

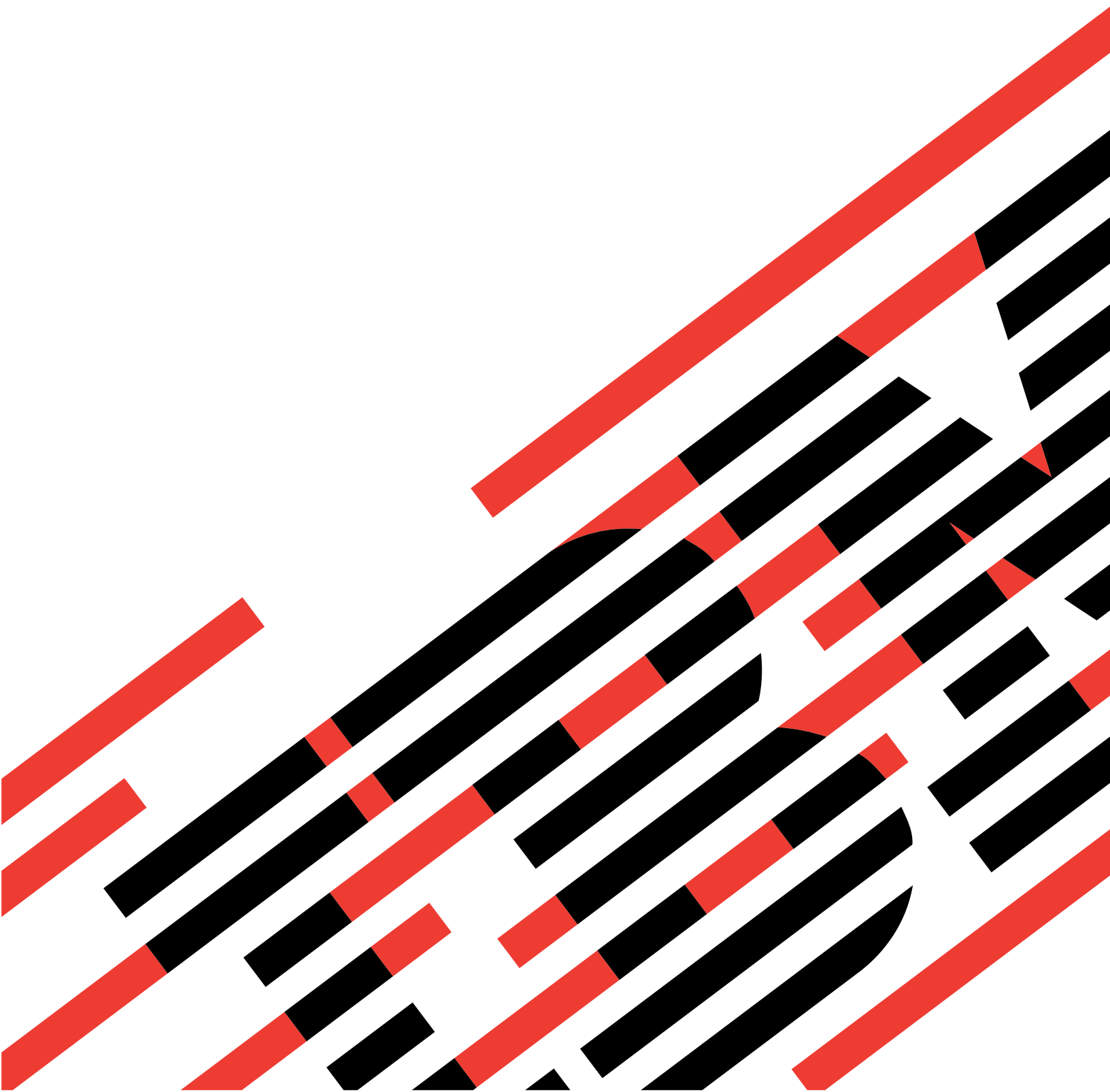
IBM

@server

iSeries

IBM Developer Kit per Java

Versione 6





@server

iSeries

IBM Developer Kit per Java

Versione 6

Indice

Capitolo 1. IBM Developer Kit per Java	1
Novità in V5R2 per IBM Developer Kit per Java	2
Modifiche ad una versione specifica	3
Novità incluse il 26 Settembre 2002	3
Novità incluse dal 30 agosto 2002.	3
Come esaminare le novità o le modifiche	3
Novità in V5R2 per JDK (Java Development Kit) 1.1.8	3
Novità in V5R2 per J2SDK (Java 2 Software Development Kit), edizione standard, versione 1.4	4
Stampare questo argomento	4
Introduzione a IBM Developer Kit per Java	5
Installare IBM Developer Kit per Java	5
Installare un programma su licenza con il comando Ripristino programma su licenza	5
Supporto per più JDK (Java Development Kit)	6
Installare le estensioni per IBM Developer Kit per Java	7
Scaricare ed installare i pacchetti Java sul server iSeries	8
Eseguire il programma Java Hello World per la prima volta	9
Connettere un'unità di rete al server iSeries.	10
Creare un indirizzario sul server iSeries	11
Creare un indirizzario utilizzando la riga di immissione comandi	11
Creare un indirizzario utilizzando iSeries Navigator	12
Creare, compilare ed eseguire un programma HelloWorld Java	12
Creare e modificare i file origine Java	13
Con iSeries Access per Windows	14
Su una stazione di lavoro	14
Con EDTF	14
Con SEU (Source Entry Utility)	14
Gestire le applicazioni Java utilizzando iSeries Navigator	14
Personalizzare il proprio server iSeries per IBM Developer Kit per Java	15
Classpath Java	16
Proprietà di sistema Java	18
File SystemDefault.properties	18
Proprietà di sistema Java per JDK (Java Development Kit) 1.1.8	19
Valori della proprietà di sistema os400.stdio.convert e os400.child.stdio.convert	23
Valori della proprietà di sistema os400.stdin, os400.stdout e os400.stderr	23
Valori numerici os400.verify.checks.disable	23
Proprietà di sistema Java per J2SDK (Java 2 Software Development Kit), Edizione Standard	24
Creare un programma Java internazionalizzato	28
Variabile di ambiente del fuso orario sul proprio server iSeries	29
Configurare il fuso orario	29
Locali Java	40
Esempio: internazionalizzazione delle date utilizzando la classe java.util.DateFormat	42
Esempio: internazionalizzazione del pannello numerico utilizzando la classe java.util.NumberFormat	43
Esempio: internazionalizzazione di dati specifici della locale utilizzando la classe java.util.ResourceBundle	43
Codifiche del carattere Java	44
Valori file.encoding e CCSID iSeries	45
Valori file.encoding predefiniti	48
Compatibilità tra rilasci	50
Accedere all'IBM Developer Kit per Java	50
Accedere al proprio database iSeries con l'unità di controllo JDBC di IBM Developer Kit per Java	50
Introduzione a JDBC	52
Tipi di unità di controllo JDBC	53

Requisiti JDBC	54
Supporto didattico JDBC	55
Esempio: JDBC	56
Utilizzare JNDI per gli esempi	60
Collegamenti	61
DriverManager	62
Esempio: utilizzare JDBC nativo e JDBC di Toolbox contemporaneamente	64
Esempio: proprietà di accesso.	66
Esempio: ID utente e parola d'ordine non validi	69
Proprietà di collegamento	70
Esempio: creare UDBDataSource e collegarlo con JNDI	79
Esempio: creare UDBDataSourceBind e impostare le proprietà DataSource	79
Esempio: ottenere un contesto iniziale prima di collegare UDBDataSource	80
Esempio: creare UDBDataSource e ottenere un ID utente e una parola d'ordine	81
Utilizzare i DataSource con UDBDataSource	82
Proprietà DataSource	83
Altre implementazioni DataSource	89
Interfaccia DatabaseMetaData di IBM Developer Kit per Java	90
Creare un oggetto DatabaseMetaData	90
Reperire informazioni generali	90
Determinare il supporto della funzione	91
Limiti del sorgente dati	91
Oggetti SQL e relativi attributi	91
Supporto transazione	91
Modifiche in JDBC 3.0.	92
Esempio: interfaccia DatabaseMetaData di IBM Developer Kit per Java	92
Esempio: utilizzare i ResultSet di metadati che possiedono più di una colonna	92
Eccezioni	94
SQLException.	95
Esempio: SQLException	96
SQLWarning	97
DataTruncation	98
Troncamento non presidiato	100
Transazioni	100
Modalità di sincronizzazione automatica.	101
Livelli di isolamento della transazione	101
Punti di salvataggio	103
Transazioni distribuite	104
Transazioni con JTA	104
Utilizzare il supporto UDBXADDataSource per le transazioni distribuite e i lotti	105
Proprietà XADDataSource	105
ResultSet e transazioni	106
Multiplexing	106
Sincronizzazione a due fasi e registrazione della transazione	107
Esempio: utilizzare JTA per gestire una transazione	107
Esempio: più collegamenti che operano su una transazione	109
Esempio: utilizzare un collegamento con più transazioni	111
Esempio: ResultSet sospesi	113
Esempio: terminare una transazione	116
Esempio: sospendere e ripristinare una transazione	118
Tipi Statement	121
Statement.	122
Esempio: utilizzare il metodo executeUpdate dell'oggetto Statement	123
PreparedStatement	124
Elaborare PreparedStatement	126
Esempio: utilizzare PreparedStatement per ottenere un ResultSet	127

Esempio: ParameterMetaData	129
CallableStatement	130
Elaborare CallableStatement	132
Esempio: interfaccia CallableStatement per IBM Developer Kit per Java	133
Esempio: creare una procedura con più ResultSet	134
Esempio: creare una procedura con i parametri di immissione ed emissione	136
Esempio: creare una procedura con i valori di ritorno	137
ResultSet	138
Caratteristiche del ResultSet	138
Esempio: ResultSet sensibili e non sensibili	140
Esempio: sensibilità del ResultSet	142
Movimenti del cursore	144
Richiamare i dati del ResultSet	146
Modificare ResultSet	147
Esempio: eliminare i valori da una tabella tramite un altro cursore dell'istruzione	148
Esempio: modificare i valori con un'istruzione tramite un altro cursore dell'istruzione	150
Creare ResultSet	152
Esempio: interfaccia ResultSet di IBM Developer Kit per Java	152
Esempio: interfaccia ResultSetMetaData di IBM Developer Kit per Java	153
Lotto di oggetti JDBC	154
Utilizzare il supporto DataSource per la creazione di lotti di oggetti	155
Esempio: impostare il lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource.	156
Esempio: eseguire la verifica sulle prestazioni del lotto di collegamenti	157
Proprietà di ConnectionPoolDataSource.	157
Creare lotti di istruzioni basate su DataSource	159
Esempio: effettuare la verifica delle prestazioni di due DataSource	159
Creare il lotto di collegamenti	160
Aggiornamenti batch	163
Aggiornamento batch Statement	163
Aggiornamento batch PreparedStatement	164
BatchUpdateException	164
Supporto di inserimento bloccato	165
Tipi di dati avanzati	166
Tipi "distinct"	166
LOB (Large Object)	167
Tipi di dati SQL3 non supportati.	168
Scrivere il codice che utilizza i BLOB	168
Esempio: BLOB	168
Esempio: aggiornare i BLOB	169
Esempio: utilizzare i BLOB	170
Scrivere il codice che utilizza i CLOB	171
Esempio: CLOB	172
Esempio: aggiornare i CLOB	173
Esempio: utilizzare i CLOB	174
Scrivere il codice che utilizza i Datalink	175
Esempio: Datalink	175
Esempio: Tipi Distinct	177
RowSet	178
Caratteristiche del RowSet	178
DB2CachedRowSet	179
Utilizzare DB2CachedRowSet	180
Creare e popolare un DB2CachedRowSet	181
Accedere ai dati del DB2CachedRowSet e alla manipolazione del cursore	184
Modificare i dati DB2CachedRowSet e riflettere le modifiche riportate nel sorgente dati	188
Altre caratteristiche del DB2CachedRowSet	192

DB2JdbcRowSet	197
Eventi del DB2JdbcRowSet	199
Suggerimenti sulle prestazioni per l'unità di controllo JDBC di IBM Developer Kit per Java	201
Accedere ai database utilizzando il supporto DB2 SQLJ di IBM Developer Kit per Java	204
Strumenti SQLJ	204
Limitazioni DB2 SQLJ	204
Profili SQLJ (Structured Query Language for Java)	204
Il programma di conversione SQLJ (structured query language for Java) (sqlj)	205
Precompilare le istruzioni SQL in un profilo utilizzando il programma di personalizzazione del profilo DB2 SQLJ, db2profc	205
Stampare il contenuto dei profili DB2 SQLJ (db2profp e profp)	208
Programma di installazione del programma di controllo del profilo SQLJ (profdb).	209
Convertire un'istanza di profilo serializzato in un formato di classe Java utilizzando lo strumento di conversione del profilo SQLJ (profconv)	209
Incorporare le istruzioni SQL nell'applicazione Java	210
Variabili host in SQLJ (Structured Query Language for Java)	211
Esempio: incorporare le istruzioni SQL nell'applicazione Java	211
Compilare ed eseguire i programmi SQLJ	214
Routine SQL Java.	215
Utilizzare le routine SQL Java	216
Procedure memorizzate Java	217
Stile del parametro JAVA	217
Stile del parametro DB2GENERAL	219
Limitazioni sulle procedure memorizzate Java	221
Funzioni scalari Java definite dall'utente.	221
Stile del parametro Java	221
Stile del parametro DB2GENERAL	222
Limitazioni sulle funzioni Java definite dall'utente	225
Funzioni della tabella Java definite dall'utente	226
Procedure SQLJ che manipolano i file JAR	228
SQLJ.INSTALL_JAR	228
SQLJ.REMOVE_JAR	229
SQLJ.REPLACE_JAR	229
SQLJ.UPDATEJARINFO	230
SQLJ.RECOVERJAR	231
Convenzioni per passare un parametro per le UDF e le procedure memorizzate	232
Java con altri linguaggi di programmazione	232
Utilizzare JNI (Java Native Interface) per i metodi nativi	233
API di richiamo Java	235
Funzioni dell'API di richiamo	236
Supporto per più JVM (Java virtual machine)	238
Esempio: API di richiamo Java	238
Considerazioni sui sottoprocessi e i metodi nativi di Java	244
Metodi nativi e JNI (Java native interface)	245
Stringhe nei metodi nativi	245
Stringhe di costanti letterali nei metodi nativi	245
Convertire le stringhe dinamiche in e da EBCDIC, Unicode e UTF-8	246
Esempi: utilizzare JNI (Java Native Interface) per i metodi nativi.	246
Metodi nativi di PASE OS/400 IBM per Java	251
Variabili di ambiente PASE OS/400 Java.	252
Esempi: esempio di variabili di ambiente per PASE OS/400 IBM.	253
Utilizzare QIBM_JAVA_PASE_CHILD_STARTUP	253
Gestire le librerie di metodi nativi	254
Convenzioni di denominazione della libreria Java AIX e PASE OS/400	254
Ordine di ricerca della libreria Java	255
Codici di errore PASE OS/400 Java	256

Errori all'avvio	256
Errori al tempo di esecuzione	257
Esempio: metodo nativo PASE OS/400 IBM per Java.	257
Eseguire l'esempio del metodo nativo PASE OS/400 per Java	257
Confronto tra ILE (Integrated Language Environment) e Java	257
Utilizzare java.lang.Runtime.exec().	258
Esempio: Richiamare un altro programma Java con java.lang.Runtime.exec().	259
Esempio: chiamare un programma CL con java.lang.Runtime.exec()	260
Esempio: chiamare un comando CL con java.lang.Runtime.exec()	260
Comunicazioni tra processi	261
Utilizzare i socket per la comunicazione tra processi	261
Esempio: utilizzare i socket per la comunicazione tra processi	262
Utilizzare i flussi di immissione ed emissione per la comunicazione tra processi	265
Esempio: utilizzare i flussi di immissione ed emissione per la comunicazione tra processi	265
Esempio: Chiamare Java da C	266
Esempio: chiamare Java da RPG	266
Piattaforma Java	267
Applicazioni e applet Java	268
JVM (Java virtual machine)	268
JRE (Java runtime environment)	269
Interprete Java	269
File di classe e JAR Java	269
Sottoprocessi Java	270
JDK (Java Development Kit) di Sun Microsystems, Inc.	271
Pacchetti Java	271
Strumenti Java	272
Argomenti avanzati	272
Classi, pacchetti e indirizzari Java	273
File nell'IFS (integrated file system)	274
Autorizzazioni file Java nell'IFS (integrated file system)	274
Eseguire Java in un lavoro batch	275
Eseguire la propria applicazione Java su un host che non dispone di una GUI (graphical user interface)	276
AWT (Abstract Window Toolkit) Remoto di IBM Developer Kit per Java	276
Impostare AWT (Abstract Window Toolkit) Remoto per Java su un pannello remoto.	277
Rendere i file di classe di AWT Remoto (Abstract Window Toolkit) per Java accessibili al pannello remoto.	277
Aggiungere RAWTGui.zip o RAWTGui.jar al CLASSPATH del pannello remoto	278
Avviare AWT (Abstract Window Toolkit) Remoto per Java su un pannello remoto	279
Eseguire un programma Java utilizzando AWT (Abstract Window Toolkit) remoto	279
Eseguire un programma Java utilizzando AWT (Abstract Window Toolkit) remoto con Netscape	280
Stampare con AWT (Abstract Window Toolkit) Remoto	281
Proprietà di AWT (Abstract Window Toolkit) Remoto	281
Limitazioni di SecurityManager per AWT (Abstract Window Toolkit) Remoto.	282
Esempio: impostare AWT (Abstract Window Toolkit) Remoto per Java ^(TM) su un pannello remoto di Windows	282
CBJ (Class Broker for Java)	283
Impostare Class Broker for Java sul pannello remoto	283
Installare Class Broker per Java su un server iSeries.	283
Installare Class Broker per Java su Windows o UNIX.	284
Contenuto del pacchetto di cbj_1.1.jar	284
Native Abstract Windowing Toolkit	286
Installare NAWT	287
Installare PASE OS/400	287
Installare le PTF di NAWT.	287

Installare iSeries Tools for Developers PRPQ	287
Creare un file della parola d'ordine VNC	288
Configurare le proprietà di sistema Java	288
Avviare il server VNC	288
Impostare le variabili di ambiente	289
Verificare la procedura di installazione	289
Installare versioni meno recenti di iSeries Tools for Developers	289
Determinare se l'utente dispone di una versione di PRPQ aggiornata	289
Installare VNC	289
Suggerimenti sull'utilizzo di VNC	290
Avviare un server del pannello VNC da un programma CL	290
Arrestare un server del pannello VNC	290
Capitolo 2. Sicurezza Java	291
Modello di sicurezza Java	292
JCE (Java Cryptography Extension)	292
JSSE (Java Secure Socket Extension)	293
Utilizzare SSL (JSSE, versione 1.0.8)	294
Preparare il server iSeries per il supporto SSL (secure socket layer)	294
Fornitori ad accesso crittografico	295
Modificare il codice Java in modo da utilizzare le produzioni socket	295
Esempi: modificare il codice Java in modo da utilizzare le produzioni socket del server	296
Esempi: modificare il codice Java in modo da utilizzare le produzioni socket del client.	298
Modificare il codice Java in modo da utilizzare SSL (secure socket layer)	299
Esempi: modificare il server Java in modo da utilizzare SSL (secure socket layer)	299
Esempi: modificare il client Java in modo da utilizzare SSL (secure socket layer)	301
Selezionare un certificato digitale da utilizzare	303
Utilizzare il certificato digitale quando si esegue l'applicazione Java	303
Certificati digitali e proprietà -os400.certificateLabel	304
Contenitori certificati digitali e proprietà -os400.certificateContainer	304
Utilizzare JSSE (Java Secure Socket Extension), versione 1.4	304
Configurare il server iSeries per supportare JSSE	305
Requisiti software	305
Modificare i fornitori JSSE	305
Gestori della sicurezza	305
Fornitori JSSE	305
Fornitore JSSE Java puro	305
Fornitore JSSE nativo di iSeries	305
Modificare il fornitore JSSE predefinito	306
Proprietà della sicurezza JSSE	306
Proprietà di sistema JSSE Java.	307
Proprietà che si applicano ad entrambi i fornitori	307
Proprietà che si applicano solo al fornitore JSSE nativo di iSeries	308
Informazioni aggiuntive	308
Utilizzare il fornitore JSSE nativo di iSeries	308
Valori protocollo per il metodo SSLContext.getInstance	308
Implementazione KeyStore nativa di iSeries	309
Limitazioni di utilizzo del fornitore nativo di iSeries	309
Esempi: IBM JSSE (Java Secure Sockets Extension).	309
Esempio: client SSL che utilizza un oggetto SSLContext	310
Esempio: server SSL che utilizza un oggetto SSLContext	312
Capitolo 3. JAAS (Java Authentication and Authorization Service)	315
Preparare e configurare un server iSeries per JAAS (Java Authentication and Authorization Service)	315
Esempi di JAAS (Java Authentication and Authorization Service)	317

Capitolo 4. IBM JGSS (Java Generic Security Service)	319
Concetti su JGSS	320
Principal e credenziali	320
Come stabilire il contesto	321
Proteggere e scambiare messaggi	321
Rilasciare e ripulire risorse	321
Meccanismi di sicurezza	321
Configurare il proprio server iSeries per l'utilizzo di IBM JGSS	322
Configurare il proprio server iSeries per l'utilizzo di JGSS con J2SDK, versione 1.3	322
Requisiti software	322
Configurare il proprio server per l'utilizzo di JGSS	322
Modificare i fornitori JGSS	323
Gestori della sicurezza	323
Configurare JGSS per utilizzare il fornitore JGSS nativo di iSeries	323
Requisiti software	323
Specificare il fornitore JGSS nativo di iSeries	323
Configurare il server iSeries per utilizzare JGSS con J2SDK, versione 1.4	324
Modificare i fornitori JGSS	324
Gestori della sicurezza	324
Fornitore JGSS	324
Modificare il fornitore JGSS	325
Utilizzare il gestore della sicurezza	325
Autorizzazioni JVM	325
Controlli delle autorizzazioni JAAS	326
Controllo DelegationPermission	326
Controllo ServicePermission	327
Eseguire le applicazioni IBM JGSS	327
Ottenere le credenziali Kerberos e creare le chiavi segrete	328
Gli strumenti Kinit e Ktab	328
Utilizzare il fornitore JGSS Java puro	328
Utilizzare il fornitore JGSS nativo di iSeries	328
Interfaccia di collegamento a JAAS Kerberos	328
Autorizzazioni a JVM e JAAS	329
Opzioni del file di configurazione JAAS	329
Opzioni per il nome principal	330
Richiedere nome principal e parola d'ordine	330
Opzioni per il tipo di credenziale	331
File delle normative e di configurazione	331
File di configurazione Kerberos	331
File di configurazione JAAS	331
File normativa JAAS	331
File delle proprietà di sicurezza principale Java	332
Tabella chiavi del server e cache delle credenziali	332
Sviluppare le applicazioni IBM JGSS	333
Fasi di programmazione dell'applicazione IBM JGSS	334
Token di trasporto JGSS	334
Sequenza delle operazioni in un'applicazione JGSS	334
Creare un GSSManager	334
Creare un GSSName	335
Esempi: utilizzare GSSName	335
Creare un GSSCredential	335
Creare GSSContext	336
Richiedere servizi di sicurezza facoltativi	336
Come stabilire il contesto	337
Utilizzare i servizi per messaggi	338
Inviare messaggi	338

Ricevere i messaggi	339
Cancellare il contesto	339
Utilizzare JAAS con l'applicazione JGSS	340
Eseguire il debug	341
Classe di debug JGSS	341
Esempi: IBM JGSS (Java Generic Security Service)	342
Descrizione dei programmi di esempio	342
Visualizzare gli esempi IBM JGSS	342
Visualizzare i programmi di esempio	343
Visualizzare i file di configurazione e delle normative	343
Esempio: programma client IBM JGSS non-JAAS	343
Esempio: programma server IBM JGSS non-JAAS	351
Esempio: programma client IBM JGSS abilitato a JAAS	363
Esempio: programma server IBM JGSS abilitato a JAAS	364
Esempio: file di configurazione Kerberos	366
Esempio: file di configurazione collegamento JAAS	366
Esempio: file delle normative JAAS	367
Esempio: file delle normative Java	368
Esempi: scaricare e visualizzare informazioni javadoc per gli esempi di IBM JGSS	370
Esempi: scaricare ed eseguire i programmi di esempio	370
Esempi: scaricare gli esempi IBM JGSS	370
Informazioni correlate	371
Esempi: preparare l'esecuzione dei programmi di esempio	371
Informazioni correlate	371
Esempi: eseguire i programmi di esempio	371
Informazioni correlate	372
Informazioni di riferimento javadoc IBM JGSS	372

Capitolo 5. Ottimizzare le prestazioni del programma Java con IBM Developer Kit per Java	373
Considerazioni sulle prestazioni di Java runtime	373
Programmi di caricamento classi nella cache	374
Selezionare la modalità da utilizzare durante l'esecuzione di un programma Java	375
Interprete Java	378
Compilazione statica	378
Considerazioni sulle prestazioni della compilazione statica di Java	378
Compilatore JIT (Just-In-Time)	379
Confronto tra il compilatore JIT (Just-In-Time) e l'elaborazione diretta	379
Livelli di ottimizzazione	379
Raccolta di dati inutili Java	380
Raccolta di dati inutili avanzata di IBM Developer Kit per Java	381
Considerazioni sulle prestazioni della raccolta di dati inutili Java	381
Considerazioni sulle prestazioni di richiamo del metodo nativo Java	381
Considerazioni sulle prestazioni di allineamento del metodo Java	381
Considerazioni sulle prestazioni dell'eccezione Java	382
Strumenti delle prestazioni delle tracce di chiamata Java	382
Strumento delle prestazioni delle tracce di chiamata Java	382
Strumenti delle prestazioni di modellamento Java	382
JVMPi (Java Virtual Machine Profiler Interface)	383
Raccogliere dati delle prestazioni Java	383
Strumento Performance Data Collector	384
Strumento Java Performance Data Converter	384
Eseguire programma di conversione dei dati delle prestazioni Java	385
Esempio: eseguire Java Performance Data Converter	385

Capitolo 6. Comandi e strumenti per IBM Developer Kit per Java	387
Strumenti Java supportati da IBM Developer Kit per Java	387

Strumenti Java	388
Strumento ajar Java	388
Strumento appletviewer Java	388
Eseguire lo strumento appletviewer Java con Abstract Window Toolkit Remoto	388
Strumento extcheck Java	389
Strumento idlj Java	389
Strumento jar Java	389
Strumento jarsigner Java	389
Strumento javac Java	389
Strumento javadoc Java	390
Strumenti Java	390
Strumento javah Java	390
Strumento javakey Java	391
Strumento javap Java	391
Keytool Java	392
Strumento native2ascii Java	392
Policytool Java	392
Strumento rmic Java	392
Strumento rmid Java	392
Strumento rmiregistry Java	392
Strumento serialver Java	393
Strumento tnameserv Java	393
Comando Java in Qshell	393
Comandi CL supportati da Java	394
Comando ANZJVM (Analisi JVM - Java Virtual Machine)	394
Eseguire il comando ANZJVM	395
Forzare un ciclo di raccolta di dati inutili.	395
Considerazioni sul comando ANZJVM	395
Esempio: comando ANZJVM	396
File di emissione di spool per il comando ANZJVM.	396
Esempio: comando CHGJVAPGM (Modifica programma Java)	401
Stringhe del parametro LICOPT (Licensed Internal Code option).	401
Esempio: comando CRTJVAPGM (Creazione programma Java)	405
Esempio: comando DLTJVAPGM (Cancellazione programma Java)	406
Esempio: comando DMPJVM (Esecuzione dump JVM)	406
Esempio: comando DSPJVAPGM (Visualizzazione programma Java)	407
Comando JAVA.	408
Esempio: utilizzare comando RUNJVA (Esecuzione programma Java)	408
Comandi iSeries Navigator supportati da Java	408
Capitolo 7. Pacchetti facoltativi	411
JNDI (Java Naming and Directory Interface)	411
Guida di programmazione del fornitore IBM JNDI LDAP	412
Creare un contesto iniziale	413
URL LDAP V3	414
Collegamento al server e supporto SASL	415
Ricerca e ottenere attributi	417
Aggiungere e cancellare le voci in un indirizzario	419
Modificare attributi.	419
Ridenominare una voce dell'indirizzario	420
Riferimenti e riferimenti di ricerca	420
Controlli LDAP	421
Attributi binari	422
Schema	423
Modulo aggiuntivo SASL	425
Memorizzazione nella cache dal lato del client	426

Reperire la versione della classe IBMJNDI	428
Considerazioni di conformità e proprietà aggiuntive	428
JSSL	429
JavaMail	429
Java Print Service	430
Capitolo 8. Effettuare il debug dei programmi utilizzando IBM Developer Kit per Java	431
Effettuare il debug di un programma Java	431
Effettuare il debug dei programmi Java utilizzando l'opzione *DEBUG	432
Visualizzazioni di debug iniziali per i programmi Java	432
Impostare i punti di interruzione	433
Eseguire le istruzioni dei programmi Java da sottoporre a debug	434
Determinare il valore delle variabili nei programmi Java	435
Effettuare il debug dei programmi del metodo nativo e Java	436
Effettuare il debug di un programma Java da un altro pannello	436
Variabile di ambiente QIBM_CHILD_JOB_SNDINQMSG	437
Effettuare il debug delle classi Java caricate tramite un programma di caricamento classi personalizzato	438
Servlet di debug	438
Struttura del programma di debug della piattaforma Java	439
JVMDI (Java Virtual Machine Debug Interface)	439
JDWP (Java Debug Wire Protocol)	439
Avviare JDWP in QShell	439
Avviare JDWP da una riga comandi CL	439
JDI (Java Debug Interface)	440
Rilevare perdite di memoria	440
Capitolo 9. Risolvere i problemi di IBM Developer Kit per Java	441
Limiti	441
Rilevare le registrazioni lavori per un'analisi del problema Java	441
Raccogliere dati per un'analisi dei problemi Java	442
Acquisire il supporto di IBM Developer Kit per Java	443
Capitolo 10. Esempi di codice per IBM Developer Kit per Java	445
Capitolo 11. Riferimenti di IBM Developer Kit per Java	449
Informazioni sull'Esonero di responsabilità del codice	449

Capitolo 1. IBM Developer Kit per Java



IBM(R) Developer Kit per Java^(TM) è ottimizzato per l'utilizzo in ambiente server iSeries^(TM). Esso utilizza la compatibilità della programmazione Java e delle interfacce utente, in modo che sia possibile sviluppare le proprie applicazioni per il server iSeries.

IBM Developer Kit per Java consente di creare ed eseguire i programmi Java sul proprio server iSeries. IBM Developer Kit per Java è un'implementazione compatibile della tecnologia Java della Sun Microsystems, Inc. quindi si presume che l'utente abbia familiarità con la relativa documentazione JDK (Java Development Kit). Per facilitare la gestione delle loro e delle nostre informazioni, forniamo dei collegamenti alle informazioni di Sun Microsystems, Inc.

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

Technology java.sun.com  .

Selezionare uno qualsiasi di questi argomenti per ulteriori dettagli su come utilizzare IBM Developer Kit per Java:

- Stampare questo argomento fornisce dettagli su come scaricare un file PDF stampabile o un pacchetto compresso dei file HTML di IBM Developer Kit per Java.
- Novità in V5R2 evidenzia gli ultimi aggiornamenti sulle informazioni e sul prodotto.
- Introduzione fornisce informazioni sull'installazione, sulla configurazione e sulla modalità per creare ed eseguire semplici programmi Hello World Java, sullo scaricamento, sull'installazione e sulla compatibilità tra rilasci.
- Personalizzazione fornisce istruzioni su come personalizzare la configurazione del proprio fuso orario, delle proprietà di sistema e del classpath sul proprio server.
- Compatibilità fornisce informazioni sulla compatibilità dei file di classe Java da un rilascio all'altro.
- Accedere al proprio database spiega come IBM Developer Kit per Java consente ai programmi di accedere ai file di database iSeries.
- Java con altri linguaggi di programmazione mostra come chiamare un codice scritto in linguaggi diversi da Java utilizzando JNI (Java Native Interface), `java.lang.Runtime.exec()`, la comunicazione tra processi e l'API di richiamo Java.
- Piattaforma Java descrive l'ambiente per sviluppare e gestire le applet e le applicazioni Java e consiste nel linguaggio Java, nei pacchetti Java e nella JVM (Java virtual machine).
- Argomenti avanzati fornisce istruzioni su come eseguire Java in lavoro batch e descrive le autorizzazioni file Java necessarie nell'IFS (Integrated File System) per visualizzare, eseguire o effettuare il debug di un programma Java.
- Eseguire su un host che non dispone di una GUI contiene informazioni su come impostare ed eseguire i programmi Java con AWT (Abstract Window Toolkit) Remoto, Class Broker per Java o NAWT (Native Abstract Windowing Toolkit).
- Sicurezza fornisce dettagli sull'autorizzazione adottata e spiega come è possibile utilizzare SSL per rendere sicuri i flussi di socket nella propria applicazione Java.
- Prestazioni fornisce informazioni su come ottimizzare le prestazioni Java.

- Comandi e strumenti fornisce dettagli su come utilizzare i comandi e gli strumenti Java.
- Pacchetti facoltativi elenca pacchetti come JavaMail e JNDI (Java Naming and Directory Service) che è possibile utilizzare in modo facoltativo per sviluppare le proprie applicazioni Java.
- Effettuare il debug spiega come effettuare il debug dei programmi Java.
- Risolvere i problemi 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 sull'acquisizione del supporto per IBM Developer Kit per Java.
- Esempi di codice si collega direttamente a tutti gli esempi di codice in queste informazioni.
- Riferimento si collega direttamente a tutte le informazioni sul riferimento Javadoc e API.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Novità in V5R2 per IBM Developer Kit per Java

Questo argomento evidenzia le modifiche apportate alla versione V5R2 di IBM Developer Kit per Java^(TM). Sono state messe in rilievo le modifiche specifiche a JDK(Java Development Kit) 1.1.8 e J2SDK (Java 2 Software Development Kit), Edizione Standard, versione 1.4, separatamente. Gli aggiornamenti eseguiti dopo il rilascio generale di V5R2 appaiono in basso nel seguente elenco.

Introduzione

- Supporto di più JDK dispone di informazioni su ogni JDK supportato dall'IBM.
- Connettere un'unità di rete al proprio server iSeries e Creare un indirizzario sul proprio server iSeries sono stati spostati nell'Introduzione.
- Creare, compilare ed eseguire un programma Hello World Java contiene numerose modifiche.

Personalizzazione

- Esistono nuove Proprietà di sistema, incluse quelle relative ai Programmi di caricamento classi di memoria cache.

Accedere al database

- La sezione JDBC è stata sottoposta ad un'intensa revisione.
- Sono state aggiunte le sezioni Procedure memorizzate Java e Funzioni scalari Java definite dall'utente.

Eseguire su un host che non dispone di una GUI

- Consultare l'argomento Eseguire un programma Java utilizzando Abstract Window Toolkit Remoto per aggiornamenti.

Comandi e strumenti

- Il comando ANZJVM (Analisi JVM (Java Virtual Machine)) è stato aggiunto alla sezione dei comandi CL.
- E' stato aggiunto lo strumento idlj Java.
- Sono stati modificati i comandi di iSeries Navigator supportati da Java per rispecchiare numerose modifiche a iSeries Navigator.

Pacchetti facoltativi

- E' stata aggiunta la guida di programmazione del fornitore JNDI LDAP.

Effettuare il debug

- Esiste un nuovo supporto di debug per le classi Java caricate tramite un programma di caricamento classi personalizzato.

Esempi di codice

- Sono stati aggiunti ulteriori esempi di codice.

Stampare questo argomento

- Stampare questo argomento contiene un PDF delle informazioni su IBM Developer Kit per Java.

Riferimento

- La sezione Riferimento a IBM Developer Kit per Java è stata aggiunta e contiene informazioni sul riferimento API e Javadoc.

Modifiche ad una versione specifica

Fare clic sui collegamenti sottostanti per informazioni specifiche sulla versione di propria scelta:

- Java Development Kit, versione 1.1.8
- Java 2 Software Development Kit, Edizione standard, versione 1.4

Novità incluse il 26 Settembre 2002

Supporto per più JDK (Java Development Kit)

Questo aggiornamento tecnico chiarifica l'ordine di precedenza che OS/400^(R) utilizza per determinare il JDK predefinito quando si installano più JDK.

Chiamare i comandi CL da un programma Java

Questo aggiornamento tecnico aggiunge informazioni relative ai delimitatori richiesti da JDK quando vengono chiamati i comandi CL (control language) dai propri programmi Java.

Novità incluse dal 30 agosto 2002

NAWT (Native Abstract Windowing Toolkit)



Le modifiche apportate alle informazioni relative a NAWT (Native Abstract Windowing Toolkit) includono il supporto per JDK (Java Development Kit) versione 1.3 e gli aggiornamenti alle istruzioni per l'installazione di NAWT. Le istruzioni di installazione riflettono ora i miglioramenti apportati in PRPQ 5799-PTL, utilizzato da NAWT.

JGSS (Java Generic Security Service)

JGSS (Java Generic Security Service) è un nuovo argomento che fornisce le informazioni relative a IBM JGSS, un'interfaccia generica per autenticare e rendere sicuri i messaggi. A questa interfaccia è possibile collegare una varietà di meccanismi di sicurezza basati su chiavi segrete, chiavi pubbliche o altre tecnologie di sicurezza.

Come esaminare le novità o le modifiche

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

- L'immagine  per contrassegnare dove iniziano le informazioni nuove o modificate.
- L'immagine  per contrassegnare dove terminano le informazioni nuove o modificate.

Per reperire altre informazioni sulle novità o le modifiche in questo rilascio, consultare la Memo per gli

utenti .

Novità in V5R2 per JDK (Java Development Kit) 1.1.8

Non esistono modifiche nella V5R2 specifiche per JDK (Java^(TM) Development Kit) 1.1.8.

Novità in V5R2 per J2SDK (Java 2 Software Development Kit), edizione standard, versione 1.4

In questo argomento vengono evidenziate le modifiche apportate alla V5R2 di IBM Developer Kit per Java[™] per J2SDK (Java 2 Software Development Kit), edizione standard, versione 1.4.

Nota: in questa sezione, si discutono soltanto le modifiche univoche o interessanti in J2SDK, versione 1.4. Le informazioni generali sugli aggiornamenti dall'argomento Novità per JDK (Java Development Kit) 1.1.8 si applicano anche a J2SDK, versione 1.4.

Personalizzazione

- La variabile di ambiente del fuso orario mostra aspetti univoci dell'impostazione della variabile del fuso orario su un server iSeries. E' necessario effettuare ciò per utilizzare **getSystemTimeZoneID()** nativo, che fa parte di J2SDK (Java 2 SDK), Edizione Standard, versione 1.3 e successive.
- Sono state aggiunte delle nuove proprietà di sistema per J2SDK.

Sicurezza

- JAAS (Java Authentication and Authorization Service) è un'estensione standard a JDK 1.3 (Java 2 Software Development Kit, v 1.3) e successive. Correntemente, Java 2 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 incrementa il modello di sicurezza Java 2 con questo supporto.
- JCI (Java Cryptography Extension) 1.2 è un'estensione standard a J2SDK (Java 2 Software Development Kit), Edizione Standard. L'implementazione di JCE su un server iSeries è compatibile con l'implementazione di Sun Microsystems, Inc. Questa documentazione tratta gli aspetti univoci dell'implementazione iSeries. Si presume una familiarità dell'utente con tale materiale generale relativa alle estensioni JCE.

Pacchetti facoltativi

- L'API Servizio di stampa Java consente di stampare su tutte le piattaforme Java. Gli ambienti del tempo di esecuzione Java e terzi possono fornire i moduli aggiuntivi del generatore motore per produrre vari formati per stampare, come PDF, Postscript e AFP.
- L'API JavaMail è una serie di classi astratte che modellano un sistema elettronico (e-mail). L'API fornisce una framework indipendente dalla piattaforma e dal protocollo per creare applicazioni di messaggeria e di e-mail basate su Java.

Stampare questo argomento


Per visualizzare o scaricare la versione PDF, selezionare IBM Developer Kit per Java[™] (circa 2159 KB o 450 pagine).

Salvataggio dei file PDF

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

1. Fare clic con il tastino destro del mouse su PDF nel browser (fare clic con il tastino destro del mouse sul collegamento precedente).
2. Fare clic su **Salva con nome...**
3. Portarsi sull'indirizzario in cui si desidera salvare il PDF.
4. Fare clic su **Salva**.

Scaricamento di Adobe Acrobat Reader

Se si desidera utilizzare Adobe Acrobat Reader per visualizzare o stampare questi PDF, è possibile scaricare una copia dal sito Web Adobe (www.adobe.com/products/acrobat/readstep.html) .

Introduzione a IBM Developer Kit per Java

Se l'utente non ha ancora utilizzato IBM Developer Kit per Java^(TM), è necessario seguire queste fasi per installarlo, configurarlo ed eseguire un semplice programma Hello World Java.

1. Se l'utente ha già familiarità con le informazioni di IBM Developer Kit per Java, consultare novità per collegamenti agli ultimi aggiornamenti del prodotto e informazioni.
2. Installare IBM Developer Kit per Java.
3. Configurare il proprio sistema.
4. Se l'utente non conosce ancora queste informazioni e non ha ancora utilizzato IBM Developer Kit per Java, consultare Esegui il programma Hello World Java per la prima volta. Questo argomento illustra due metodi di eseguire un semplice programma Hello World Java con IBM Developer Kit per Java. E' un modo conveniente per verificare se è stato installato correttamente IBM Developer Kit per Java.
5. A questo punto si è pronti a creare, compilare ed eseguire il proprio programma Hello World Java. Per le fasi da seguire, consultare Creare, compilare ed eseguire un programma Hello World Java.
6. Se si è interessati a creare più applicazioni Java rispetto alle proprie, consultare questi argomenti:
 - Creare e modificare i file origine Java mostra tre modi differenti tramite cui è possibile creare e modificare i propri file origine Java.
 - Scaricare e installare pacchetti Java su un server iSeries fornisce assistenza nell'utilizzare i pacchetti Java in maniera più efficace. Esso fornisce dettagli sui pacchetti con GUI (graphical user interface), IFS (Integrated File System) e sensibilità al maiuscolo e minuscolo, così come sulla gestione dei file ZIP e dei file JAR.
 - Compatibilità tra rilasci fornisce informazioni sulla compatibilità da un rilascio ad un altro.

Installare IBM Developer Kit per Java

L'installazione di IBM Developer Kit per Java^(TM) consente di creare ed eseguire programmi Java sul proprio server iSeries.

Per installare IBM Developer Kit per Java, effettuare quanto segue:

1. Immettere il comando GO LICPGM (Gestione programmi su licenza) 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) 5722-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 nella lista, è possibile aggiungerla immettendo l'opzione 1 (Installazione) nel campo Opzione. Immettere 5722JV1 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.

Una volta installato IBM Developer Kit per Java sul proprio server iSeries, è possibile scegliere di personalizzare il proprio sistema.

Consultare Esegui il programma Hello World Java per la prima volta per informazioni preliminari su IBM Developer Kit per Java.

Installare 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 iSeries, 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 installando da un'unità nastro, l'ID unità è solitamente nel formato **TAPXX**, dove **XX** è un numero, come **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 JDK (Java Development Kit)

Il server iSeries supporta più JDK (Java Development Kit) e J2SDK (Java 2 SDK), Edizione Standard. Il server iSeries supporta l'utilizzo di più JDK simultaneamente, ma solo tramite più Java virtual machine. Una Java virtual machine singola esegue un JDK specificato.

Individuare il JDK che si sta utilizzando o che si desidera utilizzare e selezionare l'opzione corrispondente da installare. E' possibile installare più di un JDK alla volta. La proprietà di sistema `java.version` determina quale JDK eseguire. Quando una Java virtual machine è attiva e in esecuzione, la modifica della proprietà di sistema `java.version` non ha alcun effetto.

Nota: in V5R2, le opzioni 1 (JDK 1.1.6) e 2 (JDK 1.1.7) non sono più disponibili. Queste non possono essere installate o utilizzate.

Opzione	JDK	java.home	java.version
3	1.2	» /QIBM/ProdData/Java400/jdk12/ «	1.2
4	1.1.8	/QIBM/ProdData/Java400/jdk118/	1.1.8
5	1.3	/QIBM/ProdData/Java400/jdk13/	1.3
» 6	1.4	/QIBM/ProdData/Java400/jdk14/	1.4 «

» **Nota:** la versione 1.3 risulta uguale a J2SDK (Java 2 SDK), Edizione Standard, versione 1.3. «

Ad esempio, di seguito vengono riportati i risultati a seconda delle opzioni installate e del comando immesso.

Installazione	Immissione	Risultato
Opzione 3 (1.2)	java Hello	J2SDK, Edizione standard, versione 1.2 in esecuzione.

Opzione 4 (1.1.8)	java Hello	JDK 1.1.8 in esecuzione, perché solo un JDK è installato e questo risulta essere il proprio valore predefinito.
Opzione 4 (1.1.8) e Opzione 3 (1.2)	java Hello	J2SDK, Edizione standard, versione 1.2 in esecuzione perché è successiva.
» Tutte e quattro le opzioni installate	java Hello	J2SDK, Edizione Standard, versione 1.3 in esecuzione. «
Opzione 3 (1.2) e Opzione 5 (1.3)	java Hello	J2SDK, Edizione Standard, versione 1.3 in esecuzione perché è successiva.
Opzione 4 (1.1.8) e Opzione 5 (1.3)	java -Djava.version=1.1.8 Hello	JDK 1.1.8 in esecuzione, perché è stata specificata.
» Opzione 5 (1.3) e Opzione 6 (1.4)	java Hello	J2SDK, Edizione Standard, versione 1.3 in esecuzione. Anche se la versione 1.4 risulta successiva in quanto a numero, 1.3 ha la precedenza. «

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 5 (1.3)
2. Opzione 3 (1.2)
3. Opzione 6 (1.4)
4. Opzione 4 (1.1.8) «

Installare le estensioni per IBM Developer Kit per Java

Le estensioni sono pacchetti delle classi JavaTM 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 VM utilizza le classi di sistema. Il meccanismo di estensione fornisce inoltre un modo affinché l'utente reperisca le estensioni dagli URL (Uniform Resource Locator) specificati quando non sono già installati in J2SDK, versione 1.2 o successive o Java 2 Runtime Environment, Edizione Standard, versione 1.2 e successive.

» Alcuni file JAR per estensioni vengono forniti con il server iSeries. « Se si intende 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`, la lista dei file in cui il programma di caricamento classi di estensioni effettua la ricerca, viene modificata per *ogni Java virtual machine in esecuzione sul proprio server iSeries*.

Se non si desidera influenzare i programmi di caricamento classi di estensioni per altre VM sul proprio server iSeries, ma si desidera ancora creare un collegamento ad un'estensione o installare un'estensione non fornita dall'IBM con il server iSeries, seguire queste fasi:

1. Creare un indirizzario per installare le estensioni.
Utilizzare il comando MKDIR (Creazione indirizzario) dalla riga comandi iSeries 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`.
E' possibile aggiungere il nuovo indirizzario alla proprietà `java.ext.dirs` utilizzando il campo PROP del comando JAVA dalla riga comandi iSeries.

» Se il nome del nuovo indirizzario è `/home/username/ext`, il nome del proprio file di estensione è `extensionToInstall.jar` e il nome del proprio programma Java è `Hello`, i comandi che si immettono dovrebbero risultare in questo modo:

```
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'))
```

Scaricare ed installare i pacchetti Java sul server iSeries

Per scaricare, installare e utilizzare i pacchetti Java^(TM) in modo più efficace su un server iSeries, esaminare quanto segue:

- Pacchetti con GUI (Graphical User Interface)
- Sensibilità al maiuscolo e minuscolo e IFS (Integrated File System)
- Gestione file ZIP e gestione file JAR
- Framework delle estensioni Java

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 grafiche. Ad esempio, è possibile utilizzare un PC, una stazione di lavoro tecnica o un computer di rete. Il server iSeries fornisce una capacità AWT (Abstract Window Toolkit) Remoto. Tale capacità esegue le applicazioni su un server iSeries utilizzando un intero intervallo di capacità grafiche su un'unità di visualizzazione collegata al TCP/IP (Transmission Control Protocol/Internet Protocol) appropriato. Per informazioni più specifiche su installazione, impostazione e utilizzo globale, consultare Impostare AWT (Abstract Window Toolkit) Remoto.

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 sull'IFS, consultare le informazioni File System nell'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 sempre un linguaggio sensibile al maiuscolo e al minuscolo. Anche se `wrklnk '/home/Hello.class'` e `wrklnk '/home/hello.class'` producono gli stessi risultati, `JAVA CLASS(Hello)` e `JAVA CLASS(hello)` chiamano differenti classi. «

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, convertite in formato macchina interno e, se specificato, trasformate in codice macchina iSeries. E' 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 l'utente non ha la certezza se un programma Java corrente sia associato al proprio file di classe o ad un file JAR, è necessario utilizzare il comando DSPJVAPGM (Visualizzazione programma Java) per visualizzare informazioni sul programma Java sul proprio server iSeries.

Nei precedenti rilasci di IBM Developer Kit per Java, è stato 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 ajar.
- Si modifica o si ricrea un file JAR utilizzando lo strumento jar.
- Si sostituisce un file JAR utilizzando il comando COPY OS/400 o il programma di utilità cp Qshell.

Se si accede ad un file JAR nell'IFS (Integrated File System) tramite iSeries Access per Windows o da un'unità connessa ad un PC, 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.

Consultare Prestazioni del tempo di esecuzione Java per informazioni più dettagliate sulle caratteristiche delle prestazioni dei file JAR.

Framework delle estensioni Java

In Java 2 SDK, Edizione Standard, versione 1.2 e successive, le estensioni sono pacchetti delle 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 VM utilizza le classi di sistema. Il meccanismo di estensione fornisce inoltre un modo affinché l'utente reperisca le estensioni dagli URL specificati quando non sono già installati nel JDK (Java Development Kit) o Java 2 Runtime Environment, Edizione Standard.

Consultare Installare le estensioni per IBM Developer Kit per Java per informazioni sull'installazione delle estensioni.

Eeguire il programma Java Hello World per la prima volta

E' possibile ottenere il programma Hello World Java^(TM) e procedere all'esecuzione in uno dei modi seguenti:

1. E' possibile semplicemente eseguire 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 5722-JV1 *BASE e almeno una delle opzioni siano elencate come installate.
 - b. Immettere java Hello sulla riga comandi del Menu principale iSeries. Premere Invio per eseguire il programma Hello World Java.
 - c. Se IBM Developer Kit per Java è stato installato correttamente, Hello World compare 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 è in esecuzione, controllare per assicurarsi che l'installazione sia stata completata con esito positivo oppure consultare Acquisire il supporto per IBM Developer Kit per Java per informazioni sul servizio.
2. E' possibile inoltre eseguire il proprio programma Hello di Java. Per ulteriori informazioni sul modo in cui creare il proprio programma Hello di Java, consultare Creare, compilare ed eseguire un programma Hello World Java.

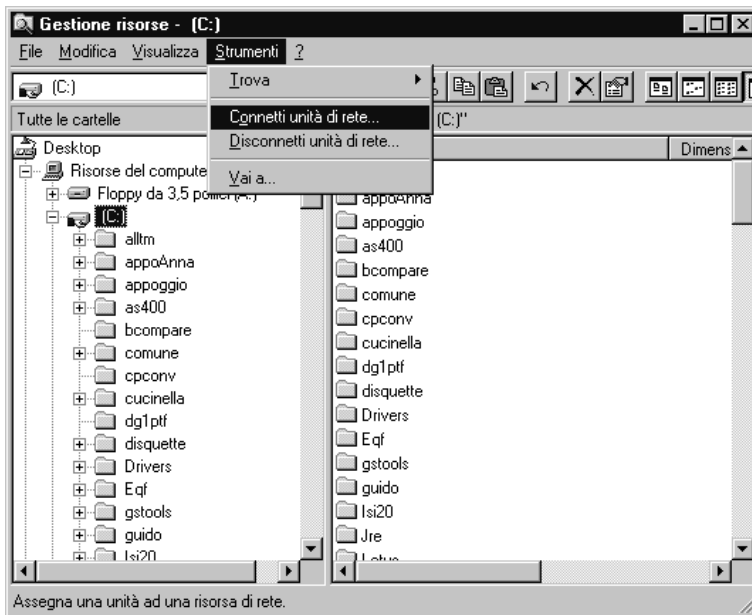
Connettere un'unità di rete al server iSeries

» Per connettere un'unità di rete al proprio server iSeries, assicurarsi che si abbia iSeries Access per Windows installato sul proprio server e sulla propria stazione di lavoro. Per ulteriori informazioni su come installare e configurare iSeries Access per Windows, consultare Installare iSeries Access per Windows.

E' necessario disporre di un collegamento configurato per il server iSeries prima che sia possibile connettere un'unità di rete.

Per connettere un'unità di rete, seguire queste fasi:

1. Aprire Gestione Risorse di Windows^(R):
 - a. Fare clic con il tastino destro del mouse sul pulsante **Avvia** sulla barra delle applicazioni Windows.
 - b. Fare clic su **Esplora** nel menu.
2. Selezionare **Connetti unità di rete** dal menu **Strumenti**.

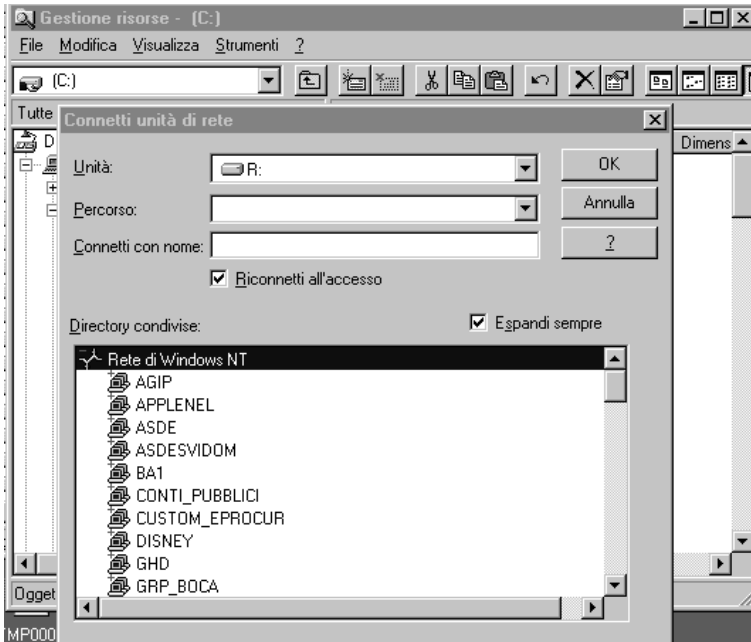


3. Selezionare l'unità che si intende utilizzare per collegarsi al proprio server iSeries.

4. Immettere il nome percorso sul proprio server. Ad esempio:

\\MYSERVER

dove **MYSERVER** è il nome del proprio server iSeries.



5. Controllare la casella **Riconnetti all'accesso** se è vuota.
6. Fare clic su **OK** per terminare.

La propria unità connessa viene ora visualizzata nella sezione **Tutte le cartelle** di Windows Explorer. <<

Creare un indirizzario sul server iSeries

>> E' necessario creare un indirizzario sul proprio server iSeries dove è possibile salvare le proprie applicazioni Java^(TM). Esistono due modi per effettuare ciò:

- Creare un indirizzario utilizzando iSeries Navigator
Scegliere questa opzione se si dispone di iSeries Access per Windows installato. Se si ha intenzione di utilizzare iSeries Navigator per compilare, ottimizzare ed eseguire il proprio programma Java, è necessario selezionare questa opzione per assicurarsi che il proprio programma venga salvato nella corretta ubicazione per eseguire queste operazioni.
- Creare un indirizzario utilizzando la riga di immissione comandi
Scegliere questa opzione se non si dispone di iSeries Access per Windows installato.

Per informazioni su iSeries Navigator, incluse le informazioni di installazione, consultare Introduzione ad iSeries Navigator. <<

Creare un indirizzario utilizzando la riga di immissione comandi

Per creare un indirizzario sul proprio server iSeries, seguire queste fasi:

1. Collegarsi al proprio server iSeries.
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.**"

Creare un indirizzario utilizzando iSeries Navigator

» Per creare un indirizzario sul proprio server iSeries, effettuare quanto segue:

1. Aprire iSeries Navigator.
2. Fare doppio clic sul nome del proprio server nella finestra **My Connections** per collegarsi. Se il proprio server non è elencato nella finestra **My Connections**, effettuare quanto segue per aggiungerlo:
 - a. Fare clic su **File** → **Add Connection...**
 - b. Immettere il nome del proprio server nel campo **System**.
 - c. Fare clic su **Next**.
 - d. Se non è già stato immesso, immettere il proprio ID utente nel campo **Use default user ID, prompt as needed**.
 - e. Fare clic su **Next**.
 - f. Fare clic su **Verify Connection**. Ciò conferma che è possibile collegarsi al server.
 - g. Fare clic su **Finish**.
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 iSeries Navigator. E' necessario installare tale opzione selezionando **Avvio** → **Programmi** → **IBM iSeries 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 iSeries.
6. Fare clic con il tastino destro del mouse sulla cartella dove si desidera aggiungere un sottoindirizzario. Selezionare **Nuova Cartella** e immettere il nome del sottoindirizzario che si intende creare.◀◀

Creare, compilare ed eseguire un programma HelloWorld Java

La creazione del semplice programma, Hello World Java^(TM), è un ottimo punto di inizio una volta acquistata familiarità con IBM Developer Kit per Java.

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

1. » Connettere un'unità di rete al proprio server iSeries.
2. Creare un indirizzario sul proprio server iSeries per le proprie applicazioni Java.◀◀
3. Creare il file origine come file di testo ASCII (American Standard Code Information Interchange) nell'IFS (Integrated File System). » E' inoltre possibile utilizzare un prodotto IDE (integrated development environment) o un editor di testo come ad esempio Notepad di Windows^(R) per codificare la propria applicazione Java. ◀◀
 - a. Denominare il proprio file di testo HelloWorld.java. Per ulteriori informazioni su come creare e modificare il proprio file, consultare Creare e modificare i file origine Java.
 - b. Assicurarsi che il proprio file contenga questo codice sorgente:

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

4. Compilare il file origine.

- a. Immettere il comando WRKENVVAR (Gestione variabile di ambiente) per controllare la variabile di ambiente CLASSPATH. Se la variabile CLASSPATH non esiste, aggiungerla e impostarla su '.' (l'indirizzario corrente). Se la variabile CLASSPATH esiste, assicurarsi che il '.' si trovi all'inizio della lista dei nomi del percorso. Per dettagli sulla variabile di ambiente CLASSPATH, consultare Classpath Java.
 - b. Immettere il comando STRQSH (Avvio Qshell) per avviare Qshell Interpreter.
 - c. Utilizzare il comando cd (Modifica indirizzario) per modificare l'indirizzario corrente nell'indirizzario dell'IFS (Integrated File System) che contiene il file HelloWorld.java.
 - d. Immettere javac seguito dal nome del file 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. Ottimizzare l'applicazione Java.

- a. Sulla riga *Immissione comandi QSH*, immettere:

```
system "CRTJVAPGM '/mydir/myclass.class' OPTIMIZE(20)"
```

dove *mydir* è il nome del percorso dell'indirizzario in cui viene salvata la propria applicazione Java e dove *myclass* è il nome della propria applicazione Java compilata.

Nota: è possibile specificare un livello di ottimizzazione che raggiunga massimo 40. Un livello di ottimizzazione di 40 aumenta l'efficienza dell'applicazione Java, ma limita anche le funzioni di debug. Nelle prime fasi dello sviluppo di un'applicazione Java, è possibile impostare il livello di ottimizzazione su 20 cosicché è possibile eseguire più facilmente il debug dell'applicazione. Consultare il comando CRTJVAPGM e il parametro OPTIMIZE per ulteriori informazioni.

- b. Premere il tasto **Invio**.

Viene visualizzato un messaggio, che attesta che è stato creato un programma Java per la propria classe. ⏪

7. Eseguire il file di classe.
- a. Assicurarsi che il proprio Classpath Java sia impostato correttamente.
 - b. Sulla riga comandi Qshell, immettere java seguito da HelloWorld per eseguire il proprio HelloWorld.class con Java virtual machine. Ad esempio, immettere java HelloWorld. E' inoltre possibile utilizzare il comando RUNJVA (Esecuzione Java) sul proprio server iSeries per eseguire HelloWorld.class.
 - c. "Hello World" viene visualizzato sullo schermo se tutto è stato immesso in maniera corretta. ➤ Viene visualizzata la richiesta shell (per impostazione predefinita, un \$), ad indicare che Qshell è pronto per un altro comando. ⏪
 - d. Premere F3 (Fine) o F12 (Scollegamento) per ritornare al pannello di immissione comandi.

➤ E' inoltre possibile compilare, ottimizzare ed eseguire con facilità la propria applicazione Java utilizzando iSeries Navigator, una GUI per eseguire attività sul proprio server iSeries. Per istruzioni, consultare Gestione delle applicazioni Java utilizzando iSeries Navigator. Per ulteriori informazioni su iSeries Navigator, incluse le informazioni di installazione, consultare Introduzione ad iSeries Navigator. ⏪

Creare e modificare i file origine Java

E' possibile creare e modificare i file origine Java^(TM) in vari modi:

- Con iSeries Access per Windows.
- Su una stazione di lavoro.
- Con EDTF.
- Con SEU (Source Entry Utility).

Con iSeries Access per Windows

I file origine Java sono file di testo ASCII (American Standard Code for Information Interchange) nell'IFS (Integrated File System) sui server iSeries.

E' possibile creare e modificare un file origine Java con iSeries Access per Windows e un editor basato su una stazione di lavoro.

Su una stazione di lavoro

E' possibile creare un file 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 origine Java su una stazione di lavoro:

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

Con EDTF

E' possibile modificare i file da qualsiasi file system utilizzando il comando CL EDTF. Questo è un editor simile al SEU (Source Entry Utility) per modificare i file del flusso o i file del database. Consultare il comando CL EDTF per informazioni.

Con SEU (Source Entry Utility)

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

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

1. Creare un membro del file origine utilizzando SEU.
2. Utilizzare il comando CPYTOSTMF (Copia nel file di flusso) per copiare il membro del file 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 informazioni sulla memorizzazione dei file, consultare File nell'IFS (Integrated file system).

Gestire le applicazioni Java utilizzando iSeries Navigator

iSeries Navigator consente di compilare, ottimizzare ed eseguire le applicazioni Java^(TM) semplicemente puntando e facendo clic.

Requisiti di impostazione

Per utilizzare iSeries Navigator in modo tale da gestire le applicazioni Java, assicurarsi di rispettare le seguenti condizioni:

- E' necessario che iSeries Navigator, che è una parte di iSeries Access per Windows, sia installato sulla propria stazione di lavoro. Se iSeries Access per Windows non è ancora installato, consultare Introduzione a iSeries Navigator per scaricare le informazioni.
- E' necessario salvare l'applicazione Java per un indirizzario specifico sul proprio server iSeries. Il percorso corretto per il salvataggio dell'applicazione Java è il seguente:

myserver -> File Systems -> Integrated File System -> Root -> home -> mydir

Laddove **myserver** indica il nome del server iSeries e **mydir** indica il nome dell'indirizzario nel quale le applicazioni Java sono state salvate. Per ulteriori informazioni dettagliate sulla creazione di un indirizzario nel quale salvare l'applicazione Java, consultare Creare un indirizzario utilizzando iSeries Navigator.

Compilare l'applicazione Java utilizzando iSeries Navigator



Per compilare l'applicazione Java, seguire le fasi indicate:

1. Fare clic con il tasto destro del mouse su **myfile.java**, dove **myfile** indica il nome dell'applicazione Java.
2. Selezionare **Compila file Java**.
3. Nella nuova finestra, selezionare la versione JDK. Se precedentemente è stato impostato un classpath secondo le istruzioni fornite in questa guida, non è necessario specificare un classpath in questa finestra.
4. Fare clic su **OK**.

Si apre una finestra contenente una lista di errori se viene rilevato qualche errore nel programma. Altrimenti un messaggio indicante che **1 di 1 file Java sono stati compilati** compare nella parte inferiore della finestra di iSeries Navigator. Viene creato un nuovo file denominato **myfile.class**.

Ottimizzare ed eseguire l'applicazione Java utilizzando iSeries Navigator

Per ottimizzare ed eseguire l'applicazione Java, seguire le fasi indicate:

1. Fare clic con il tasto destro del mouse su **myfile.class**.
2.  Selezionare **Programma Java associato**, successivamente fare clic su **Esegui...** per eseguire il programma Java. 
3. Fare clic su **Avanzate**. Selezionare il livello di ottimizzazione desiderato.

Nota: è possibile specificare un livello di ottimizzazione che raggiunga massimo 40. Un livello di ottimizzazione di 40 aumenta l'efficienza dell'applicazione Java, ma limita anche le funzioni di debug. Nelle prime fasi dello sviluppo di un'applicazione Java, è possibile impostare il livello di ottimizzazione su 20 cosicché è possibile eseguire più facilmente il debug dell'applicazione. Per ulteriori informazioni sull'ottimizzazione, consultare Livelli di ottimizzazione. Esiste anche un'opzione denominata JIT, che sta per compilatore just-in-time. I JIT compilano il codice quando questo è necessario; questo li rende più efficaci dell'elaborazione diretta. Per ulteriori informazioni sui JIT, consultare Selezionare quale modalità utilizzare quando si esegue un programma Java.

4. Fare clic su **OK** per chiudere la finestra Opzioni avanzate.
5. Fare clic su **OK** per eseguire il programma Java.

In una nuova finestra viene visualizzata l'emissione da parte del programma. Quando l'esecuzione del programma viene terminata, compare un messaggio indicante **programma Java completato**.

Per ulteriori informazioni sulle funzioni di iSeries Navigator, consultare il menu **Aiuto** sul pannello iSeries Navigator.

Personalizzare il proprio server iSeries per IBM Developer Kit per Java

Una volta installato IBM Developer Kit per Java^(TM) sul proprio server iSeries, è possibile scegliere di personalizzare il proprio server.

Configurazione del fuso orario

Se i propri programmi Java sono sensibili all'ora del giorno, è possibile che venga richiesta la configurazione del fuso orario.

Se il valore QUTCOFFSET (Coordinated Universal Time Offset) è impostato sul valore predefinito (+00:00), Java utilizza iSeries 400time come ora corrente. Il valore predefinito dell'impostazione della proprietà di sistema Java user.timezone è UTC.

Sono richiesti un valore di sistema QUTCOFFSET e una locale aggiornata per ognuna di queste opzioni:

- Se QUTCOFFSET non è impostato sul relativo valore predefinito, quindi sensibile al fuso orario.
- Se il proprio codice Java prevede la proprietà di sistema user.timezone su un valore predefinito diverso da UTC.
- Se si specifica la proprietà di sistema Java user.timezone quando si esegue il comando java.

La categoria LC_TOD della locale contiene il campo tname, che deve essere impostato sullo stesso valore corrispondente al proprio fuso orario. Per dettagli su come creare una locale e formattare il campo tname, consultare Globalizzazione OS/400.

Configurazione della proprietà di sistema

Le proprietà di sistema Java determinano l'ambiente eseguire il programma Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in OS/400. Vengono impostate alcune proprietà all'avvio della JVM (Java virtual machine). E' possibile scegliere di utilizzare i valori predefiniti della proprietà di sistema o è possibile specificare la propria impostazione di tali valori eseguendo una di queste fasi:

1. Fornire un file in /QIBM/UserData/Java400, denominato SystemDefault.properties. I valori della proprietà specificati in questo file sovrascrivono i valori predefiniti forniti dall'IBM. Questo file imposta le proprietà di sistema predefinite per tutte le JVM (Java virtual machine) in esecuzione sul proprio server iSeries.
2. Oppure inserire un file SystemDefault.properties nel proprio percorso user.home. Questo file e le proprietà che contiene sovrascrivono quelle in /QIBM/UserData/Java400/SystemDefault.properties. Questo file imposta le proprietà di sistema predefinite per tutte le JVM (Java virtual machine) in esecuzione per l'utente.

Consultare Eseguire il programma Hello World Java per la prima volta per informazioni preliminari su IBM Developer Kit per Java.

Classpath Java

Java^(TM) virtual machine utilizza il classpath Java per trovare le classi durante il tempo di esecuzione. Gli strumenti e i comandi Java utilizzano inoltre il classpath per localizzare le classi. Il classpath di sistema predefinito, la variabile di ambiente CLASSPATH e il parametro del comando di classpath determinano quali indirizzari si ricercano quando si cerca una classe particolare.

Nota: in J2SDK (Java 2 Software Development Kit), Edizione Standard, versione 1.2 e successive, la proprietà java.ext.dirs determina il classpath per le estensioni caricate. Consultare Installare le estensioni per IBM Developer Kit per Java per ulteriori informazioni.

Il classpath del sistema predefinito viene definito dal sistema e non è possibile modificarlo. Sul proprio server iSeries, il classpath predefinito specifica dove trovare le classi che fanno parte di IBM Developer Kit, l'AWT (Abstract Window Toolkit) Remoto e altre classi del sistema.

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

E' 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 una lista di nomi percorso, separati da due punti (:), che si ricercano per trovare una classe particolare. 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.

E' possibile impostare la variabile CLASSPATH nell'ambiente Qshell utilizzando il programma di utilità di esportazione disponibile usando 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")
```

J2SDK ricerca le classi in maniera differente rispetto a JDK 1.1.x. JDK 1.1.x ricerca prima il classpath del sistema, quindi ogni classpath specificato dall'utente. J2SDK ricerca prima il classpath del bootstrap, quindi gli indirizzari di estensione, quindi il classpath.

Di conseguenza, l'ordine di ricerca per JDK 1.1.x, utilizzando il precedente codice di esempio, è:

1. Il classpath del sistema predefinito,
2. L'indirizzario di lavoro corrente,
3. Il file myclasses.zip ubicato nel file system "root" (/),
4. L'indirizzario delle classi nell'indirizzario Prodotto nel file system "root" (/).

L'ordine di ricerca per J2SDK, utilizzando il precedente esempio, è:

1. Il classpath del bootstrap, che è nella proprietà sun.boot.class.path,
2. Gli indirizzari di estensione, che si trovano 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 viene impostata sulla variabile di ambiente. Il parametro del classpath specifica una lista di nomi 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
- Comando RUNJVA (Esecuzione Java)

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

E' possibile sostituire la variabile di ambiente CLASSPATH utilizzando la proprietà java.class.path. E' possibile modificare sia la proprietà java.class.path che altre proprietà, utilizzando il file

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

In JDK 1.1.x, la proprietà os400.class.path.system influenza inoltre ciò che si ricerca quando si cercano le classi. Questa proprietà può contenere uno o tre valori: PRE, POST o NONE. Per impostazione predefinita, questa proprietà viene impostata su PRE, che provoca che il classpath del sistema predefinito venga ricercato prima del percorso. La variabile di ambiente CLASSPATH o il parametro del classpath specificano questo percorso. Se si imposta la proprietà os400.class.path.system su POST, il classpath del sistema predefinito si recherà dopo ogni classpath specificato dall'utente. Se viene utilizzato il valore NONE, il classpath predefinito non viene ricercato e si ricerca solo il classpath specificato dall'utente.

» In J2SDK, è possibile che l'opzione -Xbootclasspath venga utilizzata con lo stesso effetto. -Xbootclasspath/a:path accoda path al classpath del bootstrap predefinito, /p:path antepone path al classpath del bootstrap e /r:path sostituisce il classpath del bootstrap con path. «

Nota: specificare con cautela NONE o POST in quanto si verificano risultati imprevisti se non è possibile trovare una classe di sistema o se essa viene sostituita in modo non corretto da una classe definita dall'utente. Quindi, sarebbe opportuno consentire la ricerca del classpath predefinito del sistema prima di ogni classpath specificato dall'utente.

Consultare Proprietà di sistema Java per informazioni su come stabilire l'ambiente in cui si eseguono 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^(TM) determinano l'ambiente in cui vengono eseguiti i programmi Java. Esse sono simili ai valori di sistema o alle variabili di ambiente in OS/400^(R). Un certo numero di proprietà vengono impostate quando si avvia la JVM (Java virtual machine).

Per visualizzare le proprietà di sistema supportate da questo rilascio collegarsi alla versione scelta per informazioni dettagliate:

- JDK (Java Development Kit) 1.1.8
- J2SDK (Java 2 Software Development Kit), Edizione Standard » per le versioni 1.2, 1.3 e 1.4. «

File SystemDefault.properties

Il file SystemDefault.properties rappresenta un file di proprietà Java^(TM) standard. E' possibile specificare le proprietà predefinite in un file SystemDefault.properties. 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 SystemDefault.properties che si trova nell'indirizzario principale vengono utilizzate solamente per le Java virtual machine avviate o per gli utenti che specificano la proprietà user.home = /YourUserHome/.

E' possibile specificare i valori delle proprietà nel file SystemDefault.properties nello stesso modo con cui lo si fa in qualsiasi file di proprietà Java.

Esempio: file SystemDefault.properties

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
#Comments start with pound sign
#this means always run with JDK 1.3
java.version=1.3
#set my special property
myown.propname=6
```


Proprietà di sistema Java per JDK (Java Development Kit) 1.1.8

Le proprietà di sistema Java^(TM) determinano l'ambiente in cui viene eseguito il programma Java. Essi sono simili ai valori di sistema o alle variabili di ambiente di OS/400^(R). Un certo numero di proprietà vengono impostate quando si avvia la JVM (Java virtual machine).

Le proprietà di sistema vengono impostate su questi valori predefiniti di sistema nel JDK 1.1.8.

Proprietà di sistema	Valori predefiniti di sistema
awt.toolkit	com.ibm.rawt.client.CToolkit
file.encoding.pkg	sun.io
file.separator	/ (barra in avanti)
java.class.version	45.3
java.home	consultare Supporto per più JDK (Java Development Kit) per dettagli.
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
line.separator	\n
os.arch	PowerPC
os.name	OS/400
os400.class.path.rawt	0
os400.class.path.security.check	20 Valori validi: 0—Nessun controllo di sicurezza 10—equivalente a RUNJVA CHPATH(*IGNORE) 20—equivalente a RUNJVA CHPATH(*WARN) 30—equivalente a RUNJVA CHPATH(*SECURE)
os400.class.path.tools	0
os400.create.type	direct Valori validi: interpret—equivalente a RUNJVA OPTIMIZE(*INTERPRET) e INTERPRET(*OPTIMIZE) o INTERPRET(*YES) direct—Altrimenti
os400.defineClass.optLevel	20
os400.enbpfrcol	0 Valori validi: 0—equivalente a CRTJVAPGM ENBPFRCOL(*NONE) 1—equivalente a CRTJVAPGM ENBPFRCOL(*ENTRYEXIT) 7—equivalente a CRTJVAPGM ENBPFRCOL(*FULL)
os400.interpret	0 Valori validi: 0—equivalente a CRTJVAPGM INTERPRET(*NO) 1—equivalente a CRTJVAPGM INTERPRET(*YES)
os400.optimization	10 Valori validi: 0—equivalente a CRTJVAPGM OPTIMIZE(*INTERPRET) 10—equivalente a CRTJVAPGM OPTIMIZE(10) 20—equivalente a CRTJVAPGM OPTIMIZE(20) 30—equivalente a CRTJVAPGM OPTIMIZE(30) 40—equivalente a CRTJVAPGM OPTIMIZE(40)

Proprietà di sistema	Valori predefiniti di sistema
os400.run.mode	program_create_type Valori validi: interpret—equivalente a RUNJVA OPTIMIZE(*INTERPRET) e INTERPRET(*OPTIMIZE) o INTERPRET(*YES) program_create_type—Altrimenti
os400.stdin.allowed	0
os400.verify.checks.disable	65535 os400.class. Questo valore della proprietà di sistema è una stringa che rappresenta la somma di uno o più valori numerici. Per una lista di questi valori, consultare i valori numerici os400.verify.checks.disable.
path.separator	: (due punti)

Questa serie di proprietà di sistema è impostata in base a informazioni aggiuntive sul sistema.

Proprietà di sistema	Descrizione
file.encoding	Correla il lavoro CCSID OS/400 al corrispondente CCSID ISO ASCII. Inoltre, imposta il valore file.encoding sul valore Java che rappresenta il CCSID (coded character set identifier) di ISO ASCII. Consultare valori file.encoding e CCSID iSeries per una tabella che indica la relazione tra i valori file.encoding possibili e il CCSID (coded character set identifier) iSeries più strettamente corrispondente.
java.class.path	Il percorso utilizzato per ubicare le classi. Assume per impostazione predefinita il valore del classpath di sistema predefinito seguito dal classpath specificato dall'utente. E' possibile modificare la proprietà di sistema java.class.path utilizzando la proprietà di sistema os400.class.path.system.
java.compiler	Specifica se il codice è compilato con il compilatore JIT (Just-In-Time) (jitc) oppure sia con il compilatore JIT che con l'elaborazione diretta (jitc_de).
java.version	Determina quale JDK (Java Development Kit) eseguire. Se si specifica un JDK non installato, viene restituito un messaggio di errore. Se non si specifica un JDK, il JDK più recente risulta l'impostazione predefinita. Se solo un JDK è installato, questa rappresenta il valore predefinito. Consultare Supporto per più JDK per dettagli sulla versione.
os.version	Ottiene il livello di rilascio OS/400 dall'API (application program interface) di richiamo delle informazioni sul prodotto.
os400.CertificateContainer	Indirizza il supporto SSL (secure socketlayer) Java in modo tale che utilizzi il contenitore di certificati specificato 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 file chiave nell'integrated file system.

Proprietà di sistema	Descrizione
os400.CertificateLabel	E' 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	Consente il controllo della conversione dati per stdin, stdout e stderr in Java. La conversione dei dati si verifica per impostazione predefinita nella Java virtual machine per convertire i dati ASCII in o da EBCDIC. E' possibile attivare o disattivare queste conversioni con questa proprietà, la quale influenza ogni processo secondario avviato da questo processo utilizzando il metodo runtime.exec(). Consultare valori predefiniti.
os400.class.path.system	PRE (questo valore fa sì che il classpath di sistema predefinito os400 sia anteposto sulla parte del classpath specificata dall'utente durante la creazione della proprietà di sistema java.class.path). Altri valori sono POST (la classe predefinita di sistema viene accodata alla parte del classpath specificata dall'utente) e NONE (viene utilizzato solo il classpath specificato dall'utente). Il valore predefinito è PRE. La proprietà non è sensibile al maiuscolo e al minuscolo. Ad esempio, è possibile specificare NONE, none, noNe. Il nome della proprietà, tuttavia, risulta sensibile al maiuscolo e al minuscolo. Ad esempio, non è possibile specificare OS400.CLASS.PATH.SYSTEM. Per evitare potenziali problemi, è necessario modificare questa proprietà di sistema.
» 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. E' possibile specificare os400.file.create.auth=RWX o os400.dir.create.auth=RWX, dove R=lettura, W=scrittura e X=esecuzione. Risulta valida qualsiasi combinazione di queste autorizzazioni. <<
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.jit.mmi.threshold	Imposta il numero di volte in cui viene eseguito un metodo prima che venga compilato con il compilatore JIT. <<
» os400.pool.size	Definisce quanto spazio (in kilobyte) rendere disponibile per ogni lotto heap nell'heap locale del sottoprocesso. <<

Proprietà di sistema	Descrizione
os400.runtime.exec	<ul style="list-style-type: none"> EXEC (valore predefinito per 1.3 e per le versioni successive) - Richiamare le funzioni tramite runtime.exec() utilizzando l'interfaccia EXEC. Questo è il valore più compatibile con le altre piattaforme. QSHELL (valore predefinito per 1.2 e per le versioni precedenti) - Richiamare le funzioni tramite runtime.exec() utilizzando QSHELL Interpreter. Questo consente l'utilizzo della sostituzione di variabile e del richiamo delle funzioni incorporate.
os400.secureApplication	Associa il programma Java che si avvia utilizzando questa proprietà di sistema (os400.secureApplication) con il nome dell'applicazione sicura registrata. E' possibile visualizzare i nomi dell'applicazione sicura registrata utilizzando il DCM (Digital Certificate Manager).
os400.stderr	Consente la correlazione di stderr a un file o un socket. Consultare valori predefiniti.
os400.stdin	Consente la correlazione di stdin a un file o un socket. Consultare valori predefiniti.
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 Java virtual machine per convertire i dati ASCII in o da EBCDIC. E' possibile attivare o disattivare queste conversioni con questa proprietà, la quale influenza il programma Java corrente. Consultare valori predefiniti.
os400.stdout	Consente la correlazione di stdout a un file o un socket. Consultare valori predefiniti.
user.dir	CWD (Current working directory - Indirizzario di lavoro corrente utilizzando l'API getcwd.
user.home	Richiama l'indirizzario di lavoro iniziale utilizzando l'API Get (getpwnam). E' possibile posizionare un file SystemDefault.properties in un percorso user.home per sostituire le proprietà predefinite in /QIBM/UserData/Java400/SystemDefault.properties. E' possibile personalizzare il server iSeries per specificare la propria serie di valori di proprietà predefiniti.
user.language	La 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 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 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 Java virtual machine utilizza questa proprietà di sistema per ottenere il nome del fuso orario utilizzando l'API QlgRetrieveLocalInformation. Se non è disponibile alcuna informazione sul fuso orario, user.timezone è impostato su UTC (Universal Time Coordinate).

Valori della proprietà di sistema `os400.stdio.convert` e `os400.child.stdio.convert`

La tabella seguente indica i valori di sistema sia per la proprietà di sistema `os400.stdio.convert` che per la proprietà di sistema `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 <code>file.encoding</code> al CCSID del lavoro durante la lettura o la scrittura.
1	Solo i dati <code>stdin</code> vengono convertiti dal CCSID del lavoro al <code>file.encoding</code> avviene la lettura.
2	Solo i dati <code>stdout</code> vengono convertiti dal <code>file.encoding</code> al CCSID del lavoro durante la scrittura.
3	Vengono eseguite sia conversioni <code>stdin</code> che <code>stdout</code> .
4	Solo i dati <code>stderr</code> vengono convertiti dal <code>file.encoding</code> al CCSID del lavoro durante la scrittura.
5	Vengono eseguite sia conversioni <code>stdin</code> che <code>stderr</code> .
6	Vengono eseguite sia le conversioni <code>stdout</code> che <code>stderr</code> .
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.	<code>file:/QIBM/UserData/Java400/Output.file</code>
Porta	HostName	Indirizzo porta	<code>port:myhost:2000</code>
Porta	TCPAddress	Indirizzo porta	<code>port:1.1.11.111:2000</code>

Valori numerici `os400.verify.checks.disable`

Il valore della proprietà di sistema `os400.verify.checks.disable` è una stringa che rappresenta la somma di una o più valori numerici dalla lista seguente:

Valore	Descrizione
1	Saltare i controlli di accesso per le classi locali: indica che si desidera che la Java [™] 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 <code>NoClassDefFoundError</code> durante il caricamento iniziale: indica che si desidera che la JVM (Java virtual machine) ignori <code>NoClassDefFoundErrors</code> , 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 <code>LocalVariableTable</code>: indica che se si incontra un errore nella <code>LocalVariableTable</code> di una classe, la classe opera come se la <code>LocalVariableTable</code> non esistesse. Altrimenti gli errori nella <code>LocalVariableTable</code> danno come risultato un <code>ClassFormatError</code> .
7	Valore utilizzato durante il tempo di esecuzione.

E' 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 dalla lista, utilizzare questa sintassi del comando iSeries:

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

Proprietà di sistema Java per J2SDK (Java 2 Software Development Kit), Edizione Standard

Le proprietà di sistema Java^(TM) determinano l'ambiente in cui viene eseguito il programma Java. Queste sono simili ai valori di sistema o alle variabili di ambiente in OS/400. Un certo numero di proprietà vengono impostate quando si avvia la JVM (Java virtual machine).

» Le proprietà di sistema sono impostate su questi valori predefiniti di sistema in J2SDK (Java 2 Software Development Kit), Edizione Standard, versione 1.4. Alcune proprietà di sistema hanno valori predefiniti diversi se utilizza il comando CL JAVA o RUNJAVA piuttosto che utilizzare l'API di richiamo JNI (Java Native Interface). La tabella seguente riflette l'utilizzo dell'API. «

Proprietà di sistema	Valori predefiniti di sistema
awt.toolkit	Per JDK 1.1.x, il valore predefinito è com.ibm.rawt.client.CToolkit. Per J2SDK, il valore predefinito è com.ibm.rawt2.ahost.java.awt.AHToolkit.
file.encoding.pkg	sun.io
file.separator	/ (barra in avanti)
java.class.version	» 48.0 «
java.ext.dirs	» /QIBM/ProdData/Java400/jdk14/lib/ext:/QIBM/UserData/Java400/ext «
java.home	consultare Supporto per più JDK (Java Development Kit) per dettagli.
java.library.path	Lista librerie OS/400
java.policy	» /QIBM/ProdData/Java400/jdk14/lib/security/java.policy «
java.specification.name	Specifica di linguaggio Java
java.specification.vendor	Sun Microsystems, Inc.
java.specification.version	» 1.4 «
» sun.boot.class.path «	Class_Path_Sys
java.use.policy	true
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
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	» 1.4 «
line.separator	\n
os.arch	PowerPC

Proprietà di sistema	Valori predefiniti di sistema
os.name	OS/400
os400.class.path.rawt	0
os400.class.path.security.check	20 Valori validi: 0—Nessun controllo di sicurezza 10—equivalente a RUNJVA CHPATH(*IGNORE) 20—equivalente a RUNJVA CHPATH(*WARN) 30—equivalente a RUNJVA CHPATH(*SECURE)
os400.class.path.tools	0
os400.create.type	» interpret « Valori validi: interpret—equivalente a RUNJVA OPTIMIZE(*INTERPRET) e INTERPRET(*OPTIMIZE) o INTERPRET(*YES) direct—Altrimenti
os400.defineClass.optLevel	20
os400.enbpfrcol	0 Valori validi: 0—equivalente a CRTJVAPGM ENBPFRCOL(*NONE) 1—equivalente a CRTJVAPGM ENBPFRCOL(*ENTRYEXIT) 7—equivalente a CRTJVAPGM ENBPFRCOL(*FULL) Per un valore diverso da zero, il JIT genera eventi *JVAENTRY, *JVAEXIT, *JVAPRECALL e *JVAPOSTCALL.
os400.interpret	0 Valori validi: 0—equivalente a CRTJVAPGM INTERPRET(*NO) 1—equivalente a CRTJVAPGM INTERPRET(*YES)
os400.optimization	» 0 « Valori validi: 0—equivalente a CRTJVAPGM OPTIMIZE(*INTERPRET) 10—equivalente a CRTJVAPGM OPTIMIZE(10) 20—equivalente a CRTJVAPGM OPTIMIZE(20) 30—equivalente a CRTJVAPGM OPTIMIZE(30) 40—equivalente a CRTJVAPGM OPTIMIZE(40)
os400.run.mode	» jitc_de « Valori validi: interpret—equivalente a RUNJVA OPTIMIZE(*INTERPRET) e INTERPRET(*OPTIMIZE) o INTERPRET(*YES) program_create_type jitc_de—Altrimenti
os400.stdin.allowed	0
os400.verify.checks.disable	65535 Questo valore della proprietà di sistema è una stringa che rappresenta la somma di uno o più valori numerici. Per una lista di questi valori, consultare i valori numerici os400.verify.checks.disable.
path.separator	: (due punti)

Questa serie di proprietà di sistema è impostata in base a informazioni aggiuntive sul sistema.

Proprietà di sistema	Descrizione
file.encoding	Correla il lavoro CCSID OS/400 al corrispondente CCSID ISO ASCII. Inoltre, imposta il valore file.encoding sul valore Java che rappresenta il CCSID (coded character set identifier) di ISO ASCII. Consultare valori file.encoding e CCSID iSeries per una tabella che indica la relazione tra i valori file.encoding possibili e il CCSID (coded character set identifier) iSeries più strettamente corrispondente.
java.class.path	Il percorso utilizzato per ubicare le classi. Assume il valore predefinito per il classpath specificato dall'utente.
java.compiler	Specifica se il codice è compilato con il compilatore JIT (Just-In-Time) (jitc) oppure sia con il compilatore JIT che con l'elaborazione diretta (jitc_de).
java.version	Determina quale JDK (Java Development Kit) eseguire. Se si specifica un JDK non installato, viene restituito un messaggio di errore. Se non si specifica un JDK, il JDK più recente risulta l'impostazione predefinita. Se solo un JDK è installato, questa rappresenta il valore predefinito. Consultare Supporto per più JDK per dettagli sulla versione.
os.version	Ottiene il livello di rilascio OS/400 dall'API (application programm interface) di richiamo delle informazioni sul prodotto.
os400.CertificateContainer	Indirizza il supporto SSL (secure socketlayer) Java in modo tale che utilizzi il contenitore di certificati specificato 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	E' 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 socketlayer) 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	Consente il controllo della conversione dati per stdin, stdout e stderr in Java. La conversione dei dati si verifica per impostazione predefinita nella Java virtual machine per convertire i dati ASCII in o da EBCDIC. E' possibile attivare o disattivare queste conversioni con questa proprietà, la quale influenza ogni processo secondario avviato da questo processo utilizzando il metodo runtime.exec(). Consultare valori predefiniti.
os400.class.path.system	➤ Questa proprietà di sistema viene ignorata per J2SDK. ⚡

Proprietà di sistema	Descrizione
» os400.define.class.cache.file	Questa proprietà specifica il nome di un file JAR o ZIP. Il valore predefinito è null. Consultare Programmi di caricamento di classe nella memoria cache. <<
» os400.define.class.cache.hours	Questa proprietà rappresenta un valore decimale. Il valore predefinito è 168 e il valore decimale massimo è 9999. Consultare Programmi di caricamento di classe nella memoria cache. <<
» os400.define.class.cache.maxpgms	Questa proprietà rappresenta un valore decimale. Il valore predefinito è 5000 e il valore decimale massimo è 40000. Consultare Programmi di caricamento di classe nella memoria cache. <<
os400.exception.trace	Specificando questa proprietà si verifica l'invio delle eccezioni più recenti all'emissione standard quando esiste la JVM. Il valore assegnato a questa proprietà viene attualmente ignorato, ma è possibile che questa situazione si modifichi in futuro. Questa proprietà è utilizzata solamente per il debug.
» 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. E' possibile specificare os400.file.create.auth=RWX o os400.dir.create.auth=RWX, dove R=lettura, W=scrittura e X=esecuzione. Risulta valida qualsiasi combinazione di queste autorizzazioni. <<
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.jit.mmi.threshold	Imposta il numero di volte in cui viene eseguito un metodo prima che venga compilato con il compilatore JIT. <<
» os400.pool.size	Definisce quanto spazio (in kilobyte) rendere disponibile per ogni lotto heap nell'heap locale del sottoprocesso. <<
os400.runtime.exec	<ul style="list-style-type: none"> EXEC (valore predefinito per 1.3 e per le versioni successive) - Richiamare le funzioni tramite runtime.exec() utilizzando l'interfaccia EXEC. Questo è il valore più compatibile con le altre piattaforme. QSHELL (valore predefinito per 1.2 e per le versioni precedenti) - Richiamare le funzioni tramite runtime.exec() utilizzando QSHELL Interpreter. Questo consente l'utilizzo della sostituzione di variabile e del richiamo delle funzioni incorporate.
os400.secureApplication	Associa il programma Java che si avvia utilizzando questa proprietà di sistema (os400.secureApplication) con il nome dell'applicazione sicura registrata. E' possibile visualizzare i nomi dell'applicazione sicura registrata utilizzando il DCM (Digital Certificate Manager).
» os.400.security.properties	Consente un controllo completo del file java.security utilizzato. Quando si specifica questo supporto, J2SDK non utilizzerà nessun altro file java.security, compresa la specifica J2SDK java.security predefinita <<

Proprietà di sistema	Descrizione
os400.stderr	Consente la correlazione di stderr a un file o un socket. Consultare valori predefiniti.
os400.stdin	Consente la correlazione di stdin a un file o un socket. Consultare valori predefiniti.
os400.stdin.allowed	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. Il valore predefinito è 0.
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. E' possibile attivare o disattivare queste conversioni con questa proprietà, la quale influenza il programma Java corrente. Consultare valori predefiniti.
os400.stdout	Consente la correlazione di stdout a un file o un socket. Consultare valori predefiniti.
user.dir	CWD (Current working directory - Indirizzario di lavoro corrente utilizzando l'API getcwd.
user.home	Richiama l'indirizzario di lavoro iniziale utilizzando l'API Get (getpwnam). E' possibile posizionare un file SystemDefault.properties in un percorso user.home per sostituire le proprietà predefinite in /QIBM/UserData/Java400/SystemDefault.properties. E' possibile personalizzare il server iSeries per specificare la propria serie di valori di proprietà predefiniti.
user.language	La 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 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 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 Java virtual machine utilizza questa proprietà di sistema per ottenere il nome del fuso orario utilizzando l'API QlgRetrieveLocalInformation. Se non è disponibile alcuna informazione sul fuso orario, user.timezone è impostato su UTC (Universal Time Coordinate).

Creare un programma Java internazionalizzato

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

Per creare un programma Java internazionalizzato, effettuare quanto segue:

1. Isolare i dati e il codice sensibile alla locale. Ad esempio, stringhe, date e numeri nel proprio programma.
2. Impostare o acquisire la locale utilizzando la classe Locale.
3. Formattare date e numeri per specificare una locale se non si intende utilizzare la locale predefinita.

4. Creare pacchetti di risorse per gestire stringhe e altri dati sensibili alla locale.

Per eseguire queste attività nei propri programmi Java, far riferimento a questi esempi:

- Internazionalizzazione di date utilizzando la classe `java.util.DateFormat`
- Internazionalizzazione del pannello numerico utilizzando la classe `java.util.NumberFormat`
- Internazionalizzazione di dati specifici della locale utilizzando la classe `java.util.ResourceBundle`

Per ulteriori informazioni sull'internazionalizzazione, fare clic su uno di questi collegamenti:

- Globalizzazione OS/400
- Internazionalizzazione di Sun Microsystems, Inc.

Variabile di ambiente del fuso orario sul proprio server iSeries

Con J2SDK (JavaTM 2 SDK), Edizione Standard, versione 1.4, è possibile impostare il fuso orario della propria JVM (Java Virtual Machine) utilizzando il metodo nativo `getTimeZoneID()`. Il server iSeries utilizza l'oggetto `*LOCALE` che fa parte dell'oggetto `*ENV`. Impostare il campo `tname` nell'oggetto `*LOCALE` sul valore di sistema appropriato. Tale valore viene quindi restituito come oggetto della stringa Java associato da `getTimeZoneID()`.

Configurare il fuso orario: La JVM richiede sia il valore di sistema `QUTCFFSET` che le informazioni sull'ora del giorno nella `LOCALE` affinché sia impostato il lavoro corrente per stabilire correttamente l'ora locale. `QUTCFFSET` è un valore di sistema che specifica il numero di ore di differenza tra l'ora locale corrente e l'UTC (universal time coordinate). Per il CST (central standard time), tale valore dovrebbe essere -6:00. Per il CDT (central daylight time), il valore corretto è -5:00. Il valore `QUTCFFSET` consente alla JVM di determinare il valore corretto per UTC.

Le informazioni `LOCALE` per un lavoro vengono impostate creando un oggetto `*LOCALE` che contiene informazioni sull'ora del giorno e che specifica che `*LOCALE` per il lavoro utilizza il valore di sistema `QLOCALE` o la parola chiave `LOCALE` sul profilo utente per il lavoro. Per dettagli sulla creazione e sull'utilizzo di `LOCALE`, consultare la pubblicazione `Globalizzazione OS/400`.

L'impostazione corretta delle informazioni `*LOCALE` consente a JVM di impostare il valore predefinito della proprietà `user.timezone` sul corretto fuso orario. E' possibile impostare manualmente la proprietà `user.timezone` sulla riga comandi per sostituire l'impostazione predefinita fornita dall'oggetto `*LOCALE`.

Segue un esempio delle informazioni `LC_TOD` che è necessario includere nell'oggetto `*LOCALE` per configurare il corretto fuso orario per Java:

```
LC_TOD

% TZDIFF is number of minutes difference from GMT
tzdiff    -300
% Timezone name (this is the value that you would have passed to
% the JVM as the user.timezone property.) See abbreviations later
% in this document.
tname     "<C><S><T>"
% Name used for daylight savings time.
dstname   "<C><D><T>"
% DST Start in this part of the US is the first Sunday in April at 2am
dststart  4,1,1,7200
% DST End in this area of US is Last Sunday in October.
dstend    10,-1,1,7200
% shift in seconds
dstshift  3600

END LC_TOD
```

La seguente tabella indica i valori di sistema e gli oggetti stringa Java associati.

Nota: il valore di sistema "Hong Kong" rappresenta la Cina (Hong Kong S.A.R.).

Valore di sistema	Oggetto stringa Java
Africa/Abidjan	Africa/Abidjan
Africa/Accra	Africa/Accra
Africa/Addis_Abeba	Africa/Addis_Abeba
Africa/Algeri	Africa/Algeri
Africa/Asmara	Africa/Asmara
Africa/Bamako	GMT
Africa/Bangui	Africa/Bangui
Africa/Banjul	Africa/Banjul
Africa/Bissau	Africa/Bissau
Africa/Blantyre	Africa/Blantyre
Africa/Brazzaville	Africa/Luanda
Africa/Bujumbura	Africa/Bujumbura
Africa/Cairo	Africa/Cairo
Africa/Casablanca	Africa/Casablanca
Africa/Ceuta	Europa/Parigi
Africa/Conakry	Africa/Conakry
Africa/Dakar	Africa/Dakar
Africa/Dar_es_Salaam	Africa/Dar_es_Salaam
Africa/Gibuti	Africa/Gibuti
Africa/Douala	Africa/Douala
Africa/El_Aaiun	Africa/Casablanca
Africa/Freetown	Africa/Freetown
Africa/Gaborone	Africa/Gaborone
Africa/Harare	Africa/Harare
Africa/Johannesburg	Africa/Johannesburg
Africa/Kampala	Africa/Kampala
Africa/Khartoum	Africa/Khartoum
Africa/Kigali	Africa/Kigali
Africa/Kinshasa	Africa/Kinshasa
Africa/Lagos	Africa/Lagos
Africa/Libreville	Africa/Libreville
Africa/Lome	Africa/Lome
Africa/Luanda	Africa/Luanda
Africa/Lubumbashi	Africa/Lubumbashi
Africa/Lusaka	Africa/Lusaka
Africa/Malabo	Africa/Malabo
Africa/Maputo	Africa/Maputo
Africa/Maseru	Africa/Maseru
Africa/Mbabane	Africa/Mbabane
Africa/Mogadiscio	Africa/Mogadiscio
Africa/Monrovia	Africa/Monrovia

Valore di sistema	Oggetto stringa Java
Africa/Nairobi	Africa/Nairobi
Africa/Ndjamena	Africa/Ndjamena
Africa/Niamey	Africa/Niamey
Africa/Nouakchott	Africa/Nouakchott
Africa/Ouagadougou	Africa/Ouagadougou
Africa/Porto-Novo	Africa/Porto-Novo
Africa/Sao_Tome	Africa/Sao_Tome
Africa/Timbuktu	Africa/Timbuktu
Africa/Tripoli	Africa/Tripoli
Africa/Tunisi	Africa/Tunisi
Africa/Windhoek	Africa/Windhoek
America/Adak	America/Adak
America/Anchorage	America/Anchorage
America/Anguilla	America/Anguilla
America/Antigua	America/Antigua
America/Araguaina	America/San_Paolo
America/Aruba	America/Aruba
America/Asuncion	America/Asuncion
America/Atka	America/Adak
America/Barbados	America/Barbados
America/Belize	America/Belize
America/Bogotá	America/Bogotá
America/Boise	America/Denver
America/Buenos_Aires	America/Buenos_Aires
America/Cancun	America/Chicago
America/Caracas	America/Caracas
America/Cayenne	America/Cayenne
America/Cayman	America/Cayman
America/Chicago	America/Chicago
America/Chihuahua	America/Denver
America/Costa_Rica	America/Costa_Rica
America/Cuiaba	America/Cuiaba
America/Curacao	America/Curacao
America/Dawson	America/Los_Angeles
America/Dawson_Creek	America/Dawson_Creek
America/Denver	America/Denver
America/Detroit	America/New_York
America/Dominica	America/Dominica
America/Edmonton	America/Edmonton
America/EI_Salvador	America/EI_Salvador
America/Ensenada	America/Los_Angeles

Valore di sistema	Oggetto stringa Java
America/Fort_Wayne	America/Indianapolis
America/Fortaleza	America/Fortaleza
America/Glace_Bay	America/Halifax
America/Godthab	America/Godthab
America/Goose_Bay	America/Thule
America/Grand_Turk	America/Grand_Turk
America/Granada	America/Granada
America/Guadalupe	America/Guadalupe
America/Guatemala	America/Guatemala
America/Guayaquil	America/Guayaquil
America/Guyana	America/Guyana
America/Halifax	America/Halifax
America/L'Avana	America/L'Avana
America/Indiana/Indianapolis	America/Indianapolis
America/Indianapolis	America/Indianapolis
America/Inuvik	America/Denver
America/Iqaluit	America/New_York
America/Giamaica	America/Giamaica
America/Juneau	America/Anchorage
America/La_Paz	America/La_Paz
America/Lima	America/Lima
America/Los_Angeles	America/Los_Angeles
America/Louisville	America/New_York
America/Managua	America/Managua
America/Manaus	America/Manaus
America/Martinica	America/Martinica
America/Mazatlan	America/Mazatlan
America/Menominee	America/Winnipeg
America/Città_del_Messico	America/Città_del_Messico
America/Miquelon	America/Miquelon
America/Montevideo	America/Montevideo
America/Montreal	America/Montreal
America/Montserrat	America/Montserrat
America/Nassau	America/Nassau
America/New_York	America/New_York
America/Nipigon	America/New_York
America/Nome	America/Anchorage
America/Noronha	America/Noronha
America/Panama	America/Panama
America/Pangnirtung	America/Thule
America/Paramaribo	America/Paramaribo

Valore di sistema	Oggetto stringa Java
America/Phoenix	America/Phoenix
America/Port-au-Prince	America/Port-au-Prince
America/Port_of_Spain	America/Port_of_Spain
America/Porto_Acre	America/Porto_Acre
America/Porto_Rico	America/Porto_Rico
America/Rainy_River	America/Chicago
America/Rankin_Inlet	America/Chicago
America/Regina	America/Regina
America/Santiago	America/Santiago
America/Santo_Domingo	America/Santo_Domingo
America/San_Paolo	America/San_Paolo
America/Scoresbysund	America/Scoresbysund
America/Shiprock	America/Denver
America/St_Johns	America/St_Johns
America/St_Kitts	America/St_Kitts
America/St_Lucia	America/St_Lucia
America/St_Thomas	America/St_Thomas
America/St_Vincent	America/St_Vincent
America/Tegucigalpa	America/Tegucigalpa
America/Thule	America/Thule
America/Thunder_Bay	America/New_York
America/Tijuana	America/Tijuana
America/Tortola	America/Tortola
America/Vancouver	America/Vancouver
America/Virgin	America/St_Thomas
America/Whitehorse	America/Los_Angeles
America/Winnipeg	America/Winnipeg
America/Yakutat	America/Anchorage
America/Yellowknife	America/Denver
Antartide/Casey	Antartide/Casey
Antartide/DumontDURville	Antartide/DumontDURville
Antartide/Mawson	Antartide/Mawson
Antartide/McMurdo	Antartide/McMurdo
Antartide/Palmer	Antartide/Palmer
Antartide/Polo_Sud	Antartide/McMurdo
Artide/Longyearbyen	Europa/Oslo
Asia/Aden	Asia/Aden
Asia/Almaty	Asia/Almaty
Asia/Amman	Asia/Amman
Asia/Anadyr	Asia/Anadyr
Asia/Aqtau	Asia/Aqtau

Valore di sistema	Oggetto stringa Java
Asia/Aqtobe	Asia/Aqtobe
Asia/Ashkhabad	Asia/Ashkhabad
Asia/Bagdad	Asia/Bagdad
Asia/Bahrain	Asia/Bahrain
Asia/Baku	Asia/Baku
Asia/Bangkok	Asia/Bangkok
Asia/Beirut	Asia/Beirut
Asia/Bishkek	Asia/Bishkek
Asia/Brunei	Asia/Brunei
Asia/Calcutta	Asia/Calcutta
Asia/Chungking	Asia/Shanghai
Asia/Colombo	Asia/Colombo
Asia/Dacca	Asia/Dacca
Asia/Damasco	Asia/Damasco
Asia/Dubai	Asia/Dubai
Asia/Dushanbe	Asia/Dushanbe
Asia/Gaza	Asia/Amman
Asia/Harbin	Asia/Shanghai
Asia/Hong_Kong	Asia/Hong_Kong
Asia/Irkutsk	Asia/Irkutsk
Asia/Istanbul	Europa/Istanbul
Asia/Giacarta	Asia/Giacarta
Asia/Jayapura	Asia/Jayapura
Asia/Gerusalemme	Asia/Gerusalemme
Asia/Kabul	Asia/Kabul
Asia/Kamchatka	Asia/Kamchatka
Asia/Karachi	Asia/Karachi
Asia/Kashgar	Asia/Shanghai
Asia/Katmandu	Asia/Katmandu
Asia/Krasnoyarsk	Asia/Krasnoyarsk
Asia/Kuala_Lumpur	Asia/Kuala_Lumpur
Asia/Kuwait	Asia/Kuwait
Asia/Macao	Asia/Macao
Asia/Magadan	Asia/Magadan
Asia/Manila	Asia/Manila
Asia/Muscat	Asia/Muscat
Asia/Nicosia	Asia/Nicosia
Asia/Novosibirsk	Asia/Novosibirsk
Asia/Omsk	Asia/Novosibirsk
Asia/Phnom_Penh	Asia/Phnom_Penh
Asia/Pyongyang	Asia/Pyongyang

Valore di sistema	Oggetto stringa Java
Asia/Qatar	Asia/Qatar
Asia/Rangoon	Asia/Rangoon
Asia/Riyadh	Asia/Riyadh
Asia/Saigon	Asia/Saigon
Asia/Seul	Asia/Seul
Asia/Shanghai	Asia/Shanghai
Asia/Singapore	Asia/Singapore
Asia/Taipei	Asia/Taipei
Asia/Tashkent	Asia/Tashkent
Asia/Tbilisi	Asia/Tbilisi
Asia/Teheran	Asia/Teheran
Asia/Tel_Aviv	Asia/Gerusalemme
Asia/Thimbu	Asia/Thimbu
Asia/Tokyo	Asia/Tokyo
Asia/Ujung_Pandang	Asia/Ujung_Pandang
Asia/Ulan_Bator	Asia/Ulan_Bator
Asia/Urumqi	Asia/Shanghai
Asia/Vientiane	Asia/Vientiane
Asia/Vladivostok	Asia/Vladivostok
Asia/Yakutsk	Asia/Yakutsk
Asia/Yekaterinburg	Asia/Yekaterinburg
Asia/Yerevan	Asia/Yerevan
Atlantico/Azzorre	Atlantico/Azzorre
Atlantico/Bermuda	Atlantico/Bermuda
Atlantico/Canarie	Atlantico/Canarie
Atlantico/Capo_Verde	Atlantico/Capo_Verde
Atlantico/Faeroe	Atlantico/Faeroe
Atlantico/Jan_Mayen	Atlantico/Jan_Mayen
Atlantico/Madeira	Europa/Londra
Atlantico/Reykjavik	Atlantico/Reykjavik
Atlantico/South_Georgia	Atlantico/South_Georgia
Atlantico/St_Helena	Atlantico/St_Helena
Atlantico/Stanley	Atlantico/Stanley
Australia/ACT	Australia/Sydney
Australia/Adelaide	Australia/Adelaide
Australia/Brisbane	Australia/Brisbane
Australia/Broken_Hill	Australia/Broken_Hill
Australia/Canberra	Australia/Sydney
Australia/Darwin	Australia/Darwin
Australia/Hobart	Australia/Hobart
Australia/LHI	Australia/Lord_Howe

Valore di sistema	Oggetto stringa Java
Australia/Lord_Howe	Australia/Lord_Howe
Australia/Melbourne	Australia/Sydney
Australia/NSW	Australia/Sydney
Australia/North	Australia/Darwin
Australia/Perth	Australia/Perth
Australia/Queensland	Australia/Brisbane
Australia/South	Australia/Adelaide
Australia/Sydney	Australia/Sydney
Australia/Tasmania	Australia/Hobart
Australia/Victoria	Australia/Sydney
Australia/West	Australia/Perth
Australia/Yancowinna	Australia/Broken_Hill
Brasile/Acre	America/Porto_Acre
Brasile/DeNoronha	America/Noronha
Brasile/Orientale	America/San_Paolo
Brasile/Occidentale	America/Manaus
CET	Europa/Parigi
CST	America/Chicago
CST6CDT	America/Chicago
Canada/Atlantico	America/Halifax
Canada/Centrale	America/Winnipeg
Canada/East-Saskatchewan	America/Regina
Canada/Orientale	America/Montreal
Canada/Zona montuosa	America/Edmonton
Canada/Newfoundland	America/St_Johns
Canada/Pacifico	America/Vancouver
Canada/Saskatchewan	America/Regina
Canada/Yukon	America/Los_Angeles
Cile/Continentale	America/Santiago
Cile/EasterIsland	Pacifico/Isola di Pasqua
Cuba	America/L'Avana
EET	America/Indianapolis
EST5EDT	America/New_York
Egitto	Africa/Cairo
Eire	Europa/Dublino
Etc/GMT	GMT
Etc/GMT0	GMT
Etc/Greenwich	GMT
Etc/UCT	UTC
Etc/UTC	UTC
Etc/Universale	UTC

Valore di sistema	Oggetto stringa Java
Etc/Zulu	UTC
Europa/Amsterdam	Europa/Amsterdam
Europa/Andorra	Europa/Andorra
Europa/Atene	Europa/Atene
Europa/Belfast	Europa/Londra
Europa/Belgrado	Europa/Belgrado
Europa/Berlino	Europa/Berlino
Europa/Bratislava	Europa/Praga
Europa/Bruxelles	Europa/Bruxelles
Europa/Bucarest	Europa/Bucarest
Europa/Budapest	Europa/Budapest
Europa/Chisinau	Europa/Chisinau
Europa/Copenhagen	Europa/Copenhagen
Europa/Dublino	Europa/Dublino
Europa/Gibilterra	Europa/Gibilterra
Europa/Helsinki	Europa/Helsinki
Europa/Istanbul	Europa/Istanbul
Europa/Kaliningrad	Europa/Kaliningrad
Europa/Kiev	Europa/Kiev
Europa/Lisbona	Europa/Lisbona
Europa/Ljubljana	Europa/Belgrado
Europa/Londra	Europa/Londra
Europa/Lussemburgo	Europa/Lussemburgo
Europa/Madrid	Europa/Madrid
Europa/Malta	Europa/Malta
Europa/Minsk	Europa/Minsk
Europa/Monaco	Europa/Monaco
Europa/Mosca	Europa/Mosca
Europa/Oslo	Europa/Oslo
Europa/Parigi	Europa/Parigi
Europa/Praga	Europa/Praga
Europa/Riga	Europa/Riga
Europa/Roma	Europa/Roma
Europa/Samara	Europa/Samara
Europa/San_Marino	Europa/Roma
Europa/Sarajevo	Europa/Belgrado
Europa/Simferopol	Europa/Simferopol
Europa/Skopje	Europa/Belgrado
Europa/Sofia	Europa/Sofia
Europa/Stoccolma	Europa/Stoccolma
Europa/Tallinn	Europa/Tallinn

Valore di sistema	Oggetto stringa Java
Europa/Tirana	Europa/Tirana
Europa/Vaduz	Europa/Vaduz
Europa/Vaticano	Europa/Roma
Europa/Vienna	Europa/Vienna
Europa/Vilnius	Europa/Vilnius
Europa/Varsavia	Europa/Varsavia
Europa/Zagabria	Europa/Belgrado
Europa/Zurigo	Europa/Zurigo
Azienda	GMT
GB	Europa/Londra
GB-Eire	Europa/Londra
GMT	GMT
GMT0	GMT
Greenwich	GMT
HST	Pacifico/Honolulu
Hongkong	Asia/Hong_Kong
Islanda	Atlantico/Reykjavik
Indiano/Antananarivo	Indiano/Antananarivo
Indiano/Chagos	Indiano/Chagos
Indiano/Christmas	Indiano/Christmas
Indiano/Cocos	Indiano/Cocos
Indiano/Comoro	Indiano/Comoro
Indiano/Kerguelen	Indiano/Kerguelen
Indiano/Mahé	Indiano/Mahé
Indiano/Maldives	Indiano/Maldives
Indiano/Mauritius	Indiano/Mauritius
Indiano/Mayotte	Indiano/Mayotte
Indiano/Reunion	Indiano/Reunion
Iran	Asia/Teheran
Israele	Asia/Gerusalemme
Giamaica	America/Giamaica
Giappone	Asia/Tokyo
Libia	Africa/Tripoli
MET	Europa/Parigi
MST	America/Phoenix
MST7MDT	America/Denver
Messico/BajaNorte	America/Tijuana
Messico/BajaSur	America/Mazatlan
Messico/Generale	America/Città_del_Messico
NZ	Pacifico/Auckland
NZ-CHAT	Pacifico/Chatham

Valore di sistema	Oggetto stringa Java
Navajo	America/Denver
PRC	Asia/Shanghai
PST	America/Los_Angeles
PST8PDT	America/Los_Angeles
Pacifico/Apia	Pacifico/Apia
Pacifico/Auckland	Pacifico/Auckland
Pacifico/Chatham	Pacifico/Chatham
Pacifico/Isola di Pasqua	Pacifico/Isola di Pasqua
Pacifico/Efate	Pacifico/Efate
Pacifico/Enderbury	Pacifico/Enderbury
Pacifico/Fakaofu	Pacifico/Fakaofu
Pacifico/Fiji	Pacifico/Fiji
Pacifico/Funafuti	Pacifico/Funafuti
Pacifico/Galapagos	Pacifico/Galapagos
Pacifico/Gambier	Pacifico/Gambier
Pacifico/Guadalcanal	Pacifico/Guadalcanal
Pacifico/Guam	Pacifico/Guam
Pacifico/Honolulu	Pacifico/Honolulu
Pacifico/Kiritimati	Pacifico/Kiritimati
Pacifico/Kosrae	Pacifico/Kosrae
Pacifico/Majuro	Pacifico/Majuro
Pacifico/Marchesi	Pacifico/Marchesi
Pacifico/Nauru	Pacifico/Nauru
Pacifico/Niue	Pacifico/Niue
Pacifico/Norfolk	Pacifico/Norfolk
Pacifico/Noumea	Pacifico/Noumea
Pacifico/Pago_Pago	Pacifico/Pago_Pago
Pacifico/Palau	Pacifico/Palau
Pacifico/Pitcairn	Pacifico/Pitcairn
Pacifico/Ponape	Pacifico/Ponape
Pacifico/Port_Moresby	Pacifico/Port_Moresby
Pacifico/Rarotonga	Pacifico/Rarotonga
Pacifico/Saipan	Pacifico/Saipan
Pacifico/Samoa	Pacifico/Pago_Pago
Pacifico/Tahiti	Pacifico/Tahiti
Pacifico/Tarawa	Pacifico/Tarawa
Pacifico/Tongatapu	Pacifico/Tongatapu
Pacifico/Truk	Pacifico/Truk
Pacifico/Wake	Pacifico/Wake
Pacifico/Wallis	Pacifico/Wallis
Polonia	Europa/Varsavia

Valore di sistema	Oggetto stringa Java
Portogallo	Europa/Lisbona
ROC	Asia/Taipei
ROK	Asia/Seul
Singapore	Asia/Singapore
SystemV//AST4ADT	America/Thule
SystemV/CST6CDT	America/Chicago
SystemV/EST5EDT	America/New_York
SystemV/MST7MDT	America/Denver
SystemV/PST8PDT	America/Los_Angeles
SystemV/YST9YDT	America/Anchorage
Turchia	Europa/Istanbul
UCT	UTC
Stati Uniti/Alaska	America/Anchorage
Stati Uniti/Isole Aleutine	America/Adak
Stati Uniti/Arizona	America/Phoenix
Stati Uniti/Centrale	America/Chicago
Stati Uniti/East-Indiana	America/Indianapolis
Stati Uniti/Oriente	America/New_York
Stati Uniti/Hawaii	Pacifico/Honolulu
Stati Uniti/Michigan	America/New_York
Stati Uniti/Zona montuosa	America/Denver
Stati Uniti/Pacifico	America/Los_Angeles
Stati Uniti/Pacific-New	America/Los_Angeles
Stati Uniti/Samoa	Pacifico/Pago_Pago
UTC	UTC
Universale	UTC
W-SU	Europa/Mosca
WET	Europa/Londra
Zulu	UTC

Locali Java

Una locale è un'area del mondo geografica o politica che condivide la stessa lingua e gli stessi costumi. In Java^(TM), la classe Locale rappresenta una locale.

Locali Java supportate

IBM Developer Kit per Java supporta queste locali. I lavori iSeries e CNTRYID determinano la locale predefinita. Consultare proprietà di sistema Java per ulteriori dettagli.

Nome locale in JDK 1.1.6	Nome locale ISO	Lingua/Paese o regione
ar	ar_EG	Arabo/Egitto
be	be_BY	Bielorusso/Bielorussia
bg	bg_BG	Bulgaro/Bulgaria

Nome locale in JDK 1.1.6	Nome locale ISO	Lingua/Paese o regione
ca	ca_ES	Catalano/Spagna
cs	cs_CZ	Ceco/Repubblica Ceca
da	da_DK	Danese/Danimarca
de	de_DE	Tedesco/Germania
de_AT	de_AT	Tedesco/Austria
de_CH	de_CH	Tedesco/Svizzera
el	el_GR	Greco/Grecia
en	en_US	Inglese/USA
en_AU	en_AU	Inglese/Australia
en_CA	en_CA	Inglese/Canada
en_GB	en_GB	Inglese/Gran Bretagna
en_IE	en_IE	Inglese/Irlanda
en_NZ	en_NZ	Inglese/Nuova Zelanda
en_ZA	en_ZA	Inglese/Sud Africa
es	es_ES	Spagnolo/Spagna
es_AR	es_AR	Spagnolo/Argentina
es_BO	es_BO	Spagnolo/Bolivia
es_CL	es_CL	Spagnolo/Cile
es_CR	es_CR	Spagnolo/Costa Rica
es_DO	es_DO	Spagnolo/Repubblica Dominicana
es_EC	es_EC	Spagnolo/Ecuador
es_GT	es_GT	Spagnolo/Guatemala
es_HN	es_HN	Spagnolo/Honduras
es_MX	es_MX	Spagnolo/Messico
es_NI	es_NI	Spagnolo/Nicaragua
es_PA	es_PA	Spagnolo/Panama
es_PE	es_PE	Spagnolo/Perù
es_PR	es_PR	Spagnolo/Portorico
es_PY	es_PY	Spagnolo/Paraguay
es_SV	es_SV	Spagnolo/El Salvador
es_UY	es_UY	Spagnolo/Uruguay
es_VE	es_VE	Spagnolo/Venezuela
et	et_EE	Estone/Estonia
fi	fi_FI	Finlandese/Finlandia
fr	fr_FR	Francese/Francia
fr_BE	fr_BE	Francese/Belgio
fr_CA	fr_CA	Francese/Canada
fr_CH	fr_CH	Francese/Svizzera
hr	hr_HR	Croato/Croazia
hu	hu_HU	Ungherese/Ungheria
is	is_IS	Islandese/Islanda

Nome locale in JDK 1.1.6	Nome locale ISO	Lingua/Paese o regione
it	it_IT	Italiano/Italia
it_CH	it_CH	Italiano/Svizzera
iw	iw_IL	Ebraico/Israele
ja	ja_JP	Giapponese/Giappone
ko	ko_KR	Coreano/Corea
lt	lt_LT	Lituano/Lituania
lv	lv_LV	Lettone/Lettonia
mk	mk_MK	Macedone/Macedonia
nl	nl_NL	Olandese/Paesi Bassi
nl_BE	nl_BE	Olandese/Belgio
no	no_NO_B	Norvegese/Norvegia
no_NO_NY	no_NO_NY	Norvegese/Norvegia, NY
pl	pl_PL	Polacco/Polonia
pt	pt_PT	Portoghese/Portogallo
ro	ro_RO	Rumeno/Romania
ru	ru_RU	Russo/Russia
sh	sh_SP	Serbo-Croato/Serbia
sk	sk_SK	Slovacco/Slovacchia
sl	sl_SI	Sloveno/Slovenia
sq	sq_AL	Albanese/Albania
sr	sr_SP	Serbo/Serbia
sv	sv_SE	Svedese/Svezia
tr	tr_TR	Turco/Turchia
uk	uk_UA	Ucraino/Ucraina
zh	zh_CN	Cinese semplificato
zh_TW	zh_TW	Cinese tradizionale

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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
//*****
// 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));
}
}

```

Per ulteriori informazioni, consultare [Creare un programma Java \(TM\) internazionalizzato](#).

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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

//*****
// 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));
    }
}

```

Per ulteriori informazioni, consultare [Creare un programma Java \(TM\) internazionalizzato](#).

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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
//*****  
// 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"));  
    }  
}
```

Per ulteriori informazioni, consultare [Creare un programma Java ^{\(TM\)} internazionalizzato](#).

Codifiche del carattere Java

Internamente, JVM ovvero Java^(TM) 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. Consultare la tabella CCSID iSeries e valori `file.encoding` per una descrizione della correlazione di `file.encoding` a CCSID sul server iSeries.

Quando un programma Java legge i dati, ci si aspetta che essi 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 ricevuti tramite socket TCP/IP (Transmission Control Protocol/Internet Protocol) utilizzando il pacchetto `.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 sul server iSeries, JVM esegue una conversione sui dati da convertire dalla normale codifica di carattere di `file.encoding` in un CCSID corrispondente al CCSID del lavoro iSeries. 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 Javaclasses `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 nel o dal database DB2/400, tramite le API JDBC, vengono convertiti nel o dal database iSeries CCSID.

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

Per ulteriori informazioni sull'internazionalizzazione, consultare Globalizzazione OS/400.

E' inoltre possibile consultare Internazionalizzazione di Sun Microsystems, Inc. per ulteriori informazioni.

Valori `file.encoding` e CCSID iSeries: Questa tabella mostra la relazione tra i possibili valori `file.encoding` e il CCSID (coded character set identifier) iSeries più strettamente corrispondente.

<code>file.encoding</code>	CCSID	Descrizione
Big5	950	T-Cinese BIG-5 ASCII a 8-bit
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 Spagnolo, 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

file.encoding	CCSID	Descrizione
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	856	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
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
Cp943	943	Dati PC misti in giapponese per open env
Cp943C	943	Dati PC misti in giapponese per open env
Cp948	948	Cinese tradizionale ASCII a 8-bit IBM
Cp949	949	Coreano ASCII a 8-bit KSC5601
Cp950	950	T-Cinese BIG-5 ASCII a 8-bit
Cp964	964	EUC Cinese tradizionale
Cp970	970	EUC Coreano
Cp1006	1006	ISO Urdu a 8-bit
Cp1025	1025	IBM EBCDIC Cirillico

file.encoding	CCSID	Descrizione
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	1124	ISO Ucraina a 8-bit
Cp1250	1250	MS-Win Latino-2
Cp1251	1251	MS-Win Cirillico
Cp1252	1252	MS-Win Latino-1
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	33722	EUC per Giapponese
EUC_KR	970	EUC per Coreano
EUC_TW	964	EUC per Cinese tradizionale
GB2312	1381	S-Cinese GB ASCII a 8-bit
GBK	1386	Nuovo Cinese semplificato ASCII 9 a 8-bit
ISO2022CN_CNS	Nessuno disponibile	ASCII a 7-bit per Cinese tradizionale
ISO2022CN_GB	Nessuno disponibile	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 ISO Latino-1
ISO8859_2	912	ISO 8859-2 ISO Latino-2
ISO8859_3	913	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)

file.encoding	CCSID	Descrizione
JIS0201	897	Standard aziendale Giapponese X0201
JIS0208	952	Standard aziendale Giapponese X0208
JIS0212	953	Standard aziendale Giapponese X0212
Johab	Nessuno disponibile	Codifica Hangul di composizione Coreana (completa)
K018_R		Cirillico
KSC5601	949	Coreano ASCII a 8-bit
MS874	874	MS-Win Thailandia
SJIS	932	Giapponese ASCII a 8-bit
TIS620	874	Standard aziendale thailandese 620
UTF8	1208	UTF-8 (CCSID IBM 1208, che non è ancora disponibile sul server iSeries)
Unicode	13488	UNICODE, UCS-2
UnicodeBig	13488	Lo stesso di Unicode
UnicodeBigUnmarked		Unicode con contrassegno senza ordine byte
UnicodeLittle		Unicode con ordine byte little-endian
UnicodeLittleUnmarked		UnicodeLittle con nessun contrassegno di ordine 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 (coded character set identifier) iSeries all'avvio di JVM ovvero Java^(TM) virtual machine.

CCSID iSeries	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

CCSID iSeries	File.encoding predefinito	Descrizione
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
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)

CCSID iSeries	File.encoding predefinito	Descrizione
28709	Cp948	Parte SBCS del Cinese tradizionale EBCDIC misto (CCSID 937)

Compatibilità tra rilasci

I file di classe Java^(TM) sono compatibili con le versioni successive (JDK 1.1.x -> 1.2.x -> 1.3.x >> -> 1.4.x <<) fin tanto che questi non utilizzano quelle poche caratteristiche che Sun ha interrotto o per le quali ha modificato il supporto (consultare la documentazione Sun). I file di classe sono anche compatibili con versioni precedenti (>> 1.4.x -> 1.3.x -> 1.2.x -> 1.1.x <<) fin tanto che i programmi utilizzano solo le caratteristiche Java disponibili nel livello precedente JDK. Questo risulta vero anche sul server iSeries, poiché è una implementazione compatibile. Consultare The Source for Java Technology java.sun.com



per informazioni relative alla disponibilità rilascio per rilascio.

Quando i programmi Java su un server iSeries sono ottimizzati utilizzando il comando CRTJVAPGM (Creazione programma Java), un JVAPGM viene collegato al file di classe. La struttura interna di questi JVAPGM è stata modificata su V4R4. Ciò significa che i JVAPGM creati prima della V4R4 non sono validi su V4R4 e su rilasci successivi. E' necessario che questi vengano ricreati. Se non è intrapresa alcuna azione, il sistema crea automaticamente un JVAPGM allo stesso livello di ottimizzazione di quello precedente. E', comunque, consigliabile effettuare un CRTJVAPGM manualmente, specialmente con i file JAR o ZIP. Questo produce l'ottimizzazione migliore con la minore dimensione del programma.

Per migliori prestazioni al livello di ottimizzazione 40, è consigliabile effettuare CRTJVAPGM su ogni modifica del rilascio OS/400 o della versione JDK. Questo è vero in special modo se la funzione JDKVER viene utilizzata su CRTJVAPGM, poiché ciò ottiene come risultato l'allineamento dei metodi di Sun JDK nel JVAPGM. Ciò rappresenta un grande vantaggio per le prestazioni. Se, tuttavia, vengono effettuate modifiche nel JDK su rilasci successivi che invalidano quegli allineamenti, è possibile che i programmi vengano eseguiti più lentamente rispetto alle ottimizzazioni più basse. Questo si verifica perché è necessario eseguire un codice con caratteri speciali per ottenere l'operazione appropriata.

Consultare Prestazioni del tempo di esecuzione Java per ulteriori informazioni dettagliate sulle prestazioni.

Accedere all'IBM Developer Kit per Java

Con IBM Developer Kit per Java^(TM), è possibile tramite i programmi Java accedere ai file database in tre modi:

- Unità di controllo JDBC spiega come l'unità di controllo JDBC di IBM Developer Kit per Java consente ai programmi Java di accedere ai file di database.
- Supporto SQLJ spiega come IBM Developer Kit per Java consente all'utente di utilizzare le istruzioni SQL incorporate nella propria applicazione Java.
- >> Nella Routine SQL Java viene spiegato come è possibile utilizzare le procedure memorizzate e le funzioni definite dall'utente Java per accedere ai programmi Java.<<

Accedere al proprio database iSeries con l'unità di controllo JDBC di IBM Developer Kit per Java

L'unità di controllo JDBC di IBM Developer Kit per Java^(TM), conosciuta anche come unità di controllo "nativa", fornisce l'accesso programmatico ai file del database iSeries. 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, reperire i risultati e trasmettere le modifiche verso il database. >> E' inoltre possibile utilizzare l'API JDBC per interagire con più risorse dati in un ambiente eterogeneo, distribuito.

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.

Per utilizzare l'unità di controllo JDBC, esaminare quanto segue:

Introduzione a JDBC

E' possibile seguire il supporto didattico per scrivere un programma JDBC ed eseguirlo sul proprio server iSeries.

Collegamenti

Un programma di applicazione può avere più collegamenti allo stesso tempo. E' possibile rappresentare un collegamento ad una sorgente dati in JDBC utilizzando un oggetto Connection. Gli oggetti Statement si creano tramite gli oggetti Connection, per l'elaborazione delle istruzioni SQL rispetto al database.

DatabaseMetaData

L'interfaccia DatabaseMetaData viene utilizzata dagli strumenti e dai server dell'applicazione per stabilire come interagire con una sorgente dati assegnata. Le applicazioni possono inoltre utilizzare i metodi DatabaseMetaData per ottenere informazioni su una sorgente dati specifica.

Eccezioni

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.

Transazioni

Una transazione è un'unità logica di lavoro. Le transazioni sono utilizzate per fornire integrità di dati, una semantica corretta dell'applicazione e una visualizzazione coerente di dati durante l'accesso simultaneo. Tutte le unità compatibili con JDBC devono supportare le transazioni.

Tipi Statement

L'interfaccia Statement e le relative sottoclassi PreparedStatement e CallableStatement vengono utilizzate per elaborare i comandi SQL rispetto al database. Le istruzioni SQL determinano la creazione di oggetti ResultSet.

ResultSet

L'interfaccia ResultSet fornisce l'accesso ai risultati generati eseguendo delle interrogazioni. E' possibile pensare i dati di un ResultSet come una tabella con un numero specifico di colonne e di righe. Per impostazione predefinita, le righe della tabella vengono ripristinate in sequenza. All'interno di una riga, è possibile accedere ai valori della colonna in qualsiasi ordine.

Creare lotti di oggetti JDBC

Dal momento che molti oggetti utilizzati in JDBC, come Connection, Statement e ResultSet, sono dispendiosi da creare, è possibile ottenere benefici significativi sulle prestazioni utilizzando la creazione di lotti di oggetti JDBC. Con la creazione di tali lotti, è possibile riutilizzare questi oggetti invece di crearli ogni volta che sono necessari.

Aggiornamenti batch

Il supporto di aggiornamento batch consente di passare più aggiornamenti al database come una singola transazione tra il programma utente e il database. Gli aggiornamenti batch possono migliorare in modo significativo le prestazioni quando è necessario eseguire più aggiornamenti contemporaneamente.

Tipi di dati avanzati

Esistono numerosi tipi di dati denominati SQL3 forniti nel database iSeries. I tipi di dati SQL3

forniscono all'utente estrema flessibilità. Essi sono ideali per gli oggetti Java serializzati di memorizzazione, documenti XML (Extensible Markup Language) e dati multimediali come canzoni, immagini del prodotto, fotografie di impiegati e filmati. I tipi di dati SQL3 includono quanto segue:

- Tipi "distinct"
- Large Object come Binary Large Object, Character Large Object e Double Byte Character Large Object
- Datalink

RowSet


La specifica RowSet è progettata per essere più di una framework rispetto all'implementazione effettiva. Le interfacce RowSet definiscono una serie di funzionalità principale di cui dispongono tutti i RowSet.


Transazioni distribuite

La JTA (Java Transaction API) dispone di un supporto per transazioni complesse. Essa fornisce inoltre il supporto per separare le transazioni dagli oggetti Connection. JTA e JDBC lavorano insieme per separare le transazioni dagli oggetti Connection e consentono che un singolo collegamento operi contemporaneamente su più transazioni. Viceversa, ciò consente all'utente di avere più collegamenti che operano su una singola transazione.

Suggerimenti sulle prestazioni

E' possibile ottenere le migliori prestazioni possibili dalle proprie applicazioni JDBC con questi suggerimenti sulle prestazioni.

Per ulteriori informazioni su JDBC, esaminare la documentazione JDBC  da Sun Microsystem, Inc.

Per ulteriori informazioni sull'unità di controllo JDBC nativa iSeries, consultare la pagina Web del JDBC di IBM Developer Kit per Java  .



Introduzione a JDBC

» L'unità di controllo JDBC (JavaTM Database Connectivity) fornita con il Developer Kit per Java viene chiamata unità di controllo JDBC del Developer Kit per Java. Questa unità è inoltre comunemente conosciuta come unità di controllo JDBC nativa.

Per selezionare quale unità 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 l'unità di controllo JDBC nativa per le prestazioni. Ciò include la maggior parte dei servlet delle soluzioni JSP (JavaServer Page) e le applicazioni scritte per un'esecuzione locale su un server iSeries.
- I programmi che devono collegarsi ad un server iSeries remoto utilizzano l'unità di controllo JDBC di Toolbox. Toolbox è un'implementazione efficace di JDBC ed è fornito come parte del Toolbox per Java. Essendo puro Java, l'unità di controllo JDBC di Toolbox è semplice da impostare per i client e richiede un'impostazione del server minima.
- I programmi eseguiti su un server iSeries e che necessitano il collegamento ad un database remoto diverso da iSeries utilizzano l'unità di controllo JDBC nativa e impostano un collegamento DRDA (Distributed Relational Database Architecture) a quel server remoto.

Per informazioni preliminari su JDBC, esaminare quanto segue:

Tipi di unità di controllo JDBC

Questo argomento definisce i tipi di unità di controllo JDBC. Tali tipi vengono definiti per suddividere in categorie la tecnologia utilizzata per collegarsi al database.

Requisiti

Questo argomento indica i requisiti necessari per accedere a quanto segue:

- JDBC principale
- Pacchetto facoltativo JDBC 2.0
- JTA (Java Transaction API)

Supporto didattico JDBC

Questa è una prima fase importante verso la scrittura di un programma JDBC e la relativa esecuzione su un server iSeries con l'unità di controllo JDBC nativa.



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

Tipo 1: Le unità 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 le unità di controllo ODBC esistono per molte piattaforme RDBMS (Relational Database Management System). JNI (Java Native Interface) viene utilizzata per richiamare funzioni ODBC dall'unità di controllo JDBC.

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

Tipo 2: Le unità di 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. Le unità di controllo di tipo 2 sono generalmente più rapide rispetto a quelle di tipo 1.

Affinché le unità di controllo di tipo 2 funzionino, è necessario il codice binario installato e configurato. Anche un'unità di controllo di tipo 2 utilizza la JNI. Non è possibile utilizzare tale unità in un'applet in quanto le applet non possono caricare un codice nativo. E' possibile che un'unità di controllo JDBC di tipo 2 richieda l'installazione di un software di rete DBMS (Database Management System).

L'unità di controllo JDBC del Developer Kit per Java è un'unità di controllo JDBC di tipo 2.

Tipo 3: Queste unità 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.

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

Tipo 4: Un'unità 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'unità di controllo JDBC di tipo 4.

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

L'unità di controllo JDBC di Toolbox è un'unità di controllo JDBC di tipo 4, che indica che l'API è un'unità di controllo pura del protocollo di rete Java.



Requisiti JDBC: ➤ Prima di scrivere e impiegare le proprie applicazioni JDBC, potrebbe essere necessario installare quanto segue:

- JDBC principale
- Pacchetto facoltativo JDBC 2.0
- JTA (Java Transaction API)

JDBC principale: Per l'accesso di JDBC (JavaTM Database Connectivity) principale al database locale, non esistono requisiti. L'intero supporto è incorporato, pre-installato e configurato.

Note:

- in passato, era necessario aggiungere un collegamento simbolico per JDK 1.2 e versioni successive. Questo requisito è stato eliminato con V4R5 PTF SF65439 e V5R1 PTF SI00959.
- in passato, era necessario assicurarsi che esistesse almeno una voce nell'indirizzario del database relazionale per collegarsi ad un sistema (normalmente *LOCAL per collegarsi al database locale). Anche questo requisito è stato abolito. Se non esiste alcuna voce per il database locale, essa viene creata utilizzando il nome del sistema quando si accede al sistema locale. E' ancora necessario configurare le voci dell'indirizzario del database relazionale per qualsiasi sistema remoto a cui ci si intende collegare con l'unità di controllo JDBC nativa.

Pacchetto facoltativo JDBC 2.0: Se è necessario utilizzare le classi del pacchetto facoltativo JDBC 2.0, è necessario includere il file jdbc2_0-stdext.jar nel proprio classpath. Questo file JAR (Java ARchive) contiene tutte le interfacce standard necessarie per scrivere la propria applicazione in modo da utilizzare il pacchetto facoltativo JDBC 2.0. Per aggiungere il file JAR al proprio classpath delle estensioni, creare un collegamento simbolico dall'indirizzario delle estensioni UserData nell'ubicazione del file JAR. E' necessario eseguire quest'operazione soltanto una volta; il file JAR del pacchetto facoltativo JDBC 2.0 è sempre disponibile nelle proprie applicazioni durante il tempo di esecuzione. Utilizzare il seguente comando per aggiungere il pacchetto facoltativo al classpath delle estensioni:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jdbc2_0-stdext.jar')
NEWLNK('/QIBM/UserData/Java400/ext/jdbc2_0-stdext.jar')
```

Nota: questo requisito è soltanto per JDK 1.2 e 1.3. Dal momento che JDK 1.4 è il primo rilascio con un supporto JDBC 3.0, tutto il JDBC (cioè, il JDBC principale e il pacchetto facoltativo) si sposta nel file JAR del tempo di esecuzione JDK di base che il proprio programma rileva sempre.

JTA (Java Transaction API): Se è necessario utilizzare JTA (Java Transaction API) nella propria applicazione, è necessario includere il file jta-spec1_0_1.jar nel proprio classpath. Questo file JAR contiene tutte le interfacce standard necessarie per scrivere la propria applicazione in modo da utilizzare JTA. Per aggiungere il file JAR al proprio classpath delle estensioni, creare un collegamento simbolico dall'indirizzario delle estensioni UserData nell'ubicazione del file JAR. Questa è un'operazione da effettuare una sola volta e, quando viene completata, il file JAR JTA è sempre disponibile nella propria applicazione durante il tempo di esecuzione. Utilizzare il seguente comando per aggiungere JTA al classpath delle estensioni:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jta-spec1_0_1.jar')
NEWLNK('/QIBM/UserData/Java400/ext/jta-spec1_0_1.jar')
```


Conformità JDBC: L'unità di controllo JDBC nativa è conforme a tutte le specifiche JDBC pertinenti. Il livello di conformità dell'unità di controllo JDBC non dipende dal rilascio OS/400, ma dal rilascio JDK che si utilizza. Il livello di conformità dell'unità di controllo JDBC nativa per i vari JDK viene elencato come segue:

Rilascio JDK	Livello di conformità dell'unità di controllo JDBC
JDK 1.1	Questo JDK è conforme a JDBC 1.0.
JDK 1.2	Questo JDK è conforme a JDBC 2.0 e supporta il pacchetto facoltativo JDBC 2.1.
JDK 1.3	Questo JDK è conforme a JDBC 2.0 e supporta il pacchetto facoltativo JDBC 2.1 (non esiste alcuna modifica correlata a JDBC per JDK 1.3).
JDK 1.4	Questo JDK è conforme a JDBC 3.0, ma il pacchetto facoltativo JDBC non esiste più (il relativo supporto fa ora parte del JDK principale).



Supporto didattico JDBC: ➤ Quello che segue è un supporto didattico per la scrittura di un programma JDBC (JavaTM Database Connectivity) e per la relativa esecuzione su un server iSeries con l'unità di controllo JDBC nativa. Esso è progettato per mostrare all'utente le fasi basilari necessarie affinché il proprio programma esegua JDBC.

Il programma di esempio crea una tabella e la popola con alcuni dati. Esso elabora un'interrogazione 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 il programma di esempio e incollarlo in un file sulla propria 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 propria stazione di lavoro al proprio server iSeries. Da una richiesta comandi, immettere i seguenti comandi:

```
ftp <iSeries server name>
<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 nel proprio server iSeries se lo si trasferisce nell'IFS (Integrated File System). Gli 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 il 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. E' possibile effettuare ciò con il seguente comando:

```
ADDENVVAR ENVVAR(CLASSPATH) VALUE('/home/cujo:/QIBM/ProdData/Java400/jdk13/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) prop(BasicJDBC)
java BasicJDBC
```

E' 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 iSeries 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.
```

Riferimenti: Per ulteriori informazioni su Java e JDBC, consultare le seguenti risorse:

- Sito Web esterno dell'unità di controllo JDBC 
- Sito web esterno dell'unità di controllo JDBC di Toolbox 
- Pagina JDBC di Sun 
- Forum Java/JDBC per iSeries e per utenti iSeries
- Indirizzo e-mail di JDBC IBM



Esempio: JDBC:  Questo è un esempio su come utilizzare il programma BasicJDBC.

Esempio: BasicJDBC

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////
//
// Esempio BasicJDBC. Questo programma utilizza l'unità di controllo JDBC nativa
// affinché Developer Kit per Java crei una tabella semplice ed elabori un'interrogazione
// che visualizzi i dati in tale tabella.
//
// Sintassi del comando:
//   BasicJDBC
//
////////////////////////////////////
//
// Questo sorgente è un esempio di unità di controllo JDBC IBM Developer per Java.
// L'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 dall'IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. L'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 for 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 object-oriented. 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 creato 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 interrogazione
// 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 a 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 utilizzare 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'unità di controllo JDBC disponibile
        // per gestire una richiesta di collegamento utente. La
seguinte riga
        // fa in modo che l'unità di controllo JDBC venga
caricata 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 qqpl.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 viene utilizzato per creare un
        // oggetto 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 qqpl.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 qqpl.basicjdbc(id int, name char(15))");

        // Utilizzare l'oggetto statement per popolare la tabella con alcuni dati.
        s.executeUpdate("insert into qqpl.basicjdbc values(1, 'Frank Johnson')");
        s.executeUpdate("insert into qqpl.basicjdbc values(2, 'Neil Schwartz')");
        s.executeUpdate("insert into qqpl.basicjdbc values(3, 'Ben Rodman')");
        s.executeUpdate("insert into qqpl.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 un'interrogazione 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 un'interrogazione SQL. Le
    // interrogazioni restituiscono oggetti ResultSet usati per consultare i
    // dati forniti dall'interrogazione.
    ResultSet rs = s.executeQuery("select * from qqpl.basicjdbc");

    // Visualizzare la parte superiore della 'tabella' e iniziale 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();
    }
}
}

```




Utilizzare JNDI per gli esempi: ➤ I DataSource lavorano di pari passo con JNDI ovvero Java^(TM) 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. JNDI 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.

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. Fare clic su **Continua** per scaricare JNDI 1.2.1. Viene visualizzato un accordo della licenza.
3. Fare clic su **Accetta** e quindi fare clic su **Contesto FS** per visualizzare le opzioni di scaricamento per il supporto del contesto JNDI.
4. Scaricare fscontext.zip e decomprimere il file dalla propria stazione di lavoro.
5. Utilizzare FTP per trasferire fscontext.jar e providerutil.jar sul proprio sistema e inserirli in /QIBM/UserData/Java400/ext. Questo è l'indirizzario delle estensioni e i file JAR che si inseriscono in questo contesto vengono trovati automaticamente quando si esegue la propria applicazione (cioè, non sono necessari nel proprio 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. E' 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. <<

Collegamenti

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

E' 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.

Per dettagli su come ottenere i collegamenti, consultare le seguenti sezioni:

DriverManager

Il DriverManager è una classe statica che gestisce la serie di unità di controllo JDBC disponibili affinché siano utilizzate da un'applicazione.

Proprietà Connection

La tabella elenca valide proprietà di collegamento dell'unità di controllo JDBC, i relativi valori e descrizioni.

Utilizzare i DataSource con UDBDataSource

E' possibile disporre un DataSource con la classe UDBDataSource impostandolo in modo che abbia proprietà specifiche e quindi collegandolo ad un servizio dell'indirizzario tramite l'utilizzo di JNDI (Java Naming and Directory Interface).

Proprietà DataSource

La tabella elenca valide proprietà DataSource, i relativi valori e descrizioni.

Altre implementazioni DataSource

Esistono altre implementazioni dell'interfaccia DataSource fornite con l'unità di controllo JDBC nativa. La loro funzione è di servire solo come ponte finché non vengano adottati l'UDBDataSource e le relative funzioni.

Una volta ottenuto il collegamento, è possibile utilizzarlo per completare le seguenti attività JDBC:

- Creare vari tipi di oggetti Statement per interagire con il database.
- Controllare le transazioni rispetto al database.
- Reperire metadati riguardo al database.



DriverManager: ➤ DriverManager è una classe statica fornita da JDK ovvero Java^(TM) Developer Kit. Esso è responsabile di gestire la serie delle unità di controllo JDBC (Java Database Connectivity) disponibili per l'utilizzo da parte di un'applicazione. Le applicazioni possono utilizzare più unità di controllo JDBC contemporaneamente se necessario. Un programma di applicazione specifica un'unità di controllo JDBC utilizzando un URL (Uniform Resource Locator). Passando un URL per un'unità di controllo JDBC specifica al DriverManager, l'applicazione informa il DriverManager sul tipo di collegamento JDBC che sarebbe opportuno restituire all'applicazione.

Prima di questa operazione, DriverManager deve conoscere le unità 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 l'unità di controllo JDBC nativa:

Esempio: caricare l'unità di controllo JDBC nativa

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Caricare l'unità di controllo JDBC nativa nel DriverManager per
// renderla disponibile per le richieste getConnection.
```

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

Le unità di controllo JDBC sono progettate 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, l'unità di controllo JDBC nativa è disponibile per il DriverManager da gestire. La seguente riga del codice richiede un oggetto Connection che utilizza l'URL di JDBC nativa:

Esempio: richiedere un oggetto Connection

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Ottenere un collegamento che utilizza l'unità di controllo JDBC nativa.  
  
Connection c = DriverManager.getConnection("jdbc:db2:*local");
```

Il formato più semplice dell'URL JDBC è una lista di tre valori separati dai due punti. Il primo valore nella lista rappresenta il protocollo che è sempre jdbc per gli URL JDBC. Il secondo valore è il sottoprotocollo e db2 viene utilizzato per specificare l'URL JDBC nativo. Il terzo valore è il nome del sistema per stabilire il collegamento ad un sistema specifico. E' possibile utilizzare due valori specifici per collegarsi al database locale. Essi sono *LOCAL e localhost (entrambi non sono sensibili al maiuscolo e al minuscolo). E' 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 a 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: quando non viene specificato, l'ID utente e la parola d'ordine correntemente utilizzati per il collegamento vengono anche utilizzati per stabilire la connessione al database.

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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Ottenere un collegamento che utilizza l'unità di controllo JDBC nativa.  
  
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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Ottenere un collegamento che utilizza l'unità di controllo JDBC nativa.  
  
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 Connection per una lista completa delle proprietà di collegamento per l'unità di controllo JDBC nativa.

Proprietà URL: Un altro modo per specificare le proprietà è inserirle in una lista sull'oggetto URL stesso. Ogni proprietà nella lista è separata da un punto e virgola e la lista 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: specificare proprietà URL

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Ottenere un collegamento che utilizza l'unità di controllo JDBC nativa.  
  
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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Ottenere un collegamento che utilizza l'unità di controllo JDBC nativa.  
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.

Consultare i seguenti esempi per ulteriori informazioni:

- Utilizzare JDBC nativo e JDBC di Toolbox contemporaneamente
- Proprietà Access
- ID utente e parola d'ordine non validi



Esempio: utilizzare JDBC nativo e JDBC di Toolbox contemporaneamente: ➤ Questo è un esempio della modalità di utilizzo del collegamento JDBC nativo e JDBC di Toolbox in un programma.

Esempio: utilizzare JDBC nativo e JDBC di Toolbox contemporaneamente

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////  
//  
// Esempio di GetConnections.  
//  
// E' evidente che questo programma è in grado di utilizzare entrambe le unità di  
// controllo JDDBS in un programma contemporaneamente. In questo programma vengono  
// creati Due oggetti Connection. Uno è il collegamento JDBC nativo e l'altro è  
// altro collegamento JDBC.  
//  
// questa tecnica è appropriata in quanto consente di utilizzare diverse unità di  
// controllo JDDBS per diverse attività contemporaneamente. Ad esempio, l'unità di  
// controllo Toolbox JDDBS è ideale per collegarsi ai server iSeries remoti e  
// l'unità di controllo JDDBS nativa è più veloce dei collegamenti locali.  
// E' possibile utilizzare le potenzialità di ogni unità di controllo contemporaneamente  
// nella propria applicazione scrivendo un codice simile a questo esempio.
```

```

//
//
// Questo sorgente è un esempio di unità di controllo JDBC IBM Developer per Java.
// L'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 dall'IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. L'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 for 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 iSeries user ID");
            System.out.println(" and <password> is the password for that user ID");
            System.exit(0);
        }

        // Registrare entrambe le unità 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 unità 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 Toolbox 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: proprietà di accesso: ➤ Questo è un esempio di modalità di utilizzo della proprietà Access.

Esempio: proprietà Access

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

// Nota: questo programma presuppone che esista l'indirizzario cujosql.

```

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: ID utente e parola d'ordine non validi: ➤ Questo è un esempio della modalità di utilizzo della proprietà Connection in modalità di denominazione SQL.

Esempio: ID utente e parola d'ordine non validi

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////
//
// Esempio InvalidConnect.
//
// Questo programma utilizza la proprietà Connection nella modalità di denominazione SQL.
//
////////////////////////////////////
//
// Questo sorgente è un esempio di unità di controllo JDBC IBM Developer per Java.
// L'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 dall'IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. L'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 for 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 l'unità 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: Questa tabella contiene le proprietà di collegamento dell'unità di controllo JDBC valide, i relativi valori e le descrizioni:

Proprietà	Valori	Significato
"access"	"all", "read call", "read only"	Si può utilizzare questo valore per limitare il tipo di operazioni che è possibile effettuare con un collegamento specifico. Il valore predefinito è "all" ed essenzialmente indica che il collegamento ha accesso totale all'API JDBC. Il valore "read call" consente al collegamento soltanto di effettuare interrogazioni e di chiamare procedure memorizzate. Viene arrestato un tentativo di aggiornare il database tramite un'istruzione SQL. E' possibile utilizzare il valore "read only" per limitare un collegamento soltanto alle interrogazioni. Le chiamate a procedure memorizzate e le istruzioni di aggiornamento vengono arrestate.
➤ "batch style"	"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. Il valore predefinito è dato dal funzionamento della specifica JDBC 2.0. ◀

Proprietà	Valori	Significato
"block size"	"0", "8", "16", "32", "64", "128", "256", "512"	<p>Questo è il numero di righe selezionate alla volta per la serie dei risultati. Per una tipica elaborazione soltanto di inoltro di una serie dei risultati, si ottiene un blocco di questa dimensione. Quindi non si accede al database in quanto ogni riga viene elaborata dalla propria applicazione. Solo quando viene raggiunta la fine del blocco il database richiede un altro blocco di dati.</p> <p>Questo valore viene utilizzato solo se la proprietà "blocking enabled" è impostata su "true".</p> <p>Impostare la proprietà della dimensione del blocco su "0" equivale a impostare la proprietà "blocking enabled" su "false".</p> <p>Il valore predefinito è di utilizzare il blocco con una dimensione blocco "32". Correntemente, questa è una decisione arbitraria ed è possibile che il valore predefinito cambi in futuro.</p> <p>Correntemente il blocco non viene utilizzato sulle serie di risultati che è possibile scorrere.</p>
"blocking enabled"	"true", "false"	<p>Questo valore viene utilizzato per determinare se sia opportuno o meno che il collegamento utilizzi il blocco sul reperimento della riga della serie dei risultati. E' possibile che il blocco migliori in maniera significativa le prestazioni delle serie dei risultati.</p> <p>L'impostazione predefinita di questa proprietà è "true".</p>

Proprietà	Valori	Significato
» "cursor hold"	"true", "false"	<p>Questo valore specifica se sia necessario o meno che le serie dei risultati rimanga aperta quando viene effettuato 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.</p> <p>L'impostazione predefinita di questa proprietà è "true".</p> <p>Questa proprietà del valore funziona come valore predefinito per tutte le serie dei risultati effettuate per il collegamento. Con il supporto alla capacità di conservabilità del cursore aggiunto in JDBC 3.0, tale valore predefinito viene semplicemente sostituito se un'applicazione specifica in seguito una differente capacità di conservabilità. <<</p>
» "data truncation"	"true", "false"	<p>Questo valore specifica se è opportuno che il troncamento dei dati di caratteri generi avvertenze ed eccezioni (true) o che i dati siano troncati senza che venga emesso alcun messaggio (false). Se il valore predefinito è "true", sarebbe opportuno rispettare il troncamento di dati dei campi di caratteri. <<</p>
» "date format"	"julian", "mdy", "dmy", "ymd", "usa", "iso", "eur", "jis"	<p>Questa proprietà consente di modificare il modo in cui le date vengono formattate. <<</p>
» "date separator"	"/", "-", ".", " ", "b"	<p>Questa proprietà consente di modificare il separatore della data. Essa è valida solo in combinazione con alcuni valori dateFormat (secondo le regole del sistema). <<</p>
» "decimal separator"	".", ",", ""	<p>Questa proprietà consente di modificare il separatore decimale. <<</p>



Proprietà	Valori	Significato
"do escape processing"	"true", "false"	<p>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 piattaforme, quindi il database legge le clausole di uscita e sostituisce la versione specifica del sistema appropriata per l'utente.</p> <p>Ciò è positivo, ma obbliga il sistema ad un lavoro supplementare. Nel caso in cui si ha la certezza di utilizzare solo istruzioni SQL che già contengono una sintassi SQL iSeries valida, si raccomanda di impostare questo valore su "false" per migliorare le prestazioni.</p> <p>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).</p> <p>Tale valore viene aggiunto a causa dell'insufficienza della specifica JDBC. E' 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.</p>
 "errors"	"basic", "full"	<p>Questa proprietà consente di restituire il testo dell'errore del secondo livello di sistema nei messaggi dell'oggetto SQLException. Il valore predefinito è quello di base che restituisce solo il testo del messaggio standard. </p>

Proprietà	Valori	Significato
<p>» "libraries"</p>	<p>Una lista di librerie, separata dagli spazi. (Una lista di librerie può anche essere separata da due punti o da virgole.)</p>	<p>Questa proprietà consente di inserire una lista di librerie nella lista di librerie del lavoro del server o di impostare la libreria predefinita specifica.</p> <p>La proprietà "naming" influenza il funzionamento di questa proprietà. Nel caso dell'impostazione predefinita, dove "naming" è impostato su "sql", JDBC funziona come ODBC. L'elenco librerie non ha alcun effetto 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 nella lista diventa la libreria predefinita. Se una libreria predefinita viene specificata sull'URL di collegamento (come in "jdbc:db2:*local/mylibrary"), essa sostituisce ogni valore in questa proprietà.</p> <p>Nel caso in cui "naming" è impostato su "system", ogni libreria specificata per questa proprietà viene aggiunta alla lista di librerie dell'utente e la lista di librerie viene ricercata per definire dei riferimenti della tabella non qualificati. <<</p>

Proprietà	Valori	Significato
<p>» "lob threshold"</p>	<p>Qualsiasi valore inferiore a 500000</p>	<p>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.</p> <p>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 dei risultati, è ancora facile superare questo limite, che causa l'esito negativo della selezione. E' necessario prestare attenzione alle modalità in cui la proprietà di dimensione blocco e questa proprietà interagiscono con la dimensione di un blocco di dati.</p> <p>Il valore predefinito è 0. I localizzatori si utilizzano sempre per i dati lob. »</p>
<p>"naming"</p>	<p>"sql", "system"</p>	<p>Questa proprietà consente di utilizzare sia la sintassi di denominazione iSeries tradizionale che la sintassi di denominazione sql standard. Una denominazione "system" indica che sarebbe opportuno utilizzare un carattere "/" per separare i valori della tabella e della raccolta, mentre una denominazione "sql" indica che sarebbe opportuno utilizzare un carattere "." per separare i valori.</p> <p>L'impostazione di questo valore ha ramificazioni anche per quanto riguarda la libreria predefinita. Consultare la proprietà libraries per ulteriori informazioni su questo argomento.</p> <p>Il valore predefinito è utilizzare una denominazione "sql".</p>

Proprietà	Valori	Significato
"password"	qualsiasi valore	<p>Questa proprietà consente di specificare una parola d'ordine per il collegamento. Questa proprietà non funziona correttamente senza specificare la proprietà "user". Queste proprietà consentono di effettuare i collegamenti al database come un utente diverso da quello che esegue il lavoro iSeries.</p> <p>Specificare le proprietà "user" e "password" equivale a utilizzare il metodo di collegamento con la firma getConnection(String url, String userId, String password).</p>
» "prefetch"	"true", "false"	<p>Questa proprietà specifica se è opportuno che il programma di controllo selezioni o meno i primi dati per la serie dei risultati subito dopo l'elaborazione o attenda fino alla richiesta dei dati. Se l'impostazione predefinita è "true", sarebbe opportuno sottoporre a prefetch i dati.</p> <p>Per applicazioni che utilizzano l'unità JDBC nativa, ciò costituisce raramente un problema. La proprietà esiste innanzitutto per un utilizzo interno con procedure memorizzate Java e con funzioni definite dall'utente in cui è importante che il motore del database non selezioni alcun dato dalle serie dei risultati per conto dell'utente prima che venga richiesto. <<</p>
» "reuse objects"	"true", "false"	<p>Questa proprietà specifica se è opportuno che l'unità di controllo tenti di riutilizzare alcuni tipi di oggetti dopo che l'utente li ha chiusi. Questo è un aggiornamento delle prestazioni. Il valore predefinito è "true". <<</p>

Proprietà	Valori	Significato
» "server trace"	Una rappresentazione di stringhe di un numero intero	<p>Questa proprietà abilita la traccia del lavoro del server JDBC. Se la traccia del server è abilitata, la traccia inizia quando il client si collega al server e termina alla fine del collegamento.</p> <p>I dati della traccia vengono raccolti nei file di spool sul server. E' possibile attivare più livelli di traccia del server in combinazione aggiungendo le costanti e passando il risultato sul metodo set.</p> <p>Nota: questa proprietà viene utilizzata solitamente dal personale di assistenza e i relativi valori non vengono discussi ulteriormente. <<</p>
» "time format"	"hms", "usa", "iso", "eur", "jis"	Questa proprietà consente di modificare il modo in cui vengono formattati i valori dell'ora. <<
» "time separator"	":", ".", ",", "b"	Questa proprietà consente di modificare il separatore dell'ora. Esso è valido solo in combinazione con altri valori timeFormat (secondo le regole del sistema). <<
"trace"	"true", "false"	<p>Questa proprietà consente di attivare la traccia del collegamento. E' possibile utilizzarla come semplice assistente del debug. Esistono possibilità di aggiornare questa funzione in futuro. Consultare D2. L'unità di controllo JDBC ha emesso un'eccezione. Cosa fare? per ulteriori informazioni sul debug.</p> <p>Il valore predefinito è "false", che non utilizza la traccia.</p>
"transaction isolation"	"none", "read committed", "read uncommitted", "repeatable read", "serializable"	<p>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.</p> <p>Il valore predefinito per questa proprietà è "none", in quanto l'impostazione predefinita di JDBC è la modalità di sincronizzazione automatica.</p>

Proprietà	Valori	Significato
"translate binary"	"true", "false"	<p>E' possibile utilizzare questa proprietà per forzare l'unità di controllo JDBC in modo che tratti i valori di dati "binary" e "varbinary" come se fossero "char" e "varchar".</p> <p>Il valore predefinito per questa proprietà è "false", dove i dati binari non vengono trattati come dati di caratteri.</p>
 "use block insert"	"true", "false"	<p>Questa proprietà consente all'unità di controllo JDBC nativa 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.</p> <p>Sarebbe opportuno che le applicazioni che attivano questa proprietà si collegassero soltanto al sistema locale nel tentativo di eseguire aggiornamenti sottoposti a batch. Esse non dovrebbero utilizzare DRDA per stabilire collegamenti remoti in quanto non è possibile gestire l'inserimento bloccato su DRDA.</p> <p>E' 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 nella lista di valori. Questo è un requisito del motore di inserimento bloccato del sistema.</p> <p>Il valore predefinito è "false". </p>

Proprietà	Valori	Significato
"user"	qualsiasi valore	<p>Questa proprietà consente di specificare un ID utente per il collegamento. Questa proprietà non funziona correttamente se non si specifica anche la proprietà "password". Queste proprietà consentono di effettuare i collegamenti al database come un utente diverso da quello che esegue il lavoro iSeries.</p> <p>Specificare le proprietà "user" e "password" equivale a utilizzare il metodo Connection con la firma getConnection(String url, String userId, String password).</p>

Esempio: creare UDBDataSource e collegarlo con JNDI: ➤ Questo è un esempio di come creare UDBDataSource e collegarlo con JNDI.

Esempio: creare UDBDataSource e collegarlo con JNDI

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Importare i pacchetti richiesti. Al momento dello sviluppo,
// la classe specifica dell'unità 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");

        // Reperire un contesto JNDI. Il contesto serve come
        // root per l'ubicazione a 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: creare UDBDataSourceBind e impostare le proprietà DataSource: ➤ Questo è un esempio di come creare UDBDataSource e impostare l'ID utente e la parola d'ordine come proprietà DataSource.

Esempio: creare UDBDataSourceBind e impostare le proprietà DataSource

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Importare i pacchetti richiesti. Al momento dello sviluppo,
// la classe specifica dell'unità 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");

        // Reperire un contesto JNDI. Il contesto serve come
        // root per l'ubicazione a 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: ottenere 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 sul contesto per restituire un oggetto di tipo DataSource affinché l'applicazione lo utilizzi.

Esempio: ottenere un contesto iniziale prima di collegare UDBDataSource

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Importare i pacchetti richiesti. Non esistono codici
// specifici dell'unità 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
    {
        // Reperire un contesto JNDI. Il contesto serve come
        // root per l'ubicazione a cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Reperire 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 interrogazioni 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: creare UDBDataSource e ottenere 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 durante il tempo di esecuzione.

Esempio: creare UDBDataSource e ottenere un ID utente e una parola d'ordine

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

/// Importare i pacchetti richiesti. Non esiste codice
// specifico dell'unità 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
    {
        // Reperire un contesto JNDI. Il contesto serve come
        // root per l'ubicazione a cui sono collegati gli oggetti
        // o in cui si trovano all'interno di JNDI.
        Context ctx = new InitialContext();

        // Reperire 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 interrogazioni 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();
}
}

```

Utilizzare i DataSource con UDBDataSource: ➤ Le interfacce DataSource sono state progettate per consentire un'ulteriore flessibilità nell'utilizzo delle unità di controllo JDBC (JavaTM Database Connectivity). E' 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.

Un vantaggio dei DataSource è che consentono alle unità di controllo JDBC di funzionare a favore dell'applicazione senza avere un impatto direttamente sul processo di sviluppo dell'applicazione. Per ulteriori informazioni, consultare Creare lotti di collegamenti, Creare lotti di istruzioni e Transazioni distribuite.

UDBDataSourceBind: Il programma UDBDataSourceBind è un esempio di come creare UDBDataSource e collegarlo con 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, reperisce un contesto JNDI e collega l'oggetto ad un nome con il contesto JNDI.

Il codice del tempo di sviluppo è specifico del fornitore. E' necessario che l'applicazione importi l'implementazione DataSource specifica che intende gestire. Nella lista 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 reperimento 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 di applicazione può utilizzare questo DataSource per reperire collegamenti al database e il lavoro correlato a JDBC.

UDBDataSourceUse: Il programma UDBDataSourceUse è un esempio di applicazione JDBC che utilizza l'applicazione disposta in precedenza.

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 nell'interfaccia DataSource, quindi non esiste alcun bisogno che sia consapevole della classe di implementazione. Ciò rende l'applicazione portabile.

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. E' 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 alle unità 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, essa si collega al server iSeries 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. Consultare Esempio: creare UDBDataSourceBind e impostare le proprietà DataSource su come utilizzare questa tecnica.
- utilizzando il metodo getConnection DataSource che acquisisce un ID utente e una parola d'ordine durante il tempo di esecuzione. Consultare Esempio: creare UDBDataSource e ottenere un ID utente e una parola d'ordine su come utilizzare questa tecnica.


Esistono alcune proprietà che è possibile specificare per UDBDataSource così come esistono proprietà che è possibile specificare per i collegamenti creati con DriverManager. Far riferimento a Proprietà DataSource per una lista delle proprietà supportate per l'unità di controllo JDBC nativa.

Anche se queste liste sono simili, non è certo che siano simili nei rilasci futuri. Si consiglia di iniziare la codifica nell'interfaccia DataSource.

Nota: anche l'unità di controllo JDBC nativa dispone di altre due implementazioni DataSource, ma non si consiglia un loro utilizzo diretto.

- DB2DataSource
- DB2StdDataSource



Proprietà DataSource:  Questa tabella contiene le proprietà di sorgenti dati valide, i relativi valori e le descrizioni:

Metodo Set (tipo di dati)	Valori	Descrizione
setAccess (Stringa)	"all", "read call", "read only"	<p>Si può utilizzare questa proprietà per limitare il tipo di operazioni che è possibile effettuare con un collegamento specifico. Il valore predefinito è "all" ed essenzialmente indica che il collegamento ha accesso totale all'API JDBC ovvero Java^(TM) Database Connectivity (JDBC).</p> <p>Il valore "read call" consente al collegamento soltanto di eseguire interrogazioni e di chiamare procedure memorizzate. Ogni tentativo di aggiornare il database tramite un'istruzione SQL provoca un'SQLException.</p> <p>Il valore "read only" limita il collegamento soltanto alle interrogazioni. Il tentativo di elaborare una chiamata ad una procedura memorizzata provoca un'SQLException.</p>
setBatchStyle (Stringa)	"2.0", "2.1"	<p>La specifica 2.1 JDBC definisce un altro metodo per gestire le eccezioni in aggiornamento batch. L'unità può adattarsi a ciascuna di esse. E' necessario che il valore predefinito si applichi come stabilito nella specifica 2.0 JDBC.</p>
setUseBlocking (booleano)	"true", "false"	<p>Questa proprietà viene utilizzata per determinare se sarebbe opportuno o meno che il collegamento utilizzasse il blocco sul reperimento della riga della serie dei risultati. E' possibile che il blocco migliori in maniera significativa le prestazioni delle serie dei risultati.</p> <p>L'impostazione predefinita di questa proprietà è "true".</p>

Metodo Set (tipo di dati)	Valori	Descrizione
setBlockSize (int)	"0", "8", "16", "32", "64", "128", "256", "512"	<p>Questa proprietà indica il numero di righe selezionate alla volta per una serie dei risultati. Per una tipica elaborazione soltanto di inoltro di una serie dei risultati, si ottiene un blocco di questa dimensione se il database dispone di molte righe che soddisfano l'interrogazione. Solo quando viene raggiunta la fine del blocco nella memoria interna dell'unità di controllo JDBC, viene inviata un'altra richiesta per un blocco di dati al database.</p> <p>Questo valore viene utilizzato solo se la proprietà useBlocking è impostata su "true". Fare riferimento a setUseBlocking per ulteriori informazioni.</p> <p>Impostare la proprietà della dimensione del blocco su "0" equivale a chiamare UseBlocking(false).</p> <p>Il valore predefinito è di utilizzare il blocco con una dimensione blocco "32". Questa è una decisione arbitraria ed è possibile che il valore predefinito cambi in rilasci futuri.</p> <p>Correntemente il blocco non viene utilizzato sulle serie di risultati che è possibile scorrere.</p> <p>L'utilizzo di blocchi influenza il grado di sensibilità del cursore che ha l'applicazione utente. Un cursore sensibile vede le modifiche effettuate da altre istruzioni SQL. Tuttavia, a causa della memorizzazione di dati nella memoria cache, le modifiche vengono individuate quando è necessario selezionare i dati dal database.</p>

Metodo Set (tipo di dati)	Valori	Descrizione
setCursorHold (booleano)	"true", "false"	<p>Questa proprietà specifica se è necessario o meno che le serie dei risultati rimanga aperta quando viene effettuato il commit di una transazione. Un valore di "true" indica che un'applicazione può accedere alle relative serie dei risultati aperte una volta chiamato il commit. Un valore di "false" indica che il commit chiude ogni cursore aperto durante il collegamento.</p> <p>L'impostazione predefinita di questa proprietà è "true".</p> <p>Questa proprietà funziona come valore predefinito per tutte le serie dei risultati effettuate per il collegamento. Con il supporto del cursore aggiunto in JDBC 3.0 (consultare la sezione Caratteristiche di ResultSet per dettagli), questo valore predefinito viene semplicemente sostituito se un'applicazione specifica in seguito un differente supporto del cursore.</p>
setDataTruncation (booleano)	"true", "false"	<p>Questa proprietà specifica quanto segue:</p> <ul style="list-style-type: none"> • Se è opportuno che il troncamento dei dati di caratteri generi avvertenze ed eccezioni (true) • Se è opportuno che i dati siano troncati senza che venga emesso alcun messaggio (false). <p>Consultare DataTruncation per ulteriori dettagli.</p>
setDatabaseName (Stringa)	Qualsiasi nome	<p>Questa proprietà specifica il database a cui il DataSource tenta di collegarsi. Il valore predefinito è *LOCAL. E' 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.</p>
setDataSourceName (Stringa)	Qualsiasi nome	<p>Questa proprietà consente di passare un nome JNDI (Java Naming and Directory Interface) ConnectionPoolDataSource per supportare il lotto di collegamenti.</p>
setDateFormat (Stringa)	"julian", "mdy", "dmy", "ymd", "usa", "iso", "eur", "jis"	<p>Questa proprietà consente di modificare il modo in cui le date vengono formattate.</p>

Metodo Set (tipo di dati)	Valori	Descrizione
setDateSeparator (Stringa)	"/", "-", ".", ":", ":", "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).
setDecimalSeparator (Stringa)	".", ",", "	Questa proprietà consente di modificare il separatore decimale.
setDescription (Stringa)	Qualsiasi nome	Questa proprietà consente l'impostazione di questa descrizione di testo dell'oggetto DataSource.
setDoEscapeProcessing (booleano)	"true", "false"	Questa proprietà specifica se è stata effettuata l'elaborazione di uscita sulle istruzioni SQL. Il valore predefinito per questa proprietà è "true".
setFullErrors (booleano)	"true", "false"	Questa proprietà consente di restituire il testo dell'errore del secondo livello dell'intero sistema nei messaggi dell'oggetto SQLException. Il valore predefinito è "false".
setLibraries (Stringa)	Una lista di librerie, separata dagli spazi	Questa proprietà consente di inserire una lista di librerie nella libreria del lavoro del server. Questa proprietà viene utilizzata soltanto quando si utilizza setSystemNaming(true).
setLobThreshold (int)	Qualsiasi valore inferiore a 500000	Questa proprietà indica al programma di controllo di inserire i valori effettivi invece dei localizzatori LOB (Locator Object) se la colonna LOB è più piccola rispetto alla dimensione della soglia.
setLoginTimeout (int)	Qualsiasi valore	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setNetworkProtocol (int)	Qualsiasi valore	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setPassword (Stringa)	Qualsiasi stringa	Questa proprietà consente di specificare una parola d'ordine per il collegamento. Tale proprietà viene ignorata se non si imposta un ID utente.
setPortNumber (int)	Qualsiasi valore	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setPrefetch (booleano)	"true", "false"	Questa proprietà specifica se è opportuno che il programma di controllo selezioni i primi dati per la serie dei risultati subito dopo l'elaborazione o attenda fino alla richiesta dei dati. Il valore predefinito è "true".

Metodo Set (tipo di dati)	Valori	Descrizione
setReuseObjects (booleano)	"true", "false"	Questa proprietà specifica se è opportuno che l'unità di controllo tenti di riutilizzare alcuni tipi di oggetti dopo che l'utente li ha chiusi. Questo è un aggiornamento delle prestazioni. Il valore predefinito è "true".
setServerName (Stringa)	Qualsiasi nome	Questa proprietà viene correntemente ignorata e pianificata per un utilizzo futuro.
setServerTraceCategories (int)	Una rappresentazione di stringhe di un numero intero	Questa proprietà abilita la traccia del lavoro del server JDBC. Se la traccia del server è abilitata, la traccia inizia quando il client si collega al server e termina alla fine del collegamento. I dati della traccia vengono raccolti nei file di spool sul server. E' possibile attivare più livelli di traccia del server in combinazione aggiungendo le costanti e passando il risultato sul metodo set. Nota: questa proprietà viene utilizzata dal personale di assistenza ed i relativi valori non vengono discussi.
setSystemNaming (booleano)	"true", "false"	Questa proprietà consente di specificare se è opportuno separare le raccolte e le tabelle con un punto (denominazione SQL) o una barra (denominazione del sistema). Questa proprietà determina inoltre se viene utilizzata una libreria predefinita (denominazione SQL) o una lista di librerie (denominazione del sistema). Il valore predefinito è una denominazione SQL.
setTimeFormat (Stringa)	"hms", "usa", "iso", "eur", "jis"	Questa proprietà consente di modificare il modo in cui i valori dell'ora vengono formattati.
setTimeSeparator (Stringa)	".", ".", ",", "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).
setTrace (booleano)	"true", "false"	Questa proprietà può abilitare una traccia semplice. Il valore predefinito è "false".
setTransactionIsolationLevel (Stringa)	"none", "read committed", "read uncommitted", "repeatable read", "serializable"	Questa proprietà consente la specifica del livello di isolamento della transazione. Il valore predefinito per questa proprietà è "none", in quanto l'impostazione predefinita di JDBC è la modalità di sincronizzazione automatica.

Metodo Set (tipo di dati)	Valori	Descrizione
setTranslateBinary (Booleano)	"true", "false"	<p>E' possibile utilizzare questa proprietà per forzare l'unità di controllo JDBC in modo che tratti i valori di dati "binary" e "varbinary" come se fossero "char" e "varchar".</p> <p>Il valore predefinito per questa proprietà è "false".</p>
setUseBlockInsert (booleano)	"true", "false"	<p>Questa proprietà consente all'unità di controllo JDBC nativa 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.</p> <p>Sarebbe opportuno che le applicazioni che attivano questa proprietà si collegassero soltanto al sistema locale nel tentativo di eseguire aggiornamenti sottoposti a batch. Esse non dovrebbero utilizzare DRDA per stabilire collegamenti remoti in quanto non è possibile gestire l'inserimento bloccato su DRDA.</p> <p>E' 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 nella lista di valori. Questo è un requisito del motore di inserimento bloccato del sistema.</p> <p>Il valore predefinito è "false".</p>
setUser (Stringa)	qualsiasi valore	<p>Questa proprietà consente l'impostazione di un ID utente per ottenere i collegamenti. Questa proprietà richiede che l'utente imposti anche la proprietà della parola d'ordine.</p>



Altre implementazioni DataSource: ➤ Esistono due implementazioni dell'interfaccia DataSource che sono incluse con l'unità di controllo JDBC nativa. Queste implementazioni DataSource devono essere considerate obsolete. Dal momento che è ancora possibile utilizzarle, non sono consentiti miglioramenti

futuri; ad esempio, non vengono aggiunti lotti di istruzioni e collegamenti considerevoli a queste implementazioni. Queste implementazioni esistono fin tanto che non viene adottata l'interfaccia UDBDataSource e le relative funzioni correlate.

DB2DataSource: DB2DataSource era un'implementazione precedente dell'interfaccia DataSource e non è compatibile con la specifica completa (cioè questa precede nel tempo la specifica). DB2DataSource esiste oggi solo per consentire agli utenti WebSphere^(R) di migrare a rilasci correnti e non devono essere utilizzati altrimenti.

DB2StdDataSource: DB2StdDataSource è la versione corretta dell'implementazione DB2DataSource diventata compatibile con la specifica una volta che la specifica del pacchetto facoltativo JDBC è diventata quella finale. La nuova versione è stata fornita per non violare il codice già scritto sulla versione DB2DataSource.

Se esistono applicazioni scritte che fanno uso di queste implementazioni DataSource, la migrazione su UDBDataSource è un'attività di minima entità se tutte le vecchie proprietà sono supportate. E' consigliata la migrazione a UDBDataSource per ottenere la funzionalità delle nuove classi UDBDataSource. <<

Interfaccia DatabaseMetaData di IBM Developer Kit per Java

L'interfaccia DatabaseMetaData viene implementata dall'unità di controllo JDBC ovvero IBM Developer Kit per Java^(TM) per fornire informazioni sulle relative sorgenti dati sottostanti. Essa viene utilizzata principalmente dagli strumenti e server dell'applicazione per stabilire come interagire con una sorgente dati forniti. Le applicazioni possono inoltre utilizzare i metodi DatabaseMetaData per ottenere informazioni su una sorgente dati, ma ciò avviene meno frequentemente.

L'interfaccia DatabaseMetaData include oltre 150 metodi che è possibile suddividere in categorie in base ai seguenti tipi di informazioni che forniscono:

- Informazioni generali sul sorgente dati
- Supporto del sorgente dati per una funzione data
- Limiti del sorgente dati
- Oggetti SQL e relativi attributi
- Supporto di transazione fornito dal sorgente dati

L'interfaccia DatabaseMetaData contiene inoltre oltre 40 campi che sono costanti utilizzate come valori di ritorno per vari metodi DatabaseMetaData.

Consultare Modifiche in JDBC 3.0 per informazioni sulle modifiche effettuate sui metodi nell'interfaccia DatabaseMetaData.

Creare 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 sul sorgente dati sottostanti. Il seguente esempio crea un oggetto DatabaseMetaData e lo utilizza per determinare il numero massimo di caratteri consentiti per un nome di tabella:

Esempio: creare un oggetto DatabaseMetaData

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

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

Reperire informazioni generali: Alcuni metodi DatabaseMetaData vengono utilizzati sia per reperire dinamicamente informazioni generali su una sorgente 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

Determinare il supporto della funzione: E' possibile utilizzare un ampio gruppo di metodi DatabaseMetaData per stabilire se una funzione o un gruppo di funzioni date sono supportate dall'unità o dal sorgente dati sottostanti. 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 del sorgente dati: Un altro gruppo di metodi fornisce i limiti imposti da una sorgente 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 sorgente 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 nel sorgente dati. Esempi di questa categoria includono quanto segue:

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

Supporto transazione: Un esiguo gruppo di metodi fornisce informazioni sulla semantica della transazione supportata dal sorgente dati. Esempi di questa categoria includono quanto segue:

- supportsMultipleTransactions
- getDefaultTransactionIsolation

Consultare Esempio: interfaccia DatabaseMetaData per IBM Developer Kit per Java 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
- getUDTs
- 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 rilasci JDK, essa riceve SQLException quando tenta di accedere a questi campi che non esistono in rilasci JDK precedenti. SafeGetUDTs è un esempio di come è possibile scrivere un'applicazione per gestire JDK 1.4, JDK 1.3 e i precedenti rilasci JDK.



Esempio: interfaccia DatabaseMetaData di IBM Developer Kit per Java: Questo esempio mostra come restituire una lista di tabelle.

Esempio 1: restituire una lista di tabelle

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

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

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

// Richiamare una lista 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();
```

Per ulteriori informazioni, consultare l'interfaccia DatabaseMetaData di IBM Developer Kit per Java^(TM).

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

Esempio: utilizzare i ResultSet di metadati che possiedono più di una colonna

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
//
// Esempio SafeGetUDTs. Questo programma mostra un modo per gestire i
// ResultSet metadati che hanno più colonne in JDK 1.4 rispetto
// rispetto ai rilasci precedenti.
//
// Sintassi del comando:
//   java SafeGetUDTs
//
////////////////////////////////////
//
// Questo sorgente è un esempio di unità di controllo JDBC IBM Developer per Java.
// L'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 dall'IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. L'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 for 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()) {

            // Selezionare tutte le colonne disponibili dal rilascio
            // 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

» Il linguaggio Java[™] utilizza delle eccezioni per fornire funzioni di gestione errori 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". E' possibile utilizzare lo stesso meccanismo per emettere eccezioni nei propri programmi Java.

Per maggiori informazioni sulle eccezioni, esaminare le seguenti sezioni:

SQLException

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

SQLWarning

I metodi generano un oggetto SQLWarning se essi causano un'avvertenza di accesso del database. I metodi nelle seguenti interfacce possono generare SQLWarning:

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

DataTruncation

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.

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.



SQLException: ➤ La classe SQLException e i relativi sottotipi forniscono informazioni sugli errori e le avvertenze che si verificano durante l'accesso ad una sorgente 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 all'unità di controllo o al database di notificare più errori in una singola richiesta. Non esistono attualmente istanze in cui l'unità di controllo JDBC nativa può eseguire questa operazione. Queste informazioni vengono fornite soltanto come riferimento e non come chiara indicazione che l'unità 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, l'unità di controllo JDBC nativa emette raramente delle SQLException effettive. Essa 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 l'unità 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 l'unità 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 reperito e restituito all'utente. La classe DatabaseException provoca un errore che il database riconosce e notifica all'unità di controllo JDBC per cui creare l'oggetto dell'eccezione.

DB2JDBCException.java: DB2JDBCException vengono generate per condizioni di errore che provengono dall'unità di controllo JDBC stessa. La funzionalità di questa classe di eccezioni è fondamentalmente differente; l'unità 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, l'unità di controllo JDBC aderisce agli SQLState del database. Il codice fornitore per eccezioni emesse dall'unità 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 l'unità di controllo JDBC riconosce e per cui genera l'eccezione. Quando è stata eseguita durante lo sviluppo del rilascio, è stata creata la seguente emissione. Si noti che il vertice dello stack contiene DB2JDBCException. Questa è un'indicazione che l'errore è stato riportato dall'unità di controllo JDBC prima di effettuare la richiesta al database. <<

Esempio: SQLException: >> Questo è un esempio di come rilevare un'SQLException ed eseguire il dump di tutte le informazioni che esso fornisce.

Esempio: SQLException

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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: ➤ I metodi nelle interfacce seguenti creano un oggetto SQLWarning se determinano un'avvertenza per l'accesso al database:

- 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. E' necessario che il metodo getWarnings sia chiamato sull'oggetto appropriato per richiamare l'oggetto SQLWarning. E' possibile emettere la sottoclasse DataTruncation di SQLWarning, tuttavia, in alcune circostanze. E' rilevante il fatto che l'unità di controllo JDBC nativa 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.

E' 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. E' necessario che l'applicazione richieda attivamente le informazioni sull'avvertenza.

Come le SQLException, è possibile che le SQLWarning siano collegate l'una all'altra. E' 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.

L'unità 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: >> 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. 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 che sarebbe stato opportuno trasferire.
- 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 l'unità di controllo fornisca le informazioni su di esso.

Notificare tramite i metodi set() e update(): Esiste una sottile differenza tra le unità di controllo JDBC. Alcune unità di controllo come quelle native e JDBC di Toolbox acquisiscono e notificano questioni sul troncamento di dati al momento dell'impostazione del parametro. Queste operazioni vengono effettuate sul metodo set PreparedStatement o sul metodo update ResultSet. Altre unità di controllo riportano il problema al momento dell'elaborazione dell'istruzione tramite i metodi execute, executeQuery o updateRow.

L'impossibilità di riportare 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 due vantaggi:

- E' 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, l'unità 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.

I metodi ResultSet.update() emettono eccezioni DataTruncation: In alcuni rilasci 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 sarebbe opportuno che le unità di controllo JDBC emettessero eccezioni in questi casi. Come risultato, l'unità di controllo JDBC funziona in questo modo.

Non dovrebbe esserci 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 Prepared statement 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 su un server iSeries.

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 sarebbe opportuno che il troncamento dati su un valore da scrivere nel database emettesse un'eccezione. Se il troncamento dati non viene eseguito sul valore scritto nel database, si dovrebbe creare 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 dell'interrogazione non danneggiano mai il contenuto del database. Quindi, le situazioni di troncamento dati vengono sempre gestite inviando delle avvertenze.
- In 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, l'unità 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: E' stata disponibile una proprietà del troncamento dati per molti rilasci. 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 adatta 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 rilasci 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 dell'unità 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 dell'unità 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.
```

L'unità di controllo JDBC nativa emette eccezioni per un troncamento dati non significativo: L'unità di controllo JDBC nativa 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 l'unità 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, come la proprietà del troncamento dati menzionato in precedenza, interessa solo tipi a base di caratteri come CHAR, VARCHAR, CHAR FOR BIT DATA e VARCHAR FOR BIT DATA.



Transazioni

» Una **transazione** è un'unità logica di lavoro. Per completare un'unità logica di lavoro, è possibile intraprendere numerose azioni rispetto a 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 vengono utilizzate per fornire l'integrità dei dati, per correggere la semantica dell'applicazione e procurare una vista di dati coerente durante l'accesso simultaneo. Tutte le unità di controllo compatibili con JDBC (JavaTM Database Connectivity) devono supportare le transazioni.

Nota: questa sezione riguarda solo le transazioni locali e il concetto JDBC standard delle transazioni. Java e l'unità di controllo JDBC nativa supportano JTA (Java Transaction API), le transazioni distribuite e 2PC (two-phase commit protocol-protocollo di sincronizzazione 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.

Per utilizzare le transazioni nell'applicazione, consultare quanto segue:

Modalità di sincronizzazione automatica

JDBC utilizza la modalità di sincronizzazione automatica laddove ogni aggiornamento al database è immediatamente reso permanente.

Livelli di isolamento della transazione

I livelli di isolamento della transazione specificano quali dati sono visibili alle istruzioni nell'ambito di una transazione e influiscono direttamente sul livello di accesso simultaneo.

Punti di salvataggio

I punti di salvataggio sono dei punti di controllo sui quali l'applicazione può effettuare il ripristino allo stato precedente senza eliminare l'intera transazione. Reperire le seguenti informazioni sui punti di salvataggio:

- Impostare e ripristinare allo stato precedente i punti di salvataggio
- Rilasciare un punto di salvataggio



Modalità di sincronizzazione automatica: ➤ Per impostazione predefinita, JDBC utilizza una modalità di operazione denominata sincronizzazione automatica. 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 sincronizzazione automatica. 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 sincronizzazione automatica è in esecuzione.

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

E' possibile abilitare e disabilitare dinamicamente la modalità di sincronizzazione automatica durante un collegamento. La sincronizzazione automatica viene abilitata nel modo seguente, presupponendo che il sorgente dati esista già:

```
Connection connection = dataSource.getConnection();  
  
Connection.setAutoCommit(false); // Disabilita la sincronizzazione automatica.
```

Se l'impostazione di sincronizzazione automatica viene modificata a metà di una transazione, qualsiasi lavoro in sospeso viene automaticamente sottoposto a commit. Si genera una SQLException se l'autocommit 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 sorgente 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. La Transazione A effettua un ripristino allo stato precedente.

E' 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 un'interrogazione 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: DB2/400 non espone l'applicazione alle anomalie del database consentite ai livelli prescritti a causa delle 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 l'unità 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 sincronizzata. 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.

E' 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 sincronizzazione. La specifica JDBC non definisce `TRANSACTION_NONE` nello stesso modo. `TRANSACTION_NONE` viene definito nella specifica JDBC come un livello nel quale l'unità di controllo non supporta le transazioni e non è un'unità 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'unità di controllo JDBC viene stabilito dall'implementazione. Il livello predefinito di isolamento della transazione per il livello di isolamento della transazione predefinito dell'unità di controllo JDBC nativa è

NONE. Ciò consente all'unità 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.

L'unità di controllo JDBC nativa consente di passare 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". E' responsabilità dell'applicazione tenere traccia del livello in cui l'esecuzione si verifica se viene indicato come requisito nell'applicazione.

Nei rilasci precedenti, l'unità di controllo JDBC gestiva la selezione di "true" da parte dell'utente per la sincronizzazione automatica modificando il livello di isolamento della transazione su none poiché il sistema non possedeva alcun concetto relativo a una modalità di sincronizzazione automatica impostata 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 sincronizzazione automatica dal concetto di un livello di isolamento della transazione. Perciò è assolutamente giustificata l'esecuzione al livello JDBC_TRANSACTION_SERIALIZABLE con la sincronizzazione automatica abilitata. L'unico scenario non valido riguarda l'esecuzione al livello JDBC_TRANSACTION_NONE senza la modalità di sincronizzazione automatica. Non è possibile che l'applicazione ottenga il controllo sui limiti di sincronizzazione 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 iSeries: La piattaforma iSeries ha nomi comuni per i livelli di isolamento della transazione che non corrispondono ai nomi forniti dalla specifica JDBC. La tabella seguente mette in corrispondenza i nomi utilizzati dalla piattaforma iSeries, ma questi non sono equivalenti a quelli utilizzati dalla specifica JDBC:

Livello JDBC*	Livello iSeries
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 iSeries *NONE e *NC per chiarezza. Non si tratta di una corrispondenza diretta di livello di specifica per iSeries. <<

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 sui quali l'applicazione può effettuare il ripristino allo stato precedente senza eliminare l'intera transazione. I punti di salvataggio sono nuovi in JDBC 3.0, nel senso che l'applicazione deve essere in esecuzione su JDK (JavaTM) Development Kit) 1.4 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 non è utilizzato con rilasci 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 l'unità 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).

Impostare e ripristinare allo stato precedente i punti di salvataggio: E' possibile impostare i punti di salvataggio lungo l'intera transazione. E' possibile che l'applicazione ripristini allo stato precedente 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 ripristino allo stato precedente per il 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 sincronizzata. La tabella di database contiene i valori FIRST e THIRD.

Esempio: impostare e ripristinare allo stato precedente i punti di salvataggio

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 sincronizzazione automatica possa causare problemi, questi non possono essere ripristinati allo stato precedente in quanto il loro periodo di attività termina con la fine di una transazione.

Rilasciare un punto di salvataggio: E' 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 effettuare il ripristino allo stato precedente su di esso. Quando si effettua il commit o il rollback di una transazione, tutti i punti di salvataggio vengono automaticamente rilasciati. Quando viene effettuato il ripristino allo stato precedente su un punto di salvataggio, tutti i successivi punti di salvataggio vengono rilasciati.



Transazioni distribuite

» In genere, le transazioni nella JDBC ovvero Java^(TM) 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 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)

1.0.1  .

La 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 ha intenzione di gestire JTA, far riferimento a 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). 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

XADDataSource. Tale classe contiene il supporto per impostare un lotto di collegamenti esattamente nello stesso modo della classe principale ConnectionPoolDataSource.

Con un'istanza XADDataSource, è possibile reperire 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.

Esempio: la sezione denominata utilizzare JTA per gestire una transazione mostra come utilizzare JTA per gestire una transazione in un'applicazione.

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

Proprietà XADDataSource: In aggiunta alle proprietà fornite da ConnectionPoolDataSource, l'interfaccia XADDataSource 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.

Metodo Set (tipo di dati)	Valori	Descrizione
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.

Esaminare Esempio: terminare una transazione che mostra questa funzionalità.

Sospendere e ripristinare: 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.

Esaminare Esempio: sospendere e ripristinare una transazione che mostra questa funzionalità.

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.

Esaminare Esempio: ResultSet sospesi che mostra questa funzionalità.

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. Correntemente, 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.

Questo esempio mostra più collegamenti che lavorano su una singola transazione.

Questo esempio mostra un singolo collegamento con più transazioni che si verificano contemporaneamente.

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.

Sincronizzazione a due fasi e registrazione della transazione: Le API JTA rendono pienamente esplicite le responsabilità del protocollo di sincronizzazione 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 sincronizzazione a due fasi e ogni registrazione associata richiesta per l'atomicità della transazione rispetto a quei database. Normalmente la sincronizzazione 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 della sincronizzazione 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.



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

Esempio: utilizzare JTA per gestire una transazione

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

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

```
public class JTACCommit {
```

```

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

    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))");
    } finally {
        s.close();
        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 il sorgente dati venga riportato da un UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource) 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. Consultare 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
// altro 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 sincronizzata.
int rc = xaRes.prepare(xid);

// La transazione viene sincronizzata tramite XAResource.
// L'oggetto JDBCConnection non viene usato per sincronizzare
// la 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.

Esempio: più collegamenti che operano su una transazione

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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 UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource)
            ctx.lookup("XADDataSource");
        // 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 lasciata di nuovo al
        // programma applicativo.
        Xid xid = JDXTA.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, sincronizzare la transazione come una singola unità
// E' 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 {
        try {
            if (c1 != null) {
                c1.close();
            }
        }
        catch (SQLException e) {
            System.out.println("Note: Cleanup exception " +
                e.getMessage());
        }
        try {
            if (c2 != null) {
                c2.close();
            }
        }
        catch (SQLException e) {
            System.out.println("Note: Cleanup exception " +
                e.getMessage());
        }
        try {
            if (c3 != null) {
                c3.close();
            }
        }
        catch (SQLException e) {
            System.out.println("Note: Cleanup exception " +
                e.getMessage());
        }
    }
}
}
}
}

```



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

Esempio: utilizzare un collegamento con più transazioni

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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))");

        } 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 il sorgente dati venga riportato da un UDBXADDataSource.
            UDBXADDataSource ds = (UDBXADDataSource) 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 lasciata 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 sincronizzato
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: Cleanup 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.

Esempio: ResultSet sospesi

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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.')");

        } 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 il sorgente dati venga riportato da un UDBXADDataSource.
            UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADDataSource");

            // 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
// altro 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 ResultSer 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 sincronizzazione 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: Cleanup exception.");
                e.printStackTrace();
            }
        }
    }
}

```



Esempio: terminare una transazione: ➤ Questo è un esempio di chiusura di una transazione nell'applicazione.

Esempio: terminare una transazione

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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 il sorgente dati venga riportato da un UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADDataSource");

        // 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
        // altro 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.");
        }

        // Sincronizzare la 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: Cleanup exception.");
                e.printStackTrace();
            }
        }
    }
}

```



Esempio: sospendere e ripristinare una transazione: ➤ Questo è un esempio di transazione sospesa e successivamente ripristinata.

Esempio: sospendere e ripristinare una transazione

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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 il sorgente dati venga riportato da un UDBXDataSource.
            UDBXDataSource ds = (UDBXDataSource) ctx.lookup("XADDataSource");

            // 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
            // altro 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 un'interrogazione. Questa
// elaborazione e altre elaborazioni di transazioni eseguibili dalla
// transazione sono separate da quella effettuata precedentemente con XID.
Statement nonXASstmt = conn.createStatement();
ResultSet nonXARS = nonXASstmt.executeQuery("SELECT *FROM CUJOSQL.JTATABLE");
while (nonXARS.next()) {
    // Elaborazione...
}
nonXARS.close();
nonXASstmt.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 sincronizzazione 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: Cleanup 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) rispetto al 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 rilasci precedenti, era possibile fornire il numero esatto di oggetti `Statement` che potevano essere creati. E' impossibile che si verifichi ciò in questo rilascio 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.

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

Per utilizzare le istruzioni e le relative sottoclassi, consultare quanto segue:

Statement

Si utilizza un oggetto `Statement` per l'elaborazione di un'istruzione SQL statica e l'ottenimento dei risultati prodotti da questa.

PreparedStatement

`PreparedStatement` è una sottoclasse dell'interfaccia `Statement` e fornisce supporto per l'aggiunta di parametri alle istruzioni SQL.

CallableStatement

Le `CallableStatement` estendono l'interfaccia `PreparedStatement` e forniscono un supporto per i parametri di emissione e di immissione/emissione in aggiunta al parametro di immissione fornito da `PreparedStatement`.

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.



Statement: ➤ Un oggetto Statement viene utilizzato per l'elaborazione di un'istruzione SQL statica e per l'ottenimento di risultati prodotti da questa. E' 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.

Creare 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();
```

Specificare le 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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// 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.

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

Restituire risultati dalle interrogazioni SQL: Se deve essere elaborata un'istruzione dell'interrogazione SQL che restituisce un oggetto ResultSet, si dovrebbe utilizzare il metodo executeQuery(). E' 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.

Restituire i 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.

Elaborare la 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, l'unità 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 passare 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.

Il metodo cancel: I metodi dell'unità di controllo JDBC nativa 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 l'unità 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. E' possibile utilizzare il metodo `cancel` per arrestare le interrogazioni SQL in esecuzione sul sistema. <<

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

Esempio: utilizzare il metodo `executeUpdate` dell'oggetto `Statement`

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 l'unità di controllo JDBC nativa. Se non è possibile registrare
        // l'unità di controllo, 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 iSeries 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 un'interrogazione 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 interrogazione 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 supportare il lotto di istruzioni dell'unità di controllo JDBC.

Creare PreparedStatements: 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 = ?");
```

Specificare le caratteristiche ResultSet ed il 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);
```

Gestire i 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 dati Java. 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 Javadoc per il 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.

Consultare Esempio: `ParameterMetaData` per le modalità di utilizzo di questo nuovo supporto in un programma dell'applicazione. <<

Elaborare 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 in 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.

Restituire risultati dalle interrogazioni SQL: Se un'istruzione di interrogazione 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`.

Restituire i 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`.

Elaborare la 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,

l'unità 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: utilizzare `PreparedStatement` per ottenere un `ResultSet`: >> Questo rappresenta un esempio dell'utilizzo del metodo `executeQuery` dell'oggetto `PreparedStatement` per ottenere un `ResultSet`.

Esempio: utilizzare `PreparedStatement` per ottenere un `ResultSet`

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 l'unità di controllo JDBC nativa. Se non è possibile
        // registrare l'unità di controllo, 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 iSeries 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 un'interrogazione 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: ParameterMetaData: ➤ Questo è un esempio della modalità di utilizzo dell'interfaccia ParameterMetaData per richiamare le informazioni sui parametri.

Esempio: ParameterMetaData

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
//
// 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 unità di controllo JDBC IBM Developer per Java.
// L'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 dall'IBM con la sola funzione illustrativa.
// Questi esempi non sono stati interamente testati in tutte le
// condizioni. L'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 for 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));
        }
    }
}

```



CallableStatement: ➤ L'interfaccia CallableStatement estende PreparedStatement e fornisce il supporto per i parametri di immissione/emissione ed 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, hanno il potenziale per un nuovo utilizzo ottimizzato (consultare PreparedStatement per 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.

Creare CallableStatements: 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. E' 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 interrogazioni internamente e abbia lasciato i ResultSet derivati da quelle interrogazioni, aperti e disponibili per il programma chiamante. E' 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.

Specificare le caratteristiche ResultSet ed il 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);
```

Gestire i 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 all'unità 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.

Fare riferimento all'Esempio: creare una procedura con parametri di immissione ed emissione per ulteriori informazioni.

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 deve essere 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");    //Contrassegno parametro 1, parm 1 procedura memorizzata  
  
cs.setString(2, "Third");    //Contrassegno parametro 2, parm 3 procedura memorizzata
```

Accedere per nome ai parametri di procedura memorizzata: I parametri nelle procedure memorizzate hanno nomi associati ad essi come mostra la seguente dichiarazione della procedura memorizzata:

Esempio: parametri della procedura memorizzata

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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.
```

Per ulteriori informazioni, consultare Elaborare le CallableStatement.



Elaborare CallableStatement: ➤ L'elaborazione di chiamate ad una procedura memorizzata SQL con un oggetto CallableStatement viene compiuta con gli stessi metodi utilizzati con un oggetto PreparedStatement.

Restituire i risultati per le procedure memorizzate: Se un'istruzione dell'interrogazione SQL viene elaborata all'interno di una procedura memorizzata, è possibile rendere disponibili i risultati dell'interrogazione per il programma che chiama la procedura memorizzata. E' inoltre possibile chiamare più interrogazioni nell'ambito di una procedura memorizzata e che il programma chiamante elabori tutti i ResultSet disponibili.

Consultare Esempio: creare una procedura con più ResultSet per ulteriori informazioni.

Nota: se una procedura memorizzata viene elaborata con `executeQuery` e non restituisce `ResultSet`, viene emessa una `SQLException`.

Accedere ai ResultSet simultaneamente: Restituire i risultati per le procedure memorizzate tratta i `ResultSet` e le procedure memorizzate e fornisce un esempio che si applica a tutti i rilasci del JDK ovvero Java^(TM) 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, 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 su un sistema precedente la V5R2 non dispone di questo supporto.

Restituire i conteggi di aggiornamento per le procedure memorizzate: Restituire i conteggi di aggiornamento per le procedure memorizzate è una funzione discussa nella specifica JDBC, ma non è attualmente supportata sulla piattaforma iSeries. 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 Elaborare le procedure memorizzate che dispongono di un valore di ritorno per ulteriori informazioni.

Elaborare le 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, l'unità 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.

Elaborare procedure memorizzate che dispongono di un valore di ritorno: La piattaforma iSeries 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.

Consultare Esempio: creare una procedura con valori di ritorno per ulteriori informazioni.



Esempio: interfaccia CallableStatement per IBM Developer Kit per Java: Questo è un esempio della modalità di utilizzo dell'interfaccia `CallableStatement`.

Esempio: interfaccia `CallableStatement`

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

// Collegarsi al server iSeries.
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 reperiti 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();

```

Per ulteriori informazioni, consultare CallableStatement.

Esempio: creare una procedura con più ResultSet: ➤ **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 l'unità di controllo JDBC nativa. Se non è possibile
        // registrare l'unità 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 iSeries 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(); // 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: creare una procedura con i parametri di immissione ed emissione: Nota: consultare 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 l'unità di controllo JDBC nativa. Se non è possibile
        // registrare l'unità 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 iSeries 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: creare una procedura con i valori di ritorno: Nota: consultare 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 l'unità di controllo JDBC nativa. Se non è possibile
        // registrare l'unità 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 iSeries locale
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Creare una procedura con un valore di ritorno. Notare che il supporto del
            // valore di ritorno è nuovo nella V4R5.
            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");

            // E' 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 un accesso ai risultati generati dalle interrogazioni in esecuzione. 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 ripristinate in sequenza. All'interno di una riga, è possibile accedere ai valori della colonna in qualsiasi ordine.

Per utilizzare l'oggetto del `ResultSet`, consultare quanto segue:

Caratteristiche del ResultSet

Questa sezione individua le caratteristiche del `ResultSet` come quelle riportate di seguito:

- Tipi di `ResultSet`
- Simultaneità
- Capacità di chiudere il `ResultSet` sincronizzando l'oggetto `Connection`.
- Specifica delle caratteristiche del `ResultSet`

Movimenti del cursore

Le unità di controllo JDBC (JavaTM Database Connectivity) iSeries supportano `ResultSet` che è possibile scorrere. 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.

Richiamare i dati del ResultSet

Individua il modo in cui un oggetto del `ResultSet` fornisce i metodi per ottenere i dati di colonna relativi a una riga.

Modificare ResultSet

Con le unità di controllo JDBC iSeries, è possibile modificare i `ResultSet` eseguendo queste attività:

- Aggiornare le righe
- Cancellare le righe
- Inserire le righe
- Modificare gli aggiornamenti della posizione

Creare ResultSet

E' possibile creare un oggetto del `ResultSet` utilizzando i metodi `executeQuery` forniti dalle interfacce `Statement`, `PreparedStatement` o `CallableStatement`. Questa sezione concerne inoltre la chiusura degli oggetti del `ResultSet` quando questi non sono più necessari nell'applicazione.



Caratteristiche del ResultSet: » Per impostazione predefinita tutti i `ResultSet` creati hanno una tipologia di solo inoltro, una simultaneità di solo lettura e i cursori sono congelati sui limiti della

sincronizzazione. Un'eccezione a questo risulta nelle modifiche che WebSphere apporta correntemente all'impostazione predefinita del conservabilità del cursore in modo tale che i cursori siano implicitamente chiusi quando vengono sincronizzati. 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 (JavaTM Database Connectivity) 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 in vari modi un ResultSet. Questo tipo di cursore non è sensibile alle modifiche apportate al database mentre è aperto. Esso contiene righe che soddisfano l'interrogazione quando l'interrogazione 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 sempre un'impostazione di solo inoltro. I cursori che si possono aggiornare sono aggiunti in JDBC 2.0.

Nota: il blocco abilitato e le proprietà di collegamento alla 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 dell'unità di controllo JDBC stessa.

Consultare Esempio: ResultSet sensibili e non sensibili che indica la differenza tra i ResultSet sensibili e non sensibili quando le righe vengono inserite in una tabella.

Consultare Esempio: sensibilità del ResultSet che indica il modo in cui una modifica può influire su una clausola where di un'istruzione SQL basata sulla sensibilità del ResultSet.

Simultaneità: La simultaneità determina se è possibile aggiornare il ResultSet. I tipi vengono nuovamente definiti da costanti nell'interfaccia ResultSet. Le impostazioni disponibili della simultaneità sono le seguenti:

CONCUR_READ_ONLY

E' 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. E' possibile posizionare queste modifiche nel database sottostante. Consultare Modificare i ResultSet per ulteriori informazioni.

I ResultSet JDBC 1.0 hanno sempre un'impostazione di solo inoltro. ResultSet aggiornabili sono stati aggiunti nel JDBC 2.0.

Nota: secondo la specifica JDBC, è consentito all'unità 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, l'unità di controllo JDBC invia un messaggio di attenzione 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, l'unità di controllo modifica la sensibilità a TYPE_SCROLL_SENSITIVE e crea un messaggio di attenzione indicante che la richiesta è stata modificata.

Conservabilità: La caratteristica di conservabilità determina se la chiamata della sincronizzazione sull'oggetto Connection chiude il ResultSet. L'API JDBC per la gestione di tale caratteristica è nuova nella versione 3.0. L'unità di controllo JDBC nativo, tuttavia, fornisce una proprietà di collegamento per numerosi rilasci che consentono di specificare quell'impostazione predefinita per tutti i ResultSet creati sotto il collegamento (consultare Proprietà Connection e Proprietà DataSource). 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 il JDBC nativo.

CLOSE_CURSORS_ON_COMMIT

Tutti i cursori aperti vengono chiusi quando viene richiamata la clausola commit.

Nota: il richiamo del ripristino allo stato precedente 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 l'unità di controllo che si sta gestendo esegue se vengono gestiti cursori nei limiti di sincronizzazione. L'unità di controllo di JDBC del Toolbox utilizza anche il valore predefinito HOLD_CURSORS_ON_COMMIT, ma l'unità di controllo JDBC relativa a UDB di Windows^(R) NT ha un'impostazione predefinita di CLOSE_CURSORS_ON_COMMIT.

Specificare le caratteristiche del 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. E' possibile specificare queste caratteristiche tramite variazioni sovraccariche dei metodi createStatement, prepareStatement e prepareCall.

Consultare gli argomenti seguenti per specificare le caratteristiche di ResultSet:

- Specificare le caratteristiche di ResultSet per Statement
- Specificare le caratteristiche di ResultSet e il supporto di chiavi create automaticamente per PreparedStatement
- Specificare le caratteristiche di ResultSet e il supporto di chiavi create automaticamente per CallableStatement

Nota: esistono metodi ResultSet per ottenere il tipo di ResultSet e la simultaneità del ResultSet, ma non esiste alcun metodo per ottenere il congelamento del ResultSet. <<

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.

Esempio: ResultSet sensibili e non sensibili

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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 basata sulla sensibilità del ResultSet.

Esempio: sensibilità del ResultSet

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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)");

    } 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: ➤ Il metodo `ResultSet.next` viene utilizzato per spostare una riga alla volta tramite un `ResultSet`. Con JDBC (JavaTM Database Connectivity) 2.0, le unità di controllo JDBC `iSeries`

supportano ResultSet che si possono scorrere. 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(). E' 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. <ul style="list-style-type: none"> • 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".

Metodo	Descrizione
Absolute (int row)	<p>Il metodo sposta il cursore sulla riga specificata dal valore della riga.</p> <p>Se il valore della riga è positivo, il cursore viene posizionato sulla riga corrispondente al numero di righe indicate dal valore a partire dall'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.</p> <p>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.</p> <p>Se il valore della riga è 0, questo metodo opera come beforeFirst.</p> <p>Il metodo restituisce "true" se il cursore viene posizionato su una riga valida, altrimenti restituisce "false".</p>



Richiamare i dati del ResultSet: ➤ L'oggetto del ResultSet fornisce numerosi metodi per ottenere dati di colonne per una riga. Tutti sono nel formato `get<Type>`, dove `<Type>` rappresenta un tipo di dati JavaTM. 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 e 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 a Javadoc per il pacchetto `java.sql` per dettagli esaurienti.

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. E' 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.

Consultare Esempio: interfaccia ResultSet per IBM Developer Kit per Java per un esempio sul modo in cui utilizzare l'interfaccia ResultSet.

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 reperire i dati. Il codice seguente utilizza `ResultSetMetaData` per determinare ogni tipo di colonna nella serie di risultati:

Esempio: utilizzare `ResultSetMetaData` per determinare ogni tipo di colonna in una serie di risultati

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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);
```

Consultare Esempio: interfaccia ResultSetMetaData per IBM Developer Kit per Java per un esempio sul modo in cui utilizzare l'interfaccia ResultSetMetaData.



Modificare ResultSet: ➤ L'impostazione predefinita relativa ai ResultSet è di sola lettura. Tuttavia con JDBC (JavaTM Database Connectivity) 2.0, le unità di controllo JDBC iSeries forniscono un supporto completo per ResultSet aggiornabili. E' possibile fare riferimento alla simultaneità del ResultSet per aggiornare i ResultSet.

Aggiornare le righe: E' 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 reperimento 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.

E' 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 quale riga è stata aggiornata. In modo corrispondente, anche il metodo updatesAreDetected restituisce "false".

Cancellare le righe: E' 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: E' 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.

Supportare gli aggiornamenti della posizione: 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 l'unità di controllo JDBC nativo.

Consultare Esempio: modificare valori con un'istruzione tramite un altro cursore di istruzione per ulteriori informazioni.

Consultare Esempio: eliminare valori da una tabella tramite un altro cursore di istruzione per ulteriori informazioni. <<

Esempio: eliminare i valori da una tabella tramite un altro cursore dell'istruzione: >> Questo è un esempio di come eliminare i valori da una tabella tramite un altro cursore dell'istruzione.

Esempio: eliminare i valori da una tabella tramite un altro cursore dell'istruzione

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 l'unità 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: modificare i valori con un'istruzione tramite un altro cursore dell'istruzione: ➤ Questo è un esempio di come modificare i valori con un'istruzione tramite un altro cursore dell'istruzione.

Esempio: modificare i valori con un'istruzione tramite un altro cursore dell'istruzione

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 l'unità 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 ();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

Creare ResultSet: ➤ Per creare un oggetto ResultSet, è possibile utilizzare i metodi `executeQuery` dalle interfacce `Statement`, `PreparedStatement` o `CallableStatement`. Esistono, tuttavia, altri metodi disponibili. Per esempio, i metodi `DatabaseMetaData` come `getColumns`, `getTables`, `getUDTs`, `getPrimaryKeys` e così di seguito, restituiscono `ResultSet`. E' anche possibile avere una singola istruzione SQL che restituisce più `ResultSet` per l'elaborazione. E' possibile inoltre utilizzare il metodo `getResultSet` per reperire un oggetto `ResultSet` dopo la chiamata al metodo `execute` fornito dalle interfacce `Statement`, `PreparedStatement` o `CallableStatement`.

Consultare Esempio: creare una procedura con più `ResultSet` per ulteriori informazioni.

Chiudere 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.

E' 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 l'unità 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. E' consentita la chiamata della chiusura più volte su un `ResultSet`. ⏪

Esempio: interfaccia ResultSet di IBM Developer Kit per Java: Questo è un esempio del modo in cui utilizzare l'interfaccia `ResultSet`.

Esempio 1: interfaccia ResultSet

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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: interfaccia ResultSetMetaData di IBM Developer Kit per Java: Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.sql.*;
```

```
/**
ResultSetMetaDataExample.java
```

Questo programma mostra l'uso di un ResultSetMetaData e di un ResultSet per visualizzare tutti i metadata relativi ad un ResultSet creato interrogando una tabella. L'utente inoltra 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");
                }
            }
        }
    }
}

```

Lotto di oggetti JDBC

» La creazione di lotti di oggetti è l'argomento più comune da affrontare quando si parla di JDBC (JavaTM Database Connectivity) e di 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.

Per utilizzare la creazione di lotti di oggetti nei propri programmi JDBC, esaminare quanto segue:

Utilizzare il supporto DataSource per la creazione di lotti di oggetti

E' 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.

Proprietà ConnectionPoolDataSource

E' possibile configurare l'interfaccia ConnectionPoolDataSource utilizzando la serie di proprietà che essa fornisce.

Creare lotti di istruzioni basate su DataSource

E' possibile utilizzare la creazione di lotti di istruzioni con un lotto di collegamenti. La proprietà maxStatements dell'interfaccia UDBCConnectionPoolDataSource consente a DataSource di specificare quante istruzioni è possibile inserire in un lotto durante un collegamento.

Creare la propria soluzione di creazione di lotti

E' 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.



Utilizzare il supporto DataSource per la creazione di lotti di oggetti: ➤ L'utilizzo di DataSource consente di avere più applicazioni che condividono una configurazione comune per l'accesso al 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 dell'unità 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.

Impostare la 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. E' possibile inoltre consultare la JNDI (Java Naming and Directory Interface) per ulteriori dettagli sul ruolo che JNDI assume in questo 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.

Fare riferimento a Esempio: sottoporre a verifica le prestazioni del lotto di collegamenti per ulteriori dettagli.

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
```

```
Java program completed.
```

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 si ha bisogno di collegamenti simultanei, è necessario configurare ConnectionPoolDataSource per mettere in corrispondenza le necessità dell'utente e le risorse. <<

Esempio: impostare il lotto di collegamenti con UDBDataSource e

UDBConnectionPoolDataSource: >> Questo è un esempio della modalità di utilizzo del lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource.

Esempio: impostare il lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 il sorgente dati lotto di collegamenti
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Creare un sorgente dati standard che vi faccia riferimento.
```

```

        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting pooling");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
    }
}

```



Esempio: eseguire la 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 non di non creazione di lotti.

Esempio: sottoporre a verifica le prestazioni della creazione di lotti di collegamenti

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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: ➤ L'interfaccia ConnectionPoolDataSource fornisce una serie di proprietà per la configurazione. Le descrizioni di queste proprietà vengono fornite nella seguente tabella.

Proprietà	Descrizione
initialPoolSize	Quando un lotto viene inizializzato per primo, 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 connessioni iniziale da creare.
maxPoolSize	<p>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.</p> <p>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, l'unità di controllo JDBC crea un nuovo collegamento basato sulle proprietà DataSource e restituisce il collegamento.</p> <p>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.</p>
minPoolSize	<p>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.</p> <p>In tali casi, l'unità 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.</p> <p>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.</p>
maxIdleTime	<p>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).</p> <p>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.</p>
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 l'unità 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 raggiunta la `minPoolSize`. ◀

Creare lotti di istruzioni basate su DataSource: ▶ Un'altra proprietà disponibile sull'interfaccia `UDBConnectionPoolDataSource` è `maxStatements`. Questa proprietà 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.

Esempio: la verifica delle prestazioni di due `DataSource` controlla un `DataSource` che utilizza solo la creazione di lotti di collegamenti e l'altro `DataSource` che utilizza la creazione di lotti di collegamenti e di lotti di istruzioni.

L'esempio seguente è emesso dall'esecuzione di questo programma in modo locale 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: effettuare la verifica delle prestazioni di due DataSource: ▶ Questo è un esempio di verifica su un `DataSource` che utilizza solo la creazione di lotti di collegamenti e un altro `DataSource` che utilizza la creazione di lotti di collegamenti e istruzioni.

Esempio: effettuare la verifica delle prestazioni di due `DataSource`

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 il sorgente dati 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);
}
}


```

Creare il lotto di collegamenti: ➤ E' possibile sviluppare la creazione di lotti di istruzioni e collegamenti senza richiedere alcun supporto per DataSource o facendo affidamento su un altro prodotto.

Le tecniche della creazione di lotti vengono dimostrate su una piccola applicazione Java, ma sono ugualmente applicabili ai servlet o alle applicazioni a più livelli di grandi dimensioni. Questo esempio viene utilizzato per mostrare le considerazioni riguardo le prestazioni.

L'applicazione della dimostrazione ha due funzioni:

- Inserire un nuovo indice e un nuovo nome all'interno di una tabella database.
- Leggere il nome relativo a un dato indice da una tabella.


E' possibile scaricare il codice completo per l'applicazione dalla pagina web JDBC Developer Kit per Java di IBM .

L'applicazione dell'esempio non è eseguita in modo soddisfacente. L'esecuzione di 100 chiamate al metodo `getValue` e di 100 chiamate al metodo `putValue` tramite questo codice impiega una media di 31.86 secondi su una stazione di controllo standard.

Il problema è che si verifica un sovraccarico di lavoro del database 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, bisognerebbe creare 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.

Esiste una modifica per il codice chiamante relativo al metodo `getValue`. Il metodo `putValue` non è visualizzato, ma la modifica esatta viene apportata ad esso ed è disponibile dalla pagina web JDBC

Developer Kit per Java di IBM.  Anche la creazione di istanze dell'oggetto del lotto di collegamenti non viene visualizzata. E' possibile chiamare il programma di creazione e inoltrare alcuni oggetti di collegamento che si desidera inserire nel lotto. E' necessario eseguire questa fase quando si inizializza l'applicazione.

L'esecuzione dell'applicazione precedente (cioè se si verificano 100 richieste del metodo `getValue` e 100 richieste del metodo `putValue`) con queste modifiche impiega una media di 13.43 secondi con il codice di creazione di lotti di collegamenti impostato. Senza la creazione di tale lotti si impiega meno della metà del tempo di elaborazione relativo al caricamento lavoro rispetto al tempo di elaborazione originale.

Creare il 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 altro 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 interrogazione:

```
SELECT * from employee where salary > 100000
```

L'iterazione seguente potrebbe generare la seguente interrogazione:

```
SELECT * from employee where salary > 50000
```

Questa è la stessa interrogazione, ma utilizza un parametro differente. E' possibile che entrambe le interrogazioni siano realizzate con la seguente interrogazione.

```
SELECT * from employee where salary > ?
```

E' possibile successivamente impostare il contrassegno di parametro (denotato dal punto interrogativo) su 100000 durante l'elaborazione della prima interrogazione e su 50000 durante l'elaborazione della seconda interrogazione. 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.
- E' 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 raccogliitore dati inutili Java e inoltre incrementa nel tempo le prestazioni rispetto a molti utenti.

E' possibile modificare il programma di dimostrazione per inserire nel lotto oggetti PreparedStatement piuttosto che Connection. La modifica del programma consente il riutilizzo di più oggetti e il miglioramento delle prestazioni. E' possibile iniziare registrando la classe che contiene gli oggetti da inserire nel lotto. E' necessario che questa classe comprima le varie risorse da utilizzare. Per quanto riguarda l'esempio del lotto di collegamenti, Connection era l'unica risorsa inserita nel lotto, per cui non esisteva alcuna necessità di avere una classe di compressione. E' necessario che ogni oggetto inserito nel lotto contenga una risorsa Connection e due risorse PreparedStatement. E' possibile successivamente creare una classe di lotti che contenga oggetti di accesso al database invece di collegamenti.

Infine, è necessario che l'applicazione venga modificata per ottenere un oggetto di accesso al database e per specificare quale risorsa derivata dall'oggetto si desidera utilizzare. L'applicazione rimane la stessa tranne che per l'individuazione della risorsa specifica.

Con questa modifica, la stessa verifica eseguita adesso impiega una media di 0.83 secondi. Il tempo di esecuzione è circa 38 volte più veloce della versione originale del programma.

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.

E' 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) è basato sull'SQL dinamico e tende ad essere lento. E' 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 possono essere regolate senza sacrificare la conservabilità del codice o della progettazione object-oriented. 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

>> Una nuova funzione in JDBC 2.0 è il supporto di aggiornamento batch. Questa funzione consente di passare qualsiasi aggiornamento al database come transazione singola tra il programma dell'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 tempo all'utente.

Esistono due tipi di aggiornamenti batch:

- Aggiornamenti batch che utilizzano oggetti Statement.
- Aggiornamenti batch che utilizzano oggetti PreparedStatement.

Per utilizzare il supporto di aggiornamento batch, consultare quanto segue:

Aggiornamento batch Statement

Prima di eseguire un'aggiornamento batch Statement, è necessario assicurarsi che la sincronizzazione automatica sia disattivata. Quando l'impostazione della sincronizzazione automatica è disattivata, è possibile creare un oggetto Statement standard. E' quindi possibile aggiungere le istruzioni al batch con il metodo `addBatch`. Una volta aggiunte tutte le istruzioni che si desidera al batch, è possibile elaborarle tutte con il metodo `executeBatch` o svuotare il batch in qualsiasi momento con il metodo `clearBatch`.

Aggiornamento batch PreparedStatement

Un batch `PreparedStatement` è simile al batch Statement. Tuttavia, un batch `PreparedStatement` funziona sempre fuori dalla stessa istruzione preparata e si modificano solo i parametri in quell'istruzione.

BatchUpdateException

Quando una chiamata al metodo `executeBatch` ha esito negativo, viene emessa una `BatchUpdateException`. La `BatchUpdateException` 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.

Supporto inserimento bloccato

E' possibile utilizzare un inserimento bloccato in un'operazione `iSeries` per inserire più righe in una tabella database alla volta.



Aggiornamento batch Statement: >> Per eseguire un aggiornamento batch Statement, è necessario disattivare la sincronizzazione automatica. In JDBC ovvero Java^(TM) Database Connectivity, la sincronizzazione automatica è attiva per impostazione predefinita. La sincronizzazione automatica 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 la sincronizzazione automatica e un'istruzione al centro del batch ha esito negativo, non è possibile ripristinare l'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. Consultare Transazioni per ulteriori dettagli.

Dopo aver disattivato la sincronizzazione automatica, 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. E' 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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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.

Esempio: aggiornamento batch PreparedStatement

Nota: consultare 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: ➤ 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.

Attualmente, le prestazioni standard dell'elaborazione degli aggiornamenti sottoposti a batch sono equivalenti a quelle dell'elaborazione di ogni istruzione in modo indipendente. E' 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 su la 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, l'unità 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.



Supporto di inserimento bloccato: ➤ Un **inserimento bloccato** è un tipo di operazione speciale su un server iSeries che fornisce un metodo altamente ottimizzato per inserire più righe in una tabella database alla volta. E' 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; l'unità di controllo JDBC nativa è stata modificata per sfruttare questa funzione.

A causa di limitazioni del sistema durante l'utilizzo del supporto di inserimento bloccato, l'impostazione predefinita per l'unità di controllo JDBC nativa deve avere l'inserimento bloccato disabilitato. E' possibile abilitarlo tramite la proprietà Connection o DataSource. E' 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. La lista di limitazioni è la seguente:

- L'istruzione SQL utilizzata deve essere un'istruzione INSERT con una clausola VALUES, ad indicare che non è un'istruzione INSERT con SUBSELECT. L'unità di controllo JDBC riconosce questa limitazione ed agisce di conseguenza.
- E' necessario utilizzare PreparedStatement, ad indicare che non esiste un supporto ottimizzato per gli oggetti dell'istruzione. L'unità 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. L'unità 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.
- E' 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. L'unità 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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// 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 dove esiste una quantità considerevole di lavoro in esecuzione, sarebbe opportuno modificare le aspettative in modo appropriato.



Tipi di dati avanzati

» Esistono numerosi tipi di dati nuovi denominati SQL3 forniti nel database iSeries con V4R4 e-PACK. JDBC Java^(TM) Database Connectivity 2.0 e successive fornisce il supporto per gestire questi tipi di dati che sono parte dello standard SQL99.

I tipi di dati SQL3 forniscono all'utente estrema flessibilità. Essi sono ideali per gli oggetti Java serializzati di memorizzazione, documenti XML (Extensible Markup Language) e dati multimediali come canzoni, immagini del prodotto, fotografie di impiegati e filmati.

Tipi "distinct": Il tipo "distinct" 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 "distinct" è sempre correlato ad un tipo di dati incorporati. Per ulteriori informazioni su come e quando utilizzare tipi "distinct" nel contesto di SQL, consultare i manuali di riferimento SQL.

Per utilizzare tipi "distinct" in JDBC, l'utente vi accede nello stesso modo in cui accede ad un tipo sottostante. Il metodo getUDT è nuovo e consente di richiedere quali tipi "distinct" sono disponibili sul sistema. Questo programma di esempio mostra quanto segue:

- La creazione di un tipo "distinct".
- La creazione di una tabella che utilizza tale tipo.
- L'utilizzo di PreparedStatement per impostare un parametro del tipo "distinct".
- L'utilizzo di ResultSet per restituire un tipo "distinct".
- L'utilizzo della chiamata API (Application Programming Interface) metadata a getUDT per informazioni su un tipo "distinct".

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 un supporto solo 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.

E' possibile utilizzare i BLOB nello stesso modo in cui si utilizzerebbero gli altri tipi di database standard. E' possibile passarli a procedure memorizzate, utilizzarli in istruzioni preparate e aggiornarli in serie di risultati. La classe PreparedStatement dispone di un metodo setBlob per passare i BLOB al database e la classe ResultSet aggiunge una classe getBlob per reperirli dal database. Un BLOB viene rappresentato in un programma Java da un oggetto BLOB che è un'interfaccia JDBC.

Far riferimento a Scrivere il codice che utilizza i BLOB per ulteriori informazioni su come utilizzare i BLOB.

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.

Far riferimento a Scrivere il codice che utilizza i CLOB per ulteriori informazioni su come utilizzare i CLOB.


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.

Far riferimento a Scrivere il codice che utilizza i Datalink per ulteriori informazioni su come utilizzare i Datalink.

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, i server iSeries non supportano questi tipi. Quindi, l'unità di controllo JDBC non fornisce alcuna forma di supporto per essi.




Scrivere il 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^(TM) Database Connectivity. I seguenti argomenti discutono brevemente queste attività e includono esempi su come realizzarle.

Leggere i BLOB dal database e inserire i 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. Questo esempio mostra alcuni modi in cui è possibile inserire un BLOB nel database o reperirlo dal database.

Gestire l'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. Questo esempio mostra alcune delle attività comuni che è possibile eseguire utilizzando questa API. Consultare javadoc JDBC per una lista completa dei metodi disponibili sull'oggetto Blob.

Utilizzare il 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. Questo esempio mostra alcune delle attività comuni che è possibile eseguire con un supporto BLOB in JDBC 3.0.



Esempio: BLOB:  Questo è un esempio del modo in cui è possibile inserire un BLOB nel database o reperirlo dal database.

Esempio: BLOB

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////  
// 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 l'unità di controllo JDBC nativa.
    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 portabile
    // in tutte le unità di controllo JDBC. Queste ultime non dispongono di
    // supporto quando si usa setBytes per le colonne BLOB. Viene utilizzato
    // per consentire la creazione di nuovi BLOB. Consente inoltre alle unità 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 un'interrogazione 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: aggiornare i BLOB: ➤ Questo è un esempio di come aggiornare i BLOB nelle proprie applicazioni.

Esempio: aggiornare i BLOB

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////
// 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 l'unità di controllo JDBC nativa.
        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 bytes 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: utilizzare i BLOB: ➤ Questo è un esempio della modalità di utilizzo dei BLOB nelle proprie applicazioni.

Esempio: utilizzare i BLOB

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
// 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 l'unità di controllo JDBC nativa.
        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.
    }
}

```

Scrivere il codice che utilizza i CLOB: ➤ Esistono alcune attività che è possibile eseguire con le colonne CLOB e DBCLOB del database tramite l'API (Application Programming Interface) JDBC Java^(TM) Database Connectivity. I seguenti argomenti discutono brevemente queste attività e includono esempi su come realizzarle.

Leggere i CLOB dal database e inserire i 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. Questo esempio mostra alcuni modi comuni in cui è possibile inserire un CLOB nel database o reperirlo dal database.

Gestire l'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. Questo esempio mostra alcune delle attività comuni che è possibile eseguire utilizzando questa API. Consultare Javadoc JDBC per una lista completa dei metodi disponibili sull'oggetto Clob.

Utilizzare il 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. Questo esempio mostra alcune delle attività che è possibile eseguire con un supporto CLOB in JDBC 3.0.



Esempio: CLOB: ➤ Questo è un esempio del modo in cui è possibile inserire un CLOB in un database o reperirlo da un database.

Esempio: CLOB

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////
// 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 l'unità di controllo JDBC nativa.
        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 portabile
// in tutte le unità di controllo JDBC. Queste ultime 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 un'interrogazione 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: aggiornare i CLOB: ➤ Questo è un esempio di come aggiornare i CLOB nelle proprie applicazioni.

Esempio: aggiornare i CLOB

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
// 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 l'unità di controllo JDBC nativa.
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: utilizzare i CLOB: ➤ Questo è un esempio della modalità di utilizzo dei CLOB nelle proprie applicazioni.

Esempio: utilizzare i CLOB

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
// 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 l'unità di controllo JDBC nativa.
        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.
}
}

```

Scrivere il codice che utilizza i Datalink: ➤ Il modo in cui si gestiranno i Datalink dipende dal rilascio 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.

Questo esempio mostra alcune delle attività di base della gestione delle colonne Datalink.



Esempio: Datalink: ➤ Questo è un esempio del modo in cui utilizzare i datalink nelle applicazioni.

Esempio: Datalink

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
// 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 l'unità di controllo JDBC nativa.
        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 articolo 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 un'interrogazione 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 il
        // 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 Distinct: ➤ Questo è un esempio della modalità di utilizzo dei tipi "distinct".

Esempio: tipi Distinct

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
// Questo esempio di programma mostra esempi
// di diverse attività comuni che possono
// essere eseguite con tipi distinct.
////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
        throws SQLException
    {
        // Registrare l'unità di controllo JDBC nativa.
        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();

        // E' 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

» 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 un'implementazione attuale. 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.

Per implementare i Rowset utilizzando l'unità di controllo JDBC nativa, consultare quanto segue:

Caratteristiche del RowSet

E' possibile richiedere che alcune proprietà vengano soddisfatte dai RowSet. Le proprietà comuni includono la serie di interfacce da supportare tramite il rowset risultante.

DB2JdbcRowSet

Il DB2JdbcRowSet è un RowSet collegato come un wrapper su un DB2ResultSet e fornisce supporto di gestione evento.

DB2CachedRowSet

DB2CachedRowSet è un RowSet non collegato che consente la memorizzazione dei dati DB2ResultSet all'interno dell'oggetto. Una volta che i dati sono inseriti all'interno dell'oggetto, è possibile chiudere l'oggetto DB2Connection sottostante e il DB2CachedRowSet può continuare ad essere utilizzato. Individuare le seguenti informazioni riguardanti DB2CachedRowSet:

- Utilizzare DB2CachedRowSet
- Creare e popolare un DB2CachedRowSet
- Accedere ai dati di DB2CachedRowSet e alla manipolazione del cursore
- Modificare i dati DB2CachedRowSet e riflettere le modifiche riportate nel sorgente dati
- Altre caratteristiche di DB2CachedRowSet



Caratteristiche del RowSet: » E' 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.

E' possibile che i RowSet non siano collegati al database: Esistono due categorie di RowSet:

- **Collegati**

Sebbene i RowSet collegati vengano popolati con dati, questi hanno sempre collegamenti aperti al database sottostante e servono come wrapper per l'implementazione di ResultSet.

- **Scollegati**

I RowSet scollegati non sono necessari per mantenere i collegamenti al sorgente dati in qualsiasi momento. E' 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 RowSet possiedono il supporto per la gestione dell'evento in base al modello di gestione dell'evento JavaBeans. Possiedono inoltre proprietà che è possibile impostare. E' 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: E' possibile serializzare e non i RowSet per consentire a questi di scorrere su un collegamento di rete, è possibile scriverli su un file semplice (cioè un documento di testo senza alcuna elaborazione di parola o altri caratteri di struttura), e così via. <<

DB2CachedRowSet: >> L'oggetto DB2CachedRowSet è un RowSet scollegato, che indica 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.

Utilizzare DB2CachedRowSet

E' possibile utilizzare metodi forniti da DB2CachedRowSet per migliorare le prestazioni del proprio database consentendo a vari utenti di utilizzare gli stessi dati. E' inoltre possibile distribuire i ResultSet comuni ai client creando una copia dei dati della tabella che non vengono modificati.

Creare e popolare un DB2CachedRowSet

Scoprire come creare e inserire dati in un DB2CachedRowSet seguendo queste attività:

- Utilizzare il metodo populate
- Utilizzare le proprietà DB2CachedRowSet e DataSource
- Utilizzare le proprietà DB2CachedRowSet e gli URL JDBC
- Utilizzare il metodo setConnection(Connection) per usare un collegamento database esistente
- Utilizzare il metodo execute(Connection) per usare un collegamento database esistente
- Utilizzare il metodo execute(int) per raggruppare le richieste database

Accedere ai dati DB2CachedRowSet e alla 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.

Modificare i dati DB2CachedRowSet e riflettere le modifiche riportate nel sorgente dati

Il DB2CachedRowSet utilizza gli stessi metodi dell'interfaccia ResultSet standard per effettuare le modifiche ai dati nell'oggetto RowSet. Il DB2CachedRowSet fornisce il metodo acceptChanges utilizzato per riflettere le modifiche al RowSet riportate nel database da cui i dati provengono.

Altre funzioni DB2CachedRowSet

La classe DB2CachedRowSet dispone di una funzionalità aggiuntiva che rende il suo utilizzo più flessibile. Con i metodi forniti da DB2CachedRowSet, è possibile effettuare le seguenti attività:

- Ottenere le raccolte dai DB2CachedRowSet

- Creare copie di RowSet
- Creare condivisioni per RowSet



Utilizzare DB2CachedRowSet: ➤ Dal momento che non è possibile scollegare e mettere in serie l'oggetto DB2CachedRowSet, è utile negli ambienti dove non è sempre pratico eseguire un'unità di controllo JDBC completa (ad esempio, su PDA (Personal Digital Assistants) e su telefoni cellulari abilitati Java^(TM)).

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 dell'unità 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.

E' 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 un'interrogazione al database ogni volta.

E' 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 l'interrogazione 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 l'interrogazione o memorizzare una notevole quantità di informazioni in memoria. Quando è corretto, è possibile eseguire nuovamente l'interrogazione 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 dell'interrogazione al database come una singola richiesta. Questo è un esempio di utilizzo dei DB2CachedRowSet per eliminare una parte del sovraccarico a 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.

E' 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 Esempio: eventi DB2JdbcRowSet per dettagli.

Per dettagli specifici sulla gestione dei DB2CachedRowSet, consultare i seguenti argomenti:

- Creare e popolare un DB2CachedRowSet
- Accedere ai dati di DB2CachedRowSet e alla manipolazione del cursore
- Modificare i dati DB2CachedRowSet e riflettere le modifiche riportate nel sorgente dati
- Altre caratteristiche di DB2CachedRowSet

Per informazioni sulla gestione e il modello dell'evento, consultare DB2JdbcRowSet dal momento che questo supporto funziona in modo identico per entrambi i tipi di RowSet. <<

Creare e popolare un DB2CachedRowSet: >> Esistono vari modi per inserire i dati in un DB2CachedRowSet:

- Utilizzare il metodo populate
- Utilizzare le proprietà DB2CachedRowSet e DataSource
- Utilizzare le proprietà DB2CachedRowSet e gli URL JDBC
- Utilizzare il metodo setConnection(Connection) per utilizzare un collegamento database esistente
- Utilizzare il metodo execute(Connection) al fine di utilizzare un collegamento database esistente
- Utilizzare il metodo execute(int) per raggruppare richieste database

Warning: Temporary Level 6 Header:

Utilizzare il 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: utilizzare il metodo populate

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// Stabilire un collegamento al database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Creare un'istruzione ed utilizzarla per eseguire l'interrogazione.
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();
```

Utilizzare le proprietà DB2CachedRowSet e DataSource: I DB2CachedRowSet dispongono di proprietà che consentono ai DB2CachedRowSet di accettare un'interrogazione SQL e un nome DataSource. Quindi essi utilizzano l'interrogazione 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: utilizzare le proprietà DB2CachedRowSet e DataSource

Nota: consultare 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 l'interrogazione 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();
```

Utilizzare le proprietà DB2CachedRowSet e gli URL JDBC: I DB2CachedRowSet dispongono di proprietà che consentono ai DB2CachedRowSet di accettare un'interrogazione SQL e un URL JDBC. Quindi essi utilizzano l'interrogazione e l'URL JDBC per creare dati per il proprio utilizzo. Segue un esempio di questo approccio.

Esempio: utilizzare le proprietà DB2CachedRowSet e gli URL JDBC

Nota: consultare 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 l'interrogazione 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();
```

Utilizzare il metodo setConnection(Connection) in modo da utilizzare un collegamento database esistente: Per favorire il riutilizzo degli oggetti Connection JDBC, DB2CachedRowSet fornisce un meccanismo per passare 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: utilizzare il metodo setConnection(Connection) per usare un collegamento database esistente

Nota: consultare 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 necessario 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();
```

Utilizzare il metodo execute(Connection) al fine di utilizzare un collegamento database esistente: Per favorire il riutilizzo degli oggetti Connection JDBC, il DB2CachedRowSet fornisce un meccanismo per passare 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: utilizzare il metodo execute(Connection) per usare un collegamento database esistente

Nota: consultare 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();
```

Utilizzare il 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: utilizzare il metodo `execute(int)` per raggruppare le richieste database

Nota: consultare 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 l'interrogazione 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();
```

Accedere ai dati del DB2CachedRowSet e alla manipolazione del cursore: ➤ I RowSet dipendono dai metodi ResultSet. Per molte operazioni come accesso ai dati DB2CachedRowSet spostamenti del cursore, non esiste alcuna differenza al livello di applicazione tra l'utilizzo di ResultSet e l'utilizzo di RowSet.

Accedere 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: accedere ai dati DB2CachedRowSet

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.*;
import java.math.*;

public class TestProgram
{
    public static void main(String args[])
    {
        // Registrare l'unità 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: E' 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 l'unità 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 (coll 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");
ove 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());
    }
}
}
}

```



Modificare i dati DB2CachedRowSet e riflettere le modifiche riportate nel sorgente dati: ➤ Il DB2CachedRowSet utilizza gli stessi metodi dell'interfaccia ResultSet standard per effettuare le modifiche ai dati nell'oggetto RowSet. Non esiste alcuna differenza al 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.

Cancellare, inserire e aggiornare le righe in un DB2CachedRowSet: E' 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: cancellare, inserire e aggiornare righe in un DB2CachedRowSet

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.DB2CachedRowSet;

public class RowSetSample2
{
    public static void main(String args[])
    {
        // Registrare l'unità 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 (coll 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");

ove    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());
}
}
}

```

Riflettere le 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: riflettere le modifiche ad un DB2CachedRowSet sul database sottostante

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.DB2CachedRowSet;

public class RowSetSample3
{
    public static void main(String args[])
    {
        // Registrare l'unità 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");

ove    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());
}
}
}

```



Altre caratteristiche del DB2CachedRowSet: ➤ 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 ovvero Java^(TM) 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:

- Ottenere le raccolte dai DB2CachedRowSet
- Creare copie di RowSet
- Creare condivisioni per RowSet

Ottenere le 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: ottenere raccolte dai DB2CachedRowSet

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.DB2CachedRowSet;
import java.util.*;

public class RowSetSample4
{
    public static void main(String args[])
    {
        // Registrare l'unità 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());
}
}
}

```

Creare 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: creare copie 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 l'unità 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 (coll 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");
    }
    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();
}
}
}
}

```

Creare 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: creare 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 l'unità 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 (coll 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");

    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.

Utilizzare DB2JdbcRowSet: Dal momento che l'oggetto DB2JdbcRowSet supporta eventi descritti nella specifica 3.0 di JDBC ovvero Java^(TM) Database Connectivity per tutti i RowSet, è possibile che esso funzioni come oggetto intermedio tra un database locale e altri oggetti a 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. E' 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.

Creare JDBCRowSet: Esistono vari metodi forniti per creare un oggetto DB2JDBCRowSet. Ogni oggetto viene delineato come segue.

Utilizzare le proprietà DB2JdbcRowSet e DataSource: I DB2JdbcRowSet dispongono di proprietà che accettano un'interrogazione SQL e un nome DataSource. E' 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: utilizzare le proprietà DB2JdbcRowSet e DataSource

Nota: consultare 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 l'interrogazione 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 un'interrogazione SQL e un URL JDBC. E' possibile quindi utilizzare i DB2JdbcRowSet. Segue un esempio di questo approccio:

Esempio: utilizzare le proprietà DB2JdbcRowSet e gli URL JDBC

Nota: consultare 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 l'interrogazione 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 usare un collegamento database esistente: Per favorire il riutilizzo degli oggetti Connection JDBC, il DB2JdbcRowSet consente di passare un collegamento stabilito al DB2JdbcRowSet. DB2JdbcRowSet utilizza tale collegamento per prepararsi ad usarlo quando viene chiamato il metodo execute.

Esempio: utilizzare il metodo setConnection

Nota: consultare 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();
```

Accedere 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.

Modificare i dati e riflettere le 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. E' 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: 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.DB2JdbcRowSet;

public class RowSetEvents {
    public static void main(String args[])
    {
        // Registrare l'unità 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 l'unità di controllo JDBC di IBM Developer Kit per Java

L'unità di controllo JDBC di IBM Developer Kit per Java^(TM) viene progettata in modo che sia un'interfaccia Java di 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 l'unità di controllo JDBC. I seguenti suggerimenti sono considerati una buona pratica di programmazione JDBC. La maggior parte di essi non sono specifici per l'unità di controllo JDBC nativa. Quindi le applicazioni scritte secondo queste direttive forniscono delle buone prestazioni anche se utilizzate con unità di controllo JDBC diverse dall'unità di controllo JDBC nativa.

- Evitare le interrogazioni SQL SELECT *
- Utilizzare getXXX(int) invece di getXXX(String)
- Evitare le chiamate getObject per i tipi primitivi Java
- Utilizzare PreparedStatement piuttosto che Statement
- Evitare chiamate DatabaseMetaData dispendiose
- Utilizzare il livello di sincronizzazione corretto per la propria applicazione
- Considerare la memorizzazione dei dati in Unicode
- Utilizzare le procedure memorizzate
- Utilizzare BigInt invece di Numeric/Decimal
- Chiudere le proprie risorse JDBC esplicitamente quando si è terminato di utilizzarle
- Utilizzare la creazione di lotti di collegamenti
- Considerare l'utilizzo della creazione di lotti PreparedStatement
- Utilizzare un SQL efficace

Evitare le interrogazioni SELECT * SQL

SELECT * FROM... è una modalità comune per specificare un'interrogazione in SQL. Spesso, non è necessario interrogare esplicitamente tutti i campi. Per ogni colonna da restituire, l'unità 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 l'unità 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...
```

Utilizzare 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 dall'unità 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.

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 raccoglitore di dati inutili, oltre a peggiorare le prestazioni.

Utilizzare `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 (esaminare Creare lotti di `PreparedStatement`).

Evitare le chiamate `DatabaseMetaData`

E' 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.

Utilizzare il livello di sincronizzazione corretto per la propria applicazione

JDBC fornisce numerosi livelli di sincronizzazione 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 sincronizzazione minimo. Ciò significa che le transazioni possono esaminare una parte del lavoro reciproco tramite limiti di sincronizzazione. Ciò introduce la possibilità di alcune anomalie del database. Alcuni programmatori aumentano il livello di sincronizzazione in modo da non preoccuparsi quando si verificano queste anomalie. Occorre tenere presente che livelli più elevati di sincronizzazione 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.

Considerare la memorizzazione dei dati in Unicode

Java richiede che tutti i dati di caratteri che gestisce (`Stringhe`) siano in Unicode. Quindi, qualsiasi tabella che non dispone di dati Unicode richiede che l'unità di controllo JDBC converta i dati in un senso e nell'altro, in modo da inserirli nel database e da reperirli da esso. Se la tabella è già in Unicode, l'unità 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.

Utilizzare le procedure memorizzate

L'utilizzo delle procedure memorizzate è supportato in Java. Le procedure memorizzate possono essere eseguite più rapidamente consentendo all'unità 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.

Utilizzare BigInt invece 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 rilevato in Evitare le chiamate getObject per i tipi primitivi Java, l'utilizzo di tipi di dati primitivi è preferibile rispetto all'utilizzo di tipi che richiedono una creazione dell'oggetto.

Chiudere esplicitamente le proprie risorse JDBC quando si è terminato di utilizzarle

E' 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.

Utilizzare la 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.

Considerare l'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 reperire tale oggetto ed accedere all'istruzione specifica che si desidera utilizzare. Ciò può migliorare considerevolmente le prestazioni.

Utilizzare 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 interrogazioni ottimizzate, dagli indici scelti in modo appropriato e da altri aspetti di una buona progettazione SQL.

Accedere ai database utilizzando il supporto DB2 SQLJ di IBM Developer Kit per Java

Il supporto DB2 SQLJ (Structured Query Language for Java[™]) si basa sullo standard ANSI SQLJ. Il supporto DB2 SQLJ è contenuto in IBM Developer Kit per Java. Tale supporto consente la creazione e l'esecuzione di applicazioni incorporate SQLJ.

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. Per ulteriori informazioni sulle classi del tempo di esecuzione SQLJ, fare riferimento alla documentazione dell'API del tempo di esecuzione fornita nell'implementazione reperibile in www.sqlj.org .

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 crea un profilo serializzato che contiene informazioni sulle operazioni SQLJ rilevate nel programma SQLJ.
- Il programma di personalizzazione del profilo DB2 SQLJ, `db2prof.c`, precompila le istruzioni SQL memorizzate nel profilo creato e crea un pacchetto nel database DB2.
- La Stampante di profilo DB2 SQLJ, `db2profp`, stampa il contenuto del profilo personalizzato DB2 in testo chiaro.
- 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 serializzato in 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 Supporto per più JDK (Java Development Kit) per ulteriori informazioni sull'esecuzione di più versioni di Java Development Kit.

Per informazioni sull'utilizzo di SQL nelle applicazioni Java, consultare Incorporare le istruzioni SQL nell'applicazione Java e Compilare ed eseguire programmi SQLJ.

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 creare i profili dal codice sorgente SQLJ, eseguire il programma di conversione SQLJ, `sqlj`, sul file `.sqlj`.

Per ulteriori informazioni, consultare Compilare ed eseguire i programmi SQLJ.

Il 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 sulle opzioni della riga comandi sqlj, fare riferimento al manuale SQLJ User's Guide and Reference fornito nell'implementazione reperita da www.sqlj.org .

Precompilare le istruzioni SQL in un profilo utilizzando il programma di personalizzazione del profilo DB2 SQLJ, db2profc

E' possibile utilizzare il programma di personalizzazione del profilo DB2 SQLJ, db2profc, per fare in modo che l'applicazione Java^(TM) 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 la lista di opzioni desiderata.

Le opzioni disponibili per db2profc sono le seguenti:

- -URL=<JDBC_URL>
- -user=<username>
- -password=<password>
- -package=<library_name/package_name>
- -commitctl=<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 Accedere al database iSeries con l'unità di controllo JDBC di IBM Developer Kit per Java.

-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

-commitctl=<commitment_control>

Dove *commitment_control* rappresenta il livello del controllo di sincronizzazione che si desidera. E' possibile che il controllo di sincronizzazione abbia uno qualunque dei seguenti valori del 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. E' possibile che il formato della data possegga uno qualunque dei seguenti valori:

Valore	Definizione
USA	IBM USA standard (mm.gg.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 (gg.mm.aaaa, hh.mm.ss)
JIS	Japanese Industrial Standard Christian Era (aaaa-mm-gg, hh:mm:ss)

Valore	Definizione
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. E' 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. E' possibile che il formato dell'ora assuma uno qualunque dei seguenti valori:

Valore	Definizione
USA	IBM USA standard (mm.gg.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 (gg.mm.aaaa, hh.mm.ss)
JIS	Japanese Industrial Standard Christian Era (aaaa-mm-gg, 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. E' 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.
,	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. E' 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.

Per informazioni più dettagliate su uno qualsiasi di questi campi, consultare il manuale DB2 for iSeries

SQL Programming Concepts, SC41-5611 

Stampare il contenuto dei profili DB2 SQLJ (db2profp e profp)

La stampante del profilo DB2 SQLJ, db2profp, stampa il contenuto del 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 [options] <SQLJ_profile_name>
```

Dove *SQLJ_profile_name* rappresenta il nome del profilo da stampare e *options* rappresenta la lista di opzioni desiderata.

Le opzioni disponibili per db2profp sono le seguenti:

-URL=<JDBC_URL>

Dove *JDBC_URL* rappresenta l'URL a cui ci si desidera collegare. Per ulteriori informazioni, consultare Accedere al database iSeries con l'unità di controllo JDBC di IBM Developer Kit per Java.

-user=<username>

Dove *username* rappresenta il nome dell'utente presente nel profilo utente.

-password=<password>

Dove *password* 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 ResultSet eseguite durante il tempo di esecuzione dell'applicazione vengono registrate. Queste possono essere registrate su un file o su un'emissione standard. E' 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 ResultSetcall 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.

Per ulteriori informazioni sulle opzioni della riga comandi profdb, consultare www.sqlj.org , selezionare la categoria Implementazione ed andare sulla classe sqlj.runtime.profile.util.AuditorInstaller che si trova nella documentazione API del tempo di esecuzione.

Convertire un'istanza di profilo serializzato 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 serializzato in un formato di classe Java^(TM). 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.

Incorporare le 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^(TM), 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.

Nota: per informazioni sulla compilazione e l'esecuzione delle applicazioni SQLJ, consultare *Compilare ed eseguire i programmi SQLJ*.

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) };
}
```

La struttura del metodo consiste di una clausola di elaborazione SQLJ contenente le variabili host x, y e z. Per ulteriori informazioni sulle variabili host, consultare *Variabili host in SQLJ*.

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. Durante 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:

- Interrogazioni
Ad esempio, istruzioni ed espressioni SELECT.
- Istruzioni di modifica dati SQL (DML)
Ad esempio, INSERT, UPDATE, DELETE.
- Istruzioni sui dati
Ad esempio, FETCH, SELECT..INTO.
- Istruzioni di controllo transazione
Ad esempio, COMMIT, ROLLBACK, etc.
- 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))

Per un esempio di SQLJ incorporate, consultare Esempio: incorporare le istruzioni SQL nell'applicazione Java.

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^(TM) che è un identificativo Java per un parametro, una variabile o un campo.
- Un identificativo della modalità del parametro facoltativo.

E' 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.

L'interrogazione seguente contiene la variabile host, :x. Questa variabile host è la variabile Java, il campo Java o il parametro Java x visibile nell'ambito contenente l'interrogazione.

```
SELECT COL1, COL2 FROM TABLE1 WHERE :x > COL3
```

Esempio: incorporare le istruzioni SQL nell'applicazione Java: La seguente applicazione SQLJ di esempio, App.sqlj, utilizza SQL statico per reperire e aggiornare i dati dalla tabella EMPLOYEE del database di esempio DB2.

Esempio: incorporare le istruzioni SQL nell'applicazione Java^(TM):

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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);
        }

        // reperire 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 (" firstnme= " + str2);
            System.out.println("");
        }
        cursor1.close(); // 9

        // reperire 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' };

    // reperire 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. **Dichiarare gli 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. **Inizializzare l'iteratore.** L'oggetto iteratore cursor1 viene inizializzato utilizzando il risultato di un'interrogazione. L'interrogazione memorizza il risultato in cursor1.
3. **Fare avanzare l'iteratore alla riga successiva.** Il metodo cursor1.next() restituisce il valore Booleano false se non esistono più righe da richiamare.
4. **Spostare i 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. **Inizializzare l'iteratore.** L'oggetto iteratore cursor2 viene inizializzato utilizzando il risultato di un'interrogazione. L'interrogazione memorizza il risultato in cursor2.
7. **Reperire i dati.** L'istruzione FETCH restituisce il valore corrente della prima colonna dichiarata nel cursore ByPos dalla tabella dei risultati nella variabile host str2.
8. **Controllare l'esito positivo 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 chiamato il metodo next(). Un'istruzione FETCH...INTO chiama implicitamente il metodo next().
9. **Chiudere gli iteratori.** Il metodo close() rilascia qualsiasi risorsa mantenuta dagli iteratori. E' necessario chiudere in modo esplicito gli iteratori per assicurarsi che le risorse di sistema vengano rilasciate in modo tempestivo.

Per le informazioni sul background in questo esempio, consultare Incorporare le istruzioni SQL nell'applicazione Java.

Compilare ed eseguire i programmi SQLJ

Se il programma Java[™] ha istruzioni SQLJ incorporate, è necessario seguire una procedura speciale per la compilazione e l'esecuzione.

Nota: prima di iniziare, impostare il CLASSPATH affinché contenga quanto segue:

- /QIBM/ProdData/Os400/Java400/ext/sqlj_classes.jar
- /QIBM/ProdData/Os400/Java400/ext/translator.zip
- /QIBM/ProdData/Os400/Java400/ext/runtime.zip

Per compilare ed eseguire il programma SQLJ (structured query language for Java), è necessario seguire queste fasi:

1. 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`.

2. Utilizzare lo strumento Programma di personalizzazione del profilo SQLJ, db2profrc, per installare i Programmi di personalizzazione DB2 SQLJ sui profili generati e creare i pacchetti DB2 sul sistema locale.

Ad esempio, immettere il comando:

```
db2profrc MyClass_SJProfile0.ser
```

dove *MyClass_SJProfile0.ser* rappresenta 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.

3. Eseguire il file di classe Java come un qualsiasi altro file di classe Java.

Ad esempio, immettere il comando:

```
java MyClass
```

dove *MyClass* rappresenta il nome del file di classe Java.

Routine SQL Java

» Il server iSeries fornisce la possibilità di accedere ai programmi Java^(TM) da programmi e istruzioni SQL. Questo può essere ottenuto utilizzando le procedure memorizzate Java e le UDF (user-defined function-funzioni definite dall'utente) Java. Il server iSeries supporta sia le convenzioni DB2 che SQLJ per la chiamata di procedure memorizzate Java e UDF Java. Sia le procedure memorizzate Java che le UDF Java possono utilizzare classi Java memorizzate nei file JAR. Il server iSeries utilizza procedure memorizzate definite dallo standard *SQLJ Parte 1* nella registrazione dei file JAR con il database.

Per accedere alle applicazioni Java dai programmi e dalle istruzioni SQL, consultare quanto segue:

Utilizzare le routine SQL Java

Attenersi alle seguenti fasi per utilizzare le routine SQL Java:

- Scrivere i metodi Java relativi alle routine.
- Compilare le classi Java.
- Rendere le classi compilate accessibili alla Java virtual machine utilizzata dal database.
- Registrare la routine con il database.
- Utilizzare la procedura SQL Java.

Procedure memorizzate Java

Quando viene utilizzato Java per scrivere procedure memorizzate, è possibile utilizzare i seguenti stili di inoltro dei parametri:

- Stile del parametro JAVA
- Stile del parametro DB2GENERAL

Funzioni scalari Java definite dall'utente

Una funzione scalare Java restituisce un valore da un programma Java al database. Come le procedure memorizzate Java, le funzioni scalari Java utilizzano uno dei due stili di parametro, JAVA e DB2GENERAL.

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.

Procedure SQLJ che manipolano i file JAR

Sia le procedure memorizzate Java che le UDF Java possono utilizzare le classi Java memorizzate nei file JAR Java. Individuare le seguenti informazioni sulle procedure SQLJ che manipolano i file JAR:

- SQLJ.INSTALL_JAR
- SQLJ.REMOVE_JAR
- SQLJ.REPLACE_JAR
- SQLJ.UPDATEJARINFO
- SQLJ.RECOVERJAR

Convenzioni di inoltro parametri per le procedure memorizzate e le UDF Java

Questa sezione descrive il modo in cui le tipologie di dati SQL vengono rappresentate nelle procedure memorizzate e nelle UDF Java.



Utilizzare le routine SQL Java

» E' possibile accedere ai programmi Java^(TM) dai programmi e dalle istruzioni SQL. E' 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, è necessario attenersi alle seguenti fasi:

1. Scrivere i metodi Java relativi alle routine.

Una routine SQL Java elabora un metodo Java da SQL. E' necessario registrare questo metodo utilizzando le convenzioni di inoltro dei parametri DB2^(R) o SQLJ. Consultare procedure memorizzate Java, funzioni Java definite dall'utente e funzioni di tabella Java definite dall'utente per ulteriori informazioni sulla codifica di un metodo utilizzato da una routine SQL Java.

2. Compilare le classi Java.

E' possibile compilare le routine SQL Java registrate utilizzando lo stile del parametro Java senza alcuna impostazione aggiuntiva. Tuttavia, è necessario che le routine SQL Java che utilizzano lo stile del 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 origine Java che contiene una routine che utilizza lo stile del parametro DB2GENERAL:

```
javac -DCLASSPATH=/QIBM/ProdData/Java400/ext/db2routines_classes.jar
source.java
```

3. Rendere le classi compilate accessibili alla JVM utilizzata dal database.

Le classi definite dall'utente utilizzate dalla JVM (Java virtual machine) 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 iSeries di `/sqllib/function`, l'indirizzario dove DB2 UDB 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 `runit.class` si trovi nell'indirizzario dell'IFS (Integrated File System), `/QIBM/ProdData/OS400/SQLLib/Function/foo/bar`.

E' 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 le procedure SQLJ che manipolano i file JAR per ulteriori informazioni su `SQLJ.INSTALL_JAR` così come altre procedure memorizzate per gestire i file JAR.

4. 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 conosciuta 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 del 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 una file classe ubicata nell'indirizzario /QIBM/UserData/OS400/SQLLib/Function, quindi 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.

E' possibile utilizzare iSeries Navigator per creare una procedura memorizzata o una funzione definita dall'utente che utilizza lo stile del parametro Java.

5. 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.



Procedure memorizzate Java

» Quando si utilizza Java^(TM) per scrivere procedure memorizzate, è possibile utilizzare due stili per passare un parametro. Lo stile consigliato è lo stile parametro JAVA, che corrisponde allo stile parametro specificato in SQLj: standard delle routine SQL. Il secondo stile, DB2GENERAL, è uno stile di parametro definito UDB di DB2^(R). Lo stile parametro determina inoltre le convenzioni che è necessario utilizzare quando si codifica una procedura memorizzata Java.

In aggiunta, sarebbe opportuno che l'utente fosse a conoscenza di alcune limitazioni collocate nelle procedure memorizzate Java.



Stile del parametro JAVA: » Quando si codifica una procedura memorizzata Java^(TM) 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.
- E' 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 memorizzate 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 l'interrogazione SQL specificata e restituisce sia il numero di righe nel risultato che SQLSTATE.

Esempio: procedura memorizzata con un'immissione e due emissioni

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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 della lista di parametri. Ad esempio, l'istruzione

```

CREATE PROCEDURE RETURN TWO()
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)`.

E' 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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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();
        String sql2 = "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 procedura memorizzata Java^(TM) che utilizza lo stile del 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.
- E' 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 ad 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. E' 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 l'interrogazione SQL specificata e restituisce sia il numero di righe nel risultato che `SQLSTATE`.

Esempio: procedura memorizzata con un'immissione e due emissioni

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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: ➤ Le seguenti limitazioni si applicano alle procedure memorizzate Java^(TM):

- Una procedura memorizzata Java non dovrebbe creare sottoprocessi aggiuntivi. E' 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 sempre l'ultima versione del Java Development Kit 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.
- 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 di servizio ha un nome generato dal sistema. E' possibile ottenere tale nome esaminando la registrazione lavori che ha creato la procedura memorizzata. Se l'oggetto del programma di servizio viene salvato e quindi ripristinato, la definizione della procedura viene ripristinata. Se è necessario spostare una procedura memorizzata Java da un sistema ad 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**^(TM) 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 - funzione definita dall'utente), è necessario conoscere le limitazioni inserite sulla creazione delle funzioni scalari Java.

Stile del parametro Java: Lo stile del 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.
- E' 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 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 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 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 passare un parametro per le procedure memorizzate e le UDF.

I valori NULL SQL sono rappresentati dalle variabili Java non inizializzate. Tali variabili hanno 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 del 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 del parametro DB2GENERAL: Lo stile del parametro DB2GENERAL è utilizzato dalle UDF Java. 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 del parametro DB2GENERAL, il metodo Java deve essere un metodo di istanza public void.
- I parametri del metodo Java devono essere tipi compatibili con SQL.
- E' possibile che il metodo Java verifichi un valore NULL SQL utilizzando il metodo isNull.
- Per lo stile del 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 Java, com.ibm.db2.app.UDF. Un'UDF Java che utilizza lo stile del parametro DB2GENERAL deve essere un metodo di istanza void della classe Java. Ad esempio, per un'UDF denominata sample!test3 che restituisce INTEGER ed 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 extends UDF {
    public void test3(String arg1, Blob arg2, String arg3, int result) { ... }
}
```


I parametri di un metodo Java devono essere tipi 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 passare un parametro per le procedure memorizzate e le UDF.

I valori NULL SQL sono rappresentati dalle variabili Java non inizializzate. Queste variabili hanno un valore di zero se sono tipi primitivi e un valore di 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 del 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 nella lista 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 un'interrogazione. 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 del 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. E' 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 un'interrogazione, se si specifica l'opzione `FINAL CALL` sull'istruzione `CREATE FUNCTION`, viene chiamato il metodo `public void close()` dell'oggetto (per funzioni dello stile del 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 del parametro `DB2GENERAL`. Tali variabili e metodi vengono spiegati nella seguente tabella.

Variabili e Metodi	Descrizione
<pre>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;</pre>	<p>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.</p>
<pre>public Connection getConnection();</pre>	<p>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 <code>SQLConnect()</code> null in una procedura memorizzata C.</p>
<pre>public void close();</pre>	<p>Questo metodo viene chiamato dal database alla fine di una valutazione UDF, se l'UDF è stata creata con l'opzione <code>FINAL CALL</code>. Esso è analogo alla chiamata finale per un'UDF C. Se una classe UDF Java non implementa questo metodo, questo evento viene ignorato.</p>
<pre>public boolean isNull(int i)</pre>	<p>Questo metodo verifica se un argomento di immissione con l'indice assegnato è un NULL SQL.</p>
<pre>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);</pre>	<p>Questi metodi impostano un argomento di emissione al valore assegnato. Viene emessa un'eccezione se si verifica qualcosa di sbagliato, incluse le seguenti situazioni:</p> <ul style="list-style-type: none"> • 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
<pre>public void setSQLstate(String string);</pre>	<p>E' possibile chiamare questo metodo da un'UDF per impostare l'<code>SQLSTATE</code> da restituire dalla chiamata. Se la stringa non è accettabile come <code>SQLSTATE</code>, viene emessa un'eccezione. L'utente può impostare <code>SQLSTATE</code> nel programma esterno per restituire un errore o un'avvertenza dalla funzione. In questo caso, è necessario che <code>SQLSTATE</code> contenga uno dei seguenti elementi:</p> <ul style="list-style-type: none"> • '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
<pre>public void setSQLmessage(String string);</pre>	<p>Questo metodo è simile al metodo <code>setSQLstate</code>. Esso imposta il risultato del messaggio SQL. Se la stringa non è accettabile (ad esempio, più lunga di 70 caratteri), viene emessa un'eccezione.</p>

Variabili e Metodi	Descrizione
<code>public String getFunctionName();</code>	Questo metodo restituisce il nome dell'UDF in elaborazione.
<code>public String getSpecificName();</code>	Questo metodo restituisce il nome specifico dell'UDF in elaborazione.
<code>public byte[] getDBInfo();</code>	Questo metodo restituisce una struttura DBINFO non elaborata per l'UDF in elaborazione, come una schiera di byte. E' necessario che l'UDF sia stata registrata (utilizzando CREATE FUNCTION) con l'opzione DBINFO.
<code>public String getDBname();</code> <code>public String getDBauthid();</code> <code>public String getDBver_rel();</code> <code>public String getDBplatform();</code> <code>public String getDBapplid();</code> <code>public String getDBapplid();</code> <code>public String getDBtbschema();</code> <code>public String getDBtbname();</code> <code>public String getDBcolname();</code>	Questi metodi restituiscono il valore del campo appropriato dalla struttura DBINFO dell'UDF in elaborazione. E' necessario che l'UDF sia stata registrata (utilizzando CREATE FUNCTION) con l'opzione DBINFO. I metodi <code>getDBtbschema()</code> , <code>getDBtbname()</code> e <code>getDBcolname()</code> restituiscono solo informazioni significative se una funzione definita dall'utente viene specificata nella parte destra di una clausola SET in un'istruzione UPDATE.
<code>public int getCCSID();</code>	Questo metodo restituisce il CCSID del lavoro.
<code>public byte[] getScratchpad();</code>	Questo metodo restituisce una copia dello scratchpad dell'UDF correntemente in elaborazione. E' necessario innanzitutto dichiarare l'UDF con l'opzione SCRATCHPAD.
<code>public void setScratchpad(byte ab[]);</code>	Questo metodo sostituisce lo scratchpad dell'UDF attualmente in elaborazione con il contenuto della schiera di byte fornita. E' necessario innanzitutto dichiarare l'UDF con l'opzione SCRATCHPAD. E' indispensabile che la schiera di byte abbia la stessa dimensione di quella restituita da <code>getScratchpad()</code> .
<code>public int getCallType();</code>	Questo metodo restituisce il tipo di chiamata che viene correntemente effettuata. Questi valori corrispondono ai valori C definiti in <code>sqludf.h</code> . Il seguente elenco include i possibili valori di ritorno: <ul style="list-style-type: none"> • <code>SQLUDF_FIRST_CALL</code> • <code>SQLUDF_NORMAL_CALL</code> •  <code>SQLUDF_TF_FIRST</code> • <code>SQLUDF_TF_OPEN</code> • <code>SQLUDF_TF_FETCH</code> • <code>SQLUDF_TF_CLOSE</code> • <code>SQLUDF_TF_FINAL</code> 



Limitazioni sulle funzioni Java definite dall'utente:  Le seguenti limitazioni si applicano alle UDF (user-defined function - funzioni definite dall'utente) Java^(TM):

- Un'UDF Java non dovrebbe creare sottoprocessi aggiuntivi. E' 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.

- 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 classe Java.
- Una procedura memorizzata 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.
- Un'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 fornisce ad una funzione la capacità 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 (JavaTM 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 del 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.

Punto nella scansione	NESSUNO SCRATCHPAD IN LINGUAGGIO JAVA DELLA CHIAMATA FINAL	SCRATCHPAD IN LINGUAGGIO JAVA DELLA CHIAMATA FINAL
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 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 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 che le UDF Java^(TM) possono utilizzare le classi Java memorizzate nei 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.

Per ulteriori informazioni, consultare i seguenti argomenti:

- SQLJ.INSTALL_JAR
- SQLJ.REMOVE_JAR
- SQLJ.REPLACE_JAR
- SQLJ.UPDATEJARINFO
- SQLJ.RECOVERJAR

SQLJ.INSTALL_JAR: » La procedura memorizzata SQLJ.INSTALL_JAR installa un file JAR in un sistema database. E' 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 UDB per iSeries supporta solamente un valore pari a zero.

Note sull'utilizzo: Quando si installa un file JAR, DB2 UDB per iSeries registra il file JAR nel catalogo di sistema SYSJAROBJECTS. Estrae anche i nomi dei file di classe Java^(TM) dal file JAR e registra ogni classe nel catalogo di sistema SYSJARCONTENTS. DB2 UDB per iSeries copia il file JAR su un sottoindirizzario jar/schema dell'indirizzario /QIBM/UserData/OS400/SQLLib/Function. DB2 UDB per iSeries fornisce alla nuova copia del file JAR il nome indicato nella clausola *jar-id*. Un file JAR che è stato installato da DB2 UDB per iSeries nel 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 ubicato nell'indirizzario file:/home/db2inst/classes/ è installato in DB2 UDB per iSeries 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 UDB per iSeries supporta solamente un valore pari a 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 a cui si è fatto riferimento da *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 di catalogo SYSJARCONTENTS. Questa procedura non è parte dello standard SQLJ ma viene utilizzata dal programma di creazione della procedura memorizzata di DB2 UDB per iSeries.

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

E' 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

E' 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. <<

Convenzioni per passare un parametro per le UDF e le procedure memorizzate

» 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 del parametro Java JAVA	Stile del parametro Java DB2GENERAL
SMALLINT	short	short
INTEGER	int	int
BIGINT	long	long
DECIMAL(p,s)	BigDecimal	BigDecimal
NUMERIC(p,s)	BigDecimal	BigDecimal
REAL o FLOAT(p)	float	float
DOUBLE PRECISION o FLOAT o FLOAT(p)	double	double
CHARACTER(n)	String	String
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.

JNI (Java Native Interface)

Uno dei modi in cui è possibile richiamare i codici scritti in un altro linguaggio è quello di implementare i metodi Java selezionati come 'metodi nativi.' I metodi nativi sono procedure, scritte in un altro linguaggio, che forniscono l'implementazione attuale di un metodo Java. E' possibile che i metodi nativi accedano alla Java virtual machine utilizzando la JNI (Java Native Interface). Questi metodi nativi sono in esecuzione nell'ambito del sottoprocesso Java, che è un sottoprocesso kernel, cosicché 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.

» Consultare Utilizzare JNI (Java Native Interface) per metodi nativi per ulteriori informazioni relative ai metodi nativi JNI e ILE. «

» Metodi nativi PASE OS/400

La JVM iSeries (Java virtual machine) supporta adesso l'utilizzo di metodi nativi in esecuzione su un ambiente PASE OS/400^(R). L'argomento Metodi nativi PASE OS/400 per Java consente all'utente di spostare facilmente le applicazioni Java che vengono eseguite in ambiente AIX^(R) sul server iSeries. E' possibile copiare i file di classe e le librerie dei metodi nativi AIX sull'IFS (integrated file system) di iSeries ed eseguirli da una qualsiasi richiesta comandi di sessione CL (control language), Qshell o sessione terminale PASE OS/400. «

`java.lang.Runtime.exec()`

E' possibile utilizzare `java.lang.Runtime.exec()` per richiamare i programmi o i comandi dall'ambito di un programma Java. Il metodo `exec()` avvia un altro processo nel quale è possibile eseguire qualsiasi programma o comando iSeries. 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.

E' possibile inoltre utilizzare i file di flusso per la comunicazione tra programmi. Altrimenti consultare gli esempi di comunicazione tra processi per una panoramica sulle opzioni durante la comunicazione con i programmi che sono in esecuzione in un altro processo.

» Per richiamare Java da altri linguaggi, consultare Esempio: richiamare Java da C o Esempio: richiamare Java da RPG per ulteriori informazioni. «

E' possibile inoltre utilizzare IBM Toolbox per Java per richiamare i programmi e i comandi esistenti sul server iSeries. Le code lavori e i messaggi iSeries vengono utilizzati di solito per la comunicazione tra processi con IBM Toolbox per Java.

Nota: utilizzando `Runtime.exec()`, IBM Toolbox per Java o JNI, è possibile compromettere la portabilità del programma Java. Bisogna evitare l'utilizzo di questi metodi in un ambiente Java "puro".

API di richiamo Java


Utilizzando API di richiamo Java, la quale è parte della specifica JNI (Java Native Interface), si consente un'applicazione non Java per l'utilizzo della Java virtual machine. Consente inoltre l'utilizzo del codice Java come un'estensione dell'applicazione.

Utilizzare JNI (Java Native Interface) per i metodi nativi

Sarebbe opportuno utilizzare i metodi nativi soltanto in casi dove non è possibile che Java^(TM) risponda 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 alle API (application program interface) esistenti che consentono a Java di chiamare altre API.

Per utilizzare JNI (Java Native Interface) per metodi nativi, eseguire queste fasi:

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 codifica 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 origine utilizzando il comando `CPYFRMSTMF` (Copia dal file di flusso). E' necessario copiare il file di intestazione in un membro del file origine affinché il compilatore C lo utilizzi. Utilizzare il nuovo supporto del file di flusso affinché il comando `CRTCMOD` (Create Bound ILE C/400 Program) lasci i file di intestazione C e di origine C nell'IFS. Per ulteriori informazioni relative al comando `CRTCMOD` e l'utilizzo dei file di flusso, consultare *WebSphere Development Studio: ILE C/C++ Programmer's Guide*, SC09-2712 .
6. Scrivere il codice del metodo nativo. Consultare Considerazioni sui sottoprocessi e sui metodi nativi per dettagli sui linguaggi e le funzioni utilizzate 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 è necessario passarle nella Java virtual machine. Per ulteriori informazioni, consultare Codifiche di carattere Java.
7. Se è necessario che il proprio metodo nativo interagisca con la Java virtual machine, utilizzare le funzioni fornite con JNI.
8. Compilare il proprio codice sorgente C, utilizzando `CRTCMOD`, in un oggetto modulo (*MOD).
9. Collegare uno o più oggetti moduli in un programma di servizio (*SRVPGM) utilizzando il comando `CRTSRVPGM` (Creazione programma di servizio). E' necessario che il nome di tale programma corrisponda al nome fornito nel proprio codice Java che si trova nelle chiamate alla funzione `System.load()` o `System.loadLibrary()`.
10. Se è stata utilizzata la chiamata `System.loadLibrary()` nel proprio codice Java, scegliere una delle seguenti opzioni.

Se si sta utilizzando una versione precedente a J2SDK:

Aggiungere la libreria `iSeries` che contiene il proprio nuovo programma di servizio alla lista librerie `iSeries`. Per aggiungere la libreria, utilizzare il comando `ADDLIBLE` (Aggiunta voce alla lista librerie). Ciò consente al programma Java di trovare il programma di servizio quando elabora la funzione `System.loadLibrary()`.

Se si sta utilizzando J2SDK, versione 1.2 o successive:

Non è necessario modificare la propria lista librerie. Invece è possibile scegliere di effettuare una delle seguenti operazioni:

- Includere la lista delle librerie necessarie nella variabile di ambiente `LIBPATH`. E' possibile modificare la variabile di ambiente `LIBPATH` in QShell e dalla riga comandi `iSeries`.
 - Dalla richiesta comandi Qshell, immettere:
`export LIBPATH=/QSYS.LIB/MYLIB.LIB`
`>> java -Djava.version=1.4 myclass<<`
 - O, dalla riga comandi:
`ADDENVVAR LIBPATH '/QSYS.LIB/MYLIB.LIB'`
`>> JAVA PROP((java.version 1.4)) myclass<<`
- Oppure fornire la lista nella proprietà `java.library.path`. E' possibile modificare la proprietà `java.library.path` in QShell e dalla riga comandi `iSeries`.

- Dalla richiesta comandi Qshell, immettere:


```
➤ java -Djava.library.path=/QSYS.LIB/MYLIB.LIB -Djava.version=1.4 myclass<<
```
- O, dalla riga comandi iSeries, immettere:



```
➤ JAVA PROP((java.library.path '/QSYS.LIB/MYLIB.LIB') (java.version '1.4')) 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 dei patch per `System.load(String patches)` può essere una delle seguenti:
- "path" (filename dell'IFS che specifica la libreria fornita dal programma di servizio), che è un collegamento simbolico a *SRVPGM, come `"/qsys.lib/mylib.lib/myNMsp.srvpgm"`
 - `/qsys.lib/sysNMsp.srvpgm`
 - `/qsys.lib/mylib.lib/myNMsp.srvpgm`
 - Se si dispone di una versione precedente a J2SDK: `/qsys.lib/%libl%.lib/myNMsp.srvpgm`
- Nota:** ciò è equivalente all'utilizzo del metodo `System.loadLibrary("myNMsp")`.

Nota: se il pathname viene utilizzato come costante letterale di stringhe, è necessario racchiuderlo tra virgolette. Ad esempio, `System.load("/qsys.lib/mylib.lib/myNMsp.srvpgm")`.

12. La sintassi "libya" per `System.loadLibrary(String libya)` è `misp`. Il sistema trova `misp` utilizzando *LIBL. Ad esempio, `loadLibrary("myNMsp")` è equivalente a `System.load("/qsys.lib/%libl%.lib/myNMsp.srvpgm")`. Se il "pathname" viene utilizzato come costante letterale di stringhe, è necessario racchiudere il libname tra virgolette.
- Nota:** la sintassi `%libl%` non è supportata per J2SDK.

Per una descrizione completa di JNI, far riferimento a Java Native Interface di Sun Microsystems, Inc. e The source for Java Technology java.sun.com .

Consultare Esempi: utilizzare JNI (Java Native Interface) per metodi nativi per un esempio della modalità di utilizzo della JNI per metodi nativi.

API di richiamo Java

L'API di richiamo, che fa parte della JNI (JavaTM 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 ad un programma sottoposto a più sottoprocessi di utilizzare le classi Java, in esecuzione in una singola JVM (Java virtual machine), in più sottoprocessi.

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. Una volta creata la JVM, 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 esegue la ripulitura, come eseguire i programmi di chiusura, arrestare i sottoprocessi della JVM e rilasciare le risorse della JVM.

Con una JVM pronta per l'esecuzione, è possibile che un'applicazione scritta in linguaggio C effettui una chiamata nella Java virtual machine per eseguire una qualsiasi funzione. E' inoltre possibile che essa torni dalla JVM all'applicazione C, ed essere nuovamente chiamata nella Java virtual machine e via di seguito. La JVM (Java virtual machine) viene creata una volta e non è necessario ricrearla prima di effettuare una chiamata nella Java virtual machine per eseguire una parte più o meno piccola 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 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 utilizzano i risultati STDIN in un'eccezione IO.

Per un esempio che utilizza l'API di richiamo, consultare Esempio: API di richiamo Java. Consultare Funzioni dell'API di richiamo per dettagli sulle funzioni dell'API di richiamo supportate da IBM Developer Kit per Java.

Funzioni dell'API di richiamo: IBM Developer Kit per Java^(TM) 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 sui lavori capaci di supportare più sottoprocessi.

- **JNI_GetDefaultJavaVMInitArgs**

Nota: questa funzione è supportata soltanto per JDK (Java Development Kit) 1.1.x.

Restituisce una struttura JDK 1.1 che contiene valori predefiniti per argomenti che è necessario passare in JNI_CreateJavaVM quando si crea una Java virtual machine.

Firma:

```
jint JNI_GetDefaultJavaVMInitArgs(void *args_);
```

- **JNI_GetCreatedJavaVMs**

Restituisce informazioni su tutte le Java virtual machine create.

Firma:

```
jint JNI_GetCreatedJavaVMs(JavaVM **vmBuf,  
                           jsize bufLen,  
                           jsize *nVMs);
```

vmBuf è un'area di emissione la cui dimensione viene determinata da bufLen, che è il numero dei puntatori. Ogni Java virtual machine dispone di una struttura JavaVM associata definita in java.h. Questa API memorizza un puntatore nella struttura JavaVM associata con ogni Java virtual machine creata in vmBuf, a meno che vmBuf non sia pieno. I puntatori alle strutture JavaVM sono memorizzati nell'ordine delle Java virtual machine corrispondenti create. nVM restituisce il numero di macchine virtuali correntemente create. Il proprio server iSeries supporta la creazione di più Java virtual machine, in modo che sia 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 Java virtual machine creata.

- **JNI_CreateJavaVM**

Consente di creare una Java virtual machine e successivamente di utilizzarla in un'applicazione.


Firma per Java Development Kit 1.1.x:

```
jint JNI_CreateJavaVM(JavaVM **p_vm,  
                     JNIEnv **p_env,  
                     void *vm_args);
```

Firma per J2SDK (Java 2 Software Development Kit):

```
jint JNI_CreateJavaVM(JavaVM **p_vm,  
                     void **p_env,  
                     void *vm_args);
```

p_vm è l'indirizzo di un puntatore JavaVM per la Java virtual machine appena creata. Molte altre API di richiamo JNI utilizzano p_vm per identificare la Java virtual machine. p_env è l'indirizzo di un puntatore

ambiente JNI per la 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 Java virtual machine. Quando si utilizza JDK 1.1.x, è possibile ottenere una struttura che contenga valori predefiniti chiamando `JNI_GetDefaultJavaVMInitArgs`. Per dettagli su come effettuare ciò con J2SDK, consultare [Java Native Interface](#) .

Se si avvia un comando `RUNJAVA` (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. Ad esempio, il parametro `os400.optimization` viene ignorato in questo comando:

```
JAVA CLASS(Hello) PROP((os400.optimization 0))
```

Per una lista di proprietà univoche OS/400 supportate dall'API `JNI_CreateJavaVM`, consultare [Proprietà di sistema Java](#).

Nota: con più Java virtual machine all'interno di un processo, tutte le Java virtual machine condividono la stessa memoria statica del processo assegnata per ogni metodo nativo. L'implementazione interna della Java virtual machine separa già i dati su una base per la Java virtual machine, ma è necessario considerare che, con applicazioni del metodo nativo, le Java virtual machine condividono la memoria statica del processo. Per ulteriori considerazioni, consultare [Supporto per più JVM \(Java virtual machine\)](#).

- **DestroyJavaVM**

Elimina la Java virtual machine.

Firma:

```
jint DestroyJavaVM(JavaVM *vm)
```

Quando viene creata la Java virtual machine, `vm` è il puntatore `JavaVM` che viene restituito.

- **AttachCurrentThread**

Collega un sottoprocesso ad una Java virtual machine, in modo che può utilizzare i relativi servizi.

Firma per JDK (Java Development Kit) 1.1.x:

```
jint AttachCurrentThread(JavaVM *vm,  
                          JNIEnv **p_env,  
                          void *thr_args);
```

Firma per J2SDK (Java 2 Software Development Kit):

```
jint AttachCurrentThread(JavaVM *vm,  
                          void **p_env,  
                          void *thr_args);
```


Il puntatore `JavaVM`, `vm`, identifica la Java virtual machine a cui il sottoprocesso viene collegato. `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 Java virtual machine da cui il sottoprocesso è stato scollegato.

Per una descrizione completa delle funzioni dell'API di richiamo, far riferimento alla [Java Native Interface Specification by Sun Microsystems, Inc. o The Source for Java Technology java.sun.com](#) .

Supporto per più JVM (Java virtual machine): Java^(TM) sul server iSeries, a differenza dell'implementazione del riferimento Sun Microsystems, Inc., supporta la creazione di più Java virtual machine nell'ambito di un singolo lavoro o di un singolo processo. Questo significa che è possibile chiamare con esito positivo JNI_CreateJavaVM() più di una volta in un lavoro e che JNI_GetCreatedJavaVMs() può restituire più di una Java virtual machine nella lista di risultati.

- ➤ Prima della versione V5R2, era possibile che la funzione JNI_GetCreatedJavaVMs restituisse più di una JVM nella lista JVM. In V5R2, la funzione JNI_GetCreatedJavaVMs restituisce, al massimo, una JVM
- Prima della versione V5R2, era possibile chiamare ripetutamente la funzione JNI_CreateJavaVM nell'ambito di un singolo processo e in seguito a ogni richiamo con esito positivo veniva creata una JVM distinta e separata. In V5R2, la funzione JNI_CreateJavaVM restituisce un codice di errore. (JNI_ERR -1 errore definito in jni.h) ⚡

Se si desidera creare più JVM (Java virtual machine) per l'utilizzo all'interno di un singolo lavoro o un singolo processo, bisogna considerare attentamente quanto segue:

Ambito di memorizzazione statica del metodo nativo

- I programmi di servizio che contengono implementazioni al metodo nativo vengono attivate solo una volta per lavoro, indipendentemente dal numero di Java virtual machine create. Questo implica che la memorizzazione del metodo static è inserita nell'ambito del lavoro e non nell'ambito di qualsiasi Java virtual machine specifica.
- I valori che un metodo nativo inserisce in una memorizzazione statica sono indipendenti dalla Java virtual machine che ha chiamato tale metodo. Questi valori sono visibili su qualsiasi Java virtual machine presente nel lavoro.
- Se viene intenzionalmente utilizzata la memorizzazione statica del metodo nativo in uno scenario con più Java virtual machine, bisogna considerare attentamente i requisiti necessari alla sincronizzazione, oltre all'utilizzo dei controlli e dei metodi sincronizzati, che risultano specifici per Java virtual machine. La qualifica di un metodo nativo come sincronizzato impedisce solo esecuzioni simultanee all'interno di una Java virtual machine singola ed esecuzioni non simultanee da più Java virtual machine.

Arresto di Java virtual machine

- Se una Java virtual machine viene arrestata in maniera anomala, a causa di un utente che chiama `java.lang.System.exit()` o di un errore di Java virtual machine interno, viene arrestata non solo la Java virtual machine in errore ma anche tutti i relativi sottoprocessi collegati.
- Se il sottoprocesso iniziale del processo si trova tra i sottoprocessi collegati alla Java virtual machine in errore, viene emessa un'eccezione al sottoprocesso iniziale. Se il sottoprocesso iniziale gestisce questa eccezione, è possibile che le altre Java virtual machine continuino l'esecuzione.
- Se il sottoprocesso iniziale del processo viene arrestato da un'eccezione non gestita o per qualsiasi altra ragione, tutte le Java virtual machine presenti nel processo vengono ugualmente arrestate.

Arresto anomalo da C

Se vengono utilizzate le routine `exit()` o `abort()` ILE/C in qualsiasi sottoprocesso di un lavoro con più sottoprocessi, viene immediatamente disattivato l'intero lavoro, incluse tutte le Java virtual machine.

Esempio: API di richiamo Java: Questo esempio segue il paradigma API di richiamo standard. Ad esempio, effettua quanto segue:

- Crea una JVM ovvero Java^(TM) virtual machine mediante l'utilizzo di 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.

Per compilare questo programma, è necessario collegarlo ad un programma di servizio che esporti le funzioni per avviare una nuova JVM (Java virtual machine). Seguono i punti di immissioni necessari:

- JNI_GetDefaultJavaVMInitArgs, che inizializza i parametri da creare.
- JNI_CreateJavaVM, che crea la JVM (Java virtual machine).

Quando si compila il programma, non è necessaria nessuna azione esplicita con il comando di compilazione. Il programma di servizio che esporta questi punti di immissione si trova nell'indirizzo di collegamento del sistema. Il nome del programma di servizio è QJVAJNI.

Per eseguire questo programma, utilizzare SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES). Qualsiasi lavoro che crea una Java virtuale machine deve essere capace di supportare più sottoprocessi. L'unico lavoro su un server iSeries che dispone di questa capacità è un lavoro BCI (batch immediato). L'emissione dal programma principale, così come ogni emissione dal programma, si conclude nei file di spool QPRINT. Tali file di spool sono visibili se si utilizza il comando WRKSBMJOB (Gestione lavori inoltrati) e si visualizza il lavoro iniziato con SBMJOB (Inoltro lavoro).

Nota: la routine `exit()` del tempo di esecuzione C utilizzata qui di seguito non è consigliata, a meno che l'utente non abbia la certezza che il proprio programma sia l'unico sottoprocesso nel processo. Quando viene chiamato da un processo capace di supportare più sottoprocessi, `exit()` termina immediatamente tutti i sottoprocessi in un processo.

Esempio: utilizzare l'API di richiamo Java con JDK 1.1.x.

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <jni.h>

int main (int argc, char *argv[])
{
    JDK1_1InitArgs initArgs; /* Struttura inizializzazione VM (Virtual Machine).
                             * Questa è la struttura passata dal riferimento a JNI_CreateJavaVM().
                             * Vedere jni.h per dettagli.
                             */

    JavaVM* myJVM;          /* i puntatori JavaVM e JNIEnv restituiti. */

    JNIEnv* myEnv;         /* ...dalla chiamata JNI_CreateJavaVM(). */

    char*   myClasspath;   /* Modificare il classpath, per disporre di uno proprio. */

    jclass myClass;        /* La classe che si ha intenzione di trovare, chiamata 'NativeHello'. */

    jmethodID mainID;      /* L'ID metodo della routine della classe "main". */

    jclass  stringClass;   /* Necessario per creare un argomento schiera di stringhe per 'main'. */

    jobjectArray args;     /* Poiché main prevede una schiera di stringhe, è necessario inoltrarne una. */

    /* Impostare il campo versione degli argomenti di inizializzazione. */
    initArgs.version = 0x00010001;

    /* Richiamare gli argomenti di inizializzazione predefiniti. */
    JNI_GetDefaultJavaVMInitArgs(&initArgs);

    /* Ora, si desidera aggiungere l'indirizzo alla fine del classpath,
     * in modo che findClass lo trovi correttamente. Per fare ciò, esistono due opzioni:
     * E' possibile accodare le voci classpath al classpath predefinito restituito
     * dalla chiamata a JNI_GetDefaultJavaVMInitArgs, oppure
     * utilizzare funzioni OS/400 specifiche per ottenere lo stesso risultato. Si
```

```

* tratta di una soluzione a tre fasi:
*   1. Impostare la variabile di ambiente CLASSPATH sui requisiti con 'putenv()'
*   2. Riportare gli argomenti di inizializzazione classpath su NULL, cioè
*      forza JNI_CreateJavaVM a consultare il valore CLASSPATH
*   3. Impostare la proprietà "os400.class.path.system=PRE", per forzare JNI_CreateJavaVM
*      a preaccodare il classpath di sistema predefinito al classpath reale.
*
* La prima opzione viene utilizzata in questo esempio, in quanto è più indipendente alla piattaforma
*
* Nota: Specificare il nome indirizzario nel formato UTF-8! Quindi, raggruppare
*       i blocchi di codice in istruzioni #pragma convert.
*/

#pragma convert(819)
myClasspath = malloc( strlen(initArgs.classpath) + strlen(":/CrtJvmExample") + 1 );
strcpy( myClasspath, initArgs.classpath );
strcat( myClasspath, ":/CrtJvmExample" );
initArgs.classpath = myClasspath;

#pragma convert(0)

/* Creare la JVM. */
if (JNI_CreateJavaVM(&myJVM, &myEnv, &initArgs)) {
    fprintf(stderr, "Failed to create the JVM\n");
    exit(1);
}

/* Usare la JVM appena creata per trovare la classe di esempio.
* Nota: Di nuovo, si sta lavorando con UTF-8, quindi
* raggruppare le chiamate in #pragma convert.
*/

#pragma convert(819)
if (! (myClass = (*myEnv)->FindClass(myEnv, "NativeHello"))) {

#pragma convert(0)
    /* Impossibile trovare la classe, quindi scrivere un messaggio di errore
    * in C stderr e uscire dal programma.
    */

    fprintf(stderr, "Failed to find the class 'NativeHello'\n");
    exit(1); /* L'uscita arresta l'intero processo sul server iSeries. */
}

/* Ora, richiamare l'identificativo del metodo per il punto di entrata 'main'
* della classe. Nota: la firma di 'main' è sempre uguale per
* ogni classe, "main" e "([Ljava/lang/String;)V"
* Di nuovo, si sta lavorando con UTF-8.
*/

#pragma convert(819)
if (! (mainID = (*myEnv)->GetStaticMethodID(myEnv, myClass,
                                           "main",
                                           "([Ljava/lang/String;)V"))) {
    /* L'ID metodo 'main' non è stato trovato per alcuni motivi. */
    if ( (*myEnv)->ExceptionOccurred(myEnv) ) {
        /* si è verificata un'eccezione java, quindi stamparla */
        (*myEnv)->ExceptionDescribe(myEnv);
        /* La JVM viene chiusa. */
        (*myEnv)->FatalError(myEnv, "Failed to find jmethodID of 'main()'");
    }
}

#pragma convert(0)
/* Impossibile trovare l'ID metodo 'main', quindi scrivere un messaggio di errore
* in C stderr e uscire dal programma.
*/

```

```

        fprintf(stderr, "Failed to find the 'main()' method\n");
        exit(1); /* L'uscita arresta l'intero processo sul server iSeries. */
    }

#pragma convert(819)
    if (! (stringClass = (*myEnv)->FindClass(myEnv,"java/lang/String"))) {

#pragma convert(0)
        /* Non è stato trovato java/lang/String, quindi scrivere un messaggio di errore
         * in C stderr e uscire dal programma.
         */

        fprintf(stderr, "Failed to find the java/lang/String");
        exit(1); /* l'uscita arresta l'intero processo sul server iSeries. */
    }

    /* Ora, è necessario creare una schiera di stringhe vuota,
     * poiché ([Ljava/lang/String) è una parte necessaria della firma di
     * ogni routine principale Java.
     */

    if (! (args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0))) {
        /* non è stata creata una schiera vuota, quindi scrivere un messaggio di errore
         * in C stderr e uscire dal programma.
         */

        fprintf(stderr, "Failed to create empty array of strings");
        exit(1); /* L'uscita arresta l'intero processo sul server iSeries. */
    }

    /* 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) ) {
        fprintf(stderr,"An exception occurred while running 'main'");
        exit(1);
    }

    /* Infine, eliminare la JavaVM creata. */

    if ( (*myJVM)->DestroyJavaVM(myJVM) ) {
        fprintf(stderr, "Failed to destroy the JVM\n");
        exit(1);
    }

    /* Eseguite tutte le operazioni. */

    return 0;
}

```



Esempio: utilizzare l'API di richiamo Java con J2SDK.

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

#define OS400_JVM_12
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

```

/* Specificare il pragma 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(). Consultare jni.h per dettagli
                             */
    JVM* myJVM;             /* Puntatore JVM 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 J2SDK. */
    initArgs.version = 0x00010002;

    /* 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";

    initArgs.options=options; /* Inoltrare il classpath impostato. */
    initArgs.nOptions = 1;    /* Inoltrare dell'unica opzione classpath */

    /* 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 passare 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.

Considerazioni sui sottoprocessi e i metodi nativi di Java

E' possibile utilizzare i metodi nativi per accedere alle funzioni non disponibili in Java^(TM).

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 Java virtual machine e la Java virtual machine viene arrestata in maniera anomala, viene segnalata l'eccezione iSeries MCH74A5 "Java Virtual Machine Arrestata" al sottoprocesso iniziale del processo se quel sottoprocesso è stato collegato alla Java virtual machine quando la Java virtual machine ha terminato l'esecuzione. E' possibile che la Java virtual machine 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 virtual machine 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 server iSeries, è possibile rendere l'applicazione meno portabile rispetto ad 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 durante il sottoprocesso direttamente da Java.
- E' 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 della fase di compilazione relativo alla registrazione dei metodi nativi è attualmente fornito solo per i linguaggi C e C++. Anche se possibile, la registrazione 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()` suddetta è la funzione C e C++ e non è uguale al metodo `java.lang.Runtime.exit()`.

Per ulteriori informazioni sui sottoprocessi del server iSeries, consultare Applicazioni a più sottoprocessi.


Metodi nativi e JNI (Java native interface)

I metodi nativi sono quei metodi Java^(TM) che si avviano in un linguaggio diverso da Java. E' 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 portabilità 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 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 Java virtual machine in numerosi 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, fare riferimento a Java Native Interface di Sun Microsystems, Inc., o The Source for Java Technology java.sun.com .

Stringhe nei metodi nativi

Molte funzioni JNI (Java^(TM) 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 iSeries 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. E' 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: E' 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. Per ulteriori informazioni sulle stringhe dinamiche, consultare *stringhe dinamiche*.

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.

Convertire le 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 EBCDIC (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^(TM) 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.

Esempi: utilizzare JNI (Java Native Interface) per i metodi nativi

Questo programma è un semplice esempio di JNI (Java^(TM) 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 nella lista librerie affinché venga eseguito questo esempio.

Esempio 1: classe `NativeHello`

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.


```

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 iSeries 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: consultare 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 */

#ifdef _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 il server iSeries 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 siano sempre stati presenti con JNI, causano alcune differenze univoche specifiche del server iSeries nel codice C che si scrive. E' 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 registrare 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 NativeHello

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

#include <stdlib.h>      /* malloc, free e così via */
#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, reperita 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 reperita 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)) {

    /* E' 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); /* eliminare l'eccezione in sospeso */
}

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)

```

Consultare [Utilizzare JNI \(Java Native Interface\) per metodi nativi per informazioni di background.](#) ➤

Metodi nativi di PASE OS/400 IBM per Java

La JVM (JavaTM virtual machine) iSeries ora supporta l'uso dei metodi nativi in esecuzione nell'ambiente PASE OS/400^(R). Prima della V5R2, la JVM nativa di iSeries utilizzava solo i metodi nativi ILE. Il supporto per i metodi nativi di PASE OS/400 include:

- L'uso completo della JNI (Java Native Interface) nativa di iSeries dai metodi nativi di PASE OS/400
- La capacità di richiamare i metodi nativi di PASE OS/400 dalla JVM nativa di iSeries

Questo nuovo supporto consente di indirizzare facilmente le applicazioni Java, che vengono eseguite in AIX^(R), al server iSeries. E' possibile copiare i file di classi e le librerie dei metodi nativi di AIX nell'IFS (integrated file system) sull'iSeries ed eseguirli da qualsiasi richiesta comandi di una sessione del terminale PASE OS/400, Qshell o CL (control language).

Per ulteriori informazioni sull'utilizzo dei metodi nativi di PASE OS/400 IBM per Java, consultare i seguenti argomenti:

Variabili di ambiente PASE OS/400 Java

Fornisce informazioni sulle variabili di ambiente che occorre definire prima di utilizzare i metodi nativi di PASE OS/400. Queste variabili di ambiente gestiscono gli ambienti runtime JVM e PASE OS/400.

Codici di errore PASE OS/400 Java

Fornisce una guida su come risolvere i problemi relativi ai metodi nativi di PASE OS/400, su come individuare le condizioni di errore descritte nei messaggi della registrazione lavoro OS/400 e gli errori Java runtime.

Gestire le librerie di metodi nativi

Fornisce informazioni sulle convenzioni di denominazione della libreria Java e sull'algoritmo di ricerca della libreria. Queste informazioni sono fondamentali nella gestione di più versioni di una libreria metodi nativi sul server iSeries.

Esempio: metodi nativi di PASE OS/400 IBM per Java

Fornisce informazioni su come eseguire un programma Java semplice che stampi il contenuto di una stringa 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.

Queste informazioni presumono che l'utente abbia già una certa familiarità con PASE OS/400. Per ulteriori informazioni, consultare il seguente argomento:

PASE OS/400



Variabili di ambiente PASE OS/400 Java

La JVM (Java virtual machine) utilizza le variabili che seguono per avviare gli ambienti PASE OS/400. E' necessario impostare la variabile QIBM_JAVA_PASE_STARTUP per poter eseguire l'esempio per il metodo nativo PASE OS/400 IBM per Java.

Per informazioni sull'impostazione delle variabili di ambiente per l'esempio, consultare il seguente argomento:

Esempio di variabili di ambiente per PASE OS/400 IBM.

QIBM_JAVA_PASE_STARTUP

E' necessario impostare questa variabile di ambiente quando si verificano entrambe le condizioni che seguono:

- Si stanno utilizzando i metodi nativi di PASE OS/400
- Si sta avviando Java da una richiesta comandi iSeries o Qshell

JVM utilizza questa variabile di ambiente per avviare un ambiente PASE. Il valore della variabile identifica un programma di avvio di PASE OS/400. Il server iSeries contiene due programmi di avvio di PASE OS/400:

- /usr/lib/start32: avvia un ambiente PASE OS/400 a 32 bit
- /usr/lib/start64: avvia un ambiente PASE OS/400 a 64 bit

Il formato di bit di tutti gli oggetti della libreria condivisa utilizzati da un ambiente PASE OS/400 deve corrispondere al formato di bit dell'ambiente PASE OS/400.

Non è possibile utilizzare questa variabile se si avvia Java da una sessione del terminale PASE OS/400. Una sessione del terminale PASE OS/400 utilizza sempre un ambiente PASE OS/400 a 32 bit. Qualsiasi JVM avviata da una sessione del terminale PASE OS/400 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 PASE OS/400 per una JVM secondaria sia diverso dall'ambiente PASE OS/400 della JVM principale. Una chiamata di Runtime.exec() in Java avvia una JVM secondaria (o child).

Per ulteriori informazioni, consultare Utilizzare QIBM_JAVA_PASE_CHILD_STARTUP.



Esempi: esempio di variabili di ambiente per PASE OS/400 IBM: Per utilizzare l'esempio dei metodi nativi di PASE OS/400 IBM per Java, è necessario impostare le seguenti variabili di ambiente.

PASE_LIBPATH

Il server iSeries utilizza questa variabile di ambiente PASE OS/400 per identificare l'ubicazione delle librerie dei metodi nativi di PASE OS/400. E' 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 questo esempio, consultare il seguente argomento:

Utilizzare Java, PASE OS/400 e le librerie di metodi nativi

PASE_THREAD_ATTACH

Se si imposta questa variabile di ambiente PASE OS/400 su Y, un sottoprocesso ILE, non avviato da PASE OS/400, verrà automaticamente collegato a PASE OS/400, quando richiama una procedura PASE OS/400.

Per ulteriori informazioni sulle variabili di ambiente PASE OS/400, consultare le voci appropriate nel seguente argomento:

Gestire le variabili di ambiente PASE OS/400

QIBM_JAVA_PASE_STARTUP

JVM utilizza questa variabile di ambiente per avviare un ambiente PASE OS/400. Il valore della variabile identifica un programma di avvio di PASE OS/400.

Per ulteriori informazioni, consultare il seguente argomento:

Variabili PASE OS/400 Java



Utilizzare QIBM_JAVA_PASE_CHILD_STARTUP: La variabile di ambiente QIBM_JAVA_PASE_CHILD_STARTUP indica il programma di avvio PASE OS/400 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 di Runtime.exec()
- Entrambe le JVM, principale e secondaria, utilizzano i metodi nativi di PASE OS/400
- L'ambiente PASE OS/400 delle JVM secondarie deve essere differente dall'ambiente PASE OS/400 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 OS/400 PASE delle JVM secondarie
- Quando si avvia la JVM principale da una richiesta comandi iSeries o Qshell, impostare la variabile di ambiente QIBM_JAVA_PASE_STARTUP sul programma di avvio PASE OS/400 della JVM principale.

Nota: quando si avvia la JVM principale da una sessione del terminale PASE OS/400, non impostare QIBM_JAVA_PASE_STARTUP.

Il processo della JVM secondaria eredita la variabile di ambiente QIBM_JAVA_PASE_CHILD_STARTUP. Inoltre, OS/400 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 del processo principale (parent).

La tabella che segue identifica gli ambienti PASE OS/400 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:

Ambiente di avvio			Funzionalità risultante	
Ambiente comandi	QIBM_JAVA_PASE_STARTUP	QIBM_JAVA_PASE_CHILD_STARTUP	Avvio PASE OS/400 della JVM principale	Avvio PASE OS/400 della JVM secondaria
CL o QSH	StartX definito	StartY definito	Utilizzare startX	Utilizzare startY
CL o QSH	StartX definito	Non definito	Utilizzare startX	Utilizzare startX
CL o QSH	Non definito	StartY definito	Nessun ambiente PASE OS/400	Utilizzare startY
CL o QSH	Non definito	Non definito	Nessun ambiente PASE OS/400	Nessun ambiente PASE OS/400
Sessione terminale PASE OS/400	StartX definito	StartY definito	Non consentito*	Non consentito*
Sessione terminale PASE OS/400	StartX definito	Non definito	Non consentito*	Non consentito*
Sessione terminale PASE OS/400	Non definito	StartY definito	Utilizzare ambiente sessione terminale PASE OS/400	Utilizzare startY
Sessione terminale PASE OS/400	Non definito	Non definito	Utilizzare ambiente sessione terminale PASE OS/400	Nessun ambiente PASE OS/400

* 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 PASE OS/400. A causa di tale potenziale conflitto, non è consentito l'utilizzo di QIBM_JAVA_PASE_STARTUP da una sessione del terminale PASE OS/400.



Gestire le 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.

OS/400 utilizza il primo nome libreria metodi nativi corrispondente a quello della libreria caricata dalla JVM (Java virtual machine). Per assicurarsi che OS/400 rilevi i metodi nativi corretti, è necessario evitare conflitti tra i nomi libreria e confusioni sulla libreria metodi nativi utilizzata da JVM.

Convenzioni di denominazione della libreria Java AIX e PASE OS/400: 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 PASE OS/400 per la JVM, il server utilizza tre diverse liste (nell'ordine che segue) per creare un singolo percorso di ricerca della libreria metodi nativi.

1. Lista librerie OS/400
2. Variabile di ambiente LIBPATH
3. Variabile di ambiente PASE_LIBPATH

Per effettuare la ricerca, OS/400 converte la lista 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, OS/400 utilizza il formato IFS per ricercare le librerie di metodi nativi.

La tabella che segue illustra il modo in cui OS/400 converte le voci della lista librerie nel formato IFS:

Voce lista 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: ricercare la 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)
Lista 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
1. /qsys.lib/qsys2.lib/sample2.srvpgm
2. /qsys.lib/qsys2.lib/libSample2.a
3. /qsys.lib/qsys2.lib/libSample2.so
1. /qsys.lib/qgpl.lib/sample2.srvpgm
2. /qsys.lib/qgpl.lib/libSample2.a
3. /qsys.lib/qgpl.lib/libSample2.so

1. /qsys.lib/qtemp.lib/sample2.srvpgm
2. /qsys.lib/qtemp.lib/libSample2.a
3. /qsys.lib/qtemp.lib/libSample2.so
1. /home/user1/lib32/sample2.srvpgm
2. /home/user1/lib32/libSample2.a
3. /home/user1/lib32/libSample2.so
1. /samples/lib32/sample2.srvpgm
2. /samples/lib32/libSample2.a
3. /samples/lib32/libSample2.so
1. /QOpenSys/samples/lib/SAMPLE2.srvpgm
2. /QOpenSys/samples/lib/libSample2.a
3. /QOpenSys/samples/lib/libSample2.so

OS/400 carica il primo candidato nella lista 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 OS/400 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 OS/400 nel file system QSYS. Ne risulta che i file candidati validi includono file del tipo
/home/user1/lib32/sample2.srvpgm. << >>

Codici di errore PASE OS/400 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 PASE OS/400 per Java.

Errori all'avvio: Vi sono tre nuovi codici di errore per il messaggio JVAB55C "Impossibile creare JVM (Java virtual machine)":

- 19 - Errore durante l'avvio dell'ambiente PASE OS/400. Indica un problema dell'applicazione dell'utente o del sistema operativo.

Il codice di errore 19 contiene solo testo in Inglese. Viene visualizzato il seguente testo di errore:

- Java OS/400 PASE error. OS/400 PASE is already active and the QIBM_JAVA_PASE_STARTUP environment variable is defined.

Eliminare la definizione della variabile di ambiente QIBM_JAVA_PASE_STARTUP oppure chiudere la sessione del terminale PASE OS/400 attiva.

- Java OS/400 PASE error. Unable to run OS/400 PASE startup program &programName.

Il programma PASE OS/400 identificato dalla variabile di ambiente QIBM_JAVA_PASE_STARTUP non esiste oppure non è possibile eseguirlo nell'ambiente PASE OS/400.

- Java OS/400 PASE internal error number &errorCode.

E' possibile visualizzare uno dei seguenti numeri di errore interno:

- 106 - PASE OS/400 o il formato PASE OS/400 specificato non è supportato dalla versione JDK indicata.

PASE OS/400 non è supportato dal JDK specificato oppure il formato di bit del programma di avvio di PASE OS/400 non è supportato dal JDK specificato. Per la V5R2, le combinazioni supportate sono:

- JDK 1.2 e formato a 32 bit di PASE OS/400
- JDK 1.3 e formato a 32 bit di PASE OS/400

- JDK 1.3 e formato a 64 bit di PASE OS/400

Riportare i seguenti codici di errore ad un tecnico della manutenzione:

- 101 - Nessun programma di avvio identificato.
- 102 - Impossibile richiamare il puntatore JavaVM PASE OS/400
- 103 - Impossibile individuare Qp2CallPase
- 104 - Errore dimensione puntatore PASE OS/400.
- 105 - Impossibile individuare libjvm.a di PASE OS/400.
- 20 - Operando di PASE OS/400 non valido. Riferire al tecnico della manutenzione.
- 21 - Impossibile collegare lavoro a PASE OS/400. Riferire al tecnico della manutenzione.

Errori al tempo di esecuzione: Oltre agli errori all'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, consultare Qp2CallPase.

- PASEExit - l'applicazione PASE OS/400 ha chiamato la funzione exit() oppure l'ambiente PASE OS/400 è stato chiuso in modo anomalo. Per ulteriori informazioni controllare la Registrazione lavoro e la Registrazione LIC (Licensed Internal Code). <<



Esempio: metodo nativo PASE OS/400 IBM per Java

L'esempio del metodo nativo PASE OS/400 IBM per Java chiama un'istanza di un metodo C nativo che, in seguito, utilizza la JNI (Java Native Interface) per richiamarlo nel codice Java.

Per visualizzare le versioni HTML dei file sorgente dell'esempio, utilizzare i collegamenti che seguono:

- PASEExample1.java
- PASEExample1.c

Prima di poter eseguire l'esempio del metodo nativo PASE OS/400, è necessario completare le attività che seguono:

1. Scaricare il codice sorgente dell'esempio sulla stazione di lavoro AIX
2. Preparare il codice sorgente dell'esempio
3. Preparare il server iSeries

Eseguire l'esempio del metodo nativo PASE OS/400 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 del server iSeries:

```
JAVA CLASS(PASEExample1) CLASSPATH('/home/example')
```

- Da una richiesta comandi Qshell o da una sessione del terminale PASE OS/400:

```
cd /home/example
java PASEExample1
```



Confronto tra ILE (Integrated Language Environment) e Java

L'ambiente Java^(TM) su un server iSeries è separato dall'ILE (integrated language environment). Java non è un linguaggio ILE e non può collegarsi a moduli oggetti ILE per creare programmi o programmi di servizio su un server iSeries.

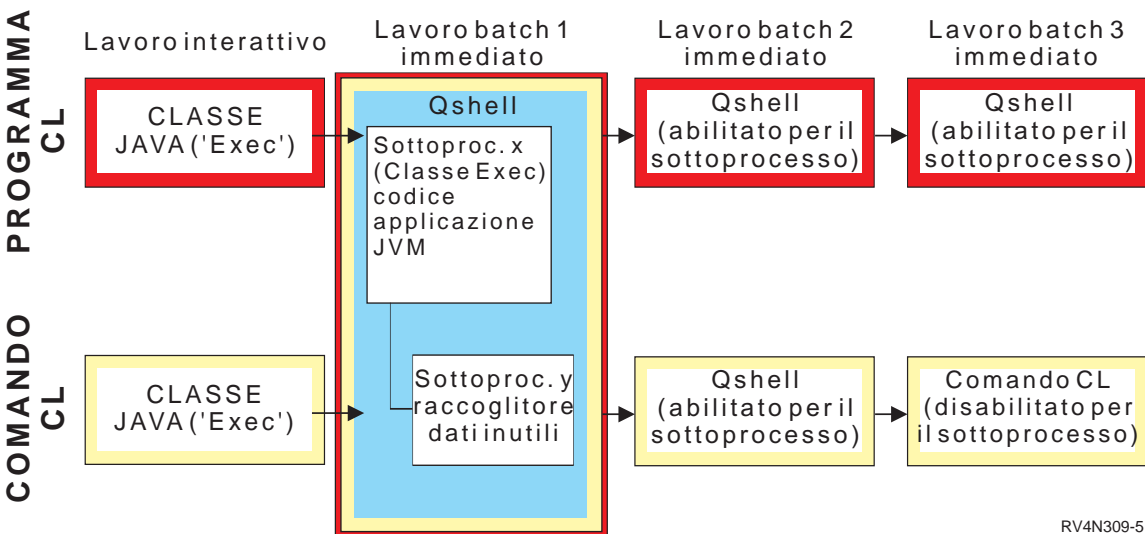
ILE	Java
I membri che fanno parte della struttura file o libreria su un server iSeries memorizzano dei codici di origine.	I file del flusso nell'IFS (Integrated File System) contengono un codice sorgente.
SEU (source entry utility) modifica file origine EBCDIC (extended binary-coded decimal interchange code).	I file origine ASCII (American Standard Code for Information Interchange) vengono solitamente modificati utilizzando un editor della stazione di lavoro.
I file origine si compilano nei moduli del codice oggetto, memorizzati nelle librerie su un server iSeries.	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.
E' possibile chiamare direttamente le funzioni scritte in altri linguaggi di programmazione ILE.	E' necessario utilizzare JNI per chiamare altri linguaggi da Java.
I linguaggi ILE vengono sempre compilati ed eseguiti come istruzioni macchina.	E' possibile interpretare e compilare i programmi Java.

Utilizzare java.lang.Runtime.exec()

Il metodo `java.lang.Runtime.exec()` chiama programmi o comandi dall'interno di un programma JavaTM. L'effettiva elaborazione che si verifica dipende esattamente dal tipo di informazioni passate al metodo `exec()`. In tutti i casi, il metodo `Runtime.exec()` crea un altro lavoro BCI (batch immediato) abilitato per il sottoprocesso. Il lavoro BCI elabora la stringa del comando che viene inserita nel metodo `Runtime.exec()`.

Per utilizzare il metodo `java.lang.Runtime.exec()`, è necessario installare Qshell Interpreter sul proprio server iSeries se si sta utilizzando JDK 1.2 o precedenti. Per ulteriori informazioni su Qshell Interpreter, consultare Qshell Interpreter.

Nota: il metodo `java.lang.Runtime.exec()` esegue programmi in un processo separato. Ciò differisce dalla funzione del sistema C, che esegue un programma nello stesso processo.



Se il comando elaborato è un programma di utilità Qshell, esso è in esecuzione nel secondo lavoro BCI e il terzo lavoro BCI non viene creato. Se il comando elaborato è un comando CL, il secondo lavoro BCI viene avviato per eseguire Qshell e il terzo lavoro BCI viene avviato per eseguire il comando CL. Un programma di utilità Qshell è un programma incorporato che QSH può eseguire direttamente. Un esempio di programma di utilità Qshell è il comando `javac`, che compila i programmi Java. L'elaborazione nel

secondo (o nel terzo) lavoro BCI è in esecuzione contemporaneamente alla Java virtual machine. Qualsiasi elaborazione di uscita o di chiusura in questi lavori non influenza la Java virtual machine originale.

Quando si chiama un comando o programma iSeries, è necessario assicurarsi che qualsiasi parametro passato al programma chiamato sia nella codepage prevista da quel programma.

Consultare chiamare un altro programma Java, chiamare un programma CL o chiamare un comando CL per esempi `java.lang.Runtime.exec()`.

Esempio: Richiamare un altro programma Java con `java.lang.Runtime.exec()`

Questo esempio descrive come chiamare un altro programma Java^(TM) con `java.lang.Runtime.exec()`. Questa classe chiama il programma Hello fornito come parte di IBM Developer Kit per Java. Quando la classe Hello registra 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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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();
        }

    } // fine metodo

} // fine classe
```

Per informazioni di background, consultare Utilizzare `java.lang.Runtime.exec()`.

Esempio: chiamare un programma CL con `java.lang.Runtime.exec()`

Questo esempio mostra come eseguire i programmi CL dall'interno di un programma Java[™]. Consultare chiamare un comando CL per un esempio di come chiamare un comando CL dall'interno di un programma Java. In questo esempio, la classe `Java 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`.

Nota: `JAVSAMPLIB` non viene creato come parte del processo di installazione del programma su licenza (LP - licensed program) di IBM Developer Kit numero 5722-JV1. E' necessario creare la libreria esplicitamente.

Esempio 1: classe `CallCLPgm`

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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();
        }
    } // fine metodo main()
} // fine classe
```

Esempio 2: visualizzare il programma CL Java

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
          OUTPUT(*PRINT)
ENDPGM
```

Per informazioni di background, consultare Utilizzare `java.lang.Runtime.exec()`.

Esempio: chiamare 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 `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`.

» Se si utilizza JDK 1.1.8 o JDK 1.2, « ogni comando inoltrato nella funzione `Runtime.getRuntime().exec()` deve essere racchiuso tra virgolette ed espresso nel formato Qshell. In aggiunta, per eseguire un comando CL da Qshell, è necessario passare la stringa:

```
"system \"CL COMMAND\""
```

Dove `CL COMMAND` è il comando CL che si intende eseguire. Quindi la riga per chiamare il comando `MYCLCOM` dovrebbe essere la seguente:

```
Runtime.getRuntime().Exec("system \"MYCLCOM\"");
```

» **Nota:** Quando si utilizza JDK 1.3 o JDK 1.4, omettere i delimitatori barra e virgolette (\"). Ad esempio, quando si utilizza la versione 1.3 di JDK o successive, la chiamata al comando MYCLCOM viene espressa nel seguente modo:

```
Runtime.getRuntime().Exec("system MYCLCOM");
```

Per maggiori informazioni, fare riferimento alla proprietà di sistema os400.runtime.exec nell'argomento Proprietà di sistema Java per Java 2 Software Development Kit, Edizione Standard. <<

Esempio 1: classe CallCLCom

» Il seguente esempio utilizza i delimitatori Qshell necessari con l'utilizzo di JDK 1.1.8 o JDK 1.2. Se si utilizza JDK versione 1.3 o successive, omettere i delimitatori. <<

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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();
        }
    } // fine metodo main()
} // fine classe
```

Per informazioni di background, consultare Utilizzare java.lang.Runtime.exec().

Comunicazioni tra processi

Quando si comunica con programmi in esecuzione su un altro processo, esistono varie opzioni.

Un'opzione è utilizzare i socket per la comunicazione 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 è utilizzare i file di flusso per la comunicazione tra programmi. Per fare ciò, utilizzare le classi System.in, System.out e System.err.

Un terza opzione è quella per utilizzare IBM Toolbox per Java^(TM) la quale fornisce le code di dati e gli oggetti dei messaggi iSeries.

» E' inoltre possibile chiamare Java da altri linguaggi. Consultare Esempio: chiamare Java da C e Esempio: chiamare Java da RPG per ulteriori informazioni. <<

Utilizzare i socket per la comunicazione tra processi

I flussi di socket comunicano tra i programmi in esecuzione su processi separati. E' possibile che i programmi vengano avviati separatamente o utilizzando il metodo java.lang.Runtime.exec() dall'interno del programma principale Java^(TM). 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 Codifiche di caratteri Java per ulteriori dettagli.

Per un esempio che utilizza i socket, consultare Esempio: utilizzare i socket per la comunicazione tra processi.

Esempio: utilizzare i socket per la comunicazione tra processi: Questo esempio utilizza i socket per comunicare tra un programma Java^(TM) e un programma C. E' necessario avviare prima il programma C, che registra 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(yyyyy nnnn)` sulla riga comandi iSeries per avviare il programma Java. `yyyyy` 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. E' necessario utilizzare inoltre questo numero porta come primo parametro sulla chiamata al programma C.

Esempio 1: classe client TalkToC

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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);
}

```

Per ulteriori informazioni, consultare Utilizzare i socket per la comunicazione tra processi.

Utilizzare i 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^(TM), è necessario assicurare che avvenga una conversione ASCII (American Standard Code for Information Interchange) o EBCDIC (extended binary-coded decimal interchange code). Consultare Codifiche di caratteri Java per ulteriori dettagli.

Per un esempio che utilizza i flussi di immissione ed emissione, esaminare Esempio: utilizzare i flussi di immissione ed emissione per la comunicazione tra processi.

Esempio: utilizzare i flussi di immissione ed emissione per la comunicazione tra processi: Questo esempio mostra come chiamare un programma C da Java^(TM) 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) di IBM Developer Kit numero 5722-JV1. E' necessario crearlo esplicitamente.

Esempio 1: classe CallPgm

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
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();
        }
    }
}
```

```

    } // fine metodo
} // fine classe

```

Esempio 2: programma C CSAMP1

Nota: consultare 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);
}

```

Per ulteriori informazioni, consultare Utilizzare i flussi di immissione ed emissione per la comunicazione tra processi.

Esempio: Chiamare Java da C

Questo è un esempio di un programma C che utilizza la funzione `system()` per chiamare il programma Hello Java.

Esempio: Chiamare Java da C

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

#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: chiamare Java da RPG

Questo è un esempio di un programma RPG che utilizza l'API QCMDEXC per chiamare il programma Hello Java^(TM).

Esempio 1: chiamare Java da RPG

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

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
C
LR

```

Piattaforma Java

» La **piattaforma JavaTM** 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 classi. 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.

Per utilizzare in modo efficace la piattaforma Java, consultare quanto segue:

Applicazioni e applet Java

E' possibile scrivere l'applet Java e includerla in una pagina HTML, nello stesso modo in cui viene inclusa un'immagine. Quando viene utilizzato 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. E' possibile inoltre scrivere un'applicazione Java che non richiede l'utilizzo di un browser web.

JVM (Java virtual machine)

E' possibile incorporare la JVM (Java virtual machine) all'interno del browser Web oppure nel sistema operativo come ad esempio IBM^(R) Operating System/400^(R) (OS/400^(R)). La Java virtual machine consiste dell'interprete Java e di JRE (Java runtime environment). L'interprete esegue l'attività di interpretazione del file di classe e delle istruzioni Java in una piattaforma specifica. La Java virtual machine è ciò che consente la scrittura e la compilazione del codice Java e l'esecuzione su qualsiasi piattaforma.

File di classe e JAR Java


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. Al contrario, il compilatore Java converte il codice sorgente Java in istruzioni Java virtual machine, che i file di classe Java memorizzano. E' 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 Java virtual machine.

Sottoprocessi Java

Java è un linguaggio di programmazione con più sottoprocessi; in questo modo è possibile che più di un sottoprocesso sia in esecuzione contemporaneamente all'interno della Java virtual machine. I sottoprocessi Java danno la possibilità a un programma Java di eseguire più attività contemporaneamente.


Java Development Kit

Il JDK (Java Development Kit) è un software distribuito da Sun Microsystems, Inc. per gli sviluppatori Java. Questo include l'interprete Java, le classi Java e gli strumenti di sviluppo Java. Reperire le seguenti informazioni sui JDK:

- Pacchetti Java
- Strumenti Java 

Applicazioni e applet Java

Un' **applet** è un programma Java^(TM) progettato per essere incluso in un documento Web HTML. 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.

Per ulteriori informazioni, consultare [Scrivere applet](#) , 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'interprete Java dalla riga comandi e specificando il file che contiene l'applicazione compilata. Le applicazioni solitamente risiedono sul sistema su cui vengono sviluppate. Esse accedono alle risorse sul sistema e sono limitate dal modello di sicurezza Java.

JVM (Java virtual machine)

La Java^(TM) virtual machine è un ambiente del tempo di esecuzione che è possibile aggiungere in un browser web o in qualsiasi sistema operativo, come IBM Operating System/400 (OS/400). La Java virtual machine esegue istruzioni generate da un compilatore Java. Essa consiste in un interprete 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 inoltre limitare le risorse di sistema a cui ogni classe caricata accede.

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.

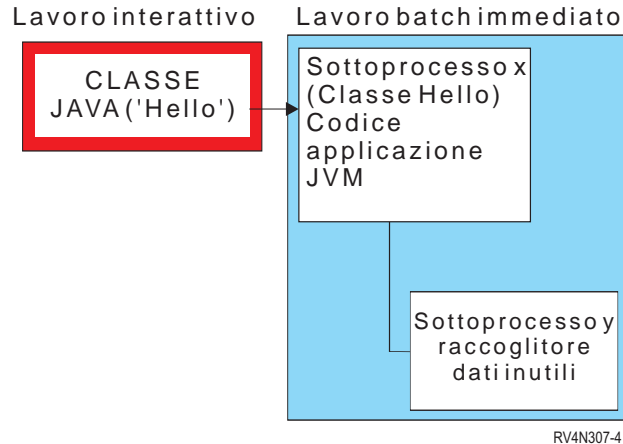
E' possibile utilizzare il comando CRTJVAPGM (Creazione programma Java) per assicurare che il codice rispetti i requisiti di sicurezza che il tempo di esecuzione Java impone per verificare i bytecode. Ciò include il controllo di limitazioni del tipo, il controllo di conversioni di dati, l'assicurarsi 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'interprete li decodifica ed esegue le istruzioni macchina necessarie per effettuare le operazioni desiderate.

Nota: l'interprete Java viene utilizzato soltanto per specificare OPTIMIZE(*INTERPRET) o INTERPRET(*YES).

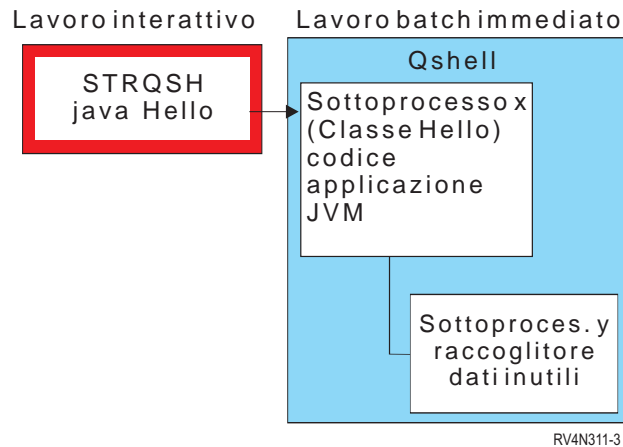
In aggiunta al caricamento e all'esecuzione dei bytecode, la Java virtual machine include un raccoglitore dati inutili che gestisce la memoria. La raccolta di dati inutili viene eseguita nello stesso momento del caricamento e dell'interpretazione dei bytecode.

JRE (Java runtime environment)

JRE viene avviato ogni qualvolta si immette il comando RUNJVA (Esecuzione Java) o il comando JAVA sulla riga comandi iSeries. Dal momento che l'ambiente Java dispone di più sottoprocessi, è necessario eseguire la Java virtual machine su un lavoro che supporti i sottoprocessi, come il lavoro BCI (batch immediato). Una volta avviata la Java virtual machine, è possibile avviare ulteriori sottoprocessi su cui viene eseguito il raccoglitore di dati inutili. L'ambiente Java tipico risulta in questo modo:



E' inoltre possibile avviare JRE utilizzando il comando java Qshell da Qshell Interpreter. In questo ambiente, Qshell Interpreter è in esecuzione su un lavoro BCI associato ad un lavoro interattivo. JRE viene avviato sul lavoro su cui è in esecuzione Qshell Interpreter.



Quando JRE 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'interprete Java fa parte della Java virtual machine che interpreta i file di classe Java per una specifica piattaforma hardware. L'interprete Java decodifica ogni bytecode ed esegue una serie di istruzioni macchina per quel bytecode.

File di classe e JAR Java

Un file **JAR (Java™ ARchive)** è un formato file che combina più file in uno. E' 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, esaminare lo Strumento jar Java. «

I **file di classe Java** sono file di flusso prodotti quando un file 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.

Sottoprocessi Java

Un **sottoprocesso** è un flusso singolo indipendente che viene eseguito all'interno di un programma. Java^(TM) 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.

E' possibile utilizzare i sottoprocessi in un programma Java per numerose situazioni. E' 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.

E' 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. E' 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*.

JDK (Java Development Kit) di Sun Microsystems, Inc.


Il JDK (JavaTM) Development Kit) è un software distribuito da Sun Microsystems, Inc. per gli sviluppatori Java. Questo include l'interprete 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 ottenere ulteriori informazioni sul JDK corrente, controllare la versione di IBM Developer Kit per Java sul server iSeries.

E' possibile controllare la versione della Java virtual machine predefinita di IBM Developer Kit per Java sul server iSeries immettendo uno dei seguenti comandi:

- `java -version` sulla richiesta comandi Qshell.
- `RUNJAVA CLASS(*VERSION)` sulla riga comandi CL.

Successivamente cercare la stessa versione JDK di Sun Microsystems, Inc. sul sito web 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, per cui è necessario avere familiarità con la relativa documentazione JDK.

Consultare i seguenti argomenti per ulteriori informazioni:

- Supporto per più JDK (Java Development Kit) fornisce informazioni sull'utilizzo di diverse Java virtual machine.
- Metodi nativi e JNI (Java Native Interface) definisce di che metodo nativo si tratta e quali sono le relative funzioni. Questo argomento spiega inoltre in breve il funzionamento della Java Native Interface.

Pacchetti Java

Un pacchetto Java rappresenta 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).

Pacchetto	Contenuto
java.applet	Classi applet
java.awt	Grafici, finestre e classi GUI (graphical user interface)
java.awt.datatransfer	Classi di conversione dati
java.awt.event	Classi e interfacce di elaborazione eventi
java.awt.image	Classi di elaborazioni immagini

Pacchetto	Contenuto
java.awt.peer	Interfacce GUI per l'indipendenza della piattaforma
java.beans	API di modello del componente JavaBeans
java.io	Classi di immissione ed emissione
java.lang	Classi di lingua principale
java.lang.reflect	Classi dell'API di riflessione
java.math	Aritmetica arbitraria di precisione
java.net	Classi di rete
java.rmi	Classi Remote Method Invocation
java.rmi.dgc	Classi correlate a RMI
java.rmi.registry	Classi correlate a RMI
java.rmi.server	Classi correlate a RMI
java.security	Classi di sicurezza
java.security.acl	Classi correlate alla sicurezza
java.security.interfaces	Classi correlate alla sicurezza
java.sql	API SQL JDBC per le classi del database
java.text	Classi di internazionalizzazione
java.util	Tipi di dati
java.util.zip	Classi di compressione e decompressione

Per ulteriori informazioni sulle API Java di Sun Microsystems, Inc., consultare il manuale Sun Microsystems, Inc. API User's Guide.

Strumenti Java

Per una lista completa di strumenti forniti da JDK di Sun Microsystems, Inc., consultare Tools Reference by Sun Microsystems, Inc. Per ulteriori informazioni su ogni strumento singolo che IBM Developer Kit per Java supporta, consultare gli Strumenti Java supportati da IBM Developer Kit per Java.

Argomenti avanzati

➤ Seguono gli argomenti avanzati per IBM Developer Kit per Java^(TM):

Classi, pacchetti e indirizzari

Ogni classe Java fa parte di un pacchetto. Il nome del pacchetto è correlato alla struttura dell'indirizzario in cui si trova la classe.

I file nell'IFS (Integrated File System)

L'IFS (Integrated File System) memorizza la classe correlata a Java, l'origine, i file ZIP e i file JAR in una struttura file gerarchica.

Autorizzazioni del file

Per eseguire o sottoporre a debug un programma Java, la propria classe, i file ZIP o i file JAR richiedono un'autorizzazione di lettura. Ricercare maggiori informazioni sulle autorizzazione del file che molti comandi CL richiedono.

Lavoro batch

E' possibile eseguire i programmi Java in lavoro batch utilizzando il comando SBMJOB (Inoltre lavoro). Ricercare maggiori informazioni sul comando SBMJOB e sul modo in cui è possibile verificare che il proprio lavoro batch sia in grado di eseguire più di un lavoro.



Classi, pacchetti e indirizzari Java

Ogni classe Java^(TM) fa parte di un pacchetto. La prima istruzione in un file origine Java indica quale classe si trova in un pacchetto. Se il file origine non contiene un'istruzione del pacchetto, la classe fa parte di un pacchetto predefinito non denominato.

Il nome del pacchetto è correlato 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 sistemi UNIX e PC. E' necessario memorizzare una classe Java in un indirizzario con un percorso indirizzario relativo che corrisponda al nome pacchetto per tale classe. Ad esempio, considerare la seguente classe Java:

```
package classes.geometry;
import java.awt.Dimension;

public class Shape {

    Dimension metrics;

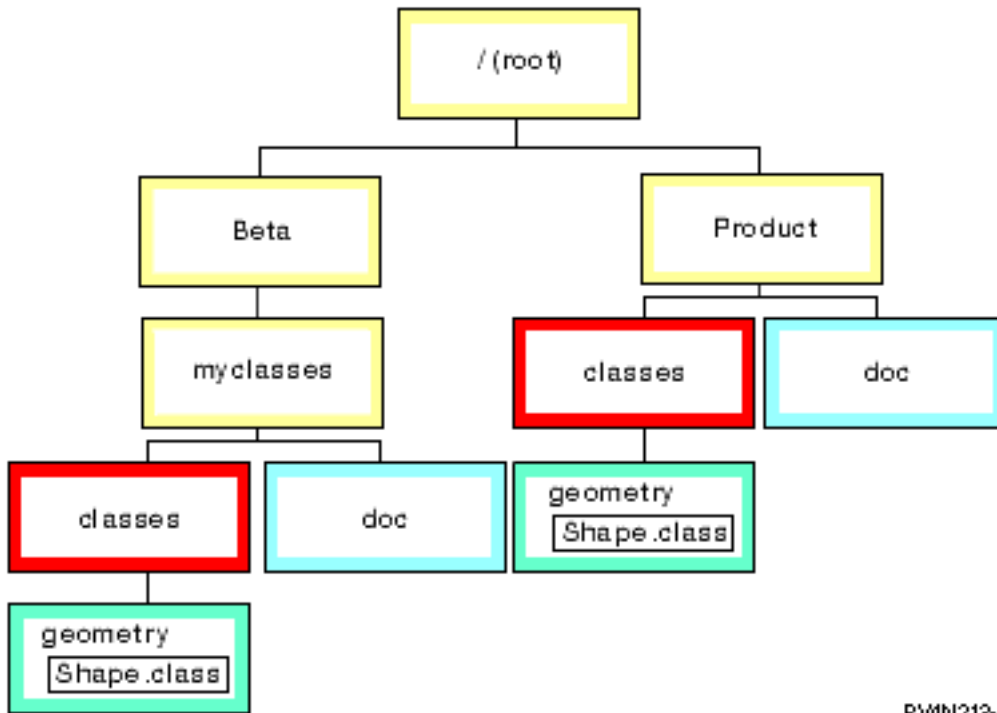
    // L'implementazione per la classe Shape verrà codificata qui ...

}
```

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 indirizzario relativo in cui è memorizzata la classe. Il programma di caricamento classi JVM trova la classe accodando il nome percorso relativo ad ogni indirizzario specificato nel classpath. Tale programma di caricamento può inoltre trovare la classe ricercando i file ZIP o JAR che sono specificati nel classpath.

Ad esempio, se la classe Shape è stata memorizzata nell'indirizzario /Product/classes/geometry nel file system "root" (/), sarebbe necessario specificare /Product nel classpath.



RV4N312-1

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 ogni 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.

File nell'IFS (integrated file system)

L'IFS (Integrated File System) memorizza i file della classe correlata a Java, i file origine, i file ZIP e i file JAR in una struttura file gerarchica. E' inoltre possibile memorizzare i file origine nell'IFS. E' possibile memorizzare i file in questi IFS:

- File system "Root" (/)
- File system apertura di sistemi (QOpenSys)
- File system definito dall'utente
- File system della libreria (QSYS.LIB)
- File system OS/2 Warp Server per iSeries (QLANSrv)
- File system unità ottica (QOPT)

Nota: altri IFS non sono supportati in quanto non sono sicuri durante i sottoprocessi.

Autorizzazioni file Java nell'IFS (integrated file system)

Per eseguire o effettuare il debug di un programma JavaTM, è necessario che il file di classe, il file JAR o il file ZIP dispongano dell'autorizzazione alla lettura (*R). E' necessario che qualsiasi indirizzario disponga 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. E' 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 l'IFS abbia indicato 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.

➤ E' inoltre possibile assegnare autorizzazioni ai nuovi file creati dai programmi Java in un IFS. Utilizzando la proprietà di sistema `os400.file.create.auth` per i file e `os400.dir.create.auth` per gli indirizzari, è possibile utilizzare qualsiasi combinazione di autorizzazioni alla lettura, scrittura e all'esecuzione. ◀

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

Eseguire Java in un lavoro batch

I programmi JavaTM 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 per gestire i flussi `System.in`, `System.out` né `System.err`.

E' 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. E' possibile reindirizzare `System.in`, `System.out` e `System.err` all'interno del programma Java. E' 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.

E' 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.

Se il campo Max attivi è inferiore o uguale a 1 e non è *NOMAX, immettere quanto segue sulla riga comandi CL:

```
CHGJOBQE SBSD(MYSBSD) JOBQ(MYJOBQ) MAXACT(*NOMAX)
```

Dove:

- MYSBSD rappresenta la descrizione del sottosistema e
- MYJOBQ rappresenta la coda lavori.

Eseguire la propria applicazione Java su un host che non dispone di una GUI (graphical user interface)

Se si desidera eseguire la propria applicazione Java^(TM) su un host che non dispone di una GUI (graphical user interface), come un server iSeries, è possibile utilizzare l'AWT (Abstract Window Toolkit) Remoto, il CBJ (Class Broker per Java) o il NAWT (Native Abstract Windowing Toolkit).

Si utilizza l'AWT Remoto con interfacce di installazione e gestione delle applicazioni del server. Queste interfacce normalmente hanno un minimo di grafica complessa e un contenuto altamente interattivo. L'AWT Remoto distribuisce un'elaborazione AWT tra il server iSeries e una stazione di lavoro. Quindi, la capacità di risposta delle operazioni altamente interattive e a grafica complessa non è veloce come le implementazioni AWT sulle piattaforme con terminali grafici collegati localmente. Per utilizzare l'AWT Remoto, consultare Impostare AWT Remoto.

E' possibile utilizzare il CBJ per servizi GUI ad alte prestazioni. Dal momento che l'AWT Remoto non è consigliato per operazioni altamente interattive e a grafica complessa, è possibile utilizzare invece il CBJ, progettato per questi ambienti. Per utilizzare il CBJ, consultare Impostare CBJ.

» E' possibile utilizzare NAWT per un'elaborazione grafico Java interamente su un server iSeries utilizzando X Window System. **X Window System** è un sistema di grafica che fornisce una base al server/client per la visualizzazione della grafica. Il server della grafica X Window è altamente portabile consentendo il supporto per una varietà di linguaggi e sistemi operativi. NAWT fornisce servlet e applicazioni Java con la capacità di utilizzare la funzionalità della grafica dell'AWT di JDK (Java Development Kit). «

AWT (Abstract Window Toolkit) Remoto di IBM Developer Kit per Java

AWT (Abstract Window Toolkit) Remoto è un'implementazione di AWT (Abstract Window Toolkit). Esso consente l'esecuzione delle applicazioni Java^(TM) senza alcuna modifica, su un host che non dispone di una GUI (Graphical user interface). Un server iSeries non supporta terminali grafici collegati localmente; quindi AWT Remoto è necessario per consentire l'esecuzione delle applicazioni grafiche Java sul server iSeries.

Si utilizza AWT Remoto con interfacce di installazione e gestione delle applicazioni server. Queste interfacce normalmente hanno un minimo di grafica complessa e un contenuto altamente interattivo. AWT Remoto distribuisce l'elaborazione AWT tra un server iSeries e una stazione di lavoro. Non è possibile che la capacità di risposta delle operazioni altamente interattive e a grafica complessa siano veloci come le implementazioni AWT sulla piattaforma con terminali grafici collegati localmente.

Dato che l'AWT Remoto di IBM Developer Kit per Java non è consigliato per operazioni altamente interattive e a grafica complessa, è possibile utilizzare invece Class Broker per Java, progettato per questi ambienti.

Per informazioni su come impostare AWT Remoto, consultare Impostare AWT (Abstract Window Toolkit) Remoto per Java su un pannello remoto.

Impostare AWT (Abstract Window Toolkit) Remoto per Java su un pannello remoto

Con AWT (Abstract Window Toolkit) Remoto, è possibile eseguire i programmi di grafica AWT JavaTM senza apportare modifiche al codice sorgente del programma e visualizzare la grafica in modo remoto. Per utilizzare AWT Remoto è necessaria l'impostazione di TCP/IP (Transmission Control Protocol/Internet Protocol) e l'installazione di Sun Microsystems, Inc., JDK (Java Development Kit) 1.1.8 o J2SDK (Java 2 SDK), Edizione Standard, sul server iSeries e sul pannello remoto.

E' possibile utilizzare qualsiasi hardware in grado di supportare la grafica, incluso IBM Network Station, come un pannello remoto per l'AWT Remoto se questi requisiti vengono soddisfatti:

- L'hardware in grado di supportare la grafica su cui è in esecuzione Windows^(R) 95, Windows NT 4.0, IBM Operating System/2^(R) (OS/2^(R)), Sun Solaris o AIX^(R)
- L'hardware configurato per l'accesso al server iSeries con TCP/IP
- Java Development Kit 1.1.8 o J2SDK

Per impostare l'AWT Remoto, completare queste attività:

1. Rendere i file di classe dell'AWT Remoto accessibili al pannello remoto copiando i file sul pannello remoto o definendo il percorso fino a un'unità di rete sul pannello remoto.
2. Aggiungere RAWTGui.zip o RAWTGui.jar al CLASSPATH del pannello remoto. Per JDK 1.1.8, aggiungere il file RAWTGui.zip al CLASSPATH del pannello remoto impostando la variabile di ambiente CLASSPATH o utilizzando il parametro `-classpath` del comando `java`. Per J2SDK, il file RAWTGui.jar viene automaticamente aggiunto al CLASSPATH quando si utilizza il parametro `-jar` del comando `java`.
3. Avviare AWT Remoto sul pannello remoto.

Per dettagli e suggerimenti per l'utilizzo dell'AWT Remoto, consultare i seguenti argomenti:

- Eseguire un programma Java utilizzando AWT (Abstract Window Toolkit) Remoto fornisce istruzioni sul modo in cui eseguire un programma Java con un server iSeries utilizzando più JDK e Netscape.
- Stampare con AWT (Abstract Window Toolkit) Remoto spiega come stampare, operazione che in sostanza è simile alla stampa standard dell'AWT Java. Inoltre mostra come stampare sul server iSeries.
- Proprietà dell'AWT (Abstract Window Toolkit) Remoto mostra come eseguire un'applicazione AWT Remoto utilizzando la proprietà `os400.class.path.rawt`.
- Limitazioni SecurityManager dell'AWT (Abstract Window Toolkit) Remoto fornisce informazioni sulle limitazioni che si applicano durante l'esecuzione delle applicazioni Java utilizzando l'AWT Remoto sotto il controllo di un SecurityManager.

Per ulteriori informazioni sull'impostazione di TCP/IP, consultare "How do I set up TCP/IP" nel manuale di

TCP/IP Configuration and Reference, SC41-5420. 

Consultare Esempio: impostare l'AWT (Abstract Window Toolkit) Remoto per Java su un pannello remoto di Windows, per un esempio sul modo in cui impostare l'AWT Remoto.

Per ulteriori informazioni sull'AWT, consultare Abstract Window Toolkit di Sun Microsystems, Inc.

Rendere i file di classe di AWT Remoto (Abstract Window Toolkit) per Java accessibili al pannello remoto

Per rendere i file di classe di AWT Remoto (Abstract Window Toolkit) accessibili al pannello remoto, seguire queste fasi in riferimento a JDK (JavaTM Development Kit) 1.1. x o in riferimento a Java 2 SDK (J2SDK), Edizione Standard, versione 1.2. E' importante tenere presente, tuttavia, che affinché AWT Remoto funzioni correttamente, la versione del file RAWTGui.jar utilizzata sul pannello remoto abilitato alla grafica deve corrispondere alla versione JDK o J2SDK utilizzata sull'host.

Se si sta utilizzando JDK 1.1.8, è possibile inoltre:

- Copiare i file di classe dell'AWT Remoto sul pannello remoto.

I file AWT Remoto vengono installati con IBM Developer Kit per Java in due file ZIP:

- /QIBM/ProdData/Java400/jdk118/RAWTApplHost.zip
- /QIBM/ProdData/Java400/jdk118/RAWTGui.zip

Il file RAWTApplHost.zip contiene le classi AWT Remoto relative al server iSeries. Il file RAWTGui.zip contiene le classi AWT Remoto relative al pannello remoto.

Copiare RAWTGui.zip da /QIBM/ProdData/Java400/jdk118 nel pannello remoto.

- Definire il percorso, /QIBM/ProdData/Java400/jdk118/RAWTGui.zip, in un'unità di rete sul pannello remoto.

Se si sta utilizzando J2SDK, versione 1.2 o una versione successiva, è possibile inoltre:

- Copiare i file di classe dell'AWT Remoto sul pannello remoto.

I file AWT Remoto vengono installati con IBM Developer Kit per Java in due file JAR:

- /QIBM/ProdData/Java400/jdk12/RAWTAHost.jar
- /QIBM/ProdData/Java400/jdk12/RAWTGui.jar

Se si sta utilizzando una versione di J2SDK diversa da 1.2, sostituire il numero di quella versione in tutte le istanze di percorso in questa sezione.

Il file RAWTAHost.jar contiene le classi AWT Remoto relative al server iSeries. Il file RAWTGui.jar contiene le classi AWT Remoto relative al pannello remoto.

Copiare RAWTGui.jar da /QIBM/ProdData/Java400/jdk12 in un'unità di rete su un pannello remoto.

- Definire il percorso, /QIBM/ProdData/Java400/jdk12/RAWTGui.jar, in un'unità di rete su un pannello remoto.

Aggiungere RAWTGui.zip o RAWTGui.jar al CLASSPATH del pannello remoto

L'impostazione di CLASSPATH consente a Java^(TM) virtual machine di trovare le classi dell'AWT (Abstract Window Toolkit) Remoto sul pannello remoto. Questa fase è necessaria solo per JDK 1.1.x. Non è necessaria con J2SDK (Java 2 SDK). Per aggiungere il file RAWTGui.zip al CLASSPATH del pannello remoto, eseguire una di queste operazioni:

- Impostare la variabile di ambiente CLASSPATH. Consultare le informazioni JDK (Java Development Kit) per il proprio pannello remoto per dettagli.

Aggiungere il percorso dove si trova il file RAWTGui.zip alla variabile di ambiente CLASSPATH.

- Utilizzare il parametro -classpath del comando java.

Quando si utilizza il comando java per avviare AWT Remoto, è possibile utilizzare il parametro -classpath per specificare un CLASSPATH. Il CLASSPATH include il percorso dove è ubicato il file RAWTGui.zip.

Ad esempio, in Windows^(R), è possibile che il parametro CLASSPATH venga visualizzato nel seguente modo:

```
-classpath c:\jdk1.1.7\lib\classes.zip;c:\rawt\RAWTGui.zip
```

J2SDK, supporto JAR versione 1.2 e successive imposta il CLASSPATH, in modo che non sia necessario impostare esplicitamente il parametro CLASSPATH. Per impostare il classpath e avviare l'AWT Remoto su un pannello remoto, immettere questo comando:

```
java -jar <PATH>RAWTGui.jar
```

dove <PATH> rappresenta l'unità e l'indirizzario completi nel quale il file RAWTGui.jar è ubicato. Ad esempio, java -jar c:\rawt2\RAWTGui.jar.

Avviare AWT (Abstract Window Toolkit) Remoto per Java su un pannello remoto

E' necessario avviare il daemon del server sul pannello remoto una volta e questo rimane attivo fino a quando non verrà chiuso. I programmi Java^(TM) che escono in un server iSeries non chiudono il daemon del server.

Nota: la finestra di dialogo di benvenuto rimane attiva quando si avvia il daemon del server. Quando il pannello della finestra di dialogo di benvenuto viene chiuso, il daemon del server termina. E' possibile minimizzare tale pannello mentre il daemon del server è attivo e utilizzarlo per terminare il daemon del server.

Per avviare il daemon del server dell'AWT (Abstract Window Toolkit) Remoto per JDK 1.1.x, immettere quanto segue sulla riga comandi:

```
java -classpath <PATH>RAWTGui.zip;C:\jdk1.1.8\lib\classes.zip
com.ibm.rawt.server.RAWTServer
```


dove <PATH> rappresenta l'unità e l'indirizzario completi nel quale il file RAWTGui.jar è ubicato. Ad esempio, java -jar c:\rawt2\RAWTGui.jar.

Per avviare il daemon del server di AWT Remoto per J2SDK, versione 1.3, immettere quanto segue sulla riga comandi:

```
java -jar <PATH>RAWTGui.jar
```

dove <PATH> rappresenta l'unità e l'indirizzario completi nel quale il file RAWTGui.jar è ubicato. Ad esempio, java -jar c:\rawt2\RAWTGui.jar.

Il daemon del server seleziona la prima porta libera superiore a 2000 quando l'applicazione Java si collega utilizzando l'AWT Remoto. L'applicazione Java utilizza questa porta finché non termina l'applicazione. Applicazioni Java aggiuntive sono collegate alle porte libere superiori a 2000. L'intervallo disponibile di porte arriva fino a 9999.

Per ulteriori informazioni sull'impostazione di TCP/IP, consultare "How do I set up TCP/IP" nel manuale TCP/IP Configuration and Reference, SC41-5420. 

Eeguire un programma Java utilizzando AWT (Abstract Window Toolkit) remoto

Per eseguire Java^(TM) utilizzando AWT (Abstract Window Toolkit) Remoto, attenersi alle seguenti fasi:

1. Avviare AWT Remoto sul pannello remoto.
2. Avviare il programma Java sul server iSeries.
 - a. Immettere il comando RUNJVA (Esecuzione Java) sulla riga comandi.
Nota: è necessario definire il classpath Java sul programma Java.
 - b. Premere F4 (Richiesta).
 - c. Immettere il nome classe del programma Java sulla riga parametro classe.
 - d. Premere F10 (Parametri aggiuntivi).
 - e. Premere il tasto Pag. giù.
 - f. Immettere RmtAwtServer sulla riga successiva parametro nome proprietà.
 - g. Immettere l'indirizzo TCP/IP (Transmission Control Protocol/Internet Protocol) (ad esempio, 1.1.11.11) del pannello remoto sulla riga successiva parametro valore proprietà.
 - h. Immettere os400.class.path.rawt sulla riga parametro nome proprietà.
 - i. Immettere 1 sulla riga parametro valore proprietà.
 - j. Immettere + per ulteriori proprietà.
 - k. Immettere java.version sulla riga parametro nome proprietà.
 - l. Immettere 1.3 sulla riga parametro valore proprietà. E' necessario che questa versione corrisponda alla versione del codice daemon RAWTGui.jar in esecuzione sul pannello remoto.

La riga comandi dovrebbe corrispondere al modello seguente, tutto su una riga:

```
java class (classname) prop(('RmtAwtServer' '1.1.11.11')
('os400.class.path.rawt' '1')('java.version' '1.3'))
```

m. Premere Invio.

E' possibile eseguire inoltre un programma Java utilizzando AWT Remoto con Netscape.

Eseguire un programma Java utilizzando AWT (Abstract Window Toolkit) remoto con Netscape:

Quando si eseguono le applicazioni JavaTM con Netscape, è possibile eseguirle in due modi.

Un'opzione è quella di avviare il server AWT (Abstract Window Toolkit) Remoto all'interno della Java virtual machine di Netscape aprendo un file HTML che contiene com.ibm.rawt.server.StartRAWT.class. Ad esempio, vedere il file RAWT.html di seguito specificato. Una volta avviato, è possibile avviare un'applicazione Java sul server iSeries.

Altrimenti, è possibile avviare il server AWT Remoto all'interno della Java virtual machine di Netscape aprendo un file HTML che contiene com.ibm.rawt.server.StartRAWT400.class e le classi IBM Toolbox per Java. Ad esempio, consultare il file RAWT400.html riportato di seguito. Una volta avviato, è possibile collegarsi al server iSeries sul quale si trova l'applicazione Java e avviare l'applicazione.

Eseguire il server AWT Remoto all'interno della Java virtual machine di Netscape

Per eseguire il server AWT Remoto all'interno della Java virtual machine di Netscape, effettuare queste fasi:

1. Modificare questo esempio di file .html per le informazioni specifiche sull'installazione relative al file RAWTGui.zip. Questo file, RAWT.html, avvia AWT Remoto all'interno della Java virtual machine di Netscape.

Esempio: avviare AWT Remoto all'interno della Java virtual machine di Netscape

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
<HTML>
<BODY TEXT="#000000" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<APPLET CODE="com.ibm.rawt.server.StartRAWT.class"
codebase="file://C:\remote_aws\jdk1.1.7\lib\RAWTGui.zip"
WIDTH=600 HEIGHT=50>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

2. Sfogliare la pagina RAWT.html con Netscape 4.05 o con una versione successiva. Dopo la concessione di tutti i privilegi richiesti, Netscape avvia il server AWT Remoto e lo esegue all'interno della Java virtual machine.
3. Avviare un'applicazione nel server iSeries utilizzando AWT Remoto.
4. Dopo la chiusura dell'applicazione, tenere premuto il pulsante Ricarica e spingere il pulsante Maiusc. per avviare nuovamente il server AWT Remoto.

Eseguire il server AWT Remoto all'interno della Java virtual machine di Netscape e collegarsi al server iSeries

Per eseguire il server AWT Remoto all'interno della Java virtual machine di Netscape e per collegarsi al server iSeries, effettuare queste fasi:

1. Modificare questo file .html di esempio per le informazioni specifiche sull'installazione relative a jt400.zip and RAWTGui.zip. Questo file, RAWT400.html, avvia l'AWT Remoto e utilizza l'IBM Toolbox per Java per collegarsi al server iSeries.

Esempio: avviare l'AWT Remoto e utilizzare il Toolbox per Java per collegarsi ad un server iSeries

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
<HTML>
<BODY TEXT="#000000" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<APPLET ARCHIVE="file://C:\jt400\lib\jt400.zip"
code="com.ibm.rawt.server.StartRAWT400.class"
codebase="file://C:\remote_awt\jdk1.1.1\lib\RAWTGui.zip"
WIDTH=600 HEIGHT=50>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

2. Sfogliare questa pagina RAWT400.html con Netscape 4.05. Dopo la concessione di tutti i privilegi richiesti, Netscape avvia l'applet di AWT Remoto che visualizza un pannello nel quale è possibile eseguire una qualsiasi di queste opzioni:
 - Collegarsi al server iSeries con AWT Remoto utilizzando IBM Toolbox per Java per accedere al server iSeries.
 - Immettere il nome e gli argomenti dell'applicazione Java con le proprietà di AWT Remoto.
 - Premere il pulsante Avvia applicazione per avviare l'applicazione Java specificata con l'AWT Remoto.

Stampare con AWT (Abstract Window Toolkit) Remoto

La stampa con AWT (Abstract Window Toolkit) Remoto è uguale alla stampa AWT Java^(TM) standard. Il pannello remoto di AWT Remoto elabora l'emissione di stampa e la indirizza su qualsiasi stampante nota al sistema operativo del pannello remoto. E' possibile che si tratti di una stampante direttamente collegata al pannello remoto o di una stampante di rete nota al sistema operativo del pannello remoto.

E' possibile selezionare la stampa sul pannello remoto o su un server iSeries. Una nuova finestra di dialogo di stampa viene visualizzata quando un'applicazione invia una richiesta di stampa. La richiesta di stampa consente di selezionare una stampante a visualizzazione remota o una stampante OS/400. Se si seleziona una stampante OS/400, compare una finestra di dialogo di collegamento. Una volta attivato il collegamento, compare il pannello finestra di dialogo di stampa. E' possibile specificare la coda di stampa, il file di stampa, il titolo della pagina di intestazione e di file OS/400. E' possibile inoltre selezionare la dimensione della pagina, la disposizione e il numero di copie.

Per utilizzare la stampa remota, è necessario installare IBM Toolbox per Java (5763-JC1) e aggiungerlo al classpath nel server iSeries:

```
QIBM/ProdData/HTTP/Public/jt400/lib/jt400.zip
```

E' possibile aggiornare il classpath aggiungendo la variabile di ambiente classpath o utilizzando il parametro classpath.

Nota: se questo messaggio compare durante la stampa nel server iSeries, IBM Toolbox per Java non è installato oppure le classi IBM Toolbox per Java non sono presenti nel classpath.

```
Failed to load class file: com/ibm/as400/access/PrintObjectList.class
Exception occurred during event dispatching:
java.lang.NoClassDefFoundError: com/ibm/as400/access/PrintObjectList
```

Proprietà di AWT (Abstract Window Toolkit) Remoto

Quando si esegue un'applicazione Java^(TM) AWT Remoto sul server iSeries, è necessario che la proprietà `os400.class.path.rawt` utilizzata possieda un valore pari a 1. Esistono numerose proprietà predefinite necessarie quando si utilizza l'AWT Remoto. Queste proprietà predefinite insieme alla versione appropriata dell'AWT Remoto e al CLASSPATH sono impostate quando si utilizza la proprietà `os400.class.path.rawt`. La versione di AWT Remoto viene impostata a seconda della versione JDK, che corrisponde alla versione predefinita, se non è specificata, oppure alla versione specificata con la proprietà `java.version`.

Di seguito vengono indicate la proprietà predefinite necessarie per l'AWT Remoto.

Per JDK 1.1.x:

- `awt.toolkit=com.ibm.rawt.CToolkit`

Per J2SDK:

- `awt.toolkit=com.ibm.rawt2.ahost.java.awt.AHToolkit`
- `java.awt.graphicsenv=com.ibm.rawt2.ahost.java.awt.AHGraphicsEnvironment`
- `java.awt.printerjob=com.ibm.rawt2.ahost.java.awt.print.AHPrinterjob`

Proprietà di AWT Remoto per il pannello remoto

Se il daemon del server o l'applicazione Java terminano in modo anomalo con questo messaggio, controllare la versione Java sul pannello remoto.

La versione JDK nell'host dell'applicazione/stazione utente non è compatibile con la versione di AWT remoto...

Per controllare il livello della versione, immettere `java -version` sulla riga comandi. E' possibile utilizzare questa nuova proprietà su un pannello remoto se si verifica un problema con la versione JDK. Questa proprietà non è applicabile per il server iSeries. Se la versione non si trova al livello 1.1.x, allora è necessario installare il livello appropriato. Se si trova al livello 1.1.x, è possibile eseguire il server di AWT Remoto o l'applicazione Java con questa proprietà che indica la versione Java:

```
-DjdkVersion=1.1.x
```

Limitazioni di SecurityManager per AWT (Abstract Window Toolkit) Remoto

SecurityManager Java^(TM) non è in genere utilizzato. Se viene installato un SecurityManager, tuttavia, è necessario che le chiamate elencate abbiano esito positivo affinché AWT Remoto funzioni.

- `SecurityManager.checkAccess(..)`
- `SecurityManager.checkMemberAccess(..)`
- `SecurityManager.checkExit(..)`
- `SecurityManager.checkRead(String file)`
- `SecurityManager.checkConnect(...)`
- `SecurityManager.checkListen(...)`
- `SecurityManager.checkAccept(...)`
- `SecurityManager.checkPropertiesAccess(..)`

Esempio: impostare AWT (Abstract Window Toolkit) Remoto per Java^(TM) su un pannello remoto di Windows

Questo esempio indica come impostare AWT Remoto su un pannello remoto di Windows^(R). Esistono molti altri modi per effettuare questa operazione, a seconda delle preferenze personali. E' possibile utilizzare un processo simile su altri sistemi operativi del pannello remoto. I processi di impostazione e avvio sono resi automatici con un file `.bat` Windows o con un'altra funzione di programmazione che il sistema operativo del pannello remoto fornisce.

Per impostare AWT (Abstract Window Toolkit) Remoto per Java^(TM) su un pannello remoto di Windows, effettuare le seguenti attività:

- Rendere i file di classe dell'AWT Remoto accessibili al pannello remoto.
Copiare i file di classe dell'AWT Remoto sul pannello remoto. Svolgere una delle seguenti attività:
 - Copiare `/QIBM/ProdData/Java400/jdk118/RAWTGui.zip` su `c:\rawt\RAWTGui.zip`.
 - Copiare `/QIBM/ProdData/Java400/jdk1x/RAWTGui.jar`, dove `x` rappresenta la propria versione di J2SDK (Java 2 Software Development Kit), Edizione standard (2, 3, o 4), su `c:\rawt2\RAWTGui.jar`.
- Avviare l'AWT Remoto sul pannello remoto immettendo quanto segue sulla riga comandi:

```
java -classpath c:\jdk1.1.8\lib\classes.zip;c:\rawt\RAWTGui.zip  
java com.ibm.rawt.server.RAWTPCServer
```

oppure

```
java -jar c:\rawt2\RAWTGui.jar
```

Per ulteriori informazioni, consultare Impostare AWT (Abstract Window Toolkit) Remoto per Java su un pannello remoto.

CBJ (Class Broker for Java)

CBJ Class Broker per Java^(TM) è una framework a scopo generale per la scrittura di applicazioni client/server in Java. Normalmente, un'applicazione client/server consiste sia in un oggetto client che in un oggetto server. Il server e il client sono responsabili per tutte le comunicazioni tra i due oggetti. Queste comunicazioni avvengono spesso tramite collegamento socket o RMI (Remote Method Invocation). RMI non è uno strumento flessibile e di facile utilizzo ed esiste una curva di apprendimento considerevole quando si tenta di utilizzare i socket in modo efficace.

CBJ è flessibile e di facile utilizzo e nasconde la complessità dei collegamenti socket. Ad eccezione di alcune chiamate alle classi CBJ inizializzano l'applicazione, un'applicazione client/server sottoposta al broker viene visualizzata come un'applicazione locale. CBJ gestisce tutte le comunicazioni e il caricamento di risorse tra il client e il server. E' quasi immediatamente evidente che alcuni oggetti del programma sono in esecuzione sul client e altri sul server. CBJ crea proxy client e server utilizzando il tempo di esecuzione CBJ. Una volta che il broker crea gli oggetti proxy, il client comunica con il server remoto chiamando metodi sul proxy del server. Allo stesso modo, l'oggetto server comunica con il client chiamando metodi sul proxy del client. Quindi, sia il lato client che il lato server vengono visualizzati come se chiamassero metodi degli oggetti locali.

Per informazioni sull'utilizzo di Class Broker per Java, consultare Impostare Class Broker per Java.

Impostare Class Broker for Java sul pannello remoto

Con CBJ (Class Broker for Java^(TM)), è possibile eseguire i programmi grafici Java abilitati a CBJ (Class Broker for Java) sul server iSeries e visualizzare i grafici in remoto.

E' possibile installare CBJ senza alcun aiuto oppure fare in modo che sia il responsabile di sistema ad installarlo. Se il responsabile di sistema installa il prodotto, tutti i programmatori possono condividere lo stesso codice Java.

E' possibile altrimenti installare CBJ su Windows^(R) 95/98/NT, UNIX^(R) o sul server iSeries. Nella maggior parte delle circostanze, è necessario installare CBJ sulla macchina client o sulla macchina server.

Nota: se la macchina client sta eseguendo un'applet del client a cui si è avuto accesso tramite un server di rete, allora non è necessario installare CBJ sulla macchina client.

Per installare CBJ sul server iSeries, consultare Installare Class Broker per Java su un server iSeries.

Per installare CBJ sulla stazione di lavoro, consultare Installare Class Broker per Java su Windows o UNIX.

Installare Class Broker per Java su un server iSeries

Per installare CBJ (Class Broker per Java^(TM)) su un server iSeries, seguire queste fasi:

1. Assicurarsi che IBM Developer Kit per Java sia installato correttamente. Consultare Installare IBM Developer Kit per Java per verificare l'installazione.
2. Selezionare un indirizzario nell'IFS (Integrated File System) per memorizzare il pacchetto CBJ denominato cbj_1.1.jar. Ad esempio, /usr/local.
3. Avviare Qshell Interpreter immettendo il comando QSH e andare all'indirizzario che si è selezionato nell'IFS. Ad esempio, /usr/local. Quando si installa IBM Developer Kit per Java, il pacchetto cbj_1.1.jar viene installato in QIBM/ProdData/Java400/ext.
4. Immettere questo comando in Qshell:

```
jar xvf "PATH"cbj_1.1.jar
```

"PATH" è il percorso indirizzario dove è collocato il pacchetto cbj_1.1.jar. Ad esempio, QIBM/ProdData/Java400/ext.

I file CBJ vengono estratti nel sottoindirizzario, denominato/usr/local/JCBroker. Consultare Contenuto del pacchetto di cbj_1.1.jar per ulteriori informazioni.

5. Creare un programma Java sul proprio server iSeries per jcb.jar, immettendo questo comando:

```
CRTJVAPGM CLSF('/usr/local/JCBroker/lib/jcb.jar')
```
6. Se si desidera utilizzare le classi CBJ (non in modalità di debug), è necessario aggiungere JCBroker\lib e JCBroker\lib\jcb.jar all'opzione del classpath nella riga comandi Java. Non è consigliato aggiungere JCBroker\lib e JCBroker\lib\jcb.jar alla variabile di ambiente CLASSPATH, in quanto verificarsi un conflitto con il caricamento della classe quando si eseguono le applet che impostano applet_jcb.jar nella tag ARCHIVE.
7. Se si intende eseguire CBJ in modalità di debug, sostituire jcb.jar con jcbd.jar nel classpath o sostituire il valore del classpath sulla riga comandi Java. E' inoltre possibile utilizzare il seguente comando:

```
CRTJVAPGM CLSF('/usr/local/JCBroker/lib/jcbd.jar')
```
8. Per ulteriori informazioni su API CBJ, esecuzione demo, modifica proprietà, progettazione e scrittura di applicazioni CBJ e altri argomenti, far riferimento al file JCBroker/index.html nel pacchetto cpj_1.1.jar.

Installare Class Broker per Java su Windows o UNIX

Per installare Class Broker per Java^(TM) (CBJ) su Windows^(R), effettuare quanto segue:

1. Assicurarsi che JDK/JRE1.1 o JDK/JRE1.2 sia installato correttamente.
2. Selezionare un indirizzario in cui memorizzare il pacchetto CBJ, denominato cbj_1.1.jar. Ad esempio, C:\.
3. Passare al proprio indirizzario (C:\) e immettere questo comando:

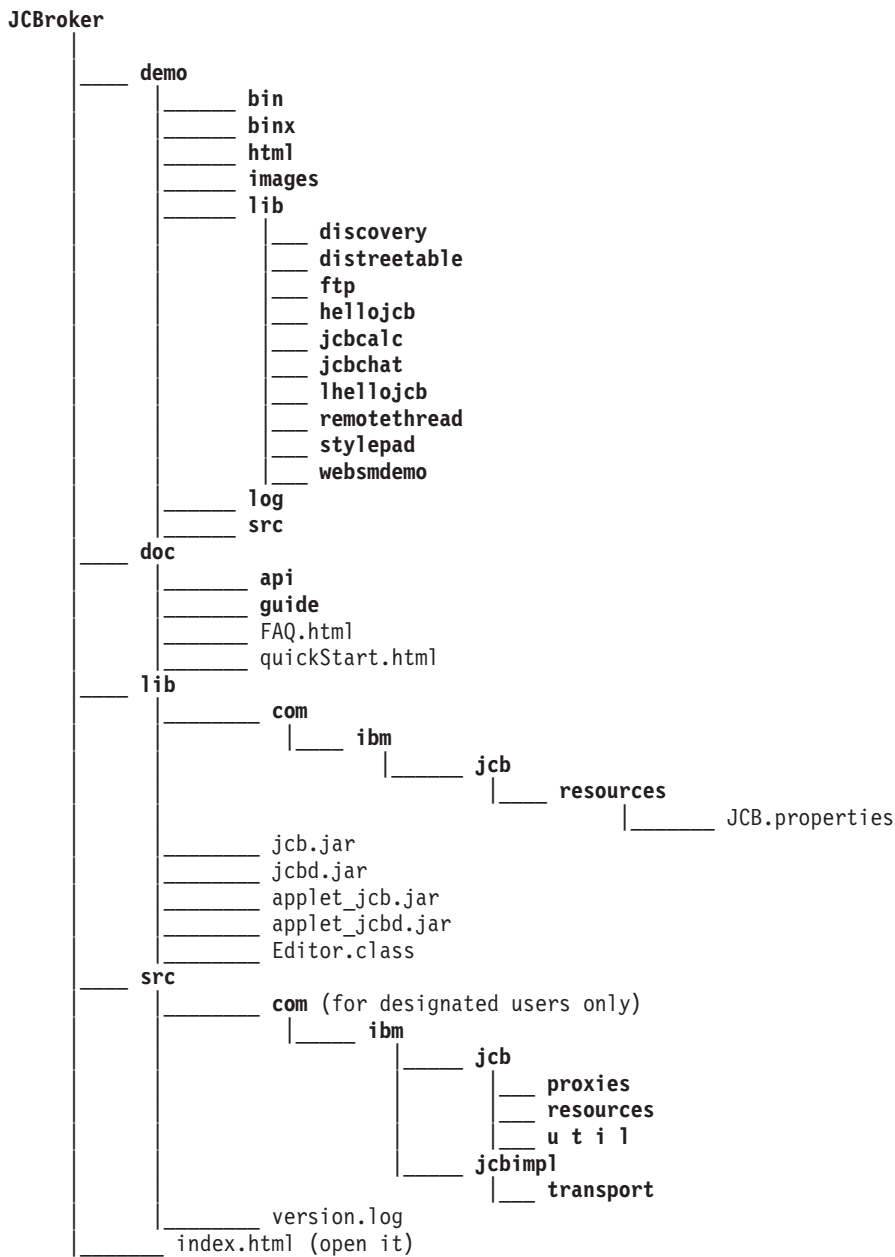
```
C:\ > jar xvf cbj_1.1.jar
```

I file CBJ vengono estratti e copiati nell'indirizzario. Consultare Contenuto del pacchetto di cbj_1.1.jar per ulteriori informazioni.

4. Se si desidera utilizzare le classi CBJ (non in modalità di debug), è necessario aggiungere C:\JCBroker\lib e C:\JCBroker\lib\jcb.jar all'opzione del classpath nella riga comandi Java.
Nota: non è consigliato di aggiungerlo alla variabile di ambiente CLASSPATH del sistema, in quanto potrebbe essere in conflitto con il caricamento della classe quando si eseguono le applet che impostano applet_jcb.jar nella tag ARCHIVE.
5. Per eseguire CBJ in modalità di debug, sostituire jcb.jar con jcbd.jar nel classpath.
Per installare CBJ su UNIX^(R), seguire le stesse fasi indicate per Windows, ad eccezione delle seguenti modifiche, dipendenti dal sistema:
 - Sostituire il separatore file Windows "\" con il separatore file UNIX "/".
 - Sostituire il separatore classpath Windows ";" con il separatore classpath UNIX ":".
 - Sostituire le variabili di ambiente del sistema Windows "%XXX%" con le variabili di ambiente del sistema UNIX "\$XXX".
6. Per ulteriori informazioni su API CBJ, esecuzione demo, modifica proprietà, progettazione e scrittura di applicazioni CBJ e altri argomenti, far riferimento al file JCBroker/index.html nel pacchetto cpj_1.1.jar.

Contenuto del pacchetto di cbj_1.1.jar

Di seguito è riportato il pacchetto CBJ ovvero Class Broker per Java^(TM) (cbj_1.1.jar). Le voci in grassetto indicano gli indirizzari principali dei sottoindirizzari. Le voci non in grassetto mostrano i file.



Il livello successivo sotto l'indirizzario JCBroker consiste nei seguenti dati:

demo

Questo indirizzario contiene i seguenti sottoindirizzari: bin, binx, html, images, lib, log e src. Tali indirizzari contengono i file .exe di Windows^(R), i file .exe di UNIX^(R), le pagine HTML, file GIF, file class e il codice sorgente (di alcuni esempi) di diverse applicazioni demo. E' possibile eseguire questi esempi demo seguendo le istruzioni demo. L'indirizzario di registrazione è un segnaposto per i file di registrazione di debug.

doc

Questo indirizzario contiene i due seguenti sottoindirizzari: api e guide. La guida API CBJ e una guida per l'utente su CBJ sono memorizzate in questo ambito. La documentazione API è stata generata con lo strumento javadoc, J2SDK. Per sfogliare il documento API o la guida per l'utente, aprire il file index.html

nell'indirizzario corrispondente. La guida per l'utente è un documento di livello introduttivo, dove la guida QuickStart è un documento della voce di traccia rapida.

Questo indirizzario contiene inoltre un file FAQ.html e il file quickStart.html che spiega come installare e utilizzare CBJ e mostra come eseguire gli esempi demo.

lib

Questo indirizzario contiene le risorse e classi CBJ necessarie all'utente per sviluppare, eseguire e disporre le nuove applicazioni di base CBJ. Le classi vengono compresse nel file jcb.jar. Per la modalità debug, le classi vengono compresse nel file jcbd.jar. Le classi incluse nella tag ARCHIVE di un'applet vengono compresse nel file applet_jcb.jar. Per la modalità debug, le classi vengono compresse nel file applet_jcbd.jar. Il file delle proprietà JCB (Java Class Broker), denominato JCB.properties, si trova nel sottoindirizzario com/ibm/jcb/resources. Viene letto dal tempo di esecuzione CBJ quando viene avviato. E' possibile modificare questo file in modo che corrisponda alle proprie preferenze. Nella sezione di installazione e impostazione vengono spiegate le proprietà. Le proprietà che si applicano solo alle applet si trovano nel file applet_JCB.properties, che fa parte di applet_jcb.jar e applet_jcbd.jar. E' inoltre possibile modificare questo file secondo le proprie preferenze durante l'esecuzione nella modalità applet. Per modificare questo file, eseguire l'applicazione Editor Java ubicata in questo indirizzario.

src

Questo indirizzario contiene il codice sorgente CBJ e il file di registrazione della versione interna, denominato version.log. L'indirizzario è vuoto ad eccezione dei pacchetti selezionati.

index.html

Questa è la pagina iniziale che presenta all'utente il resto della documentazione.

Native Abstract Windowing Toolkit

NAWT (Native Abstract Windowing Toolkit) fornisce applicazioni e servlet Java^(TM) con la capacità di utilizzare la funzione grafica in AWT (Abstract Windowing Toolkit) di JDK (Java Development Kit).

E' possibile utilizzare NAWT (detto "naw-TEE") nelle applicazioni di grafica Java che non richiedono un'interazione diretta da parte dell'utente. Un esempio di tale applicazione potrebbe essere il codice che genera un'immagine del tipo di un file codificato JPEG o GIF o di un flusso di emissione. Per questo rilascio, NAWT supporta le versioni 1.2 e 1.3 di JDK.

E' possibile inoltre utilizzare NAWT come un'alternativa a RAWT (Abstract Windowing Toolkit) Remoto in relazione alle applicazioni non interattive. In RAWT, l'elaborazione della grafica viene eseguita da un server collegato al PC. E' possibile utilizzare RAWT per una classe più generale di applicazioni basate sulla grafica, ma è necessario il PC remoto e di solito si verifica il sovraccarico di prestazioni a causa dell'attesa della rete che si determina tra il server iSeries e il server GUI del PC.

NAWT utilizza X Window System come motore di grafica sottostante, che elabora la grafica interamente su un server iSeries. X Window System è un sistema di grafica che fornisce una base client/server per la visualizzazione della grafica.


Per ulteriori informazioni sull'installazione di NAWT e sul suo utilizzo nelle applicazioni, consultare gli argomenti che seguono:

Installare NAWT

Fornisce le istruzioni per l'installazione di NAWT sul server iSeries ed informazioni sul server X Window per la grafica consigliato, su PASE OS/400, sulle correzioni di software necessarie (le PTF), su iSeries Tools for Developers PRPQ ed altro.

Suggerimenti sull'utilizzo di VNC

Fornisce informazioni su come avviare un server VNC da un programma CL e su come arrestarlo.

Per ulteriori informazioni sul server VNC utilizzato da NAWT, fare riferimento alla pagina web AT&T Research Virtual Network Computing  .

Installare NAWT

NAWT fa affidamento su X Window System come motore di grafica. Il server di grafica X Window consigliato per questo rilascio è VCN (Virtual Network Computing), che utilizza per l'esecuzione PASE (Portable Application Solutions Environment) OS/400.

PASE è un ambiente di tipo UNIX capace di eseguire la maggior parte di file eseguibili binari compilati per il sistema operativo AIX di IBM. VNC è un prodotto dei laboratori di ricerca AT&T e fornisce un server X Window *virtuale*; ovvero, un server X che non richiede all'utente un collegamento diretto all'unità video con capacità grafica.

Per installare ed eseguire NAWT (Native Abstract Windowing Toolkit), effettuare le seguenti attività:

1. Installare PASE OS/400
2. Installare le PTF NAWT
3. Installare iSeries Tools for Developers PRPQ
4. Creare un file della parola d'ordine VNC
5. Configurare le proprietà di sistema Java
6. Avviare il server VNC
7. Impostare le variabili di ambiente
8. Verificare la procedura di installazione

Installare PASE OS/400: Ordinare e installare PASE (Portable Application Solutions Environment) OS/400, 5722SS1, opzione 33. Per ulteriori informazioni, consultare PASE OS/400.

Installare le PTF di NAWT: Prima di installare qualsiasi correzione al software (PTF), assicurarsi di avere l'opzione del programma su licenza 5722JV1 corrispondente alla versione JDK che si desidera utilizzare, effettuando quanto di seguito specificato:

1. Immettere il comando GO LICPGM (Gestione programmi su licenza) sulla riga comandi.
2. Selezionare l'opzione 10 (Visualizzare programmi su licenza installati) e verificare che l'opzione il programma su licenza 5722JV1 corrispondente alla versione JDK che s'intende utilizzare, sia installata.


Le opzioni disponibili sono le seguenti:

Versione JDK	Opzione
JDK 1.2	57SSJV1 opzione 3
JDK 1.3	57SSJV1 opzione 5

➤ Applicare l'ultimo gruppo di correzioni al software di Java per reperire tutte le recenti correzioni di NAWT. Per ulteriori informazioni relative alle correzioni al software, consultare Utilizzare correzioni al software.

Installare iSeries Tools for Developers PRPQ: Installare iSeries Tools for Developers PRPQ (5799PTL). Se non si dispone di tale prodotto, è necessario ordinarlo.

Le versioni più recenti di PRPQ comprendono una versione pre-compilata abilitata a PASE OS/400 di VNC (Virtual Network Computing). Le versioni meno recenti non includono VNC. La modalità di installazione di PRPQ viene definita dalla versione a propria disposizione:

- Per le versioni di PRPQ ordinate il 14 giugno 2002 o successivamente: completare questa operazione utilizzando le istruzioni di installazione disponibili sul sito web Application Factory - iSeries Tools for Development .

Nota: per installare il supporto VNC disponibile nel PRPQ, seguire soltanto le istruzioni di installazione presenti sul sito web Application Factory. Non è necessario seguire le istruzioni relative all'impostazione.

- Per le versioni di PRPQ ordinate prima del 14 giugno 2002, fare riferimento all'argomento Installare versioni meno recenti di iSeries Tools for Developers PRPQ per completare questa operazione.


Creare un file della parola d'ordine VNC: Per impostazione predefinita, VNC richiede un file della parola d'ordine che protegga il pannello VNC contro l'accesso di un utente non autorizzato. La modalità di creazione di una parola d'ordine codificata dipende dal tipo di versione di PRPQ utilizzata:

- Per le versioni di PRPQ ordinate il 14 giugno 2002 o successivamente, utilizzare i seguenti comandi dalla richiesta comandi iSeries:

```
MKDIR DIR('/home/your_profile_name/.vnc')
QAPTL/VNCPASSWD USEHOME(*NO) PWDFILE('/home/your_profile_name/.vnc/passwd')
```

- Per le versioni di PRPQ ordinate prima del 14 giugno 2002, utilizzare i seguenti comandi dalla richiesta comandi iSeries:

```
MKDIR DIR('/home/your_profile_name/.vnc') VNCSAVF/VNCPASSWD USEHOME(*NO)
PWDFILE('/home/your_profile_name/.vnc/passwd')
```

Soltanto l'utente che avvia il server VNC, deve disporre del file relativo alla parola d'ordine. 

Configurare le proprietà di sistema Java

Impostare le proprietà di sistema Java. La prima riga configura Java in base al tipo di versione di JDK (1.2 o 1.3) che si desidera utilizzare e la seconda riga abilita NAWT:

```
java.version=version
os400.class.path.rawt=2
```

Dove *version* indica la versione **1.2** o **1.3**, a seconda di quale versione di JDK si desidera utilizzare.

Per avere una guida su come impostare le proprietà di sistema Java, consultare Personalizzare il server iSeries per IBM Developer Kit per Java.

Avviare il server VNC

Per avviare il server VNC, effettuare quanto segue:

1. Avviare una shell PASE:

```
CALL QP2TERM
```

2. Dalla shell PASE, avviare la versione configurata NAWT del server VNC:

```
/Q0penSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
```

Quando si avvia il server, verrà visualizzato un messaggio simile al seguente, "Il nuovo 'X' desktop è *systemname:1*." Tenere a mente questo nome del desktop per la prossima fase.

Nota: il numero del pannello (cioè, il numero a destra dei due punti) potrebbe essere differente se si avvia un altro server VNC. Ogni server VNC in esecuzione contemporaneamente richiede un numero di pannello univoco. Quando non si specifica il numero del pannello per la chiamata di `vncserver_java`, il programma `vncserver_java` reperirà un pannello disponibile. Per richiedere un pannello specifico utilizzare il seguente comando di avvio del programma `vncserver_java`:

```
/Q0penSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java :n
```

dove *n* è il numero del pannello che si desidera utilizzare.

Impostare le variabili di ambiente

In qualsiasi sessione in cui viene eseguito Java con NAWT, è necessario indicare a Java dove reperire il server X e i file delle autorizzazioni X. L'utente fornisce tali informazioni a Java mediante l'impostazione delle variabili di ambiente.

Nella sessione in cui si desidera eseguire i programmi Java, impostare la variabile di ambiente DISPLAY sul proprio nome sistema e sul numero di pannello (cioè, il valore immesso quando è stato eseguito `vncserver_java`).

Inoltre, impostare la variabile di ambiente XAUTHORITY su `/home/your_profile_name/.Xauthority`.

Ad esempio, nella richiesta comandi di iSeries, immettere i seguenti comandi:

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:1')
ADDENVVAR ENVVAR(XAUTHORITY) VALUE('/home/your_profile_name/.Xauthority')
```

Note:

- Il file `.Xauthority` viene creato o modificato quando viene avviato il server VNC. L'autorizzazione del server X impone l'utilizzo del protocollo di sicurezza sulla propria connessione al server X, evitando alle applicazioni di altri utenti l'intercettazione delle richieste del server X.
- L'impostazione di queste variabili di ambiente deve essere eseguita solo dall'utente che avvia realmente la JVM. Ad esempio, in un ambiente servlet, solo l'utente che avvia il motore servlet deve impostarle.

Verificare la procedura di installazione

Per eseguire l'applicazione Java di verifica NAWT, verificare che i passi sopraindicati, siano stati completati ed utilizzare il seguente comando:

```
JAVA CLASS(NAWTtest) CLASSPATH('/QIBM/ProdData/Java400/')
```

L'applicazione di verifica crea un'immagine codificata JPEG denominata `/tmp/NAWTtest.jpg` nell'IFS (Integrated File System). Verificare che l'applicazione abbia creato il file e che non abbia prodotto alcun errore Java. Se si desidera visualizzare l'immagine, è possibile utilizzare la modalità binaria per caricare il file relativo ad essa su un sistema con capacità grafica.

Installare versioni meno recenti di iSeries Tools for Developers

➤ Nelle versioni di iSeries Tools for Developers PRPQ (5799PTL) ordinate prima del 14 giugno 2002, PRPQ non comprende una versione di VNC (Virtual Network Computing) abilitata a PASE OS/400 precompilata.

Utilizzare le seguenti istruzioni per determinare se si dispone della versione di PRPQ aggiornata e per installare VNC se si possiede una versione meno recente di PRPQ.

Determinare se l'utente dispone di una versione di PRPQ aggiornata: Se si dispone di PRPQ 5799-PTL ma non si è sicuri di avere la versione aggiornata contenente VNC, verificare l'esistenza del seguente file:

```
/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
```

La versione aggiornata di PRPQ comprende il file `vncserver_java`, ma le versioni meno recenti no. Se `vncserver_java` non è presente sul server iSeries, è possibile ordinare ed installare la versione più recente di PRPQ oppure utilizzare le seguenti istruzioni per completare l'installazione di VNC.


Installare VNC: Per installare VNC su una versione meno recente di iSeries Tools for Developers PRPQ, completare i seguenti passi.

1. Creare i file di salvataggio sul proprio server iSeries immettendo i seguenti comandi:

```

crtlib vncsavf
crtsavf vncsavf/vncpasswd
crtsavf vncsavf/vnc
crtsavf vncsavf/fonts
crtsavf vncsavf/icewm

```

2. Fare clic sui collegamenti nella seguente lista per scaricare i file di salvataggio nella propria stazione di lavoro, dal sito web Application Factory - iSeries Tools for Development  :

- vnc.savf
- vncpasswd.savf
- fonts.savf
- icewm.savf

3. Utilizzare l'FTP per trasferire i file di salvataggio dalla propria stazione di lavoro al server iSeries eseguendo i seguenti comandi sulla propria stazione di lavoro:

```

ftp youriseriesserver
bin
cd /qsys.lib/vncsavf.lib
put vnc.savf
put vncpasswd.savf
put fonts.savf
put icewm.savf
quit


```

4. Ripristinare i file di salvataggio eseguire i seguenti comandi sul server iSeries:

```

RSTOBJ OBJ(*ALL) SAVLIB(VNCSAVF) DEV(*SAVF) SAVF(VNCSAVF/VNCPASSWD)
RST DEV('/Qsys.lib/vncsavf.lib/vnc.file') OBJ('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc*')
RST DEV('/Qsys.lib/vncsavf.lib/fonts.file') OBJ('/QOpenSys/QIBM/ProdData/DeveloperTools/fonts*')
RST DEV('/Qsys.lib/vncsavf.lib/icewm.file') OBJ('/QOpenSys/QIBM/ProdData/DeveloperTools/icewm*')

```

5. Continuare l'installazione di NAWT mediante creazione del file delle parole d'ordine VNC. 

Suggerimenti sull'utilizzo di VNC

 Questa pagina discute i suggerimenti aggiuntivi sull'utilizzo di VNC (Virtual Network Computing).

Avviare 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 da un programma CL. Si presume che il pannello :*n* non sia ancora in esecuzione. Si presume inoltre che sia già stato creato il file della parola d'ordine richiesta da VNC eseguendo il comando VNCPASSWD.

```

ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
call qp2shell parm('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')

```

dove *n* è il valore numerico che rappresenta il numero del pannello che si intende avviare.


Arrestare un server del pannello VNC

Per arrestare un server VNC che è stato avviato:

```

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 arrestare. 

Capitolo 2. Sicurezza Java

La maggior parte dei programmi Java^(TM) che vengono eseguiti su un server iSeries sono applicazioni, non applet, in questo modo il modello di sicurezza "sandbox" non le limita. Da un punto di vista della sicurezza, le applicazioni Java sono soggette alle stesse limitazioni di qualsiasi altro programma su un server iSeries. Per eseguire un programma Java su un server iSeries, è necessario disporre dell'autorizzazione al file di classe nell'integrated file system. Una volta che il programma è avviato, viene eseguito sotto l'autorizzazione dell'utente.

E' 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 dettagli sui due nuovi parametri dell'autorizzazione adottata, che risultano USRPRF e USEADPAUT.

➤ IBM Developer Kit per Java fornisce le seguenti caratteristiche di sicurezza per le applicazioni Java:

Il modello di sicurezza Java

Il programma di caricamento e di verifica del bytecode, all'interno della JVM (Java virtual machine), fornisce uno strumento di sicurezza che utilizza il modello di sicurezza 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.

Java Cryptography Extension

L'implementazione di JCE (Java Cryptography Extension) sul server iSeries è compatibile con l'implementazione di Sun Microsystems, Inc. Questa documentazione copre gli aspetti univoci dell'implementazione iSeries. Si presume una familiarità dell'utente con la documentazione generale relativa alla JCE.

Java Secure Socket Extension

JSSE (Java Secure Socket Extension) è l'implementazione Java del protocollo SSL (Secure Sockets Layer). JSSE utilizza i protocolli SSL e TSL (Transport Layer Security) per abilitare i client e i server a portare avanti comunicazioni protette su TCP/IP. Questa documentazione descrive gli aspetti univoci dell'implementazione iSeries di JSSE. Si presume una familiarità dell'utente con la documentazione generale relativa a JSSE.

JAAS (Java Authentication and Authorization Service)

JAAS (Java Authentication and Authorization Service) rappresenta un altro elemento di sicurezza che IBM Developer Kit per Java supporta. Attualmente, J2SDK (Java 2 Software Development Kit), Edizione Standard fornisce controlli di accesso basati sul luogo in cui il codice è stato creato e sulla persona che 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.

Java Generic Security Service

JGSS (Java Generic Security Service) rappresenta un altro elemento di sicurezza supportato da IBM Developer Kit per Java. JGSS fornisce un'interfaccia generica per proteggere lo scambio di messaggi tra le applicazioni. JGSS supporta diversi meccanismi di sicurezza basati sulla chiave segreta, la chiave pubblica o altre tecnologie di sicurezza.

Nota: per J2SDK, versione 1.4, 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. <<

Modello di sicurezza Java


E' possibile scaricare le applet Java^(TM) 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. 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`.

Per ulteriori informazioni sulla sicurezza, consultare la documentazione sulla Security by Sun Microsystems, Inc. e Secure applications with SSL.

JCE (Java Cryptography Extension)

JCE ovvero Java^(TM) Cryptography Extension 1.2 è una estensione standard in J2SDK (Java 2 Software Development Kit), Edizione standard. L'implementazione di JCE su un server iSeries è compatibile con l'implementazione di Sun Microsystems, Inc. Questa documentazione tratta gli aspetti univoci dell'implementazione iSeries. Si presume una familiarità dell'utente con tale materiale generale relativo alle estensioni JCE. Per rendere più facile per l'utente la gestione di tale documentazione e delle informazioni

iSeries, viene fornito un collegamento alla documentazione JCE di Sun  .

Sul server iSeries, il livello di codifica viene controllato dal prodotto Cryptographic Access Provider. Questo è disponibile in due versioni, 5722-AC2 e 5722-AC3. Il prodotto 5722-AC3 riconosce tutti gli algoritmi di codifica. Il prodotto 5722-AC2 non supporta Triple-DES e limita gli algoritmi simmetrici a 56 bit e gli algoritmi asimmetrici a 1024 bit.

Con l'eccezione delle limitazioni su 5722-AC2 già menzionate, il Fornitore JCE IBM supporta gli algoritmi seguenti:

- DES
- Triple-DES
- RC2
- RC4
- Blowfish
- RSA
- Diffie-Hellman
- DSA
- Mars
- MD2
- MD5
- SHA-1
- Seal

In aggiunta, questo fornisce un generatore di numeri casuali.

» Se si desidera utilizzare JCE IBM con Java 1.2, modificare il file /QIBM/ProdData/Java400/jdk12/lib/security/java.security. « La sezione del file che deve essere modificata è visualizzata nel modo seguente.

```
#
# To use the IBMJCE security provider, you need to:
# 1) Install an IBM Cryptographic Access Provider Product
# 2) uncomment the second provider entry that follows.
#
# List of providers and their preference orders:
#
security.provider.1=sun.security.provider.Sun
#security.provider.2=com.ibm.crypto.provider.IBMJCE
```

» Se si desidera utilizzare IBM JCE con la versione di Java 1.3, modificare il file /QIBM/ProdData/OS400/Java400/jdk/lib/security/java.security. « La sezione del file che deve essere modificata viene visualizzata come segue.

```
#
# To use the IBMJCE security provider, you need to:
# 1) Install an IBM Cryptographic Access Provider Product
# 2) Uncomment the third provider entry that follows.
#
# List of providers and their preference orders:
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsa.jca.Provider
#security.provider.3=com.ibm.crypto.provider.IBMJCE
```

In entrambi i casi bisogna solo cancellare un carattere.

JSSE (Java Secure Socket Extension)

JSSE (JavaTM Secure Socket Extension) è l'implementazione Java del protocollo SSL (Secure Sockets Layer). JSSE utilizza i protocolli SSL e TLS (Transport Layer Security) per abilitare i client e i server a portare avanti comunicazioni protette su TCP/IP.

JSSE fornisce le seguenti funzionalità:

- Codifica i dati
- Autentica gli ID degli utenti remoti
- Autentica i nomi dei sistemi remoti
- Esegue l'autenticazione client/server
- Assicura l'integrità del messaggio


Integrato nell'edizione standard di J2SDK (Java 2 Software Development Kit), versione 1.4, JSSE fornisce più funzioni rispetto a quelle fornite da SSL da solo. Per ulteriori informazioni, consultare i seguenti argomenti:

Utilizzare SSL (JSSE, versione 1.0.8)

SSL fornisce un mezzo di autenticazione di un server e di un client per garantire riservatezza e integrità di dati. Tutte le comunicazioni SSL cominciano con una "presentazione" (handshake) tra il server e il client. Durante la presentazione, SSL negozia il tipo di codifica che il client e il server utilizzano per comunicare l'uno con l'altro. Questo insieme di codifiche risulta una combinazione di varie caratteristiche della sicurezza disponibili tramite SSL. E' possibile utilizzare SSL con qualsiasi versione supportata di JDK precedente alla J2SDK, versione 1.4.

Utilizzare JSSE, versione 1.4

JSSE agisce come una framework che estrae i meccanismi sottostanti sia di SSL che di TLS. Estrae 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. Tali informazioni si applicano solo quando si utilizza JSSE sui server iSeries su cui è in esecuzione J2SDK, versione 1.4.

Nota: queste informazioni riguardano la versione di JSSE inviata nel pacchetto J2SDK, versione 1.4. Per le versioni precedenti di JSSE, consultare Java Secure Socket Extension sul sito web Java della Sun  [☞](#)

Utilizzare SSL (JSSE, versione 1.0.8)

E' possibile utilizzare JSSE (Java Secure Socket Extension, versione 1.0.8), che è l'implementazione Java degli SSL (secure sockets layer), per rendere l'applicazione Java^(TM) più sicura. SSL effettua le seguenti operazioni per migliorare la sicurezza della propria applicazione:

- Protegge i dati di comunicazione tramite codifica.
- Autentica gli ID utente remoto.
- Autentica i nomi del sistema remoto.

Nota: SSL utilizza un certificato digitale per codificare la comunicazione socket della propria applicazione Java. I certificati digitali sono uno standard Internet per identificare le applicazioni, gli utenti e i sistemi sicuri. E' possibile controllare i certificati digitali utilizzando IBM Digital Certificate Manager. Per ulteriori informazioni, consultare IBM Digital Certificate Manager.

Per rendere la propria applicazione Java più sicura utilizzando SSL:

- Preparare il server iSeries in modo che supporti SSL.
- Progettare la propria applicazione Java in modo che utilizzi SSL, tramite le seguenti operazioni:
 - Modificare il proprio codice socket Java in modo che utilizzi le produzioni socket se l'utente non le utilizza già.
 - Modificare il proprio codice Java in modo che utilizzi SSL.
- Utilizzare un certificato digitale per rendere la propria applicazione Java più sicura effettuando le seguenti operazioni:
 1. Selezionare un tipo di certificato digitale da utilizzare.
 2. Utilizzare il certificato digitale quando si esegue la propria applicazione.

E' inoltre possibile registrare la propria applicazione Java come applicazione sicura utilizzando l'API QsyRegisterAppForCertUse. Per ulteriori informazioni, consultare QsyRegisterAppForCertUse.

[☞](#) Per ulteriori informazioni relative alla versione Java di SSL, consultare JSSE (Java Secure Socket Extension)  di Sun Microsystems, Inc. [☞](#)

Preparare il server iSeries per il supporto SSL (secure socket layer)

Per preparare il sistema a utilizzare SSL (secure socket layer), è necessario installare LP di Digital Certificate Manager:

- [☞](#) 5722-SS1 OS/400 - Digital Certificate Manager [☞](#)

E' necessario inoltre installare uno di questi LP del Fornitore ad accesso crittografico:

- [☞](#) 5722-AC1 Fornitore ad accesso crittografico 40-Bit
- 5722-AC2 Fornitore ad accesso crittografico 56-Bit

- 5722-AC3 Fornitore ad accesso crittografico 128-Bit <<

E' necessario inoltre assicurarsi della possibilità di accedere o creare un certificato digitale sul sistema. Per ulteriori informazioni sulla gestione del certificato digitale iSeries e su Internet, consultare Introduzione a IBM Digital Certificate Manager.

Fornitori ad accesso crittografico

I Fornitori di accesso crittografico offrono diverse serie di codifiche che il sistema può utilizzare. Un insieme di codifiche risulta essere una combinazione di funzioni di sicurezza differenti. Questa lista indica quale insieme di codifiche offre ogni Fornitore di accesso crittografico:

>> 5722-AC1 Fornitore ad accesso crittografico 40-Bit <<

- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

>> 5722-AC2 Fornitore ad accesso crittografico 56-Bit <<

- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_DES_CBC_MD5
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

>> 5722-AC3 Fornitore ad accesso crittografico 128-Bit <<

- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC2_CBC_128_MD5
- SSL_RSA_WITH_DES_CBC_MD5
- SSL_RSA_WITH_3DES_EDE_CBC_MD5

E' possibile che l'utente risulti limitato dalla scelta nel Fornitore ad accesso crittografico a seconda del paese o regione nel quale si trovi. Una volta che un Fornitore ad accesso crittografico viene caricato, è possibile utilizzare qualsiasi insieme di codifiche che quel Fornitore offre.

Modificare il codice Java in modo da utilizzare le 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.

Consultare Esempi: modificare il codice Java^(TM) in modo da utilizzare le produzioni socket del server per un esempio di un programma client convertito per poter utilizzare le produzioni socket.

Consultare Esempio: modificare il codice Java in modo da utilizzare le produzioni socket del client per un esempio di un programma client convertito per poter utilizzare le produzioni socket.

Esempi: modificare il codice Java in modo da utilizzare le 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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/* 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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

/* 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();
}
}

```

Per informazioni di background, consultare [Modificare il codice Java™](#) in modo da utilizzare le produzioni socket.

Esempi: modificare il codice Java in modo da utilizzare le 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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

/* 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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

/* 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.
    }
}

```

Per informazioni di background, consultare [Modificare il codice Java^{\(TM\)} in modo da utilizzare le produzioni socket](#).

Modificare il codice Java in modo da utilizzare SSL (secure socket 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 [Modificare il codice Java^{\(TM\)} in modo da utilizzare le 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 inicializzarla:

```
SocketFactory newSF = SSLSocketFactory.getDefault();
```

3. Utilizzare la nuova `SocketFactory` per inicializzare 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.

Consultare [Esempi: modificare il client Java per utilizzare SSL \(secure socket layer\)](#) e [Esempi: modificare il server Java per utilizzare SSL \(secure socket layer\) per codici di esempio](#).

Esempi: modificare il server Java in modo da utilizzare SSL (secure socket 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: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/* 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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/* 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: modificare il client Java in modo da utilizzare SSL (secure socket 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 factorySSLClient, che utilizza SSL.

Esempio 1: classe factorySocketClient semplice senza supporto SSL

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
/* 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

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

// 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.
    }
}

```

Per informazioni di background, consultare [Modificare il codice JavaTM](#) in modo da utilizzare SSL (secure sockets layer).

Selezionare un certificato digitale da utilizzare

E' necessario considerare numerosi fattori quando si decide quale certificato digitale utilizzare. E' possibile utilizzare il certificato predefinito del sistema oppure specificare un altro certificato da utilizzare.

E' possibile utilizzare il certificato predefinito del sistema se:

- Non si possiede alcun requisito specifico di sicurezza per l'applicazione Java^(TM).
- 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. Per ulteriori informazioni sulla gestione del certificato digitale, consultare Introduzione a IBM Digital Certificate Manager.

Se non si desidera utilizzare il certificato predefinito del sistema, è necessario scegliere un altro certificato da utilizzare. E' 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.

» E' necessario utilizzare il certificato utente se:

- l'applicazione è in esecuzione come un'applicazione client.
- si desidera che il certificato identifichi l'utente che gestisce l'applicazione.

E' 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 contenitore di certificati a cui si è in grado di accedere.

Utilizzare il 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 MYCERTIFICATE si trova nel contenitore certificati digitali YOURDCC, il comando java sarà:

```
java -Dos400.certificateContainer=YOURDCC  
-Dos400.certificateLabel=MYCERTIFICATE MyClass
```

Se non è stato ancora deciso quale certificato digitale utilizzare, consultare Selezionare un certificato digitale da utilizzare. E' possibile inoltre decidere di utilizzare il certificato predefinito del sistema, memorizzato nel relativo contenitore.

Per utilizzare il certificato digitale predefinito del sistema, non è necessario alcuna specificazione di un certificato o di un contenitore del certificato. L'applicazione Java utilizza automaticamente il certificato digitale predefinito del sistema.

Per ulteriori informazioni sulla gestione del certificato digitale iSeries e su Internet, consultare Informazioni preliminari di IBM Digital Certificate Manager.

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 nei relativi contenitori. Se si desidera utilizzare un certificato predefinito del relativo contenitore, 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. E' necessario che MYCERTIFICATE si trovi nel contenitore di certificato predefinito del sistema per essere utilizzato da MyClass.

Contenitori certificati digitali e proprietà -os400.certificateContainer: I contenitori di certificati digitali memorizzano certificati digitali. Se si desidera utilizzare il contenitore di certificati predefinito del sistema iSeries, non è necessario specificare un contenitore di certificati. Per utilizzare un contenitore di certificati digitali specifico, è necessario specificare quel contenitore nel comando java utilizzando questa proprietà:

```
os400.certificateContainer=
```

Ad esempio, se il nome del contenitore di certificati che contiene il certificato digitale che si desidera utilizzare è denominato 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 nel relativo contenitore denominato 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 nel relativo contenitore, allora il comando java che bisogna immettere dovrebbe assumere questa forma:

```
java -Dos400.certificateContainer=MYDCC  
-Dos400.certificateLabel=MYCERTIFICATE MyClass
```



Utilizzare JSSE (Java Secure Socket Extension), versione 1.4

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 chiamata IBM JSSE. IBM JSSE include un fornitore JSSE nativo di iSeries ed uno Java puro.

Per ulteriori informazioni su come configurare il server iSeries in modo da supportare JSSE, utilizzare i collegamenti che seguono:

Configurare il server per supportare JSSE

Fornisce informazioni su come configurare il server iSeries per l'utilizzo di IBM JSSE. Le informazioni includono i requisiti software, le istruzioni su come modificare i fornitori JSSE e le proprietà di sistema e della sicurezza necessarie.

Utilizzare il fornitore JSSE nativo di iSeries

Fornisce informazioni sull'utilizzo delle implementazioni native di iSeries della classe KeyStore JSSE e della classe SSLConfiguration.

Esempi JSSE

Utilizzare i programmi di esempio per scoprire come è possibile utilizzare JSSE nelle applicazioni. Il codice sorgente Java di esempio mostra come i client e i server possono utilizzare gli oggetti SSLContext sia sui client che sui server per creare un ambiente di comunicazione sicuro. <<




Configurare il server iSeries per supportare JSSE

Quando si utilizza J2SDK (Java 2 Software Development Kit), versione 1.4 sul server iSeries, JSSE è già configurato. La configurazione predefinita utilizza il fornitore JSSE nativo di iSeries.

Requisiti software: Per utilizzare JSSE con J2SDK, versione 1.4, è necessario disporre di IBM Cryptographic Access Provider 128-bit (5722-AC3) installato sul server iSeries. Per ulteriori informazioni, consultare Fornitori di accesso crittografico.

Modificare i fornitori JSSE: E' possibile configurare JSSE in modo che utilizzi il fornitore JSSE Java puro invece di quello nativo di iSeries. Modificando alcune proprietà della sicurezza JSSE specifiche e proprietà di sistema Java, è possibile passare da un fornitore all'altro. Per ulteriori informazioni, consultare i seguenti argomenti:

- Fornitori JSSE
- Proprietà della sicurezza JSSE
- Proprietà di sistema Java

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 SSLPermission in Permissions in the Java 2 SDK  . << >>

Fornitori JSSE

IBM JSSE include un fornitore JSSE nativo di iSeries ed uno Java puro. La scelta del fornitore da utilizzare dipende dalle esigenze dell'applicazione.

Entrambi i fornitori JSSE rispettano le specifiche dell'interfaccia JSSE. I due fornitori possono comunicare tra di loro e con qualsiasi altra implementazione SSL o TLS (perfino con implementazioni non-Java).

Fornitore JSSE Java puro: Il fornitore JSSE Java puro fornisce le seguenti funzionalità:

- Gestisce qualsiasi tipo di oggetto KeyStore per controllare e configurare i certificati digitali (ad esempio, JKS, PKCS12 e così via)
- Consente di utilizzare qualsiasi combinazione di componenti JSSE provenienti da più implementazioni contemporaneamente (ad esempio, è possibile utilizzare un X509TrustManager proveniente da un fornitore JSSE nativo di iSeries per inizializzare un SSLContext proveniente da questo fornitore JSSE Java puro).

IBMJSSE è il nome del fornitore per l'implementazione Java pura. E' 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 nativo di iSeries: Il fornitore JSSE nativo di iSeries fornisce le seguenti funzionalità:

- Utilizza il supporto SSL nativo di iSeries
- Utilizza il DCM (Digital Certificate Manager) per configurare e controllare i certificati digitali
- Fornisce migliori prestazioni

Nota: il fornitore JSSE nativo di iSeries richiede un tipo di KeyStore iSeries univoco. Il fornitore JSSE nativo di iSeries, inoltre, non consente ai componenti provenienti da altre implementazioni di collegarsi.

IbmISeriesSslProvider è il nome dell'implementazione nativa di iSeries. E' 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.

Modificare il fornitore JSSE predefinito: E' possibile cambiare il fornitore JSSE predefinito apportando le modifiche appropriate alle proprietà della sicurezza. Per ulteriori informazioni, consultare il seguente argomento:

- Proprietà della sicurezza JSSE

Una volta modificato il fornitore JSSE, assicurarsi che le proprietà di sistema specifichino la corretta configurazione per le informazioni sul certificato digitale (keystore) richiesta dal nuovo fornitore. Per ulteriori informazioni, consultare il seguente argomento:

- Proprietà di sistema Java <<



Proprietà della sicurezza JSSE

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'indirizzo /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
```

ssl.KeyManagerFactory.algorithm

Specifica l'algoritmo KeyManagerFactory predefinito. Per il fornitore JSSE nativo di iSeries, utilizzare:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

Per il fornitore JSSE Java puro, 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 Java puro, 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 Java puro, 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 Java puro, utilizzare:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

Per ulteriori informazioni, consultare javadoc per javax.net.ssl.SSLServerSocketFactory.



Proprietà di sistema JSSE Java

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 iSeries.

Quando si utilizza il fornitore JSSE nativo di iSeries, 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 che si applicano solo al fornitore JSSE nativo di iSeries: Le proprietà che seguono si applicano solo al fornitore JSSE nativo di iSeries.

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.keyStoreType
- e javax.net.ssl.keyStorePassword

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.keyStoreType
- javax.net.ssl.keyStorePassword
- os400.secureApplication

os400.certificateLabel

L'etichetta del file di chiavi che si desidera utilizzare. JSSE utilizza questa proprietà solo quando si imposta e si utilizza la proprietà os400.certificateContainer.

Informazioni aggiuntive: Per ulteriori informazioni sulle proprietà di sistema, consultare i seguenti argomenti:

- Proprietà di sistema Java per J2SDK, versione 1.4, sui server iSeries
- System Properties sul sito web Java della Sun 



Utilizzare il fornitore JSSE nativo di iSeries

Il fornitore JSSE nativo di iSeries fornisce la serie completa di classi ed interfacce JSSE. Per utilizzare il fornitore nativo di iSeries in modo efficace, fare riferimento alle informazioni che seguono:

- Valori protocollo per il metodo SSLContext.getInstance
- Implementazione KeyStore nativa
- Limitazioni di utilizzo del fornitore nativo di iSeries
- Informazioni Javadoc per SSLConfiguration

Valori protocollo per il metodo SSLContext.getInstance: La tabella che segue identifica e descrive i valori del protocollo per il metodo SSLContext.getInstance del fornitore JSSE nativo di iSeries.

Valore protocollo	Protocolli SSL supportati
SSL	SSL versione 2, SSL versione 3 e TLS versione 1
SSLv2	SSL versione 2
SSLv3	SSL versione 3

Valore protocollo	Protocolli SSL supportati
TLS	SSL versione 2, SSL versione 3 e TLS versione 1
TLSv1	TLS versione 1
SSL_TLS	SSL versione 2, SSL versione 3 e TLS versione 1

Implementazione KeyStore nativa di iSeries: Il fornitore nativo di iSeries offre un'implementazione della classe KeyStore di tipo IbmIseriesKeyStore. Tale implementazione keystore fornisce un wrapper al supporto DCM (Digital Certificate Manager). Il contenuto del keystore si basa su uno specifico identificativo dell'applicazione o file di chiavi, parola d'ordine ed etichetta. JSSE carica le voci keystore 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 keystore. Non è possibile modificare il keystore ed è necessario effettuare qualsiasi modifica alla configurazione utilizzando il DCM.

Per ulteriori informazioni sull'utilizzo del DCM, consultare il seguente argomento:

DCM (Digital Certificate Manager)

Limitazioni di utilizzo del fornitore nativo di iSeries: Per fare in modo che il fornitore JSSE nativo di iSeries funzioni, è necessario che l'applicazione JSSE utilizzi solo componenti provenienti dall'implementazione di origine. Ad esempio, l'applicazione abilitata a JSSE nativo di iSeries 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 iSeries.

Inoltre, è necessario inizializzare le implementazioni di X509KeyManager e X509TrustManager nel fornitore nativo di iSeries utilizzando un oggetto IbmIseriesKeyStore oppure un oggetto com.ibm.as400.SSLConfiguration.

Nota: le limitazioni precedentemente menzionate potrebbero cambiare nei rilasci futuri, permettendo al fornitore JSSE nativo di iSeries di accettare i componenti non nativi (ad esempio, JKS KeyStore o IbmX509 TrustManagerFactory).



Esempi: IBM JSSE (Java Secure Sockets Extension)

Gli esempi JSSE mostrano il modo in cui un client e un server possono utilizzare il fornitore JSSE nativo di iSeries per creare un contesto che permetta comunicazioni sicure.

Nota: entrambi gli esempi utilizzano il fornitore JSSE nativo di iSeries, indipendentemente dalle proprietà specificate dal file java.security.

Esempio: client SSL che utilizza un oggetto SSLContext

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 nativa di iSeries indipendentemente da quanto specificato nel file java.security.

Esempio: server SSL che utilizza un oggetto SSLContext

Il seguente programma server utilizza un oggetto SSLContext da esso inizializzato, con un file keystore precedentemente creato. Il file keystore si chiama /home/keystore.file e la parola d'ordine keystore è password.

E' necessario che il programma di esempio disponga del file keystore per poter creare un oggetto IbmIseriesKeyStore. L'oggetto KeyStore deve specificare MY_SERVER_APP come identificativo dell'applicazione.

Per creare il file keystore, è possibile utilizzare uno dei comandi che seguono:

- Da una richiesta comandi Qshell:

```
java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
-storepass password -appid MY_SERVER_APP
```

Per ulteriori informazioni sull'utilizzo dei comandi Java con Qshell, consultare il seguente argomento:

Qshell

- Da una richiesta comandi iSeries:

```
RUNJAVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
'/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
```

Il seguente esonero di responsabilità si applica a tutti gli esempi IBM JSSE:

Esonero di responsabilità per gli esempi di codice

L'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.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'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 adeguatezza a scopi specifici.



Esempio: client SSL che utilizza un oggetto SSLContext

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
////////////////////////////////////
//
// 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.
//
////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;

/**
 * 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
         * applicazione. "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", "quot");
        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

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

////////////////////////////////////
//
// Il seguente programma server utilizza un oggetto SSLContext da esso
// inizializzato, con un file keystore precedentemente creato.
//
// Il nome e la parola d'ordine keystore del file keystore sono:
// Nome file: /home/keystore.file
// Parola d'ordine: password
//
// E' necessario che il programma di esempio disponga del file keystore per poter creare un
// oggetto IbmISeriesKeyStore. L'oggetto KeyStore deve specificare MY_SERVER_APP come
// identificativo dell'applicazione.
//
// Per creare il file keystore, 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.
//
////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;

/**
 * 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", "IbmISeriesSslProvider");
    C.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
    /*
     * Richiamare un SSLServerSocketFactory dal SSLContext.
     */
    SSLServerSocketFactory sf = c.getSSLServerSocketFactory();
    /*
     * 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();
    }
}
```



Capitolo 3. JAAS (Java Authentication and Authorization Service)

JAAS (JavaTM Authentication and Authorization Service) è un'estensione standard al J2SDK (Java 2 Software Development Kit), Edizione Standard. Correntemente, J2SDK 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'API JAAS viene utilizzato dall'IBM e Sun Microsystems, Inc. come un'estensione a J2SDK, ➤ versioni 1.2 e 1.3. ◀ IBM e Sun presentano questa estensione per consentire l'associazione di un utente o di un'identità specifica al corrente sottoprocesso Java. Ciò viene effettuato utilizzando metodi `javax.security.auth.Subject` e, se si desidera, con il sottoprocesso del sistema operativo sottostante che utilizza i metodi `com.ibm.security.auth.ThreadSubject`.

➤ **Nota:** per J2SDK, versione 1.4, JAAS non è più un'estensione, ma fa parte dell'SDK di base. ◀

L'implementazione JAAS sul server iSeries è compatibile con l'implementazione di Sun Microsystems, Inc. Questa documentazione comprende gli aspetti univoci dell'implementazione iSeries. Si presume una familiarità dell'utente con tale materiale generale relativa alle estensioni JAAS. Per facilitare all'utente la gestione di essa e delle nostre informazioni su iSeries, forniamo i seguenti collegamenti.

- Il manuale API Developers Guide fornisce informazioni sull'utilizzo dell'API JAAS nello sviluppo del software.
- Il manuale Login/Authentication Module Developers Guide focalizza gli aspetti di autenticazione di JAAS.
- Specifica API JAAS contiene informazioni Javadoc su JAAS.

Selezionare uno di questi argomenti per ulteriori dettagli su come utilizzare JAAS:

- Preparare e configurare il server iSeries per JAAS
- Esempi JAAS
- Javadoc JAAS specifico del server iSeries

Preparare e configurare un server iSeries per JAAS (Java Authentication and Authorization Service)

E' necessario soddisfare i requisiti software e configurare il proprio server iSeries per utilizzare JAAS (JavaTM Authentication and Authorization Service).

Requisiti software per eseguire JAAS 1.0 su un server iSeries

Installare i seguenti programmi su licenza:

- J2SDK (Java 2 SDK), versione 1.4
- E' necessario IBM Toolbox per Java (mod 4), programma su licenza (LP - Licensed Program) (5722-JC1) per modificare l'identità del sottoprocesso OS. Esso contiene le classi `ProfileTokenCredential` necessarie per supportare la modifica dell'identità del sottoprocesso OS di iSeries e le classi di implementazione native.

Configurare il sistema

Per configurare il sistema in modo che utilizzi JAAS, seguire queste fasi:

1. Per JDK 1.2 e 1.3, aggiungere un collegamento simbolico all'indirizzario dell'estensione per il file jaas13.jar. Il programma di caricamento classi di estensioni deve di norma caricare il file JAR. Eseguire questo comando (tutto su una riga) sulla riga comandi iSeries per aggiungere il collegamento:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas13.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jaas13.jar')
```

» **Nota:** per JDK 1.4, non è necessario aggiungere un collegamento simbolico all'indirizzario dell'estensione. JAAS fa parte dell'SDK di base per questa versione. «

2. Un file predefinito login.config è fornito in \${java.home}/lib/security che 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=your login.config file
```

3. 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 file jaas13.jar richiede questo file JAR per le classi di implementazione credenziale che fanno parte di IBM Toolbox per Java. Il programma di caricamento classi dell'applicazione può inoltre caricare questo file includendolo nel CLASSPATH. Se questo file viene caricato dall'indirizzario del classpath, non aggiungere il collegamento simbolico all'indirizzario dell'estensione.

» Il collegamento simbolico del file jt400Native.jar all'indirizzario /QIBM/ProdData/Java400/jdk14/lib/ext forza tutti gli utenti JDK 1.4 sul server all'esecuzione con questa versione del file 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/jdk13/lib/ext, eseguire questo comando sulla riga comandi iSeries per aggiungere il collegamento:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400Native.jar')
```

» Per collegare il file jt400Native.jar all'indirizzario /QIBM/ProdData/Java400/jdk14/lib/ext, eseguire questo comando sulla riga comandi iSeries 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 iSeries per aggiungere il collegamento:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('/your extension directory/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: esaminare IBM Toolbox per Java per informazioni sulle classi credenziali iSeries. Fare clic su **Classi di sicurezza**. Fare clic su **Servizi di autenticazione**. Fare clic sulla classe **ProfileTokenCredential**. Fare clic su **Pacchetto**.

4. Aggiornare i file delle normative Java 2 per concedere le appropriate autorizzazioni alle attuali 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 sull'attuale ubicazione dei file JAR.

Per utilizzare con esito positivo le classi credenziali in IBM Toolbox per Java, aggiungere quanto segue al file delle normative 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";
}
```

E' 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.

Esaminare il manuale API Developers Guide per informazioni sui file delle normative Java 2.

5. Assicurarsi che i server host iSeries 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. E' possibile verificare che i server siano avviati e in esecuzione immettendo quanto segue sulla richiesta comandi iSeries:
 - 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.

Esempi di JAAS (Java Authentication and Authorization Service)

In queste informazioni, viene fornito un collegamento ad alcuni esempi JAAS (JavaTM Authentication and Authorization Service) su un server iSeries. Esistono due esempi JAAS inclusi nella documentazione, HelloWorld e SampleThreadSubjectLogin. Fare clic su questi collegamenti per il codice sorgente e le istruzioni:

- HelloWorld
- SampleThreadSubjectLogin



Capitolo 4. 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 serie di meccanismi di sicurezza basati sulla chiave segreta, la chiave pubblica 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:

- E' possibile sviluppare l'applicazione per una singola interfaccia estratta
- E' 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 chiamata 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 (JavaTM) Authentication and Authorization Service) per la creazione e l'utilizzo delle credenziali Kerberos. E' anche possibile fare in modo che JGSS effettui i controlli di autorizzazione JAAS quando si utilizzano quelle credenziali.

IBM JGSS contiene un fornitore JGSS nativo di iSeries, un fornitore JGSS Java e le versioni Java degli strumenti per la gestione delle credenziali Kerberos (kinit, ktab e klist).

Nota: il fornitore JGSS nativo di iSeries utilizza la libreria NAS (Network Authentication Services-Servizi di autenticazione di rete) di iSeries. Quando si utilizza il fornitore nativo di iSeries, è necessario utilizzare anche i programmi di utilità Kerberos nativi di iSeries. Per ulteriori informazioni, consultare Fornitori JGSS.

Per ulteriori informazioni sull'utilizzo di JGSS, consultare i seguenti argomenti:

Concetti su JGSS

Fornisce un'introduzione ai concetti su JGSS, che include una descrizione di alto livello delle operazioni di GSS-API ed una breve descrizione dei meccanismi di sicurezza.

Configurare il server per l'utilizzo di JGSS

Fornisce informazioni su come configurare il server iSeries per l'utilizzo di IBM JGSS con l'edizione standard di J2SDK (JavaTM) 2 Software Development Kit). Queste informazioni contengono istruzioni su come identificare ed impostare le autorizzazioni necessarie per utilizzare JGSS con un gestore della sicurezza.

Eeguire le applicazioni JGSS

Fornisce informazioni su come eseguire le applicazioni JGSS sui server iSeries. La documentazione comprende una spiegazione dei concetti operativi e istruzioni per l'utilizzo di JAAS.

Sviluppare le applicazioni JGSS

Fornisce informazioni sull'utilizzo di JGSS per lo sviluppo di applicazioni sicure. E' anche possibile trovare indicazioni su come generare token di trasporto, creare oggetti JGSS, stabilire il contesto ed altro ancora.

Informazioni di riferimento javadoc JGSS

Fornisce informazioni javadoc per classi e metodi nel pacchetto di api org.ietf.jgss e per le versioni Java degli strumenti per la gestione delle credenziali Kerberos (kinit, ktab e klist).

Esempi JGSS

Utilizzare i programmi di esempio per scoprire come è possibile utilizzare JGSS nelle applicazioni. La documentazione di esempio include il codice sorgente Java, le istruzioni per l'esecuzione degli esempi, i file di normative e di configurazione, ed altro ancora.

Per ulteriori informazioni sulla sicurezza Java e sul servizio di sicurezza generica (GSS), consultare questa documentazione:

-
- J2SDK Security enhancement  di Sun Microsystems, Inc. che contiene collegamenti ad ulteriori informazioni relative a GSS-API di Java
- 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 

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali. [«](#) [»](#)

Concetti su JGSS

Le operazioni JGSS sono composte da quattro livelli distinti, secondo gli standard di GSS-API (Generic Security Service Application Programming Interface):

1. Raccogliere le credenziali dei principal
2. Creare e stabilire il contesto di sicurezza tra i principal peer comunicanti
3. Scambiare i messaggi sicuri tra i peer
4. Ripulire e rilasciare le 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
- Come stabilire il contesto
- Proteggere e scambiare messaggi
- Rilasciare e ripulire risorse
- Meccanismi di sicurezza [«](#)



Principal e credenziali

L'identità assunta da un'applicazione nelle comunicazioni sicure JGSS con un peer viene detta un principal. Un principal potrebbe essere un vero utente o un servizio non presidiato. Un principal acquisisce le credenziali specifiche di un meccanismo di sicurezza come convalida dell'identità in detto meccanismo. Ad esempio, quando si utilizza il meccanismo kerberos, la credenziale del principal è nel formato di un TGT (ticket-granting ticket) emesso da KDC (Kerberos key distribution center). In un ambiente a più meccanismi, la credenziale GSS-API può contenere elementi di più credenziali e ogni elemento rappresentare una credenziale del meccanismo sottostante.

Lo standard di GSS-API non prescrive in che modo un principal acquisisce le credenziali e le implementazioni GSS-API generalmente non forniscono la procedura di acquisizione delle credenziali. Un principal ottiene le credenziali prima di utilizzare GSS-API; quest'ultimo richiede soltanto il meccanismo di sicurezza per le credenziali per conto del principal.

IBM JGSS include le versioni Java degli strumenti di gestione delle credenziali Kerberos kinit, ktab e klist. Inoltre, IBM JGSS migliora lo standard di GSS-API fornendo un'interfaccia di collegamento kerberos facoltativa che utilizza JAAS. Il fornitore JGSS Java puro supporta l'interfaccia di collegamento facoltativa; mentre il fornitore nativo di iSeries non la supporta. Per ulteriori informazioni, consultare i seguenti argomenti:

- Come ottenere credenziali kerberos
- Fornitori JGSS



Come stabilire il contesto

Dopo aver acquisito le credenziali di sicurezza, i due peer comunicanti tra loro stabiliscono un contesto di sicurezza utilizzando le loro credenziali. Sebbene i peer stabiliscano un singolo contesto unificato, ogni peer conserva la propria copia locale del contesto. Per stabilire il contesto si richiede l'inizializzazione dell'auto-autenticazione peer nei confronti nell'accettante peer. L'iniziatore potrebbe richiedere facoltativamente la reciproca autenticazione, in qual caso l'accettante autentica se stesso rispetto all'iniziatore.

Quando il contesto viene stabilito, quest'ultimo incorpora le informazioni relative allo stato (come ad esempio le chiavi crittografiche condivise) che abilitano il successivo scambio dei messaggi sicuri sequenziali tra i due peer.

Proteggere e scambiare messaggi

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



Rilasciare e ripulire risorse

Per poter liberare le risorse, un'applicazione JGSS cancella un contesto non più necessario. Sebbene l'applicazione JGSS possa accedere al 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 istanziare 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   

Configurare il proprio server iSeries per l'utilizzo di IBM JGSS

Il modo in cui viene configurato il server iSeries per l'utilizzo di JGSS dipende da quale versione di J2SDK (Java 2 Software Development Kit) viene utilizzata sul proprio server. Per ulteriori informazioni sulla configurazione del proprio server iSeries per l'utilizzo di JGSS, utilizzare i seguenti collegamenti:

- Utilizzare JGSS con J2SDK, versione 1.3
- Utilizzare JGSS con J2SDK, versione 1.4
- Configurare JGSS per utilizzare il fornitore JGSS nativo di iSeries 



Configurare il proprio server iSeries per l'utilizzo di JGSS con J2SDK, versione 1.3

Quando si utilizza J2SDK (Java 2 Software Development Kit), versione 1.3 sul proprio server iSeries, è necessario preparare e configurare il server per l'utilizzo di 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.

Configurare il proprio server per l'utilizzo di 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 iSeries, 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 normativa Java 1.3 predefinita sul server iSeries concede le autorizzazioni appropriate a JGSS. Se si decide di pianificare la creazione di un proprio file java.policy, consultare Autorizzazioni JVM per avere una lista di autorizzazioni da concedere a ibmjssprovider.jar.

Modificare i fornitori JGSS

Dopo aver configurato il proprio server per l'utilizzo di JGSS, che utilizza il fornitore Java puro, come valore predefinito, è possibile configurare JGSS per l'utilizzo del fornitore JGSS nativo di iSeries. 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:

- Fornitori JGSS
- Configurare JGSS per l'utilizzo del fornitore JGSS nativo di iSeries

Gestori della sicurezza

Se si esegue l'applicazione IBM JGSS con un gestore della sicurezza Java abilitato, consultare Utilizzare gestore della sicurezza. << >>

Configurare JGSS per utilizzare il fornitore JGSS nativo di iSeries

IBM JGSS utilizza il fornitore Java puro come valore predefinito. L'utente dispone dell'opzione per l'utilizzo del fornitore JGSS nativo di iSeries. Per ulteriori informazioni relative ai diversi fornitori, consultare Fornitori JGSS.

Requisiti software

Il fornitore JGSS nativo di iSeries deve essere in grado di accedere alle classi contenute in IBM Toolbox per Java. Per istruzioni relative alla modalità di accesso a Toolbox per Java, consultare Abilitare il fornitore JGSS nativo di iSeries per l'accesso a IBM Toolbox per Java.

Accertarsi di aver configurato il servizio di autenticazione della rete. Per ulteriori informazioni, consultare Servizio di autenticazione della rete.

Specificare il fornitore JGSS nativo di iSeries

Prima di poter utilizzare il fornitore JGSS nativo di iSeries con J2SDK, versione 1.3, accertarsi di aver configurato il server per l'utilizzo di JGSS. Per ulteriori informazioni, consultare Configurare il proprio server iSeries per l'utilizzo di JGSS con J2SDK, versione 1.3. Se si sta utilizzando J2SDK, versione 1.4, JGSS risulta già essere configurato.

Nota: nelle seguenti istruzioni, `{java.home}` indica il percorso dell'ubicazione della versione Java che si sta utilizzando sul proprio server. Ad esempio, se si utilizza J2SDK, versione 1.4, `{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 di JGSS nativo di iSeries, completare le seguenti attività:

- Aggiungere un collegamento simbolico all'indirizzario di estensione per il file JAR del fornitore nativo di iSeries
- Aggiungere il fornitore JGSS nativo di iSeries nella lista dei fornitori della sicurezza nel file java.security

Aggiungere un collegamento simbolico

Per aggiungere un collegamento simbolico all'indirizzario di estensione del file ibmjssseriesprovider.jar, sulla riga comandi iSeries, immettere il seguente comando (su un'unica riga) e premere **INVIO**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjssseriesprovider.jar')
NEWLNK('{java.home}/lib/ext/ibmjssseriesprovider.jar')
```

Dopo aver aggiunto un collegamento simbolico all'indirizzario di estensione del file ibmjssseriesprovider.jar, il programma di caricamento della classe di estensione caricherà il file JAR.

Aggiungere il fornitore nella lista dei fornitori della sicurezza

Aggiungere il fornitore nativo alla lista dei fornitori della sicurezza nel file `java.security`.

1. Aprire il file `$(java.home)/lib/security/java.security` per la modifica.
2. Reperire la lista dei fornitori della sicurezza. Tale lista 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.rsa.jca.Provider
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
```

3. Aggiungere il fornitore JGSS nativo di iSeries alla lista dei fornitori della sicurezza prima del fornitore Java originale. In altre parole, aggiungere `com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider` nella lista 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.rsa.jca.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 nella lista e `IBMJGSSProvider` la quinta voce. Inoltre, verificare che i numeri delle voci nella lista 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`. <<



Configurare il server iSeries per utilizzare JGSS con J2SDK, versione 1.4

Quando si utilizza la versione 1.4 di J2SDK (Java 2 Software Development Kit) sul proprio server iSeries, JGSS risulta essere già configurato. La configurazione predefinita utilizza il fornitore JGSS Java puro.

Modificare i fornitori JGSS

E' possibile configurare JGSS per utilizzare il fornitore JGSS nativo di iSeries invece del fornitore JGSS Java 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:

- Fornitori JGSS
- Configurare JGSS per l'utilizzo del fornitore JGSS nativo di iSeries

Gestori della sicurezza

Se si sta eseguendo l'applicazione JGSS con il gestore della sicurezza Java abilitato, consultare Utilizzare gestore della sicurezza. << >>

Fornitore JGSS

IBM JGSS include un fornitore JGSS nativo di iSeries e un fornitore JGSS Java puro. Il fornitore che si seleziona per l'utilizzo varia in base alle necessità della propria applicazione.

Il fornitore JGSS Java puro offre le seguenti caratteristiche:

- Assicura la massima compatibilità 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 JGSS nativo di iSeries offre le seguenti caratteristiche:

- Utilizza le librerie Kerberos native di iSeries
- 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 e in questo modo sono reciprocamente compatibili. In altre parole, un'applicazione che utilizza il fornitore JGSS Java puro può interagire con un'applicazione che utilizza il fornitore JGSS nativo di iSeries.

Modificare il fornitore JGSS


Nota: se il proprio server utilizza la versione 1.3 di J2SDK, prima di modificare il fornitore JGSS nativo di iSeries, accertarsi di averlo prima configurato per l'utilizzo di JGSS. Per ulteriori informazioni, consultare i seguenti argomenti:

- Configurare il proprio server iSeries per l'utilizzo di JGSS con J2SDK, versione 1.3
- Configurare JGSS per l'utilizzo del fornitore nativo di JGSS

E' possibile modificare facilmente il fornitore JGSS utilizzando una delle seguenti opzioni:

- Modificare la lista dei fornitori per la sicurezza in `${java.home}/lib/security/java.security`


Nota: `${java.home}` indica il percorso dell'ubicazione della versione Java che si sta utilizzando sul proprio server. Ad esempio, se si utilizza la versione 1.3 di J2SDK, `${java.home}` è `/QIBM/ProdData/Java400/jdk13`.

- Specificare il nome del fornitore nell'applicazione JGSS utilizzando `GSSManager.addProviderAtFront()` o `GSSManager.addProviderAtEnd()`. Per ulteriori informazioni, consultare javadoc `GSSManager`. 



Utilizzare il 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. Per ulteriori informazioni relative alle autorizzazioni necessarie per l'utilizzo di JGSS, visualizzare i seguenti argomenti:

- Autorizzazioni JVM
- Controlli autorizzazioni JAAS 



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.

Per ulteriori informazioni sull'utilizzo delle autorizzazioni JVM, consultare [Permissions in Java 2 SDK](#) .

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" <<



Controlli delle 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. E' 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 normativa 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 normativa 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: utilizzare il 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\"";
```

Per ulteriori informazioni, consultare la classe `javax.security.auth.kerberos.DelegationPermission` nella documentazione J2SDK  sul sito web della Sun.

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: utilizzare il 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";
```

Per ulteriori informazioni, consultare la classe `javax.security.auth.kerberos.ServicePermission` nella documentazione J2SDK  sul sito web della Sun.  

Eseguire le 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

Per ulteriori informazioni sull'esecuzione delle applicazioni JGSS, consultare i seguenti argomenti:

Ottenere le credenziali Kerberos

Fornisce informazioni su come ottenere le credenziali Kerberos e creare le chiavi segrete. Fornisce istruzioni su come utilizzare JAAS per effettuare i collegamenti a Kerberos e i controlli delle autorizzazioni; fornisce un elenco delle autorizzazioni JAAS richieste dalla JVM (Java virtual machine).

File di configurazione e delle normative

Fornisce informazioni sulle diverse modalità di supporto dei file necessari per eseguire JGSS, compresi i file di configurazione, i file delle normative, il file delle proprietà della sicurezza principale Java e la cache delle credenziali.

Effettuare il debug

Fornisce informazioni sull'utilizzo della funzione di debug di JGSS per categorizzare e visualizzare utili messaggi di debug.

Esempi JGSS

Utilizzare i programmi di esempio per effettuare controlli e verifiche della configurazione di JGSS. La documentazione di esempio include il codice sorgente Java, le istruzioni per l'esecuzione degli esempi, i file di normative e di configurazione, ed altro ancora. <<



Ottenere le credenziali Kerberos e creare le 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:

- Strumenti Kinit e Ktab
- Interfaccia di collegamento a JAAS Kerberos facoltativa <<



Gli 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.

Utilizzare il 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.

- E' 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. Per ulteriori informazioni, consultare:

- Javadoc Kinit
- Javadoc Ktab
- Interfaccia di collegamento a JAAS.

Utilizzare il fornitore JGSS nativo di iSeries

Se si utilizza il fornitore JGSS nativo di iSeries, utilizzare i programmi di utilità kinit e klist della Qshell. Per ulteriori informazioni, consultare Programmi di utilità per credenziali Kerberos e tabelle di chiavi. << >>

Interfaccia di collegamento a JAAS Kerberos

IBM JGSS fornisce un'interfaccia di collegamento a JAAS (Java Authentication and Authorizaiton Service) Kerberos. E' possibile disabilitare questa funzione impostando la proprietà Java `javax.security.auth.useSubjectCredsOnly` su `false`.

Nota: sebbene il fornitore JGSS Java puro possa utilizzare l'interfaccia di collegamento, il fornitore JGSS nativo di iSeries non può.

Per ulteriori informazioni su JAAS, consultare JAAS (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. Tener 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
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 dell'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

Opzione	credsType initiator	credsType acceptor	credsType both	forward	proxy	use Ccache	use Keytab	useDefault Ccache	useDefault Keytab
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

E' 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. Per ulteriori informazioni, consultare Richiedere nome principal e parola d'ordine.

Formato del nome principal del servizio

E' 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. E' 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)

Richiedere 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` (entrami, 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. E' 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 normative e di configurazione

JGSS e JAAS dipendono da una serie di file di configurazione e delle normative. E' 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 normative JAAS.

- File di configurazione Kerberos
- File di configurazione JAAS
- File delle normative di autorizzazione JAAS
- File proprietà di sicurezza principale Java
- Tabella chiave del server e cache delle credenziali

Nota: nelle seguenti istruzioni, `{java.home}` indica il percorso all'ubicazione della versione Java che si sta utilizzando sul proprio server. Ad esempio, se si utilizza J2SDK, versione 1.4, `{java.home}` è `/QIBM/ProdData/Java400/jdk14`. Ricordarsi di sostituire `{java.home}` nelle impostazioni delle proprietà con il percorso corrispondente all'indirizzario principale Java.

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 al quale si fa riferimento nella proprietà Java `java.security.krb5.conf`
2. `{java.home}/lib/security/krb5.conf`
3. `c:\winnt\krb5.ini` su piattaforme Microsoft Windows^(R)
4. `/etc/krb5/krb5.conf` su piattaforme Solaris^(TM)
5. `/etc/krb5.conf` su piattaforme Unix^(R)

File di configurazione JAAS

L'utilizzo della funzione di collegamento JAAS richiede un file di configurazione JAAS. E' 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 JAAS (Sun Java Authentication and Authorization Service) .

File normativa JAAS

Quando si utilizza l'implementazione della normativa predefinita, JGSS concede le autorizzazioni JAAS alle entità mediante registrazione delle autorizzazioni in un file delle normative. E' possibile specificare il file delle normative JAAS mediante impostazione di una delle seguenti proprietà:

- La proprietà di sistema Java `java.security.policy`
- La proprietà di sicurezza `policy.url.<numero intero>` nel file `#{java.home}/lib/security/java.security`

Se si sta utilizzando J2SDK, versione 1.4, è facoltativo specificare un file delle normative JAAS separato. Il fornitore della normativa predefinito in J2SDK, versione 1.4 supporta le voci del file delle normative richieste da JAAS.

Per ulteriori informazioni, consultare il sito Web di Sun JAAS (Java Authentication and Authorization Service)  .

File delle proprietà di sicurezza principale Java

Una JVM (Java virtual machine) utilizza molte proprietà di sicurezza importanti che l'utente imposta mediante modifica del file delle proprietà di sicurezza principale Java. Questo file, denominato `java.security`, in genere risiede nell'indirizzario `#{java.home}/lib/security` del proprio server iSeries.

La seguente lista descrive diverse proprietà di sicurezza rilevanti per l'utilizzo di JGSS. Utilizzare le descrizioni come guida per la modifica del 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. Inoltre registra 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 normative di sistema. Ad esempio:

```
policy.provider=sun.security.provider.PolicyFile
```

policy.url.<numero intero>: URL dei file delle normative. Per utilizzare il file delle normative 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 normative 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 normative JAAS. E' possibile includere sia strutture basate su CodeSource che quelle basate sui principal, nel file delle normative JAAS. Per utilizzare il file delle normative 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^(TM) 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 (se \${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^(TM) KRB5_KTNAME
2. La voce default_keytab_name nella stanza libdefaults del file di configurazione Kerberos
3. \${user.home}/krb5_keytab <<



Sviluppare le applicazioni IBM JGSS

Per ulteriori informazioni sullo sviluppo delle applicazioni IBM JGSS, consultare i seguenti argomenti:

Istruzioni di programmazione

Per conoscere le fasi necessarie per sviluppare un'applicazione JGSS ed per utilizzare dei token di trasporto, creare gli oggetti JGSS necessari, stabilire e cancellare il contesto e utilizzare i servizi tramite messaggio.

Utilizzare JAAS con l'applicazione JGSS

Per leggere le informazioni relative all'abilitazione della funzione di collegamento kerberos JAAS di JGSS. In tali informazioni vengono inclusi i requisiti per l'utilizzo della funzione di collegamento e un frammento del codice di esempio.

Eeguire il debug

Per leggere le informazioni relative all'utilizzo del debug JGSS per categorizzare e visualizzare utili messaggi di debug.



Informazioni di riferimento javadoc JGSS

Per ricontrollare le informazioni relative a javadoc delle classi e dei metodi nel pacchetto di api org.ietf.jgss e delle versioni Java degli strumenti di gestione delle credenziali Kerberos (kinit, ktab e klist).

Esempi JGSS

Utilizzare i programmi di esempio per conoscere le modalità di utilizzo di JGSS nelle applicazioni. La documentazione di esempio include il codice sorgente Java, le istruzioni per l'esecuzione degli esempi, i file di normative e di configurazione, 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

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 sottoforma 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.

1. Creare un `GSSManager`
Un'istanza di `GSSManager` agisce come produttore per la creazione di altre istanze oggetto JGSS.
2. Creare un `GSSName`
`GSSName` rappresenta l'identità di un principal JGSS. Alcune operazioni JGSS possono localizzare ed utilizzare un principal predefinito quando si specifica un valore null per `GSSName`.
3. Creare un `GSSCredential`
`GSSCredential` incorpora le credenziali specifiche del meccanismo del principal.
4. Creare un `GSSContext`
`GSSContext` viene utilizzato per stabilire un contesto e successivi servizi tramite messaggio.
5. Selezionare servizi facoltativi sul contesto
L'applicazione deve richiedere esplicitamente servizi facoltativi, come l'autenticazione reciproca.
6. Come stabilire il contesto
L'iniziatore autentica se stesso nei confronti dell'accettante. Tuttavia, quando si richiede una reciproca autenticazione, l'accettante a sua volta autentica se stesso nei confronti dell'iniziatore.
7. Utilizzare servizi tramite messaggio
L'iniziatore e l'accettante scambiano messaggi sicuri nel contesto stabilito.
8. Cancellare il contesto
L'applicazione cancella un contesto che non è più richiesto. <<



Creare un `GSSManager`

Con la classe astratta `GSSManager` è possibile creare i seguenti oggetti JGSS:

- `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();
```




Creare 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).

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.

Esempi: utilizzare 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. E' 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);
```



Creare 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, quindi è il meccanismo predefinito.

La propria 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: ottenere le 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);
```



Creare GSSContext

IBM JGSS supporta due metodi forniti da GSSManager per la creazione di un contesto:

- metodo utilizzato dall'iniziatore del contesto
- 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 dell'iniziatore per l'accettazione del contesto, né utilizzare un contesto dell'accettante come contesto di inizio. 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: utilizzare 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 la propria applicazione può creare e simultaneamente utilizzare entrambi i tipi di contesti. <<



Richiedere servizi di sicurezza facoltativi


La propria applicazione può richiedere una serie di servizi di sicurezza facoltativi. IBM JGSS supporta i seguenti servizi facoltativi:

- 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: richiedere 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);
```



Come stabilire il contesto

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: 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());

```



Utilizzare i servizi per messaggi

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.

Inviare messaggi

IBM JGSS fornisce due gruppi di metodi per rendere sicuri i messaggi: `wrap()` e `getMIC()`.

Utilizzare `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).

Utilizzare `getMIC()`

Il metodo `getMIC` esegue le seguenti operazioni ma non può 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: utilizzare i 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
```

Ricevere i 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 [Simple Authentication and Security Layer \(SASL\)](#) .

Esempio: utilizzare 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 del messaggio applicate da
// superSecureServer. In particolare, può asserire che il messaggio non è stato
// codificato dal momento che getMIC non può mai codificare un messaggio.
```



Cancellare il contesto

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. <>

Utilizzare 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 principali 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 reperire le credenziali e le chiavi segrete dal soggetto. E' possibile disabilitare questa funzione impostando la proprietà Java `javax.security.auth.useSubjectCredsOnly` su `false`.

Nota: mentre il fornitore JGSS Java puro può utilizzare l'interfaccia di collegamento, il fornitore JGSS nativo di iSeries non può farlo.

Per ulteriori informazioni sulle caratteristiche di JAAS, consultare [Come ottenere le credenziali Kerberos e le chiavi segrete](#).

Per utilizzare la funzione di collegamento JAAS, la propria 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 Krb5CallbackHandler());
    loginContext.login();

    // Eseguire l'intera applicazione JGSS nella modalità privilegiata JAAS
    Subject.doAsPrivileged(loginContext.getSubject(),
        new JGSSOperations(), null);
}
```



Eseguire il debug

Quando si tenta di identificare i problemi JGSS, utilizzare la funzione di debug di JGSS per produrre utili messaggi categorizzati. E' possibile attivare una o più categorie impostando i valori appropriati della proprietà Java `com.ibm.security.jgss.debug`. E' 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
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 (app)

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 String.
```



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 normative e informazioni di riferimento javadoc.

E' 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 normative e compilare ed eseguire i programmi di esempio:

- Visualizzare le versioni HTML degli esempi
- Scaricare e visualizzare le informazioni javadoc di esempio
- Scaricare 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. E' 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 normative, le opzioni di debug di JGSS e il gestore della sicurezza. E' anche possibile attivare e disattivare le funzioni JAAS.

E' 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 principale. La configurazione a due server consiste nel disporre di un server principale ed uno secondario, il server principale 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 principale. Successivamente, il client delega le proprie credenziali al server principale. Ora è il server principale che utilizza, al posto del client, queste credenziali per iniziare un contesto e scambiare messaggi con il server secondario. E' anche possibile utilizzare una configurazione a due server in cui il server principale agisca come un client. In tal caso, il server principale utilizza le proprie credenziali per iniziare un contesto e scambiare messaggi protetti con il server secondario.

Può essere in esecuzione simultaneamente sul server principale, 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.



Visualizzare gli esempi IBM JGSS

I file di esempio IBM JGSS (Java Generic Security Service) includono programmi client e server, file di configurazione, file di normative e informazioni di riferimento javadoc. Utilizzare i collegamenti che seguono per visualizzare le versioni HTML degli esempi JGSS.

Per ulteriori informazioni, consultare i seguenti argomenti:

- Descrizione dei programmi di esempio

- Scaricare ed eseguire i programmi di esempio

Visualizzare i programmi di esempio

Visualizzare le versioni HTML dei programmi di esempio JGSS utilizzando i collegamenti che seguono:

- Esempio: programma client non-JAAS
- Esempio: programma server non-JAAS
- Esempio: programma client abilitato a JAAS
- Esempio: programma server abilitato a JAAS

Visualizzare i file di configurazione e delle normative

Visualizzare le versioni HTML dei file di configurazione e delle normative JGSS utilizzando i collegamenti che seguono:

- File di configurazione Kerberos
- File di configurazione JAAS
- File delle normative JAAS
- File delle normative Java



Esempio: programma client IBM JGSS non-JAAS

Per ulteriori informazioni sull'utilizzo del programma client di esempio, consultare Scaricare ed eseguire i programmi di esempio.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
// 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.
 */
```



```

        throw new GSSEException(GSSEException.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()); // XXXXXXXX
        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

Per ulteriori informazioni sull'utilizzo del programma server di esempio, consultare [Scaricare ed eseguire gli esempi IBM JGSS](#).

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

// 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.
 *
 * E' sempre in ascolto delle connessioni client
 * e genera un sottoprocesso per servire una connessione in entrata.
 * E' 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.
 *
 * E' 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 (principale)
* che ha avviato i contatti con esso (il server secondario).
* Durante le comunicazioni con il server secondario, il server principale 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 utilizzando
    * le credenziali proprie del server. Ciò assicura che il server sia in esecuzione come iniziatore
    * solo una volta. L'interrogazione 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]"

```



```

        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 GSSEException
{
    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)
        ipp = 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)
        ipp = p;
}

init(primary, // primo o secondo server
    name, // nome
    ipp, // 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 {
do {
        ssocket = new ServerSocket(myPort);

        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 utilizza 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()); // XXXXXXXX
    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

Per ulteriori informazioni sull'utilizzo del programma client di esempio, consultare Scaricare ed eseguire gli esempi IBM JGSS.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

// 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 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 a JAAS

Per ulteriori informazioni sull'utilizzo del programma server di esempio, consultare Scaricare ed eseguire gli esempi IBM JGSS.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

// 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()); // XXXXXXXX

        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;
    }
}

```



Esempio: file di configurazione Kerberos

Per ulteriori informazioni sull'utilizzo del file di configurazione di esempio, consultare Scaricare ed eseguire gli esempi IBM JGSS.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

# -----
# 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_encypes  = des-cbc-crc
default_tgs_encypes  = 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

Per ulteriori informazioni sull'utilizzo del file di configurazione di esempio, consultare Scaricare ed eseguire gli esempi IBM JGSS.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

/**
 * -----
 * 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.
 * L'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)
*   proxiabile=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 normative JAAS

Per ulteriori informazioni sull'utilizzo del file delle normative di esempio, consultare [Scaricare ed eseguire gli esempi IBM JGSS](#).

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

// -----
// File delle normative 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 normative o il suo
// contenuto per proteggere risorse in un ambiente di produzione.
//
// Esonero di responsabilità per gli esempi di codice
// L'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 dall'IBM solo a scopo illustrativo.
// Questi esempi non sono stati interamente testati in tutte le condizioni.
// L'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 normative Java

Per ulteriori informazioni sull'utilizzo del file delle normative di esempio, consultare Scaricare ed eseguire gli esempi IBM JGSS.

Nota: leggere l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

// -----
// File delle normative Java per l'esecuzione delle applicazioni di esempio di JGSS sul
// server iSeries.
// 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 normative o il suo

```

```

// contenuto per proteggere risorse in un ambiente di produzione.
//
// Esonero di responsabilità per gli esempi di codice
// L'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 dall'IBM solo a scopo illustrativo.
// Questi esempi non sono stati interamente testati in tutte le condizioni.
// L'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 1.3
    permission javax.security.auth.AuthPermission "createLoginContext";

    // 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";

    // E' 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: scaricare e visualizzare 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.

Esonero di responsabilità per gli esempi di codice

L'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.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'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 adeguatezza a scopi specifici. << >>

Esempi: scaricare ed eseguire i programmi di esempio

Prima di modificare o eseguire gli esempi, leggere la descrizione dei programmi di esempio.

Per eseguire i programmi di esempio, effettuare quanto segue:

1. scaricare i file di esempio del server iSeries
2. preparare l'esecuzione dei file di esempio
3. eseguire i programmi di esempio

Per ulteriori informazioni sulla modalità di esecuzione di un esempio, consultare Esempio: eseguire un esempio non JAAS.

Esonero di responsabilità per gli esempi di codice

L'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.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'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 adeguatezza a scopi specifici. << >>

Esempi: scaricare gli esempi IBM JGSS

Prima di modificare o eseguire gli esempi, leggere la descrizione dei programmi di esempio.

Per scaricare i file di esempio e memorizzarli sul proprio server iSeries, completare quanto segue:

1. Sul proprio server iSeries, scegliere un indirizzario esistente (o crearne uno nuovo) dove si desidera memorizzare i programmi di esempio, i file di configurazione e i file delle normative.
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 ibmjgsssample.jar, che contiene il file .class di esempio, nell'indirizzario selezionato
- creare un sottoindirizzario (denominato config) che contiene i file di configurazione e delle normative.
- creare un sottoindirizzario (denominato src) che contiene un esempio dei file origine .java.

Informazioni correlate

E' possibile leggere le attività correlate o consultare un esempio:

- preparare l'esecuzione dei file di esempio
- eseguire i programmi di esempio
- Esempio: eseguire l'esempio non JAAS<<



Esempi: preparare l'esecuzione dei programmi di esempio

Prima di modificare o eseguire gli esempi, leggere la descrizione dei programmi di esempio.

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 normative al fine di adattarli al proprio ambiente. Per ulteriori informazioni, fare riferimento ai commenti presenti in ogni file di configurazione e delle normative.
- Assicurarsi che il file java.security file contenga le impostazioni corrette del proprio server iSeries. Per ulteriori informazioni, consultare File delle proprietà di sicurezza principali Java.
- Inserire il file di configurazione Kerberos modificato (krb5.conf) nell'indirizzario del server iSeries appropriato per la versione di J2SDK che si sta utilizzando:
 - per la versione 1.3 di J2SDK: /QIBM/ProdData/Java400/jdk13/lib/security
 - per la versione 1.4 di J2SDK: /QIBM/ProdData/Java400/jdk14/lib/security

Informazioni correlate

E' possibile leggere le attività correlate o esaminare un esempio:

- scaricare i file di esempio del server iSeries
- eseguire i programmi di esempio
- Esempio: eseguire l'esempio non JAAS<<



Esempi: eseguire i programmi di esempio

Prima di modificare o eseguire gli esempi, leggere la descrizione dei programmi di esempio.

Dopo aver scaricato e modificato il codice sorgente, è possibile eseguire uno dei seguenti esempi.

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 sola o più proprietà Java facoltative, compresi i nomi dei file di configurazione e delle normative, 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 una lista 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

E' possibile leggere le attività correlate o consultare un esempio:

- scaricare i file di esempio del server iSeries
- preparare l'esecuzione dei file di esempio
- Esempio: eseguire l'esempio non JAAS



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 nelle 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`
- `kinit`
- `ktab`
- `klist`

Capitolo 5. Ottimizzare le prestazioni del programma Java con IBM Developer Kit per Java

E' opportuno prendere in considerazione alcuni aspetti delle prestazioni dell'applicazione Java^(TM) quando si crea un'applicazione Java per il proprio server iSeries. Seguono alcuni collegamenti ai dettagli e ai suggerimenti su come è possibile ottenere delle prestazioni migliori:

- Utilizzare il comando CRTJVAPGM (Creazione programma Java) prima di eseguire i file di classe Java, JAR o ZIP per migliorare le prestazioni del tempo di esecuzione del proprio codice Java.
- Modificare i propri livelli di ottimizzazione per ottenere le migliori prestazioni della compilazione statica.
- Impostare con attenzione i propri valori per ottenere le prestazioni ottimali di raccolta di dati inutili.
- Utilizzare soltanto i metodi nativi per avviare le funzioni del sistema relativamente lunghe da eseguire e non disponibili direttamente in Java.
- Utilizzare l'opzione javac -o in fase di compilazione per eseguire un allineamento del metodo e migliorare in modo significativo le prestazioni della chiamata al metodo.
- Utilizzare le eccezioni Java in casi diversi dal normale flusso attraverso la propria applicazione.

Utilizzare questi strumenti con PEX (Performance Explorer) per localizzare i problemi delle prestazioni nei propri programmi Java:

- E' possibile raccogliere gli eventi della traccia Java utilizzando la Java virtual machine di iSeries.
- Per stabilire il tempo impiegato in ogni metodo Java, utilizzare le tracce di chiamata Java.
- Il Modellamento Java localizza la quantità di memoria relativa del tempo CPU impiegato in ogni metodo Java e tutte le funzioni del sistema utilizzate dal proprio programma Java.
- Utilizzare il PDC (Performance Data Collector Java per fornire informazioni a relative al profilo dei programmi che vengono eseguiti sul server iSeries.

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'applicazione Java. 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.

In aggiunta ai dati sulle prestazioni al livello dell'applicazione, è possibile utilizzare gli strumenti delle prestazioni al livello del sistema iSeries esistenti. Tali strumenti riportano la statistica su una base del sottoprocesso Java.

Per esempi di prospetti PEX, consultare il manuale Performance Tools for iSeries, SC41-5340. 

Considerazioni sulle prestazioni di Java runtime

Per migliorare in modo significativo le prestazioni del tempo di esecuzione del codice Java^(TM), utilizzare il comando CRTJVAPGM (Creazione programma Java) prima di eseguire i file di classe, i file JAR o i file ZIP. Il comando CRTJVAPGM utilizza i bytecode per creare un programma Java che contiene istruzioni native ottimizzate per il server iSeries e associa l'oggetto del programma Java al file di classe, al file JAR o al file ZIP.

Se il comando CRTJVAPGM non viene utilizzato prima di eseguire un file di classe Java, un file JAR o un file ZIP, allora il codice Java viene eseguito più lentamente la prima volta poiché viene creato un programma Java ottimizzato al livello di ottimizzazione 10. Le esecuzioni successive sono molto più veloci

perché il programma Java è stato salvato e rimane associato al file di classe o al file JAR. E' possibile che l'esecuzione dei bytecode in modo interpretativo fornisca prestazioni accettabili durante lo sviluppo dell'applicazione, ma è possibile utilizzare anche il comando CRTJVAPGM prima di eseguire il codice Java in un ambiente di sviluppo.

Un compilatore Just-In-Time migliora le prestazioni tramite la compilazione del codice, ottimizzato su JRE di una JVM specifica. E' possibile valutare i vantaggi del non dover utilizzare il comando CRTJVAPGM rispetto ad un avvio del programma e ad un'elaborazione del tempo di esecuzione leggermente più lenti conseguenti all'utilizzo di JIT, per decidere quale tecnica di ottimizzazione si adatti meglio alle proprie esigenze.

Se l'esecuzione del programma è lenta, immettere il comando DSPJVAPGM (Visualizzazione programma Java) per visualizzare gli attributi di un programma Java.



Programmi di caricamento classi nella cache

E' possibile migliorare le prestazioni dei programmi di caricamento classi utente consentendo ai JVAPGM creati da tali programmi di essere memorizzati nella cache per il riutilizzo. Questa opzione non viene abilitata a meno che non si specifichi una proprietà Java di abilitazione:

- /QIBM/UserData/Java400/SystemDefault.properties
- /home//SystemDefault.properties
- Sul comando CL RUNJVA
- Sul comando JAVA QSH

La proprietà seguente viene utilizzata per abilitare questa funzione:

os400.define.class.cache.file

Il valore della proprietà deve corrispondere al nome (con il percorso completo) di un file Jar (Java ARchive) valido. E' necessario che il file JAR contenga un indirizzario JAR valido (come quello creato dal comando jar QSH), ma non deve presentare altri contenuti all'infuori del singolo membro richiesto per far funzionare il comando jar. E' necessario che questo file JAR non sia incluso in alcun classpath Java.

Ad esempio, aggiungere la riga seguente a /QIBM/UserData/Java400/SystemDefault.properties:

```
os400.define.class.cache.file=/QIBM/ProdData/Java400/QDefineClassCache.jar
```

Developer Kit per Java installa un file JAR adatto come /QIBM/ProdData/Java400/QDefineClassCache.jar e installa un file di proprietà dell'esempio come /QIBM/ProdData/Java400/SystemDefaultCacheExample.properties. E' possibile abilitare la funzione di memorizzazione nella cache globalmente copiando il file SystemDefaultCacheExample.properties su QIBM/UserData/Java400/ e ridenominandolo su SystemDefault.properties.

Nota: assicurarsi che non esista alcun file SystemDefault.properties in quell'indirizzario prima di apportare questa modifica. In alternativa, per influire solo su un ambiente di un singolo utente, è possibile copiare il file su /home// e ridenominarlo in modo simile.

E' possibile utilizzare DSPJVAPGM sul file JAR specificato per la memorizzazione nella cache in modo tale da determinare quanti JVAPGM sono memorizzati nella cache. Il campo **programmi Java** del pannello DSPJVAPGM indica quanti JVAPGM sono memorizzati nella cache e il campo **dimensione programma Java** indica quanta memoria viene consumata dai JVAPGM memorizzati. Altri campi del pannello risultano inutili quando viene applicato DSPJVAPGM a un file JAR utilizzato per la memorizzazione nella cache.

In aggiunta alla proprietà Java `os400.define.class.cache.file` che abilita la memorizzazione nella cache, è possibile specificare altre due proprietà per controllare le caratteristiche della memorizzazione nella cache.

os400.define.class.cache.hours

E' possibile utilizzare questa proprietà per specificare il tempo (espresso in ore) durante il quale un JVAPGM può rimanere nella memoria cache.

os400.define.class.cache.maxpgms

E' possibile utilizzare questa proprietà per specificare il numero massimo di JVAPGM che possono essere memorizzati nella cache, con la cancellazione in primo luogo dei JVAPGM più vecchi quando questo limite viene superato. Il valore di tempo utilizzato per determinare l'età di un JVAPGM memorizzato nella cache viene aggiornato ogni volta che viene fatto riferimento al JVAPGM.

Esempi:

```
os400.define.class.cache.hours=48
os400.define.class.cache.maxpgms=10000
```

Il valore predefinito per `os400.define.class.cache.hours` è 168 ore (una settimana) e il valore massimo è 9999. Il valore predefinito per `os400.define.class.cache.maxpgms` è 5000 e il valore massimo è 40000. Se un valore pari a zero è specificato per una di queste proprietà o se non è possibile analizzare il valore come un numero decimale valido, allora si utilizza il valore predefinito.

Consultare Proprietà dei sistemi per ulteriori informazioni sulle proprietà precedentemente menzionate per la memorizzazione nella cache. [◀](#)

Selezionare la modalità da utilizzare durante l'esecuzione di un programma Java

Quando si esegue un programma Java^(TM), 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. E' possibile utilizzare una delle seguenti modalità:

- Interpretato
- Elaborazione diretta
- Compilazione JIT (Just-In-Time)
- Compilazione JIT (Just-In-Time) ed elaborazione diretta

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).
Elaborazione diretta	Vengono create istruzioni macchina relative a un metodo durante la prima chiamata a quel metodo e vengono salvate per l'utilizzo alla successiva esecuzione di tale programma. Viene anche condivisa una copia per l'intero sistema. Per informazioni sull'esecuzione del programma Java utilizzando l'elaborazione diretta, consultare il comando RUNJVA (Esecuzione Java).

Modalità di selezione	Dettagli
<p>Compilazione JIT (Just-In-Time)</p>	<p>Vengono create istruzioni macchina relative ad un metodo durante la prima chiamata a quel metodo e vengono salvate per la durata dell'esecuzione della Java virtual machine.</p> <p>Per utilizzare il compilatore Just-In-Time, è necessario impostare il valore del compilatore su <code>jitc</code>. E' possibile impostare il valore aggiungendo una variabile di ambiente o impostando la proprietà di sistema <code>java.compiler</code>. Selezionare un metodo dalla lista riportata di seguito per impostare il valore del compilatore:</p> <ul style="list-style-type: none"> • Da una richiesta della riga comandi sul server iSeries, aggiungere la variabile di ambiente utilizzando il comando <code>ADDENVVAR</code> (Aggiunta variabile ambiente). Successivamente eseguire il programma Java utilizzando il comando <code>RUNJVA</code> (Esecuzione Java) o il comando <code>JAVA</code>. Ad esempio, utilizzare: <pre>ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc) JAVA CLASS(Test)</pre> • Impostare la proprietà di sistema <code>java.compiler</code> sulla riga comandi iSeries. Ad esempio, immettere <code>JAVA CLASS(Test) PROP((java.compiler jitc))</code> • Impostare la proprietà di sistema <code>java.compiler</code> sulla riga comandi di Qshell Interpreter. Ad esempio, immettere <code>java -Djava.compiler=jitc Test</code> <p>Una volta che questo valore è stato impostato, il compilatore JIT ottimizza tutto il codice Java prima dell'esecuzione.</p>

Modalità di selezione	Dettagli
Compilazione JIT (Just-In-Time) ed elaborazione diretta	<p>La maniera più diffusa di utilizzare il compilatore JIT (Just-In-Time) è con l'opzione <code>jit_de</code>. Durante l'esecuzione con questa opzione, i programmi già ottimizzati con l'elaborazione diretta vengono eseguiti in modalità di elaborazione diretta. I programmi non ottimizzati per l'ottimizzazione diretta vengono eseguiti in modalità JIT.</p> <p>Per utilizzare JIT e l'elaborazione diretta, è necessario impostare il valore del compilatore su <code>jitc_de</code>. E' possibile impostare il valore aggiungendo una variabile di ambiente o impostando la proprietà di sistema <code>java.compiler</code>. Selezionare un metodo dalla lista seguente per impostare il valore del compilatore:</p> <ul style="list-style-type: none"> • Aggiungere la variabile di ambiente tramite l'immissione del comando <code>ADDENVVAR</code> (Aggiunta variabile di ambiente) sulla riga comandi iSeries. Successivamente eseguire il programma Java utilizzando il comando <code>RUNJAVA</code> (Esecuzione Java) o il comando <code>JAVA</code>. Ad esempio, immettere <pre>ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc_de) JAVA CLASS(Test)</pre> • Impostare la proprietà di sistema <code>java.compiler</code> sulla riga comandi iSeries. Ad esempio, immettere <code>JAVA CLASS(Test) PROP((java.compiler jitc_de))</code> • Impostare la proprietà di sistema <code>java.compiler</code> sulla riga comandi di Qshell Interpreter. Ad esempio, immettere <code>java -Djava.compiler=jitc_de Test</code> <p>Una volta impostato tale valore, si utilizza il programma Java relativo al file di classe creato come elaborazione diretta. Se il programma Java non è stato creato come elaborazione diretta, il file di classe viene ottimizzato da JIT prima dell'esecuzione. Per ulteriori informazioni, consultare Confronto tra il compilatore JIT (Just-In-Time) e l'elaborazione diretta</p>

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	<code>INTERPRET(*YES)</code>	<code>-Djava.compiler=NONE</code> <code>-interpret</code>	<code>os400.run.mode="interpret"</code>
DE	<code>INTERPRET(*NO)</code>	<code>-Djava.compiler=NONE</code>	<ul style="list-style-type: none"> • <code>os400.run.mode="program_created=pc"</code> • <code>os400.create.type="direct"</code>
JIT	<code>INTERPRET(*JIT)</code>	<code>-Djava.compiler="jitc"</code>	<code>os400.run.mode="jitc"</code>
JIT_DE(default)	<code>INTERPRET(*OPTIMIZE)</code> <code>OPTIMIZE(*JIT)</code>	<code>-Djava.compiler="jitc_de"</code>	<code>os400.run.mode="jitc_de"</code>

Interprete Java

L'interprete di Java^(TM) è quella parte della JVM (Java virtual machine) che interpreta il file di classe Java di una piattaforma hardware. L'interprete Java decodifica ogni bytecode ed esegue una serie di istruzioni macchina per quel bytecode.

Compilazione statica

Il programma di conversione Java^(TM) è un componente IBM OS/400 (Operating System/400) che preelabora i file di classe per prepararli all'esecuzione utilizzando la Java virtual machine iSeries. Il programma di conversione Java crea un programma ottimizzato persistente e associato al file di classe. Nel caso predefinito, l'oggetto del programma contiene una versione compilata di istruzioni macchina RISC e 64-bit, relativa alla classe. L'interprete Java non interpreta l'oggetto programma ottimizzato durante il tempo di esecuzione. Al contrario, questo viene eseguito direttamente quando si carica il file di classe.

I programmi Java sono ottimizzati utilizzando JIT per impostazione predefinita. Per utilizzare il programma di conversione Java, eseguire il comando CRTJVAPGM o specificare l'utilizzo del programma di conversione sul comando RUNJVA o JAVA.

E' possibile utilizzare il Comando CRTJVAPGM (Creazione programma Java) per avviare in modo esplicito il programma di conversione Java. Il comando CRTJVAPGM ottimizza il file di classe o il file JAR durante la sua esecuzione, per cui non occorre eseguire alcuna operazione mentre il programma è in esecuzione. Questo migliora la velocità del programma la prima volta che viene eseguito. L'utilizzo del comando CRTJVAPGM, invece di affidarsi all'ottimizzazione predefinita, assicura la migliore ottimizzazione possibile e inoltre migliora l'utilizzo dello spazio relativo ai programmi Java associati al file di classe o al file JAR.

L'utilizzo del comando CRTJVAPGM su un file di classe, un file JAR o un file ZIP determina che tutte le classi presenti nel file siano ottimizzate e che il programma Java risultante sia persistente. In questo modo si ottengono migliori prestazioni del tempo di esecuzione. E' possibile inoltre modificare il livello di ottimizzazione o selezionare un livello di ottimizzazione diverso da quello predefinito pari a 10 utilizzando il comando CRTJVAPGM o il comando CHGJVAPGM (Modifica programma Java). Al livello di ottimizzazione 40, il collegamento interclasse si esegue tra le classi all'interno di un file JAR e in alcuni casi le classi vengono allineate. Il collegamento interclasse migliora la velocità di chiamata. L'allineamento elimina interamente il sovraccarico di una chiamata del metodo. In alcuni casi, è possibile allineare i metodi tra le classi all'interno del file JAR o del file ZIP. Specificando OPTIMIZE(*INTERPRET) sul comando CRTJVAPGM si fa in modo che ogni classe specificata sul comando sia verificata e preparata per l'esecuzione in modalità interpretato.

Il comando RUNJVA (Esecuzione Java) può inoltre specificare OPTIMIZE(*INTERPRET). Questo parametro specifica il fatto che ogni classe in esecuzione nell'ambito della Java virtual machine venga interpretata, indipendentemente dal livello di ottimizzazione dell'oggetto programma associato. Ciò risulta utile durante l'esecuzione del debug per una classe convertita con un livello di ottimizzazione pari a 40. Per forzare l'interpretazione, utilizzare INTERPRET(*YES).

» Consultare Programmi di caricamento di classe nella memoria cache per informazioni sul riutilizzo dei programmi Java creati dai programmi di caricamento di classe. «

Considerazioni sulle prestazioni della compilazione statica di Java

E' possibile determinare la velocità di conversione tramite il livello di ottimizzazione impostato. Il livello di ottimizzazione 10 converte nella maniera più veloce, ma il programma che ne risulta è in genere più lento di uno impostato su un livello di ottimizzazione più alto. Il livello di ottimizzazione 40 richiede più tempo per la conversione, ma è più probabile un'esecuzione più veloce.

Un numero limitato di programmi Java^(TM) non sono in grado di ottimizzarsi al livello 40. Per tale ragione, questi programmi che non vengono eseguiti a livello 40, possono essere eseguiti a livello 30. E' possibile

eseguire i programmi che non vengono eseguiti a livello di ottimizzazione 40 utilizzando le Stringhe di parametro LICOPT (licensed internal code optimization). Le prestazioni a livello 30, tuttavia, possono essere sufficienti per il programma.

Se esistono problemi nell'esecuzione del codice Java che sembra operare su un'altra Java virtual machine, tentare con l'utilizzo del livello di ottimizzazione 30 invece del livello 40. Se funziona e le prestazioni sono accettabili, non è necessario effettuare altre operazioni. Se c'è bisogno di prestazioni migliori, consultare Stringhe di parametro LICOPT per informazioni sul modo in cui abilitare e disabilitare i vari formati dell'ottimizzazione. Ad esempio, è possibile in primo luogo tentare di creare il programma utilizzando OPTIMIZE(40) LICOPT(NoPreresolveExtRef). Se l'applicazione contiene chiamate inattive a classi non disponibili, il valore LICOPT consente ai programmi l'esecuzione senza alcun problema.

Per determinare il livello di ottimizzazione utilizzato per creare i programmi Java, è possibile utilizzare il comando DSPJVAPGM (Visualizzazione programma Java). Per modificare il livello di ottimizzazione del programma Java, utilizzare il comando CRTJVAPGM (Creazione programma Java).

Compilatore JIT (Just-In-Time)

Un compilatore JIT (Just-In-Time) è un compilatore specifico della piattaforma che genera istruzioni macchina per ogni metodo sulla prima chiamata a tale metodo. Per migliorare le prestazioni relative all'interpretazione, il compilatore JIT compila il codice appena lo richiede l'utente.

Per capire la differenza tra il compilatore JIT e l'elaborazione diretta, esaminare Confronto tra il compilatore JIT e l'elaborazione diretta.

Far riferimento alle proprietà di sistema Java^(TM) per informazioni sulla proprietà java.compiler. Selezionare la versione in esecuzione.

Confronto tra il compilatore JIT (Just-In-Time) e l'elaborazione diretta

Se l'utente è indeciso sull'utilizzo della modalità del compilatore JIT (Just-In-Time) o dell'elaborazione diretta per eseguire il proprio programma Java^(TM), questa tabella fornisce informazioni aggiuntive affinché l'utente effettui la scelta più appropriata per la propria situazione.

Compilatore JIT (Just-In-Time)	Elaborazione diretta
Fornisce una compilazione automatica di qualsiasi metodo quando è necessario. Questo metodo è molto più rapido dell'elaborazione diretta.	L'utente compila un'intera classe o un file JAR utilizzando il comando CRTJVAPGM (Creazione programma Java) o i file vengono compilati automaticamente durante il tempo di esecuzione.
Utilizzato durante lo sviluppo del programma per tagliare i costi di ottimizzazione mentre il codice si sta modificando. Utilizzato anche nello sviluppo di applicazioni di piccole dimensioni o di utilizzo relativamente basso. E' inoltre possibile utilizzarlo con applicazioni altamente dinamiche dove il codice viene generato o rilevato durante il tempo di esecuzione.	Utilizzato per applicazioni relativamente ampie. Il codice di elaborazione diretta, al livello di ottimizzazione 40, è normalmente più rapido rispetto a JIT. La maggior parte delle applicazioni del server pronte per la distribuzione utilizzano l'elaborazione diretta al livello di ottimizzazione 40, in quanto è possibile che molti utenti le utilizzino in qualsiasi momento e il costo dell'uso ripetuto di JIT è troppo alto.
Le ottimizzazioni sono limitate a quelle che è possibile eseguire rapidamente durante il tempo di esecuzione.	Sono possibili ottimizzazioni più complesse, dato che l'ottimizzazione non viene eseguita durante il tempo di esecuzione.

Livelli di ottimizzazione

Immettendo un valore nel campo **Livello di ottimizzazione**, viene specificato il livello di ottimizzazione del programma Java^(TM) collegato all'oggetto del file di classe o all'oggetto del file JAR. E' possibile controllare la dimensione e le prestazioni del programma Java utilizzando questa opzione. Per i livelli di ottimizzazione elencati, il formato interno include le istruzioni macchina iSeries. Queste istruzioni macchina

sono ottimizzate in base al livello di ottimizzazione specificato. Il server iSeries esegue direttamente le istruzioni macchina quando il programma è in esecuzione.

Questa lista mostra le differenze che intercorrono tra i livelli di ottimizzazione e le funzioni di questi:

10

Il programma Java contiene una versione convertita dei bytecode del file di classe, ma ha un'ottimizzazione minima del compilatore aggiuntivo. E' possibile visualizzare e modificare le variabili durante l'esecuzione del debug. Per V5R1 e per una versione successiva, è possibile che un livello di ottimizzazione pari a 10 allunghi inutilmente il tempo impiegato per la compilazione del programma. E' preferibile utilizzare un livello di ottimizzazione almeno pari a 20.

20

Il programma Java contiene una versione compilata dei bytecode del file di classe ed esegue l'ottimizzazione del compilatore aggiuntivo. E' possibile visualizzare, ma non modificare le variabili durante l'esecuzione del debug.

30

Il programma Java contiene una versione compilata dei bytecode del file di classe ed esegue più ottimizzazioni del compilatore rispetto al livello di ottimizzazione 20. L'esecuzione del debug ad alti livelli di ottimizzazione risulta più difficile, perché l'ottimizzazione riduce l'abilità di fermarsi in punti precisi nel codice e di visualizzare le variabili del programma. E' possibile visualizzare, ma non modificare le variabili durante l'esecuzione del debug. E' possibile che i valori indicati non siano i valori correnti della variabile.

40

Il programma Java contiene una versione compilata dei bytecode del file di classe ed esegue più ottimizzazioni del compilatore rispetto al livello di ottimizzazione 30. Il livello di ottimizzazione pari a 40 include ottimizzazioni a classe incrociata. In un limitato numero di casi, l'ordine in cui vengono eseguiti i programmi di inizializzazione statici per classi non correlate (non correlate per eredità né perché contenute l'una nell'altra) può essere diverso rispetto a quanto profilato nella specifica di inizializzazione statica. Inoltre tale livello include ottimizzazioni che disabilitano la traccia di istruzioni e chiamate.

Nota: se il programma Java ha esito negativo durante l'ottimizzazione o lancia un'eccezione al livello di ottimizzazione 40, utilizzare il livello di ottimizzazione 30.

Raccolta di dati inutili Java

La raccolta di dati inutili è il processo di liberare la memoria che viene utilizzata da oggetti a 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 raccoglitore 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^(TM) runtime environment include un raccoglitore di dati inutili che libera la memoria non più in uso. Tale raccoglitore si attiva automaticamente, quando richiesto.

E' inoltre possibile avviare il raccoglitore di dati inutili esplicitamente sotto il controllo del programma Java che utilizza il metodo `java.lang.Runtime.gc()`.

Consultare Raccolta avanzata dati inutili di IBM Developer Kit per Java di specifiche di IBM Developer Kit per Java.

Raccolta di dati inutili avanzata di IBM Developer Kit per Java

IBM Developer Kit per Java^(TM) implementa un algoritmo del raccoglitore di dati inutili avanzato. Tale algoritmo permette la rilevazione e la raccolta di oggetti irraggiungibili senza pause significative nell'operazione del programma Java. Un raccoglitore simultaneo rileva contemporaneamente i riferimenti agli oggetti all'interno dei sottoprocessi in esecuzione, invece che in un singolo sottoprocesso.

Molti raccoglitori 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 raccoglitore di dati inutili esegue il suo lavoro. Quando questo accade, i programmi Java effettuano una pausa e si logora ogni capacità del processore multiplo della piattaforma relativa a Java, mentre il raccoglitore di dati inutili è al lavoro. L'algoritmo iSeries non arresta tutti i sottoprocessi del programma simultaneamente. Esso consente a quei sottoprocessi di continuare l'operazione mentre il raccoglitore 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 JVM (Java virtual machine). E' 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 Raccolta di dati inutili Java.

Considerazioni sulle prestazioni della raccolta di dati inutili Java

La raccolta di dati inutili sulla Java^(TM) virtual machine iSeries opera in modalità asincrona continua. E' possibile che il parametro GCHINL (Dimensione iniziale della raccolta di dati inutili) sul comando RUNJVA (Esecuzione Java) influenzi le prestazioni dell'applicazione. Questo parametro 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 su un server iSeries non è uguale all'esecuzione di tale richiamo su altre piattaforme. Java^(TM) sul server iSeries è stato ottimizzato spostando la 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) retroattive dispendiose nella Java virtual machine. Sarebbe opportuno che i metodi nativi non eseguissero routine di piccole dimensioni, che è possibile scrivere facilmente in Java. Utilizzare solo i metodi nativi per avviare le funzioni del sistema relativamente lunghe da eseguire e che non sono disponibili direttamente in Java.

Considerazioni sulle prestazioni di allineamento del metodo Java

L'allineamento del metodo può migliorare in modo significativo le prestazioni di chiamata al metodo. Qualsiasi metodo finale è un potenziale candidato per l'allineamento. La funzione di allineamento è disponibile sul server iSeries tramite l'opzione `-o javac` al momento della compilazione. La dimensione dei propri file di classe e programmi Java^(TM) convertiti aumenta se si utilizza l'opzione `-o javac`. Sarebbe opportuno considerare entrambe le caratteristiche delle prestazioni e dello spazio della propria applicazione quando si utilizza l'opzione `-o`.

Il programma di conversione Java abilita l'allineamento per i livelli di ottimizzazione 30 e 40. Il livello di ottimizzazione 30 abilita l'allineamento dei metodi finali all'interno di una singola classe. Il livello di ottimizzazione 40 abilita l'allineamento dei metodi finali all'interno di un file ZIP o JAR. E' possibile controllare l'allineamento del metodo con le stringhe del parametro LICOPT AllowInlining e NoAllowInlining. L'interprete iSeries non esegue l'allineamento del metodo.

Considerazioni sulle prestazioni dell'eccezione Java

L'architettura dell'eccezione iSeries consente capacità versatili di interruzione e ripetizione. Essa consente l'interazione di un linguaggio misto. L'emissione di eccezioni Java^(TM) su un server iSeries può essere più dispendiosa rispetto che su altre piattaforme. Ciò non dovrebbe influenzare le prestazioni globali dell'applicazione a meno che non vengano utilizzate in modo abitudinario le eccezioni Java nel normale percorso di applicazione.

Strumenti delle prestazioni delle tracce di chiamata Java

Le tracce di chiamata del metodo Java^(TM) forniscono informazioni significative sulle prestazioni riguardo al tempo impiegato in ogni metodo Java. Su altre JVM (Java virtual machine), è possibile che sia stata utilizzata l'opzione -prof (modellamento) sul comando java. Per abilitare la traccia di chiamata al metodo su un server iSeries, è necessario specificare il comando ENBPFRCOL (Abilitazione raccolta prestazioni) sulla riga comandi CRTJVAPGM (Creazione programma Java). Dopo avere creato il proprio programma Java con questa parola chiave, è possibile avviare la raccolta delle tracce di chiamata al metodo utilizzando una definizione PEX (Performance Explorer) che include il tipo di traccia di restituzione/chiamata.

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.

Strumento delle prestazioni delle tracce di chiamata Java

La Java^(TM) virtual machine iSeries abilita la traccia di alcuni eventi Java. E' 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). E' 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 il manuale Performance Tools per iSeries,

SC41-5340. 

Strumenti delle prestazioni di modellamento Java

Il profilo CPU (central processing unit) dell'intero sistema calcola la quantità relativa di tempo della CPU impiegato in ogni metodo Java^(TM) 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 del 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. Lo Strumento PDC (Performance Data Collector) fornisce anche informazioni sul profilo dei programmi in esecuzione sul server iSeries.

Nota: il profilo CPU non indica l'utilizzo relativo della CPU per i programmi Java interpretati.

JVMPI (Java Virtual Machine Profiler Interface)

» La JVMPI (JavaTM Virtual Machine Profiler Interface) è una interfaccia sperimentale per il profilo di JVM (Java virtual machine), che è stata inizialmente distribuita e implementata in J2SDK (Java 2 SDK, Edizione standard), versione 1.2. di Sun.

Il supporto JVMPI posiziona gli hook nella JVM e nel compilatore JIT (Just-in-time), che quando attivati, forniscono informazioni sull'evento ad un agent di profilo. L'agent di profilo viene implementato come una DLL (dynamic link library). Il programma per la creazione profili invia informazioni sul controllo alla JVM per l'abilitazione e la disabilitazione di eventi JVMPI. Ad esempio, è possibile che il programma per la creazione profili non sia interessato agli hook Entrata e Uscita metodo e potrebbe comunicare alla JVM di non volere ricevere queste notifiche di eventi. La JVM e JIT hanno hook di eventi JVMPI incorporati che inviano notifiche di eventi all'agent di 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 modellamento quando questi si verificano.

Per ulteriori informazioni, consultare JVMPI  di Sun Microsystems, Inc. 

Raccogliere dati delle prestazioni Java

Per raccogliere dati delle prestazioni di JavaTM su un server iSeries, effettuare quanto segue:

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:

```
ADDPXDFN DFN(YOURDFN) JOB(*ALL/YOURID/QJVACMSRV) 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:

```
ADDPXDFN DFN(YOURDFN) TYPE(*TRACE) JOB(*ALL) TRCTYPE(*SLTEVT)
SLTEVT(*YES) PGMEVT(*JVAENTRY *JVAEXIT)
```

Questa definizione *TRACE raccoglie qualunque evento di entrata e di uscita Java da qualsiasi programma Java nel sistema che si crea con ENBPFRCOL(*ENTRYEXIT). Ciò rallenta l'analisi di questo tipo di raccolta rispetto ad una traccia *STATS, in base al numero di eventi del programma Java di cui si dispone e alla durata della raccolta di dati PEX.

2. Abilitare *JVAENTRY e *JVAEXIT, nella categoria eventi del programma sulla definizione PEX, in modo che PEX riconosca l'entrata e le uscite Java.

Nota: se si sta eseguendo il codice Java utilizzando il compilatore JIT (Just-in-time), non si abilita l'entrata e l'uscita come avverrebbe se si utilizzasse il comando CRTJVAPGM per un'elaborazione diretta. Al contrario, JIT genera un codice con hook di entrata e uscita quando si utilizza la proprietà di sistema os400.enbprfcol.

3. Preparare il programma Java in modo che riporti eventi del programma all'iSeries Performance Data Collector. E' possibile effettuare tale operazione utilizzando il comando CRTJVAPGM (Creazione programma Java) in ogni programma Java su cui si intende riportare i dati delle prestazioni. E' necessario creare il programma Java utilizzando il parametro ENBPFRCOL(*ENTRYEXIT).

Nota: è necessario ripetere questa fase per ogni programma Java su cui si intende raccogliere i dati delle prestazioni. Se non si esegue questa fase, non verrà raccolto alcun dato delle prestazioni dal PEX e non verrà prodotta alcuna emissione eseguendo lo strumento JPDC (Java Performance Data Converter).

4. Avviare la raccolta dati PEX utilizzando il comando STRPEX (Avvio Performance Explorer).
5. 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 tempo genera molti dati di sistema PEX. Se vengono raccolti troppi dati, è necessario troppo tempo per elaborarli.
6. 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.
7. Eseguire lo strumento JPDC.
8. Collegare l'indirizzario dell'IFS (Integrated File System) al sistema con il programma di visualizzazione scelto: programma di visualizzazione java_g -prof o Jinsight. E' possibile copiare questo file dal proprio server iSeries e utilizzarlo come immissione in ogni strumento per la creazione di profili adatto.

Strumento Performance Data Collector

Lo strumento PDC (Performance Data Collector) fornisce informazioni sul profilo riguardo ai programmi in esecuzione nel server iSeries.

L'opzione profilo standard aziendale in molte Java^(TM) virtual machine dipende dall'implementazione della caratteristica java_g. Questa è una versione speciale di debug della Java virtual machine, che offre l'opzione -prof. Questa opzione è stata specificata su una chiamata a un programma Java. Quando questa opzione viene specificata, la Java virtual machine crea un file record che contiene informazioni sulle parti del programma Java operative per la durata del programma. La Java virtual machine crea queste informazioni in tempo reale.

Sul server iSeries, la funzione PEX (Performance Explorer) analizza i programmi e gli eventi del sistema specifici per i record. Un database DB2^(R) memorizza queste informazioni e le reperisce utilizzando le funzioni SQL. Le informazioni PEX rappresentano il contenitore per informazioni su programmi specifici che producono dati di profilo Java. Essi sono compatibili con le informazioni sul profilo del programma java_g -prof. Lo strumento JPDC (Java Performance Data Converter) fornisce informazioni sul profilo del programma e l'emissione del programma java_g -prof per uno specifico strumento IBM, noto come Jinsight.


Per informazioni sulla modalità di raccolta dei dati delle prestazioni Java, consultare Raccogliere dati delle prestazioni Java.

Strumento Java Performance Data Converter

Lo strumento JDPC (Java^(TM) Performance Data Converter) fornisce un modo per creare dei dati relativi alle prestazioni Java sui programmi Java in esecuzione sul server iSeries. Tali dati sono compatibili con l'emissione di dati delle prestazioni dell'opzione java_g -prof della Java virtual machine di Sun Microsystems, Inc. e con l'emissione Jinsight IBM.

Nota: lo strumento JDPC non produce un'emissione leggibile. Utilizzare uno strumento di modellamento java che accetti `java_g -prof` o i dati Jinsight per analizzare i dati.

Lo strumento JDPC accede ai dati PEX (Performance Explorer) di iSeries che DB2/400 (che utilizza JDBC) memorizza. Esso converte i dati in Jinsight o in tipi di prestazioni generali. Quindi, JDPC memorizza il file di emissione nell'IFS (Integrated File System) in un'ubicazione specificata dall'utente.

Nota: è necessario seguire le appropriate procedure di raccolta dati PEX di iSeries per raccogliere tali dati mentre si esegue la propria applicazione Java specificata su un server iSeries. E' necessario impostare una definizione PEX che stabilisca l'entrata ed uscita di un programma o di una procedura di raccolta e memorizzazione. Per dettagli su come raccogliere i dati e impostare una definizione PEX, consultare il manuale Performance Tools for iSeries, SC41-5340. 

Per informazioni su come eseguire JPDC, consultare Eseguire Java Performance Data Converter.

E' possibile avviare il programma JPDC utilizzando l'interfaccia della riga comandi Qshell o il comando RUNJVA (Esecuzione Java).

Eseguire programma di conversione dei dati delle prestazioni Java

Per eseguire JPDC (JavaTM) Performance Data Converter) per la raccolta dei dati sulle prestazioni, effettuare quanto segue:

1. Immettere il primo argomento di immissione, che risulta essere `general` per `java_g -prof` o `jinsight` per l'emissione Jinsight.
2. Immettere il secondo argomento di immissione, che risulta essere il nome della definizione PEX (Performance Explorer) utilizzata per raccogliere i dati.
Nota: è necessario limitare questo nome a quattro o cinque caratteri, a causa dell'utilizzo interno dei relativi collegamenti.
3. Immettere il terzo argomento di immissione, che risulta essere il nome del file che lo strumento JPDC crea. Questo file creato scrive nell'indirizzario corrente dell'integrated file system. Utilizzare il comando `cd` (PF4) per specificare un indirizzario corrente dell'integrated file system.
4. Immettere il quarto argomento di immissione, che risulta essere il nome della voce di indirizzario del database relazionale dell'host iSeries. Utilizzare il comando `WRKRDBDIRE` (Gestione voce indirizzario RDB) per esaminare di quale nome si tratta. Si tratta dell'unico database relazionale in cui è indicato `*LOCAL`.

Per eseguire questo codice è necessario che il file `/QIBM/ProdData/Java400/ext/JPDC.jar` si trovi nel classpath Java sul server iSeries. Quando questo programma ha terminato l'esecuzione, è possibile rilevare un file di emissione testo nell'indirizzario corrente.

E' possibile eseguire JPDC utilizzando la riga comandi iSeries o l'ambiente Qshell. Consultare Esempio: eseguire Java Performance Data Converter per dettagli.

Esempio: eseguire Java Performance Data Converter

E' possibile utilizzare la riga comandi iSeries o l'ambiente Qshell per eseguire JPDC (JavaTM) Performance Data Converter).

Utilizzare la riga comandi iSeries:

1. Immettere il comando `RUNJVA` (Esecuzione Java) o il comando `JAVA` sulla riga comandi iSeries.
2. Immettere `com.ibm.as400.jpdc.JPDC` sulla riga relativa al parametro classe.
3. Immettere `general pexdfn mydir/myfile myrdbdire` sulla riga parametri.
4. Immettere `'/QIBM/ProdData/Java400/ext/JPDC.jar'` sulla riga parametri del classpath.

Nota: è possibile omettere il classpath se la stringa '/QIBM/ProdData/Java400/ext/JPDC.jar' si trova nella variabile di ambiente CLASSPATH. E' possibile utilizzare il comando ADDENVVAR (Aggiunta variabile di ambiente), il comando CHGENVVAR (Modifica variabile di ambiente) o il comando WRKENVVAR (Gestione variabile di ambiente) per aggiungere questa stringa alla variabile di ambiente CLASSPATH.

Utilizzare l'ambiente Qshell:

1. Immettere il comando STRQSH (Avvio Qshell) per avviare Qshell Interpreter.
2. Immettere quanto segue sulla riga comandi:

```
java -classpath /QIBM/ProdData/Java400/ext/JPDC.jar com.ibm.as400/jpdc/JPDC  
jinsight pexdfn mydir/myfile myrdbdire
```

Nota: è possibile omettere il classpath se la stringa '/QIBM/ProdData/Java400/ext/JPDC.jar' è stata aggiunta all'ambiente corrente. E' possibile utilizzare il comando ADDENVVAR, CHGENVVAR o WRKENVVAR per aggiungere questa stringa all'ambiente corrente.

Per informazioni di background, consultare Eseguire Java Performance Data Converter.

Capitolo 6. Comandi e strumenti per IBM Developer Kit per Java

Quando si utilizza IBM Developer Kit per Java^(TM), è possibile usare gli strumenti Java con Qshell Interpreter o i comandi CL.

Se si ha un'esperienza precedente di programmazione Java, è consigliabile utilizzare gli strumenti Java Qshell Interpreter, in quanto sono simili agli strumenti che si utilizzerebbero con JDK di Sun Microsystems, Inc. Consultare Qshell Interpreter per informazioni sull'utilizzo dell'ambiente Qshell.



Se l'utente è un programmatore iSeries, è possibile utilizzare i comandi CL per Java, tipici dell'ambiente server iSeries. Continuare a leggere per ulteriori informazioni sull'utilizzo dei comandi CL e dei comandi di iSeries Navigator.

E' possibile utilizzare uno qualsiasi di questi comandi e strumenti con IBM Developer Kit per Java:

- L'ambiente Qshell include gli strumenti di sviluppo Java richiesti normalmente per lo sviluppo del programma.
- L'ambiente CL contiene i comandi CL per ottimizzare e gestire i programmi Java.
- Inoltre, i comandi di iSeries Navigator creano ed eseguono programmi Java ottimizzati.

Strumenti Java supportati da IBM Developer Kit per Java

IBM Developer Kit per Java^(TM) supporta questi strumenti:

- Strumento ajar Java
- Strumento appletviewer Java
- Strumento extcheck Java
-  Strumento idlj Java 
- Strumento jar Java
- Strumento jarsigner Java
- Strumento javac Java
- Strumento javadoc Java
- Strumento javah Java
- Strumento javakey Java
- Strumento javap Java
- Keytool Java
- Strumento native2ascii Java
- Policytool Java
- Strumento rmic Java
- Strumento rmid Java
- Strumento rmiregistry Java
- Strumento serialver Java
- Strumento tnameserv Java

Con poche eccezioni, gli strumenti Java, eccetto lo strumento ajar, supportano la sintassi e le opzioni documentate da Sun Microsystems, Inc. Tutti gli strumenti devono essere eseguiti utilizzando Qshell Interpreter.

E' 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. Consultare Comando Java in Qshell per ulteriori dettagli.

Nota: le funzioni dell'immissione comando iSeries non sono disponibili direttamente dall'interno di Qshell. Per richiamare un'immissione comando, premere F21 (Immissione comando CL).

Strumenti Java

- Strumento ajar Java
- Strumento appletviewer Java
 - Eseguire lo strumento appletviewer Java con Abstract Window Toolkit Remoto
- Strumento extcheck Java
- Strumento idlj Java
- Strumento jar Java
- Strumento jarsigner Java
- Strumento javac Java
- Strumento javadoc Java

Strumento ajar Java

Lo strumento ajar è un'interfaccia alternativa allo strumento jar che si utilizza per creare e gestire i file JAR (JavaTM ARchive). E' possibile utilizzare lo strumento ajar per gestire sia i file JAR che i file ZIP.

Se è necessaria un'interfaccia ZIP o UNZIP, utilizzare lo strumento ajar invece dello strumento jar.

Lo strumento ajar elenca il contenuto dei file JAR, estrae dai file JAR, crea nuovi file JAR e supporta molti dei formati ZIP come lo strumento jar. Inoltre, lo strumento ajar supporta l'aggiunta e la cancellazione dei file nei file JAR esistenti.

Lo strumento ajar è disponibile utilizzando Qshell Interpreter. Per ulteriori dettagli, esaminare ajar - Alternative Java archive.

Strumento appletviewer Java

Lo strumento appletviewer consente di eseguire le applet senza un browser web. Esso è compatibile con lo strumento appletviewer fornito da Sun Microsystems, Inc.

Lo strumento appletviewer è disponibile utilizzando Qshell Interpreter. E' necessario utilizzare Abstract Window Toolkit Remoto per eseguire lo strumento appletviewer. Consultare Eseguire lo strumento appletviewer Java con l'AWT Remoto per informazioni su come impostare l'AWT Remoto per utilizzare lo strumento appletviewer.

Per ulteriori informazioni sullo strumento appletviewer, esaminare lo strumento appletviewer di Sun Microsystems, Inc.

Eseguire lo strumento appletviewer Java con Abstract Window Toolkit Remoto: Per utilizzare lo strumento appletviewer Java, è necessario impostare AWT (Abstract Window Toolkit) remoto per Java su un pannello remoto di Windows^(R) ed utilizzare la classe sun.applet.AppletViewer o eseguire lo strumento appletviewer nel Qshell Interpreter con Proprietà AWT remoto.

Ad esempio, se si utilizza la classe sun.applet.AppletViewer e si esegue example1.html fuori dall'indirizzo TicTacToe, la riga comandi dovrebbe risultare in questo modo:

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html') CLASSPATH('/TicTacToe')
PROP((RmtAwtServer '1.1.11.11') (os400.class.path.rawt 1)(java version 1.3))
```

Se si utilizza lo strumento `appletviewer` in Qshell Interpreter e si esegue `example1.html` fuori dall'indirizzo `TicTacToe`, i comandi dovrebbero risultare in questo modo:

```
qsh "enter"  
cd TicTacToe "enter"  
Appletviewer -J-DRmtAwtServer=1.1.11.11 -J-Dos400.class.path.rawt=1 -J-Djava.version=1.3 example1.html
```

Nota: `-J` sono indicatori del tempo di esecuzione `Appletviewer`. `-D` sono proprietà.

Strumento `extcheck` Java

In J2SDK (Java 2 SDK), Edizione Standard, versione 1.2 e successive, lo strumento `extcheck` individua i conflitti di versione tra un file JAR di destinazione e i file JAR di estensione installati contemporaneamente. Esso è compatibile con il `keytool` fornito da Sun Microsystems, Inc.

Lo strumento `extcheck` è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni sullo strumento `extcheck`, esaminare lo strumento `extcheck` di Sun Microsystems, Inc.

Strumento `idlj` Java

► Lo strumento `idlj` genera collegamenti Java da un determinato file IDL (Interface Definition Language). Lo strumento `idlj` è inoltre conosciuto come compilatore IDL-per-Java. Esso è compatibile con lo strumento `idlj` fornito da Sun Microsystems, Inc. Questo strumento funziona soltanto per i Java Development Kit 1.3 e 1.4.

Per ulteriori informazioni sullo strumento `idlj`, esaminare lo strumento `idlj` di Sun Microsystems, Inc.

◀ `alt=`

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.

Per un'interfaccia alternativa allo strumento `jar`, esaminare lo strumento `ajar` per creare e gestire i file JAR.

Per ulteriori informazioni sui file system iSeries, esaminare `loIFS` (Integrated File System) o `File` nell'`IFS` (Integrated File System).

Per ulteriori informazioni sullo strumento `jar`, esaminare lo strumento `jar` di Sun Microsystems, Inc.

Strumento `jarsigner` Java

In J2SDK (Java 2 SDK), Edizione Standard, versione 1.2 e successive, lo strumento `jarsigner` firma i file JAR e verifica le firme sui file JAR firmati. Lo strumento `jarsigner` accede al keystore, 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.

Per ulteriori informazioni sullo strumento `jarsigner`, esaminare lo strumento `jarsigner` 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.

Se si dispone di JDK 1.1.x installato sul proprio server iSeries come valore predefinito, ma è necessario eseguire il comando `java` da una versione 1.2 o successive, immettere questo comando:

```
javac -djava.version=1.2 <my_dir> MyProgram.java
```

Per informazioni sullo strumento `javac`, esaminare lo strumento `javac` 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.

Per ulteriori informazioni sullo strumento `javadoc`, esaminare lo strumento `javadoc` di Sun Microsystems, Inc.

Strumenti Java

- Strumento `javah` Java
- Strumento `javakey` Java
- Strumento `javap` Java
- Keytool Java
- Strumento Java `native2ascii`
- Policytool Java
- Strumento `rmic` Java
- Strumento `rmid` Java
- Strumento `rmiregistry` Java
- Strumento `serialver` Java
- Strumento `tnameserv` Java

Strumento javah Java

Lo strumento `javah` facilita l'implementazione dei metodi nativi di Java^(TM). 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 puro Java al 100%. Ciò significa inoltre che la propria applicazione non è direttamente portabile tra le piattaforme. 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) iSeries. Esso deve essere copiato in un membro file prima di poter essere incluso in un programma C sul server iSeries.

Lo strumento `javah` è compatibile con lo strumento fornito da Sun Microsystems, Inc. Anche se vengono specificate queste opzioni, il server iSeries le ignora.

-td Lo strumento `javah` sul server iSeries non richiede un indirizzario temporaneo.

-stubs Java sul server iSeries supporta soltanto il formato JNI (Java Native Interface) dei metodi nativi. Gli stub venivano richiesti solo per il formato precedente a JNI dei metodi nativi.

-trace

E' collegato all'emissione del file stub .c, che Java sul server iSeries non supporta.

-v Non supportato.

Nota: è sempre necessario specificare l'opzione -jni. Il server iSeries non supporta le implementazioni del metodo nativo precedenti a JNI.

Per ulteriori informazioni sullo strumento javah, esaminare lo strumento javah di Sun Microsystems, Inc.

Strumento javakey Java

Utilizzare lo strumento javakey per creare e gestire chiavi di codifica generare il certificato, inclusa la creazione di firme digitali per le applet. Esso è compatibile con lo strumento javakey fornito da Sun Microsystems, Inc. in JDK (Java Development Kit), versioni 1.1.x.

In J2SDK (Java 2 Software Development Kit), Edizione Standard, versione 1.2 o successive, lo strumento javakey è obsoleto. A causa di malfunzionamento nelle versioni JDK 1.1.x, il codice firmato utilizzando lo strumento javakey 1.1.x viene riconosciuto come non firmato in J2SDK, versione 1.2 o successive. Se si firma un codice utilizzando J2SDK, versione 1.2 o successive, esso viene riconosciuto come non firmato nelle versioni JDK 1.1.x.

Nota: il supporto SSL (secure socket layer) iSeries non può accedere alle chiavi create da questo strumento. Al contrario, è necessario utilizzare i contenitori chiavi e certificati integrati nel server iSeries e creati o importati con il Digital Certificate Manager. Consultare *Rendere un'applicazione Java protetta con gli SSL (secure socket layer)* per ulteriori informazioni.

La compressione e la firma di applet dipendono dal proprio browser. Controllare la documentazione del proprio browser per assicurarsi che esso sia compatibile con il formato del file JAR Java e con la firma dell'applet javakey.

Nota: i file creati dallo strumento javakey contengono informazioni sensibili. Appropriate misure di sicurezza dell'IFS (Integrated File System) proteggono i file chiave pubblica e privata.

Lo strumento javakey è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni sui file system iSeries, esaminare *loIFS (Integrated File System) o File nell'IFS (Integrated File System)*.

Per ulteriori informazioni sullo strumento javakey, esaminare strumento javakey 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.

Esso è compatibile con lo strumento javap fornito da Sun Microsystems, Inc. con alcune eccezioni.

-b Questa opzione viene ignorata. La compatibilità retroattiva non è necessaria, in quanto Java sul server iSeries supporta soltanto JDK (Java Development Kit) 1.1.4 e successive.

-p Sul server iSeries, -p non è un'opzione valida. E' necessario specificare -private.

-verify Questa opzione viene ignorata. Lo strumento javap non effettua una verifica sul server iSeries.

Lo strumento javap è disponibile utilizzando Qshell Interpreter.

Nota: è 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.

Per ulteriori informazioni sullo strumento javap, esaminare lo strumento javap di Sun Microsystems, Inc.

Keytool Java

In J2SDK (Java 2 SDK), Edizione Standard, versione 1.2 o successive, il keytool crea coppie di chiavi pubbliche e private, autocertificazioni e keystore di gestione. 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.

Per ulteriori informazioni sul keytool, esaminare il keytool di Sun Microsystems, Inc.

Strumento native2ascii Java

Lo strumento native2ascii converte un file con caratteri codificati nativi (caratteri diversi da Latin 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.

Per ulteriori informazioni sullo strumento native2ascii, esaminare lo strumento native2ascii di Sun Microsystems, Inc.

Policytool Java

In J2SDK (Java 2 SDK), Edizione Standard, il policytool crea e modifica i file di configurazione normativa esterni che definiscono la normativa di sicurezza Java della propria installazione. Esso è compatibile con il policytool fornito da Sun Microsystems, Inc.

Il policytool è uno strumento GUI (graphical user interface) disponibile utilizzando Qshell Interpreter e Abstract Window Toolkit Remoto. Esaminare Abstract Window Toolkit Remoto di IBM Developer Kit per Java per ulteriori informazioni.

Per ulteriori informazioni sul policytool, esaminare il 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 J2SDK (Java 2 SDK), Edizione Standard, lo strumento rmid avvia il daemon del sistema di attivazione, in modo che sia possibile registrare e attivare gli oggetti in una Java virtual machine. Esso è compatibile con lo strumento rmid fornito da Sun Microsystems, Inc.

Lo strumento rmid è disponibile utilizzando Qshell Interpreter.

Per ulteriori informazioni sullo strumento rmid, esaminare lo 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.

Per ulteriori informazioni sullo strumento `rmiregistry`, esaminare lo 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.

Per ulteriori informazioni sullo strumento `serialver`, esaminare lo strumento `serialver` di Sun Microsystems, Inc.

Strumento `tnameserv` Java

In J2DSK (Java 2 SDK), Edizione Standard, versione 1.3  o successive,  lo strumento `tnameserv` (Transient Naming Service) fornisce l'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.

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
<code>-cs</code>	Questa opzione non è supportata.
<code>-checksource</code>	Questa opzione non è supportata.
<code>-debug</code>	Questa opzione è supportata dal programma di debug interno di iSeries.
<code>-noasyncgc</code>	La raccolta di dati inutili è sempre in esecuzione con IBM Developer Kit per Java.
<code>-noclassgc</code>	La raccolta di dati inutili è sempre in esecuzione con IBM Developer Kit per Java.
<code>-prof</code>	Il server iSeries dispone di strumenti delle prestazioni propri.
<code>-ss</code>	Questa opzione non è applicabile sul server iSeries.
<code>-oss</code>	Questa opzione non è applicabile sul server iSeries.
<code>-t</code>	Il server iSeries utilizza la funzione di traccia propria.
<code>-verify</code>	Verifica continua sul server iSeries.
<code>-verifyremote</code>	Verifica continua sul server iSeries.
<code>-noverify</code>	Verifica continua sul server iSeries.

Sul server iSeries, l'opzione `-classpath` non sostituisce il classpath predefinito. Al contrario, viene accordato al classpath predefinito di sistema. L'opzione `-classpath` sostituisce la variabile di ambiente `CLASSPATH`.

Il comando `java` in Qshell supporta opzioni nuove per il server iSeries. Esistono opzioni nuove che vengono supportate.

Opzione	Descrizione
 <code>-chkpath</code> 	Questa opzione controlla l'accesso pubblico alla scrittura agli indirizzi nel <code>CLASSPATH</code> .
<code>-opt</code>	Questa opzione specifica il livello di ottimizzazione.

Opzione	Descrizione
» -showversion	Questa opzione specifica la versione JDK. Questa versione esiste per JDK 1.3 e 1.4. «
» -verbose[:class gc jni] «	Viene visualizzato un messaggio per ciascuna ripulitura della raccolta di dati inutili.
» -Xrun[:]	Viene visualizzato un messaggio indicante che un programma di servizio e una stringa di parametri facoltativi per JVM_OnLoad sono in esecuzione durante l'avvio di JVM. «

Il comando RUNJVA (Esecuzione Java) nelle informazioni di riferimento sul comando CL descrive queste nuove opzioni in dettaglio. 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

IBM Developer Kit per Java^(TM) supporta questi comandi CL.

- » Il comando ANZJVM (Analisi JVM) richiama ed imposta le informazioni in una JVM (Java virtual machine). Questo 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 server iSeries da un file di classe Java, file ZIP o file JAR.
- Il comando DLTJVAPGM (Cancellazione programma Java) cancella un programma Java iSeries associato ad un file di classe Java, file ZIP o file JAR.
- Il comando DSPJVAPGM (Visualizzazione programma Java) visualizza informazioni su un programma Java su iSeries.
- Il comando DMPJVM (Esecuzione del dump sulla Java Virtual Machine) esegue il dump su informazioni riguardo alla JVM per un lavoro specificato su un file di stampa di spool.
- Il comando JAVA e il comando RUNJVA (Esecuzione Java) eseguono i programmi Java iSeries.

Per ulteriori informazioni, consultare le API del comando CL e del programma.

Comando ANZJVM (Analisi JVM - Java Virtual Machine)

» Il comando ANZJVM (Analisi JVM - Java^(TM) Virtual Machine) reperisce e imposta le informazioni in una JVM (Java virtual machine). Lo scopo è quello di fornire assistenza nel sottoporre a debug i propri programmi Java restituendo informazioni sulle classi attive.

Quando si esegue il comando ANZJVM, esiste un parametro per specificare se sarebbe opportuno forzare un ciclo di raccolta di dati inutili e, in questo caso, si è tentato di forzare tale ciclo prima di ogni passaggio. Se non è stato eseguito alcun ciclo di raccolta di dati inutili per la JVM analizzata, non è possibile forzare i cicli. Esistono inoltre parametri su come è necessario memorizzare le informazioni e quanto deve durare l'intervallo tra i passaggi.

Dopo l'esecuzione del comando ANZJVM, viene prodotto un file di emissione di spool. Consultare l'Esempio: comando ANZJVM e il file di emissione per ulteriori informazioni.

Far riferimento al comando ANZJVM e al diagramma di sintassi ANZJVM per ulteriori informazioni.

Far riferimento alle Considerazioni sul comando ANZJVM per ulteriori informazioni. <<

Eseguire il comando ANZJVM

>> E' possibile eseguire il comando ANZJVM (Analisi JVM - JavaTM Virtual Machine) per raccogliere informazioni sulla JVM (Java virtual machine) per un lavoro specificato. Acquisendo una copia della JVM e confrontando i dati con un'altra copia acquisita in un secondo momento, è possibile analizzare i dati per trovare eventuali mancanze dell'oggetto. Viene utilizzato un parametro di intervallo per specificare il tempo intercorso tra i passaggi dell'heap. Quando l'intervallo è impostato su zero, esistono due passaggi dell'heap, di cui il secondo inizia subito dopo la fine del primo. Vengono quindi restituite informazioni per entrambi i passaggi.

Le seguenti informazioni vengono restituite per ogni classe nell'heap:

1. Nome classe
2. Informazioni sull'heap della raccolta dati inutili
 - a. Primo passaggio
 - b. Secondo passaggio
 - c. Modificare il numero di oggetti nell'heap della raccolta dati inutili
3. Spazio dell'oggetto utilizzato
 - a. Primo passaggio
 - b. Secondo passaggio
 - c. Modificare la dimensione dell'oggetto
4. Informazioni di registro globale, che sono uguali a quanto elencato per la tabella dell'oggetto.
5. Nome del programma di caricamento

Forzare un ciclo di raccolta di dati inutili: Per ottenere una visualizzazione più chiara dell'heap, è consigliabile esaminarlo prima possibile dopo un ciclo di raccolta di dati inutili. ANZJVM dispone di un parametro FRCGC per specificare se è opportuno forzare la raccolta di dati inutili. Le opzioni possibili sono le seguenti:

- *YES
La raccolta di dati inutili viene forzata prima di ogni ripulitura dell'heap di ANZJVM.
- *NO
La raccolta di dati inutili non viene forzata da ANZJVM.<<

Considerazioni sul 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. <<

Esempio: comando ANZJVM

» Il seguente esempio del comando ANZJVM raccoglie due copie dei 60 secondi di JVM separatamente per un lavoro denominato QJVACMDSRC per il nome utente JOHN e il numero lavoro 099112. I dati vengono inseriti dalle copie in un file dell'unità di stampa QSYSPRT.

```
ANZJVM JOB(099112/JOHN/QJVACMSRV)
```

Segue un esempio del file di emissione di spool.

File di emissione di spool per il comando ANZJVM: Segue un esempio dei dati contenuti in un file di emissione di spool dopo l'esecuzione del comando ANZJVM:

```
Mon Feb 26 15:39:12 CST 2002
```

```
Job: 099112/JOHN/QJVACMSRV
```

```
Interval: 10 seconds
```

```
Total garbage collection cycles prior to running: 29
```

```
Total garbage collection cycles after running: 31
```

```
GC forced: NO
```

```
.....  
. Class loader information .  
.....
```

```
0 Default class loader  
.....
```

```
. GC heap information .  
.....
```

```
Loader
```

	Number of	pass one	objects in the	GC heap	Number of	pass two	objects in the	GC heap	Change in	the number of	objects in the	GC heap	Pass one	object size (K)	Pass two	object size (K)	Change in	object size (K)	In global	registry	Class name
0	431359	491363	60004	18979	21619	2640	NO	java/lang/ String													
0	8	8	0	0	0	0	NO	sun/misc/ URLClass Path\$ JarLoader													
0	4	4	0	0	0	0	NO	java/lang/ Oggetto													
0	7	7	0	0	0	0	NO	java/util/ zip/ Inflater													
0	2	2	0	0	0	0	NO	java/lang/ Thread													
0	2	2	0	0	0	0	NO	[Ljava/lang/ Class;													
0	13	13	0	0	0	0	NO	java/io/ File Descriptor													
0	2	2	0	0	0	0	NO	java/io/ Buffered Writer													
0	4	4	0	0	0	0	NO	[Ljava/lang/ String;													
0	1	1	0	0	0	0	NO	[Ljava/io/ ObjectStream Class\$ ObjectStream ClassEntry;													
0	404	404	0	24	24	0	NO	[Ljava/util/ HashMap\$ Entry;													
0	423	423	0	15	15	0	NO	java/util/ jar/ Attributes\$													

Nome	0	1	1	0	0	0	0	NO	java/io/BufferedInputStream
	0	2	2	0	0	0	0	NO	java/io/BufferedOutputStream
	0	1	1	0	0	0	0	NO	java/security/ProtectionDomain
	0	1	1	0	0	0	0	NO	sun/security/provider/Sun
	0	1	1	0	0	0	0	NO	java/io/Permission
File	0	128	128	0	6	6	0	NO	java/util/Hashtable\$Entry
	0	2	2	0	0	0	0	NO	java/net/URLClassLoader\$ClassFinder
	0	1	1	0	0	0	0	NO	java/lang/Runtime
	0	1	1	0	0	0	0	NO	java/util/BitSet
	0	7	7	0	0	0	0	NO	java/util/jar/JarVerifier
	0	2	2	0	0	0	0	NO	java/lang/ThreadGroup
	0	22	22	0	1	1	0	NO	java/util/Locale
	0	8	8	0	0	0	0	YES	java/io/RandomAccess
File	0	37	37	0	125	125	0	YES	[B
	0	1	1	0	0	0	0	NO	sun/misc/Launcher
	0	871	871	0	117	117	0	YES	[C
	0	1	1	0	0	0	0	NO	sun/misc/Launcher\$Factory
	0	435	435	0	10	10	0	YES	java/lang/Class
	0	1	1	0	0	0	0	NO	java/util/Collections\$EmptyList
	0	1	1	0	0	0	0	NO	java/util/Collections\$EmptyMap
	0	1	1	0	0	0	0	NO	java/lang/String\$CaseInsensitiveComparator
	0	1	1	0	2	2	0	NO	[I
	0	3	3	0	0	0	0	NO	java/lang/OutOfMemoryError
	0	1	1	0	0	0	0	NO	[J
	0	800	800	0	41	41	0	NO	java/util/HashMap\$Entry
	0	1	1	0	0	0	0	NO	java/util/Random
	0	5	5	0	0	0	0	NO	java/security/Access

0	1	1	0	0	0	0	YES	Control Context Reference\$ Reference Handler
0	1	1	0	8	8	0	NO	[S
0	1	1	0	0	0	0	NO	[Ljava/lang/ ref/Soft Reference;
0	1	1	0	0	0	0	NO	java/io/ Stream
Oggetto								Class\$ Compare Member ByName
0	7	7	0	0	0	0	NO	java/util/ jar/ JarFile\$ JarFileEntry
0	1	1	0	0	0	0	NO	java/util/ Collections\$ EmptySet
0	1	1	0	0	0	0	NO	[Ljava/ security/ cert/ Certificate;
0	1	1	0	0	0	0	NO	java/lang/ ref/ Reference Queue
0	1	1	0	0	0	0	NO	java/util/ Hashtable\$ Empty Enumerator
0	1	1	0	0	0	0	YES	sun/misc/ Launcher\$ AppClass Loader
0	1	1	0	0	0	0	NO	java/lang/ Shutdown\$ Lock
0	17	17	0	0	0	0	NO	java/util/ Vector
0	3	3	0	0	0	0	NO	java/util/ Stack
0	17	17	0	1	1	0	NO	java/net/URL
0	21	21	0	1	1	0	NO	java/lang/ ref/ Finalizer
0	1	1	0	0	0	0	NO	java/io/ Os400 FileSystem
0	1	1	0	0	0	0	NO	java/lang/ Runtime Permission
0	1	1	0	0	0	0	NO	[Ljava/io/ File;
0	2	2	0	0	0	0	NO	sun/io/ CharToByte ISO8859_1
0	1	1	0	0	0	0	NO	sun/misc/ Launcher\$ ExtClass Loader
0	1	1	0	0	0	0	NO	java/lang/

								ref/ Soft Reference
0	1	1	0	0	0	0	0	NO sun/security/ provider/ PolicyFile
0	1	1	0	0	0	0	0	NO java/io/ Stream
								Class\$ Compare Class ByName
0	1	1	0	0	0	0	0	NO sun/net/www/ protocol/ file/ Handler
0	8	8	0	0	0	0	0	NO java/util/ jar/ JarFile
0	30	30	0	2	2	0	0	NO java/util/ Hashtable
0	1	1	0	0	0	0	0	NO java/lang/ ref/ Reference\$ Lock
0	2	2	0	0	0	0	0	NO java/io/ PrintStream
0	1	1	0	0	0	0	0	NO java/util/ Hashtable\$ Empty Iterator
0	4	4	0	0	0	0	0	NO java/io/File
0	391	391	0	12	12	0	0	NO java/util/ jar/ Attributes
0	2	2	0	0	0	0	0	NO sun/misc/ URLClassPath
0	2	2	0	0	0	0	0	NO java/io/ FileInput Stream
0	2	2	0	0	0	0	0	NO java/io/ Output Stream Writer
0	11	11	0	0	0	0	0	NO java/util/ ArrayList
0	1	1	0	0	0	0	0	NO java/net/ Unknown Content Handler
0	3	3	0	0	0	0	0	NO java/lang/ ref/ Reference Queue\$Lock
0	2	2	0	0	0	0	0	NO java/io/ FileOutput Stream
0	1	1	0	0	0	0	0	NO sun/misc/ URLClass Path\$ FileLoader
0	31	31	0	5246	5246	0	0	NO [Ljava/lang/ Object;
0	1	1	0	0	0	0	0	NO java/lang/ Class Loader\$ Native

0	2	2	0	0	0	0	NO	Library
0	404	404	0	35	35	0	NO	[Ljava/lang/Thread;
0	2	2	0	0	0	0	NO	java/util/HashMap
0	1	1	0	0	0	0	YES	java/lang/ref/Boolean
0	1	1	0	0	0	0	NO	java/lang/ref/Finalizer\$FinalizerThread
0	1	1	0	0	0	0	NO	sun/security/provider/PolicyPermissions
0	7	7	0	0	0	0	NO	java/util/jar/Manifest
0	2	2	0	0	0	0	NO	sun/net/www/protocol/jar/Handler
0	1	1	0	0	0	0	NO	com/sun/rsajca/Provider
0	1	1	0	0	0	0	NO	java/util/Collections\$ReverseComparator
0	2	2	0	0	0	0	NO	[Ljava/io/ObjectStreamField;
0	1	1	0	0	0	0	NO	java/security/CodeSource
0	34	34	0	5	5	0	NO	[Ljava/util/Hashtable\$Entry;
0	2	2	0	0	0	0	NO	java/lang/ref/ReferenceQueue\$Null
0	2	2	0	0	0	0	NO	java/util/Properties
0	2	2	0	0	0	0	NO	java/util/HashSet
0	1	1	0	0	0	0	NO	[Ljava/lang/ThreadGroup;
0	1	1	0	0	0	0	NO	java/util/HashMap\$EmptyHashIterator
0	7	7	0	0	0	0	NO	java/io/ByteArrayOutputStream

.....
 . Global registry information

Loader	Number of pass one objects in the GC heap	Number of pass two objects in the GC heap	Change in the number of objects in the GC heap	Pass one object size (K)	Pass two object size (K)	Change in object size (K)	Class name
0	8	8	0	0	0	0	java/io/RandomAccessFile
0	8	8	0	64	64	0	[B
0	10	10	0	1	1	0	[C

0	439	439	0	10	10	0	java/lang/Class
0	1	1	0	0	0	0	java/lang/ref/ Reference\$ ReferenceHandler
0	1	1	0	0	0	0	sun/misc/ Launcher\$ AppClassLoader
0	1	1	0	0	0	0	java/lang/ref/ Finalizer\$ FinalizerThread



Esempio: comando CHGJVAPGM (Modifica programma Java)

Per modificare un programma Java^(TM) associato al file di classe, myJavaClassName, utilizzare il comando CHGJVAPGM (Modifica programma Java). I bytecode del file di classe del programma Java vengono interpretati. Per avviare il programma, utilizzare il comando RUNJVA (Esecuzione Java). Il programma Java viene ricreato solo se gli attributi specificati sono differenti rispetto a quelli del programma corrente.

Esempio 1: modificare un programma Java interpretato

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
CHGJVAPGM CLSF('/projectA/team2/myJavaClassName.class')
OPTIMIZE(*INTERPRET)
```

L'esempio seguente è uguale all'Esempio 1, con la differenza che il programma è ottimizzato. Esso viene modificato con OPTIMIZE(10), quindi il programma contiene istruzioni macchina compilate che vengono eseguite all'avvio del programma Java.

Esempio 2: modificare un programma Java ottimizzato in un file JAR

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
CHGJVAPGM CLSF('/projectB/myJavaappfile.jar')
OPTIMIZE(10)
```

Consultare il Comando CHGJVAPGM (Modifica programma Java) per dettagli sul parametro e diagramma della sintassi.

Stringhe del parametro LICOPT (Licensed Internal Code option)

Questa tabella visualizza le stringhe riconosciute dal parametro LICOPT (Licensed Internal Code option). Queste stringhe non sono sensibili al maiuscolo e al minuscolo, ma vengono visualizzate come associazione di caratteri per la lettura.

Stringhe del parametro LICOPT

String	Descrizione
AllFieldsVolatile	Se impostato, considera tutti i campi come volatili.
NoAllFieldsVolatile	Se impostato, nessun campo viene considerato volatile.
AllowBindingToLoadedClasses	Indica che le rappresentazioni della classe temporanea create come risultato delle chiamate defineClass all'interno di una esecuzione su una Java ^(TM) virtual machine possono essere strettamente collegate alle altre rappresentazioni di classe all'interno della stessa JVM.

String	Descrizione
NoAllowBindingToLoadedClasses	Indica che le rappresentazioni di classe temporanea create come risultato delle chiamate <code>defineClass</code> all'interno di una esecuzione su una Java virtual machine non possono essere strettamente collegate alle altre rappresentazioni di classe all'interno della stessa JVM.
AllowClassCloning	Quando più programmi Java vengono creati per un file JAR, consente che le copie delle classi derivate da un programma siano incluse in un codice creato per un altro programma. Facilita inoltre l'allineamento aggressivo.
NoAllowClassCloning	Non consente alle copie delle classi derivate da un programma di essere incluse in un codice creato per un altro programma.
AllowInterJarBinding	Consente uno stretto collegamento a classi esterne alla classe o al file JAR compilato. Facilita le ottimizzazioni aggressive.
NoAllowInterJarBinding	Non consente uno stretto collegamento alle classi esterne alla classe o al file JAR compilato. Questo sostituisce la presenza dei parametri <code>CLASSPATH</code> e <code>JDKVER</code> su <code>CRTJVAPGM</code> .
» AllowMultiThreadedCreate	Indica che <code>CRTJVAPGM</code> esegue l'operazione come al solito, utilizzando solo un sottoprocesso. «
» NoAllowMultiThreadedCreate	<code>CRTJVAPGM</code> utilizza più sottoprocessi, se questi sono disponibili, durante la creazione. «
AnalyzeObjectLifetimes	Esegue l'analisi utilizzando classi visibili per determinare quali oggetti sono di breve durata. Un oggetto di breve durata non dura più a lungo del metodo nel quale è assegnato e può essere soggetto a ottimizzazioni aggressive.
NoAnalyzeObjectLifetimes	Non consente l'analisi di oggetti di breve durata.
AllowBindingWithinJar	Indica che le rappresentazioni di classi all'interno di un file ZIP o JAR possono essere strettamente collegate ad altre rappresentazioni di classi all'interno dello stesso file ZIP o JAR.
NoAllowBindingWithinJar	Indica che le rappresentazioni di classi all'interno di un file ZIP o JAR possono non essere strettamente collegate ad altre rappresentazioni di classi all'interno dello stesso file ZIP o JAR.
AllowInlining	Comunica al programma di conversione che è consentito allineare i metodi locali. Questo è il valore predefinito per i livelli di ottimizzazione 30 e 40.
NoAllowInlining	Non comunica al programma di conversione che è consentito allineare i metodi locali.
AssumeUnknownFieldsNonvolatile	Quando non è possibile determinare gli attributi di un campo in una classe esterna, questo parametro crea un codice creato presupponendo che il campo non sia volatile.
NoAssumeUnknownFieldsNonvolatile	Quando non è possibile determinare gli attributi di un campo in una classe esterna, questo parametro crea un codice creato presupponendo che il campo sia volatile.

String	Descrizione
BindErrorHandling	Specifica quale operazione intraprendere se, come risultato del rispetto dell'opzione LIC (Licenced Internal Code) AssumeUnknownFieldsNonvolatile, PreresolveExtRef o PreLoadExtRef, il programma di caricamento classi Java virtual machine rileva una rappresentazione di classe che contiene rappresentazioni di metodi che non possono essere utilizzate nel contesto corrente.
BindInit	Utilizza chiamate collegate per i metodi init locali.
NoBindInit	Non utilizza chiamate collegate per i metodi init locali.
BindSpecial	Utilizza chiamate collegate per metodi speciali locali.
NoBindSpecial	Non utilizza chiamate collegate per metodi speciali locali.
BindStatic	Utilizza chiamate collegate per metodi statici locali.
NoBindStatic	Non utilizza chiamate collegate per metodi statici locali.
BindTrivialFields	Collega riferimenti di campo di minima entità durante la creazione del programma.
NoBindTrivialFields	Risolve i riferimenti di campo immediatamente.
BindVirtual	Utilizza chiamate collegate per metodi virtuali finali e locali.
NoBindVirtual	Non utilizza chiamate collegate per metodi virtuali finali e locali.
DeferResolveOnClass	Prende un parametro di stringa che si presume essere il nome di una classe (ad esempio, java.lang.Integer). Quando si seleziona PreresolveExtRef sul livello di ottimizzazione 40, le classi che vengono specificate con DeferResolveOnClass non si trovano in fase di preresoluzione. Ciò risulta utile se alcune classi presenti in percorsi non utilizzati nel codice non si trovano nel CLASSPATH. Ciò consente di utilizzare il livello di ottimizzazione 40 senza considerare le suddette classi specificando "DeferResolveOnClass='somepath.someclass'" per ogni classe mancante. Sono consentite più voci DeferResolveOnClass.
DevirtualizeFinalJDK	Consente a CRTJVAPGM di utilizzare la conoscenza di JDK standard per rendere non virtuali le chiamate per quei metodi JDK conosciuti come metodi finali e membri di classi finali. Questo è il valore predefinito sui livelli di ottimizzazione 30 e 40.
NoDevirtualizeFinalJDK	Non consente a CRTJVAPGM di utilizzare la conoscenza di JDK standard per rendere non virtuali le chiamate per quei metodi JDK conosciuti per essere metodi finali e membri di classi finali.
DevirtualizeRecursive	Causa la creazione di un codice speciale nel caso di alcuni metodi ricorrenti ed elimina gran parte del sovraccarico delle chiamate di tali metodi. Tuttavia una logica di controllo ulteriore viene creata sulla voce iniziale per il metodo ricorrente, cosicché le prestazioni non possano aumentare nei casi di bassa ricorrenza.
NoDevirtualizeRecursive	Non causa la creazione di un codice speciale nel caso di alcuni metodi ricorrenti.

String	Descrizione
DisableIntCse	Causa la disabilitazione di alcune ottimizzazioni di sottoespressioni comuni durante la creazione del codice per alcuni tipi di espressioni di numeri interi. Ciò aumenta l'ottimizzazione generale indicando altre opportunità di ottimizzazione al programma di conversione di ottimizzazione.
NoDisableIntCse	Causa la non disabilitazione di alcune ottimizzazioni di sottoespressioni comuni durante la creazione del codice per alcuni tipi di espressioni di numeri interi. Ciò determina generalmente codici di esecuzione migliori a livelli minori di ottimizzazione.
DoExtBlockCSE	Esegue l'eliminazione di sottoespressioni comuni del blocco di base esteso.
NoDoExtBlockCSE	Non esegue l'eliminazione di sottoespressioni comuni del blocco di base esteso.
DoLocalCSE	Esegue l'eliminazione delle sottoespressioni comuni locali.
NoDoLocalCSE	Non esegue l'eliminazione di sottoespressioni comuni locali.
EnableCseForCastCheck	Se impostato, crea un codice per castcheck che può essere DAGEd su un'istanza precedente.
NoEnableCseForCastCheck	Se non impostato, non crea un codice per castcheck che può essere DAGEd su un'istanza precedente.
ErrorReporting	Campo** di notifica di errori nel tempo di esecuzione: Fornisce l'opzione per annullare la compilazione quando si rilevano verifiche o errori nel formato della classe. 0=Notificare immediatamente tutti gli errori; 0=Notificare immediatamente tutti gli errori; 1=Rimandare la notifica di errori di verifica bytecode; 2=Rimandare la notifica di errori di verifica bytecode e di errori nel formato della classe nel tempo di esecuzione.
➤ HideInternalMethods	Rende interni i metodi nelle classi clonate, consentendo l'omissione dei metodi nel momento in cui non esiste alcun riferimento ad essi o se tutti i riferimenti sono allineati. Il valore predefinito è HideInternalMethods per l'ottimizzazione 40 e NoHideInternalMethods per l'ottimizzazione compresa tra 0 e 30. ⚡
InlineArrayCopy	Causa l'allineamento del metodo System.arraycopy in alcuni casi di schiere scalari.
NoInlineArrayCopy	Evita l'allineamento del metodo System.arraycopy.
InlineInit	Allinea i metodi init per le classi java.lang.
NoInlineInit	Non allinea i metodi init.
InlineMiscFloat	Allinea vari metodi float/double derivati da java.lang.Math.
NoInlineMiscFloat	Non allinea metodi vari float/double.
InlineMiscInt	Allinea metodi vari int/long derivati da java.lang.Math.
NoInlineMiscInt	Non allinea metodi vari int/long.
InlineStringMethods	Consente l'allineamento di alcuni metodi derivati da java/lang/String.
NoInlineStringMethods	Inibisce l'allineamento di alcuni metodi derivati da java/lang/String.

String	Descrizione
InlineTransFloat	Allinea metodi transcendental float/double derivati da java.lang.Math.
NoInlineTransFloat	Non allinea metodi transcendental float/double.
OptimizeJsr	Genera codici migliori per i bytecode "jsr" che hanno una destinazione singola.
NoOptimizeJsr	Elimina la creazione di codici migliori per i bytecode "jsr" che hanno una destinazione singola.
PreloadExtRef	Indica che le classi a cui si è fatto riferimento possono essere precaricate (senza inizializzazione della classe) sulla voce del metodo.
NoPreloadExtRef	Indica che le classi a cui si è fatto riferimento non possono essere precaricate sulla voce del metodo. Tuttavia, il parametro PreresolveExtRef sostituisce questa impostazione e determina il precaricamento e l'inizializzazione delle classi a cui si è fatto riferimento.
PreresolveExtRef	Prerisolve i metodi a cui si è fatto riferimento sulla voce del metodo.
NoPreresolveExtRef	Risolve i riferimenti di metodo immediatamente. Risolve le eccezioni "classe non rilevata" sui programmi che sono in esecuzione su altre macchine.
ProgramSizeFactor	Quando un file JAR è abbastanza grande da richiedere più programmi Java, questo valore numerico (valore predefinito 100) viene utilizzato per determinare l'ampiezza che ogni programma può raggiungere.
ShortCktAthrow	Se impostato, tenta interrompere le athrow.
NoShortCktAthrow	Se non impostato, non tenta di interrompere le athrow.
ShortCktExSubclasses	Se impostato, riconosce alcune sottoclassi di Exception e le interrompe direttamente.
NoShortCktExSubclasses	Se non impostato, non riconosce alcune sottoclassi di Exception e non le interrompe direttamente.
StrictFloat	Inibisce le ottimizzazioni a virgola mobile che non sono strettamente compatibili con la specifica Java.
NoStrictFloat	Consente le ottimizzazioni a virgola mobile che non sono strettamente compatibili con la specifica Java.

Il doppio asterisco (**) indica che queste stringhe richiedono un valore numerico per l'immissione nella sintassi di `nomestringa=numero` (con nessun spazio di separazione).

Esempio: comando CRTJVAPGM (Creazione programma Java)

Per creare un programma Java^(TM) ed associarlo al file di classe, `myJavaClassName.class`, utilizzare il comando CRTJVAPGM (Creazione programma Java). Una volta creati con OPTIMIZE(*INTERPRET), i bytecode del file di classe del programma Java vengono interpretati. Per avviare il programma, utilizzare il comando RUNJVA (Esecuzione Java).

Esempio 1: creare un programma Java interpretato

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
CRTJVAPGM CLSF('/projectA/team2/myJavaClassName.class')
OPTIMIZE(*INTERPRET)
```

Questo esempio è uguale all'Esempio 1, con la differenza che il programma è ottimizzato. Esso viene creato con OPTIMIZE(40), quindi il programma contiene istruzioni macchina compilate che vengono eseguite all'avvio del programma Java.

Esempio 2: creare un programma Java ottimizzato

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
CRTJVAPGM CLSF('/projectB/team2/myJavaClassfile.class')
OPTIMIZE(40)
```

Consultare il comando CRTJVAPGM (Creazione programma Java) per dettagli sul parametro e diagramma della sintassi.

Esempio: comando DLTJVAPGM (Cancellazione programma Java)

Il comando DLTJVAPGM (Cancellazione programma JavaTM) cancella il programma Java program associato ad un file di classe specifico denominato myJavaClassName.

Nota: il comando DLTJVAPGM non cancella il file di classe o il file ZIP.

Esempio 1: cancellare un programma Java

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
DLTJVAPGM CLSF('/projectA/team2/myJavaClassName.class')
```

Consultare il Comando DLTJVAPGM (Cancellazione programma Java) per dettagli sul parametro e diagramma della sintassi.

Esempio: comando DMPJVM (Esecuzione dump JVM)

Il comando DMPJVM (Esecuzione dump JavaTM virtual machine) esegue il dump delle informazioni relative alla JVM ovvero Java virtual machine di un lavoro specifico.

Esempio 1: eseguire il dump di una JVM (Java Virtual Machine)

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
DMPJVM JOB(099246/FRED/QJVACMSRV)
```

Il comando DMPJVM esegue il dump delle informazioni per la JVM (Java virtual machine) che è in esecuzione nel lavoro denominato 099246/FRED/QJVACMSRV.

Emissione dell'esempio:

```
JAVA VIRTUAL MACHINE INFORMATION: 099246/FRED/QJVACMSRV
.....
. Classpath
.....
/QIBM/ProdData/Java400/jdk117/lib/jdkptf117.zip:/QIBM/ProdData/Java400/jdk1
17/lib/classes.zip:/QIBM/ProdData/Java400/ext/IBMmisc.jar:/QIBM/ProdData/Ja
va400/ext/db2_classes.jar:/QIBM/ProdData/Java400/ext/jssl.jar:/QIBM/ProdDat
a/Java400/ext/ibmjssl.jar:/QIBM/ProdData/Java400/~/home/fred
.....
. Garbage collection
.....
Garbage collector parameters
  Initial size: 2048 K
  Max size: *NOMAX
Current values
  Heap size: 9476 K
```



```

Garbage collections: 0
.....
. Thread information
.....
Information for 3 thread(s) of 3 thread(s) processed
Thread: 00000001 Thread-0
  TDE: B000200002941000
  Thread priority: 5
  Thread status: Destroy wait
  Thread group: main
  Runnable: java/lang/Thread
Stack:
  None
Locks:
  None
.....
Thread: 00000003 t2
  TDE: B000100005B37000
  Thread priority: 5
  Thread status: Timed wait
  Thread group: main
  Runnable: dbgtest2
Stack:
  java/io/BufferedInputStream.read()I+11 (BufferedInputStream.java:154)
  pressEnter.theFirstMethod(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
  Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
  Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
  Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V+1
  0 (dbgtest2.java:15)
  dbgtest2.run()V+69 (dbgtest2.java:44)
  java/lang/Thread.run()V+11 (Thread.java:466)
Locks:
  None
.....
Thread: 00000002 t1
  TDE: B000100005B33000
  Thread priority: 5
  Thread status: Java wait
  Thread group: main
  Runnable: dbgtest2
Stack:
  pressEnter.theFirstMethod(Ljava/lang/String;Ljava/lang/String;
  Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
  Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
  Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
  Ljava/lang/String;)V+0 (dbgtest2.java:14)
  dbgtest2.run()V+69 (dbgtest2.java:44)
  java/lang/Thread.run()V+11 (Thread.java:466)
Locks:
  None
.....

```

Esempio: comando DSPJVAPGM (Visualizzazione programma Java)

Il comando DSPJVAPGM (Visualizzazione programma Java) visualizza il programma Java associato al file di classe specificato denominato myJavaClassName.

Esempio 1: visualizzare un programma Java

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
DSPJVAPGM CLSF('/projectA/team2/myJavaClassName.class') OUTPUT(*)
```

Consultare il Comando DSPJVAPGM (Visualizzazione programma Java) per dettagli sul parametro e diagramma della sintassi.

Comando JAVA

Il comando JAVA funziona esattamente allo stesso modo del comando RUNJVA (Esecuzione programma JavaTM). E' possibile utilizzarli in modo intercambiabile. Consultare Comando RUNJVA (Esecuzione Java) per informazioni e per i parametri che è possibile utilizzare con il comando JAVA.

Esempio: utilizzare comando RUNJVA (Esecuzione programma Java)

Il comando RUNJAV (Esecuzione JavaTM) esegue il programma Java iSeries associato alla classe.

Esempio 1: eseguire un programma Java

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
RUNJVA CLASS ('/projectA/myJavaclassname')
```

Consultare il comando RUNJVA (Esecuzione Java) per dettagli sul parametro e sul diagramma della sintassi.

Comandi iSeries Navigator supportati da Java

iSeries Navigator è una interfaccia grafica del desktop di Windows^(R). Fa parte di iSeries Access per Windows e copre numerose funzioni iSeries di cui i responsabili e gli utenti hanno bisogno per realizzare il lavoro giornaliero.

iSeries Navigator supporta JavaTM come un modulo aggiuntivo contenuto nell'opzione File System di iSeries Access per Windows. Per utilizzare il modulo aggiuntivo Java di iSeries Navigator, è necessario installare IBM Developer Kit per Java sul server iSeries. Successivamente, per installare il modulo aggiuntivo Java sul PC (personal computer), selezionare File System tramite Installazione selettiva nella cartella Client Access.

I file della classe, JAR, ZIP e Java si trovano nell'integrated file system. iSeries Navigator consente di visualizzare questi file nel pannello a 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 **Associated Java program** → **New...** dal menu di contesto si avvia il programma di conversione Java, il quale crea programmi Java iSeries associati al file di classe, JAR o ZIP. « Una casella di dialogo consente di specificare i dettagli sulla creazione del programma. E' 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.

» Selezionare **Associated Java Program** → **Edit...** del menu di contesto si modificano gli attributi dei programmi Java collegati ai file di classe Java, file ZIP oppure file JAR. «

» Selezionando **Associated Java program** → **Run...** dal menu di contesto viene eseguito il file di classe sul server iSeries. « E' 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 è stato già selezionato **Associated Java program** → **New...**, il programma Java iSeries associato al file della classe viene utilizzato durante l'esecuzione del programma. « Se un programma Java iSeries non è stato già associato al file di classe, allora esso viene creato prima dell'esecuzione del programma.

» Selezionando **Associated Java program** → **Delete...** dal menu di contesto si cancellano i programmi Java iSeries associati al file di classe, JAR o ZIP. «

Selezionando **Properties** dal menu di contesto viene visualizzata una casella di dialogo delle proprietà che contiene i separatori **Java Programs** e **Java Options**. Questi separatori consentono la visualizzazione dei dettagli sul modo in cui i programmi Java iSeries associati sono stati creati per il file di classe, JAR o ZIP.

Nota: questi pannelli rappresentano le informazioni sulla visualizzazione dei programmi Java.

Selezionando **Compile Java file** dal menu di contesto tutti i file java selezionati vengono convertiti nei relativi bytecode della classe.

» Consultare le informazioni di aiuto, incluse con iSeries Navigator, per i parametri e le opzioni delle finestre di dialogo di iSeries Navigator **New Java Program**, **Edit Java program**, **Run Java program**, **Java Programs**, **Java Options**, **Compile Java file** e **Delete Java Program**. «

Capitolo 7. Pacchetti facoltativi

» I pacchetti facoltativi definiscono le API (Application Programming Interface) che estendono l'API principale della piattaforma Java. I seguenti sono pacchetti facoltativi che è possibile utilizzare con IBM Developer Kit per Java^(TM):

JAAS (Java Authentication and Authorization Service)

JAAS (Java Authentication and Authorization Service) consente l'associazione di un utente o di un'identità specifica al sottoprocesso Java corrente.

Java Cryptography Extension

JCE (Java Cryptography Extension) fornisce una framework e implementazioni relative alla codifica, alla creazione e all'accordo di chiavi e agli algoritmi MAC (Message Authentication Code). Inoltre JCE supporta flussi protetti e oggetti sigillati.

Java Naming and Directory Interface

JNDI (Java Naming and Directory Interface) fa parte delle API (application program interface) della piattaforma di JavaSoft. Con JNDI, è possibile collegarsi a più servizi di indirizzario e di denominazione. E' possibile creare applicazioni Java abilitate per l'indirizzario trasferibili e potenti utilizzando questa interfaccia.

Java Secure Socket Layer

JSSL (Java Secure Socket Layer) rappresenta una serie di pacchetti Java che abilitano comunicazioni internet sicure. Questo implementa una versione Java dei protocolli SSL e TLS (Transport Layer Security) e include la funzionalità relativa alla codifica dati, all'autenticazione del server, all'integrità dei messaggi e all'autenticazione del client facoltativo.

JavaMail

L'API JavaMail fornisce una serie di classi astratte che modellano un sistema (e-mail) elettronico. L'API fornisce una framework indipendente dalla piattaforma e dal protocollo per creare applicazioni di messaggeria e di e-mail basate su Java.

JavaPrintService

L'API Java Print Service consente la stampa in tutte le piattaforme Java. Java 1.4 fornisce una framework nella quale i JRE (Java runtime environment) e i terzi possono fornire moduli aggiuntivi di creazione modelli per la produzione di vari formati per la stampa, come ad esempio PDF, Postscript e AFP (Advanced Function Presentation^(TM)) (AFP^(TM)).




JNDI (Java Naming and Directory Interface)


JNDI ovvero Java^(TM) Naming and Directory Interface) fa parte delle API (Application program interfaces) della piattaforma JavaSoft. Con JNDI, è possibile collegarsi, senza connessioni fisiche, a più servizi dell'indirizzario e di denominazione. E' possibile creare applicazioni Java abilitate per l'indirizzario trasferibili e potenti utilizzando questa interfaccia.

JavaSoft ha sviluppato la specifica JNDI con i soci iniziali dell'azienda, inclusi IBM, SunSoft, Novell, Netscape e Hewlett-Packard Co.


Per ulteriori informazioni su JNDI, esaminare Java Naming and Directory interface di Sun Microsystems,

Inc.  Per informazioni specifiche sull'IBM, consultare il manuale Guida di programmazione del fornitore IBM JNDI LDAP.

Guida di programmazione del fornitore IBM JNDI LDAP

» Questa guida di programmazione presuppone che l'utente abbia familiarità con JNDI (Java Naming and Directory Interface ^(TM)) e con la modalità con cui LDAP (Lightweight Directory Access Protocol) opera. Per ulteriori informazioni, consultare la Documentazione JNDI  di Sun Microsystems.

IBM offre un tecnico di manutenzione LDAP per JNDI che può essere utilizzato con SDK o con JRE (Java Runtime Environment) 1.2.2. L'utilizzo di IBM JNDI LDAP provider con SDK o JRE 1.3 o con versioni successive non è supportato; al contrario, è necessario utilizzare JNDI e JNDI LDAP provider di Sun Microsystems, Inc., che sono parte di SDK e JRE 1.3. E' possibile inoltre utilizzare il fornitore LDAP JNDI di Sun Microsystems, Inc. con SDK e JRE 1.2.2, ma è necessario scaricare questi componenti dal sito

web Sun Microsystems, Inc. JNDI  e il supporto viene fornito da Sun. Questa guida di programmazione descrive l'utilizzo del fornitore IBM JNDI LDAP con SDK o JRE 1.2.2.

Per compilare o eseguire il codice utilizzando il fornitore IBM JNDI LDAP, aggiungere quanto segue al percorso della classe:

```
/QIBM/ProdData/Java400/ext/ibmjndi.jar:/QIBM/ProdData/Java400/ext/jndi.jar
```

Questa guida di programmazione tratta i seguenti argomenti:

Creare un contesto iniziale

Questo argomento descrive il modo in cui è possibile creare un contesto iniziale per collegarsi a un server LDAP. JNDI supporta due modi per un client di gestire un server LDAP (Lightweight Directory Access Protocol):

- Il client identifica il server durante la creazione del contesto.
- Una stringa URL viene inviata direttamente ai metodi del contesto.

LDAP V3 URL

Questo argomento definisce la sintassi URL LDAP.

Collegamento al server e supporto SASL

E' necessario che un server autentichi un client prima che siano consentite alcune operazioni. LDAP fa riferimento a questo durante il collegamento al server. Il protocollo LDAP ha esteso l'autenticazione per supportare anche i meccanismi SASL (Simple Authentication and Security Layer). Questi meccanismi consentono modi più sofisticati di identificazione del client sul server senza compromettere inutilmente la sicurezza dell'utente inviando l'ID e la parola d'ordine in testo chiaro.

Ricerca e ottenere attributi

JNDI fornisce flessibilità nella ricerca degli indirizzari LDAP (Lightweight Directory Access Protocol).

Aggiungere e cancellare le voci in un indirizzario

JNDI consente di aggiungere e cancellare le voci in un indirizzario. Questo argomento include esempi sul modo in cui eseguire queste attività.

Modificare attributi

JNDI rende possibile la modifica, la creazione o l'eliminazione degli attributi da una voce di indirizzario.

Ridenominare una voce di indirizzario

JNDI consente la ridenominazione di una voce di indirizzario dovunque in relazione al contesto di base. Questo argomento identifica una proprietà che interessa il metodo di ridenominazione utilizzato per ridenominare una voce di indirizzario.

Riferimenti e riferimenti di ricerca

E' possibile che i server LDAP restituiscano sia riferimenti che riferimenti di ricerca. Un riferimento può essere restituito su qualsiasi operazione e indica che il server non congela la voce di destinazione della richiesta. Un riferimento di ricerca è restituito solo su un'operazione di ricerca.

Controlli LDAP

La specifica LDAP v3 ha aggiunto i controlli per inviare e ricevere informazioni sull'estensione. I controlli inviati a un server vengono indicati con il nome di controlli di richiesta. I controlli ricevuti da un server sono indicati con il nome di controlli di risposta.

Attributi binari

Il protocollo LDAP non fornisce alcuna distinzione tra gli attributi testuali e binari reperiti. Al contrario, esso presuppone che l'applicazione del client conosca il modo in cui gestire i dati. Questo argomento descrive tre metodi diversi per gestire ciò che accade agli attributi reperiti e per gestire la possibilità che questi siano convertiti a stringhe.

Schema

E' possibile reperire, visualizzare e aggiornare la struttura dello schema di un server LDAP. Sono supportati solo i server che forniscono informazioni sullo schema definite dalla specifica LDAP (Lightweight Directory Access Protocol).

Modulo aggiuntivo SASL

E' possibile registrare il modulo aggiuntivo SASL (Simple Authentication and Security Layer). Questo argomento fornisce esempi di codice per consentire informazioni preliminari riguardo alla creazione del modulo aggiuntivo.

Memorizzazione nella cache dal lato del client

La memorizzazione nella cache fornisce uno strumento per memorizzare informazioni richieste recentemente in maniera locale. Questo migliora le prestazioni reperendo interrogazioni ripetute localmente invece di fare riferimento a un server remoto per informazioni già ottenute.

Richiamare la versione della classe IBMJNDI

Questo argomento indica quale metodo static è possibile utilizzare per reperire la versione della classe IBMJNDI di LDAP.

Considerazioni di conformità e proprietà aggiuntive

Consultare questo argomento per conoscere quali considerazioni è necessario tenere presente quando si utilizza IBM LDAP provider e Jndi Implementor Guidelines for LDAP Service Providers di Sun. Sono indicate inoltre le proprietà che sono supportate e le proprietà che non sono state accettate.



Creare un contesto iniziale

» La JNDI (Java Naming and Directory Interface^(TM)) supporta due modi diversi per un client di gestire un server LDAP (Lightweight Directory Access Protocol). Il primo e più comune risulta per il client quello di identificare il server durante la creazione del contesto. Le operazioni vengono allora eseguite su questo collegamento aperto inviando nomi basati su DN ai metodi del contesto. Le seguenti due proprietà supportano questo tipo di operazione:

java.naming.factory.initial (Context.INITIAL_CONTEXT_FACTORY)

E' necessario che questa proprietà sia impostata su `com.ibm.jndi.LDAPCtxFactory`.

java.naming.provider.url (Context.PROVIDER_URL)

Questa proprietà identifica il nome e la porta del server LDAP nel formato di una stringa URL. Se il nome e la porta del server LDAP non possono essere identificati, IBM JNDI LDAP provider viene impostato sul valore predefinito di `ldap://hostlocale:389`.

Il codice seguente crea un collegamento all'host serverldap e reperisce una voce:

Esempio 1: creare un collegamento all'host serverldap

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
Properties env = new Properties();
env.put("java.naming.factory.initial", "com.ibm.jndi.LDAPCtxFactory");
env.put("java.naming.provider.url", "ldap://serverldap");
DirContext ctx = new InitialDirContext(env);
Attributes entry = ctx.getAttributes("cn=example,o=IBM,c=US");
```

Il secondo modo per gestire un server LDAP utilizzando JNDI è quello di inoltrare una stringa URL direttamente ai metodi di contesto. Questo processo tuttavia ha il sovraccarico di creare un nuovo collegamento per ogni operazione e dovrebbe essere evitato se tutte le operazioni sono collegate a un server singolo. La seguente proprietà supporta questo tipo di operazione:

java.naming.factory.url.pkgs (Context.URL_PKG_PREFIXES)

E' necessario impostare questa proprietà su com.ibm.jndi se le stringhe URL vengono inoltrate come immissione di nome a un metodo di controllo e se il collegamento a un server LDAP non è necessario durante la creazione del contesto.

Il codice seguente duplica l'esempio precedente, ma ritarda il collegamento al server finché il metodo getAttributes non è chiamato:

Esempio 2: creare un collegamento all'host serverldap e ritardare il collegamento al server.

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
Properties env = new Properties();
env.put("java.naming.factory.url.pkgs", "com.ibm.jndi");
DirContext ctx = new InitialDirContext(env);
Attributes entry = ctx.getAttributes("ldap://serverldap/cn=esempio,o=IBM,c=US");
```

Il fornitore supporta inoltre l'associazione dei due metodi precedenti. E' possibile cioè stabilire un collegamento a un server LDAP utilizzando java.naming.factory.initial e successivamente inoltrare una stringa URL come un'immissione di nome relativo a un metodo. Questo funziona sia se java.naming.factory.url.pkgs viene definito o meno.

Nota: qualsiasi collegamento al server LDAP aperto da **InitialDirContext** deve essere chiuso chiamando il metodo close. <<

URL LDAP V3

» IBM JNDI (Java Naming and Directory InterfaceTM) LDAP (Lightweight Directory Access Protocol) provider supporta completamente il formato URL (Uniform Resource Locator) LDAP definito in RFC 2255

 . Un URL LDAP viene definito dalla seguente sintassi:

```
schema "://" [host [ ":" port ]] [ "/"
[dn ["?" [attributes] ["?" [scope]
["?" [filter] ["?" extensions]]]]]]
```

Dove:

- *schema* indica lo schema URL. Questa libreria di classi supporta il tradizionale ldap per i collegamenti LDAP normali oppure gli ldaps per i collegamenti SSL (Secure Socket Layer).
- *host* è il nome del server LDAP. Se il nome del server LDAP non viene specificato, il valore predefinito è localhost.
- *port* rappresenta il numero della porta del server LDAP. Se il numero della porta non viene specificato, il valore predefinito è 389 per non-SSL e 636 per SSL.
- *dn* identifica l'oggetto di base per l'operazione.

- *attributes* rappresenta una lista di attributi separati dalla virgola che deve essere restituita. Se una lista di attributi non è specificata, il valore predefinito è quello di restituire tutti gli attributi.
- *scope* rappresenta l'ambito della ricerca. I valori validi relativi a questi file sono i seguenti:
 - **base**
Questo valore rappresenta l'oggetto di base.
 - **sub**
Questo valore rappresenta il sottordine di una gerarchia di file.
 - **one**
Questo valore rappresenta un livello della gerarchia di file.

Se l'ambito non è specificato, il valore predefinito è base.

- *filter* rappresenta il filtro della ricerca. Se il filtro della ricerca non è specificato, il valore predefinito è (objectclass=*).
- *extension* fornisce all'URL LDAP un meccanismo di estensibilità, consentendo alle funzioni dell'URL di essere estese. L'unica estensione supportata da IBM JNDI LDAP provider è bindname.

Collegamento al server e supporto SASL

➤ In molti casi, è necessario che un server autentichi un client prima che vengano consentite alcune operazioni. LDAP (Lightweight Directory Access Protocol) si riferisce a questo come collegamento al server.

Quando si collega al server, il client specifica quale protocollo LDAP desidera utilizzare. Esistono due versioni del protocollo LDAP definito, V2 e V3. Se il server supporta solo V2, viene restituito un errore di protocollo quando un client tenta di collegarsi come un client V3. IBM JNDI (Java Naming and Directory Interface^(TM)) LDAP provider supporta il collegamento o al client V2 o V3.

E' possibile utilizzare le seguenti proprietà durante il collegamento a un server:

java.naming.ldap.version

Questa proprietà specifica la versione del protocollo LDAP. I valori validi sono 2 o 3. Se questa proprietà non è impostata, il fornitore tenta di collegarsi come un client V3 e successivamente scende a V2 se viene restituito un errore di protocollo. Il fornitore non tenta alcuna modifica se questa proprietà viene impostata.

Oltre ad impostare una versione di protocollo, un collegamento identifica anche un utente sul server per l'autenticazione.

java.naming.security.principal (Context.SECURITY_PRINCIPAL)

Questa proprietà specifica l'ID del client. In quasi tutti i casi, si trova nel formato di un DN (distinguished name).

java.naming.security.credentials (Context.SECURITY_CREDENTIALS)

Questa proprietà specifica le credenziali del client (cioè la parola d'ordine del client).

LDAP supporta inoltre tipi diversi di meccanismi di autenticazione. Il protocollo LDAP V2 supporta solamente un tipo di collegamento riferito come collegamento semplice. Con questo meccanismo, vengono inviati al server l'ID e le credenziali in testo chiaro. Il protocollo V3 estende l'autenticazione in modo tale da supportare anche i meccanismi SASL (Simple Authentication and Security Layer). Questi meccanismi consentono modi più sofisticati di identificazione del client sul server senza compromettere inutilmente la sicurezza dell'utente inviando l'ID e la parola d'ordine in testo chiaro.

Il fornitore supporta due modi diversi di specificare il meccanismo di autenticazione. Un modo richiede il nome classe di autenticazione. Questo modo consente al fornitore di essere esteso specificando le classi di autenticazione al di fuori del fornitore. Così, è possibile registrare il proprio modulo SASL. Le seguenti proprietà supportano questo modo di specificare il meccanismo di autenticazione.

java.naming.security.sasl

Questa proprietà specifica il nome classe di autenticazione da utilizzare. Le classi seguenti vengono inviate come parte del fornitore.

com.ibm.Idap.LDAPSimpleBind

Questa proprietà specifica l'ID e le credenziali in testo chiaro, inviati al server per l'autenticazione. Questo meccanismo viene supportato sia dai server V2 che V3. Si noti che l'utilizzo di parole d'ordine di testi semplici non è consigliato su reti aperte quando non esistono autenticazioni e codifiche eseguite da un livello inferiore.

com.ibm.Idap.LDAPSaslExternal

Il metodo SASL esterno ricerca il collegamento utilizzando il protocollo di sicurezza sottostante già negoziato come ad esempio SSL. Nella maggior parte dei casi, il principal e le credenziali della sicurezza non devono essere lasciati inizializzati.

com.ibm.Idap.LDAPSaslCRAM_MD5

Il SASL CRAM-MD5 invia il principal e le credenziali di sicurezza al server per l'autenticazione utilizzando un protocollo di risposte alle domande di chiarificazione.

com.ibm.Idap.LDAPSaslGSSAPI

Il metodo SASL GSSAPI ricerca il collegamento utilizzando l'autenticazione Kerberos dopo aver ottenuto le credenziali attraverso uno strumento separato come il kinit o il collegamento integrato. Nella maggior parte dei casi, il principal e le credenziali della sicurezza che collegano argomenti non devono essere lasciati inizializzati.

java.naming.sasl.mode

Questa proprietà specifica un'impostazione di modalità inviata al modulo aggiuntivo SASL caricato. Questa impostazione viene ignorata da tutti i moduli SASL predefiniti nel fornitore.

Il secondo modo supportato di specificare il meccanismo di autenticazione è compatibile al toolkit LDAP di Sun. Invece di indicare la classe di autenticazione da caricare, viene specificato il nome del meccanismo di autenticazione. IBM JNDI LDAP provider utilizza questo metodo per specificare il meccanismo di autenticazione se la proprietà `java.naming.security.sasl` non è impostata.

java.naming.security.authentication (Context.SECURITY_AUTHENTICATION)

Questa proprietà specifica il nome del meccanismo di autenticazione da utilizzare. I valori seguenti sono supportati da questa proprietà.

- nessuno
Non si esegue alcuna autenticazione (collegamento anonimo).
- semplice
Utilizza un'autenticazione semplice.
- EXTERNAL
Utilizza un meccanismo SASL esterno.
- CRAM-MD5
Utilizza un meccanismo SASL CRAM-MD5.
- GSSAPI
Utilizza un meccanismo SASL GSS o Kerberos.

La seguente proprietà è impostata da questa libreria classi dopo che il client è stato autenticato regolarmente:

java.naming.authorization.identity

Questa proprietà è impostata per l'identità dell'autorizzazione del client. Normalmente questa risulta uguale all'ID del client specificato. Tuttavia, il meccanismo SASL può definire il DN di collegamento iniziale su un altro valore. Un esempio di ciò è il SASL esterno laddove l'identità dell'autorizzazione è memorizzata nel certificato del client.

L'esempio seguente dimostra come impostare le proprietà in modo tale da indicare che il protocollo della versione sia 3 e sia autenticato da Larry Meade tramite il meccanismo CRAM-MD5:

Esempio: impostare le proprietà

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
Properties env = new Properties();
env.put("java.naming.factory.initial", "com.ibm.jndi.LDAPCtxFactory");
env.put("java.naming.ldap.version", "3");
env.put("java.naming.provider.url", "ldap://serverldap");
env.put(Context.SECURITY_PRINCIPAL, "cn=Larry Meade, o=IBM, c=US");
env.put(Context.SECURITY_CREDENTIALS, "secret");
env.put(Context.SECURITY_AUTHENTICATION, "CRAM-MD5");
DirContext ctx = new InitialDirContext(env);
```

L'esempio precedente può essere modificato per indicare esplicitamente il nome classe SASL. La riga SECURITY_AUTHENTICATION nell'esempio precedente verrà sostituita dalla seguente riga:

```
env.put("java.naming.security.sasl", "com.ibm.ldap.LDAPSaslCRAM_MD5");
```

Ricerca e ottenere attributi

➤ La JNDI (Java Naming and Directory Interface^(TM)) fornisce una grande flessibilità nella ricerca di indirizzari LDAP (Lightweight Directory Access Protocol). I due metodi utilizzati più frequentemente in IBM JNDI LDAP provider sono search e getAttributes. Anche i seguenti metodi, tuttavia, reperiscono dati da un server LDAP:

- lookup
- lookupLink
- list
- listBindings
- getSchema
- getSchemaClassDefinition

Le seguenti proprietà interessano le operazioni di ricerca:

java.naming.ldap.derefAliases

Questa proprietà definisce il modo in cui gli oggetti alias (come definito in X.501) devono essere gestiti. Questa proprietà accetta i valori seguenti:

- always
Questo valore rimuove ogni riferimento agli alias durante la ricerca e l'ubicazione dell'oggetto di base della ricerca. Questo è il valore predefinito.
- never
Questo valore non rimuove riferimenti agli alias durante la ricerca o nell'ubicazione dell'oggetto di base della ricerca. Per aumentare le prestazioni, questa è l'impostazione consigliata.
- finding
Questo valore rimuove i riferimenti agli alias durante l'ubicazione dell'oggetto di base della ricerca, ma non durante la ricerca di subordinati dell'oggetto di base.
- searching
Questo valore rimuove i riferimenti agli alias nei subordinati dell'oggetto di base nella ricerca, ma non nell'ubicazione dell'oggetto di base della ricerca.

java.naming.batchsize (Context.BATCHSIZE)

Questo valore imposta il limite di dimensione consigliato per il numero di risultati di ricerca congelato da una NamingEnumeration restituita. Se il valore non viene specificato, allora la dimensione batch predefinita è 1. Questo assicura il minore utilizzo di memoria da parte della libreria di classi. Un valore di 0 disabilita la dimensione batch e indica che una ricerca è interrotta finché tutti i risultati non vengono raccolti.

java.naming.ldap.typesOnly

Questa proprietà è rilevante per i metodi `getAttributes` e `search`, l'ultimo dei quali viene utilizzato solamente quando l'indicatore dell'oggetto in restituzione è su "false". Questa proprietà accetta i valori seguenti:

- `true`
Questo valore restituisce solo identificativi di attributi, ma non i valori.
- `false`
Questo valore restituisce sia gli identificativi di attributi che i valori. Questo è il valore predefinito.

Il risultato di una chiamata di ricerca è una `NamingEnumeration`. Per ottenere i risultati, l'enumerazione deve essere scorsa utilizzando i metodi tradizionali `hasMoreElements` e `nextElement` oppure i metodi `NamingEnumeration`-specific `hasMore` e `next`. Gli ultimi due metodi consentono la cattura di eccezioni se si desidera prendere in considerazione e visualizzare una `ReferralException`.

Note:

- per evitare situazioni inaspettate in cui le risorse vengono assegnate e i collegamenti aperti, è necessario scorrere una `NamingEnumeration` fino alla fine (cioè fino a quando i metodi `hasMore` e `hasMoreElements` non restituiscono "false") oppure è necessario richiamare il metodo `close` dell'enumerazione.
- IBM JNDI LDAP provider sostituisce automaticamente le codifiche di carattere UTF-8 non valide nei DN (distinguished name) e nei nomi del tipo di attributo con un carattere *. Questa operazione viene eseguita per evitare che un singolo valore non valido possa causare la mancata riuscita di una intera e possibilmente lunga ricerca.

L'esempio seguente ricerca tutte le voci in cui il cognome sia smith e indica il ritorno solo dell'attributo `cn`:

Esempio: ricercare e ottenere l'attributo `cn`

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
SearchControls constraints = new SearchControls();
constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
String attrList[] = {"cn"};
constraints.setReturningAttributes(attrList);
NamingEnumeration results =
    ctx.search("o=IBM,c=US", "(sn=smith)", constraints);
while (results.hasMore()) {
    SearchResult si =(SearchResult)results.next();
    System.out.println(si.getName());
    Attributes attrs = si.getAttributes();
    if (attrs == null) {
        System.out.println("  No attributes");
        continue;
    }
    NamingEnumeration ae = attrs.getAll();
    while (ae.hasMoreElements()) {
        Attribute attr =(Attribute)ae.next();
        String id = attr.getID();
        Enumeration vals = attr.getAll();
        while (vals.hasMoreElements())
            System.out.println("  "+id + ": " + vals.nextElement());
    }
}
```

L'esempio seguente utilizza il metodo `list` per visualizzare i nomi sotto un DN (distinguished name) di base:

Esempio: visualizzare i nomi sotto un DN di base

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
String url="ldap://serverldap:389/o=IBM,c=US";
NamingEnumeration listResults=ctx.list(url);
while (listResults.hasMore()) {
    NameClassPair ncp = (NameClassPair) listResults.next();
    System.out.println(ncp.getName());
}
```



Aggiungere e cancellare le voci in un indirizzario

» JNDI (Java Naming and Directory Interface[™]) consente l'aggiunta e la cancellazione di voci in un indirizzario. L'esempio seguente aggiunge una nuova voce con gli attributi classe di oggetti, numero di stanza e numero di telefono:

Esempio: aggiungere una voce in un indirizzario

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
BasicAttribute objClasses = new BasicAttribute("objectclass");
objClasses.add("person");
objClasses.add("organizationalPerson");
objClasses.add("inetOrgPerson");

BasicAttributes attrs = new BasicAttributes();
attrs.put(objClasses);
attrs.put("roomnumber", "2000");
attrs.put("telephonenumber", "1-800-use-LDAP");

ctx.createSubcontext(name, attrs);
```

L'esempio seguente elimina una voce:

Esempio: cancellare una voce in un indirizzario

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
ctx.destroySubcontext(name);
```

Modificare attributi: JNDI rende possibile la modifica, la creazione o l'eliminazione di attributi da una voce di indirizzario. Il seguente esempio sostituisce l'attributo numero di stanza di una voce:

Esempio: modificare un attributo

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
ctx.modifyAttributes(name,
    DirContext.REPLACE_ATTRIBUTE,
    new BasicAttributes("roomnumber", "5000"));
```

L'esempio seguente aggiunge un valore dell'attributo numero telefonico nuovo a una voce ed elimina l'attributo numero di stanza:

Esempio: modificare una voce da un indirizzario

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
ModificationItem[] mods=new ModificationItem[2];
mods[0] = new ModificationItem(DirContext.ADD_ATTRIBUTE,
    new BasicAttribute("telephonenumber", "456-7777"));
mods[1] = new ModificationItem(DirContext.REMOVE_ATTRIBUTE,
    new BasicAttribute("roomnumber"));
ctx.modifyAttributes(name, mods);
```

Ridenominare una voce dell'indirizzario: E' possibile utilizzare il metodo rename per ridenominare una voce di indirizzario dovunque in relazione al contesto di base.

La seguente proprietà interessa il metodo rename:

java.naming ldap.deleteRDN

Questa proprietà elimina il vecchio RDN quando la voce viene ridenominata. L'impostazione predefinita è impostata su "true".

Se la proprietà è impostata su "false", il vecchio RDN viene conservato come un valore di attributo della voce.

Il seguente è un esempio di richiamo del metodo rename:

Esempio: ridenominare una voce di indirizzario

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
String oldname="cn=bill smith";
String newname="cn=bill smith, ou=programmer";
ctx.rename(oldname, newname);
```



Riferimenti e riferimenti di ricerca

» I server LDAP (Lightweight Directory Access Protocol) possono restituire sia riferimenti che riferimenti di ricerca. Un riferimento può essere restituito su qualsiasi operazione e indica che il server non congela la voce di destinazione della richiesta. Un riferimento di ricerca viene restituito solamente su un'operazione di ricerca. Il riferimento di ricerca indica che il server è stato in grado di ubicare la voce a cui viene fatto riferimento da baseObject, ma non è stato in grado di ricercare tutte le voci nell'ambito e sotto il baseObject. Un server può restituire uno o più riferimenti di ricerca.

E' possibile configurare un contesto per gestire i riferimenti e i riferimenti di ricerca in uno di questi tre modi:

1. E' possibile impostare il contesto in modo tale che questo segua automaticamente il riferimento ed esegua l'operazione sul server di riferimento. IBM JNDI (Java Naming and Directory Interface^(TM)) LDAP provider riconosce automaticamente ed evita i loop di riferimento; in sostanza ciò si verifica in quelle situazioni in cui un riferimento si ricollega a uno già trattato in precedenza nella catena.
2. E' possibile impostare il contesto in modo tale che questo lanci una ReferralException quando viene ricevuto un riferimento o un riferimento di ricerca. Ciò è utile se la gestione automatica è in qualche modo insufficiente; ad esempio, quando ogni server richiede un collegamento diverso.
3. E' possibile impostare il contesto in modo tale che questo ignori il riferimento e continui come se nulla fosse successo. Nel caso di un riferimento di ricerca, solo le voci rilevate sul server di origine vengono restituite.

Le seguenti proprietà di ambiente sono definite da IBM JNDI LDAP provider per gestire i riferimenti e i riferimenti di ricerca:

java.naming.referral (Context.REFERRAL)

Questa proprietà è impostata sia su follow, throw o ignore. Se la proprietà non è impostata, si imposta quella predefinita per seguire automaticamente i riferimenti.

java.naming ldap.referral.limit

Questa proprietà definisce il numero di hop di riferimento che la libreria di classi crea durante la ricerca di riferimenti. Se il valore della proprietà non viene specificato, il valore predefinito è 10.

java.naming ldap.referral.bind

Se questa proprietà è impostata su "true", le classi, quando seguono automaticamente i riferimenti, si collegano a qualsiasi server riferito utilizzando lo stesso meccanismo e le stesse

credenziali del contesto di origine. Se la proprietà viene impostata su "false", le classi non si collegano (il che significa, accesso anonimo). La funzionalità predefinita richiede il collegamento.

Le regole seguenti si applicano durante la cattura di riferimenti e di riferimenti di ricerca.

- Se il riferimento contiene una porta, allora viene utilizzato. Altrimenti, viene utilizzata la porta derivata dal collegamento principale.
- Viene mantenuto il tipo di collegamento di sicurezza derivato dal collegamento principale. In sostanza, se il collegamento principale si trova su SSL, allora anche tutti i riferimenti considerati si trovano su SSL.

Nota: è possibile che un contesto lanci ancora una `ReferralException` anche quando è impostato su `follow`. Questo si verifica se il limite hop di riferimento viene superato o se non è possibile che il contesto si connetta o si colleghi a uno qualsiasi dei server riferiti.

L'esempio seguente cattura e visualizza un riferimento e un riferimento di ricerca su una richiesta di ricerca. Per ulteriori informazioni sulla gestione delle eccezioni di riferimento, consultare `ReferralException`



di Sun Microsystems, Inc.

Esempio: catturare e visualizzare un riferimento e riferimenti di ricerca

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
ctx.addToEnvironment(ctx.REFERRAL, "throw");
try {
    NamingEnumeration results = ctx.search(url);
    while (true) {
        try {
            if (!results.hasMore())
                break;
            SearchResult si =(SearchResult) results.next();
            System.out.println(si.getName());
        } catch (ReferralException re) {
            System.out.println("Reference caught");
        }
        do {
            System.out.println(re.getReferralInfo());
        } while (re.skipReferral());
    }
} catch (ReferralException re) {
    System.out.println("Referral caught");
    do {
        System.out.println(re.getReferralInfo());
    } while (re.skipReferral());
}
```

Controlli LDAP

» La specifica LDAP (Lightweight Directory Access Protocol) ha aggiunto controlli per inviare e ricevere informazioni sull'estensione. I controlli forniti a un server sono indicati con il nome di **controlli di richiesta**, un esempio dei quali sarebbe la richiesta a un server volta a ordinare i risultati tramite un attributo specificato. Se i controlli vengono supportati o meno dipende solamente dal server (cioè, è possibile eseguire un controllo su un tipo di server, ma lo stesso controllo può avere esito negativo su un altro server). I controlli ricevuti da un server sono indicati con il nome di **controlli di risposta**.

La specifica 1.2 di JNDI (Java Naming and Directory Interface ^(TM)) separa i controlli di richiesta in due categorie distinte, quelle utilizzate durante il collegamento a un server e quelle utilizzate in tutte le altre operazioni. Per ulteriori informazioni sui controlli di richiesta, fare riferimento a `LdapContext` nella documentazione JNDI di Sun.

IBM JNDI LDAP provider dispone di un controllo predefinito denominato ManageDsaIT. Questo controllo forza il server a trattare i riferimenti di ricerca come voci LDAP normali, consentendo a questi di essere visualizzati e modificati al posto dei dati a cui si riferiscono. L'esempio seguente dimostra come abilitare questo controllo:

Esempio: abilitare il controllo ManageDsaIT

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import com.ibm.jndi.ldap.control.ManageDsaIT;

Control[] cntl = new Control[1];
cntl[0] = new ManageDsaIT();
ctx.setRequestControls(cntl);
```

I controlli di collegamento sono attivi solo quando il client si sta collegando a un server.

Il seguente è un esempio di come abilitare i controlli di collegamento:

```
LdapContext ctx = new InitialLdapContext(env, cntl);
```

L'API seguente viene utilizzata per reperire gli ultimi controlli di risposta ricevuti:

```
Control[] cntl = ctx.getResponseControls();
```

Fare riferimento alla documentazione di Sun per dettagli su come rendere le produzioni di controllo capaci di mettere in corrispondenza i dati di controllo grezzi con le classi di controllo. <<

Attributi binari

>> Il protocollo LDAP (Lightweight Directory Access Protocol) non fornisce alcuna distinzione tra gli attributi testuali e binari reperiti. Al contrario, esso presuppone che l'applicazione del client conosca il modo in cui gestire i dati. IBM JNDI LDAP provider fornisce un aiuto perché converte gli attributi testuali e li restituisce come stringhe Java. Tuttavia, il fornitore deve sapere quali attributi sono binari e quali attributi rappresentano i dati del carattere. Il fornitore supporta tre metodi diversi per gestire ciò che accade agli attributi reperiti e per gestire la possibilità che questi siano convertiti in stringhe.

Quando un attributo viene reperito, il fornitore controlla una lista di nomi di attributi binari conosciuti. Il fornitore è programmato per riconoscere la serie seguente di attributi binari LDAP comuni:

- userPassword
- userCertificate
- cACertificate
- authorityRevocationList
- certificateRevocationList
- deltaRevocationList
- crossCertificatePair
- x500UniqueIdentifier
- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedObject
- thumbnailPhoto
- thumbnailLogo
- supportedAlgorithms
- protocolInformation

E' possibile specificare la lista dei nomi degli attributi binari utilizzando la seguente proprietà:

java.naming.ldap.attributes.binary (LDAPCtx.ATTRIBUTES_BINARY)

Una lista separata da spazi relativa ai nomi di attributi binari definiti dall'utente in aggiunta alla serie predefinita indicata dal fornitore.

L'esempio seguente identifica due ulteriori attributi binari definiti dall'utente:

Esempio: specificare una lista di nomi di attributi binari


Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
ctx.addToEnvironment(LDAPCtx.ATTRIBUTES_BINARY,
    "gifPhoto fingerPrint");
```

Il secondo metodo per la gestione degli attributi binari consiste nel riconoscimento dell'opzione di descrizione binaria supportata da alcuni server V3. Quando ";binary" viene accodato al nome dell'attributo così come "jpegPhoto;binary", un valore binario viene designato. Il fornitore riconosce un'opzione di descrizione dell'attributo binario quando è collegato come un utente V3.


Infine il fornitore tenta di convertire qualsiasi attributo che non è definito come binario e che non contiene la relativo opzione di descrizione. Se la conversione ha esito negativo, i dati vengono restituiti come binari. Tuttavia non bisogna fare affidamento su questa operazione perché è possibile che siano restituiti dati non UTF-8 convertiti in modo erroneo. <<

Schema

>> E' possibile reperire, visualizzare e aggiornare la struttura dello schema di un server LDAP (Lightweight Directory Access Protocol). Sono supportati solo i server che forniscono informazioni sullo schema definite dal documento Lightweight Directory Access Protocol (v3) .

Per reperire lo schema di un server, utilizzare il metodo getSchema. Lo schema restituito viene rappresentato come una gerarchia, in cui ogni livello subordinato risulta un componente diverso dello schema. IBM JNDI (Java Naming and Directory InterfaceTM) LDAP provider è in grado di analizzare i componenti dello schema seguente:

- Tipi di attributi, memorizzati all'interno del sottordine AttributeDefinition
- Classi di oggetti, memorizzate all'interno del sottordine ClassDefinition
- Descrizioni di sintassi, memorizzate all'interno del sottordine SyntaxDefinition
- Regole di corrispondenza, memorizzate all'interno del sottordine MatchingRule
- Tipi di attributi IBM, memorizzati all'interno del sottordine IBMAttributeDefinition

Il contenuto delle voci, escluso IBMAttributeDefinition, ha una corrispondenza univoca con lo schema definito in Lightweight Directory Access Protocol (v3): Definizioni di sintassi dell'attributo .

IBMAttributeDefinition estende uno schema di attributo in modo tale da congelare le informazioni specifiche di IBM. Viene definito con il seguente formato BFN (Backus-Naur form):

```
IBMAttributeTypesDescription = "(" whsp
    numericoid whsp
    [ "DBNAME" qdescrs ] ; at most 2 names (table, column)
    [ "ACCESS-CLASS" whsp IBMAccessClass whsp ]
    [ "LENGTH" wlen whsp ] ; maximum length of attribute
    [ "EQUALITY" [ IBMwlen ] whsp ] ; create index for matching rule
    [ "ORDERING" [ IBMwlen ] whsp ] ; create index for matching rule
    [ "APPROX" [ IBMwlen ] whsp ] ; create index for matching rule
    [ "SUBSTR" [ IBMwlen ] whsp ] ; create index for matching rule
    [ "REVERSE" [ IBMwlen ] whsp ] ; reverse index for substring
whsp ")"
```

```

IBMAccessClass =
    "NORMAL"           / ; questo è il valore predefinito
    "SENSITIVE"       /
    "CRITICAL"        /
    "RESTRICTED"     /
    "SYSTEM"          /
    "OBJECT"

```

```

IBMwlen = whsp len

```

Le definizioni dello schema che vengono restituite dal server ma non sono supportate da questa libreria di classi, vengono salvate e non analizzate. Queste voci contengono esattamente due attributi, uno con un objectclass che corrisponde al nome del tipo di schema (ad esempio, objectclass=adddef) e un altro con una lista di valori. E' possibile visualizzare ma non aggiornare le definizioni di schema non supportate.

L'esempio seguente reperisce l'intera gerarchia dello schema:

Esempio: reperire una gerarchia dello schema

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

DirContext schemaCtx = ctx.getSchema("");
SearchControls cons = new SearchControls();
cons.setSearchScope(SearchControls.SUBTREE_SCOPE);
NamingEnumeration ne = schemaCtx.search("",
    "(|(NUMERICOID=*)(objectclass=*))", cons);

```

L'esempio seguente reperisce lo schema relativo all'attributo cn:

Esempio: reperire lo schema relativo all'attributo cn

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

DirContext schemaCtx = ctx.getSchema("");
Attributes attrs = schemaCtx.getAttributes("AttributeDefinition/cn");

```

L'esempio seguente tenta di aggiungere una definizione di schema a una nuova classe di oggetti:

Esempio: aggiungere una definizione di schema

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```

DirContext schemaCtx = ctx.getSchema("");
BasicAttributes attrs = new BasicAttributes();
attrs.put("NAME", "javaObject");
attrs.put("NUMERICOID", "1.3.6.1.4.1.42.2.27.4.2.2");
Attribute may = new BasicAttribute("MAY");
may.add("javaClassName");
may.add("javaSerializedObject");
attrs.put(may);
attrs.put("DESC", "Serialized Java object");
attrs.put("AUXILIARY", "true");
attrs.put("SUP", "top");
schemaCtx.createSubcontext("ClassDefinition/javaObject", attrs);

```

La classe com.ibm.jndi.LDAPSchemaCtx è stata estesa in particolar modo per gestire le definizioni di schema derivate da un file. Due programmi di creazione pubblici supportano la lettura delle informazioni da un disco. Un programma prende come argomento un filename singolo mentre l'altro una schiera di filename. Il metodo dumpSchema salva la definizione dello schema su un file. L'esempio seguente dimostra questo supporto:

Esempio: il metodo dumpSchema

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
LDAPSchemaCtx ctx = new LDAPSchemaCtx("schema.file");
ctx.dumpSchema("schema.sav");
```

Modulo aggiuntivo SASL

➤ E' possibile registrare il modulo aggiuntivo SASL (Simple Authentication and Security Layer). E' necessario che il modulo aggiuntivo SASL sia derivato dalla classe di base astratta LDAPSaslBind. E' necessario implementare il metodo di collegamento che chiama successivamente il metodo SendBindRequest per comunicare con il server. Il seguente è un esempio di un modulo aggiuntivo di collegamento semplice:

Esempio: modulo aggiuntivo di collegamento

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.io.IOException;
import com.ibm.asn1.ASN1Exception;
import com.ibm.ldap.*;

public class SimpleBind extends LDAPSaslBind
{
    public boolean bind(String dn, String credentials)
        throws IOException, ASN1Exception, LDAPException
    {
        return SendBindRequest("SIMPLE", dn, credentials);
    }
}
```

Se il protocollo SASL include domande di chiarificazione del server, il metodo SendBindRequest deve essere richiamato più volte, possibilmente utilizzando il metodo getServerCredentials per reperire le informazioni su tali domande. Il seguente è un esempio del modulo aggiuntivo CRAM-MD5 SASL:

Esempio: modulo aggiuntivo SASL CRAM-MD5

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import java.net.*;
import java.io.*;
import java.security.*;
import com.ibm.util.*;
import com.ibm.asn1.ASN1Exception;
import com.ibm.ldap.*;

/*
    Challenge-Response Authentication Mechanism / MD5 hash.
    Vedere RFC 2195 ("IMAP/POP AUTHorize Extension for Simple
    Challenge/Response") e draft-ietf-ldapext-authmeth-02
    ("Authentication Methods for LDAP") per dettagli.
*/
public class CramMD5 extends LDAPSaslBind
{
    public boolean bind(String dn, String credentials)
        throws IOException, ASN1Exception, LDAPException
    {
        String clientCreds;

        // Inviare la richiesta di collegamento iniziale
        if (SendBindRequest("CRAM-MD5", dn, null) == true)
            return false;

        // Generare md5 hash dal secret del client e dal challenge
```

```

// del server e inviarli al server
try {
    clientCreds = "dn: " + new String(stringUTF(dn)) + " " +
        HMAC_MD5(credentials, getServerCredentials());
} catch (NoSuchAlgorithmException nsae) {
    throw new IOException(nsae.toString());
}
putCredentials(clientCreds);
return SendBindRequest();
}

/*
    Hashed Message Authentication Code. Vedere RFC 2104
    ("HMAC: Keyed-Hashing for Message Authentication")
    per dettagli.
*/
public static String HMAC_MD5(String secret, String text)
    throws NoSuchAlgorithmException
{
    MessageDigest md5;
    byte[] ipad, opad, key;
    int i;

    // Inizializzazione
    md5 = MessageDigest.getInstance("MD5");
    ipad = new byte[64];
    opad = new byte[64];
    key = secret.getBytes();

    // Se la chiave è più grande della dimensione del blocco allora la chiave hash
    if (key.length > 64)
        key = md5.digest(key);

    // Eseguire XOR di ipad e opad con chiave (riempito fino a 64 byte).
    for (i = 0; i < key.length; ++i) {
        ipad[i] = (byte)(0x36 ^ key[i]);
        opad[i] = (byte)(0x5c ^ key[i]);
    }
    while (i < 64) {
        ipad[i] = 0x36;
        opad[i++] = 0x5c;
    }

    // Testo e risultato hash ipad XOR
    md5.update(ipad);
    key = md5.digest(text.getBytes());

    // Risultato hash opad XOR e hash precedente
    md5.update(opad);
    key = md5.digest(key);

    // Restituire una rappresentazione decimale di hash (32 byte)
    return Hex.toString(key, false);
}
}

```



Memorizzazione nella cache dal lato del client

» La memorizzazione nella cache fornisce la possibilità di memorizzare le informazioni richieste recentemente in maniera locale. Questo migliora le prestazioni reperendo interrogazioni ripetute localmente invece di fare riferimento a un server remoto per informazioni già ottenute. Lo svantaggio è che non esiste un modo per determinare se un valore nella memoria cache sia stato modificato dal momento

del reperimento. La bozza di internet, A Simple Caching Scheme for Lightweight Directory Access Protocol (LDAP) e X.500 Directories, si riferisce a questo problema definendo un attributo di "durata" che indica la quantità di tempo in cui una voce può ragionevolmente rimanere nella memoria cache prima che diventi inutilizzabile. L'implementazione della memorizzazione nella cache dal lato del client nella libreria delle classi supporta completamente questa bozza.

L'algoritmo per la memorizzazione nella cache utilizzato in questa libreria di classi è completamente basato su una richiesta di ricerca. I dati non vengono duplicati localmente; in questo modo non è possibile interrogare una voce in due modalità diverse dalla memoria cache. Ad esempio, una ricerca di "cn=Joe*" seguita da un'interrogazione relativa a "cn=Joe Smith" non reperisce il valore dalla memoria cache, anche se è possibile che i risultati si trovino localmente come parte della serie di risultati "cn=Joe*". La duplicazione dei dati di LDAP contiene molte particolarità e non rientra nell'ambito dello schema di memorizzazione nella cache, in special modo per quanto riguarda Java^(TM). La ricerca locale d'interrogazioni complicate in Java potrebbe richiedere un tempo maggiore rispetto al tempo che si impiegherebbe a reperire i dati da un server ottimizzato.

La memorizzazione nella cache non dovrebbe essere utilizzata per i dati che con più probabilità si modificano velocemente oppure quando vengono reperite informazioni importanti. La memorizzazione nella cache viene disabilitata per impostazione predefinita in questa libreria di classi, e in caso venga abilitata, questa fornisce la possibilità di ignorare la memoria cache e reperire i risultati direttamente dal server. Tuttavia i dati LDAP sono tipicamente statici e spesso sono basati su informazioni, cosicché la memorizzazione nella cache con valori di durata accurati diventa un modello valido per molte applicazioni.

Quando abilitata, la memorizzazione nella cache viene utilizzata su operazioni che reperiscono dati dal server e include i seguenti metodi:

- `getAttributes`
- `search`
- `lookup`
- `lookupLink`
- `list`
- `listBindings`
- `getSchema`
- `getSchemaClassDefinition`

La memoria cache è creata quando un oggetto del tipo `LDAPCache` crea istanze. E' possibile che più contesti condividano una memoria cache singola, nel senso che un valore posizionato nella memoria cache da un contesto può essere reperito da un altro contesto. La memorizzazione nella cache in IBM JNDI LDAP provider è anche sicura durante i sottoprocessi. Un'altra caratteristica di `LDAPCache` è che essa può essere serializzata in un disco e ripristinata successivamente. E' possibile modificare la dimensione della memoria cache basata sul numero di voci reperite.

Un contesto non utilizza `LDAPCache` direttamente, ma fa affidamento su `LDAPCacheControl` per gestire la memoria cache. `LDAPCacheControl` fornisce una vista individualizzata all'interno di una memoria cache condivisa. `LDAPCacheControl` rende possibile l'ottimizzazione delle impostazioni sulla base di una richiesta come ad esempio il valore di durata e la possibilità di ignorare e reperire i risultati direttamente dal server. Sebbene è possibile utilizzare `LDAPCacheControl` sulla base di una richiesta, è possibile inoltre che questo venga condiviso tra più contesti. Qualsiasi riferimento a `LDAPCacheControl` in un contesto viene inoltrato a contesti secondari creati dai metodi `lookup`, `lookupLink` e `listBindings`.

La proprietà `java.naming.control.cache` viene definita dal fornitore come un modo per associare un controllo di memoria cache a un contesto. Questo può essere fatto o durante la creazione di un contesto oppure successivamente attraverso il metodo `addToEnvironment`.

Per abilitare la memorizzazione nella cache, è necessario creare istanze su un oggetto di tipo `LDAPCacheControl` e associarlo a un contesto. La creazione di `LDAPCacheControl` richiede un oggetto `LDAPCache` come argomento. Il seguente è un esempio di creazione di un contesto tramite la memorizzazione nella cache abilitata:

Esempio: creare un contesto con la memorizzazione nella cache abilitata

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

```
import com.ibm.jndi.LDAPCtx;
import com.ibm.ldap.LDAPCache;
import com.ibm.ldap.LDAPCacheControl;

LDAPCache cache = new LDAPCache();
LDAPCacheControl cacheControl = new LDAPCacheControl(cache);
env.put("java.naming.control.cache", cacheControl);
DirContext ctx = new InitialDirContext(env);
```

E' possibile ottimizzare le impostazioni sulla base di una richiesta. Le impostazioni rimangono effettive fino a quando non vengono eliminate. Quanto segue forza un valore di durata di 60 secondi:

```
cacheControl.putTTL(60);
cacheControl.putHonorServerTTL(false);
ctx.search(...);
```

Quanto segue ignora qualsiasi valore memorizzato nella cache e reperisce i risultati direttamente dal server:

```
cacheControl.putReadFlag(false);
ctx.search(...);
```

Nota: a causa della proprietà `java.naming.batchsize`, vengono aggiunti elementi alla memoria cache finché i risultati non vengono completamente enumerati. <<

Reperire la versione della classe IBMJNDI

>> E' possibile ottenere la versione delle classi LDAP (Lightweight Directory Access Protocol) utilizzando il seguente metodo statico:

```
String version = com.ibm.jndi.LDAPCtx.getVersion();
```

Considerazioni di conformità e proprietà aggiuntive

Le seguenti differenze conosciute esistono tra IBM JNDI (Java ^(TM) Naming and Directory Interface) LDAP provider JNDI Implementor Guidelines for LDAP Service Providers (Bozze 0.2) di Sun .

1. La federazione di denominazione non è supportata.
2. L'utilizzo del protocollo SSL (Secure Socket Layer) non è supportato da IBM JNDI LDAP provider sui server iSeries.
3. Non è necessario utilizzare i IBM LDAP provider per memorizzare gli oggetti Java serializzati in un indirizzario LDAP.
4. L'inoltro di componenti URL oltre il necessario non risulta in una `ConfigurationException` né in una `InvalidNameException`.
5. IBM JNDI LDAP provider possiede il proprio supporto di modulo aggiuntivo SASL (Simple Authentication and Security Layer). Questo non supporta l'API (Application Programming Interface) di SASL Java perché l'API attualmente esiste come anteprima del pacchetto.
6. Il fornitore supporta eventi di notifica non sollecitati, ma non supporta eventi di modifica oggetto o spazio nome.
7. Il valore predefinito per `java.naming.referral` è `follow` e non `ignore`. Quando questa proprietà è impostata su `ignore`, il fornitore non aggiunge automaticamente il controllo `ManageDSAlt` alle richieste, né emette una `PartialResultException` quando un riferimento viene ricevuto.

Anche la proprietà seguente viene supportata:

com.ibm.jndi.Ldap.so_timeout

Questa proprietà definisce il numero di millisecondi che un blocco contesto attende per i dati provenienti da un server. Il supero tempo predefinito è di 5 minuti. Un supero tempo pari a zero verrà interpretato come un supero tempo infinito.

Le proprietà seguenti sono state considerate obsolete:

- java.naming.Ldap.noBind
- java.naming.control.server



JSSL

» Il JSSL (Java Secure Socket Layer) è una serie di pacchetti Java che consentono comunicazioni sicure in Internet. Questo implementa una versione Java di protocolli SSL e TLS (Transport Layer Security) e include la funzionalità relativa alla codifica dei dati, all'autenticazione del server, all'integrità del messaggio e all'autenticazione facoltativa del client. Utilizzando JSSL, è possibile sviluppare applicazioni che forniscono un passaggio di dati sicuro tra un client e un server che esegue qualsiasi protocollo di applicazione (come HTTP, Telnet, NNTP e FTP) sopra TCP/IP.

Per ulteriori informazioni relative a JSSL, consultare JSSE (Java Secure Socket Extension)  di Sun Microsystems, Inc. 

JavaMail


» L'API JavaMail^(TM) fornisce una serie di classi astratte che modellano 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.

Fare riferimento alla documentazione di Sun Microsystems, Inc. JavaMail per ulteriori informazioni. 



Java Print Service



» L'API Java Print Service[™] consente la stampa su tutte le piattaforme Java. Java 1.4 fornisce un framework nel quale i Java runtime environment e terzi possono fornire moduli aggiuntivi di creazione flussi per la produzione di vari formati stampa, come il PDF, il Postscript e l'AFP (Advanced Function Presentation). Questi moduli aggiuntivi creano formati di emissione da chiamate grafiche bidimensionali (2D).

Fare riferimento alla documentazione di Sun Microsystems Java Print Service per ulteriori informazioni



Capitolo 8. Effettuare il debug dei programmi utilizzando IBM Developer Kit per Java

Se è necessario effettuare il debug dei propri programmi Java^(TM), selezionare una delle seguenti opzioni:

- Effettuare il debug di un programma Java
- Effettuare il debug dei programmi del metodo nativo e Java
- Effettuare il debug di un programma Java da un altro pannello
-  Effettuare debug delle classi Java caricate tramite un programma di caricamento classi personalizzato 
- Effettuare il debug dei servlet

Quando si effettua il debug di un programma Java, il proprio programma Java è effettivamente in esecuzione nella JVM (Java virtual machine) in un lavoro BCI (batch immediato). Il proprio 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. Quando il programma Java termina, termina il lavoro servito e viene visualizzato un messaggio che indica Job being serviced ended.


Non è possibile effettuare il debug di programmi Java in esecuzione con il compilatore JIT (Just-In-Time). Se un file non dispone di un programma Java associato, il valore predefinito è eseguire il JIT. E' possibile disabilitare tale valore in molti modi per consentire il debug:

- Specificare la proprietà `java.compiler=NONE` quando si avvia la Java virtual machine.
- Specificare `OPTION(*DEBUG)` sul comando `RUNJVA` (Esecuzione Java).
- Specificare `INTERPRET(*YES)` sul comando `RUNJVA` (Esecuzione Java).
- Utilizzare `CRTJVAPGM OPTIMIZATION(10)` per creare un programma Java associato prima di avviare la JVM (Java virtual machine).

Nota: nessuna di queste soluzioni influenza una JVM (Java virtual machine) in esecuzione. Se una JVM (Java virtual machine) non è stata avviata con una di queste alternative, è necessario arrestarla e riavviarla per effettuare il debug.

L'interfaccia tra i due lavori viene stabilita quando si specifica l'opzione `*DEBUG` sul comando `RUNJVA` (Esecuzione Java).

Per ulteriori informazioni sul programma di debug del sistema, consultare il manuale WebSphere

Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  e le informazioni di aiuto in linea.

Effettuare il debug di un programma Java

Esistono diversi modi per effettuare il debug di un programma Java^(TM). E' possibile 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.

Per eseguire il debug dei programmi Java, effettuare quanto segue:

1. Compilare il programma Java utilizzando l'opzione `DEBUG`, che è l'opzione `-g` sullo strumento `javac`. Consultare Effettuare il debug dei programmi Java utilizzando l'opzione `*DEBUG` per ulteriori dettagli.
2. Inserire il file di classe (`.class`) e il file origine (`.java`) nello stesso indirizzario sul proprio server iSeries.
3. Eseguire il programma Java utilizzando il comando `RUNJVA` (Esecuzione Java) sulla riga comandi iSeries. Specificare `OPTION(*DEBUG)` sul comando `RUNJVA` (Esecuzione Java).


```

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. E' 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
                                *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 comparse dove si dispone di maggiore spazio per immettere il nome file di classe qualificato del pacchetto.

Impostare i punti di interruzione

E' 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 |
|                                     |
+-----+
```

Quando avviene una corrispondenza con il punto di interruzione, se si desidera impostare punti di interruzione che sono rilevati all'interno del sottoprocesso corrente, utilizzare il comando TBREAK.

Per ulteriori informazioni sui comandi del programma di debug del sistema, consultare il manuale

WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  e le informazioni di supporto in linea.

Per informazioni sulle variabili di valutazione quando un programma arresta l'esecuzione su un punto di interruzione, consultare Valutare variabili nei programmi Java^(TM).

Eseguire le istruzioni dei programmi Java da sottoporre a debug

E' possibile eseguire le istruzioni del programma durante l'esecuzione del debug. E' possibile ignorare o eseguire dettagliatamente altre funzioni. I programmi Java^(TM) 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. E' possibile inoltre avviare la funzione di esecuzione istruzione ogni volta che viene raggiunto un punto di interruzione. Per informazioni sull'impostazione dei punti di interruzione, consultare 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 arrestare la funzione di esecuzione istruzione e continuare l'esecuzione del programma, premere F12 (Ripresa).

Per ulteriori informazioni sulla funzione di esecuzione istruzioni, consultare il manuale WebSphere

Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  e informazioni di aiuto in linea.

Per informazioni sulla valutazione delle variabili quando un programma arresta l'esecuzione su un'istruzione, consultare Valutare le variabili nei programmi Java.

Determinare il valore delle 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:

- Immettere EVAL VariableName sulla riga comandi del debug.
- Posizionare il cursore sul nome della variabile nel codice sorgente visualizzato e premere F11 (Visualizzazione variabile).

Utilizzare il comando EVAL per determinare il valore delle variabili in un programma Java^(TM).

Nota: è inoltre possibile modificare il contenuto di una variabile utilizzando il comando EVAL. Per ulteriori informazioni sulle variazioni del comando EVAL, consultare il manuale WebSphere Development Studio:

ILE C/C++ Programmer's Guide, SC09-2712  e le informazioni contenute nell'aiuto 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.

- Se si determina il valore di una variabile che è una schiera, viene visualizzato 'ARR' seguito da un identificativo per tale schiera. E' 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. E' 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.

Effettuare il debug dei programmi del metodo nativo e Java

E' possibile effettuare il debug dei programmi del metodo nativo e dei programmi Java^(TM) nello stesso momento. 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). » E' necessario compilare *SRVPGM e crearlo con i dati di debug. «

Per effettuare il debug dei programmi Java e dei programmi del metodo nativo nello stesso tempo:

1. Premere F14 (Gestione lista moduli) quando il proprio sorgente programma Java viene visualizzato per mostrare il pannello WRKMODLST (Gestione lista 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 Impostare i punti di interruzione.
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.

Effettuare il debug di un programma Java da un altro pannello

Durante il debug di un programma Java^(TM), il sorgente programma viene visualizzato ogniqualvolta si incontra 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.

E' inoltre possibile effettuare il debug di un programma Java già in esecuzione, in questo modo, purché non utilizzi il compilatore JIT (Just-In-Time).

Per effettuare il debug di Java da un altro pannello, eseguire queste operazioni:

1. E' necessario congelare il programma Java, mentre si avvia l'impostazione per effettuare il debug. E' possibile congelare il programma Java, operando affinché il programma effetti quanto segue:
 - Attendere l'immissione dalla tastiera.
 - Attendere un intervallo di tempo.

- Eseguire il loop per esaminare una variabile, che richiede all'utente di impostare un valore per collocare 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 nella lista Sottosistema/Lavoro. Cercare nella lista Utente il proprio ID utente. Cercare BCI nella lista 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 classe Java su cui si desidera effettuare il debug. Può essere sia il nome classe specificato sul comando Java che 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. Per ulteriori informazioni sull'impostazione dei punti di interruzione, consultare Impostare i punti di interruzione.
 - h. Premere F12 (Ripresa) per continuare ad eseguire il programma.
 3. Arrestare il conservabilità 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).

Nota: assicurarsi che sia stato disabilitato JIT (Just-In-Time) quando si avvia la Java virtual machine nel lavoro originale. E' possibile effettuare ciò con la proprietà `java.compiler=NONE`. Se è in esecuzione JIT durante il debug, è possibile che si verifichino risultati imprevisti.

Consultare variabile di ambiente `QIBM_CHILD_JOB_SNDINQMSG` per ulteriori informazioni su questa variabile che controlla se il lavoro BCI attende prima di chiamare la Java virtual machine.

Variabile di ambiente `QIBM_CHILD_JOB_SNDINQMSG`

La variabile di ambiente `QIBM_CHILD_JOB_SNDINQMSG` controlla se il lavoro BCI (batch immediato), nel quale è in esecuzione la Java^(TM) 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.

E' 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.

Effettuare il debug delle classi Java caricate tramite un programma di caricamento classi personalizzato

» Per eseguire il debug di una classe caricata tramite un programma di caricamento classi personalizzato, effettuare quanto segue.

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 ovvero Java^(TM) virtual machine, aggiungere come al solito soltanto `*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		<code>pkg1.test14_</code>	<code>*LIBL</code>	<code>*CLASS</code>

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 della classe viene visualizzato e può essere sottoposto a debug.◀

Servlet di debug

» Effettuare il debug dei servlet è un caso speciale di effettuare il debug delle classi caricate tramite un programma di caricamento classi personalizzato. I servlet vengono eseguiti nel tempo di esecuzione Java^(TM) del server HTTP IBM. Un modo per effettuare il debug dei servlet è seguendo le istruzioni per le classi caricate tramite un programma di caricamento classe di personalizzazione.

Un altro modo per effettuare il debug di un servlet è il seguente ◀

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` 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.

Struttura del programma di debug della piattaforma Java

» La JPDA (Java™ Platform Debugger Architecture) è formata da tre parti:

- JVMDI (Java Virtual Machine Debug Interface)
- JDWP (Java Debug Wire Protocol)
- JDI (Java Debug Interface)

Tutte e tre le 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 iSeries.

JVMDI (Java Virtual Machine Debug Interface)

In J2SDK (Java™ 2 SDK), Edizione Standard, versione 1.2 o successive, la JVMDI (Java Virtual Machine Debug Interface) fa parte delle API (application program interface) della piattaforma di Sun Microsystems, Inc. JVMDI consente a chiunque di scrivere un programma di debug Java per un server iSeries nel codice C di iSeries. Non è necessario che il programma di debug conosca la struttura interna della Java virtual machine dato che utilizza interfacce JVMDI. JVMDI è l'interfaccia di livello più basso in JPDA più vicina alla Java virtual machine.

Il programma di debug esegue nello stesso lavoro capace di supportare più sottoprocessi della Java virtual machine. Esso utilizza le API di richiamo JNI (Java Native Interface) per creare una Java virtual machine. Esso inserisce, quindi, un hook all'inizio di un metodo principale della classe utente e chiama tale metodo. Quando il metodo principale viene avviato, l'hook viene attivato e si avvia il debug. Sono disponibili le funzioni di debug tipiche, come impostare i punti di interruzione, avanzare, visualizzare e modificare le variabili.

Il programma di debug gestisce la comunicazione tra il lavoro dove la Java virtual machine è in esecuzione e un lavoro che gestisce l'interfaccia dell'utente. Tale interfaccia si trova sul proprio server iSeries o su un altro sistema.

Un programma di servizio, denominato QJVAJVMDI che si trova nella libreria QSYS, supporta le funzioni JVMDI.

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. E' possibile utilizzare JDWP da un sistema remoto o su un socket locale. Esso è un livello eliminato dalla JVMDI, ma è un'interfaccia più complessa.

Avviare JDWP in QShell

Per avviare JDWP ed eseguire la classe Java SomeClass, immettere il seguente comando in QShell:

```
java -interpret -Xrunjdpw: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 desidera; dt_socket è il nome del SRVPGM che gestisce il trasferimento JDWP e non cambia.

Per opzioni aggiuntive che è possibile utilizzare con -Xrunjdpw, consultare Opzioni di richiamo VM di Sun



di Sun Microsystems, Inc.

Avviare JDWP da una riga comandi CL

Per utilizzare l'opzione -Xrun con il comando CL, è possibile definire la proprietà os400.xrun.option 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)
PROP((os400.xrun.option 'jdpw:transport=dt_socket,address=8000,
server=y,suspend=n'))
```

Molte funzioni JVMDI non funzionano ai livelli di ottimizzazione 10 e 20. Quindi, si raccomanda di controllare che le proprie applicazioni eseguano con l'interprete dato che tutte le funzioni lo gestiscono.

JDI (Java Debug Interface)

JDI (Java Debug Interface) è un'interfaccia di linguaggio Java ad alto livello fornita per lo sviluppo dello strumento. JDI nasconde la complessità di JVMDI 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 JDBA, consultare [Panoramica su Java Platform Debugger Architecture](#)  di Sun Microsystems, Inc.



Rilevare perdite di memoria

» 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. E' 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 ANZJVM una serie di volte con un intervallo di tempo maggiore, l'utente dovrebbe essere in grado di stabilire con un alto grado di certezza da quale oggetto provenga la perdita. «

Capitolo 9. Risolvere i problemi di IBM Developer Kit per Java

Se si riscontrano problemi durante l'utilizzo di IBM Developer Kit per Java^(TM), eseguire uno qualsiasi dei seguenti passaggi per determinare l'origine del problema.

- **»** E' possibile notare alcuni limiti durante l'utilizzo di IBM Developer Kit per Java. Questo argomento identifica tutti i limiti, le restrizioni o le funzionalità univoche note. **«**
- Rilevare una registrazione lavori dal lavoro che ha eseguito il comando Java. Inoltre, individuare la registrazione lavori dal lavoro BCI (batch immediato) nel quale è stato eseguito il programma Java per analizzare la causa dell'errore.
- Raccogliere dati utili per un APAR (authorized program analysis report).
- Applicare le PTF (program-temporary fix).
- Conoscere il modo in cui ottenere supporto se si rileva un errore potenziale in IBM Developer Kit per Java.

Limiti

Quando si utilizza l'IBM Developer Kit per Java^(TM), esistono alcune limitazioni sul modo di utilizzarlo. Questa lista identifica tutte le limitazioni, restrizioni o funzionalità univoche conosciute.

- 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 prolunga 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 132 GB.
- Il numero degli oggetti creati è limitato a 132 milioni. **«**
- Il parametro backlog java.net su un server iSeries può funzionare in modo diverso rispetto a un'altra piattaforma. 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.
- **»** E' possibile utilizzare solo la Java virtual machine, indipendentemente dalla versione che si sta utilizzando, in ambienti in grado di supportare più sottoprocessi (cioè sicuri a livello dei sottoprocessi). Il server iSeries è sicuro a livello dei sottoprocessi, ma alcuni file system non lo sono. L'argomento integrated file system ha una lista di file system non sicuri a livello dei sottoprocessi. **«**

Rilevare le registrazioni lavori per un'analisi del problema Java

Utilizzare la registrazione lavori dal lavoro che ha eseguito il comando Java^(TM) e la registrazione lavori BCI (batch immediato) dove è stato eseguito il programma Java, per analizzare le cause dell'errore Java. E' possibile che entrambe contengano importanti informazioni sull'errore.

Esistono due modi per trovare la registrazione lavori per il lavoro BCI. E' 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.

E' 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 iSeries.
2. Andare alla fine della lista.
3. Cercare l'ultimo lavoro nella lista, 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.

Nota: se la data e l'ora non corrispondono alla data e all'ora in cui l'utente si è scollegato, continuare a cercare nella lista 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 QDFTJOBDB troppo alto o il valore LOG per la descrizione lavoro QDFTJOBDB 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 iSeries.
4. Andare alla fine della lista.
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.

Nota: se la data e l'ora non corrispondono alla data e all'ora in cui l'utente si è scollegato, continuare a cercare nella lista 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.

Raccogliere dati per un'analisi dei problemi Java

Per raccogliere 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^(TM) che ha causato il problema durante l'esecuzione.
3. E' possibile utilizzare il comando SAV per salvare gli oggetti dall'IFS (Integrated File System). E' necessario salvare altri file della classe che questo programma deve eseguire. E' 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: salvare un indirizzario


Nota: consultare 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 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. Sarebbe opportuno includere:
 - 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).

Acquisire il supporto di IBM Developer Kit per Java


I servizi di supporto per IBM Developer Kit per Java^(TM) vengono forniti nei soliti termini e condizioni per i prodotti software iSeries. I servizi di supporto includono servizi di programma, supporto vocale e servizi di consultazione. Utilizzare le informazioni in linea fornite sull'Home Page iSeries di IBM  sotto l'argomento "Supporto" per ulteriori informazioni. Utilizzare i Servizi di supporto IBM per 5722-JV1 (IBM Developer Kit per Java). O contattare il proprio rappresentante IBM locale.

E' possibile che venga richiesto, in base ad una direttiva dell'IBM, di acquisire un livello più recente di IBM Developer Kit per Java per ricevere servizi del programma continuati. Per ulteriori informazioni, consultare Supporto per più JDK (Java Development Kit).

I difetti di definizione del programma di 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) di IBM Developer Kit per Java sono supportate nei servizi di consultazione, a meno che:

1. E' chiaramente un difetto dell'API di Java come dimostrato dalla nuova creazione in un programma relativamente semplice.
2. E' una questione che richiede una chiarifica della documentazione,
3. E' 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. E' possibile trovare ulteriori esempi su Internet nella Home Page IBM iSeries  su una base non supportata.

LP di IBM Developer Kit per Java fornisce informazioni sulla risoluzione dei problemi. Se l'utente crede che esista un difetto potenziale nell'API di IBM Developer Kit per Java, è necessario un semplice programma che dimostri l'errore.

Capitolo 10. Esempi di codice per IBM Developer Kit per Java

Segue una lista di esempi di codice per IBM Developer Kit per Java^(TM).

Nota: consultare l'Esonero di responsabilità per gli esempi di codice per importanti informazioni legali.

Comandi CL

- >> ANZJVM<<
- CHGJVAPGM
- CRTJVAPGM
- DLTJVAPGM
- DMPJVM
- DSPJVAPGM
- RUNJVA

Internazionalizzazione

- DateFormat
- NumberFormat
- ResourceBundle

JDBC

- >> Proprietà Access
- Blob<<
- Interfaccia CallableStatement
- >> Modificare i valori con un'istruzione tramite un altro cursore dell'istruzione
- Clob
- Creare UDBDataSource e collegarlo con JNDI
- Creare UDBDataSource e ottenere un ID utente e una parola d'ordine
- Creare UDBDataSourceBind e impostare le proprietà DataSource<<
- Interfaccia DatabaseMetaData
- >> Creare UDBDataSource e collegarlo con JNDI
- Datalink
- Tipi "distinct"<<
- Incorporare le istruzioni SQL
- >> Terminare una transazione
- ID utente e parola d'ordine non validi
- JDBC
- Più collegamenti che operano su una transazione
- Ottenere un contesto iniziale prima di collegare UDBDataSource
- ParameterMetaData
- Eliminare i valori da una tabella tramite un altro cursore dell'istruzione<<
- Interfaccia ResultSet
- >> Sensibilità di ResultSet
- ResultSet sensibili e non sensibili

- Impostare il lotto di collegamenti con UDBDataSource e UDBConnectionPoolDataSource
- SQLException
- Sospendere e ripristinare una transazione
- ResultSet sospesi
- Sottoporre a verifica le prestazioni del lotto di collegamenti
- Sottoporre a verifica le prestazioni dei due DataSource
- Aggiornare i BLOB
- Aggiornare i CLOB
- Utilizzare un collegamento con più transazioni
- Utilizzare i BLOB
- Utilizzare i CLOB
- Utilizzare le proprietà DB2CachedRowSet e DataSource
- Utilizzare le proprietà DB2CachedRowSet e gli URL JDBC
- Utilizzare JTA per gestire una transazione
- Utilizzare i ResultSet metadati che dispongono di più di una colonna
- Utilizzare JDBC nativo e JDBC di Toolbox contemporaneamente
- Utilizzare PreparedStatement per ottenere un ResultSet
- Utilizzare il metodo execute(Connection) al fine di utilizzare un collegamento database esistente
- Utilizzare il metodo execute(int) per sottoporre a batch tutte le richieste database
- Utilizzare il metodo populate
- Utilizzare il metodo setConnection(Connection) per utilizzare un collegamento database esistente
- Utilizzare il metodo executeUpdate dell'oggetto Statement

JAAS (Java Authentication and Authorization Service)

- Esempio HelloWorld JAAS
- Esempio SampleThreadSubjectLogin JAAS

Servizio di sicurezza generico di Java

- Esempio programma client non-JAAS
- Esempio programma server non-JAAS
- Esempio programma client abilitato JAAS
- Esempio programma server abilitato JAAS

Interfaccia dell'indirizzario e denominazione Java

- Aggiungere una voce in un indirizzario
- Cancellare una voce in un indirizzario
- Aggiungere una voce in un indirizzario
- Ridenominare una voce dell'indirizzario
- Specificare una lista di nomi di attributi binari

Estensione SSL (Secure Sockets layer) Java

- Client e server SSL che utilizzano un oggetto SSLContext

Java con altri linguaggi di programmazione

- Chiamare un programma CL
- Chiamare un comando CL

- Chiamare un altro programma Java
- Chiamare Java da C
- Chiamare Java da RPG
- Flussi di immissione ed emissione
- API di richiamo
- Metodo nativo PASE OS/400 per Java <<
- Socket

>> **Pacchetti facoltativi**

- JCE<<

Strumenti delle prestazioni

- Programma di conversione dati delle prestazioni Java

>> **Eseguire un host senza GUI**

- Impostare AWT Remoto<<

SQLJ

- Incorporare le istruzioni SQL nella propria applicazione Java

SSL (Secure socket layer)

- Produzioni socket
- Produzioni socket server
- SSL (Secure socket layer)
- Server di SSL (Secure socket layer)

Capitolo 11. Riferimenti di IBM Developer Kit per Java

» Seguono i riferimenti per il Developer Kit per Java^(TM):

Javadoc

- Javadoc JAAS specifico di iSeries
- Specifica API JAAS

Piattaforma 2 Java, Edizione Standard, versione 1.3.1

- Specifica API v1.3.2 Piattaforma 2 Java, Edizione Standard
- AWT (Abstract Window Toolkit)
- IDL Java
- Framework metodo di immissione
- Internazionalizzazione
- API JDBC
- JNI - Java Native Interface
- RMI Java (Remote Method Invocation)
- RMI - Remote Method Invocation
- Sicurezza
- Strumenti SDK Java 2»

Informazioni sull'Esonero di responsabilità del codice

Questo documento contiene esempi di programmazione.

L'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.

Questo codice di esempio è fornito dall'IBM con la sola funzione illustrativa. Questi esempi non sono stati interamente testati in tutte le condizioni. L'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. Le garanzie implicite di non contraffazione, commerciabilità e adeguatezza a scopi specifici sono espressamente vietate.



Printed in Denmark by IBM Danmark A/S