viene riportato da
        // questo punto in poi è esattamente uguale a qualsiasi altra
        // applicazione JDBC.
        Connection connection = ds.getConnection();
```

```
        // Il collegamento può essere usato per creare oggetti Statement e
        // aggiornare il database o elaborare le query come segue.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // Il collegamento viene chiuso prima del termine dell'applicazione.
        connection.close();
    }
}
```

Esempio: ParameterMetaData

Questo è un esempio della modalità di utilizzo dell'interfaccia ParameterMetaData per richiamare le informazioni sui parametri.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////////////////////////////////////////////
//
// Esempio ParameterMetaData. Questo programma illustra
// il nuovo supporto JDBC 3.0 per l'acquisizione delle informazioni
// sui parametri in una PreparedStatement.
//
// Sintassi del comando:
//    java PMD
//
//////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di programma di controllo JDBC IBM Developer per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class PMD {

    // Punto di immissione del programma.
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Ottenere l'installazione.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

```
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.MYTABLE VALUES(?, ?, ?)");
        ParameterMetaData pmd = ps.getParameterMetaData();

        for (int i = 1; i < pmd.getParameterCount(); i++) {
            System.out.println("Parameter number " + i);
            System.out.println("  Class name is " + pmd.getParameterClassName(i));
            // Nota: modalità relativa a input, output o inout
            System.out.println("  Mode is " + pmd.getParameterClassName(i));
            System.out.println("  Type is " + pmd.getParameterType(i));
            System.out.println("  Type name is " + pmd.getParameterTypeName(i));
            System.out.println("  Precision is " + pmd.getPrecision(i));
            System.out.println("  Scale is " + pmd.getScale(i));
            System.out.println("  Nullable? is " + pmd.isNullable(i));
            System.out.println("  Signed? is " + pmd.isSigned(i));
        }
    }
}
```

Esempio: modifica di valori con un'istruzione tramite il cursore di un'altra istruzione

Quest'esempio Java mostra come modificare i valori con un'istruzione tramite il cursore di un'altra istruzione.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class UsingPositionedUpdate {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {

        UsingPositionedUpdate test = new UsingPositionedUpdate();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }



/**
Gestire quanto necessario per il lavoro di impostazione necessario.
**/
    public void setup() {
        try {
            // Registrare il programma di controllo JDBC.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
            } catch (SQLException e) {
                // Ignorare i problemi.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");
```

```
            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
In questa sezione, deve essere aggiunto il codice necessario
per la verifica. Se è necessario solo un collegamento al database,
può essere utilizzata la variabile globale 'connection'.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Aggiornare ogni valore utilizzando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
                                                                   + " CUJOSQL.WHERECUREX
                                                                       SET COL_VALUE = 'CHANGED'
                                                                       WHERE CURRENT OF "
                                                                   + rs.getCursorName ());


            // Eseguire il loop del ResultSet e aggiornare ogni voce.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Ripulire le risorse dopo averle utilizzate.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }



/**
In questa sezione, inserire tutti i lavori di ripulitura per la verifica.
**/
    public void cleanup() {
        try {
            // Chiudere il collegamento globale aperto in setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Caught exception: ");
```

```
            e.printStackTrace();
        }
    }



/**
Visualizzare il contenuto della tabella.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Esempio: interfaccia ResultSet per IBM Developer Kit per Java

Questo è un esempio su come utilizzare l'interfaccia ResultSet.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

/**
ResultSetExample.java

Questo programma mostra l'uso di un ResultSetMetaData e di
una ResultSet per visualizzare tutti i dati nella tabella sebbene
il programma che richiama i dati non sa quali elementi verranno
visualizzati nella tabella (l'utente inoltra i valori per la
tabella e la libreria).
**/
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage:  java ResultSetExample <library> <table>");
            System.out.println(" where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Ottenere un collegamento al database e preparare l'istruzione.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");
```

```
            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
            int rowCount = 0;
            while (rs.next()) {
                rowCount++;
                System.out.println("Data for row " + rowCount);
                for (int i = 1; i <= colCount; i++)
                    System.out.println("   Row " + i + ": " + rs.getString(i));
            }

        } catch (Exception e) {
            // Gestire gli errori.
            System.out.println("Oops... we have an error... ");
            e.printStackTrace();
        } finally {
            // Garantire una ripulitura continua. Se il collegamento viene chiuso,
            // anche l'istruzione presente in esso verrà terminata.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Critical error - cannot close connection object");
                }
            }
        }
    }
}
```

Esempio: sensibilità del ResultSet

Il seguente esempio indica come una modifica è in grado di influire su una clausola where di un'istruzione SQL in base alla sensibilità del ResultSet.

Alcune delle formattazioni di questo esempio possono non essere corrette al fine di inserire questo esempio in una pagina stampata.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class Sensitive2 {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive2 test = new Sensitive2();

        test.setup();
        test.run("sensitive");
        test.cleanup();

        test.setup();
        test.run("insensitive");
        test.cleanup();
    }


    public void setup() {

        try {
```

```
        System.out.println("Native JDBC used");
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        connection = DriverManager.getConnection("jdbc:db2:*local");

        Statement s = connection.createStatement();
        try {
            s.executeUpdate("drop table cujosql.sensitive");
        } catch (SQLException e) {
            // Ignorato.
        }

        s.executeUpdate("create table cujosql.sensitive(col1 int)");
        s.executeUpdate("insert into cujosql.sensitive values(1)");
        s.executeUpdate("insert into cujosql.sensitive values(2)");
        s.executeUpdate("insert into cujosql.sensitive values(3)");
        s.executeUpdate("insert into cujosql.sensitive values(4)");
        s.executeUpdate("insert into cujosql.sensitive values(5)");


        try {
            s.executeUpdate("drop table cujosql.sensitive2");
        } catch (SQLException e) {
            // Ignorato.
        }

        s.executeUpdate("create table cujosql.sensitive2(col2 int)");
        s.executeUpdate("insert into cujosql.sensitive2 values(1)");
        s.executeUpdate("insert into cujosql.sensitive2 values(2)");
        s.executeUpdate("insert into cujosql.sensitive2 values(3)");
        s.executeUpdate("insert into cujosql.sensitive2 values(4)");
        s.executeUpdate("insert into cujosql.sensitive2 values(5)");

        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select col1, col2 From cujosql.sensitive,
                                       cujosql.sensitive2 where col1 = col2");

        rs.next();
        System.out.println("value is " + rs.getInt(1));
```

```
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));

            System.out.println("fetched the four rows...");

            // Un'altra istruzione crea un valore che non corrisponde alla clausola where.
            Statement s2 =
              connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
              ResultSet.CONCUR_UPDATEABLE);
            ResultSet rs2 = s2.executeQuery("select *
            from cujosql.sensitive where col1 = 5 FOR UPDATE");
            rs2.next();
            rs2.updateInt(1, -1);
            rs2.updateRow();
            s2.close();

            if (rs.next()) {
                System.out.println("There is still a row: " + rs.getInt(1));
            } else {
                System.out.println("No more rows.");
            }

        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("----------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other
            than an SQLException was thrown: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Esempio: ResultSet sensibili e non sensibili

L'esempio seguente indica le differenze tra i ResultSet sensibili e non sensibili quando vengono inserite righe in una tabella.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class Sensitive {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive test = new Sensitive();

        test.setup();
        test.run("sensitive");
        test.cleanup();

        test.setup();
        test.run("insensitive");
        test.cleanup();
    }


    public void setup() {

        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("drop table cujosql.sensitive");
            } catch (SQLException e) {
                // Ignorato.
            }

            s.executeUpdate("create table cujosql.sensitive(col1 int)");
            s.executeUpdate("insert into cujosql.sensitive values(1)");
            s.executeUpdate("insert into cujosql.sensitive values(2)");
            s.executeUpdate("insert into cujosql.sensitive values(3)");
            s.executeUpdate("insert into cujosql.sensitive values(4)");
            s.executeUpdate("insert into cujosql.sensitive values(5)");
            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            if (e instanceof SQLException) {
                SQLException another = ((SQLException) e).getNextException();
                System.out.println("Another:  " + another.getMessage());
            }
        }
    }


    public void run(String sensitivity) {
        try {
            Statement s = null;
            if (sensitivity.equalsIgnoreCase("insensitive")) {
                System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
                s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
            } else {
                System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
                s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
            }
```

```
            ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

            // Selezionare i cinque valori presenti.
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            System.out.println("fetched the five rows...");

            // Nota:  se si seleziona l'ultima riga, ResultSet sembra
            //        chiuso e le nuove righe successive che vengono aggiunte
            //        non vengono riconosciute.

            // Consentire un'altra istruzione per l'inserimento di un nuovo valore.
            Statement s2 = connection.createStatement();
            s2.executeUpdate("insert into cujosql.sensitive values(6)");
            s2.close();

            // Se una riga viene riconosciuta si basa sull'impostazione della sensibilità.
            if (rs.next()) {
                System.out.println("There is a row now: " + rs.getInt(1));
            } else {
                System.out.println("No more rows.");
            }


        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("-----------------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

Esempio: impostazione del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource

Questo è un esempio della modalità di utilizzo del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class ConnectionPoolingSetup
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Creare un'implementazione ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object");

        // Stabilire un contesto JNDI e collegare l'origine dati del lotto di collegamenti
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Creare una origine dati standard che vi faccia riferimento.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting pooling");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
  }
}
```

Esempio: SQLException

Questo esempio illustra come rilevare un'SQLException ed eseguire il dump di tutte le informazioni che esso fornisce.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;

public class ExceptionExample {

    public static Connection connection = null;

    public static void main(java.lang.String[] args) {

        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            int count = s.executeUpdate("insert into cujofake.cujofake values(1, 2,3)");

            System.out.println("Did not expect that table to exist.");


        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
```

```
            System.out.println("-----------------------------------------------------");
            e.printStackTrace();
        } catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
            ex.printStackTrace();
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                System.out.println("Exception caught attempting to shutdown...");
            }
        }
    }
}
```

Esempio: sospensione e ripristino di una transazione

Questo è un esempio di transazione sospesa e successivamente ripristinata.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxSuspend {

    public static void main(java.lang.String[] args) {
        JTATxSuspend test = new JTATxSuspend();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.)");

            s.close();
        } finally {
            if (c != null) {
                c.close();
```

```
        }
    }
}


/**
 * Questa verifica usa il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Per transazioni XA, è necessario un identificativo transazione.
        // Un'implementazione dell'interfaccia XID non è inclusa all'unità di
        // di controllo JDBC. Vedere Transazioni con JTA per una
        // descrizione di questa interfaccia per creare la classe relativa ad essa.
        Xid xid = new XidImpl();

        // Il collegamento da XAResource può essere utilizzato come qualsiasi
        // collegamento JDBC.
        Statement     stmt   = c.createStatement();

        // La risorsa XA deve essere notificata prima di avviare qualsiasi
        // elaborazione di transazioni.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Creare un ResultSet durante l'elaborazione JDBC e selezionare una riga.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Il metodo end viene chiamato con l'opzione suspend. I
        // ResultSets associato alla transazione corrente sono 'on hold'.
        // In questo stato non sono né accessibili e né inviabili.
        xaRes.end(xid, XAResource.TMSUSPEND);


        // Con la transazione può essere eseguita un'altra elaborazione. Ad esempio
        // è possibile creare un'istruzione ed elaborare una query. Questa
        // elaborazione e altre elaborazioni di transazioni eseguibili dalla
        // transazione sono separate sa quella effettuata precedentemente con XID.
        Statement nonXAStmt = conn.createStatement();
        ResultSet nonXARS = nonXAStmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
        while (nonXARS.next()) {
            // Elaborazione...
        }
        nonXARS.close();
        nonXAStmt.close();


        // Se si tenta di utilizzare risorse di transazioni sospese,
        // si otterrà un'eccezione.
        try {
            rs.getString(1);
            System.out.println("Value of the first row is " + rs.getString(1));
        } catch (SQLException e) {
            System.out.println("This was an expected exception - " +
```

```
                                    "suspended ResultSet was used.");
            }


            // Riprendere la transazione sospesa e completare l'elaborazione.
            // Il ResultSet è rimasto immutato dall'inizio della sospensione.
            xaRes.start(newXid, XAResource.TMRESUME);
            rs.next();
            System.out.println("Value of the second row is " + rs.getString(1));


            // Quando la transazione è stata completata, chiuderla
            // e eseguire il commit di qualsiasi elaborazione sia presente in essa.
            xaRes.end(xid, XAResource.TMNOFLAGS);
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Something has gone wrong.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

Esempio: ResultSet sospesi

Questo è un esempio del modo in cui un oggetto Statement viene rielaborato sotto un'altra transazione per eseguire il lavoro.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEffect {

    public static void main(java.lang.String[] args) {
        JTATxEffect test = new JTATxEffect();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

```
        c = DriverManager.getConnection("jdbc:db2:*local");
        s = c.createStatement();

        try {
            s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
        } catch (SQLException e) {
            // Ignorare... non esiste
        }

        s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
        s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
        s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.)");

        s.close();
    } finally {
        if (c != null) {
            c.close();
        }
    }
}


/**
 * Questa verifica usa il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Per transazioni XA, è necessario un identificativo transazione.
        // Un'implementazione dell'interfaccia XID non è inclusa all'unità di
        // controllo JDBC. Consultare Transazioni con JTA
        // per una descrizione di questa interfaccia per creare
        // la classe relativa ad essa.
        Xid xid = new XidImpl();

        // Il collegamento da XAResource può essere utilizzato come qualsiasi
        // collegamento JDBC.
        Statement     stmt   = c.createStatement();

        // La risorsa XA deve essere notificata prima di avviare qualsiasi
        // elaborazione di transazioni.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Creare un ResultSet durante l'elaborazione JDBC e selezionare una riga.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Il metodo end viene chiamato con l'opzione suspend. I
        // ResultSets associato alla transazione corrente sono 'on hold'.
        // In questo stato non sono né accessibili e né inviabili.
        xaRes.end(xid, XAResource.TMSUSPEND);


        // Nel frattempo, può essere eseguita un'altra elaborazione all'esterno della transazione
        // I ResultSets sotto la transazione possono essere chiusi se l'oggetto
```

```
            // Statement utilizzato per crearli viene riutilizzato.
            ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
            while (nonXARS.next()) {
                // Elaborazione...
            }


            // Tentare di ritornare alla transazione sospesa. Il ResultSet della
            // transazione sospesa è scomparso in quanto l'istruzione è stata
            // elaborata nuovamente.
            xaRes.start(newXid, XAResource.TMRESUME);
            try {
                rs.next();
            } catch (SQLException ex) {
                System.out.println("This exception is expected. " +
                                   "The ResultSet closed due to another process.");
            }


            // Quando la transazione è stata completata, chiuderla
            // e eseguire il commit di qualsiasi elaborazione sia presente in essa.
            xaRes.end(xid, XAResource.TMNOFLAGS);
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Something has gone wrong.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

Esempio: esecuzione della verifica sulle prestazioni del lotto di collegamenti

Questo è un esempio di come sottoporre a verifica le prestazioni dell'esempio della creazione di lotti rispetto a quelle dell'esempio in cui non vengono creati lotti.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;

public class ConnectionPoolingTest
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        Context ctx = new InitialContext();
        // Eseguire il lavoro senza un lotto:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("\nStart timing the non-pooling DataSource version...");

        long startTime = System.currentTimeMillis();
```

```
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));

        // Eseguire il lavoro con il lotto:
        ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
}
```

Esempio: verifica delle prestazioni di due DataSource

Questo è un esempio di verifica di un DataSource che utilizza solo lotti di collegamenti e di un altro DataSource che utilizza sia lotti di collegamenti che lotti di istruzioni.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class StatementPoolingTest
{

    public static void main(java.lang.String[] args)
    throws Exception
    {
        Context ctx = new InitialContext();

        System.out.println("deploying statement pooling data source");
        deployStatementPoolDataSource();

        // Gestire solo la creazione lotto di collegamenti.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the connection pooling only version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));


        // Gestire la creazione lotti di istruzioni aggiunti.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("\nStart timing the statement pooling version...");

        startTime = System.currentTimeMillis();
```

```
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }



    private static void deployStatementPoolDataSource()
    throws Exception
    {
        // Creare un'implementazione ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object with Statement pooling");
        cpds.setMaxStatements(10);

        // Stabilire un contesto JNDI e collegare l'origine dati del lotto di collegamenti
        Context ctx = new InitialContext();
        ctx.rebind("StatementSupport", cpds);


        // Creare un datasource standard che vi faccia riferimento.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting statement pooling");
        ds.setDataSourceName("StatementSupport");
        ctx.rebind("StatementPoolingDataSource", ds);

    }
}
```

Esempio: aggiornamento di BLOB

Questo è un esempio di come aggiornare i BLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// UpdateBlobs è un'applicazione di esempio
// che mostra alcune API fornendo il supporto
// per modificare gli oggetti Blob e
// riflettere tali modifiche nel
// database.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetBlobs.
////////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();
```

```
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

      rs.next();
      Blob blob1 = rs.getBlob(1);
      rs.next();
      Blob blob2 = rs.getBlob(1);


      // Troncare un BLOB.
      blob1.truncate((long) 150000);
      System.out.println("Blob1's new length is " + blob1.length());

      // Aggiornare parte del BLOB con una nuova schiera di byte.
      // Il seguente codice contiene i byte che si trovano nelle
      // posizioni 4000-4500 e li imposta nelle posizioni 500-1000.

      // Ottenere parte del BLOB come una schiera di byte.
      byte[] bytes = blob1.getBytes(4000L, 4500);

      int bytesWritten = blob2.setBytes(500L, bytes);

      System.out.println("Bytes written is " + bytesWritten);

      // I byte vengono rilevati nella posizione 500 in blob2
      long startInBlob2 = blob2.position(bytes, 1);

      System.out.println("pattern found starting at position " + startInBlob2);

      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: aggiornamento dei CLOB

Questo è un esempio di come aggiornare i CLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////
// UpdateClobs è un'applicazione di esempio
// che mostra alcune API fornendo il supporto
// per modificare gli oggetti Clob e
// riflettere tali modifiche nel
// database.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetClobs.
////////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
```

```
      rs.next();
      Clob clob1 = rs.getClob(1);
      rs.next();
      Clob clob2 = rs.getClob(1);


      // Troncare un CLOB.
      clob1.truncate((long) 150000);
      System.out.println("Clob1's new length is " + clob1.length());

      // Aggiornare una parte del CLOB con un nuovo valore String.
      String value = "Some new data for once";
      int charsWritten = clob2.setString(500L, value);

      System.out.println("Characters written is " + charsWritten);

      // I byte possono essere rilevati nella posizione 500 in clob2
      long startInClob2 = clob2.position(value, 1);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: utilizzo di un collegamento con più transazioni

Questo è un esempio del modo in cui utilizzare un collegamento singolo con più transazioni.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTAMultiTx {

    public static void main(java.lang.String[] args) {
        JTAMultiTx test = new JTAMultiTx();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }
```

```
        s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");

        s.close();
    } finally {
        if (c != null) {
            c.close();
        }
    }
}


/**
 * Questa verifica usa il supporto JTA per gestire le transazioni.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Dal DataSource, ottenere un oggetto XAConnection che
        // contenga un XAResource e un oggetto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();
        Statement     stmt   = c.createStatement();

        // Per transazioni XA, è necessario un identificativo transazione.
        // Ciò non intende implicare che tutti gli XID siano uguali.
        // Ogni XID deve essere univoco per distinguere le diverse transazioni
        // che si verificano.
        // Il supporto per la creazione di XID viene lasciato di nuovo al
        // programma applicativo.
        Xid xid1 = JDXATest.xidFactory();
        Xid xid2 = JDXATest.xidFactory();
        Xid xid3 = JDXATest.xidFactory();

        // Effettuare l'elaborazione tramite tre transazioni per questo collegamento.
        xaRes.start(xid1, XAResource.TMNOFLAGS);
        int count1 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-A')");
        xaRes.end(xid1, XAResource.TMNOFLAGS);

        xaRes.start(xid2, XAResource.TMNOFLAGS);
        int count2 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-B')");
        xaRes.end(xid2, XAResource.TMNOFLAGS);

        xaRes.start(xid3, XAResource.TMNOFLAGS);
        int count3 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-C')");
        xaRes.end(xid3, XAResource.TMNOFLAGS);

        // Preparare tutte le transazioni.
        int rc1 = xaRes.prepare(xid1);
        int rc2 = xaRes.prepare(xid2);
        int rc3 = xaRes.prepare(xid3);

        // Due transazioni sono sottoposte a commit e una a rollback.
        // Il tentativo di inserire il secondo valore nella tabella non
        // è stato sottoposto a commit
        xaRes.commit(xid1, false);
        xaRes.rollback(xid2);
        xaRes.commit(xid3, false);


    } catch (Exception e) {
```

```
            System.out.println("Something has gone wrong.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

Esempio: utilizzo di BLOB

Questo è un esempio di come utilizzare i BLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// UseBlobs è un'applicazione di esempio
// che mostra alcune API associate
// agli oggetti Blob.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetBlobs.
//////////////////////////////////////
import java.sql.*;

public class UseBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

      rs.next();
      Blob blob1 = rs.getBlob(1);
      rs.next();
      Blob blob2 = rs.getBlob(1);


      // Determinare la lunghezza di un LOB.
      long end = blob1.length();
      System.out.println("Blob1 length is " + blob1.length());

      // Quando si gestiscono i LOB, l'indicizzazione che si riferisce ad essi
      // si basa su 1 e non su 0 come per le stringhe e le schiere.
      long startingPoint = 450;
      long endingPoint = 500;

      // Ottenere parte del BLOB come una schiera di byte.
      byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

      // Rilevare dove viene individuato un BLOB secondario o una schiera di byte all'interno di
      // un BLOB. L'impostazione di questo programma inserisce due copie identiche di un
```

```
      // BLOB casuale nel database. Quindi, la posizione iniziale della schiera di byte
      // estratta dal blob1 può essere rilevata nella posizione iniziale
      // nel blob2. L'eccezione si verificherebbe se esistessero 50 byte casuali
      // identici nei LOB in precedenza.
      long startInBlob2 = blob2.position(outByteArray, 1);

      System.out.println("pattern found starting at position " + startInBlob2);

      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: utilizzo di CLOB

Questo è un esempio di come utilizzare i CLOB nelle applicazioni Java.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////
// UpdateClobs è un'applicazione di esempio
// che mostra alcune API fornendo il supporto
// per modificare gli oggetti Clob e
// riflettere tali modifiche nel
// database.
//
// Questo programma deve essere eseguito
// una volta completato il programma PutGetClobs.
//////////////////////////////////////
import java.sql.*;

public class UseClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrare il programma di controllo JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Errore di impostazione.
      }

      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");

      rs.next();
      Clob clob1 = rs.getClob(1);
      rs.next();
      Clob clob2 = rs.getClob(1);


      // Determinare la lunghezza di un LOB.
      long end = clob1.length();
      System.out.println("Clob1 length is " + clob1.length());

      // Quando si gestiscono i LOB, l'indicizzazione che si riferisce ad essi
      // si basa su 1 e non su 0 come per le stringhe e le schiere.
      long startingPoint = 450;
      long endingPoint = 50;

      // Ottenere parte del CLOB come una schiera di byte.
      String outString = clob1.getSubString(startingPoint, (int)endingPoint);
      System.out.println("Clob substring is " + outString);

      // Rilevare dove viene trovato un CLOB secondario o una stringa all'interno di
```

```
      // un CLOB. L'impostazione di questo programma inserisce due copie identiche di un
      // CLOB ripetitivo nel database. Quindi, la posizione iniziale della stringa
      // estratta dal clob1 può essere rilevata nella posizione iniziale
      // nel clob2 se la ricerca inizia vicino alla posizione in cui inizia
   // la stringa.
      long startInClob2 = clob2.position(outString, 440);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // La chiusura del collegamento chiude anche stmt e rs.
   }
}
```

Esempio: utilizzo di JTA per gestire una transazione

Questo è un esempio del modo in cui utilizzare JTA (Java Transaction API) per gestire una transazione in una applicazione.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTACommit {

    public static void main(java.lang.String[] args) {
        JTACommit test = new JTACommit();

        test.setup();
        test.run();
    }


    /**
     * Ripulire l'esecuzione precedente in modo che questa verifica possa riprendere
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignorare... non esiste
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }


    /**
```

```
     * Questa verifica usa il supporto JTA per gestire le transazioni.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Presupporre che l'origine dati venga riportata da un UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // Dal DataSource, ottenere un oggetto XAConnection che
            // contenga un XAResource e un oggetto Connection.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();

            // Per transazioni XA, è necessario un identificativo transazione.
            // Un'implementazione dell'interfaccia XID non è inclusa all'unità di
            // di controllo JDBC. Vedere Transazioni con JTA per una descrizione di
            // questa interfaccia per creare una classe relativa ad essa.
            Xid xid = new XidImpl();

            // Il collegamento da XAResource può essere utilizzato come qualsiasi
            // collegamento JDBC.
            Statement     stmt   = c.createStatement();

            // La risorsa XA deve essere notificata prima di avviare qualsiasi
            // elaborazione di transazioni.
            xaRes.start(xid, XAResource.TMNOFLAGS);

            // Viene eseguito un lavoro JDBC standard.
            int count =
              stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is pretty fun.')");

            // Quando viene completato un lavoro transazione, la risorsa XA deve
            // essere notificata di nuovo.
            xaRes.end(xid, XAResource.TMSUCCESS);

            // La transazione rappresentata dell'ID transazione viene preparata
            // per essere sottoposti a commit.
            int rc = xaRes.prepare(xid);

            // La transazione viene sottoposta a commit tramite XAResource.
            // L'oggetto JDBCConnection non viene usato per eseguire il commit
            // della transazione quando viene utilizzato JTA.
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Something has gone wrong.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

Esempio: utilizzo dei ResultSet di metadati che hanno più di una colonna

Questo rappresenta un esempio del modo in cui utilizzare i Resultset di metadati che possiedono più di una colonna.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
//////////////////////////////////////////////////////////////////////////////
//
// Esempio SafeGetUDTs. Questo programma mostra un modo per gestire i
// ResultSet metadati che hanno più colonne in JDK 1.4 rispetto
// ai release precedenti.
//
// Sintassi del comando:
//    java SafeGetUDTs
//
//////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di programma di controllo JDBC IBM Developer per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class SafeGetUDTs {

    public static int jdbcLevel;

    // Nota: il blocco static viene eseguito prima che inizi main.
    // Quindi, è presente access in jdbcLevel in
    // main.
    {
        try {
            Class.forName("java.sql.Blob");

            try {
                Class.forName("java.sql.ParameterMetaData");
                // Rilevata un'interfaccia JDBC 3.0. Deve supportare JDBC 3.0.
                jdbcLevel = 3;
            } catch (ClassNotFoundException ez) {
                // Impossibile trovare la classe JDBC 3.0 ParameterMetaData.
                // Deve essere in esecuzione sotto una JVM con il solo
                // supporto di JDBC 2.0.
                jdbcLevel = 2;
```

```
            }

        } catch (ClassNotFoundException ex) {
            // Impossibile trovare la classe JDBC 2.0 Blob. Deve essere in
            // esecuzione sotto una JVM con il solo supporto di JDBC 1.0.
            jdbcLevel = 1;
        }
    }

    // Punto di immissione del programma.
    public static void main(java.lang.String[] args)
    {
        Connection c = null;

        try {
            // Richiamare l'unità registrata.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            c = DriverManager.getConnection("jdbc:db2:*local");
            DatabaseMetaData dmd = c.getMetaData();

            if (jdbcLevel == 1) {
                System.out.println("No support is provided for getUDTs.  Just return.");
                System.exit(1);
            }

            ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN%", null);
            while (rs.next()) {

                // Richiamare tutte le colonne disponibili dal release
                // JDBC 2.0.
                System.out.println("TYPE_CAT is " + rs.getString("TYPE_CAT"));
                System.out.println("TYPE_SCHEM is " + rs.getString("TYPE_SCHEM"));
                System.out.println("TYPE_NAME is " + rs.getString("TYPE_NAME"));
                System.out.println("CLASS_NAME is " + rs.getString("CLASS_NAME"));
                System.out.println("DATA_TYPE is " + rs.getString("DATA_TYPE"));
                System.out.println("REMARKS is " + rs.getString("REMARKS"));

                // Selezionare tutte le colonne aggiunte in JDBC 3.0.
                if (jdbcLevel > 2) {
                    System.out.println("BASE_TYPE is " + rs.getString("BASE_TYPE"));
                }
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            if (c != null) {
                try {
                    c.close();
                } catch (SQLException e) {
                    // Ignorare l'eccezione di chiusura.
                }
            }
        }
    }
}
```

Esempio: utilizzo simultaneo di JDBC e JDBC di IBM Toolbox per Java

Questo esempio illustra come utilizzare il collegamento JDBC nativo e il collegamento JDBC di IBM Toolbox per Java in un programma.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle “Informazioni sull’esonero di responsabilità e licenza del codice” a pagina 580.

```
//////////////////////////////////////////////////////////////////////////////
//
// Esempio di GetConnections.
//
// Questo programma è in grado di utilizzare entrambi i programmi di controllo
// JDBC in un programma contemporaneamente. Vengono creati due oggetti Connection
// in questo programma applicativo. Uno è un collegamento JDBC nativo e uno è un collegamento JDBC
// IBM Toolbox per Java.
//
// Questa tecnica è appropriata in quanto consente di utilizzare diversi programmi di
// controllo JDBC per diverse attività contemporaneamente. Ad esempio, il programma di
// controllo JDBC di IBM Toolbox per Java è ideale per il collegamento ad un System i5 remoto
// ed il programma di controllo JDBC nativo è più veloce per i collegamenti locali.
// È possibile utilizzare le potenzialità di ogni programma di controllo contemporaneamente
// nella propria applicazione scrivendo un codice simile a questo esempio.
//
//////////////////////////////////////////////////////////////////////////////
//
// Questo sorgente è un esempio di programma di controllo JDBC IBM Developer per Java.
// IBM concede una licenza non esclusiva per utilizzare ciò come esempio
// da cui creare funzioni simili personalizzate, in base a
// richieste specifiche.
//
// Questo esempio è fornito da IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità
// implicita, rispetto alla funzionalità o alle funzioni di questi programmi.
//
// Tutti i programmi qui contenuti vengono forniti all'utente "COSI' COME SONO"
// senza garanzie di alcun tipo. Tutte le garanzie di commerciabilità
// e adeguatezza a scopi specifici sono esplicitamente
// vietate.
//
// IBM Developer Kit per Java
// (C) Copyright IBM Corp. 2001
// Tutti i diritti riservati.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////
import java.sql.*;
import java.util.*;

public class GetConnections {

   public static void main(java.lang.String[] args)
   {
       // Verificare l'immissione.
       if (args.length != 2) {
          System.out.println("Usage (CL command line):  java GetConnections PARM(<user> <password>)");
          System.out.println(" where <user> is a valid System i5 user ID");
          System.out.println("   and <password> is the password for that user ID");
          System.exit(0);
       }

       // Registrare entrambi i programmi di controllo.
      try {
         Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
         Class.forName("com.ibm.as400.access.AS400JDBCDriver");
      } catch (ClassNotFoundException cnf) {
         System.out.println("ERROR: One of the JDBC drivers did not load.");
         System.exit(0);
      }

      try {
         // Ottenere un collegamento con ogni programma di controllo.
```

```
         Connection conn1 = DriverManager.getConnection("jdbc:db2://localhost", args[0], args[1]);
         Connection conn2 = DriverManager.getConnection("jdbc:as400://localhost", args[0], args[1]);

         // Verificare che siano diversi.
         if (conn1 instanceof com.ibm.db2.jdbc.app.DB2Connection)
            System.out.println("conn1 is running under the native JDBC driver.");
         else
            System.out.println("There is something wrong with conn1.");

         if (conn2 instanceof com.ibm.as400.access.AS400JDBCConnection)
            System.out.println("conn2 is running under the IBM Toolbox for Java JDBC driver.");
         else
            System.out.println("There is something wrong with conn2.");

         conn1.close();
         conn2.close();
      } catch (SQLException e) {
         System.out.println("ERROR: " + e.getMessage());
      }
   }
}
```

Esempio: utilizzo di PreparedStatement per ottenere un ResultSet

Questo rappresenta un esempio dell'utilizzo del metodo executeQuery di un oggetto PreparedStatement per ottenere un ResultSet.

Nota: utilizzando gli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import java.util.Properties;

public class PreparedStatementExample {

    public static void main(java.lang.String[] args)
    {
        // Caricare quanto segue da un oggetto proprietà.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Registrare il programma di controllo JDBC nativo. Se il programma di controllo non può
        // essere registrato, la verifica non può continuare.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        //    Questo programma crea una tabella utilizzata
        //    successivamente dalle istruzioni preparate.
        try {
            // Creare le proprietà del collegamento.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Collegarsi al database locale.
            c = DriverManager.getConnection(URL, properties);

            // Creare un oggetto Statement.
            s = c.createStatement();
```

```
    // Cancellare la tabella di verifica se esistente. Notare che
    // questo esempio presuppone che l'intera MYLIBRARY della
    // raccolta esista sul sistema.
    try {
        s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
    } catch (SQLException e) {
        // Continuare... probabilmente la tabella non esiste.
    }

    // Eseguire un'istruzione SQL che crea una tabella nel database.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Chiudere le risorse del database
    try {
        if (s != null) {
            s.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}


//    Questo programma utilizza poi un'istruzione preparata per inserire
//    molte righe nel database.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Creare un oggetto PreparedStatement utilizzato per inserire dati
    // nella tabella.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

    for (int i = 0; i < nameArray.length; i++) {
        ps.setString(1, nameArray[i]);      // Impostare il nome dalla propria schiera.
        ps.setInt(2, i+1);                  // Impostare l'ID.
        ps.executeUpdate();
    }

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Chiudere le risorse del database
    try {
        if (ps != null) {
            ps.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}


//    Utilizzare un'istruzione preparata per interrogare la tabella
//    database creata e restituire i dati che derivano da essa. In
//    questo esempio, il parametro usato viene impostato arbitrariamente
//    su 5, ciò indica che vengono restituite tutte le righe in cui il campo ID
//    sia minore o uguale a 5.
try {
    ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
                            "WHERE ID <= ?");
```

```
            ps.setInt(1, 5);

            // Eseguire una query SQL sulla tabella.
            ResultSet rs = ps.executeQuery();
            // Visualizzare tutti i dati nella tabella.
            while (rs.next()) {
                System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("Database processing has failed.");
            System.out.println("Reason: " + sqle.getMessage());
        } finally {
            // Chiudere le risorse del database
            try {
                if (ps != null) {
                    ps.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Statement.");
            }

            try {
                if (c != null) {
                    c.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Connection.");
            }

        }
    }
}
```

Esempio: utilizzo del metodo executeUpdate dell'oggetto Statement

Questo è un esempio sul modo in cui utilizzare il metodo executeUpdate dell'oggetto Statement.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import java.util.Properties;

public class StatementExample {

    public static void main(java.lang.String[] args)
    {

        // Suggerimento: caricare quanto segue dall'oggetto proprietà.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Registrare il programma di controllo JDBC nativo. Se il programma di controllo non può
        // essere registrato, la verifica non può continuare.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        try {
```

```
            // Creare le proprietà del collegamento.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Collegarsi al database System i5 locale.
            c = DriverManager.getConnection(URL, properties);

            // Creare un oggetto Statement.
            s = c.createStatement();
            // Cancellare la tabella verifiche se presente. Nota: questo
            // esempio presuppone che MYLIBRARY della raccolta
            // esista sul sistema.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Continuare... probabilmente la tabella non esiste.
            }

            // Eseguire un'istruzione SQL che crea una tabella nel database.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

            // Eseguire alcune istruzioni SQL che inseriscono record nella tabella.
            s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('RICH', 123)");
            s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('FRED', 456)");
            s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('MARK', 789)");

            // Eseguire una query SQL sulla tabella.
            ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

            // Visualizzare tutti i dati nella tabella.
            while (rs.next()) {
                System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("Database processing has failed.");
            System.out.println("Reason: " + sqle.getMessage());
        } finally {
            // Chiudere le risorse del database
            try {
                if (s != null) {
                    s.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Statement.");
            }
             }

            try {
                if (c != null) {
                    c.close();
                }
            } catch (SQLException e) {
                System.out.println("Cleanup failed to close Connection.");
            }
        }
    }
}
```

Esempi: HelloWorld JAAS

Questi esempi mostrano all'utente i tre file necessari per compilare ed eseguire HelloWorld per JAAS.

HelloWorld.java

Segue l'origine per il file HelloWorld.java.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/*
 * ===========================================================================
 * Materiale su licenza - Proprietà di IBM
 *
 * (C) Copyright IBM Corp. 2000 Tutti i diritti riservati.
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * ===========================================================================
 *
 * File: HelloWorld.java
 */

import java.io.*;
import java.util.*;
import java.security.Principal;
import java.security.PrivilegedAction;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;

/**
 * Questa applicazione SampleLogin tenta di autenticare un utente.
 *
 * Se l'utente è riuscito ad autenticarsi,
 * viene visualizzato il nome utente e il numero delle credenziali.
 *
 * @version 1.1, 09/14/99
 */
public class HelloWorld {

    /**
     * Tentativo di autenticazione dell'utente.
     */
    public static void main(String[] args) {
        // usare il LoginModules configurato per la voce "helloWorld"
        LoginContext lc = null;
        try {
            lc = new LoginContext("helloWorld", new MyCallbackHandler());
        } catch (LoginException le) {
            le.printStackTrace();
            System.exit(-1);
        }

        // l'utente ha 3 tentativi per riuscire ad autenticarsi
        int i;
        for (i = 0; i < 3; i++) {
            try {

                // tentativo di autenticazione
                lc.login();

                // se non vengono riportati errori, l'autenticazione è riuscita
                break;

            } catch (AccountExpiredException aee) {

                System.out.println("Your account has expired");
                System.exit(-1);
```

```
            } catch (CredentialExpiredException cee) {

                System.out.println("Your credentials have expired.");
                System.exit(-1);

            } catch (FailedLoginException fle) {

                System.out.println("Authentication Failed");
                    try {
                        Thread.currentThread().sleep(3000);
                    } catch (Exception e) {
                        // ignorare
                    }

            } catch (Exception e) {

                System.out.println("Unexpected Exception - unable to continue");
                e.printStackTrace();
                System.exit(-1);
            }
        }

        // Tutti e 3 i tentativi hanno avuto esito negativo?
        if (i == 3) {
            System.out.println("Sorry");
            System.exit(-1);
        }

        // Consultare gli elementi a disposizione dei Principal:
        Iterator principalIterator = lc.getSubject().getPrincipals().iterator();
        System.out.println("\n\nAuthenticated user has the following Principals:");
        while (principalIterator.hasNext()) {
            Principal p = (Principal)principalIterator.next();
            System.out.println("\t" + p.toString());
        }

        // Consultare il lavoro basato sul Principal:
        Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
            public Object run() {
                System.out.println("\nYour java.home property: "
                                +System.getProperty("java.home"));

                System.out.println("\nYour user.home property: "
                                +System.getProperty("user.home"));

                File f = new File("foo.txt");
                System.out.print("\nfoo.txt does ");
                if (!f.exists()) System.out.print("not ");
                System.out.println("exist in your current directory");

                System.out.println("\nOh, by the way ...");

                try {
                    Thread.currentThread().sleep(2000);
                } catch (Exception e) {
                    // ignorare
                }
                System.out.println("\n\nHello World!\n");
                return null;
            }
        }, null);
        System.exit(0);
    }
}
```

```
/**
 * L'applicazione deve implementare CallbackHandler.
 *
 * Questa applicazione si basa sul testo. Quindi visualizza le informazioni
 * all'utente utilizzando OutputStreams System.out e System.err,
 * e raccoglie l'immissione per l'utente utilizzando InputStream, System.in.
 */
class MyCallbackHandler implements CallbackHandler {

    /**
     * Richiamare una schiera di Callback.
     *
     *
     * @param richiama una schiera di oggetti Callback che contengono le
     *          informazioni richieste dal servizio di sicurezza sottostante
     *          che devono essere richiamate o visualizzate.
     *
     * @exception java.io.IOException se si verifica un errore di immissione o emissione.
     *
     * @exception UnsupportedCallbackException se l'implementazione di questo
     *          metodo non supporta uno o più Callback specificati
     *          nel parametro callbacks.
     */
    public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException {

    for (int i = 0; i < callbacks.length; i++) {
        if (callbacks[i] instanceof TextOutputCallback) {

        // visualizzare il messaggio in base al tipo specificato
        TextOutputCallback toc = (TextOutputCallback)callbacks[i];
        switch (toc.getMessageType()) {
        case TextOutputCallback.INFORMATION:
            System.out.println(toc.getMessage());
            break;
        case TextOutputCallback.ERROR:
            System.out.println("ERROR: " + toc.getMessage());
            break;
        case TextOutputCallback.WARNING:
            System.out.println("WARNING: " + toc.getMessage());
            break;
        default:
            throw new IOException("Unsupported message type: " +
                    toc.getMessageType());
        }

        } else if (callbacks[i] instanceof NameCallback) {

        // richiedere all'utente un nome utente
        NameCallback nc = (NameCallback)callbacks[i];

        // ignorare il defaultName fornito
        System.err.print(nc.getPrompt());
        System.err.flush();
        nc.setName((new BufferedReader
            (new InputStreamReader(System.in))).readLine());

        } else if (callbacks[i] instanceof PasswordCallback) {

        // richiedere all'utente informazioni sensibili
        PasswordCallback pc = (PasswordCallback)callbacks[i];
        System.err.print(pc.getPrompt());
        System.err.flush();
        pc.setPassword(readPassword(System.in));

        } else {
        throw new UnsupportedCallbackException
```

```
            (callbacks[i], "Unrecognized Callback");
        }
    }
    }

    // Legge la parola d'ordine utente da un determinato flusso di immissione.
    private char[] readPassword(InputStream in) throws IOException {

    char[] lineBuffer;
    char[] buf;
    int i;

    buf = lineBuffer = new char[128];

    int room = buf.length;
    int offset = 0;
    int c;

    loop:   while (true) {
        switch (c = in.read()) {
        case -1:
        case '\n':
        break loop;

        case '\r':
        int c2 = in.read();
        if ((c2 != '\n') && (c2 != -1)) {
            if (!(in instanceof PushbackInputStream)) {
            in = new PushbackInputStream(in);
            }
            ((PushbackInputStream)in).unread(c2);
        } else
            break loop;

        default:
        if (--room < 0) {
            buf = new char[offset + 128];
            room = buf.length - offset - 1;
            System.arraycopy(lineBuffer, 0, buf, 0, offset);
            Arrays.fill(lineBuffer, ' ');
            lineBuffer = buf;
        }
        buf[offset++] = (char) c;
        break;
        }
    }

    if (offset == 0) {
        return null;
    }

    char[] ret = new char[offset];
    System.arraycopy(buf, 0, ret, 0, offset);
    Arrays.fill(buf, ' ');

    return ret;
    }
}
```

HWLoginModule.java

Segue l'origine per HWLoginModule.java.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/*
 * ===========================================================================
 * Materiale su licenza - Proprietà di IBM
 *
 * (C) Copyright IBM Corp. 2000 Tutti i diritti riservati.
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * ===========================================================================
 *
 * File: HWLoginModule.java
 */

package com.ibm.security;

import java.util.*;
import java.io.IOException;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;
import com.ibm.security.HWPrincipal;

/**
 * Questo LoginModule autentica gli utenti con una parola d'ordine.
 *
 * Questo LoginModule riconosce solo un utente che ha immesso
 *     la parola d'ordine richiesta:   Go JAAS
 *
 * Se l'utente è riuscito ad autenticarsi,
 * un HWPrincipal con il nome utente
 * viene aggiunto a Subject.
 *
 * Questo LoginModule riconosce l'opzione di debug.
 * Se impostato su true nella configurazione del collegamento,
 * vengono inviati messaggi di debug al flusso di emissione, System.out.
 *
 * @version 1.1, 09/10/99
 */
public class HWLoginModule implements LoginModule {

    // stato iniziale
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;

    // opzione configurabile
    private boolean debug = false;

    // lo stato di autenticazione
    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    // nome utente e parola d'ordine
    private String user name;
    private char[] password;

    private HWPrincipal userPrincipal;

    /**
     * Inizializzare questo LoginModule.
     *
     * @param subject il soggetto da autenticare.
     *
     * @param callbackHandler un CallbackHandler per comunicare con
     *          l'utente finale (richiedendo nomi utente e
```

```
 *          parole d'ordine, ad esempio).
 *
 * @param sharedState stato LoginModule condiviso.
 *
 * @param options opzioni specificate nella configurazione del
 *          collegamento per questo particolare
 *          LoginModule.
 */
public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options) {

this.subject = subject;
this.callbackHandler = callbackHandler;
this.sharedState = sharedState;
this.options = options;

// inizializzare le opzioni configurate
debug = "true".equalsIgnoreCase((String)options.get("debug"));
}

/**
 * Autenticare l'utente richiedendo un nome utente e una parola d'ordine.
 *
 *
 * @return true in tutti i casi in quanto questo LoginModule
 *      non deve essere ignorato.
 *
 * @exception FailedLoginException se l'autenticazione ha esito negativo.
 *
 * @exception LoginException se questo LoginModule
 *      non è in grado di eseguire l'autenticazione.
 */
public boolean login() throws LoginException {

// richiedere un nome utente e una parola d'ordine
if (callbackHandler == null)
    throw new LoginException("Error: no CallbackHandler available " +
        "to garner authentication information from the user");

Callback[] callbacks = new Callback[2];
callbacks[0] = new NameCallback("\n\nHWModule user name: ");
callbacks[1] = new PasswordCallback("HWModule password: ", false);

try {
    callbackHandler.handle(callbacks);
    user name = ((NameCallback)callbacks[0]).getName();
    char[] tmpPassword = ((PasswordCallback)callbacks[1]).getPassword();
    if (tmpPassword == null) {
    // gestire una parola d'ordine NULL come una parola d'ordine vuota
    tmpPassword = new char[0];
    }
    password = new char[tmpPassword.length];
    System.arraycopy(tmpPassword, 0,
        password, 0, tmpPassword.length);
    ((PasswordCallback)callbacks[1]).clearPassword();

} catch (java.io.IOException ioe) {
    throw new LoginException(ioe.toString());
} catch (UnsupportedCallbackException uce) {
    throw new LoginException("Error: " + uce.getCallback().toString() +
    " not available to garner authentication information " +
    "from the user");
}

// stampare le informazioni di debug
if (debug) {
    System.out.println("\n\n\t[HWLoginModule] " +
```

```
            "user entered user name: " +
            user name);
    System.out.print("\t[HWLoginModule] " +
            "user entered password: ");
    for (int i = 0; i > password.length; i++)
    System.out.print(password[i]);
    System.out.println();
}

// verificare la parola d'ordine
if (password.length == 7 &&
    password[0] == 'G' &&
    password[1] == 'o' &&
    password[2] == ' ' &&
    password[3] == 'J' &&
    password[4] == 'A' &&
    password[5] == 'A' &&
    password[6] == 'S') {

    // autenticazione riuscita!!!
    if (debug)
    System.out.println("\n\t[HWLoginModule] " +
            "authentication succeeded");
    succeeded = true;
    return true;
} else {

    // autenticazione non riuscita -- ripulire lo stato
    if (debug)
    System.out.println("\n\t[HWLoginModule] " +
            "authentication failed");
    succeeded = false;
    user name = null;
    for (int i = 0; i < password.length; i++)
    password[i] = ' ';
    password = null;
    throw new FailedLoginException("Password Incorrect");
}
}

/**
 * Questo metodo viene chiamato se l'autenticazione globale di LoginContext
 * ha avuto esito positivo
 * (REQUIRED, REQUISITE, SUFFICIENT e OPTIONAL LoginModules pertinenti
 * riusciti).
 *
 * Se questo tentativo di autenticazione LoginModule ha esito positivo
 * (controllato richiamando lo stato privato salvato dal metodo
 * login), questo metodo associa un
 * SolarisPrincipal
 * al Subject presente nel
 *          LoginModule.  Se questo tentativo di autenticazione di LoginModule
 * ha esito negativo, questo metodo elimina qualsiasi
 * stato originariamente salvato.
 *
 * @exception LoginException se il commit ha esito negativo.
 *
 * @return true se i tentativi di collegamento e commit di LoginModule
 *      o false in caso contrario.
 */
public boolean commit() throws LoginException {
if (succeeded == false) {
    return false;
} else {
    // aggiungere un Principal (identità autenticata)
    // al Subject
```

```
    // presupporre che l'utente che autentichiamo sia HWPrincipal
    userPrincipal = new HWPrincipal(user name);
    final Subject s = subject;
    final HWPrincipal sp = userPrincipal;
    java.security.AccessController.doPrivileged
    (new java.security.PrivilegedAction() {
    public Object run() {
        if (!s.getPrincipals().contains(sp))
        s.getPrincipals().add(sp);
        return null;
    }
    });

    if (debug) {
    System.out.println("\t[HWLoginModule] " +
            "added HWPrincipal to Subject");
    }

    // in qualsiasi caso, ripulire lo stato
    user name = null;
    for (int i = 0; i > password.length; i++)
    password[i] = ' ';
    password = null;

    commitSucceeded = true;
    return true;
}
}

/**
 * Questo metodo viene chiamato se l'autenticazione globale di LoginContext
 * ha esito negativo.
 * (REQUIRED, REQUISITE, SUFFICIENT e OPTIONAL LoginModules pertinenti
 * non riusciti).
 *
 * Se questo tentativo di autenticazione LoginModule ha esito negativo
 * (controllato richiamando lo stato privato salvato dal metodo
 * di collegamento e dal metodo di commit),
 * questo metodo ripulisce qualsiasi stato originariamente salvato.
 *
 * @exception LoginException se l'interruzione ha esito negativo.
 *
 * @return false se questo tentativo di collegamento o commit per LoginModule
 *      ha esito negativo e true in caso contrario.
 */
public boolean abort() throws LoginException {
if (succeeded == false) {
    return false;
} else if (succeeded == true && commitSucceeded == false) {
    // collegamento riuscito ma autenticazione globale non riuscita
    succeeded = false;
    user name = null;
    if (password != null) {
    for (int i = 0; i > password.length; i++)
        password[i] = ' ';
    password = null;
    }
    userPrincipal = null;
} else {
    // autenticazione globale e commit riuscite,
    // ma un altro commit ha avuto esito negativo.
    logout();
}
return true;
}

/**
```

```
     * Scollegare l'utente.
     *
     * Questo metodo elimina l'HWPrincipal
     * aggiunto dal metodo commit.
     *
     * @exception LoginException se lo scollegamento non riesce.
     *
     * @return true in tutti i casi in quanto questo LoginModule
     *      non deve essere ignorato.
     */
    public boolean logout() throws LoginException {

    final Subject s = subject;
    final HWPrincipal sp = userPrincipal;
    java.security.AccessController.doPrivileged
        (new java.security.PrivilegedAction() {
        public Object run() {
        s.getPrincipals().remove(sp);
        return null;
        }
    });

    succeeded = false;
    succeeded = commitSucceeded;
    user name = null;
    if (password != null) {
        for (int i = 0; i > password.length; i++)
        password[i] = ' ';
        password = null;
    }
    userPrincipal = null;
    return true;
    }
}
```

HWPrincipal.java

Segue l'origine per HWPrincipal.java.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/*
 * ================================================================================
 * Materiale su licenza - Proprietà di IBM
 *
 * (C) Copyright IBM Corp. 2000 Tutti i diritti riservati.
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * ================================================================================
 *
 * File: HWPrincipal.java
 */

package com.ibm.security;

import java.security.Principal;

/**
 * questa classe implementa l'interfaccia Principal
 * e rappresenta il tester HelloWorld.
 *
 * @version 1.1, 09/10/99
 * @author D. Kent Soper
 */
public class HWPrincipal implements Principal, java.io.Serializable {
```

```
    private String name;

    /*
     * Creare un HWPrincipal con il nome fornito.
     */
    public HWPrincipal(String name) {
        if (name == null)
            throw new NullPointerException("illegal null input");

        this.name = name;
    }

    /*
     * Restituire il nome del HWPrincipal.
     */
    public String getName() {
        return name;
    }

    /*
     * Restituire una rappresentazione di stringa di HWPrincipal.
     */
    public String toString() {
        return("HWPrincipal:  " + name);
    }

    /*
     * Confronta l'oggetto specificato con HWPrincipal per uguaglianza.
     * Restituisce true se il particolare oggetto è anche un HWPrincipal e i
     * due HWPrincipals hanno lo stesso nome utente.
     */
    public boolean equals(Object o) {
        if (o == null)
            return false;

        if (this == o)
            return true;

        if (!(o instanceof HWPrincipal))
            return false;
        HWPrincipal that = (HWPrincipal)o;

        if (this.getName().equals(that.getName()))
            return true;
        return false;
    }

    /*
     * Restituire un codice hash per HWPrincipal.
     */
    public int hashCode() {
        return name.hashCode();
    }
}
```

Esempio: JAAS SampleThreadSubjectLogin

Questo esempio mostra l'implementazione della classe SampleThreadSubjectLogin.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
////////////////////////////////////////////////////////////////////////
//
// 5761-JV1
//
////////////////////////////////////////////////////////////////////////
```

```
//
// Nome file:   SampleThreadSubjectLogin.java
//
// Classe:      SampleThreadSubjectLogin
//
//////////////////////////////////////////////////////////////////
//
// ATTIVITÀ DI MODIFICA:
//
//
// FINE ATTIVITÀ DI MODIFICA
//
//////////////////////////////////////////////////////////////////

import com.ibm.security.auth.ThreadSubject;

import com.ibm.as400.access.*;

import java.io.*;

import java.util.*;

import java.security.Principal;

import javax.security.auth.*;

import javax.security.auth.callback.*;

import javax.security.auth.login.*;


/**
 * Questa applicazione SampleThreadSubjectLogin autentica un singolo
 * utente, scambia l'identità del sottoprocesso OS nell'utente autenticato
 * e quindi scrive "Hello World" in un file autorizzato privatamente,
 * thread.txt, nell'indirizzario di verifica dell'utente.
 *
 * L'utente deve immettere l'id utente e la parola d'ordine per
 * autenticarsi.
 *
 * Se tale operazione riesce, vengono visualizzati il nome utente e
 * il numero delle credenziali.
 *
 *

 Istruzioni di impostazione e di esecuzione:

  1) Creare un nuovo utente, JAAS14, richiamando
 "CRTUSRPRF USRPRF(JAAS14) PASSWORD() TEXT('JAAS sample user id')"
 con l'autorizzazione della classe *USER.

 2) Assegnare un file di verifica fittizia
"yourTestDir/thread.txt", e
 concedere privatamente a JAAS14 l'autorizzazione *RWX ad esso per l'accesso alla scrittura.

 3) Copiare SampleThreadSubjectLogin.java nell'indirizzario di
verifica.

 4) Modificare l'indirizzario corrente sull'indirizzario di
verifica e compilare il
 codice sorgente java.

   Immettere -

   strqsh

   cd 'yourTestDir'
```

```
   javac -J-Djava.version=1.4
     -classpath /qibm/proddata/os400/java400/ext/jaas14.jar:
                /QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar:.
     -d ./classes
     *.java

 5) Copiare threadLogin.config, threadJaas.policy e threadJava2.policy
 nell'indirizzario di verifica.

 6) Se non è stato già eseguito, aggiungere il collegamento simbolico all'indirizzario
 dell'estensione per il file jaas14.jar.
 Il programma di caricamento classi di estensioni deve di norma caricare il file JAR.

 ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas14.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jaas14.jar')


 7) Se non è stato già eseguito questo esempio, aggiungere il
collegamento simbolico all'indirizzario
 dell'estensione per i file jt400.jar e jt400ntv.jar. Questo determina
 il caricamento dei file da parte del programma di caricamento
classi di estensioni. È possibile inoltre che il programma di
caricamento classi delle applicazioni
 carichi questi file includendoli nel CLASSPATH.
 Se questi file vengono caricati dall'indirizzario del percorso della
classe,
 non aggiungere il collegamento simbolico all'indirizzario
dell'estensione.
 Il file jaas14.jar richiede questi file JAR per le classi
 di implementazione delle credenziali che sono parte del prodotto del programma su licenza
 di IBM Toolbox per Java (5761-JC1).
 (Consultare l'argomento IBM Toolbox per Java per la documentazione
 sulle classi di credenziali rilevate nel segmento di sinistra
 sotto Classi Security => Autenticazione. Selezionare il
collegamento alla
 classe ProfileTokenCredential. Sulla parte superiore selezionare
 'Questo pacchetto' per
 l'intero pacchetto Java com/ibm/as400/security/auth. È
possibile inoltre rilevare Javadoc per le
 classi di autenticazione tramite la selezione 'Javadoc' =>
 'Classi Access' sul segmento sinistro. Selezionare 'Tutti i
pacchetti' sulla parte superiore
 e individuare i pacchetti com.ibm.as400.security.*)


 ADDLNK OBJ('/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400.jar')

 ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')

////////////////////////////////////
 NOTE IMPORTANTI -
////////////////////////////////////

 Quando si aggiornano i file sulla politica di Java2
 per un'applicazione reale tenere presente la concessione dei
 permessi appropriati per le ubicazioni attuali dei file JAR di IBM Toolbox
 per Java. Sebbene questi sono simbolicamente collegati agli
 indirizzari di estensione precedentemente elencati cui sono concessi
 java.security.AllPermission nel file
 ${java.home}/lib/security/java.policy file, l'autorizzazione è basata sulla
 ubicazione reale dei file JAR.

 Ad esempio, per utilizzare regolarmente le classi di credenziali
 in IBM Toolbox per Java, bisogna aggiungere quanto segue al file
```

```
 delle politiche Java2 dell'applicazione -

 grant  codeBase "file:/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar"
 {
   permission javax.security.auth.AuthPermission "modifyThreadIdentity";
   permission java.lang.RuntimePermission "loadLibrary.*";
   permission java.lang.RuntimePermission "writeFileDescriptor";
   permission java.lang.RuntimePermission "readFileDescriptor";
 }

 È inoltre necessario aggiungere questi permessi al codeBase dell'applicazione
 poiché le operazioni eseguite dai file JAR dell'IBM Toolbox
 per Java non vengono eseguite in modalità privilegiata.

 Questo esempio concede già questi permessi a tutte le classi java
 omettendo il parametro codeBase nel file threadJava2.policy.

 8) Assicurarsi che i Server host siano avviati e in esecuzione.
 Le classi ProfileTokenCredential che risiedono in IBM Toolbox per Java,
 cioè jt400.jar, vengono utilizzate come credenziali collegate
 al soggetto autenticato dal programma SampleThreadSubjectLogin.java.
 Le classi di credenziali di IBM Toolbox per Java richiedono l'accesso
 ai Server host.

 9) Richiamare SampleThreadSubjectLogin durante il collegamento come utente che
 non possiede alcun accesso a 'yourTestDir/thread.txt'.

 10) Avviare l'esempio immettendo i seguenti comandi CL  =>

 CHGCURDIR DIR('yourTestDir')

 JAVA CLASS(SampleThreadSubjectLogin)
  CLASSPATH('yourTestDir/classes')
  PROP((java.version '1.3')
       (java.security.manager)
       (java.security.auth.login.config
        'yourTestDir/threadLogin.config')
       (java.security.policy
        'yourTestDir/threadJava2.policy')
       (java.security.auth.policy
        'yourTestDir/threadJaas.policy'))

 Immettere l'id utente e la parola d'ordine quando richiesto dal passaggio 1.

 11) Controllare yourTestDir/thread.txt per la voce "Hello World".

 *
 **/


public class SampleThreadSubjectLogin {
/**
 * Tentativo di autenticazione dell'utente.
 *
 * @param args
 *      Argomenti di immissione per questa applicazione (ignorato).
 *
 */
    public static void main(String[] args) {

    // usare il LoginModules configurato per la voce "AS400ToolboxApp"
    LoginContext lc = null;
    try {
        // se fornito, viene usato lo stesso subject per più tentativi di collegamento
        lc = new LoginContext("AS400ToolboxApp",
                  new Subject(),
                  new SampleCBHandler());
```

```
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}

// l'utente ha 3 tentativi per riuscire ad autenticarsi
int i;
for (i = 0; i < 3; i++) {
    try {

    // tentativo di autenticazione
    lc.login();

    // se non vengono riportati errori, l'autenticazione è riuscita
    break;

    } catch (AccountExpiredException aee) {

    System.out.println("Your account has expired");
    System.exit(-1);

    } catch (CredentialExpiredException cee) {

    System.out.println("Your credentials have expired.");
    System.exit(-1);

    } catch (FailedLoginException fle) {

    System.out.println("Authentication Failed");
    try {
        Thread.currentThread().sleep(3000);
    } catch (Exception e) {
        // ignorare
    }

    } catch (Exception e) {

    System.out.println("Unexpected Exception - unable to continue");
    e.printStackTrace();
    System.exit(-1);
    }
}

// Tutti e 3 i tentativi hanno auto esito negativo?
if (i == 3) {
    System.out.println("Sorry authentication failed");
    System.exit(-1);
}


// visualizzare i principal & le credenziali autenticate
System.out.println("Authentication Succeeded");

System.out.println("Principals:");

Iterator itr = lc.getSubject().getPrincipals().iterator();

while (itr.hasNext())
    System.out.println(itr.next());

itr = lc.getSubject().getPrivateCredentials().iterator();

while (itr.hasNext())
    System.out.println(itr.next());

itr = lc.getSubject().getPublicCredentials().iterator();
```

```
    while (itr.hasNext())
        System.out.println(itr.next());


        // eseguire un'elaborazione basata sul Principal:
    ThreadSubject.doAsPrivileged(lc.getSubject(), new java.security.PrivilegedAction() {
           public Object run() {
           System.out.println("\nYour java.home property: "
                  +System.getProperty("java.home"));
           System.out.println("\nYour user.home property: "
                     +System.getProperty("user.home"));
           File f = new File("thread.txt");
           System.out.print("\nthread.txt does ");
           if (!f.exists()) System.out.print("not ");
           System.out.println("exist in your current directory");

           try {
               // scrivere "Hello World number   x"  in thread.txt
               PrintStream ps = new PrintStream(new FileOutputStream("thread.txt", true), true);

               long flen = f.length();
               ps.println("Hello World number    " +
                  Long.toString(flen/22) +
                  "\n");
               ps.close();
           } catch (Exception e) {
               e.printStackTrace();
           }

           System.out.println("\nOh, by the way, " + SampleThreadSubjectLogin.getCurrentUser());
           try {
               Thread.currentThread().sleep(2000);
           } catch (Exception e) {
            // ignorare
               }
           System.out.println("\n\nHello World!\n");
           return null;
           }
       }, null);

       System.exit(0);

    }// end main()


// Restituisce l'identità OS corrente per il sottoprocesso principale dell'applicazione.
// (Questa routine usa le classi che derivano da IBM Toolbox per Java)
// Nota - Le applicazioni in esecuzione su un sottoprocesso secondario non possono
// utilizzare questa API per determinare l'utente corrente.
    static public String getCurrentUser() {

    try {
        AS400 localSys = new AS400("localhost", "*CURRENT", "*CURRENT");

        int ccsid = localSys.getCcsid();
        ProgramCall qusrjobi = new ProgramCall(localSys);
        ProgramParameter[] parms = new ProgramParameter[6];

        int rLength = 100;
        parms[0] = new ProgramParameter(rLength);
        parms[1] = new ProgramParameter(new AS400Bin4().toBytes(rLength));
        parms[2] = new ProgramParameter(new AS400Text(8, ccsid, localSys).toBytes("JOBI0600"));
        parms[3] = new ProgramParameter(new AS400Text(26,ccsid, localSys).toBytes("*"));
        parms[4] = new ProgramParameter(new AS400Text(16,ccsid, localSys).toBytes(""));
        parms[5] = new ProgramParameter(new AS400Bin4().toBytes(0));

        qusrjobi.setProgram(QSYSObjectPathName.toPath("QSYS", "QUSRJOBI", "PGM"), parms);
```

```
        AS400Text uidText = new AS400Text(10, ccsid, localSys);

    // Richiamare l'API QUSRJOBI
        qusrjobi.run();

        byte[] uidBytes = new byte[10];
        System.arraycopy((qusrjobi.getParameterList())[0].getOutputData(), 90, uidBytes, 0, 10);

        return ((String)(uidText.toObject(uidBytes))).trim();
    }


    catch (Exception e) {
        e.printStackTrace();
    }

    return "";
    }

} //end SampleThreadSubjectLogin class



/**
 * CallbackHandler viene passati ai servizi di sicurezza sottostanti
 * in modo che possano interagire con l'applicazione per richiamare
 * specifici dati di autenticazione, come i nomi utente
 * e le parole d'ordine oppure per visualizzare determinate
 * informazioni, come messaggi di errore e di avvertenza.
 *
 * I CallbackHandler vengono implementati in una modalità applicazione e
 * dipendente dalla piattaforma. L'implementazione decide come richiamare
 * e visualizzare le informazioni a seconda dei
 * Callback inoltrati.
 *
 * Questa classe fornisce un CallbackHandler di esempio per le applicazioni
 * in esecuzione all'interno di un ambiente i5/OS. Comunque, non si è voluto
 * soddisfare i requisiti delle applicazioni di produzione.
 * Come indicato, CallbackHandler viene infine considerato come
 * dipendente dall'applicazione, poiché le singole applicazioni hanno un
 * controllo di errori univoco, una gestione dati e i requisiti
 * dell'interfaccia utente.
 *
 * Vengono gestiti i seguenti callback:
 *
• *
• NameCallback *
• PasswordCallback *
• TextOutputCallback *
 *
 * Per semplicità, la richiesta viene gestita in modo interattivo tramite
 * l'immissione e l'emissione standard. Tuttavia, ciò è irrilevante quando
 * l'immissione standard viene fornita dalla console, questo approccio
 * consente di visualizzare le parole d'ordine così come vengono
 * immesse. Evitare ciò nelle applicazioni di
 * produzione.
 *
 * Questo CallbackHandler consente anche di acquisire un nome e una
 * parola d'ordine attraverso un meccanismo alternativo e di
 * impostare direttamente sull'handler di evitare l'interazione
 * utente sui rispettivi Callback.
 *
 */
class SampleCBHandler implements CallbackHandler {
    private String name_ = null;
```

```
    private String password_ = null;
/**
 * Creare un nuovo SampleCBHandler.
 *
 */
public SampleCBHandler() {
    this(null, null);
}
/**
 * Creare un nuovo SampleCBHandler.
 *
 * È possibile specificare facoltativamente un nome e una parola d'ordine
 * per evitare la necessità di richiedere le informazioni
 * sui rispettivi Callback.
 *
 * @param name
 *      Il valore predefinito di name callback. Un valore null
 *      indica che queste informazioni devono essere richieste
 *      all'utente. Un valore diverso da null non può avere lunghezza
 *      zero o superare i 10 caratteri.
 *
 * @param password
 *      Il valore predefinito di password callback. Un valore null
 *      indica che queste informazioni devono essere richieste
 *      all'utente. Un valore diverso da null non può avere lunghezza
 *      zero o superare i 10 caratteri.
 */
public SampleCBHandler(String name, String password) {
    if (name != null)
        if ((name.length()==0) || (name.length()>10))
            throw new IllegalArgumentException("name");
    name_ = name;

    if (password != null)
        if ((password.length()==0) || (password.length()>10))
            throw new IllegalArgumentException("password");
    password_ = password;
}
/**
 * Gestire un determinato name callback.
 *
 * Innanzitutto controllare se il nome è stato inoltrato al
 * programma di creazione. Se sì, assegnarlo al
 * callback e saltare la richiesta.
 *
 * Se un valore non è stato preimpostato, richiedere
 * il nome utilizzando l'immissione e l'emissione standard.
 *
 * @param c
 *      Il NameCallback.
 *
 * @exception java.io.IOException
 *      Se si verifica un errore di immissione o emissione.
 *
 */
private void handleNameCallback(NameCallback c) throws IOException {
    // Controllare il valore nella cache
    if (name_ != null) {
        c.setName(name_);
        return;
    }
    // Nessun valore preimpostato; tentativo di stdin/out
    c.setName(
        stdIOReadName(c.getPrompt(), 10));
}
/**
 * Gestire un determinato name callback.
```

```
 *
 * Innanzitutto controllare se una parola d'ordine è stata inoltrata al
 * programma di creazione. Se sì, assegnarla al
 * callback e saltare la richiesta.
 *
 * Se un valore non è stato preimpostato, richiedere
 * la parola d'ordine utilizzando l'immissione e l'emissione standard.
 *
 * @param c
 *      Il PasswordCallback.
 *
 * @exception java.io.IOException
 *      Se si verifica un errore di immissione o emissione.
 *
 */
private void handlePasswordCallback(PasswordCallback c) throws IOException {
    // Controllare il valore nella cache
    if (password_ != null) {
        c.setPassword(password_.toCharArray());
        return;
    }

    // Nessun valore preimpostato; tentativo di stdin/out
    // Nota - Non per uso di produzione.
    //        La parola d'ordine non viene celata dall'I/E console standard.
    if (c.isEchoOn())
        c.setPassword(
            stdIOReadName(c.getPrompt(), 10).toCharArray());
    else
    {

        // Nota - La parola d'ordine non viene celata dall'I/E console standard.
        c.setPassword(stdIOReadName(c.getPrompt(), 10).toCharArray());

    }
}
/**
 * Gestire un determinato callback di emissione testo.
 *
 * Se il testo è informativo o un'avvertenza, il testo
 * viene scritto nell'emissione standard. Se il
 * callback definisce un messaggio di errore, il testo
 * viene scritto nell'errore standard.
 *
 * @param c
 *      Il TextOutputCallback.
 *
 * @exception java.io.IOException
 *      Se si verifica un errore di immissione o emissione.
 *
 */
private void handleTextOutputCallback(TextOutputCallback c) throws IOException {
    if (c.getMessageType() == TextOutputCallback.ERROR)
        System.err.println(c.getMessage());
    else
        System.out.println(c.getMessage());
}
/**
 * Richiamare o visualizzare le informazioni richieste nei
 * Callback forniti.
 *
 * L'implementazione del metodo handle controlla la(e)
 * istanza(e) dell'oggetto(i) Callback
 * inoltrati per richiamare o visualizzare le informazioni
 * richieste.
 *
 * @param callbacks
```

```
 *      Una schiera di oggetti Callback forniti da un
 *      servizio di sicurezza sottostante che contiene le
 *      informazioni richieste da richiamare o visualizzare.
 *
 * @exception java.io.IOException
 *      Se si verifica un errore di immissione o emissione.
 *
 * @exception UnsupportedCallbackException
 *      Se l'implementazione di questo metodo non supporta uno o
 *      più Callback specificati nel parametro
 *      callbacks.
 *
 */
public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException
{
    for (int i=0; i<callbacks.length; i++) {
        Callback c = callbacks[i];

        if (c instanceof NameCallback)
            handleNameCallback((NameCallback)c);
        else if (c instanceof PasswordCallback)
            handlePasswordCallback((PasswordCallback)c);
        else if (c instanceof TextOutputCallback)
            handleTextOutputCallback((TextOutputCallback)c);
        else
            throw new UnsupportedCallbackException
                (callbacks[i]);
    }
}
/**
 * Visualizzare la stringa indicata usando l'emissione standard,
 * seguita da uno spazio per separare l'immissione
 * successiva.
 *
 * @param prompt
 *      Il testo da visualizzare.
 *
 * @exception IOException
 *      Se si verifica un errore di immissione o emissione.
 *
 */
private void stdIOPrompt(String prompt) throws IOException {
    System.out.print(prompt + ' ');
    System.out.flush();
}
/**
 * Leggere una Stringa dall'immissione standard, arrestata da
 * maxLength o da un riporto a capo.
 *
 * @param prompt
 *      Il testo da visualizzare nell'emissione standard immediatamente
 *      prima di leggere il valore richiesto.
 *
 * @param maxLength
 *      Lunghezza massima della stringa da restituire.
 *
 * @return
 *      La stringa immessa. Il valore restituito non contiene
 *      spazi iniziali o finali e viene convertito in
 *      caratteri maiuscoli.
 *
 * @exception IOException
 *      Se si verifica un errore di immissione o emissione.
 *
 */
private String stdIOReadName(String prompt, int maxLength) throws IOException {
```

```
    stdIOPrompt(prompt);
    String s =
        (new BufferedReader
            (new InputStreamReader(System.in))).readLine().trim();
    if (s.length() < maxLength)
        s = s.substring(0,maxLength);
    return s.toUpperCase();
}

}//end SampleCBHandler class
```

Esempio: programma client IBM JGSS non JAAS

Utilizzare questo client di esempio JGSS insieme al server di esempio JGSS.

Per ulteriori informazioni sull'utilizzo del programma client di esempio, consultare "Esempi: scaricamento ed esecuzione di programmi JGSS di esempio" a pagina 408.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Programma client di esempio IBM JGSS 1.0

package com.ibm.security.jgss.test;
import org.ietf.jgss.*;
import com.ibm.security.jgss.Debug;

import java.io.*;
import java.net.*;
import java.util.*;

/**
 * Un client di esempio JGSS;
 * da utilizzare insieme con il server di esempio JGSS.
 * Il client stabilisce per primo un contesto con il server,
 * successivamente invia un messaggio codificato seguito da un MIC al server.
 * Il MIC viene calcolato sul testo chiaro che prima era codificato.
 * Il client richiede al server l'autenticazione
 * (autenticazione reciproca) mentre si stabilisce il contesto.
 * Delega, inoltre, le proprie credenziali al server.
 *
 * Imposta la variabile JAVA
 * javax.security.auth.useSubjectCredsOnly su false
 * in modo che JGSS non acquisirà le credenziali tramite JAAS.
 *
 * Il client rileva i parametri di immissione e li completa
 * con le informazioni provenienti dal file jgss.ini; qualsiasi immissione necessaria, ma non
 * fornita sulla riga comandi, viene prelevata dal file jgss.ini.
 *
 * Utilizzo: Client [opzioni]
 *
 * L'opzione -? fornisce un messaggio di aiuto che indica le opzioni supportate.
 *
 * Questo client di esempio non utilizza JAAS.
 * Il client può essere in esecuzione sul server e sul client di esempio JAAS.
 * Consultare {@link JAASClient JAASClient} per un client di esempio che utilizza JAAS.
 */

class Client
{
    private Util testUtil      = null;
    private String myName      = null;
    private GSSName gssName    = null;
    private String serverName = null;
    private int servicePort    = 0;
    private GSSManager mgr     = GSSManager.getInstance();
```

```
private GSSName service    = null;
private GSSContext context        = null;
private String program     = "Client";
private String debugPrefix = "Client: ";
private TCPComms tcp       = null;
private String data               = null;
private byte[] dataBytes   = null;
private String serviceHostname= null;
private GSSCredential gssCred = null;

private static Debug debug        = new Debug();

private static final String usageString =
      "\t[-?] [-d | -n nome] [-s NomeServer]"
    + "\n\t[-h HostServer [:port]] [-p porta] [-m msg]"
    + "\n"
    + "\n  -?\t\t\thelp; genera questo messaggio"
    + "\n  -n nome\t\til nome principal del client (senza dominio)"
    + "\n  -s NomeServer\t\til nome principal del server (senza dominio)"
    + "\n  -h HostServer[:port]\til nome host del server"
    +         " (e numero porta facoltativa)"
    + "\n  -p porta\t\tla porta su cui il server sarà in ascolto"
    + "\n  -m msg\t\til messaggio da inviare al server secondario";


// Il chiamante deve chiamare initialize (può essere necessario chiamare prima processArgs).
public Client (String programName) throws Exception
{
    testUtil = new Util();
    if (programName != null)
    {
        program = programName;
        debugPrefix = programName + ": ";
    }
}

// Il chiamante deve chiamare initialize (può essere necessario chiamare prima processArgs).
Client (String programName, boolean useSubjectCredsOnly) throws Exception
{
    this(programName);
    setUseSubjectCredsOnly(useSubjectCredsOnly);
}

public Client(GSSCredential myCred,
              String serverNameWithoutRealm,
              String serverHostname,
              int    serverPort,
              String message)
    throws Exception
{
    testUtil = new Util();

    if (myCred != null)
    {
        gssCred = myCred;
    }
    else
    {
        throw new GSSException(GSSException.NO_CRED, 0,
                               "Null input credential");
    }

    init(serverNameWithoutRealm, serverHostname, serverPort, message);
}

void setUseSubjectCredsOnly(boolean useSubjectCredsOnly)
{
```

```
    final String subjectOnly = useSubjectCredsOnly ? "true" : "false";
    final String property = "javax.security.auth.useSubjectCredsOnly";

    String temp = (String)java.security.AccessController.doPrivileged(
                    new sun.security.action.GetPropertyAction(property));

    if (temp == null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
          + "setting useSubjectCredsOnly property to "
          + useSubjectCredsOnly);

        // Proprietà non impostata. Impostarla sul valore specificato.

        java.security.AccessController.doPrivileged(
             new java.security.PrivilegedAction() {
               public Object run() {
                  System.setProperty(property, subjectOnly);
                  return null;
               }
             });
    }
    else
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
          + "useSubjectCredsOnly property already set "
          + "in JVM  to " + temp);
    }
}

private void init(String myNameWithoutRealm,
                  String serverNameWithoutRealm,
                  String serverHostname,
                  int    serverPort,
                  String message) throws Exception
{
    myName = myNameWithoutRealm;
    init(serverNameWithoutRealm, serverHostname, serverPort, message);
}

private void init(String serverNameWithoutRealm,
                  String serverHostname,
                  int    serverPort,
                  String message) throws Exception
{
    // nome del peer
    if (serverNameWithoutRealm != null)
    {
        this.serverName = serverNameWithoutRealm;
    }
    else
    {
        this.serverName = testUtil.getDefaultServicePrincipalWithoutRealm();
    }

    // host del peer
    if (serverHostname != null)
    {
        this.serviceHostname = serverHostname;
    }
    else
    {
        this.serviceHostname = testUtil.getDefaultServiceHostname();
    }

    // porta del peer
    if (serverPort > 0)
```

```
    {
        this.servicePort = serverPort;
    }
    else
    {
        this.servicePort = testUtil.getDefaultServicePort();
    }

    // messaggio per il peer
    if (message != null)
    {
        this.data = message;
    }
    else
    {
        this.data = "The quick brown fox jumps over the lazy dog";
    }

    this.dataBytes = this.data.getBytes();

    tcp = new TCPComms(serviceHostname, servicePort);
}


void initialize() throws Exception
{
     Oid krb5MechanismOid = new Oid("1.2.840.113554.1.2.2");

    if (gssCred == null)
    {
        if (myName != null)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                              + "creating GSSName USER_NAME for "
                              + myName);

            gssName = mgr.createName(
                           myName,
                           GSSName.NT_USER_NAME,
                            krb5MechanismOid);

            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                              + "Canonicalized GSSName=" + gssName);
        }
        else
            gssName = null; // per credenziali predefinite


        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "creating"
           + ((gssName == null)? " default " : " ")
           + "credential");

        gssCred = mgr.createCredential(
                               gssName,
                               GSSCredential.DEFAULT_LIFETIME,
                               (Oid)null,
                               GSSCredential.INITIATE_ONLY);
        if (gssName == null)
        {
            gssName = gssCred.getName();

            myName  = gssName.toString();

            debug.out(Debug.OPTS_CAT_APPLICATION,
                debugPrefix + "default credential principal=" + myName);
        }
    }
```

```
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + gssCred);

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
       + "creating canonicalized GSSName for serverName " + serverName);

    service = mgr.createName(serverName,
                             GSSName.NT_HOSTBASED_SERVICE,
                              krb5MechanismOid);

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
       + "Canonicalized server name = " + service);

    debug.out(Debug.OPTS_CAT_APPLICATION,
                         debugPrefix + "Raw data=" + data);
}


void establishContext(BitSet flags) throws Exception
{
    try {

      debug.out(Debug.OPTS_CAT_APPLICATION,
                         debugPrefix + "creating GSScontext");

      Oid defaultMech = null;
      context = mgr.createContext(service, defaultMech, gssCred,
                                  GSSContext.INDEFINITE_LIFETIME);

      if (flags != null)
      {
          if (flags.get(Util.CONTEXT_OPTS_MUTUAL))
          {
              debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                   + "requesting mutualAuthn");

              context.requestMutualAuth(true);
          }

          if (flags.get(Util.CONTEXT_OPTS_INTEG))
          {
              debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                   + "requesting integrity");

              context.requestInteg(true);
          }

          if (flags.get(Util.CONTEXT_OPTS_CONF))
          {
              context.requestConf(true);
              debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                      + "requesting confidentiality");
          }

          if (flags.get(Util.CONTEXT_OPTS_DELEG))
          {
              context.requestCredDeleg(true);
              debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                       + "requesting delegation");
          }

          if (flags.get(Util.CONTEXT_OPTS_REPLAY))
          {
              context.requestReplayDet(true);
              debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                  + "requesting replay detection");
          }
```

```
            if (flags.get(Util.CONTEXT_OPTS_SEQ))
            {
                context.requestSequenceDet(true);
                debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                          + "requesting out-of-sequence detection");
            }
            // Aggiungere altro in seguito!
        }

        byte[] response = null;
        byte[] request = null;
        int len = 0;
        boolean done = false;
        do {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "Calling initSecContext");

            request = context.initSecContext(response, 0, len);

            if (request != null)
            {
                debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                  + "Sending initial context token");

                tcp.send(request);
            }
            done = context.isEstablished();

            if (!done)
            {
                debug.out(Debug.OPTS_CAT_APPLICATION,
                    debugPrefix + "Receiving response token");

                byte[] temp = tcp.receive();
                response = temp;
                len = response.length;
            }
        } while(!done);

        debug.out(Debug.OPTS_CAT_APPLICATION,
              debugPrefix + "context established with acceptor");

    } catch (Exception exc) {
        exc.printStackTrace();
        throw exc;
    }
}

void doMIC() throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "generating MIC");
    byte[] mic = context.getMIC(dataBytes, 0, dataBytes.length, null);

    if (mic != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "sending MIC");
        tcp.send(mic);
    }
    else
        debug.out(Debug.OPTS_CAT_APPLICATION,
                          debugPrefix + "getMIC Failed");
}

void doWrap() throws Exception
{
    MessageProp mp = new MessageProp(true);
```

```
        mp.setPrivacy(context.getConfState());

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "wrapping message");

        byte[] wrapped = context.wrap(dataBytes, 0, dataBytes.length, mp);

        if (wrapped != null)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION,
                    debugPrefix + "sending wrapped message");

            tcp.send(wrapped);
        }
        else
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "wrap Failed");
    }

    void printUsage()
    {
        System.out.println(program + usageString);
    }

    void processArgs(String[] args) throws Exception
    {
        String port            = null;
        String myName          = null;
        int servicePort               = 0;
        String serviceHostname = null;

        String sHost = null;
        String msg = null;

        GetOptions options = new GetOptions(args, "?h:p:m:n:s:");
        int ch = -1;
        while ((ch = options.getopt()) != options.optEOF)
        {
            switch(ch)
            {
                case '?':
                    printUsage();
                    System.exit(1);

                case 'h':
                    if (sHost == null)
                    {
                        sHost = options.optArgGet();
                        int p = sHost.indexOf(':');
                        if (p != -1)
                        {
                            String temp1 = sHost.substring(0, p);
                            if (port == null)
                                port = sHost.substring(p+1, sHost.length()).trim();
                            sHost = temp1;
                        }
                    }
                    continue;

                case 'p':
                    if (port == null)
                        port = options.optArgGet();
                    continue;

                case 'm':
                    if (msg == null)
                        msg = options.optArgGet();
                    continue;
```

```
            case 'n':
                if (myName == null)
                    myName = options.optArgGet();
                continue;

            case 's':
                if (serverName == null)
                    serverName = options.optArgGet();
                continue;
        }
    }

    if ((port != null) && (port.length() > 0))
    {
        int p = -1;
        try {
            p = Integer.parseInt(port);
        } catch (Exception exc) {
            System.out.println("Bad port input: "+port);
        }

        if (p != -1)
            servicePort = p;
    }

    if ((sHost != null) && (sHost.length() > 0)) {
            serviceHostname = sHost;
    }

    init(myName, serverName, serviceHostname, servicePort, msg);
}

void interactWithAcceptor(BitSet flags) throws Exception
{
    establishContext(flags);
    doWrap();
    doMIC();
}

void interactWithAcceptor() throws Exception
{
    BitSet flags = new BitSet();
    flags.set(Util.CONTEXT_OPTS_MUTUAL);
    flags.set(Util.CONTEXT_OPTS_CONF);
    flags.set(Util.CONTEXT_OPTS_INTEG);
    flags.set(Util.CONTEXT_OPTS_DELEG);
    interactWithAcceptor(flags);
}

void dispose() throws Exception
{
    if (tcp != null)
    {
        tcp.close();
    }
}

public static void main(String args[]) throws Exception
{
    System.out.println(debug.toString()); // XXXXXXX
    String programName = "Client";
    Client client = null;
    try {
        client = new Client(programName,
                            false); // non utilizzare le credenziali di Subject.
        client.processArgs(args);
        client.initialize();
```

```
            client.interactWithAcceptor();
        } catch (Exception exc) {
            debug.out(Debug.OPTS_CAT_APPLICATION,
                            programName + " Exception: " + exc.toString());
            exc.printStackTrace();
            throw exc;
        } finally {
            try {
                if (client != null)
                    client.dispose();
            } catch (Exception exc) {}
        }

        debug.out(Debug.OPTS_CAT_APPLICATION, programName + ": done");
    }
}
```

Esempio: programma server IBM JGSS non JAAS

Quest'esempio contiene un server di esempio JGSS utilizzato insieme ad un client di esempio JGSS.

Per ulteriori informazioni sull'utilizzo del programma server di esempio, consultare "Esempi: scaricamento ed esecuzione di programmi JGSS di esempio" a pagina 408.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Programma server di esempio IBM JGSS 1.0

package com.ibm.security.jgss.test;

import org.ietf.jgss.*;
import com.ibm.security.jgss.Debug;
import java.io.*;
import java.net.*;
import java.util.*;

/**
 * Un server di esempio JGSS; da utilizzare insieme con un client di esempio JGSS.
 *
 * È sempre in ascolto delle connessioni client
 * e genera un sottoprocesso per servire una connessione in entrata.
 * È in grado di eseguire più sottoprocessi contemporaneamente.
 * In altre parole, può servire più client allo stesso tempo.
 *
 * Ogni sottoprocesso stabilisce prima un contesto con il client,
 * poi attende un messaggio codificato seguito da un MIC.
 * Presume che il client abbia calcolato il MIC in base al testo chiaro
 * codificato dal client.
 *
 * Se il client delega le proprie credenziali al server, le credenziali
 * delegate vengono utilizzate per comunicare con un server secondario.
 *
 * È possibile, inoltre, avviare il server in modo che agisca come un client e come
 * un server (utilizzando l'opzione -b). In tal caso, il primo
 * sottoprocesso generato dal server utilizza le credenziali proprie del principal del server
 * per comunicare con il server secondario.
 *
 * Occorre che il server secondario sia stato avviato prima del server (primario)
 * che ha avviato i contatti con esso (il server secondario).
 * Durante le comunicazioni con il server secondario, il server primario agisce come
 * un iniziatore JGSS (cioè, il client), stabilendo un contesto e avviando lo
 * scambio di informazioni codificate e del MIC, tramite messaggio, con il server secondario.
 *
 * Il server rileva i parametri di immissione e li completa
 * con le informazioni provenienti dal file jgss.ini; qualsiasi immissione necessaria, ma non
```

```
 * fornita sulla riga comandi, viene prelevata dal file jgss.ini.
 * Vengono utilizzati i valori predefiniti incorporati, nel caso in cui non esista alcun file jgss.ini
 * o non sia stata specificata una variabile particolare nel file ini.
 *
 * Utilizzo: Server [opzioni]
 *
 * L'opzione -? fornisce un messaggio di aiuto che indica le opzioni supportate.
 *
 * Questo server di esempio non utilizza JAAS.
 * Imposta la variabile JAVA
 * javax.security.auth.useSubjectCredsOnly su false
 * in modo che JGSS non acquisirà le credenziali tramite JAAS.
 * Il server può essere in esecuzione sui server e sui client di esempio JAAS.
 * Consultare {@link JAASServer JAASServer} per un server di esempio che utilizza JAAS.
 */

class Server implements Runnable
{
    /*
     * NOTE:
     * Questa classe, Server, può essere in esecuzione contemporaneamente in
     * più sottoprocessi. Le variabili statiche sono costituite da variabili
     * impostate dagli argomenti della riga comandi e da variabili (come
     * le credenziali proprie del server, gssCred) impostate una sola volta
     * durante l'inizializzazione. Tali variabili non cambiano,
     * una volta impostate, e vengono condivise da tutti i sottoprocessi in esecuzione.
     *
     * L'unica variabile statica che viene modificata dopo l'impostazione iniziale
     * è 'beenInitiator', che è impostata su 'true'
     * dal primo sottoprocesso in modo che il server possa essere in esecuzione come iniziatore usando
     * le credenziali proprie del server. Ciò assicura che il server sia in esecuzione come iniziatore
     * solo una volta. La query e la modifica di 'beenInitiator' sono sincronizzate
     * tra i sottoprocessi.
     *
     * La variabile 'tcp' non è statica ed è impostata per sottoprocesso
     * in modo da rappresentare il socket al quale si è connesso il client servito
     * dal sottoprocesso.
     */

    private static Util testUtil             = null;
    private static int myPort                = 0;
    private static Debug debug               = new Debug();
    private static String myName             = null;
    private static GSSCredential gssCred     = null;
    private static String serviceNameNoRealm = null;
    private static String serviceHost        = null;
    private static int    servicePort        = 0;
    private static String serviceMsg         = null;
    private static GSSManager mgr            = null;
    private static GSSName gssName           = null;
    private static String program            = "Server";
    private static boolean clientServer      = false;
    private static boolean primaryServer     = true;

    private static boolean beenInitiator     = false;

    private static final String usageString =
         "\t[-?] [-# numero] [-d | -n nome] [-p porta]"
       + "\n\t[-s NomeServer] [-h HostServer [:port]] [-P PortaServer] [- msg]"
       + "\n"
       + "\n  -?\t\t\thelp; genera questo messaggio"
       + "\n  -# numero\t\til server primario o secondario"
       +         " \n\t\t\t(1 = primario, 2 = secondario; default = primo)"
       + "\n  -n nome\t\til nome principal del server (senza dominio)"
       + "\n  -p porta\t\tla porta su cui il server sarà in ascolto"
       + "\n  -s NomeServer\t\til nome principal del server secondario"
       +         " (senza dominio)"
```

```
   + "\n  -h HostServer[:port]\til nome host del server secondario"
   +         " (e numero porta facoltativa)"
   + "\n  -P porta\t\til numero di porta del server secondario"
   + "\n  -m msg\t\til messaggio da inviare al server secondario"
   + "\n  -b \t\tin esecuzione sia come client sia come server"
   +         " utilizzando le credenziali proprie del server";

// Le variabili non statiche sono specifiche del sottoprocesso
// poiché ogni sottoprocesso esegue un'istanza separata di questa classe.

private String debugPrefix = null;
private TCPComms tcp       = null;

static {
    try {
        testUtil = new Util();
    } catch (Exception exc) {
        exc.printStackTrace();
        System.exit(1);
    }
}

Server (Socket socket) throws Exception
{
    debugPrefix = program + ": ";
    tcp = new TCPComms(socket);
}

Server (String program) throws Exception
{
    debugPrefix = program + ": ";
    this.program = program;
}

Server (String program, boolean useSubjectCredsOnly) throws Exception
{
    this(program);
    setUseSubjectCredsOnly(useSubjectCredsOnly);
}

void setUseSubjectCredsOnly(boolean useSubjectCredsOnly)
{
    final String subjectOnly = useSubjectCredsOnly ? "true" : "false";
    final String property = "javax.security.auth.useSubjectCredsOnly";

    String temp = (String)java.security.AccessController.doPrivileged(
                    new sun.security.action.GetPropertyAction(property));

    if (temp == null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
          + "setting useSubjectCredsOnly property to "
          + (useSubjectCredsOnly ? "true" : "false"));

        // Proprietà non impostata. Impostarla sul valore specificato.

        java.security.AccessController.doPrivileged(
             new java.security.PrivilegedAction() {
               public Object run() {
                  System.setProperty(property, subjectOnly);
                  return null;
               }
             });
    }
    else
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
```

```
          + "useSubjectCredsOnly property already set "
          + "in JVM  to " + temp);
    }
}

private void init(boolean  primary,
          String   myNameWithoutRealm,
          int      port,
          String serverNameWithoutRealm,
          String serverHostname,
          int    serverPort,
          String   message,
          boolean  clientServer)
    throws Exception
{
    primaryServer = primary;
    this.clientServer = clientServer;

    myName = myNameWithoutRealm;

    // la porta
    if (port > 0)
    {
        myPort = port;
    }
    else if (primary)
    {
        myPort = testUtil.getDefaultServicePort();
    }
    else
    {
        myPort = testUtil.getDefaultService2Port();
    }

    if (primary)
    {
        ///// nome del peer
        if (serverNameWithoutRealm != null)
        {
            serviceNameNoRealm = serverNameWithoutRealm;
        }
        else
        {
            serviceNameNoRealm =
                  testUtil.getDefaultService2PrincipalWithoutRealm();
        }

        // host del peer
        if (serverHostname != null)
        {
            if (serverHostname.equalsIgnoreCase("localHost"))
            {
                serverHostname = InetAddress.getLocalHost().getHostName();
            }

            serviceHost = serverHostname;
        }
        else
        {
            serviceHost = testUtil.getDefaultService2Hostname();
        }

        // porta del peer
        if (serverPort > 0)
        {
            servicePort = serverPort;
        }
```

```
        else
        {
            servicePort = testUtil.getDefaultService2Port();
        }

        // messaggio per il peer
        if (message != null)
        {
            serviceMsg = message;
        }
        else
        {
            serviceMsg = "Hi there! I am a server."
                         + "But I can be a client, too";
        }
    }

    String temp = debugPrefix + "details"
                  + "\n\tPrimary:\t" + primary
                  + "\n\tName:\t\t" + myName
                  + "\n\tPort:\t\t" + myPort
                  + "\n\tClient+server:\t" + clientServer;
    if (primary)
    {
        temp += "\n\tOther Server:"
                  + "\n\t\tName:\t" + serviceNameNoRealm
                  + "\n\t\tHost:\t" + serviceHost
                  + "\n\t\tPort:\t" + servicePort
                  + "\n\t\tMsg:\t" + serviceMsg;
    }

    debug.out(Debug.OPTS_CAT_APPLICATION, temp);
}


void initialize() throws GSSException
{
   debug.out(Debug.OPTS_CAT_APPLICATION,
                     debugPrefix + "creating GSSManager");

    mgr = GSSManager.getInstance();

    int usage = clientServer ? GSSCredential.INITIATE_AND_ACCEPT
                             : GSSCredential.ACCEPT_ONLY;

    if (myName != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                           + "creating GSSName for " + myName);

        gssName = mgr.createName(myName,
                                 GSSName.NT_HOSTBASED_SERVICE);

        Oid krb5MechanismOid = new Oid("1.2.840.113554.1.2.2");
        gssName.canonicalize(krb5MechanismOid);

        debug.out(Debug.OPTS_CAT_APPLICATION,
              debugPrefix + "Canonicalized GSSName=" + gssName);
    }
    else
        gssName = null;

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "creating"
           + ((gssName == null)? " default " : " ")
           + "credential");

    gssCred = mgr.createCredential(
```

```
                        gssName, GSSCredential.DEFAULT_LIFETIME,
                        (Oid)null, usage);
    if (gssName == null)
    {
        gssName = gssCred.getName();
        myName  = gssName.toString();

        debug.out(Debug.OPTS_CAT_APPLICATION,
           debugPrefix + "default credential principal=" + myName);
    }
}



void processArgs(String[] args) throws Exception
{
    String port            = null;
    String name    = null;
    int iport      = 0;

    String sport   = null;
    int isport     = 0;
    String sname   = null;
    String shost   = null;
    String smessage        = null;

    boolean primary = true;
    String status  = null;

    boolean defaultPrinc = false;
    boolean clientServer = false;

    GetOptions options = new GetOptions(args, "?#:p:n:P:s:h:m:b");
    int ch = -1;
    while ((ch = options.getopt()) != options.optEOF)
    {
        switch(ch)
        {
            case '?':
                printUsage();
                System.exit(1);

            case '#':
                if (status == null)
                    status = options.optArgGet();
                continue;

            case 'p':
                if (port == null)
                    port = options.optArgGet();
                continue;

            case 'n':
                if (name == null)
                    name = options.optArgGet();
                continue;

            case 'b':
                clientServer = true;
                continue;

            ////// L'altro server

            case 'P':
                if (sport == null)
                    sport = options.optArgGet();
                continue;
```

```
        case 'm':
            if (smessage == null)
                smessage = options.optArgGet();
            continue;

        case 's':
            if (sname == null)
                sname = options.optArgGet();
            continue;

        case 'h':
            if (shost == null)
            {
                shost = options.optArgGet();
                int p = shost.indexOf(':');
                if (p != -1)
                {
                    String temp1 = shost.substring(0, p);
                    if (sport == null)
                        sport = shost.substring
                                    (p+1, shost.length()).trim();
                    shost = temp1;
                }
            }
            continue;
    }
}

if (defaultPrinc && (name != null))
{
    System.out.println(
      "ERROR: '-d' and '-n ' options are mutually exclusive");
    printUsage();
    System.exit(1);
}

if (status != null)
{
    int p = -1;
    try {
        p = Integer.parseInt(status);
    } catch (Exception exc) {
        System.out.println( "Bad status input: "+status);
    }

    if (p != -1)
    {
        primary = (p == 1);
    }
}

if (port != null)
{
    int p = -1;
    try {
        p = Integer.parseInt(port);
    } catch (Exception exc) {
        System.out.println("Bad port input: "+port);
    }
    if (p != -1)
        iport = p;
}

if (sport != null)
{
    int p = -1;
```

```
        try {
            p = Integer.parseInt(sport);
        } catch (Exception exc) {
            System.out.println( "Bad server port input: "+port);
        }
        if (p != -1)
            isport = p;
    }

    init(primary,  // primo o secondo server
         name,     // nome
         iport,    // porta
         sname,    // nome dell'altro server
         shost,    // nome host dell'altro server
         isport,   // porta dell'altro server
         smessage, // messaggio per l'altro server
         clientServer); // se deve essere in esecuzione come
                                    // iniziatore con le proprie credenziali
}

void processRequests() throws Exception
{
    ServerSocket ssocket = null;
    Server server = null;
    try {
        ssocket = new ServerSocket(myPort);
        do {
           debug.out(Debug.OPTS_CAT_APPLICATION,
                 debugPrefix + "listening on port " + myPort + " ...");
            Socket csocket = ssocket.accept();

           debug.out(Debug.OPTS_CAT_APPLICATION,
               debugPrefix + "incoming connection on " + csocket);

            server = new Server(csocket); // impostare socket client per sottoprocesso
            Thread thread = new Thread(server);
           thread.start();
            if (!thread.isAlive())
                server.dispose(); // chiudere il socket client
       } while(true);
    } catch (Exception exc) {
       debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "*** ERROR processing requests ***");
        exc.printStackTrace();
    } finally {
        try {
            if (ssocket != null)
               ssocket.close(); // chiudere il socket server
            if (server != null)
                server.dispose(); // chiudere il socket client
        } catch (Exception exc) {}
    }
}

void dispose()
{
    try {
        if (tcp != null)
        {
            tcp.close();
            tcp = null;
        }
    } catch (Exception exc) {}
}

boolean establishContext(GSSContext context) throws Exception
{
```

```
    byte[] response = null;
    byte[] request = null;

    debug.out(Debug.OPTS_CAT_APPLICATION,
                      debugPrefix + "establishing context");

    do {
        request = tcp.receive();
        if (request == null || request.length == 0)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "Received no data; perhaps client disconnected");

            return false;
        }

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "accepting");
        if ((response = context.acceptSecContext
                            (request, 0, request.length)) != null)
        {
             debug.out(Debug.OPTS_CAT_APPLICATION,
                       debugPrefix + "sending response");
             tcp.send(response);
        }
    } while(!context.isEstablished());

    debug.out(Debug.OPTS_CAT_APPLICATION,
                debugPrefix + "context established - " + context);

    return true;
}

byte[] unwrap(GSSContext context, byte[] msg) throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "unwrapping");

    MessageProp mp = new MessageProp(true);
    byte[] unwrappedMsg = context.unwrap(msg, 0, msg.length, mp);

    debug.out(Debug.OPTS_CAT_APPLICATION,
                         debugPrefix + "unwrapped msg is:");
    debug.out(Debug.OPTS_CAT_APPLICATION, unwrappedMsg);

    return unwrappedMsg;
}

void verifyMIC (GSSContext context, byte[] mic, byte[] raw) throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "verifying MIC");

    MessageProp mp = new MessageProp(true);
    context.verifyMIC(mic, 0, mic.length, raw, 0, raw.length, mp);

    debug.out(Debug.OPTS_CAT_APPLICATION,
                   debugPrefix + "successfully verified MIC");
}

void useDelegatedCred(GSSContext context) throws Exception
{
    GSSCredential delCred = context.getDelegCred();
    if (delCred != null)
    {
        if (primaryServer)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                    "Primary server received delegated cred; using it");
            runAsInitiator(delCred); // utilizzando credenziali delegate
```

```
        }
        else
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                   "Non-primary server received delegated cred; "
                        + "ignoring it");

        }
    }
    else
    {
         debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                                    "ERROR: null delegated cred");
    }
}

public void run()
{
    byte[] response = null;
    byte[] request = null;
    boolean unwrapped      = false;
    GSSContext context     = null;

    try {
      Thread currentThread = Thread.currentThread();
      String threadName    = currentThread.getName();

      debugPrefix = program + " " + threadName + ": ";

      debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                             + "servicing client ...");

     debug.out(Debug.OPTS_CAT_APPLICATION,
          debugPrefix + "creating GSSContext");

      context = mgr.createContext(gssCred);

      // Stabilire prima il contesto con l'iniziatore.
      if (!establishContext(context))
          return;

      // Elaborare, quindi, i messaggi provenienti dall'iniziatore.
      // Si prevede di ricevere un messaggio codificato seguito da un MIC.
      // Il MIC deve essere stato calcolato sul testo chiaro
      // ricevuto codificato.
      // Utilizzare le credenziali delegate, se presenti.
      // Quindi, eseguire come iniziatore che usa credenziali proprie, se necessario; solo
      // il primo sottoprocesso esegue ciò.

      do {
         debug.out(Debug.OPTS_CAT_APPLICATION,
              debugPrefix + "receiving per-message request");

          request = tcp.receive();
          if (request == null || request.length == 0)
          {
             debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                 + "Received no data; perhaps client disconnected");

             return;
          }

          // Previsto prima messaggio codificato.
          if (!unwrapped)
          {
              response = unwrap(context, request);
              unwrapped = true;
```

```
            continue; // richiesta successiva
          }

          // Seguito da un MIC.
          verifyMIC(context, request, response);

          // Impersonare l'iniziatore se sono state delegate le sue credenziali.
          if (context.getCredDelegState())
              useDelegatedCred(context);

          debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
              + "clientServer=" + clientServer
              + ", beenInitiator=" + beenInitiator);

          // Se necessario, eseguire come iniziatore utilizzando le credenziali proprie.
          if (clientServer)
              runAsInitiatorOnce(currentThread);

          debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "done");
          return;

      } while(true);

    } catch (Exception exc) {
       debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "ERROR");
        exc.printStackTrace();

        // Eliminare gli errori per sottoprocesso per evitare
        // l'inattività del server a causa di errori in
        // sottoprocessi singoli.
        return;
    } finally {
        if (context != null)
        {
            try {
                context.dispose();
            } catch (Exception exc) {}
        }
    }
}

synchronized void runAsInitiatorOnce(Thread thread)
    throws InterruptedException
{
    if (!beenInitiator)
    {
        // impostare indicatore true al più presto per evitare che successivi sottoprocessi
        // tentino un runAsInitiator.
        beenInitiator = true;

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                "About to run as initiator with own creds ...");

        //thread.sleep(30*1000, 0);
        runAsInitiator();
    }
}


void runAsInitiator(GSSCredential cred)
{
    Client client = null;
    try {
        client = new Client(cred,
                            serviceNameNoRealm,
                            serviceHost,
                            servicePort,
```

```
                                serviceMsg);

            client.initialize();

            BitSet flags = new BitSet();
            flags.set(Util.CONTEXT_OPTS_MUTUAL);
            flags.set(Util.CONTEXT_OPTS_CONF);
            flags.set(Util.CONTEXT_OPTS_INTEG);

            client.interactWithAcceptor(flags);

        } catch (Exception exc) {
               debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                  + "Exception running as initiator");

            exc.printStackTrace();
        } finally {
            try {
                client.dispose();
            } catch (Exception exc) {}
        }
    }

    void runAsInitiator()
    {
        if (clientServer)
        {
           debug.out(Debug.OPTS_CAT_APPLICATION,
               debugPrefix + "running as initiator with own creds");

            runAsInitiator(gssCred); // utilizzare credenziali proprie;
        }
        else
        {
           debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
               + "Cannot run as initiator with own creds "
               + "\nbecause not running as both initiator and acceptor.");
        }
    }

    void printUsage()
    {
        System.out.println(program + usageString);
    }

    public static void main(String[] args) throws Exception
    {
        System.out.println(debug.toString()); // XXXXXXX
        String programName = "Server";
        try {
            Server server = new Server(programName,
                                       false); // non utilizzare credenziali da Subject
            server.processArgs(args);
            server.initialize();
            server.processRequests();
        } catch (Exception exc) {
            debug.out(Debug.OPTS_CAT_APPLICATION, programName + ": EXCEPTION");
            exc.printStackTrace();
            throw exc;
        }
    }
}
```

Esempio: programma client IBM JGSS abilitato a JAAS

Questo programma di esempio effettua un collegamento a JAAS e opera all'interno del contesto di collegamento a JAAS. Non imposta la variabile javax.security.auth.useSubjectCredsOnly, lasciando che la

variabile assuma il valore predefinito di ″true″ in modo che GSS Java acquisisca le credenziali dal Soggetto JAAS associato al contesto di accesso creato dal client.

Per ulteriori informazioni sull'utilizzo del programma client di esempio, consultare "Esempi: scaricamento ed esecuzione di programmi JGSS di esempio" a pagina 408.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Programma client IBM Java GSS 1.0 abilitato a JAAS di esempio

package com.ibm.security.jgss.test;
import com.ibm.security.jgss.Debug;
import com.ibm.security.auth.callback.Krb5CallbackHandler;
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.security.PrivilegedExceptionAction;

/**
 * Un client di esempio Java GSS che utilizza JAAS.
 *
 * Effettua un collegamento a JAAS ed opera all'interno del contesto di collegamento a JAAS così creato.
 *
 * Non imposta la variabile JAVA
 * javax.security.auth.useSubjectCredsOnly, lasciando
 * la variabile impostata sul valore predefinito true,
 * in questo modo Java GSS acquisisce le credenziali dal Subject JAAS
 * associato al contesto di collegamento (creato dal client).
 *
 * JAASClient equivale alla relativa superclasse {@link Client Client}
 * sotto tutti gli altri aspetti e
 * può essere eseguito sui client e i server di esempio non JAAS.
 */

class JAASClient extends Client
{
    JAASClient(String programName) throws Exception
    {
        // Non impostare useSubjectCredsOnly. Impostare solo il nome del programma.
        // Se non impostato, useSubjectCredsOnly viene impostato sul valore predefinito "true".
        super(programName);
    }

    static class JAASClientAction implements PrivilegedExceptionAction
    {
         private JAASClient client;

         public JAASClientAction(JAASClient client)
         {
             this.client = client;
         }

         public Object run () throws Exception
         {
             client.initialize();
             client.interactWithAcceptor();
             return null;
         }
    }

    public static void main(String args[]) throws Exception
    {
        String programName = "JAASClient";
        JAASClient client = null;
        Debug dbg = new Debug();
```

```
        System.out.println(dbg.toString()); // XXXXXXX

        try {
            client = new JAASClient(programName);//utilizzare credenziali Subject
            client.processArgs(args);

            LoginContext loginCtxt = new LoginContext("JAASClient",
                                       new Krb5CallbackHandler());

            loginCtxt.login();

            dbg.out(Debug.OPTS_CAT_APPLICATION,
                       programName + ": Kerberos login OK");

            Subject subject = loginCtxt.getSubject();

            PrivilegedExceptionAction jaasClientAction
                           = new JAASClientAction(client);

            Subject.doAsPrivileged(subject, jaasClientAction, null);

        } catch (Exception exc) {
            dbg.out(Debug.OPTS_CAT_APPLICATION,
                        programName + " Exception: " + exc.toString());
            exc.printStackTrace();
            throw exc;
        } finally {
            try {
                if (client != null)
                    client.dispose();
            } catch (Exception exc) {}
        }

        dbg.out(Debug.OPTS_CAT_APPLICATION,
                       programName + ": Done ...");
    }
}
```

Esempio: programma server IBM JGSS abilitato

Questo programma di esempio effettua un collegamento a JAAS e opera all'interno del contesto di collegamento a JAAS.

Per ulteriori informazioni sull'utilizzo del programma server di esempio, consultare "Esempi: scaricamento ed esecuzione di programmi JGSS di esempio" a pagina 408.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
// Programma server IBM Java GSS 1.0 abilitato a JAAS di esempio

package com.ibm.security.jgss.test;
import com.ibm.security.jgss.Debug;
import com.ibm.security.auth.callback.Krb5CallbackHandler;
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.security.PrivilegedExceptionAction;

/**
 * Un server di esempio Java GSS che utilizza JAAS.
 *
 * Effettua un collegamento a JAAS ed opera all'interno del contesto di collegamento a JAAS così creato.
 *
 * Non imposta la variabile JAVA
 * javax.security.auth.useSubjectCredsOnly, lasciando
 * la variabile impostata sul valore predefinito true,
```

```
 * in questo modo Java GSS acquisisce le credenziali dal Subject JAAS
 * associato al contesto di collegamento (creato dal server).
 *
 * JAASServer equivale alla relativa superclasse {@link Server Server}
 * sotto tutti gli altri aspetti e
 * può essere eseguito sui client e i server di esempio non JAAS.
 */

class JAASServer extends Server
{
    JAASServer(String programName) throws Exception
    {
        super(programName);
    }

    static class JAASServerAction implements PrivilegedExceptionAction
    {
        private JAASServer server = null;

        JAASServerAction(JAASServer server)
        {
            this.server = server;
        }

        public Object run () throws Exception
        {
            server.initialize();
            server.processRequests();

            return null;
        }
    }

    public static void main(String[] args) throws Exception
    {
        String programName     = "JAASServer";
        Debug dbg = new Debug();

        System.out.println(dbg.toString()); // XXXXXXX

        try {
            // Non impostare useSubjectCredsOnly.
            // Se non impostato, useSubjectCredsOnly viene impostato sul valore predefinito "true".

            JAASServer server = new JAASServer(programName);

            server.processArgs(args);

            LoginContext loginCtxt = new LoginContext(programName,
                                        new Krb5CallbackHandler());

            dbg.out(Debug.OPTS_CAT_APPLICATION, programName + ": Login in ...");

            loginCtxt.login();

            dbg.out(Debug.OPTS_CAT_APPLICATION, programName +
                                            ": Login successful");

            Subject subject = loginCtxt.getSubject();

            JAASServerAction serverAction = new JAASServerAction(server);

            Subject.doAsPrivileged(subject, serverAction, null);
        } catch (Exception exc) {
            dbg.out(Debug.OPTS_CAT_APPLICATION, programName +  " EXCEPTION");
            exc.printStackTrace();
```

```
            throw exc;
        }
    }
}
```

Esempi: IBM Java Secure Sockets Extension 1.4

Gli esempi JSSE mostrano il modo in cui un client e un server possono utilizzare il fornitore System i5 JSSE nativo per creare un contesto che abilita le comunicazioni sicure.

Nota: entrambi gli esempi utilizzano il fornitore JSSE nativo di System i5, indipendentemente dalle proprietà specificate dal file java.security.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

Esempio: richiamo di un programma CL con java.lang.Runtime.exec()

Questo esempio mostra come eseguire i programmi CL dall'interno di un programma Java. In questo esempio, la classe Java class CallCLPgm esegue un programma CL.

Il programma CL utilizza il comando DSPJVAPGM (Visualizzazione programma Java) per visualizzare il programma associato al file di classe Hello. Questo esempio presume che il programma CL sia stato compilato ed esista in una libreria denominata JAVSAMPLIB. L'emissione dal programma CL è nel file di spool QSYSPRT.

Consultare "Esempio: richiamo di un comando CL con java.lang.Runtime.exec()" a pagina 232 per un esempio di come richiamare un comando CL dall'interno di un programma Java.

Nota: JAVSAMPLIB non viene creato come parte del processo di installazione del programma su licenza (LP - licensed program) IBM Developer Kit numero 5761-JV1. È necessario creare la libreria esplicitamente.

Esempio 1: classe CallCLPgm

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallCLPgm
{
   public static void main(String[] args)
   {
      try
      {
         Process theProcess =
                 Runtime.getRuntime().exec("/QSYS.LIB/JAVSAMPLIB.LIB/DSPJVA.PGM");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }
   }  // end main() method
} // end class
```

Esempio 2: visualizzazione del programma CL Java

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
                OUTPUT(*PRINT)
ENDPGM
```

Esempio: richiamo di un comando CL con java.lang.Runtime.exec()

Questo esempio mostra come eseguire un comando CL (Control Language) dall'interno di un programma Java.

In questo esempio, la classe Java esegue un comando CL. Il comando CL utilizza il comando CL DSPJVAPGM (Visualizzazione programma Java) per visualizzare il programma associato al file di classe Hello. L'emissione dal comando CL è nel file di spool QSYSPRT.

Quando si imposta la proprietà di sistema os400.runtime.exec su EXEC (valore predefinito), i comandi che si immettono nella funzione `Runtime.getRuntime().exec()` utilizzano il seguente formato:

```
Runtime.getRuntime()Exec("system CLCOMMAND");
```

dove *CLCOMMAND* è il comando CL che si desidera eseguire.

Nota: quando si imposta os400.runtime.exec su QSHELL, è necessario aggiungere barra e virgolette (\"). Ad esempio il comando precedente appare in questo modo:

```
Runtime.getRuntime()Exec("system \"CLCOMMAND\"");
```

Esempio: classe per richiamare un comando CL

Il seguente codice implica l'utilizzo del valore predefinito di EXEC per la proprietà di sistema os400.runtime.exec.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallCLCom
{
   public static void main(String[] args)
   {
      try
      {
         Process theProcess =
            Runtime.getRuntime().exec("system DSPJVAPGM CLSF('/com/ibm/as400/system/Hello.class')
                                      OUTPUT(*PRINT)");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }
   }  // end main() method
} // end class
```

Concetti correlati

"Utilizzo di java.lang.Runtime.exec()" a pagina 229
Utilizzare il metodo java.lang.Runtime.exec per richiamare comandi o programmi dall'interno di un programma Java. Il metodo java.lang.Runtime.exec() consente di creare uno o più lavori aggiuntivi abilitati a sottoprocessi. Tali lavori elaborano la stringa dei comandi trasmessa al metodo.

"Elenco delle proprietà di sistema Java" a pagina 16
Le proprietà di sistema Java determinano l'ambiente in cui vengono eseguiti i programmi Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in i5/OS.

Esempio: richiamo di un altro programma Java con java.lang.Runtime.exec()

Questo esempio descrive come chiamare un altro programma Java con java.lang.Runtime.exec(). Questa classe richiama il programma Hello fornito come parte di IBM Developer Kit per Java. Quando la classe Hello scrive su System.out, questo programma ottiene un handle al flusso e può leggere da quest'ultimo.

Nota: viene utilizzato Qshell Interpreter per richiamare il programma.

Esempio 1: classe CallHelloPgm

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallHelloPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallHelloPgm.main() invoked");

      // chiamare la classe Hello
      try
      {
          theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }

      // leggere dal flusso di emissione standard del programma chiamato
      try
      {
         inStream = new BufferedReader(
                             new InputStreamReader( theProcess.getInputStream() ));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error on inStream.readLine()");
         e.printStackTrace();
      }

   } // end method

} // end class
```

Esempio: richiamo di Java da C

Questo è un esempio di un programma C che utilizza la funzione system() per chiamare il programma Hello Java.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
#include <stdlib.h>

 int main(void)
 {
```

```
   int result;

 /* La funzione di sistema passa la stringa fornita al processore comandi CL
    per l'elaborazione. */

   result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
 }
```

Esempio: richiamo di Java da RPG

Questo è un esempio di un programma RPG che utilizza l'API QCMDEXC per chiamare il programma Hello Java.

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
D*             DEFINE  THE PARAMETERS FOR THE QCMDEXC API
D*
DCMDSTRING        S             25    INZ('JAVA CLASS(''com.ibm.as400.system.Hello'')')
DCMDLENGTH        S             15P 5 INZ(25)
D*             NOW THE CALL TO QCMDEXC WITH THE 'JAVA' CL COMMAND
C                   CALL      'QCMDEXC'
C                   PARM                    CMDSTRING
C                   PARM                    CMDLENGTH
C*       This next line displays 'DID IT' after you exit the
C*       Java Shell via F3 or F12.
C     'DID IT'      DSPLY
C*       Set On LR to exit the RPG program
C                   SETON                                        LR
C
```

Esempio: utilizzo dei flussi di immissione ed emissione per la comunicazione tra processi

Questo esempio mostra come chiamare un programma C da Java e utilizzare flussi di immissione ed emissione per la comunicazione tra processi.

In questo esempio, il programma C registra una stringa sul relativo flusso di emissione standard e il programma Java legge questa stringa e la visualizza. Questo esempio presume che sia stata creata una libreria denominata JAVSAMPLIB e che sia stato creato il programma CSAMP1 al suo interno.

Nota: JAVSAMPLIB non viene creato come parte del processo di installazione del programma su licenza (LP - licensed program) IBM Developer Kit numero 5761-JV1. È necessario crearlo esplicitamente.

Esempio 1: classe CallPgm

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.io.*;

public class CallPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallPgm.main() invoked");

      // chiamare il programma CSAMP1
      try
      {
          theProcess = Runtime.getRuntime().exec(
```

```
                    "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
      }
      catch(IOException e)
      {
         System.err.println("Error on exec() method");
         e.printStackTrace();
      }

      // leggere dal flusso di emissione standard del programma chiamato
      try
      {
         inStream = new BufferedReader(new InputStreamReader
                         (theProcess.getInputStream()));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error on inStream.readLine()");
         e.printStackTrace();
      }

   } // end method

 } // end class
```

Esempio 2: programma C CSAMP1

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
     /* Convertire la stringa in ASCII nel tempo di compilazione */
#pragma convert(819)
     printf("Program JAVSAMPLIB/CSAMP1 was invoked\n");
#pragma convert(0)
     /* Stdout può essere memorizzato nel buffer, quindi svuotare il buffer */

     fflush(stdout);

}
```

Esempio: API di richiamo Java

Questo esempio segue il paradigma API di richiamo standard.

Effettua quanto segue:

- Crea una JVM (Java virtual machine) utilizzando JNI_CreateJavaVM.
- Utilizza la JVM (Java virtual machine) per trovare il file di classe che si intende eseguire.
- Rileva il methodID per il metodo principale della classe.
- Chiama il metodo principale della classe.
- Notifica gli errori se si verifica un'eccezione.

Quando si crea il programma, il programma di servizio QJVAJNI o QJVAJNI64 fornisce le funzioni dell'API di richiamo JNI_CreateJavaVM. JNI_CreateJavaVM crea la JVM (Java virtual machine).

Nota: QJVAJNI64 è un nuovo programma di servizio per il metodo nativo teraspace/LLP64 e per il supporto dell'API di richiamo.

Questi programmi di servizio risiedono nell'indirizzario di collegamento del sistema e per tanto non occorre identificarli in modo esplicito su un comando di creazione CL (Control Language). Ad esempio, non è necessario identificarli in modo esplicito quando si utilizza il comando CRTPGM (Creazione programma) o il comando CRTSRVPGM (Creazione programma di servizio.

Per eseguire il programma, una delle opzioni possibili è quella di utilizzare il seguente comando CL:

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```

Qualsiasi lavoro che crei una JVM (Java virtual machine) deve essere in grado di supportare più sottoprocessi. L'emissione dal programma principale, così come una qualsiasi emissione dal programma, viene inserita in file di spool QPRINT. I file di spool sono visibili quando si utilizza il comando CL WRKSBMJOB (Gestione lavori inoltrati) e si visualizza il lavoro avviato utilizzando il comando CL SBMJOB (Inoltro lavoro).

Esempio: utilizzo dell'API di richiamo Java

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
#define OS400_JVM_15
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

/* Specificare il programma che fa in modo che tutte le stringhe letterali nel codice
 * sorgente vengano memorizzate in ASCII (il quale, per le stringhe
 * utilizzare, equivale a UTF-8)
 */

#pragma convert(819)

/* Procedura: Oops
 *
 * Descrizione: La routine del programma di aiuto viene chiamata quando una funzione JNI
 *              restituisce un valore zero, indicando un errore serio.
 *              Questa routine riporta l'eccezione a stderr e
 *              chiude senza preavviso la JVM con un FatalError.
 *
 * Parametri:   env -- JNIEnv* da utilizzare per le chiamate JNI
 *              msg -- char* che punta alla descrizione errore in UTF-8
 *
 * Nota:        Il controllo non viene restituito dopo la chiamata a FatalError
 *              e non viene restituito da questa procedura.
 */

void Oops(JNIEnv* env, char *msg) {
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*env)->FatalError(env, msg);
}

/* Questa è la routine "main" del programma. */
int main (int argc, char *argv[])
{

    JavaVMInitArgs initArgs; /* Struttura di inizializzazione VM (Virtual Machine),
                              * passata dal riferimento a JNI_CreateJavaVM(). Vedere jni.h per dettagli
                              */
    JavaVM* myJVM;           /* Puntatore JavaVM impostato dalla chiamata a JNI_CreateJavaVM */
    JNIEnv* myEnv;           /* Puntatore JNIEnv impostato dalla chiamata a JNI_CreateJavaVM */
    char*   myClasspath;     /* Classpath 'string' modificabile */
```

```
|      jclass  myClass;         /* La classe da chiamare, 'NativeHello'. */
|      jmethodID mainID;        /* L'ID metodo della routine 'main'. */
|      jclass  stringClass;     /* Necessario per creare l'arg String[] per main */
|      jobjectArray args;       /* String[] stesso */
|      JavaVMOption options[1]; /* Schiera opzioni -- usare le opzioni per impostare classpath */
|      int     fd0, fd1, fd2;   /* descrittore file per IO */
|
|      /* Aprire i descrittori file in modo che IO sia operativo. */
|      fd0 = open("/dev/null1", O_CREAT|O_TRUNC|O_RDWR,   S_IRUSR|S_IROTH);
|      fd1 = open("/dev/null2", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);
|      fd2 = open("/dev/null3", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);
|
|      /*  Impostare il campo versione degli argomenti di inizializzazione per J2SE v1.5. */
|      initArgs.version = 0x00010005;
|    /*  Per utilizzare J2SDK v1.4, impostare initArgs.version = 0x00010004; */
|
|
|      /* Ora, si desidera specificare l'indirizzario per la classe da eseguire nel classpath.
|       * Con Java2, classpath viene passato come una opzione.
|       * Nota: specificare il nome indirizzario in formato UTF-8. Quindi, raggruppare
|       *       i blocchi di codice in istruzioni #pragma convert.
|       */
|      options[0].optionString="-Djava.class.path=/CrtJvmExample";
|    /* Per utilizzare J2SDK v1.4, sostituire '1.5' con '1.4'.
|    options[1].optionString="-Djava.version=1.5" */
|
|      initArgs.options=options;  /* Inoltrare il classpath impostato. */
|      initArgs.nOptions = 1;     /* Inoltrare le opzioni di classpath e versione */
|
|      /*  Creare la JVM -- un codice di ritorno diverso da zero indica che si è verificato
|       *  un errore. Ritornare a EBCDIC e scrivere un messaggio in stderr
|       *  prima di uscire dal programma.
|       */
|      if (JNI_CreateJavaVM("myJVM, (void **)"myEnv, (void *)"initArgs)) {
|     #pragma convert(0)
|            fprintf(stderr, "Failed to create the JVM\n");
|     #pragma convert(819)
|            exit(1);
|      }
|
|      /*  Utilizzare la JVM appena creata per trovare la classe di esempio,
|       *  chiamata 'NativeHello'.
|       */
|      myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
|      if (! myClass) {
|          Oops(myEnv, "Failed to find class 'NativeHello'");
|      }
|
|      /* Ora, richiamare l'identificativo del metodo per il punto di entrata 'main'
|       * della classe.
|       * Nota: la firma di 'main' è sempre uguale per qualsiasi
|       *       classe chiamata dal seguente comando java:
|       *           "main" , "([Ljava/lang/String;)V"
|       */
|      mainID = (*myEnv)->GetStaticMethodID(myEnv,myClass,"main",
|                                          "([Ljava/lang/String;)V");
|      if (! mainID) {
|          Oops(myEnv, "Failed to find jmethodID of 'main'");
|      }
|
|      /* Richiamare jclass per String per creare la schiera
|       * di String da inoltrare a 'main'.
|       */
|      stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
|      if (! stringClass) {
|          Oops(myEnv, "Failed to find java/lang/String");
|      }
```

```
    /*  Ora, è necessario creare una schiera di stringhe vuota,
     *  poiché main richiede una schiera di questo tipo come parametro.
     */
    args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
    if (! args) {
        Oops(myEnv, "Failed to create args array");
    }

    /* Ora, si ha l'ID metodo di main e la classe, quindi è possibile
     * chiamare il metodo main.
     */
    (*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);

    /* Controllare errori. */
    if ((*myEnv)->ExceptionOccurred(myEnv)) {
        (*myEnv)->ExceptionDescribe(myEnv);
    }

    /* Infine, eliminare la JavaVM creata. */
    (*myJVM)->DestroyJavaVM(myJVM);

    /* Eseguite tutte le operazioni. */
    return 0;
}
```

Per ulteriori informazioni, consultare “API di richiamo Java” a pagina 213.

Esempio: metodo nativo IBM i5/OS PASE per Java

L’esempio del metodo nativo IBM i5/OS PASE per Java chiama un’istanza di un metodo C nativo che, in seguito, utilizza la JNI (Java Native Interface) per richiamarlo nel codice Java. Invece di accedere alla stringa direttamente dal codice Java, l’esempio chiama un metodo nativo che, in seguito, richiama in Java, attraverso la JNI, per ottenere il valore stringa.

Per visualizzare le versioni HTML dei file di origine dell’esempio, utilizzare i collegamenti che seguono:

Nota: attraverso l’utilizzo degli esempi del codice, si accettano i termini di “Informazioni sull’esonero di responsabilità e licenza del codice” a pagina 580.

- “Esempio: PaseExample1.java” a pagina 555
- “Esempio: PaseExample1.c” a pagina 555

Prima di poter eseguire l’esempio di metodo nativo PASE i5/OS, è necessario completare le attività nelle seguenti sezioni:

1. “Esempio: scaricamento del codice sorgente dell’esempio sulla stazione di lavoro AIX” a pagina 556
2. “Esempio: preparazione del codice sorgente di esempio” a pagina 556
3. “Esempio: preparazione di System i5 all’esecuzione dell’esempio di metodo nativo PASE per Java” a pagina 557

Esecuzione dell’esempio di metodo nativo PASE i5/OS per Java

Una volta completate le attività precedentemente descritte, è possibile eseguire l’esempio. Utilizzare uno dei comandi che seguono per eseguire il programma di esempio:

- Da una richiesta comandi i5/OS:

  ```
  JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
  ```

- Da una richiesta comandi Qshell o da una sessione del terminalei5/OS PASE:

  ```
  cd /home/example
  java PaseExample1
  ```

Esempio: PaseExample1.java

Questo programma di esempio carica la libreria metodo nativo 'PaseExample1'. Il codice sorgente per il metodo nativo è contenuto in PaseExample1.c. Il metodo printString in questo programma Java utilizza un metodo nativo, getStringNative, per richiamare il valore della stringa. Il metodo nativo semplicemente restituisce la chiamata al metodo getStringCallback di questa classe.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////////////////////////////////////////////
//
// Questo programma di esempio carica la libreria metodo nativo 'PaseExample1'.
// Il codice sorgente per il metodo nativo si trova in PaseExample1.c
// Il metodo printString in questo programma Java utilizza un metodo nativo,
// getStringNative, per richiamare il valore della stringa. Il metodo nativo
// semplicemente restituisce la chiamata al metodo getStringCallback di questa classe.
//
////////////////////////////////////////////////////////////////////////////



public class PaseExample1 {
    public static void main(String args[]) {
 PaseExample1 pe1 = new PaseExample1("String for PaseExample1");
 pe1.printString();
    }

    String str;

    PaseExample1(String s) {
 str = s;
    }


    //--------------------------------------------------------------
    public void printString() {
 String result = getStringNative();
 System.out.println("Value of str is '" + result + "'");
    }

    // Ciò chiama getStringCallback attraverso JNI.
    public native String getStringNative();

    // Chiamato da getStringNative via JNI.
    public String getStringCallback() {
 return str;
    }


    //--------------------------------------------------------------
    static {
 System.loadLibrary("PaseExample1");
    }

}
```

Esempio: PaseExample1.c

Questo metodo nativo implementa il metodo getStringNative della classe PaseExample1. Utilizza la funzione JNI CallObjectMethod per richiamare il metodo getStringCallback della classe PaseExample1.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/*
 *

 * Questo metodo nativo implementa il metodo getStringNative della classe
 * PaseExample1. Utilizza la funzione JNI CallObjectMethod per richiamare
```

```
 * il metodo getStringCallback della classe PaseExample1.
 *
 * Compilare questo codice in AIX per creare il modulo 'libPaseExample1.so'.
 *
 */

#include "PaseExample1.h"
#include <stdlib.h>

/*
 * Classe:     PaseExample1
 * Metodo:    getStringNative
 * Firma: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_PaseExample1_getStringNative(JNIEnv* env, jobject obj) {
   char* methodName = "getStringCallback";
   char* methodSig = "()Ljava/lang/String;";
   jclass clazz = (*env)->GetObjectClass(env, obj);
   jmethodID methodID = (*env)->GetMethodID(env, clazz, methodName, methodSig);
   return (*env)->CallObjectMethod(env, obj, methodID);
}
```

Esempio: scaricamento del codice sorgente dell'esempio sulla stazione di lavoro AIX

Prima di poter eseguire l'esempio del metodo nativo IBM i5/OS PASE per Java, è necessario scaricare un file compresso che contiene il codice sorgente. Per scaricare il file compresso sulla stazione di lavoro AIX, effettuare quanto segue.

1. Creare un indirizzario temporaneo sulla stazione di lavoro AIX che si desidera contenga i file sorgente.
2. Scaricare il codice sorgente dell'esempio i5/OS PASE nell'indirizzario temporaneo.
3. Decomprimere i file di esempio nell'indirizzario temporaneo.

Per ulteriori informazioni sull'esempio per il metodo nativo IBM i5/OS PASE per Java, consultare i seguenti argomenti:

"Esempio: metodo nativo IBM i5/OS PASE per Java" a pagina 226
L'esempio del metodo nativo IBM i5/OS PASE per Java chiama un'istanza di un metodo C nativo che, in seguito, utilizza la JNI (Java Native Interface) per richiamarlo nel codice Java. Invece di accedere alla stringa direttamente dal codice Java, l'esempio chiama un metodo nativo che, in seguito, richiama in Java, attraverso la JNI, per ottenere il valore stringa.

"Esempio: preparazione del codice sorgente di esempio"
Prima di trasferire l'esempio di metodo nativo PASE IBM i5/OS per Java sul server, è necessario compilare il codice sorgente, creare un file di inclusione C e creare un oggetto di libreria condivisa.

"Esempio: preparazione di System i5 all'esecuzione dell'esempio di metodo nativo PASE per Java" a pagina 557
Prima di eseguire l'esempio di metodo nativo IBM i5/OS PASE per Java, è necessario preparare il server ad eseguire l'esempio. Tale preparazione richiede la copia dei file sul server e l'aggiunta delle variabili di ambiente necessarie per eseguire l'esempio.

Esempio: preparazione del codice sorgente di esempio

Prima di trasferire l'esempio di metodo nativo PASE IBM i5/OS per Java sul server, è necessario compilare il codice sorgente, creare un file di inclusione C e creare un oggetto di libreria condivisa.

L'esempio include i seguenti file sorgente C e Java:

- **PaseExample1.c:** file codice sorgente C che contiene un'implementazione di getStringNative().
- **PaseExample1.java:** il file di codice sorgente Java che richiama il metodo getStringNative nativo nel programma C.

Si utilizza il file .class Java compilato per creare un file di inclusione C, PaseExample1.h, che contiene un prototipo di funzione per il metodo getStringNative contenuto nel codice sorgente C.

Per preparare il codice sorgente dell'esempio sulla stazione di lavoro AIX, effettuare quanto segue:

1. utilizzare il comando che segue per compilare il codice sorgenteJava:

   ```
   javac PaseExample1.java
   ```

2. utilizzare il comando che segue per creare un file di inclusione C che contenga i prototipi del metodo nativo:

   ```
   javah -jni PaseExample1
   ```

   Il nuovo file di inclusione C (PaseExample1.h) contiene un prototipo di funzione per il metodo getStringNative. Il codice sorgente C di esempio (PaseExample1.c) contiene già le informazioni da copiare e modificare dal file di inclusione C per utilizzare il metodo getStringNative. Per ulteriori informazioni sull'utilizzo della JNI, consultare la pagina relativa a Java Native Interface sul sito web della Sun.

3. Utilizzare il comando che segue per compilare il codice sorgente C e creare un oggetto libreria condiviso.

   ```
   xlc -G -I/usr/local/java/J1.5.0/include PaseExample1.c -o libPaseExample1.so
   ```

   Il nuovo file oggetto libreria condiviso (libPaseExample1.so) contiene la libreria di metodi nativi ″PaseExample1″ utilizzata dall'esempio.

   Nota: può essere necessario modificare l'opzione -I in modo che punti all'indirizzario che contiene i file di inclusione del metodo nativo Java corretti (ad esempio, jni.h) per il sistema AIX.

Per ulteriori informazioni sull'esempio per il metodo nativo IBM i5/OS PASE per Java, consultare i seguenti argomenti:

"Esempio: metodo nativo IBM i5/OS PASE per Java" a pagina 226
L'esempio del metodo nativo IBM i5/OS PASE per Java chiama un'istanza di un metodo C nativo che, in seguito, utilizza la JNI (Java Native Interface) per richiamarlo nel codice Java. Invece di accedere alla stringa direttamente dal codice Java, l'esempio chiama un metodo nativo che, in seguito, richiama in Java, attraverso la JNI, per ottenere il valore stringa.

"Esempio: scaricamento del codice sorgente dell'esempio sulla stazione di lavoro AIX" a pagina 556
Prima di poter eseguire l'esempio del metodo nativo IBM i5/OS PASE per Java, è necessario scaricare un file compresso che contiene il codice sorgente. Per scaricare il file compresso sulla stazione di lavoro AIX, effettuare quanto segue.

"Esempio: preparazione di System i5 all'esecuzione dell'esempio di metodo nativo PASE per Java"
Prima di eseguire l'esempio di metodo nativo IBM i5/OS PASE per Java, è necessario preparare il server ad eseguire l'esempio. Tale preparazione richiede la copia dei file sul server e l'aggiunta delle variabili di ambiente necessarie per eseguire l'esempio.

Esempio: preparazione di System i5 all'esecuzione dell'esempio di metodo nativo PASE per Java

Prima di eseguire l'esempio di metodo nativo IBM i5/OS PASE per Java, è necessario preparare il server ad eseguire l'esempio. Tale preparazione richiede la copia dei file sul server e l'aggiunta delle variabili di ambiente necessarie per eseguire l'esempio.

Per preparare il server, completare i passi che seguono:

1. Creare i seguenti indirizzari IFS (integrated file system) sul server che si desidera contenga i file di esempio. Ad esempio, utilizzare il seguente comando CL (control language) per creare l'indirizzario denominato /home/example:

   ```
   mkdir /home/example
   ```

2. Copiare i seguenti file nel nuovo indirizzario:
   - PaseExample1.class

- libPaseExample1.so

3. Da una richiesta comandi i5/OS, utilizzare i seguenti comandi CL (Control Language) per aggiungere le variabili di ambiente necessarie:

```
addenvvar PASE_THREAD_ATTACH 'Y'
addenvvar PASE_LIBPATH '/home/example'
addenvvar QIBM_JAVA_PASE_STARTUP '/usr/lib/start32'
```

Nota: Quando si utilizzano i metodi nativi PASE da una sessione del terminale i5/OS PASE, un ambiente PASE a 32 bit è già avviato. In questo caso, impostare solo PASE_THREAD_ATTACH su Y e PASE_LIBPATH sul percorso per le librerie del metodo nativo PASE. In questa situazione, quando si definisce QIBM_JAVA_PASE_STARTUP, la JVM non viene avviata con esito positivo.

Per ulteriori informazioni sulle variabili di ambiente aggiunte, consultare "Esempi: variabili di ambiente per IBM i5/OS PASE" a pagina 222.

Per ulteriori informazioni sull'esempio per il metodo nativo IBM i5/OS PASE per Java, consultare i seguenti argomenti:

"Esempio: metodo nativo IBM i5/OS PASE per Java" a pagina 226
L'esempio del metodo nativo IBM i5/OS PASE per Java chiama un'istanza di un metodo C nativo che, in seguito, utilizza la JNI (Java Native Interface) per richiamarlo nel codice Java. Invece di accedere alla stringa direttamente dal codice Java, l'esempio chiama un metodo nativo che, in seguito, richiama in Java, attraverso la JNI, per ottenere il valore stringa.

"Esempio: scaricamento del codice sorgente dell'esempio sulla stazione di lavoro AIX" a pagina 556
Prima di poter eseguire l'esempio del metodo nativo IBM i5/OS PASE per Java, è necessario scaricare un file compresso che contiene il codice sorgente. Per scaricare il file compresso sulla stazione di lavoro AIX, effettuare quanto segue.

"Esempio: preparazione del codice sorgente di esempio" a pagina 556
Prima di trasferire l'esempio di metodo nativo PASE IBM i5/OS per Java sul server, è necessario compilare il codice sorgente, creare un file di inclusione C e creare un oggetto di libreria condivisa.

Esempi: utilizzo della JNI (Java Native Interface) per i metodi nativi

Questo programma è un semplice esempio di JNI (Java Native Interface) in cui viene utilizzato un metodo nativo C per visualizzare ″Hello, World.″ Utilizzare lo strumento `javah` con il file di classe NativeHello per generare il file NativeHello.h. Questo esempio presume che l'implementazione C di NativeHello faccia parte di un programma di servizio denominato NATHELLO.

Nota: è necessario che la libreria dove è ubicato il programma di servizio NATHELLO si trovi nell'elenco librerie affinché venga eseguito questo esempio.

Esempio 1: classe NativeHello

Nota: utilizzando gli esempi del codice, si accettano le condizioni contenute nelle "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
public class NativeHello {

    //  Dichiarare un campo di tipo 'String' nell'oggetto NativeHello.
    //  Questo è un campo 'instance', quindi ogni oggetto NativeHello
    //  ne contiene uno.
    public String theString;           // variabile dell'istanza

    //  Dichiarare il metodo nativo stesso. Questo metodo nativo
    //  crea un nuovo oggetto string e crea un riferimento a quest'ultimo
    //  in 'theString'
    public native void setTheString(); // metodo nativo per impostare la stringa

    //  Questo codice 'static initializer' viene chiamato prima che la classe
    //  venga utilizzata.
```

```
    static {

        //  Tentare il caricamento della libreria metodo nativa. Se non la
        //  si trova, scrivere un messaggio in 'out' e tentare un percorso codificato.
        //  Se anche questa possibilità ha esito negativo, uscire.
        try {

            //  System.loadLibrary usa l'elenco librerie in JDK 1.1,
            //  e usa la proprietà java.library.path o la variabile di ambiente LIBPATH
            //  in JDK1.2
              System.loadLibrary("NATHELLO");
            }

        catch (UnsatisfiedLinkError e1) {

            //  Non è stato trovato il programma di servizio.
            System.out.println
              ("I did not find NATHELLO *SRVPGM.");
            System.out.println ("(I will try a hardcoded path)");

            try {

                //  System.load prende il percorso completo del modulo IFS.
                System.load ("/qsys.lib/jniexample.lib/nathello.srvpgm");
                }

        catch (UnsatisfiedLinkError e2) {

            //  A questo punto il programma è completato! Scrivere il messaggio
            //  e uscire.
            System.out.println
              ("<sigh> I did not find NATHELLO *SRVPGM anywhere. Goodbye");
            System.exit(1);
            }
        }
    }

    //  Di seguito viene riportato il codice 'main' di questa classe. Quando si immette
    //  'java NativeHello' sulla riga comandi, viene effettuato quanto segue.
    public static void main(String argv[]){

        //  Assegnare ora un nuovo oggetto NativeHello.
        NativeHello nh = new NativeHello();

        //  Ripetere l'ubicazione.
        System.out.println("(Java) Instantiated NativeHello object");
        System.out.println("(Java) string field is '" + nh.theString + "'");
        System.out.println("(Java) Calling native method to set the string");

        //  Questa è la chiamata al metodo nativo.
        nh.setTheString();

        //  Ora, stampare il valore dopo la chiamata da sottoporre ad un doppio controllo.
        System.out.println("(Java) Returned from the native method");
        System.out.println("(Java) string field is '" + nh.theString + "'");
        System.out.println("(Java) All done...");
    }
}
```

Esempio 2: file di intestazione NativeHello.h generato

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class NativeHello */
```

```
#ifndef _Included_NativeHello
#define _Included_NativeHello
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     NativeHello
 * Method:    setTheString
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_NativeHello_setTheString
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

Questo esempio NativeHello.c mostra l'implementazione del metodo nativo in C. Tale esempio mostra come collegare Java ai metodi nativi. Tuttavia, esso pone in rilievo le complicazioni derivanti dal fatto che System i5 è internamente una macchina EBCDIC (extended binary-coded decimal interchange code). Esso mostra inoltre i problemi dovuti alla mancanza attuale di veri elementi di internazionalizzazione in JNI.

Questi motivi, sebbene non siano nuovi per JNI, causano alcune differenze univoche specifiche per il server System i5 nel codice C che si scrive. È bene ricordare che se si sta scrivendo in stdout o stderr o si sta leggendo da stdin, i propri dati vengono probabilmente codificati in formato EBCDIC.

Nel codice C, è possibile convertire facilmente la maggior parte delle stringhe di costanti letterali, quelle che contengono solo caratteri a 7 bit, nel formato UTF-8 richiesto da JNI. Per effettuare tale operazione, racchiudere tra parentesi tonde le stringhe letterali con pragma di conversione code-page. Tuttavia, dal momento che è possibile scrivere informazioni direttamente in stdout o stderr dal proprio codice C, è possibile lasciare alcune costanti letterali in EBCDIC.

Nota: le istruzioni #pragma convert(0) convertono i dati di caratteri in EBCDIC. Le istruzioni #pragma convert(819) convertono i dati di caratteri in ASCII (American Standard Code for Information Interchange). Tali istruzioni convertono i dati di caratteri nel programma C in fase di compilazione.

Esempio: 3: implementazione del metodo NativeHello.c della classe Java

```
#include <stdlib.h>       /* malloc, free, and so forth */
#include <stdio.h>        /* fprintf(), and so forth */
#include <qtqiconv.H>     /* iconv() interface */
#include <string.h>       /* memset(), and so forth */
#include "NativeHello.h"  /* generated by 'javah-jni' */

/*  Tutte le stringhe letterali sono code page ISO-8859-1 Latin 1 (e con caratteri a 7 bit,
sono anche automaticamente UTF-8). */
#pragma convert(819)  /* gestire tutte le stringhe letterali come ASCII */

/*  Riportare ed eliminare un'eccezione JNI.  */
static void HandleError(JNIEnv*);

/*  Stampare una stringa UTF-8 in stderr nel CCSID (coded character */
set identifier) del lavoro corrente.  */
static void JobPrint(JNIEnv*, char*);

/*  Costanti che descrivono la direzione da prendere:  */
#define CONV_UTF2JOB 1
#define CONV_JOB2UTF 2

/*  Convertire una stringa dal CCSID del lavoro in UTF-8 o viceversa.  */
int StringConvert(int direction, char *sourceStr, char *targetStr);

/*  Implementazione metodo nativo di 'setTheString()'.  */
```

```
JNIEXPORT void JNICALL Java_NativeHello_setTheString
(JNIEnv *env, jobject javaThis)
{
    jclass thisClass;     /* classe per l'oggetto 'this' */
    jstring stringObject; /* nuova stringa, da inserire nel campo 'this' */
    jfieldID fid;         /* ID campo richiesto per aggiornare il campo in 'this' */
    jthrowable exception; /* eccezione, richiamata utilizzando ExceptionOccurred */

    /*  Scrivere lo stato nella console.  */
    JobPrint(env, "( C ) In the native method\n");

    /* Creare il nuovo oggetto string.  */
    if (! (stringObject = (*env)->NewStringUTF(env, "Hello, native world!")))
    {
        /*  Quasi per ogni funzione in JNI, un valore di ritorno null indica che
        si è verificato un errore e che è stata inserita un'eccezione laddove potrà
        essere richiamata da 'ExceptionOccurred()'. In questo caso, l'errore sarà
        tipicamente irreversibile, ma per gli scopi di questo esempio, proseguire,
        ricevere l'errore e continuare.  */
        HandleError(env);
        return;
    }

    /* richiamare la classe dell'oggetto 'this', necessario per richiamare fieldID  */
    if (! (thisClass = (*env)->GetObjectClass(env,javaThis)))
    {
        /*  Una classe null restituita da GetObjectClass indica che si è verificato
        un problema. Invece di gestire questo problema, ritornare semplicemente a
        Java e essere coscienti che questa operazione 'emette' automaticamente
        l'eccezione Java memorizzata.  */
        return;
    }

    /*  Richiamare fieldID per aggiornamento.  */
    if (! (fid = (*env)->GetFieldID(env,
                                    thisClass,
                                    "theString",
                                    "Ljava/lang/String;")))
    {
        /*  Un fieldID null restituito da GetFieldID indica che si è verificato
        un problema. Riportare il problema da questo punto ed eliminarlo.
        Lasciare la stringa invariata.  */
        HandleError(env);
        return;
    }

    JobPrint(env, "( C ) Setting the field\n");

    /*  Effettuare l'aggiornamento reale.
    Nota: SetObjectField è un esempio di interfaccia che non restituisce
    un valore di ritorno verificabile. In questo caso, questo valore è
    necessario per chiamare ExceptionOccurred() per vedere se si è verificato
    un problema con la memorizzazione del valore */
    (*env)->SetObjectField(env, javaThis, fid, stringObject);

    /*  Vedere se l'aggiornamento è riuscito. In caso contrario, riportare l'errore.  */
    if ((*env)->ExceptionOccurred(env)) {

        /*  È stato restituito un oggetto eccezione non null da ExceptionOccurred,
        quindi si è verificato un problema ed è necessario riportare l'errore. */
        HandleError(env);
    }

    JobPrint(env, "( C ) Returning from the native method\n");
    return;
```

```
}

static void HandleError(JNIEnv *env)
{
    /*  Una routine semplice per riportare e gestire un'eccezione.  */
    JobPrint(env, "( C  ) Error occurred on JNI call: ");
    (*env)->ExceptionDescribe(env); /* scrivere i dati eccezione nella console */
    (*env)->ExceptionClear(env);    /* clear the exception that was pending */
}

static void JobPrint(JNIEnv *env, char *str)
{
    char   *jobStr;
    char   buf[512];
    size_t len;

    len = strlen(str);

    /*  Stampare solo la stringa non vuota. */
    if (len) {
        jobStr = (len >= 512) ? malloc(len+1) : &buf;
        if (! StringConvert(CONV_UTF2JOB, str, jobStr))
            (*env)->FatalError
              (env,"ERROR in JobPrint: Unable to convert UTF2JOB");
        fprintf(stderr, jobStr);
        if (len >= 512) free(jobStr);
    }
}

int StringConvert(int direction, char *sourceStr, char *targetStr)
{
    QtqCode_T source, target;     /* parametri per stabilire iconv  */
    size_t    sStrLen, tStrLen;   /* copie locali di lunghezza stringa */
    iconv_t   ourConverter;       /* descrittore di conversione reale */
    int       iconvRC;            /* codice di ritorno dalla conversione */
    size_t    originalLen;        /* lunghezza originale di sourceStr */

    /*  Creare copie locali di dimensioni di immissione ed emissione inizializzate
    sulla dimensione della stringa di immissione. iconv() richiede che i parametri
    lunghezza vengano passati per indirizzo (cioè come int*).          */
    originalLen = sStrLen = tStrLen = strlen(sourceStr);

    /*  Inizializzare i parametri in QtqIconvOpen() su zero.  */
    memset(&source,0x00,sizeof(source));
    memset(&target,0x00,sizeof(target));

    /*  A seconda della direzione del parametro, impostare SOURCE
    o TARGET CCSID su ISO 8859-1 Latin.  */
    if (CONV_UTF2JOB == direction ) {
        source.CCSID = 819;
    }
    else {
        target.CCSID = 819;
    }

    /*  Creare l'oggetto converter iconv_t.  */
    ourConverter = QtqIconvOpen(&target,&source);

    /*  Assicurarsi di avere un convertitore valido, altrimenti viene restituito 0.  */
    if (-1 == ourConverter.return_value) return 0;

    /*  Eseguire la conversione.  */
    iconvRC = iconv(ourConverter,
                    (char**) &sourceStr,
                    &sStrLen,
                    &targetStr,
                    &tStrLen);
```

```
    /*  Se la conversione ha esito negativo, viene restituito zero.  */
    if (0 != iconvRC ) return 0;

    /*  Chiudere il descrittore di conversione.  */
    iconv_close(ourConverter);

    /*  targetStr restituisce un puntatore al carattere che ha appena
    passato l'ultimo carattere convertito, quindi impostare null
    ora.  */
    *targetStr = '\0';

    /*  Restituisce il numero di caratteri elaborati. */
    return originalLen-tStrLen;

}

#pragma convert (0)
```

"Utilizzo della Java Native Interface per i metodi nativi" a pagina 211
Sarebbe opportuno utilizzare i metodi nativi soltanto in casi in cui Java puro non è in grado di rispondere alle esigenze di programmazione dell'utente.

Esempio: utilizzo dei socket per la comunicazione tra processi

Questo esempio utilizza i socket per comunicare tra un programmaJava e un programma C.

È necessario avviare prima il programma C, in ascolto su un socket. Una volta che il programma Java si collega al socket, il programma C invia a questo una stringa utilizzando quel collegamento del socket. La stringa inviata dal programma C rappresenta una stringa ASCII (American Standard Code for Information Interchange) nella codepage 819.

Il programma Java deve essere avviato utilizzando questo comando, `java TalkToC xxxxx nnnn` sulla riga comandi Qshell Interpreter o su un'altra piattaforma Java. Altrimenti immettere `JAVA TALKTOC PARM(xxxxx nnnn)` sulla riga comandi i5/OS per avviare il programma Java. `xxxxx` rappresenta il nome del dominio o l'indirizzo IP (Internet Protocol) del sistema sul quale il programma C è in esecuzione. `nnnn` rappresenta il numero della porta del socket che il programma C sta utilizzando. È necessario utilizzare inoltre questo numero porta come primo parametro sulla chiamata al programma C.

Esempio 1: classe client TalkToC

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.net.*;
import java.io.*;

class TalkToC
{
   private String host = null;
   private int port = -999;
   private Socket socket = null;
   private BufferedReader inStream = null;


   public static void main(String[] args)
   {
      TalkToC caller = new TalkToC();
      caller.host = args[0];
      caller.port = new Integer(args[1]).intValue();
      caller.setUp();
      caller.converse();
      caller.cleanUp();
```

```
}  // end main() method

public void setUp()
{
   System.out.println("TalkToC.setUp() invoked");

   try
   {
      socket = new Socket(host, port);
      inStream = new BufferedReader(new InputStreamReader(
                                    socket.getInputStream()));
   }
   catch(UnknownHostException e)
   {
      System.err.println("Cannot find host called: " + host);
      e.printStackTrace();
      System.exit(-1);
   }
   catch(IOException e)
   {
      System.err.println("Could not establish connection for " + host);
      e.printStackTrace();
      System.exit(-1);
   }
} // end setUp() method

public void converse()
{
   System.out.println("TalkToC.converse() invoked");

   if (socket != null && inStream != null)
   {
      try
      {
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Conversation error with host " + host);
         e.printStackTrace();
      }

   } // end if

}  // end converse() method

public void cleanUp()
{
   try
   {

      if(inStream != null)
      {
         inStream.close();
      }
      if(socket != null)
      {
         socket.close();
      }
   } // end try
   catch(IOException e)
   {
      System.err.println("Error in cleanup");
      e.printStackTrace();
      System.exit(-1);
```

```
      }
   } // end cleanUp() method

} // end TalkToC class
```

SockServ.C viene avviato con l'inoltro in un parametro relativo al numero di porta. Ad esempio, `CALL SockServ '2001'`.

Esempio 2: programma del server SockServ.C

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/time.h>

void main(int argc, char* argv[])
{
   int    portNum = atoi(argv[1]);
   int    server;
   int    client;
   int    address_len;
   int    sendrc;
   int    bndrc;
   char*  greeting;
   struct sockaddr_in  local_Address;
   address_len = sizeof(local_Address);

   memset(&local_Address,0x00,sizeof(local_Address));
   local_Address.sin_family = AF_INET;
   local_Address.sin_port = htons(portNum);
   local_Address.sin_addr.s_addr = htonl(INADDR_ANY);

   #pragma convert (819)
   greeting = "This is a message from the C socket server.";
   #pragma convert (0)

   /*  allocate socket    */
   if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
   {
      printf("failure on socket allocation\n");
      perror(NULL);
      exit(-1);
   }

   /* do bind   */
   if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
   {
     printf("Bind failed\n");
     perror(NULL);
     exit(-1);
   }

   /* invoke listen   */
   listen(server, 1);

   /* wait for client request */
   if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
   {
     printf("accept failed\n");
```

```
      perror(NULL);
      exit(-1);
    }


    /* send greeting to client    */
    if((sendrc = send(client, greeting, strlen(greeting),0))<0)
    {
       printf("Send failed\n");
       perror(NULL);
       exit(-1);
    }

    close(client);
    close(server);

}
```

Esempio: inserimento di istruzioni SQL nell'applicazione Java

La seguente applicazione SQLJ di esempio, App.sqlj, utilizza SQL statico per richiamare e aggiornare i dati dalla tabella EMPLOYEE del database di esempio DB2.

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ; // 1
#sql iterator App_Cursor2 (String) ;

class App
{

  /**********************
   ** Registrare unità **
   **********************/

 static
 {
   try
   {
     Class.forName("com.ibm.db2.jdbc.app.DB2Driver").newInstance();
   }
   catch (Exception e)
   {
     e.printStackTrace();
   }
 }

  /*******************
   **      Main      **
   *******************/

 public static void main(String argv[])
 {
   try
   {
     App_Cursor1 cursor1;
     App_Cursor2 cursor2;

     String str1 = null;
     String str2 = null;
     long   count1;
```

```
// L'URL è jdbc:db2:dbname
String url = "jdbc:db2:sample";

DefaultContext ctx = DefaultContext.getDefaultContext();
if (ctx == null)
{
  try
  {
    // collegarsi con id/parole d'ordine predefiniti
    Connection con = DriverManager.getConnection(url);
    con.setAutoCommit(false);
    ctx = new DefaultContext(con);
  }
  catch (SQLException e)
  {
    System.out.println("Error: could not get a default context");
    System.err.println(e) ;
    System.exit(1);
  }
  DefaultContext.setDefaultContext(ctx);
}

// richiamare i dati dal database
System.out.println("Retrieve some data from the database.");
#sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

// visualizzare la serie di risultati
// cursor1.next() restituisce false quando non ci sono più righe
System.out.println("Received results:");
while (cursor1.next()) // 3
{
  str1 = cursor1.empno(); // 4
  str2 = cursor1.firstnme();

  System.out.print (" empno= " + str1);
  System.out.print (" firstname= " + str2);
  System.out.println("");
}
cursor1.close(); // 9

// richiamare il numero di impiegati dal database
#sql { SELECT count(*) into :count1 FROM employee }; // 5
if (1 == count1)
  System.out.println ("There is 1 row in employee table");
else
  System.out.println ("There are " + count1
                      + " rows in employee table");

// aggiornare il database
System.out.println("Update the database.");
#sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

// richiamare i dati aggiornati dal database
System.out.println("Retrieve the updated data from the database.");
str1 = "000010";
#sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

// visualizzare la serie di risultati
// cursor2.next() restituisce false quando non ci sono più righe
System.out.println("Received results:");
while (true)
{
  #sql { FETCH :cursor2 INTO :str2 }; // 7
  if (cursor2.endFetch()) break; // 8

  System.out.print (" empno= " + str1);
  System.out.print (" firstname= " + str2);
```

```
      System.out.println("");
    }
    cursor2.close(); // 9

    // rollback dell'aggiornamento
    System.out.println("Rollback the update.");
    #sql { ROLLBACK work };
    System.out.println("Rollback done.");
  }
  catch( Exception e )
  {
    e.printStackTrace();
  }
 }
}
```

[1]Dichiarazione di iteratori. Questa sezione dichiara due tipi di iteratori:

- App_Cursor1: dichiara i tipi e i nomi dei dati di colonna e restituisce i valori delle colonne a seconda del nome di colonna (collegamento denominato a colonne).
- App_Cursor2: dichiara i tipi dei dati di colonna e restituisce i valori delle colonne tramite la posizione della colonna (collegamento di posizione alle colonne).

[2]Inizializzazione dell'iteratore. L'oggetto iteratore cursor1 viene inizializzato utilizzando il risultato di una query. La query memorizza il risultato in cursor1.

[3]Avanzamento dell'iteratore alla riga successiva. Il metodo cursor1.next() restituisce il valore Booleano `false` se non esistono più righe da richiamare.

[4]Spostamento di dati. Il metodo del programma di accesso denominato empno() restituisce il valore della colonna denominata empno sulla riga corrente. Il metodo di accesso denominato firstnme() restituisce il valore della colonna denominata firstnme sulla riga corrente.

[5]Dati SELECT in una variabile host. L'istruzione SELECT trasferisce il numero di righe presente in una tabella in una variabile host count1.

[6] Inizializzazione dell'iteratore. L'oggetto iteratore cursor2 viene inizializzato utilizzando il risultato di una query. La query memorizza il risultato in cursor2.

[7]Richiamo di dati. L'istruzione FETCH restituisce il valore corrente della prima colonna dichiarata nel cursore ByPos dalla tabella dei risultati nella variabile host str2.

[8]Controllo della riuscita di un'istruzione FETCH.INTO. Il metodo endFetch() restituisce un valore Booleano true se l'iteratore non è posizionato su una riga, cioè se l'ultimo tentativo di selezionare una riga ha avuto esito negativo. Il metodo endFetch() restituisce false se l'ultimo tentativo di selezionare una riga ha avuto esito positivo. DB2 tenta di selezionare una riga quando viene richiamato il metodo next(). Un'istruzione FETCH...INTO chiama implicitamente il metodo next().

[9]Chiusura degli iteratori. Il metodo close() rilascia qualsiasi risorsa mantenuta dagli iteratori. È necessario chiudere in modo esplicito gli iteratori per assicurarsi che le risorse di sistema vengano rilasciate in modo tempestivo.

Esempi: modifica del codice Java per l'utilizzo delle produzioni socket del client

Questi esempi indicano il modo in cui modificare una classe semplice di socket, denominata simpleSocketClient, cosicché questa utilizzi le produzioni socket per creare tutti i socket. Il primo esempio

mostra la classe simpleSocketClient senza produzioni socket. Il secondo esempio mostra la classe simpleSocketClient con produzioni socket. Nel secondo esempio, simpleSocketClient viene ridenominato factorySocketClient.

Esempio 1: programma del client di socket senza produzioni socket

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* Programma client socket semplice */

import java.net.*;
import java.io.*;

public class simpleSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Creare il socket e collegarsi al server.
    Socket  s = new Socket(args[0], serverPort);
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Esempio 2: programma del client di socket semplice con produzioni socket

```
/* Programma client produzione socket semplice */

// Notare che javax.net.* viene importato per selezionare la classe SocketFactory.
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Modificare il programma simpleSocketClient originale per creare un
    // SocketFactory e utilizzarlo per creare i socket.
```

```
    SocketFactory socketFactory = SocketFactory.getDefault();

    // Ora la produzione crea il socket. Questa è l'ultima modifica
    // al programma simpleSocketClient originale.

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Esempi: modifica del codice Java per l'utilizzo delle produzioni socket del server

Questi esempi indicano il modo in cui modificare una classe semplice di socket, denominata simpleSocketServer, cosicché questa utilizzi le produzioni socket per creare tutti i socket. Il primo esempio mostra la classe simpleSocketServer senza produzioni socket. Il secondo esempio mostra la classe simpleSocketServer con produzioni socket. Nel secondo esempio, simpleSocketServer viene ridenominato factorySocketServer.

Esempio 1: programma del server di socket senza produzioni socket

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* File simpleSocketServer.java*/

import java.net.*;
import java.io.*;

public class simpleSocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    ServerSocket    serverSocket =
      new ServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete gli elementi inviati...

    byte buffer[] = new byte[4096];

    int bytesRead;

    // leggere fino all'"eof" restituito
    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead); // riscriverlo
      os.flush();    // flush del buffer di emissione
    }
```

```
    s.close();
    serverSocket.close();
  }       // end main()

}       // end class definition
```

Esempio 2: programma del server di socket con produzioni socket

```
/* File factorySocketServer.java  */

// importare javax.net per selezionare la classe ServerSocketFactory
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Modificare il simpleSocketServer originale per utilizzare un
    // ServerSocketFactory per creare socket server.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ora fare in modo che la produzione crei il socket server. Questa è l'ultima
    // modifica rispetto al programma originale.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete gli elementi inviati...

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }


    s.close();
    serverSocket.close();
  }

}
```

Esempi: modifica del client Java per l'utilizzo di SSL (secure sockets layer)

Questi esempi indicano il modo in cui modificare una classe, denominata factorySocketClient, in modo da utilizzare SSL (secure socket layer). Il primo esempio mostra la classe factorySocketClient che non utilizza SSL. Il secondo esempio mostra la stessa classe, ridenominata factorySSLSocketClient, che utilizza SSL.

Esempio 1: classe factorySocketClient semplice senza supporto SSL

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* Programma client produzione socket semplice */

import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    SocketFactory socketFactory = SocketFactory.getDefault();

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Esempio 2: classe factorySocketClient semplice con supporto SSL

```
// Notare che è stato importato javax.net.ssl.* per selezionare il supporto SSL
import javax.net.ssl.*;
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySSLSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySSLSocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);
```

```
    // Modificare ciò per creare un SSLSocketFactory invece di un SocketFactory.
    SocketFactory socketFactory = SSLSocketFactory.getDefault();

    // Non è necessario modificare altro.
    // Questo è il vantaggio di utilizzare le produzioni!
    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // Il resto del programma prosegue da qui.
```

Esempi: modifica del server Java per l'utilizzo di SSL (secure sockets layer)

Questi esempi indicano il modo in cui modificare una classe, denominata factorySocketServer, per utilizzare SSL (secure socket layer).

Il primo esempio mostra la classe factorySocketServer che non utilizza SSL. Il secondo esempio mostra la stessa classe, ridenominata factorySSLSocketServer, che utilizza SSL.

Esempio 1: classe factorySocketServer semplice senza supporto SSL

Nota: attraverso l'utilizzo degli esempi del codice, si accettano i termini di "Informazioni sull'esonero di responsabilità e licenza del codice" a pagina 580.

```
/* File factorySocketServer.java  */
// importare javax.net per selezionare la classe ServerSocketFactory
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Modificare il simpleSocketServer originale per utilizzare un
    // ServerSocketFactory per creare socket server.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ora fare in modo che la produzione crei il socket server. Questa è l'ultima
    // modifica rispetto al programma originale.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete solo gli elementi inviati.

    byte buffer[] = new byte[4096];
```

```
    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

Esempio 2: classe factorySocketServer semplice con supporto SSL

```
/* File factorySocketServer.java  */

// importare javax.net per selezionare la classe ServerSocketFactory
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Modificare il simpleSocketServer originale per utilizzare un
    // ServerSocketFactory per creare socket server.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ora fare in modo che la produzione crei il socket server. Questa è l'ultima
    // modifica rispetto al programma originale.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // un server reale gestirebbe più di un client come questo...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Questo server ripete solo gli elementi inviati.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

Risoluzione dei problemi di IBM Developer Kit per Java

Questo argomento, mostra come trovare le registrazioni lavori e raccogliere i dati per l'analisi di un programma Java. Questo argomento fornisce inoltre informazioni sulle PTF (program temporary fix) e su come ottenere assistenza per IBM Developer Kit per Java.

Se le prestazioni del programma peggiorano dopo che questo è stato in esecuzione più a lungo è possibile aver codificato per errore una perdita di memoria. È possibile utilizzare il JavaWatcher, un componente di System i iDoctor, per ottenere un aiuto nell'esecuzione del debug sul programma e per individuare le perdite di memoria. Per ulteriori informazioni, consultare Heap Analysis Tools for Java.

Limiti

Quest'elenco identifica limiti e restrizioni noti o comportamenti unici in IBM Developer Kit per Java.

- Quando una classe viene caricata e le relative superclassi non vengono rilevate, l'errore indica che la classe originale non è stata rilevata. Ad esempio, se la classe B estende la classe A e la classe A non viene rilevata durante il caricamento della classe B, l'errore indica che la classe B non è stata rilevata, anche se è la classe A che in realtà non è stata rilevata. Quando si verifica un errore che indica che una classe non è stata rilevata, assicurarsi che la classe e tutte le relative superclassi si trovino nel CLASSPATH. Questo discorso vale anche per le interfacce che vengono implementate dalla classe caricata.
- L'heap della raccolta di dati inutili è limitato a 240 GB.
- Non c'è un limite esplicito per il numero di oggetti creati.
- Il parametro java.net backlog su System i5 può comportarsi in modo diverso rispetto alle altre piattaforme. Ad esempio:
  - Listen backlogs 0, 1
    - Listen(0) indica che un collegamento in sospeso viene consentito; questo non disabilita un socket.
    - Listen(1) indica che un commento in sospeso viene consentito e indica le stesse operazioni valide per Listen(0).
  - Listen backlogs > 1
    - Questo consente a numerose richieste in sospeso di rimanere sulla coda di ascolto. Se arriva una nuova richiesta di collegamento e la coda si trova al limite, allora viene cancellata una delle richieste in sospeso.
- È possibile utilizzare solo la JVM (Java virtual machine), indipendentemente dalla versione di JDK che si sta utilizzando, in ambienti in grado di supportare più sottoprocessi (cioè sicuri a livello dei sottoprocessi). La piattaforma System i5 è sicura a livello dei sottoprocessi, ma alcuni file system non lo sono. Per un elenco di file system che non sono sicuri a livello dei sottoprocessi, consultare l'argomento Integrated File System.

Ricerca delle registrazioni lavori per un'analisi dei problemi Java

Utilizzare la registrazione lavori dal lavoro che ha eseguito il comando Java e la registrazione lavori BCI (batch immediato) dove è stato eseguito il programma Java, per analizzare le cause dell'errore Java. È possibile che entrambe contengano importanti informazioni sull'errore.

Esistono due modi per trovare la registrazione lavori per il lavoro BCI. È possibile trovare il nome del lavoro BCI registrato nella registrazione lavori del lavoro che ha eseguito il comando Java. Quindi, utilizzare quel nome lavoro per trovare la registrazione lavori per il lavoro BCI.

È inoltre possibile trovare la registrazione lavori per il lavoro BCI effettuando le seguenti fasi:

1. Immettere il comando WRKSBMJOB (Gestione lavori inoltrati) sulla riga comandi i5/OS.
2. Andare alla fine dell'elenco.
3. Cercare l'ultimo lavoro nell'elenco, denominato QJVACMDSRV.

4. Immettere l'opzione 8 (Gestione file di spool) per quel lavoro.
5. Viene visualizzato un file denominato QPJOBLOG.
6. Premere F11 per vedere la vista 2 dei file di spool.
7. Verificare che la data e l'ora corrispondano alla data e all'ora in cui si è verificato l'errore.

   Se la data e l'ora non corrispondono alla data e all'ora in cui l'utente si è scollegato, continuare a cercare nell'elenco dei lavori inoltrati. Tentare di trovare una registrazione lavori QJVACMDSRV con una data e ora che corrispondano alla data e all'ora in cui l'utente si è scollegato.

Se l'utente non trova una registrazione lavori per il lavoro BCI, è possibile che non ne sia stata prodotta una. Ciò si verifica se si imposta il valore ENDSEP per la descrizione lavoro QDFTJOBD troppo alto o il valore LOG per la descrizione lavoro QDFTJOBD specifica *NOLIST. Controllare questi valori e modificarli in modo che si produca una registrazione lavori per il lavoro BCI.

Per produrre una registrazione lavori per il lavoro che ha eseguito il comando RUNJVA (Esecuzione Java), effettuare quanto segue:

1. Immettere SIGNOFF *LIST.
2. Quindi ricollegarsi.
3. Immettere il comando WRKSPLF (Gestione file di spool) sulla riga comandi i5/OS.
4. Andare alla fine dell'elenco.
5. Trovare un file denominato QPJOBLOG.
6. Premere F11.
7. Verificare che la data e l'ora corrispondano alla data e all'ora in cui si è emesso il comando di scollegamento.

   Se la data e l'ora non corrispondono alla data e all'ora in cui l'utente si è scollegato, continuare a cercare nell'elenco dei lavori inoltrati. Tentare di trovare una registrazione lavori QJVACMDSRV con una data e ora che corrispondano alla data e all'ora in cui l'utente si è scollegato.

Raccolta di dati per l'analisi dei problemi Java

Per raccogliere i dati per un APAR (authorized program analysis report), seguire queste fasi.

1. Includere una descrizione completa del problema.
2. Salvare il file della classe Java che ha causato il problema durante l'esecuzione.
3. È possibile utilizzare il comando SAV per salvare gli oggetti dall'IFS (Integrated File System). È necessario salvare altri file della classe che questo programma deve eseguire. È inoltre possibile salvare e inviare in un intero indirizzario affinché IBM lo utilizzi nel tentativo di riprodurre il problema, se necessario. Questo è un esempio di come salvare un intero indirizzario.

   Esempio: come salvare un indirizzario

   Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

   ```
   SAV DEV('/QSYS.LIB/TAP01.DEVD') OBJ(('/mydir'))
   ```

   Se possibile, salvare i file di origine per ogni classe Java coinvolta nel problema. Ciò è utile all'IBM per riprodurre e analizzare il problema.
4. Salvare ogni programma di servizio che contenga metodi nativi richiesti per eseguire il programma.
5. Salvare ogni file di dati necessario per eseguire il programma Java.
6. Aggiungere una descrizione completa di come riprodurre il problema.

   Questa comprende:

   - Il valore della variabile di ambiente CLASSPATH.
   - Una descrizione del comando Java che è stato eseguito.
   - Una descrizione di come rispondere ad ogni immissione richiesta dal programma.

7. Includere ogni registrazione VLIC (Vertical licensed internal code) che si sia verificata in prossimità del momento dell'errore.
8. Aggiungere la registrazione lavori sia dal lavoro interattivo che dal lavoro BCI dove era in esecuzione la JVM (Java virtual machine).

Applicazione delle PTF (program temporary fix)

A partire da i5/OS V5R4, è possibile utilizzare il comando CL DSPJVMJOB (Visualizzazione dei lavori Java Virtual Machine) per gestire i lavori JVM ed applicare le PTF mentre il sistema è attivo.

Molte PTF (Program Temporary Fix) Java influiscono sulla JVM in modo tale che se si applica il codice mentre è in esecuzione un lavoro JVM, l'applicazione della PTF causerà dei risultati imprevisti. In passato, l'applicazione di alcune PTF Java veniva ritardata fino al momento in cui era possibile eseguire un IPL (initial program load) sul sistema per verificare che non ci fosse alcun lavoro JVM in esecuzione sul sistema. Per molti utenti, comunque, questo era scomodo. È stata quindi aggiunta una pre-condizione JVM in modo che molte di quelle PTF ritardate potessero venire applicate immediatamente *se nessuna JVM è attiva sul sistema*. Il comando DSPJVMJOB consente di vedere quali lavori contengono JVM in esecuzione. Con tale informazione, è possibile terminare i lavori che contengono JVM attive prima di applicare le PTF, invece di aspettare di eseguire un IPL prima di poter applicare le PTF.

Per maggiori informazioni sul comando DSPJVMJOB, consultare la sezione relativa alla visualizzazione dei lavori Java Virtual Machine nell'argomento CL.

Informazioni correlate

Manutenzione e gestione di i5/OS e del software correlato

Utilizzo di correzioni software

Come ottenere il supporto per IBM Developer Kit per Java

I servizi di supporto per IBM Developer Kit per Java sono forniti nei soliti termini e condizioni per i prodotti software System i. I servizi di supporto includono servizi di programma, supporto vocale e servizi di consultazione.

Utilizzare le informazioni online fornite sulla IBM System i Home Page sotto l'argomento ″Support″ per ulteriori informazioni. Utilizzare IBM Support Services per 5761-JV1 IBM Developer Kit per Java). O contattare il proprio rappresentante IBM locale.

IBM può richiedere di ottenere un livello più recente di IBM Developer Kit per Java per ricevere i CPS (Continued Program Services). Per ulteriori informazioni, consultare Supporto per più JDK (Java Development Kit).

I difetti di risoluzione del programma IBM Developer Kit per Java sono supportati nei servizi del programma o nel supporto vocale. Le questioni del debug o della programmazione applicazione di definizione sono supportate nei servizi di consultazione.

Le chiamate API (application program interface) IBM Developer Kit per Java sono supportate nei servizi di consultazione, tranne quando:

1. È chiaramente un difetto dell'API di Java come dimostrato dalla nuova creazione in un programma relativamente semplice.
2. È una questione che richiede una chiarifica della documentazione,
3. È una questione sulla collocazione di esempi o di documentazione.

L'intera assistenza di programmazione è supportata dai servizi di consultazione. Ciò include gli esempi del programma forniti nel prodotto del programma su licenza (LP - licensed program) di IBM Developer Kit per Java. Degli ulteriori esempi potrebbero essere disponibili su Internet nella IBM System i Home Page su una base non supportata.

LP di IBM Developer Kit per Java fornisce informazioni sulla risoluzione dei problemi. Se si ritiene che esista un potenziale difetto nella API IBM Developer Kit per Java, è richiesto un semplice programma che dimostri l'errore.

Informazioni correlate per IBM Developer Kit per Java

Sono qui di seguito elencate alcune fonti correlate all'argomento IBM Developer Kit per Java.

Siti Web

- Java.sun.com: la fonte per sviluppatori Java (www.java.sun.com)
  Visitare il sito di Sun Microsystems, Inc. per informazioni sui vari utilizzi per Java, comprese le nuove tecnologie.
- Area dedicata alla IBM developerWorks Java technology
  Offre informazioni, formazione e strumenti di ausilio nell'utilizzo dei prodotti Java, IBM e di altre tecnologie per creare soluzioni aziendali.
- IBM alphaWorks Java
  Include informazioni sulle nuove tecnologie Java, compresi i download e i collegamenti alle risorse di sviluppo.

Javadoc

Le seguenti informazioni di riferimento ai Javadoc sono correlate a IBM Developer Kit per Java:

- Javadoc JAAS specifico per System i5
- Specifica dell'API JAAS
- Java 2 Platform, Standard Edition API Specification di Sun Microsystems, Inc.

Consultare le seguenti informazioni di riferimento correlate a IBM Developer Kit per Java.

JNDI (Java Naming and Directory Interface)

La JNDI (Java Naming and Directory Interface) fa parte dell'API (Application program interface) della piattaforma JavaSoft. Con JNDI, è possibile collegarsi, senza connessioni fisiche, a più servizi dell'indirizzario e di denominazione. È possibile creare applicazioni Java abilitate all'indirizzario trasferibili e potenti utilizzando questa interfaccia.

JavaSoft ha sviluppato la specifica JNDI con importanti partner industriali, ad esempio IBM, SunSoft, Novell, Netscape e Hewlett-Packard Co.

Nota: Il JRE (i5/OS Java Runtime Environment) e le versioni di J2SE (Java 2 Platform, Standard Edition) offerti da IBM Developer Kit per Java includono il fornitore LDAP Sun. Poiché il supporto i5/OS Java include il fornitore LDAP di Sun, tale supporto non include più il file ibmjndi.jar. Il file ibmjndi.jar offriva un fornitore di servizi LDAP sviluppato da IBM per versioni precedenti di J2SDK.

Java Naming and Directory interface by Sun Microsystems, Inc.

JavaMail

L'API JavaMail(TM) fornisce una serie di classi astratte che modella un sistema elettronico (e-mail). L'API fornisce funzioni di posta per la lettura e l'invio di posta generale e richiede tecnici di manutenzione per implementare i protocolli.

I tecnici di manutenzione implementano protocolli specifici. Ad esempio, SMTP (Simple Mail Transfer Protocol) è un protocollo di trasmissione per l'invio di e-mail. POP3 (Post Office Protocol 3) è il protocollo standard per la ricezione di e-mail. IMAP (Internet Message Access Protocol) è un protocollo alternativo a POP3.

In aggiunta ai tecnici di manutenzione, JavaMail richiede la JAF (JavaBeans Activation Framework) per gestire il contenuto della posta che non sia di semplice testo. Ciò include MIME (Multipurpose Internet Mail Extensions), pagine URL (Uniform Resource Locator) e allegati file.

Tutti i componenti JavaMail vengono forniti come parte di IBM Developer Kit per Java. Questi componenti includono quanto segue:

- **mail.jar** Questo file JAR contiene API JavaMail, il tecnico di manutenzione SMTP, il tecnico di manutenzione POP3 e il tecnico di manutenzione IMAP.
- **activation.jar** Questo file JAR contiene il JavaBeans Activation Framework.

 JavaMail

JPS (Java Print Service)

L'API JPS (Java Print Service) consente la stampa su tutte le piattaforme Java. Java 1.4 e le versioni successive forniscono una framework in cui i JRE (Java runtime environment) e terze parti possono fornire moduli aggiuntivi di creazione flussi per la produzione di vari formati di stampa, come il PDF, il Postscript e l'Advanced Function Presentation (AFP). Questi moduli aggiuntivi creano i formati di emissione dalle chiamate grafiche bidimensionali (2D).

Un servizio di stampa System i5 rappresenta un'unità di stampa configurata su System i5 con il comando i5/OS CRTDEVPRT (Crea descrizione unità (Stampante)). Specificare i parametri delle informazioni di pubblicazione quando si crea un'unità di stampa. Ciò aumenta il numero di attributi del servizio di stampa supportati dai servizi di stampa System i5.

Se una stampante supporta SNMP (Simple Network Management Protocol), configurare la stampante sul server. Specificare *IBMSNMPDRV come valore del parametro del programma di controllo di sistema sul comando CRTDEVPRT. I servizi di stampa utilizzano SNMP per richiamare informazioni specifiche (attributi servizio stampa) su una stampante configurata.

Le tipologie di documento (Doc Flavors) supportate da System i5 includono *AFPDS, *SCS, *USERASCII - (PCL), *USERASCII - (Postscript) e *USERASCII - (PDF). Specificare le tipologie di documento (Doc Flavors) supportate dalla stampante nel parametro Flussi dati supportati, all'interno delle Informazioni di pubblicazione del comando CRTDEVPRT.

Quando un'applicazione utilizza un servizio di stampa per stampare un lavoro (documento) su System i5, il servizio colloca il documento in un file di spool su una coda di emissione con lo stesso nome dell'unità di stampa (lo stesso nome specificato nell'attributo PrinterName). Avviare un programma di scrittura di stampa con il comando STRPRTWTR prima che i documenti vengano stampati sull'unità di stampa.

Oltre agli attributi definiti dalla specifica JPS (Java Print Service), i servizi di stampa System i5 supportano i seguenti attributi per tutte le tipologie di documento (Doc Flavors):

- PrinterFile (specifica un file di stampa, nome e libreria, da utilizzare durante la creazione del file di spool)
- SaveSpooledFile (indica se salvare o meno il file di spool)
- UserData (una stringa lunga 10 caratteri di dati definiti dall'utente)
- JobHold (indica se congelare o meno il file di spool)
- SourceDrawer (indica il cassetto di alimentazione da utilizzare per il supporto di emissione)

Come abilitare JPS durante l'utilizzo di JDK 1.5

Quelli che seguono sono i collegamenti simbolici da configurare per abilitare il JPS (Java Print Service):

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjps.jar')
     NEWLNK('/QIBM/ProdData/Java400/jdk15/lib/ext/ibmjps.jar')
     LNKTYPE(*SYMBOLIC)

ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
     NEWLNK('/QIBM/ProdData/Java400/jdk15/lib/ext/jt400Native.jar')
     LNKTYPE(*SYMBOLIC)
```

Informazioni correlate

JPS (Java Print Service) di Sun Microsystems, Inc.

Informazioni sull'esonero di responsabilità e licenza del codice

IBM fornisce una licenza non esclusiva per utilizzare tutti gli esempi del codice di programmazione da cui creare funzioni simili personalizzate, in base a richieste specifiche.

FATTE SALVE LE GARANZIE INDEROGABILI DI LEGGE, IBM, I SUOI SVILUPPATORI DI PROGRAMMI E FORNITORI NON FORNISCONO GARANZIE O DICHIARAZIONI DI ALCUN TIPO, ESPRESSE O IMPLICITE, INCLUSE, A TITOLO ESEMPLIFICATIVO, GARANZIE O CONDIZIONI IMPLICITE DI COMMERCIABILITÀ O IDONEITÀ PER UNO SCOPO PARTICOLARE, INCLUSE LE GARANZIE DI FUNZIONAMENTO ININTERROTTO, RELATIVE AL CODICE O AL SUPPORTO TECNICO, SE ESISTENTE.

IN NESSUN CASO IBM, I SUOI SVILUPPATORI DI PROGRAMMI O FORNITORI SONO RESPONSABILI PER QUANTO SEGUE ANCHE SE INFORMATI DELLA POSSIBILITÀ DEL VERIFICARSI DI TALI DANNI:

1. PERDITA O DANNEGGIAMENTO DI DATI;
2. DANNI PARTICOLARI, INCIDENTALI, DIRETTI O INDIRETTI O QUALSIASI DANNO ECONOMICO CONSEGUENTE; OPPURE
3. PERDITE DI PROFITTI, AFFARI, ENTRATE O SPESE ANTICIPATE.

LA LEGISLAZIONE DI ALCUNI PAESI NON CONSENTE L'ESCLUSIONE O LA LIMITAZIONE DELLE GARANZIE DI DANNI DIRETTI, INCIDENTALI O CONSEQUENZIALI, PERTANTO ALCUNE O TUTTE LE SUDDETTE ESCLUSIONI O LIMITAZIONI POTREBBERO NON ESSERE APPLICABILI.

Appendice. Informazioni particolari

Queste informazioni sono state progettate per prodotti e servizi offerti negli Stati Uniti.

IBM potrebbe non fornire ad altri paesi prodotti, servizi o funzioni discussi in questo documento. Contattare il rappresentante IBM locale per informazioni sui prodotti e servizi correntemente disponibili nella propria area. Qualsiasi riferimento ad un prodotto, programma o servizio IBM non implica che sia possibile utilizzare soltanto tali prodotti, programmi o servizi IBM. In sostituzione a quanto fornito da IBM, è possibile utilizzare qualsiasi prodotto, programma o servizio funzionalmente equivalente che non violi alcun diritto di proprietà intellettuale di IBM. Tuttavia la valutazione e la verifica dell'uso di prodotti o servizi non IBM ricadono esclusivamente sotto la responsabilità dell'utente.

IBM può avere brevetti o domande di brevetto in corso relativi a quanto trattato nel presente documento. La fornitura di questa pubblicazione non implica la concessione di alcuna licenza su tali brevetti. Chi desiderasse ricevere informazioni relative a licenza può rivolgersi per iscritto a:

IBM Director of Commercial Relations
IBM Europe
Schoenaicher Str. 220
D-7030 Boeblingen
Deutschland

Per informazioni sulle richieste di licenze relative al doppio byte (DBCS), contattare il reparto proprietà intellettuale IBM nel proprio paese o inviare le richieste per iscritto all'indirizzo:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

Le disposizioni contenute nel seguente paragrafo non si applicano al Regno Unito o ad altri paesi nei quali tali disposizioni non siano congruenti con le leggi locali: IBM FORNISCE QUESTA PUBBLICAZIONE COSI' COM'È SENZA ALCUNA GARANZIA, ESPLICITA O IMPLICITA, IVI INCLUSE EVENTUALI GARANZIE DI COMMERCIABILITÀ ED IDONEITÀ AD UNO SCOPO PARTICOLARE. Alcuni stati non consentono la recessione da garanzie implicite o esplicite in alcune transazioni, quindi questa specifica potrebbe non essere applicabile in determinati casi.

Queste informazioni potrebbero contenere imprecisioni tecniche o errori tipografici. Si effettuano periodicamente modifiche alle informazioni qui accluse; queste modifiche saranno inserite in nuove edizioni della pubblicazione. IBM può apportare perfezionamenti e/o modifiche nel(i) prodotto(i) e/o nel(i) programma(i) descritto(i) in questa pubblicazione in qualsiasi momento senza preavviso.

Qualsiasi riferimento a siti web non IBM, contenuto in queste informazioni, viene fornito solo per comodità e non implica in alcun modo l'approvazione di tali siti. Le informazioni reperibili nei siti web non sono parte integrante delle informazioni relative a questo prodotto IBM, pertanto il loro utilizzo ricade sotto la responsabilità dell'utente.

IBM può utilizzare o distribuire le informazioni fornite in qualsiasi modo ritenga appropriato senza obblighi verso l'utente.

Sarebbe opportuno che coloro che hanno la licenza per questo programma e desiderano avere informazioni su di esso allo scopo di consentire: (i) lo scambio di informazioni tra programmi creati in maniera indipendente e non (compreso questo), (ii) l'uso reciproco di tali informazioni, contattassero:

IBM Europe

Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Tali informazioni possono essere disponibili, soggette a termini e condizioni appropriate, compreso in alcuni casi il pagamento di una tariffa.

Il programma su licenza descritto in questa pubblicazione e tutti il relativo materiale disponibile viene fornito da IBM nei termini dell'IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code o qualsiasi altro accordo equivalente tra le parti.

Qualsiasi dato sulle prestazioni contenuto in questa pubblicazione è stato stabilito in un ambiente controllato. Quindi i risultati ottenuti in altri ambienti operativi potrebbero variare in modo significativo. È possibile che alcune misurazioni siano state effettuate su sistemi a livello di sviluppo e non esiste alcuna garanzia che tali misurazioni siano le stesse su sistemi generalmente disponibili. Inoltre, è possibile che alcune misurazioni siano state calcolate tramite estrapolazione. I risultati effettivi possono variare. Sarebbe opportuno che gli utenti di questa pubblicazione verificassero i dati applicabili per il relativo ambiente specifico.

Le informazioni riguardanti prodotti non IBM sono ottenute dai fornitori di tali prodotti, dai loro annunci pubblicati o da altre fonti pubblicamente reperibili. IBM non ha testato tali prodotti e non può confermare l'inadeguatezza delle prestazioni, della compatibilità o di altre richieste relative a prodotti non IBM. Domande inerenti alle prestazioni di prodotti non IBM dovrebbero essere indirizzate ai fornitori di tali prodotti.

Tutte le specifiche relative alle direttive o intenti futuri di IBM sono soggette a modifiche o a revoche senza notifica e rappresentano soltanto scopi ed obiettivi.

Tutti i prezzi IBM mostrati sono i prezzi al dettaglio suggeriti da IBM, sono attuali e soggetti a modifica senza preavviso. I prezzi al fornitore possono variare.

Queste informazioni sono solo per scopi di pianificazione. Le presenti informazioni sono soggette a modifiche prima che i prodotti descritti siano resi disponibili.

Queste informazioni contengono esempi di dati e report utilizzati in quotidiane operazioni aziendali. Per illustrarle nel modo più completo possibile, gli esempi includono i nomi di individui, società, marchi e prodotti. Tutti questi nomi sono fittizi e qualsiasi somiglianza con nomi ed indirizzi utilizzati da gruppi aziendali realmente esistenti è puramente casuale.

LICENZA DI COPYRIGHT:

Queste informazioni contengono programmi applicativi di esempio nella lingua di origine, che illustrano le tecniche di programmazione su varie piattaforme operative. È possibile copiare, modificare e distribuire questi programmi di esempio in qualsiasi formato senza pagare all'IBM, allo scopo di sviluppare, utilizzare, commercializzare o distribuire i programmi dell'applicazione conformi all'interfaccia di programmazione dell'applicazione per la piattaforma operativa per cui i programmi di esempio vengono scritti. Questi esempi non sono stati interamente testati in tutte le condizioni. IBM, perciò, non fornisce nessun tipo di garanzia o affidabilità implicita, rispetto alla funzionalità o alle funzioni di questi programmi.

Ogni copia o copia parziale dei Programmi di esempio o di qualsiasi loro modifica, deve includere il seguente avviso relativo al copyright:

© (nome della società) (anno). Parti di questo codice provengono da IBM Corp. Sample Programs. © Copyright IBM Corp. _immettere l'anno o gli anni_. Tutti i diritti riservati.

Se si sta utilizzando la versione in formato elettronico di questo manuale, le fotografie e le illustrazioni a colori potrebbero non essere visualizzate.

Informazioni sull'interfaccia di programmazione

Questi documenti di pubblicazione di IBM Developer Kit per Java riguardano Interfacce di programmazione che consentono al cliente di scrivere programmi per ottenere i servizi di IBM Developer Kit per Java.

Marchi

I seguenti termini sono marchi dell'International Business Machines Corporation negli Stati Uniti e in altri paesi:

Advanced Function Presentation
AFP
AIX
alphaWorks
C/400
DB2
DB2 Universal Database
developerWorks
Distributed Relational Database Architecture
DRDA
i5/OS
IBM
Integrated Language Environment
iSeries
OS/400
PowerPC
System i
System i5
VisualAge
WebSphere

Adobe, il logo Adobe, PostScript ed il logo PostScript sono marchi di Adobe Systems Incorporated negli Stati Uniti e/o negli altri paesi.

Microsoft, Windows, Windows NT e il logo Windows sono marchi registrati della Microsoft Corporation negli Stati Uniti e/o negli altri paesi.

Java e tutti i marchi e i logo basati su Java sono marchi o marchi registrati della Sun Microsystems, Inc. negli Stati Uniti e/o negli altri paesi.

UNIX è un marchio registrato negli Stati Uniti e in altri paesi con licenza esclusiva di Open Group.

Nomi di altre società, prodotti o servizi possono essere marchi di altre società.

Termini e condizioni

Le autorizzazioni per l'utilizzo di queste pubblicazioni vengono concesse in base alle seguenti disposizioni.

Uso personale: È possibile riprodurre queste pubblicazioni per uso personale, non commerciale a condizione che vengano conservate tutte le indicazioni relative alla proprietà. Non è possibile distribuire, visualizzare o produrre lavori derivati di tali pubblicazioni o di qualsiasi loro parte senza chiaro consenso da parte di IBM.

Uso commerciale: È possibile riprodurre, distribuire e visualizzare queste pubblicazioni unicamente all'interno del proprio gruppo aziendale a condizione che vengano conservate tutte le indicazioni relative alla proprietà. Non è possibile effettuare lavori derivati di queste pubblicazioni o riprodurre, distribuire o visualizzare queste pubblicazioni o qualsiasi loro parte al di fuori del proprio gruppo aziendale senza chiaro consenso da parte di IBM.

Fatto salvo quanto espressamente concesso in questa autorizzazione, non sono concesse altre autorizzazioni, licenze o diritti, espressi o impliciti, relativi alle pubblicazioni o a qualsiasi informazione, dato, software o altra proprietà intellettuale qui contenuta.

IBM si riserva il diritto di ritirare le autorizzazioni qui concesse qualora, a propria discrezione, l'utilizzo di queste pubblicazioni sia a danno dei propri interessi o, come determinato da IBM, qualora non siano rispettate in modo appropriato le suddette istruzioni.

Non è possibile scaricare, esportare o ri-esportare queste informazioni se non pienamente conformi con tutte le leggi e le norme applicabili, incluse le leggi e le norme di esportazione degli Stati Uniti.

IBM NON RILASCIA ALCUNA GARANZIA RELATIVAMENTE AL CONTENUTO DI QUESTE PUBBLICAZIONI. LE PUBBLICAZIONI SONO FORNITE "COSI' COME SONO", SENZA ALCUN TIPO DI GARANZIA, ESPRESSA O IMPLICITA, INCLUSE, A TITOLO ESEMPLIFICATIVO, GARANZIE IMPLICITE DI COMMERCIABILITA' ED IDONEITA' PER UNO SCOPO PARTICOLARE.

Stampato in Italia