System i

Programar el IBM Developer Kit para Java

Versión 6 Release 1

System i

Programar el IBM Developer Kit para Java

Versión 6 Release 1

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información de “Avisos”, en la página 585.

Esta edición atañe a la versión 6, release 1, modificación 0 de IBM Developer Kit para Java (producto número 5761-JV1) y a todos los releases y modificaciones ulteriores hasta que se indique lo contrario en nuevas ediciones. Esta versión no funciona en todos los modelos RISC (reduced instruction set computer) ni en los modelos CISC.

Contenido

IBM Developer Kit para Java

IBM Developer Kit para Java se ha optimizado para poder utilizarlo en el entorno System i5. Utiliza la compatibilidad de las interfaces de usuario y programación Java para que usted pueda desarrollar sus propias aplicaciones de System i5.

IBM Developer Kit para Java le permite crear y ejecutar programas Java en el System i5. IBM Developer Kit para Java tiene una implementación compatible de la tecnología Java de Sun Microsystems, Inc., lo que nos hace suponer que usted está familiarizado con la documentación de Java Development Kit (JDK). Para facilitarle el trabajo con esa información y con la nuestra, le proporcionamos enlaces con la información de Sun Microsystems, Inc.

Si por alguna razón no funcionan los enlaces que nos llevan a la documentación de Java Development Kit de Sun Microsystems, Inc., vea la documentación de consulta HTML de Sun para localizar la información que necesita. Hallará esta información en la World Wide Web, en The Source for Java Technology java.sun.com.

Nota: Encontrará información legal importante en: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

Novedades de la V6R1

Aquí encontrará la información nueva o la que ha cambiado notablemente en el temario de IBM Developer Kit para Java.

Retirada del soporte de ejecución directa

En V6R1, el proceso directo ha dejado de estar soportado. Esto quiere decir que el nivel de optimización de los programas Java se ignora y que se emplea OPTIMIZE(*INTERPRET) cuando se crea un programa Java en V6R1. Los detalles están en los siguientes temas:

- "Compatibilidad entre releases" en la página 33
- "Consideraciones sobre el rendimiento de Java" en la página 414
- "Compilación Java estática" en la página 417
- "Consideraciones sobre el rendimiento de la compilación estática Java" en la página 418

Cambios realizados en el soporte de autorización adoptada

El soporte de la autorización adoptada por perfil de usuario mediante programas Java se retira en V6R1. En: "Cambios realizados en la autorización adoptada en V6R1" en la página 258 se explica cómo determinar si las aplicaciones emplean la autorización adoptada y cómo modificarlas para que se ajusten a este cambio.

Hardware de extensión de criptografía Java (JCE)

La implementación de IBMJCECAI5OS amplía la extensión de criptografía Java (JCE) y la arquitectura de criptografía Java (JCA) para añadir la capacidad de utilizar criptografía por hardware por medio de las

interfaces de la arquitectura criptográfica común (CCA) de IBM. Vea: “Utilizar la criptografía por hardware” en la página 273.

Interfaz de herramientas de la máquina virtual Java (JVMTI)

La JVMTI es una interfaz que sirve para analizar la máquina virtual Java (JVM). La JVMTI sustituye a la interfaz de perfilador de la máquina virtual Java (JVMPI) y a la interfaz del depurador de la máquina virtual Java (JVMDI). Encontrará los detalles en: “Interfaz de herramientas de la máquina virtual Java (JVMTI)” en la página 420.

Mejoras realizadas en la extensión de sockets seguros Java (JSSE)

JSSE 6 está disponible en la V6R1. Hallará más información en: “Utilizar la extensión de sockets seguros Java 6” en la página 323.

Nuevos mandatos CL Java

En la V6R1 se han añadido varios mandatos CL relacionados con Java:

- Si emplea la máquina virtual de tecnología IBM para Java, puede utilizar el mandato CL Trabajar con trabajos de JVM (WRKJVMJOB) para recoger datos de rendimiento. Vea: “Utilizar el mandato Trabajar con trabajos de JVM” en la página 422.
- Si utiliza la máquina virtual de tecnología IBM para Java, puede emplear el mandato CL Generar vuelco de JVM (GENJVMDMP) para generar vuelcos Java, del sistema y de la memoria dinámica. Vea: “Utilizar el mandato Generar vuelco de JVM” en la página 446.
- El mandato Imprimir trabajo de JVM (PRTJVMJOB) le permite imprimir máquinas virtuales Java (JVM) que se estén ejecutando en trabajos activos. Vea: “Mandatos CL soportados por Java” en la página 431.

JDBC 4.0

JDBC 4.0 está en conformidad con J2SE 6. Los cambios realizados en la interfaz MetaData en JDBC 4.0 se indican en: “Cambios en JDBC 4.0” en la página 68.

Cambios de vigencia

Se han actualizado los temas siguientes para que reflejen las opciones soportadas de 5761-JV1 en la V6R1:

- “Soporte para múltiples opciones del LP 5761-JV1” en la página 6
- “Instalar la máquina virtual de tecnología IBM para Java” en la página 4
- “Lista de propiedades Java del sistema” en la página 16

Cambios varios

- Ahora, el protocolo Internet versión 6 (IPv6) ya está plenamente soportado en la JVM clasica de IBM Developer Kit para Java y en la JVM de tecnología IBM para Java. Para obtener información general sobre IPv6, vea el Protocolo Internet versión 6 en el tema de configuración de TCP/IP, en Information Center.
- La configuración de huso horario ha cambiado en la V6R1. Hallará más información en: “Configuración del huso horario” en la página 25.
- Se han realizado varios cambios en la información de “NAWT (Native Abstract Windowing Toolkit)” en la página 248.
- Se ha añadido información de depuración nueva para la JVM de tecnología IBM para Java. Vea: “Depuración del sistema en el caso de la tecnología IBM para Java” en la página 434.

Cómo ver las novedades o los cambios

En esta información se utilizan las siguientes imágenes para ayudarle a ver dónde se han realizado cambios técnicos:

- La imagen señala el lugar en el que empieza la información nueva o cambiada.
- La imagen señala el lugar en el que acaba la información nueva o cambiada.

Para localizar más información sobre las novedades o los cambios realizados en este release, vea: Memorándum para los usuarios.

Archivo PDF de IBM Developer Kit para Java

Puede ver e imprimir un archivo PDF de esta información.

Para ver o descargar la versión PDF de este documento, seleccione IBM Developer Kit para Java (alrededor de 4585 KB).

Cómo guardar los archivos PDF

Si desea guardar un archivo PDF en su estación de trabajo para verlo o imprimirlo:

1. En el navegador, pulse el enlace del PDF con el botón derecho del ratón.
2. Pulse la opción que guarda el PDF localmente.
3. Navegue hasta el directorio en el que desea guardar el archivo PDF.
4. Pulse **Guardar**.

Cómo descargar Adobe Reader

Para poder ver o imprimir estos archivos PDF, debe instalar Adobe en su sistema. Puede descargar una copia gratuita desde el sitio Web de Adobe (www.adobe.com/products/acrobat/readstep.html) .

Instalar y configurar IBM Developer Kit para Java

Si todavía no ha utilizado IBM Developer Kit para Java, siga estos pasos para instalarlo, y practique con él ejecutando un sencillo programa Java Hello World.

"Novedades de la V6R1" en la página 1
Aquí encontrará la información nueva o la que ha cambiado notablemente en el temario de IBM Developer Kit para Java.

"Personalizar el System i5 para IBM Developer Kit para Java" en la página 13
Tras instalar IBM Developer Kit para Java en el servidor, podrá personalizar el servidor.

"Descargar e instalar paquetes Java" en la página 8
Utilice esta información para descargar, instalar y utilizar paquetes Java de manera más eficaz en la plataforma System i.

"Compatibilidad entre releases" en la página 33
En V6R1, el proceso directo ha dejado de estar soportado. Este cambio afecta a los parámetros OPTIMIZE y ENBPFRCOL.

Instalar IBM Developer Kit para Java

La instalación de IBM Developer Kit para Java le permite crear y ejecutar programas Java en su sistema.

El programa bajo licencia 5761-JV1 viene con los discos CD del sistema, por lo que se presintala JV1. Entre el mandato Ir a programa bajo licencia (GO LICPGM) y seleccione la opción 10 (Visualizar). Si no ve listado este programa bajo licencia, lleve a cabo los pasos siguientes:

1. Entre el mandato GO LICPGM en la línea de mandatos.
2. Seleccione la opción 11 (Instalar programa bajo licencia).
3. Elija la opción 1 (Instalar) para el programa bajo licencia (LP) 5761-JV1 *BASE, y seleccione la opción que coincida con el Java Development Kit (JDK) que se quiere instalar. Si la opción que desea instalar no se visualiza en la lista, puede añadirla a la misma especificando la opción 1 (Instalar) en el campo de opción. Entre 5761JV1 en el campo del programa bajo licencia, y teclee el número de opción en el campo de opción de producto.

   Nota: se puede instalar más de una opción a la vez.

Una vez que haya instalado IBM Developer Kit para Java en el servidor, puede optar por personalizar el sistema.

Conceptos relacionados

"Personalizar el System i5 para IBM Developer Kit para Java" en la página 13
Tras instalar IBM Developer Kit para Java en el servidor, podrá personalizar el servidor.

Tareas relacionadas

"Ejecutar el primer programa Java Hello World" en la página 9
Este tema le ayudará a ejecutar el primer programa Java.

Instalar la máquina virtual de tecnología IBM para Java

La máquina virtual de tecnología IBM para Java está disponible en ambas versiones, la de 32 bites y la de 64 bites. Siga estas instrucciones para instalar la máquina virtual de tecnología IBM para Java.

La máquina virtual de tecnología IBM para Java viene incluida en el programa bajo licencia 5761-JV1. El programa bajo licencia 5761-JV1 viene con los discos CD del sistema. Para acceder a la opción de tecnología IBM para Java, siga estos pasos:

1. Entre el mandato Ir a programa bajo licencia (GO LICPGM) y seleccione la opción 10 (Visualizar).
2. Si no ve este programa bajo licencia en la lista, lleve a cabo los pasos siguientes:
   a. Entre el mandato GO LICPGM en la línea de mandatos.
   b. Seleccione la opción 11 (Instalar programa bajo licencia).
   c. Elija la opción 1 (Instalar) para el programa bajo licencia (LP) 5761-JV1 *BASE, y seleccione la opción que desea instalar.
3. Añada la variable de entorno pertinente. En la línea de mandatos, teclee uno de estos mandatos:
   a. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit')`
   b. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk50/64bit')`
   c. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit')`
   d. `ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit')`

Si no está seguro de qué JVM está utilizando en este momento, puede averiguarlo mediante los siguientes métodos. Si ve `IBM J9 VM` en el resultado, está utilizando Tecnología IBM para Java.

- Busque en las anotaciones de trabajo el trabajo que contiene la JVM. Habrá un mensaje que indique qué JVM está utilizando.
- Como parte del mandato Java que utiliza para ejecutar la aplicación, añada `-showversion`. Verá una línea adicional que muestra la JVM que está utilizando.
- Desde qsh o qp2term, ejecute `java -version`.

Información relacionada

Releases y tamaños de los programas bajo licencia

Consideraciones a tener en cuenta al utilizar la máquina virtual de tecnología IBM para Java:

Tenga en cuenta estas consideraciones antes de utilizar la máquina virtual de tecnología IBM para Java.

Utilizar métodos nativos con Tecnología IBM para Java

Si desea utilizar la tecnología IBM para Java y tener programas que empleen métodos nativos, debe compilar los programas teniendo habilitado el almacenamiento teraespacio. Dado que el almacenamiento teraespacio no está habilitado por defecto, es probable que deba recompilarlos. Es necesario porque el objeto Java está en almacenamiento i5/OS PASE, que está correlacionado sobre el almacenamiento teraespacio, y se devuelve un puntero al almacenamiento teraespacio. Asimismo, las funciones de JNI (como GetxxxArrayRegion) tienen un parámetro que señala hacia una memoria intermedia en la que se colocan los datos. Esta puntero debe señalar hacia el almacenamiento teraespacio para que se habilite la función de JNI en i5/OS PASE con el fin de copiar los datos en este almacenamiento. Si no ha compilado el programa teniendo habilitado el almacenamiento teraespacio e intenta ejecutar el método nativo con tecnología IBM para Java, recibirá el mensaje de escape `MCH4443 (Modelo de almacenamiento no válido para el programa destino LOADLIB)`.

Autorización adoptada

La autorización adoptada de los programas Java no está soportada en la máquina virtual de tecnología IBM para Java.

Mensajes de diagnóstico y archivos

Cuando los métodos nativos ILE se encuentran con problemas, aparecerán mensajes en las anotaciones de trabajo. Cuando los métodos nativos de la máquina virtual de tecnología IBM para Java o de PASE se encuentran con problemas, los archivos de diagnóstico se volcarán en el IFS. Hay varios tipos de estos "archivos núcleo", como los siguientes: `core.*.dmp`, `javacore.*.txt`, `Snap*.trc` y `heapdump.*.phd`. El tamaño de los archivos oscila entre decenas de KB hasta cientos de MB. En la mayoría de los casos, los problemas graves producen archivos de mayor tamaño. Los archivos de gran tamaño pueden consumir rápida e inexorablemente grandes cantidades de espacio en el IFS. A pesar del espacio que consumen, estos archivos son útiles para la depuración. Siempre que sea posible, debe conservar estos archivos hasta que se haya podido resolver el problema subyacente.

Hallará más información en el tema Advanced control of dump agents del manual Java Diagnostics Guide.

Consideraciones en torno a la migración

Antes de migrar de la JVM clásica, que es una máquina virtual de 64 bites, a la versión de 32 bites de tecnología IBM para Java, tenga en cuenta que puede haber numerosas limitaciones al utilizar el entorno de 32 bites. Por ejemplo, la cantidad de memoria direccionable es mucho menor. En la modalidad de 32 bites, la memoria dinámica de objetos Java no puede sobrepasar los 3 gigabytes. También estará limitado a ejecutar unas 1000 hebras aproximadamente. Hallará más información en: "Soporte para múltiples opciones del LP 5761-JV1" en la página 6.

Conceptos relacionados

"Cambios realizados en la autorización adoptada en V6R1" en la página 258
El soporte de la autorización adoptada por perfil de usuario mediante programas Java se retira en V6R1. En este tema se explica cómo determinar si las aplicaciones emplean la autorización adoptada y cómo modificarlas para que se ajusten a este cambio.

Instalar un programa bajo licencia con el mandato Restaurar programa bajo licencia

Los programas que aparecen en la pantalla *Instalar programas bajo licencia* son aquellos que están soportados por la instalación de LICPGM cuando el servidor era nuevo. Ocasionalmente, quedan disponibles programas nuevos que no aparecían en la lista como programas bajo licencia en el servidor. Si este es el caso del programa que desea instalar, debe utilizar el mandato Restaurar programa bajo licencia (RSTLICPGM) para instalarlo.

Para instalar un programa bajo licencia con el mandato Restaurar programa bajo licencia (RSTLICPGM), siga estos pasos:

1. Coloque la cinta o el CD-ROM que contiene el programa bajo licencia en la unidad adecuada.
2. En la línea de mandatos de i5/OS, teclee:

   RSTLICPGM

   y pulse la tecla Intro.

   Aparece la pantalla *Restaurar programa bajo licencia (RSTLICPGM)*.
3. En el campo *Producto*, escriba el número de ID del programa bajo licencia que desea instalar.
4. En el campo *Dispositivo*, especifique el dispositivo de instalación.

   Nota: si instala desde una unidad de cintas, el ID de dispositivo está generalmente en el formato **TAP*xx***, donde ***xx*** es un número, como **01**.
5. Conserve los valores predeterminados para los demás parámetros de la pantalla *Restaurar programa bajo licencia*. Pulse la tecla Intro.
6. Aparecen más parámetros. Conserve también estos valores predeterminados. Pulse la tecla Intro. El programa empieza a instalarse.

Cuando el programa bajo licencia haya terminado de instalarse, aparecerá de nuevo la pantalla *Restaurar programa bajo licencia*.

Soporte para múltiples opciones del LP 5761-JV1

La plataforma del System i5 admite múltiples versiones de los Java Development Kits (JDK) y de la plataforma Java 2, Standard Edition.

Nota: En esta documentación, en función del contexto, el término JDK se refiere a cualquier versión soportada del JDK o de la plataforma Java 2, Standard Edition (J2SE). Normalmente, el contexto en el que aparece JDK incluye una referencia al número de versión y de release específicos.

El System i5 permite utilizar simultáneamente múltiples JDK, pero solo mediante múltiples máquinas virtuales Java. Una sola máquina virtual Java ejecuta un JDK especificado. Se puede ejecutar una máquina virtual Java por cada trabajo.

Localice el JDK que usted utiliza, o que desea utilizar, y seleccione la opción correspondiente para instalarlo. Vea: “Instalar IBM Developer Kit para Java” en la página 3, si desea instalar más de un JDK en un momento dado.

Con la JVM clásica, la propiedad java.version determina qué JDK hay que ejecutar. Una vez que esté en marcha una máquina virtual Java, el hecho de cambiar la propiedad java.version del sistema no afecta para nada al JDK. Si utiliza Tecnología IBM para Java, seleccionará qué opción de 5761-JV1 hay que ejecutar (y, por lo tanto, qué modalidad de JDK/bites), estableciendo la variable de entorno JAVA_HOME. Esto difiere de lo que ocurre con la JVM clásica, que puede usar la propiedad java.version del sistema como entrada.

En la siguiente tabla figuran las opciones soportadas para este release.

Opciones de 5761-JV1	JAVA_HOME	java.version
Opción 6 - Clásico 1.4	/QIBM/ProdData/Java400/jdk14/	1.4
Opción 7 - Clásico 5.0	/QIBM/ProdData/Java400/jdk15/	1.5
Opción 8 - Tecnología IBM para Java 5.0 32 bites	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit	1.5
Opción 9 - Tecnología IBM para Java 5.0 64 bites	/QOpenSys/QIBM/ProdData/JavaVM/jdk50/64bit	1.5
Opción 10 - Clásico 6	/QIBM/ProdData/Java400/jdk6	1.6

Opciones de 5761-JV1	JAVA_HOME	java.version
Opción 11 - Tecnología IBM para Java 6 32 bites	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit	1.6
Opción 12 - Tecnología IBM para Java 6 64 bites	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit	1.6

El JDK elegido por defecto en este entorno de múltiples JDK depende de qué opciones de 5761-JV1 se hayan instalado. La tabla siguiente ofrece algunos ejemplos. La información que sigue solo atañe a los JDK clásicos. Únicamente puede acceder a los JDK de Tecnología IBM para Java estableciendo la variable de entorno JAVA_HOME, o haciendo una llamada directa de un binario totalmente calificado.

Instale	Entre	Resultado
Opción 6 (1.4)	`java Hello`	Se ejecuta J2SDK, Standard Edition, versión 1.4.
Opción 7 (1.5) y opción 6 (1.4)	`java Hello`	Se ejecuta J2SDK, Standard Edition, versión 1.5.

Nota: Si instala un solo JDK, el JDK predeterminado es el que ha instalado. Si instala más de un JDK, el orden de prioridad siguiente determina cuál es el JDK predeterminado:

1. Opción 7 - Clásico 5.0
2. Opción 10 - Clásico 6
3. Opción 6 - Clásico 1.4
4. Opción 8 - Tecnología IBM para Java 5.0 32 bites
5. Opción 9 - Tecnología IBM para Java 5.0 64 bites
6. Opción 11- Tecnología IBM para Java 6 32 bites
7. Opción 12 - Tecnología IBM para Java 6 64 bites

Instalar extensiones de IBM Developer Kit para Java

Las extensiones son paquetes de clases Java que pueden utilizarse para ampliar las funciones de la plataforma central. Las extensiones se empaquetan en uno o más archivos ZIP o JAR y se cargan en la máquina virtual Java mediante un cargador de clases de extensión.

El mecanismo de extensión permite que la máquina virtual Java utilice las clases de extensión de la misma forma que la máquina virtual utiliza las clases del sistema. El mecanismo de extensión también proporciona una forma de recuperar las extensiones a partir de los localizadores uniformes de recursos (URL) especificados cuando aún no están instaladas en Java 2 Platform, Standard Edition (J2SE).

Algunos archivos JAR de extensiones vienen con i5/OS. Si desea instalar una de estas extensiones, entre el mandato:

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

Donde

`extensionToInstall.jar`

es el nombre del archivo ZIP o JAR que contiene la extensión que desea instalar.

Nota: Los archivos JAR de extensiones no suministradas por IBM pueden colocarse en el directorio /QIBM/UserData/Java400/ext.

Cuando crea un enlace o añade un archivo a una extensión del directorio /QIBM/UserData/Java400/ext, la lista de archivos en la que busca el cargador de clases de extensión cambia para *cada máquina virtual Java que se ejecute en el servidor*. Si no quiere que los cargadores de clases de extensión correspondientes a las otras máquinas virtuales Java del servidor resulten afectados, pero aún así desea crear un enlace a una extensión o bien instalar una extensión no suministrada por IBM con el servidor, siga estos pasos:

1. Cree un directorio para instalar las extensiones. Utilice el mandato Crear directorio (MKDIR) desde la línea de mandatos de i5/OS o el mandato `mkdir` desde el intérprete Qshell.
2. Coloque el archivo JAR de extensión en el directorio que ha creado.
3. Añada el directorio nuevo a la propiedad java.ext.dirs. Puede añadir el directorio nuevo a la propiedad java.ext.dirs utilizando el campo PROP del mandato JAVA desde la línea de mandatos de i5/OS.

Si el nombre del directorio nuevo es /home/username/ext, el nombre del archivo de extensión es extensionToInstall.jar y el nombre del programa Java es Hello, los mandatos que especifique deberán ser los siguientes:

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

CPY OBJ('/productA/extensionToInstall.jar') TODIR('/home/username/ext') o
copie el archivo en /home/username/ext utilizando FTP (protocolo de transferencia de archivos).

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

Descargar e instalar paquetes Java

Utilice esta información para descargar, instalar y utilizar paquetes Java de manera más eficaz en la plataforma System i.

Paquetes con interfaces gráficas de usuario

Los programas Java que se utilizan con una interfaz gráfica de usuario (GUI) requieren el uso de un dispositivo de presentación con posibilidades de visualización gráfica. Por ejemplo, se puede utilizar un PC, una estación de trabajo técnica o una máquina de red. Puede utilizar Native Abstract Windowing Toolkit (NAWT) para proporcionar a las aplicaciones y servlets Java las prestaciones completas de las funciones de gráficos de Abstract Windowing Toolkit (AWT) de Java 2 Platform, Standard Edition (J2SE). Para obtener más información, consulte el apartado Native Abstract Windowing Toolkit (NAWT)

La sensibilidad a las mayúsculas y minúsculas y el sistema de archivos integrado

El sistema de archivos integrado proporciona sistemas de archivos sensibles a las mayúsculas y minúsculas y también otros no sensibles a las mayúsculas y minúsculas por lo que a los nombres de archivo se refiere. QOpenSys es un ejemplo de sistema de archivos sensible a las mayúsculas y minúsculas dentro del sistema de archivos integrado. El sistema de archivos root, '//', es un ejemplo de sistema de archivos no sensible a las mayúsculas y minúsculas. Para obtener más información, consulte el tema Sistema de archivos integrado.

Aunque un JAR o una clase puede encontrarse en un sistema de archivos no sensible a mayúsculas y minúsculas, Java sigue siendo un lenguaje sensible a mayúsculas y minúsculas. Mientras que `wrklnk '/home/Hello.class'` y `wrklnk '/home/hello.class'` generan los mismos resultados, `JAVA CLASS(Hello)` y `JAVA CLASS(hello)` llaman a clases distintas.

Manejo de archivos ZIP y JAR

Los archivos ZIP y JAR contienen un conjunto de clases Java. Cuando se utliza el mandato Crear programa Java (CRTJVAPGM) en uno de estos archivos, se verifican las clases y estas se convierten a un formato máquina interno. Los archivos ZIP y JAR pueden recibir el mismo trato que cualquier otro archivo de clase individual. Si se asocia un formato máquina interno a uno de estos archivos, dicho formato permanece asociado al archivo. En las ejecuciones futuras y con objeto de mejorar el rendimiento,

se utilizará el formato máquina interno en lugar del archivo de clase. Si no está seguro de si existe un programa Java actual asociado al archivo de clase o al archivo JAR, utilice el mandato Visualizar programa Java (DSPJVAPGM) para visualizar información sobre el programa Java en el servidor.

En los releases anteriores de IBM Developer Kit para Java, era necesario volver a crear un programa Java si se cambiaba de algún modo el archivo JAR o ZIP, debido a que el programa Java conectado no hubiera podido utilizarse. Esto ya no es así. En muchos casos, si se cambia un archivo JAR o ZIP, el programa Java sigue siendo válido y no hace falta volver a crearlo. Si se efectúan cambios parciales, como al actualizar un solo archivo de clase dentro de un archivo JAR, solo es necesario volver a crear los archivos de clase afectados que se encuentran dentro del archivo JAR.

Los programas Java permanecen conectados al archivo JAR después de realizarse los cambios más habituales en el archivo JAR. Por ejemplo, estos programas Java permanecen conectados al archivo JAR después de:

- Cambiar o crear de nuevo un archivo JAR con la herramienta jar.
- Sustituir un archivo JAR con el mandato COPY de OS/400 o el programa de utilidad `cp` de Qshell.

Si accede a un archivo JAR del sistema de archivos integrado mediante System i Access para Windows o desde una unidad correlacionada de un PC, estos programas Java permanecen conectados al archivo JAR después de:

- Arrastrar otro archivo JAR y soltarlo en el archivo JAR del sistema de archivos integrado existente.
- Cambiar o crear de nuevo el archivo JAR del sistema de archivos integrado con la herramienta jar.
- Sustituir el archivo JAR del sistema de archivos integrado utilizando el mandato copy de PC.

Cuando se cambia o sustituye un archivo JAR, el programa Java conectado a él deja de ser actual.

Existe un único caso en el que los programas Java no permanecen conectados al archivo JAR. Los programas Java conectados se destruyen si se utiliza el protocolo de transferencia de archivos (FTP) para sustituir el archivo JAR. Esto ocurre, por ejemplo, si se utiliza el mandato `put` de FTP para sustituir el archivo JAR.

Encontrará información más detallada sobre las características de rendimiento de los archivos JAR en: “Consideraciones sobre el rendimiento de Java” en la página 414.

Infraestructura de extensiones Java

En J2SE, las extensiones son paquetes de clases Java que sirven para ampliar las funciones de la plataforma núcleo. Una extensión o aplicación está empaquetada en uno o más archivos JAR. El mecanismo de extensión permite que la máquina virtual Java utilice las clases de extensión de la misma forma que la máquina virtual utiliza las clases del sistema. El mecanismo de extensión también proporciona una forma de recuperar las extensiones a partir de los URL especificados cuando aún no están instaladas en J2SE o en Java 2 Runtime Environment, Standard Edition.

Para obtener información sobre cómo instalar extensiones, vea: “Instalar extensiones de IBM Developer Kit para Java” en la página 7.

Ejecutar el primer programa Java Hello World

Este tema le ayudará a ejecutar el primer programa Java.

Para ejecutar el programa Java Hello World, puede elegir uno de estos procedimientos:

1. Puede ejecutar sencillamente el programa Java Hello World que venía con IBM Developer Kit para Java.

   Para ejecutar el programa que se incluye, lleve a cabo los siguientes pasos:

a. Compruebe que IBM Developer Kit para Java está instalado, entrando el mandato Ir a programa bajo licencia (GO LICPGM). A continuación, seleccione la opción 10 (Visualizar programas bajo licencia instalados). Verifique que en la lista figuran como instalados el programa bajo licencia 5761-JV1 *BASE y al menos una de las opciones.
b. Entre `java Hello` en la línea de mandatos del menú principal de i5/OS. Pulse Intro para ejecutar el programa Java Hello World.
c. Si IBM Developer Kit para Java se ha instalado correctamente, aparece `Hello World` en la pantalla de la shell Java. Pulse F3 (Salir) o F12 (Salir) para volver a la pantalla de entrada de mandatos.
d. Si la clase Hello World no se ejecuta, compruebe si la instalación se ha realizado satisfactoriamente o consulte: "Obtener soporte para IBM Developer Kit para Java" en la página 580, donde podrá obtener información de servicio.

2. También puede ejecutar su propio programa Java Hello. Para saber cómo puede crear su propio programa Java Hello, vea: "Crear, compilar y ejecutar un programa Java HelloWorld" en la página 11.

Correlacionar una unidad de red con el servidor

Para correlacionar una unidad de red, siga estos pasos.

1. Asegúrese de que tiene instalado System i Access para Windows en el servidor y en la estación de trabajo. Para obtener más información sobre cómo instalar y configurar System i Access para Windows, vea: Instalar System i Access para Windows. Debe tener configurada una conexión para el servidor para poder correlacionar una unidad de red.
2. Abra el Explorador de Windows:
   a. Pulse el botón **Inicio** con el botón derecho del ratón en la barra de tareas de Windows.
   b. Pulse **Explorar** en el menú.
3. Seleccione **Correlacionar unidad de red** en el menú **Herramientas**.
4. Seleccione la unidad que desea utilizar para conectarse con el servidor.
5. Escriba el nombre de la vía de acceso al servidor. Por ejemplo, `\\`*`MISERVIDOR`*, siendo *MISERVIDOR* el nombre de su servidor.
6. Marque el recuadro **Reconectar al iniciar la sesión** si está en blanco.
7. Pulse **Aceptar** para finalizar.

La unidad correlacionada aparece ahora en la sección **Todas las carpetas** del Explorador de Windows.

Crear un directorio en el servidor

En el servidor, debe crear un directorio en el que poder guardar las aplicaciones Java.

Información relacionada

Cómo empezar con System i Navigator

Crear un directorio utilizando System i Navigator

Elija esta opción si tiene instalado System i Access para Windows. Si piensa utilizar System i Navigator para compilar y ejecutar el programa Java, debe seleccionar esta opción para asegurar que el programa se guarda en la ubicación correcta con el fin de realizar estas operaciones.

Para crear un directorio en el System i, siga estos pasos.

1. Abra System i Navigator.
2. Efectúe una doble pulsación sobre el nombre del servidor en la ventana **Mis conexiones** para iniciar la sesión. Si el servidor no aparece en la ventana **Mis conexiones**, siga estos pasos para añadirlo:
   a. Pulse **Archivo → Añadir conexión...**
   b. Escriba el nombre del servidor en el campo **Sistema**.
   c. Pulse **Siguiente**.

d. Si aún no lo ha hecho, especifique el ID de usuario en el campo **Utilizar ID de usuario por omisión, solicitar cuando sea necesario**.
e. Pulse **Siguiente**.
f. Pulse **Verificar conexión**. Así confirmará que puede conectarse al servidor.
g. Pulse **Finalizar**.

3. Expanda la carpeta situada bajo la conexión que desea utilizar. Busque una carpeta denominada **Sistemas de archivos**. Si no ve esta carpeta, es que no se ha seleccionado la opción para instalar los sistema de archivos durante la instalación de System i Navigator. Debe instalar la opción Sistemas de archivos de System i Navigator seleccionando **Inicio → Programas → System i Access para Windows → Instalación selectiva**.
4. Expanda la carpeta **Sistemas de archivos** y busque la carpeta **Sistema de archivos integrado**.
5. Expanda la carpeta **Sistema de archivos integrado** y, a continuación, expanda la carpeta **Raíz**. Al expandir la carpeta **Raíz**, verá la misma estructura que al ejecutar el mandato WRKLNK ('/') en la línea de mandatos de i5/OS.
6. Pulse con el botón derecho del ratón sobre la carpeta a la que desea añadir un subdirectorio. Seleccione **Nueva carpeta** y especifique el nombre del subdirectorio que desea crear.

Crear un directorio utilizando la línea de entrada de mandatos

Siga estas instrucciones para crear un directorio si no tiene instalado System i Access para Windows.

Para crear un directorio en el servidor, siga estos pasos.

1. Inicie sesión en el servidor.
2. En la línea de mandatos, escriba:

   ```
   CRTDIR DIR('/midir')
   ```

   donde *midir* es el nombre del directorio que está creando.

   Pulse la tecla **Intro**.

   Aparece un mensaje al final de la pantalla, que indica "**Directorio creado.**"

Crear, compilar y ejecutar un programa Java HelloWorld

La tarea de crear un programa Java Hello World constituye un excelente punto de partida para familiarizarse con IBM Developer Kit para Java.

Para crear, compilar y ejecutar su propio programa Java Hello World, lleve a cabo los siguientes pasos:

1. Correlacione una unidad de red con el sistema.
2. Cree un directorio en el servidor para sus aplicaciones Java.
3. Cree el archivo fuente como archivo de texto ASCII (American Standard Code for Information Interchange) en el sistema de archivos integrado. Puede utilizar un producto de entorno de desarrollo integrado (IDE) o un editor basado en texto (como el Bloc de notas de Windows) para escribir el código de la aplicación Java.
   a. Dé al archivo de texto el nombre `HelloWorld.java`.
   b. Asegúrese de que el archivo contiene el código fuente siguiente:

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

4. Compile el archivo fuente.
   a. Especifique el mandato Arrancar Qshell (STRQSH) para iniciar el intérprete Qshell.
   b. Utilice el mandato cd (cambiar de directorio) para pasar del directorio actual al directorio del sistema de archivos integrado que contiene el archivo `HelloWorld.java`.

c. Entre `javac` seguido del nombre del archivo tal y como lo haya guardado en el disco. Por ejemplo, entre `javac HelloWorld.java`.

5. Establezca las autorizaciones de archivo sobre el archivo de clase del sistema de archivos integrado.
6. Ejecute el archivo de clase.
   a. Asegúrese de que la vía de acceso de clases Java está bien configurada.
   b. En la línea de mandatos de Qshell, escriba `java` seguido de `HelloWorld` para ejecutar `HelloWorld.class` con la máquina virtual Java. Por ejemplo, especifique `java HelloWorld`. También puede utilizar el mandato Ejecutar Java (RUNJVA) en su sistema para ejecutar `HelloWorld.class`: RUNJVA CLASS(HelloWorld)
   c. Si se ha especificado todo correctamente, se mostrará `Hello World` en la pantalla. Si se ejecuta en el entorno Qshell, aparece el indicador de la shell (que por defecto es un signo $), para indicar que Qshell está preparado para otro mandato.
   d. Pulse F3 (Salir) o F12 (Desconectar) para volver a la pantalla de entrada de mandato.

También le resultará fácil compilar y ejecutar la aplicación Java empleando System i Navigator, que es una interfaz gráfica de usuario para realizar taras en el sistema.

"Correlacionar una unidad de red con el servidor" en la página 10
Para correlacionar una unidad de red, siga estos pasos.

"Crear un directorio en el servidor" en la página 10
En el servidor, debe crear un directorio en el que poder guardar las aplicaciones Java.

"Crear y editar archivos fuente Java"
Puede crear y editar archivos fuente Java de varias maneras: utilizando System i Access para Windows, en una estación de trabajo, con EDTF y con SEU.

"Vía de acceso de clases Java" en la página 13
La máquina virtual The Java (JVM) utiliza la vía de acceso de clases Java para localizar las clases en tiempo de ejecución. Los mandatos y las herramientas Java utilizan también la vía de acceso de clases para localizar las clases. La vía de acceso de clases por omisión del sistema, la variable de entorno CLASSPATH y el parámetro de mandato de vía de acceso de clases determinan en qué directorios se realiza la búsqueda cuando se desea hallar una clase determinada.

"Autorizaciones de archivo Java en el sistema de archivos integrado" en la página 247
Para ejecutar o depurar un programa Java, es necesario que el archivo de clase, JAR o ZIP tenga autorización de lectura (*R). Los directorios necesitan la autorización de lectura y la de ejecución (*RX).

Mandato Crear programa Java (CRTJVAPGM)

Mandato Ejecutar Java (RUNJVA)

"Mandatos de System i Navigator soportados por Java" en la página 432
System i Navigator es una interfaz gráfica del escritorio Windows. Forma parte de System i Access para Windows y cubre muchas de las funciones de i5/OS que los necesitan los administradores o los usuarios para llevar a cabo su trabajo diario. Los mandatos de System i Navigator le permiten crear y ejecutar programas Java.

Qué es System i Navigator

Crear y editar archivos fuente Java

Puede crear y editar archivos fuente Java de varias maneras: utilizando System i Access para Windows, en una estación de trabajo, con EDTF y con SEU.

Con System i Access para Windows

Los archivos fuente Java son archivos de texto ASCII (American Standard Code for Information Interchange) del sistema de archivos integrado.

Puede crear y editar un archivo fuente Java con System i Access para Windows y un editor basado en estación de trabajo.

En una estación de trabajo

Se puede crear un archivo fuente Java en una estación de trabajo. A continuación, hay que transferirlo al sistema de archivos integrado utilizando para ello el protocolo de transferencia de archivos (FTP).

Para crear y editar archivos fuente Java en una estación de trabajo:

1. Cree el archivo ASCII en la estación de trabajo con el editor que prefiera.
2. Conéctese al servidor con FTP.
3. Transfiera el archivo fuente al directorio del sistema de archivos integrado como archivo binario, para que así conserve el formato ASCII.

Con EDTF

El mandato CL Editar archivo (EDTF) le permite editar archivos de cualquier sistema de archivos. EDTF es un editor similar al programa de utilidad para entrada del fuente (SEU) para editar archivos continuos o archivos de base de datos. Hallará información en: Mandato CL Editar archivo (EDTF).

Con el programa de utilidad para entrada del fuente

Puede crear un archivo fuente Java como archivo de texto si emplea el programa de utilidad para entrada del fuente (SEU).

Para crear el archivo fuente Java como archivo de texto mediante SEU, siga estos pasos:

1. Cree un miembro de archivo fuente con SEU.
2. Utilice el mandato Copiar en archivo continuo (CPYTOSTMF) para copiar el miembro de archivo fuente en un archivo continuo del sistema de archivos integrado y convertir al mismo tiempo los datos a formato ASCII.

Si ha de realizar cambios en el código fuente, cambie el miembro de base de datos con SEU y copie de nuevo el archivo.

Para obtener información sobre cómo almacenar archivos, vea: “Archivos relacionados con Java en el IFS” en la página 247.

Personalizar el System i5 para IBM Developer Kit para Java

Tras instalar IBM Developer Kit para Java en el servidor, podrá personalizar el servidor.

Vía de acceso de clases Java

La máquina virtual The Java (JVM) utiliza la vía de acceso de clases Java para localizar las clases en tiempo de ejecución. Los mandatos y las herramientas Java utilizan también la vía de acceso de clases para localizar las clases. La vía de acceso de clases por omisión del sistema, la variable de entorno CLASSPATH y el parámetro de mandato de vía de acceso de clases determinan en qué directorios se realiza la búsqueda cuando se desea hallar una clase determinada.

En Java 2 Platform, Standard Edition (J2SE), la propiedad java.ext.dirs determina la vía de acceso de clases de las extensiones que se cargan. Consulte “Instalar extensiones de IBM Developer Kit para Java” en la página 7 para obtener más información.

La vía de acceso de clases de rutina de carga por omisión está definida por el sistema y no debe cambiarse. En el servidor, la vía de acceso de clases de rutina de carga por omisión especifica dónde se encuentran las clases que forman parte de IBM Developer Kit para Java, de Native Abstract Window Toolkit (NAWT) y otras clases del sistema.

Para hallar cualquier otra clase en el sistema, debe especificar la vía de acceso de clases en la que debe realizarse la búsqueda; para ello, utilice la variable de entorno CLASSPATH o el parámetro de vía de acceso de clases. El parámetro de vía de acceso de clases que se utilice en una herramienta o en un mandato prevalecerá sobre el valor especificado en la variable de entorno CLASSPATH.

Para trabajar con la variable de entorno CLASSPATH, utilice el mandato Trabajar con variable de entorno (WRKENVVAR). Desde la pantalla WRKENVVAR, se puede añadir o cambiar la variable de entorno CLASSPATH. Los mandatos Añadir variable de entorno (ADDENVVAR) y Cambiar variable de entorno (CHGENVVAR) añaden y cambian, respectivamente, la variable de entorno CLASSPATH.

El valor de la variable de entorno CLASSPATH es una lista de nombres de vías de acceso, separados por el signo de dos puntos (:), en las que se busca una clase determinada. Un nombre de vía de acceso es una secuencia de cero o más nombres de directorio. Estos nombres de directorio van seguidos del nombre del directorio, el archivo ZIP o el archivo JAR en el que se ha de realizar la búsqueda en el sistema de archivos integrado. Los componentes del nombre de vía de acceso van separados por medio del carácter barra inclinada (/). Utilice un punto (.) para indicar cuál es el directorio de trabajo actual.

Para establecer la variable CLASSPATH del entorno Qshell, puede emplear el programa de utilidad de exportación que está disponible al utilizar el intérprete Qshell.

| Estos mandatos añaden la variable CLASSPATH al entorno Qshell y establecen que tenga el valor
| `".:/myclasses.zip:/Product/classes"`

- El mandato siguiente establece la variable CLASSPATH del entorno Qshell:

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

- Este mandato establece la variable CLASSPATH desde la línea de mandatos:

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

J2SE busca primero en la vía de acceso de clases de rutina de carga, a continuación en los directorios de extensiones y finalmente en la vía de acceso de clases. El orden de búsqueda de J2SE, utilizando el ejemplo anterior, es:

1. La vía de acceso de clases de rutina de carga, que se encuentra en la propiedad sun.boot.class.path,
2. Los directorios de extensiones, que se encuentran en la propiedad java.ext.dirs,
3. El directorio de trabajo actual,
4. El archivo myclasses.zip que se encuentra en el sistema de archivos ″root″ (/),
5. El directorio de clases del directorio Product que hay en el sistema de archivos ″root″ (/).

Al entrar en el entorno Qshell, la variable CLASSPATH queda establecida en la variable de entorno. El parámetro de vía de acceso de clases especifica una lista de nombres de vía de acceso. La sintaxis es idéntica a la de la variable de entorno CLASSPATH. Las herramientas y mandatos que disponen de un parámetro de vía de acceso de clases son:

- Mandato `java` de Qshell
- Herramienta `javac`
- Herramienta `javah`
- Herramienta `javap`
- Herramienta `javadoc`
- Herramienta `rmic`
- Mandato Ejecutar Java (RUNJVA)

Hallará más información sobre estos mandatos en: “Mandatos y herramientas de IBM Developer Kit para Java” en la página 422. Si utiliza el parámetro de vía de acceso de clases con cualquiera de estos mandatos o herramientas, el parámetro hará caso omiso de la variable de entorno CLASSPATH.

Puede alterar temporalmente la variable de entorno CLASSPATH utilizando la propiedad java.class.path. Puede cambiar la propiedad java.class.path, así como otras propiedades, utilizando el archivo SystemDefault.properties. Los valores del archivo SystemDefault.properties alteran temporalmente la variable de entorno CLASSPATH. Para obtener información sobre el archivo SystemDefault.properties, vea: “Archivo SystemDefault.properties” en la página 16.

En J2SE, la opción -Xbootclasspath también afecta a los directorios en que el sistema busca al buscar clases. El hecho de utilizar -Xbootclasspath/a:*path* añade *path* a la vía de acceso de clases de rutina de carga por omisión, /p:*path* coloca *path* antes de la vía de acceso de clases de rutina de carga, y :*path* sustituye a la vía de acceso de clases de rutina de carga por *path*.

Nota: Tenga cuidado al especificar -Xbootclasspath porque pueden producirse resultados imprevistos si no se encuentra una clase del sistema o si esta se sustituye de forma incorrecta por una clase definida por el usuario. Por lo tanto, debería permitir que la búsqueda de clases se realice primero en la vía de acceso de clases por omisión del sistema, antes que en una vía de acceso de clases especificada por el usuario.

Para obtener información sobre cómo determinar el entorno en el que se ejecutan los programas Java, vea: “Propiedades Java del sistema”.

Para obtener más información, consulte las secciones Las API de programa y mandato CL o Sistema de archivos integrado.

Propiedades Java del sistema

Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Al iniciar una instancia de una máquina virtual Java (JVM) se establecen los valores de las propiedades del sistema que afectan a esa JVM.

Puede optar por utilizar los valores predeterminados para las propiedades Java del sistema o bien puede especificar valores para ellas siguiendo estos métodos:

- Añadir parámetros a la línea de mandatos (o a la API de invocación de la interfaz Java nativa (JNI)) al iniciar el programa Java.
- Utilizar la variable de entorno de nivel de trabajo QIBM_JAVA_PROPERTIES_FILE para señalar hacia un archivo de propiedades específico. Por ejemplo:

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

- Crear un archivo SystemDefault.properties que se crea en el directorio user.home.
- Utilizar el archivo /QIBM/userdata/java400/SystemDefault.properties.

El i5/OS y la JVM determinan los valores de las propiedades Java del sistema, siguiendo este orden de prioridad:

1. Línea de mandatos o API de invocación de JNI
2. Variable de entorno QIBM_JAVA_PROPERTIES_FILE
3. Archivo user.home SystemDefault.properties
4. /QIBM/UserData/Java400/SystemDefault.properties
5. Valores de propiedades del sistema por omisión

Archivo SystemDefault.properties

El archivo SystemDefault.properties es un archivo de propiedades Java estándar que le permite especificar propiedades predeterminadas del entorno Java.

El archivo SystemDefault.properties que se encuentra en el directorio inicial tiene prioridad sobre el archivoSystemDefault.properties que se encuentra en el directorio /QIBM/UserData/Java400.

Las propiedades que establezca en el archivo /YourUserHome/SystemDefault.properties afectarán solamente a las siguientes máquinas virtuales Java específicas:

- Las JVM que inicie sin especificar una propiedad user.home distinta
- Las JVM que inicien otros usuarios especificando la propiedad `user.home = /YourUserHome/`

Ejemplo: archivo SystemDefault.properties

El ejemplo que sigue establece varias propiedades Java:

```
#Los comentarios empiezan por el signo de almohadilla
#Utilizar J2SE 1.5
java.version=1.5
#Esto establece una propiedad especial
myown.propname=6
```

Lista de propiedades Java del sistema

Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Al iniciar una máquina virtual Java (JVM) se establecen las propiedades del sistema para esa instancia de la JVM. Para obtener más información sobre cómo especificar valores para propiedades Java del sistema, vea las siguientes páginas:

- “Propiedades Java del sistema” en la página 15
- “Archivo SystemDefault.properties”

Si desea más información sobre las propiedades Java del sistema, vea: “Propiedades Java del sistema en JSSE 1.4” en la página 287 y “Propiedades Java del sistema en JSSE 1.5” en la página 305.

En la tabla que sigue figuran las propiedades Java del sistema para las opciones soportadas de 5761-JV1 clásico.

Para cada propiedad, la tabla indica el nombre de la propiedad y los valores predeterminados pertinentes o bien una breve descripción. La tabla indica qué propiedades del sistema tienen valores distintos en las diferentes versiones de la plataforma Java 2, Standard Edition (J2SE). Cuando la columna en la que figuran los valores predeterminados no indica versiones distintas de J2SE, todas las versiones soportadas de J2SE utilizan ese valor predeterminado.

awt.toolkit	sun.awt.motif.MToolkit awt.toolkit quedará sin establecer a menos que os400.awt.native=true o java.awt.headless=true
file.encoding	ISO8859_1 (valor por predeterminado) Correlaciona el identificador de juego de caracteres codificado (CCSID) con el CCSID ASCII ISO correspondiente. Asimismo, establece que el valor de file.encoding es igual al valor Java que representa el CCSID ASCII ISO. Si desea obtener una tabla que muestre la relación entre los valores posibles de file.encoding y el CCSID que más se aproxima, vea: “Valores de file.encoding y CCSID de System i5” en la página 27.
file.encoding.pkg	sun.io

file.separator	/ (barra inclinada)
i5os.crypto.device	Especifica el dispositivo criptográfico que hay que utilizar. Si no se establece esta propiedad, se emplea el dispositivo predeterminado CRP01.
i5os.crypto.keystore	Especifica el archivo de almacén de claves CCA que hay que utilizar. Si no se establece esta propiedad, se emplea el archivo de almacén de claves mencionado en la descripción de dispositivo criptográfico.
java.awt.headless	• J2SE v1.4: false • J2SE 5.0: false (valor predeterminado) • JDK 6: false Esta propiedad especifica si la API de Abstract Windowing Toolkit (AWT) opera en modalidad acéfala o no. El valor por predeterminado false hace que la función completa de AWT solamente esté disponible al habilitar AWT estableciendo os400.awt.native como true. Establecer esta propiedad como true da soporte a la modalidad AWT headless y también fuerza explícitamente que os400.awt.native sea true.
java.class.path	. (punto) (valor por predeterminado) Designa la vía de acceso que i5/OS utiliza para localizar clases. Toma por defecto la CLASSPATH especificada por el usuario.
java.class.version	• J2SE v1.4: 48.0 • J2SE 5.0: 49.0 • JDK 6: 50.0
java.compiler	• jitc (valor predeterminado) - Especifica que el código se va a compilar con el compilador Just-In-Time (JIT) (jitc). • jitc_de - Este valor se mantiene por cuestión de compatibilidad con los releases anteriores. El proceso directo no está soportado en V6R1, por lo que este valor provocará el mismo comportamiento de tiempo de ejecución que el valor de la propiedad jitc. • NONE - Especifica que todo el código se ejecutará utilizando el intérprete de bytecode.
java.ext.dirs	J2SE v1.4: • /QIBM/ProdData/OS400/Java400/jdk/lib/ext: • /QIBM/ProdData/Java400/jdk14/lib/ext: • /QIBM/UserData/Java400/ext J2SE 5.0: (valor predeterminado) • /QIBM/ProdData/Java400/jdk15/lib/ext: • /QIBM/UserData/Java400/ext JDK 6: • /QIBM/ProdData/Java400/jdk6/lib/ext • /QIBM/UserData/Java400/ext
java.home	J2SE v1.4: /QIBM/ProdData/Java400/jdk14 J2SE v1.5: /QIBM/ProdData/Java400/jdk15 (valor predeterminado) JDK 6: /QIBM/ProdData/Java400/jdk6 Esta propiedad solo se utiliza para datos de salida. Encontrará los detalles en: “Soporte para múltiples opciones del LP 5761-JV1” en la página 6.
java.library.path	Lista de bibliotecas de i5/OS

java.net.preferIPv4Stack	• false (no's) - valor predeterminado • true En las máquinas de pila dual, las propiedades del sistema se proporcionan para establecer la pila de protocolo preferida (IPv4 o IPv6), así como los tipos de familias de direcciones preferidas (inet4 o inet6). La pila IPv6 es la preferida por defecto, ya que en una máquina de pila dual el socket IPv6 puede comunicarse con iguales IPv4 y IPv6. Este valor puede cambiarse con esta propiedad. Hallará más información en el manual Networking IPv6 User Guide.
java.net.preferIPv6Addresses	• true • false (no's) (valor predeterminado) Aunque IPv6 está disponible en el sistema operativo, la preferencia predeterminada es preferir una dirección correlacionada con IPv4 antes que una dirección IPv6. Esta propiedad controla si se utilizan direcciones IPv6 (true) o IPv4 (false). Hallará más información en el manual Networking IPv6 User Guide.
java.policy	J2SE v1.4: /QIBM/ProdData/OS400/ Java400/jdk/lib/security/java.policy J2SE v5.0: /QIBM/ProdData/ Java400/jdk15/lib/security/java.policy (valor predeterminado) JDK 6: /QIBM/ProdData/Java400/jdk6/lib/security/java.policy
java.specification.name	Especificación de la API de plataforma Java (valor predeterminado)
java.specification.vendor	Sun Microsystems, Inc.
java.specification.version	• J2SE v1.4: 1.4 • J2SE v5.0: 1.5 (valor predeterminado) • JDK 6: 1.6
java.use.policy	true
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
java.version	• 1.4.2 • 1.5.0 (valor predeterminado) • 1.6.0 Determina qué versión de J2SE desea ejecutar. Esta propiedad solo es válida cuando se utiliza la JVM clásica. Si se utiliza la tecnología IBM para Java, la propiedad "java.version" se ignora; la versión de JDK se determina a partir del valor de la variable de entorno JAVA_HOME. Si se instala una sola versión de J2SE, esa será la versión predeterminada. Si se especifica una versión que no esté instalada, se obtiene un mensaje de error. Si no se especifica una versión, se utiliza la versión más reciente de J2SE como valor predeterminado.
java.vm.name	VM clásica
java.vm.specification.name	Especificación de la máquina virtual Java
java.vm.specification.vendor	Sun Microsystems, Inc.
java.vm.specification.version	1.0
java.vm.vendor	IBM Corporation

java.vm.version	• J2SE v1.4: 1.4 • J2SE v5.0: 1.5 (valor predeterminado) • JDK 6: 1.6
line.separator	\n
os.arch	PowerPC
os.name	OS/400
os.version	V6R1M0 (valor predeterminado) Obtiene el nivel de release de i5/OS a partir de la interfaz de programación de aplicaciones (API) Recuperar información de producto.
os400.awt.native	Controla si la API de Abstract Windowing Toolkit (AWT) está soportada o no. Los valores válidos son true y false. El valor predeterminado es false a menos que se establezca java.awt.headless=true, en cuyo caso os400.awt.native es true implícitamente.
os400.certificateContainer	Indica al soporte de capa de sockets segura (SSL) de Java que utilice el contenedor de certificados especificado para el programa Java que se ha iniciado y la propiedad que se ha especificado. Si especifica la propiedad os400.secureApplication del sistema, se hace caso omiso de esta propiedad del sistema. Por ejemplo, teclee `-Dos400.certificateContainer=/home/username/mykeyfile.kdb` o cualquier otro archivo de claves del sistema de archivos integrado.
os400.certificateLabel	Puede especificar esta propiedad del sistema junto con la propiedad os400.certificateContainer del sistema. Esta propiedad le permite seleccionar qué certificado del contenedor especificado desea que la capa de sockets segura (SSL) utilice. Por ejemplo, entre `-Dos400.certificateLabel=myCert`, donde myCert es el nombre de etiqueta que usted asigna al certificado por medio del Gestor de certificados digitales (DCM) al crear o importar el certificado.
os400.child.stdio.convert	Controla la conversión de datos para stdin, stdout y stderr en Java. La conversión entre datos ASCII y datos EBCDIC (Extended Binary Coded Decimal Interchange Code) se produce por defecto en la máquina virtual Java (JVM). El hecho de utilizar esta propiedad para activar y desactivar estas conversiones solo afecta a los procesos hijo que este proceso inicie con el método Runtime.exec(). El valor de esta propiedad pasa a ser el valor predeterminado de os400.stdio.convert en los procesos hijo. Vea: “Valores de las propiedades os400.stdio.convert y os400.child.stdio.convert del sistema” en la página 23.
os400.class.path.security.check	20 (valor predeterminado) Valores válidos: • 0 Sin comprobación de seguridad • 10 Equivalente a RUNJVA CHKPATH(*IGNORE) • 20 Equivalente a RUNJVA CHKPATH(*WARN) • 30 Equivalente a RUNJVA CHKPATH(*SECURE)

os400.class.path.tools	0 (valor predeterminado) Valores válidos: • 0: No hay herramientas Sun en la propiedad java.class.path • 1: Añade el archivo de herramietas específico de J2SE como prefijo de la propiedad java.class.path En J2SE v1.4, la vía de acceso a tools.jar es: /QIBM/ProdData/OS400/Java400/jdk/lib/ En J2SE v5.0, la vía de acceso a tools.jar es: /QIBM/ProdData/Java400/jdk15/lib/ En JDK 6, la vía de acceso a tools.jar es: /QIBM/ProdData/Java400/jdk6/lib/
os400.create.type	Los valores válidos son: • interpret • direct Estos dos valores se ignoran. Si hay que crear un programa Java, el programa se creará sin código de ejecución directa.
os400.define.class.cache.file	el valor predeterminado es /QIBM/ProdData/Java400/QDefineClassCache.jar Especifica el nombre de un archivo JAR o ZIP. Vea ″Utilizar la caché para los cargadores de clases de usuario″ en Consideraciones sobre el rendimiento de Java.
os400.define.class.cache.hour	• valor predeterminado = 768 • valor decimal máximo = 9999 Especifica un valor decimal. Vea ″Utilizar la caché para los cargadores de clases de usuario″ en Consideraciones sobre el rendimiento de Java.
os400.define.class.cache.maxpgms	• valor predeterminado = 20000 • valor decimal máximo = 40000 Especifica un valor decimal. Vea ″Utilizar la caché para los cargadores de clases de usuario″ en Consideraciones sobre el rendimiento de Java.
os400.defineClass.optLevel	En la V6R1, se ignoran todos los valores. Si hay que crear un programa Java, el programa se creará sin código de ejecución directa.
os400.display.properties	Si se establece que este valor sea igual a 'true', se imprimirán todas las propiedades de la máquina virtual Java en la salida estándar. No se reconocen otros valores.
os400.enbpfrcol	Esta propiedad indica al compilador just-in-time (JIT) que genere ganchos de recogida de rendimiento en el código generado por JIT. Los valores válidos son: • 0 - valor predeterminado. No se pueden recoger datos de rendimiento desde el código compilado por JIT. • 1 - el compilador JIT genera eventos *JVAENTRY y *JVAEXIT. • 7 - Equivale al valor 1.
os400.exception.trace	Esta propiedad se utiliza para la depuración. Si se especifica esta propiedad, las excepciones más recientes se envían a la salida estándar cuando se sale de la VM.

os400.file.create.auth, os400.dir.create.auth	Estas propiedades especifican las autorizaciones asignadas a los archivos y directorios. Si se especifican las propiedades sin valores o con valores no soportados, la autorización de uso público es *NONE. Puede especificar os400.file.create.auth=RWX o os400.dir.create.auth=RWX, donde R=lectura, W=escritura y X=ejecución. Cualquier combinación de estas autorizaciones es válida.
os400.file.io.mode	Convierte el CCSID del archivo si es diferente del valor de file.encoding al especificar TEXT, en lugar del valor predeterminado, que es BINARY.
os400.gc.heap.size.init	Una alternativa a utilizar -Xms (establecer el tamaño de GC inicial). Se recomienda que los clientes continúen utilizando -Xms a menos que no tengan otra opción za que esta propiedad es específica de i5/OS. Esta propiedad se introdujo principalmente para que los usuarios puedan especificar un tamaño de GC inicial en el archivo SystemDefault.properties. **Nota:** Utilice esta propiedad con cuidado; ya que prevalecerá sobre -Xms si se especifica. El valor debe ser un entero en tamaño de kilobytes y sin comas.
os400.gc.heap.size.max	Una alternativa a utilizar -Xmx (establecer el tamaño de GC máximo). Se recomienda que los clientes continúen utilizando -Xmx a menos que no tengan otra opción za que esta propiedad es específica de i5/OS. Esta propiedad permite especificar un tamaño de GC máximo en el archivo SystemDefault.properties. **Nota:** Utilice esta propiedad con cuidado; ya que prevalecerá sobre -Xmx si se especifica. El valor debe ser un entero en tamaño de kilobytes y sin comas.
os400.interpret	• 0 (valor predeterminado) Equivalente a CRTJVAPGM INTERPRET(*NO) • 1 Equivalente a CRTJVAPGM INTERPRET(*YES)
os400.jit.mmi.threshold	Establece el número de veces que un método se ejecuta utilizando el MMI (Mixed-Mode Interpreter) antes de que i5/OS utilice el compilador JIT para compilar el método en instrucciones de máquina nativa. Normalmente no deberá cambiar el valor predeterminado, que es 2000. • El valor cero inhabilita MMI y compila métodos cuando se les llama por primera vez. • Los valores inferiores al valor predeterminado tienden a prolongar el tiempo de arranque y degradar el rendimiento general. • Los valores superiores al valor predeterminado degradan el rendimiento inicialmente hasta que se alcanza el umbral, tendiendo entonces a mejorar el rendimiento de ejecución general.
os400.job.file.encoding	Esta propiedad solo se utiliza para datos de salida. Enumera la codificación de archivo del trabajo i5/OS en el que está la JVM.
os400.optimization	Los valores válidos son 0, 10, 20, 30 y 40. En la V6R1, se ignoran todos los valores. Si hay que crear un programa Java, el programa se crea sin código de ejecución directa.
os400.pool.size	Define cuánto espacio (en kilobytes) se debe establecer como disponible para cada agrupación de almacenamiento dinámico en el almacenamiento dinámico local de hebras.

os400.run.mode	• jitc (valor predeterminado) Utilizar el compilador JIT para determinar cómo se crea y ejecuta el código • jitc_de Es igual que el valor predeterminado de jitc: utilizar el compilador JIT para determinar cómo se genera y ejecuta el código. • interpret Equivale a RUNJVA OPTIMIZE(*INTERPRET) y INTERPRET(*OPTIMIZE), o a INTERPRET(*YES) • program_create_type Utilizar el compilador JIT para determinar cómo se crea y ejecuta el código
os400.run.verbose	Si se establece este valor como 'true', se imprimirá la carga de clases verbose como salida estándar. No se reconocen otros valores. Se consigue lo mismo que especificando -verbose en QSHELL u OPTION(*VERBOSE) en los mandatos CL, excepto que esta propiedad funciona en el archivo SystemDefault.properties.
os400.runtime.exec	• EXEC (valor predeterminado) Invocar funciones mediante runtime.exec() utilizando la interfaz EXEC. • QSHELL Invocar funciones mediante runtime.exec() utilizando el intérprete Qshell. Para obtener más información, consulte "Utilizar java.lang.Runtime.exec()" en la página 230.
os400.secureApplication	Asocia el programa Java que se inicia al utiliza esta propiedad del sistema (os400.secureApplication) al nombre de la aplicación segura registrada. Para ver los nombres de las aplicaciones seguras registradas, utilice el Gestor de certificados digitales (DCM).
os400.security.properties	Permite tener pleno control de qué archivo java.security se utiliza. Si se especifica esta propiedad, J2SE no utilizará ningún otro archivo java.security, ni siquiera el valor predeterminado java.security específico de J2SE.
os400.stderr	Permite correlacionar stderr con un archivo o un socket. Vea: "Valores de las propiedades os400.stdin, os400.stdout y os400.stderr del sistema" en la página 24.
os400.stdin	Permite correlacionar stdin con un archivo o un socket. Vea: "Valores de las propiedades os400.stdin, os400.stdout y os400.stderr del sistema" en la página 24.
os400.stdin.allowed	1 (valor predeterminado) Especifica si stdin está permitido (1) o no (0). Si el llamador está ejecutando un trabajo por lotes, stdin no debe estar permitido.
os400.stdio.convert	Permite controlar la conversión de datos para la entrada estándar, la salida estándar y la salida de errores estándar en Java. La conversión de datos se produce por defecto en la máquina virtual Java para convertir los datos entre ASCII y EBCDIC. Esta propiedad le permite activar y desactivar las conversiones, lo cual afecta al programa Java actual. Vea: "Valores de las propiedades os400.stdio.convert y os400.child.stdio.convert del sistema" en la página 23. En el caso de los programas Java iniciados con el método Runtime.exec(), vea: os400.child.stdio.convert.
os400.stdout	Permite correlacionar stdout con un archivo o un socket. Consulte los valores predeterminados.
os400.xrun.option	Esta propiedad del sistema permite utilizar la opción Qshell -Xrun especificando una propiedad. Puede utilizarla para especificar un programa agente para ejecutarse durante el inicio de la JVM.

os400.verify.checks.disable	65535 (valor predeterminado) Este valor de propiedad del sistema es una serie que representa la suma de uno o más valores numéricos. Hallará una lista de estos valores en: "Valores de la propiedad os400.verify.checks.disable del sistema" en la página 25.
os400.vm.inputargs	Esta propiedad solo se utiliza para datos de salida. Visualizará los argumentos que la máquina virtual ha recibido como entradas. Esta propiedad puede ser muy útil para depurar lo especificado al arrancar la JVM.
path.separator	: (dos puntos)
sun.boot.class.path	Enumera todos los archivos necesarios para el cargador de clases de arranque predeterminado. No cambie este valor.
user.dir	Directorio de trabajo actual que utiliza la API getcwd.
user.home	Recupera el directorio de trabajo inicial utilizando la API Obtener (getpwnam). Puede colocar un archivo SystemDefault.properties en la vía de acceso user.home para alterar temporalmente las propiedades predeterminadas en /QIBM/UserData/Java400/SystemDefault.properties. Puede personalizar el sistema para que especifique su propio conjunto de valores de propiedades predeterminadas.
user.language	La máquina virtual Java emplea esta propiedad del sistema para leer el valor de LANGID del trabajo y utilizarlo con el fin de localizar el correspondiente idioma.
user.name	La máquina virtual Java utiliza esta propiedad del sistema para recuperar el verdadero nombre del perfil de usuario desde la sección de seguridad (Security.UserName) de la base informática de confianza (TCB).
user.region	La máquina virtual Java utiliza esta propiedad del sistema para leer el valor de CNTRYID del trabajo y lo emplea para determinar la región del usuario.
user.timezone	• La JVM se dirige primero al objeto QLOCALE del sistema. • Si no se encuentra, entonces la JVM se dirigirá al valor del sistema QTIMZON. El campo 'Nombre alternativo' del objeto QTIMZON sirve para asignar la propiedad Java user.timezone de la JVM. El valor del campo 'Nombre alternativo' debe constar como mínimo de 3 caracteres, de lo contrario, no se utilizará. • Si el campo 'Nombre alternativo' del objeto QTIMZON tiene menos de 3 caracteres, la JVM intentará localizar un valor de GMT coincidente basándose en la diferencia horaria actual del sistema. Ejemplo: un objeto QTIMZON cuyo campo Nombre Alternativo esté vacío y cuya diferencia horaria sea igual a -5 daría como resultado el valor `user.timezone=GMT-5`. • Si todavía no se ha encontrado un valor, la JVM toma por defecto una user.timezone igual a la hora universal coordinada (UTC). Hallará más información en: Los ID de huso horario que se pueden especificar para la propiedad user.timezone, en WebSphere Software Information Center.

Conceptos relacionados

"Personalizar el System i5 para IBM Developer Kit para Java" en la página 13
Tras instalar IBM Developer Kit para Java en el servidor, podrá personalizar el servidor.

Valores de las propiedades os400.stdio.convert y os400.child.stdio.convert del sistema:

En las tablas que siguen figuran los valores del sistema para las propiedades os400.stdio.convert y os400.child.stdio.convert del sistema.

Tabla 1. Valores del sistema para **os400.stdio.convert**

Valor	Descripción
Y (predeterminado)	Se convierten todos los datos de stdio durante las operaciones de lectura o escritura, y la conversión se realiza del valor de file.encoding al CCSID del trabajo o viceversa.
N	No se realiza ninguna conversión de stdio durante las operaciones de lectura o escritura.
1	Solo se convierten los datos de stdin durante las operaciones de lectura, y la conversión se realiza del CCSID del trabajo a file.encoding.
2	Solo se convierten los datos de stdout durante las operaciones de escritura, y la conversión se realiza de file.encoding al CCSID del trabajo.
3	Se realizan las conversiones de stdin y stdout.
4	Solo se convierten los datos de stderr durante las operaciones de escritura, y la conversión se realiza de file.encoding al CCSID del trabajo.
5	Se realizan las conversiones de stdin y stderr.
6	Se realizan las conversiones de stdout y stderr.
7	Se realizan todas las conversiones de stdio.

Tabla 2. Valores del sistema para **os400.child.stdio.convert**

Valor	Descripción
N (predeterminado)	No se realiza ninguna conversión de stdio durante las operaciones de lectura o escritura.
Y	Se convierten todos los datos de stdio durante las operaciones de lectura o escritura, y la conversión se realiza del valor de file.encoding al CCSID del trabajo o viceversa.
1	Solo se convierten los datos de stdin durante las operaciones de lectura, y la conversión se realiza del CCSID del trabajo a file.encoding.
2	Solo se convierten los datos de stdout durante las operaciones de escritura, y la conversión se realiza de file.encoding al CCSID del trabajo.
3	Se realizan las conversiones de stdin y stdout.
4	Solo se convierten los datos de stderr durante las operaciones de escritura, y la conversión se realiza de file.encoding al CCSID del trabajo.
5	Se realizan las conversiones de stdin y stderr.
6	Se realizan las conversiones de stdout y stderr.
7	Se realizan todas las conversiones de stdio.

Valores de las propiedades os400.stdin, os400.stdout y os400.stderr del sistema:

La tabla siguiente muestra los valores del sistema para las propiedades de sistema os400.stdin, os400.stdout y os400.stderr.

Valor	Nombre de ejemplo	Descripción	Ejemplo
File	UnNombreArchivo	UnNombreArchivo es una vía absoluta o relativa de acceso al directorio actual.	archivo:/QIBM/UserData/ Java400/Output.file
Port	NombreSistPral	Dirección del puerto	port:misistpral:2000
Port	DirecciónTCP	Dirección del puerto	port:1.1.11.111:2000

Valores de la propiedad os400.verify.checks.disable del sistema:

El valor de la propiedad os400.verify.checks.disable del sistema es una serie que representa la suma de uno o más valores numéricos de la siguiente lista.

Valor	Descripción
1	**Eludir comprobaciones de acceso para clases locales**: Indica que desea que la máquina virtual Java eluda las comprobaciones de acceso en los campos y métodos privados y protegidos de las clases que se cargan a partir del sistema de archivos local. Resulta útil cuando se transfieren aplicaciones que contienen clases interiores que hagan referencia a campos y métodos protegidos y privados de las clases que las engloban.
2	**Suprimir NoClassDefFoundError durante carga temprana:** Indicates que desea que la máquina virtual Java ignore los NoClassDefFoundErrors, que se producen durante las comprobaciones de verificación tempranas de la conversión de tipos y el acceso a campos o métodos.
4	**Eludir la comprobación de LocalVariableTable:** indica que, si se encuentra un error en la tabla LocalVariableTable de una clase, la clase funcionará como si LocalVariableTable no existiese. Por lo demás, los errores de LocaleVariableTable dan como resultado ClassFormatError.
7	Valor utilizado en el momento de la ejecución.

Se puede indicar el valor en formato decimal, hexadecimal u octal. Se hará caso omiso de los valores menores que cero. Por ejemplo, para seleccionar los dos primeros valores de la lista, utilice esta sintaxis de mandatos:

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

Internacionalización

Puede personalizar sus programas Java para una zona específica del mundo creando programas Java internacionalizados. Utilizando el huso horario, los entornos locales y la codificación de caracteres, puede asegurarse de que el programa Java refleja la hora, lugar e idioma correctos.

Internacionalización de Sun Microsystems, Inc.

Globalización de i5/OS

Configuración del huso horario

Cuando tenga programas Java que sean sensibles al huso horario, deberá configurar el huso horario en el sistema para que los programas Java utilicen la hora correcta.

La manera más sencilla de configurar el huso horario consiste en establecer que el valor QTIMZON del sistema sea igual a uno de los objetos *TIMZON proporcionados por i5/OS. Para determinar la hora local correctamente, la máquina virtual Java (JVM) exige que se establezca debidamente el valor QUTCOFFSET del sistema y la propiedad Java user.timezone del sistema. Estas dos cosas se consiguen automáticamente al establecer el valor QTIMZON del sistema. Los objetos TIMZON contienen un nombre largo alternativo que especifica el valor de la propiedad Java user.timezone que se empleará, por lo que debe seleccionar el valor de QTIMZON que contiene el nombre alternativo pertinente. Por ejemplo, el objeto TIMZON QN0600CST2 contiene el nombre alternativo America/Chicago y proporciona el soporte de hora correcto para el huso horario central de Estados Unidos.

Nota: El valor de la propiedad user.timezone del sistema proporcionado por el valor QTIMZON del sistema se puede alterar temporalmente de dos formas:

- Especificando el valor de user.timezone explícitamente en la línea de mandatos o en el archivo SystemDefault.properties
- Utilizando LOCALE para establecer el valor de user.timezone

El método LOCALE se utilizaba antes de que el soporte del valor QTIMZON del sistema estuviera disponible en i5/OS, pero ha dejado de estar recomendado.

Valor del sistema de i5/OS: QTIMZON

Mandato CL Trabajar con desc de huso horario (WRKTIMZON)

Tema Trabajar con entornos locales en el soporte de idioma nacional (NLS)

Información de consulta de Javadoc de huso horario, de Sun Microsystems, Inc.

Codificaciones de caracteres de Java

Los programas Java pueden convertir datos en distintos formatos, permitiendo a las aplicaciones transferir y utilizar información de muchas clases de juegos de caracteres internacionales.

Internamente, la máquina virtual Java (JVM) siempre trabaja con datos en formato Unicode. Sin embargo, todos los datos transferidos a la JVM o desde ella tienen un formato que se corresponde con la propiedad file.encoding. Los datos que entran en la JVM para lectura se convierten de file.encoding a Unicode, y los datos que salen de la JVM se convierten de Unicode a file.encoding.

Los archivos de datos de los programas Java se almacenan en el sistema de archivos integrado. Los archivos del sistema de archivos integrado están marcados con un identificador de juego de caracteres (CCSID) que identifica la codificación de caracteres de los datos que hay en el archivo.

Cuando un programa Java lee datos, estos deberían tener la codificación de caracteres que se corresponde con el valor de la propiedad file.encoding. Cuando un programa Java escribe datos en un archivo, los datos se escriben en una codificación de caracteres que se corresponden con el valor de la propiedad file.encoding. Esto es igualmente aplicable a los archivos de código fuente Java (archivos .java) procesados por el mandato javac y a los datos que se envían y reciben a través de los sockets del protocolo de control de transmisión/protocolo Internet (TCP/IP) utilizando el paquete java.net.

Los datos que se leen o escriben en System.in, System.out y System.err se manejan de distinta forma que los datos que se leen o escriben en otros orígenes cuando están asignados a la entrada estándar (stdin), a la salida estándar (stdout) y a la salida de error estándar (stderr). Puesto que, normalmente, la entrada estándar, la salida estándar y la salida de errores estándar están conectadas a dispositivos EBCDIC en el servidor System i, la JVM realiza en los datos una conversión para que pasen de tener la codificación de caracteres normal file.encoding a tener un CCSID que coincida con el CCSID del trabajo de System i. Cuando System.in, System.out o System.err se redirigen a un archivo o a un socket y no se dirigen a la entrada estándar, a la salida estándar ni a la salida de error estándar, esta conversión de datos adicional no se realiza y los datos siguen teniendo la codificación de caracteres correspondiente a file.encoding.

Cuando es necesario leer o escribir datos en un programa Java utilizando una codificación de caracteres distinta de file.encoding, el programa puede emplear las clases Java de E/S java.io.InputStreamReader, java.io.FileReader, java.io.OutputStreamReader y java.io.FileWriter. Estas clases Java permiten especificar un valor de file.encoding que tiene prioridad sobre la propiedad file.encoding predeterminada que utiliza en ese momento la JVM.

Los datos destinados a o procedentes de la base de datos DB2, por medio de las API de JDBC, se convierten a o desde el CCSID de la base de datos de System i.

Los datos transferidos a o desde otros programas mediante la interfaz Java nativa (JNI) no se convierten.

Globalización

Internacionalización de Sun Microsystems, Inc.

Valores de file.encoding y CCSID de System i5:

Esta tabla muestra la relación que existe entre los valores posibles de file.encoding y el identificador de juego de caracteres (CCSID) de System i5 que más se aproxima.

Para obtener más información sobre el soporte de file.encoding, vea Codificaciones soportadas de Sun Microsystems, Inc. Enlace fuera de Information Center

file.encoding	CCSID	Descripción
ASCII	367	Código estándar americano para intercambio de información
Big5	950	BIG-5 para chino tradicional ASCII de 8 bits
Big5_HKSCS	950	Big5_HKSCS
Big5_Solaris	950	Big5 con siete correlaciones de caracteres ideográficos Hanzi adicionales para el entorno local Solaris zh_TW.BIG5
CNS11643	964	Juego de caracteres nacionales para chino tradicional
Cp037	037	EBCDIC de IBM para EE.UU., Canadá, los Países Bajos, ...
Cp273	273	EBCDIC de IBM para Alemania y Austria
Cp277	277	EBCDIC de IBM para Dinamarca y Noruega
Cp278	278	EBCDIC de IBM para Finlandia y Suecia
Cp280	280	EBCDIC de IBM para Italia
Cp284	284	EBCDIC de IBM para España y América Latina
Cp285	285	EBCDIC de IBM para el Reino Unido
Cp297	297	EBCDIC de IBM para Francia
Cp420	420	EBCDIC de IBM para los países árabes
Cp424	424	EBCDIC de IBM para Israel
Cp437	437	PC de EE. UU. ASCII de 8 bits
Cp500	500	EBCDIC de IBM internacional
Cp737	737	Griego MS-DOS ASCII de 8 bits
Cp775	775	Báltico MS-DOS ASCII de 8 bits
Cp838	838	EBCDIC de IBM para Tailandia
Cp850	850	Multinacional Latin-1 ASCII de 8 bits
Cp852	852	Latin-2 ASCII de 8 bits
Cp855	855	Cirílico ASCII de 8 bits

file.encoding	CCSID	Descripción
Cp856	0	Hebreo ASCII de 8 bits
Cp857	857	Latin-5 ASCII de 8 bits
Cp860	860	ASCII de 8 bits para Portugal
Cp861	861	ASCII de 8 bits para Islandia
Cp862	862	Hebreo ASCII de 8 bits
Cp863	863	ASCII de 8 bits para Canadá
Cp864	864	Árabe ASCII de 8 bits
Cp865	865	ASCII de 8 bits para Dinamarca y Noruega
Cp866	866	Cirílico ASCII de 8 bits
Cp868	868	Urdu ASCII de 8 bits
Cp869	869	Griego ASCII de 8 bits
Cp870	870	EBCDIC de IBM en Latin-2
Cp871	871	EBCDIC de IBM para Islandia
Cp874	874	ASCII de 8 bits para Tailandia
Cp875	875	EBCDIC de IBM para Grecia
Cp918	918	EBCDIC de IBM en Urdu
Cp921	921	Báltico ASCII de 8 bits
Cp922	922	ASCII de 8 bits para Estonia
Cp930	930	EBCDIC de IBM en japonés katakana ampliado
Cp933	933	EBCDIC de IBM para Corea
Cp935	935	EBCDIC de IBM en chino simplificado
Cp937	937	EBCDIC de IBM en chino tradicional
Cp939	939	EBCDIC de IBM en japonés latino ampliado
Cp942	942	Japonés ASCII de 8 bits
Cp942C	942	Variante de Cp942
Cp943	943	Datos mixtos de PC japonés para el entorno abierto
Cp943C	943	Datos mixtos de PC japonés para el entorno abierto
Cp948	948	Chino tradicional IBM ASCII de 8 bits
Cp949	944	KSC5601 para coreano ASCII de 8 bits
Cp949C	949	variante de Cp949
Cp950	950	BIG-5 para chino tradicional ASCII de 8 bits
Cp964	964	Chino tradicional EUC
Cp970	970	Coreano EUC
Cp1006	1006	Urdu de 8 bits ISO
Cp1025	1025	EBCDIC de IBM para Cirílico
Cp1026	1026	EBCDIC de IBM para Turquía
Cp1046	1046	Árabe ASCII de 8 bits
Cp1097	1097	EBCDIC de IBM en Farsi
Cp1098	1098	Persa ASCII de 8 bits
Cp1112	1112	EBCDIC de IBM en báltico
Cp1122	1122	EBCDIC de IBM para Estonia

file.encoding	CCSID	Descripción
Cp1123	1123	EBCDIC de IBM para Ucrania
Cp1124	0	8 bits ISO para Ucrania
Cp1140	1140	Variante de Cp037 con carácter Euro
Cp1141	1141	Variante de Cp273 con carácter Euro
Cp1142	1142	Variante de Cp277 con carácter Euro
Cp1143	1143	Variante de Cp278 con carácter Euro
Cp1144	1144	Variante de Cp280 con carácter Euro
Cp1145	1145	Variante de Cp284 con carácter Euro
Cp1146	1146	Variante de Cp285 con carácter Euro
Cp1147	1147	Variante de Cp297 con carácter Euro
Cp1148	1148	Variante de Cp500 con carácter Euro
Cp1149	1149	Variante de Cp871 con carácter Euro
Cp1250	1250	Latin-2 MS-Win
Cp1251	1251	Cirílico MS-Win
Cp1252	1252	Latin-1 MS-Win
Cp1253	1253	Griego MS-Win
Cp1254	1254	Turco MS-Win
Cp1255	1255	Hebreo MS-Win
Cp1256	1256	Árabe MS-Win
Cp1257	1257	Báltico MS-Win
Cp1258	1251	Ruso MS-Win
Cp1381	1381	GB para chino simplificado ASCII de 8 bits
Cp1383	1383	Chino simplificado EUC
Cp33722	33722	Japonés EUC
EUC_CN	1383	EUC para chino simplificado
EUC_JP	5050	EUC para japonés
EUC_JP_LINUX	0	JISX 0201, 0208 , codificación EUC Japonés
EUC_KR	970	EUC para coreano
EUC_TW	964	EUC para chino tradicional
GB2312	1381	GB para chino simplificado ASCII de 8 bits
GB18030	1392	Chino simplificado, estándar PRC
GBK	1386	Nuevo ASCII 9 de 8 bits para chino simplificado
ISCII91	806	Codificación ISCII91 de scripts Indic
ISO2022CN	965	ISO 2022 CN, Chino (solo conversión a Unicode)
ISO2022_CN_CNS	965	CNS11643 en formato ISO 2022 CN, Chino tradicional (solo conversión de Unicode)
ISO2022_CN_GB	1383	GB2312 en formato ISO 2022 CN, Chino simplificado (solo conversión de Unicode)
ISO2022CN_CNS	965	ASCII de 7 bits para chino tradicional
ISO2022CN_GB	1383	ASCII de 7 bits para chino simplificado
ISO2022JP	5054	ASCII de 7 bits para japonés

file.encoding	CCSID	Descripción
ISO2022KR	25546	ASCII de 7 bits para coreano
ISO8859_1	819	ISO 8859-1 Alfabeto Latino Núm. 1
ISO8859_2	912	ISO 8859-2 ISO Latin-2
ISO8859_3	0	ISO 8859-3 ISO Latin-3
ISO8859_4	914	ISO 8859-4 ISO Latin-4
ISO8859_5	915	ISO 8859-5 ISO Latin-5
ISO8859_6	1089	ISO 8859-6 ISO Latin-6 (árabe)
ISO8859_7	813	ISO 8859-7 ISO Latin-7 (griego/latino)
ISO8859_8	916	ISO 8859-8 ISO Latin-8 (hebreo)
ISO8859_9	920	ISO 8859-9 ISO Latin-9 (ECMA-128, Turquía)
ISO8859_13	0	Alfabeto Latino Núm. 7
ISO8859_15	923	ISO8859_15
ISO8859_15_FDIS	923	ISO 8859-15, Alfabeto Latino Núm. 9
ISO-8859-15	923	ISO 8859-15, Alfabeto Latino Núm. 9
JIS0201	897	JIS X0201
JIS0208	5052	JIS X0208
JIS0212	0	JIS X0212
JISAutoDetect	0	Detecta y convierte de Shift-JIS, EUC-JP, ISO 2022 JP (solo conversión a Unicode)
Johab	0	Codificación (completa) Hangul para coreano.
K018_R	878	Cirílico
KSC5601	949	Coreano ASCII de 8 bits
MacÁrabe	1256	Árabe Macintosh
MacCentroEuropa	1282	Latin-2 Macintosh
MacCroata	1284	Croata Macintosh
MacCirílico	1283	Cirílico Macintosh
MacDingbat	0	Dingbat Macintosh
MacGriego	1280	Griego Macintosh
MacHebreo	1255	Hebreo Macintosh
MacIslandia	1286	Islandia Macintosh
MacRoman	0	Macintosh Roman
MacRumanía	1285	Rumanía Macintosh
MacSímbolo	0	Símbolo Macintosh
MacTailandés	0	Tailandés Macintosh
MacTurco	1281	Turco Macintosh
MacUcrania	1283	Ucrania Macintosh
MS874	874	MS-Win para Tailandia
MS932	943	JaponésWindows
MS936	936	Chino simplificadoWindows
MS949	949	CoreanoWindows
MS950	950	Chino tradicionalWindows

file.encoding	CCSID	Descripción
MS950_HKSCS	NA	Chino tradicional Windows con extensiones de Hong Kong, Región Administrativa Especial de China
SJIS	932	Japonés ASCII de 8 bits
TIS620	874	TIS 620
ASCII-EE.UU.	367	Código estándar americano para intercambio de información
UTF8	1208	UTF-8 (CCSID 1208 de IBM, que todavía no está disponible en la plataforma System i5)
UTF-16	1200	Formato de transformación UCS de dieciséis bits, orden de bytes identificado por una marca de orden de bytes opcional
UTF-16BE	1200	Formato de transformación Unicode de dieciséis bits, orden de bytes de gran memoria numérica
UTF-16LE	1200	Formato de transformación Unicode de dieciséis bits, orden de bytes de pequeña memoria numérica
UTF-8	1208	Formato de transformación UCS de ocho bits
Unicode	13488	UNICODE, UCS-2
UnicodeBig	13488	Idéntico a Unicode
UnicodeBigUnmarked		Unicode sin marca de orden de bytes
UnicodeLittle		Unicode con orden de bytes little-endian
UnicodeLittleUnmarked		UnicodeLittle sin marca de orden de bytes

En Valores de file.encoding por omisión hallará los valores predeterminados.

Valores de file.encoding por omisión:

Esta tabla muestra cómo se establece el valor de file.encoding tomando como base el identificador de juego de caracteres (CCSID) de System i cuando se inicia la máquina virtual Java.

CCSID de System i	File.encoding por omisión	Descripción
37	ISO8859_1	Inglés para EE. UU., Canadá, Nueva Zelanda y Australia; portugués para Portugal y Brasil; y holandés para los Países Bajos
256	ISO8859_1	Internacional nº 1
273	ISO8859_1	Alemán/Alemania, alemán/Austria
277	ISO8859_1	Danés/Dinamarca, noruego/Noruega, noruego/Noruega, NY
278	ISO8859_1	Finlandés/Finlandia
280	ISO8859_1	Italiano/Italia
284	ISO8859_1	Catalán/España, español/España
285	ISO8859_1	Inglés/Gran Bretaña, inglés/Irlanda
290	Cp943C	Parte SBCS del CCSID mixto EBCDIC japonés (CCSID 5026)
297	ISO8859_1	Francés/Francia
420	Cp1046	Árabe/Egipto
423	ISO8859_7	Grecia

CCSID de System i	File.encoding por omisión	Descripción
424	ISO8859_8	Hebreo/Israel
500	ISO8859_1	Alemán/Suiza, francés/Bélgica, francés/Canadá, francés/Suiza
833	Cp970	Parte SBCS del CCSID mixto EBCDIC coreano (CCSID 933)
836	Cp1383	Parte SBCS del CCSID mixto EBCDIC para chino simplificado (CCSID 935)
838	TIS620	Tailandés
870	ISO8859_2	Checo/República Checa, croata/Croacia, húngaro/Hungría, polaco/Polonia
871	ISO8859_1	Islandés/Islandia
875	ISO8859_7	Griego/Grecia
880	ISO8859_5	Bulgaria (ISO 8859_5)
905	ISO8859_9	Ampliado de Turquía
918	Cp868	Urdu
930	Cp943C	Japonés EBCDIC mixto (similar al CCSID 5026)
933	Cp970	Coreano/Corea
935	Cp1383	Chino simplificado
937	Cp950	Chino tradicional
939	Cp943C	Japonés EBCDIC mixto (similar al CCSID 5035)
1025	ISO8859_5	Bielorruso/Bielorrusia, búlgaro/Bulgaria, macedonio/Macedonia, ruso/Rusia
1026	ISO8859_9	Turco/Turquía
1027	Cp943C	Parte SBCS del CCSID mixto EBCDIC japonés (CCSID 5035)
1097	Cp1098	Persa
1112	Cp921	Lituano/Lituania, letón/Letonia, báltico
1388	GBK	CCSID mixto EBCDIC para chino simplificado (se incluye GBK)
5026	Cp943C	CCSID mixto EBCDIC japonés (katakana ampliado)
5035	Cp943C	CCSID mixto EBCDIC japonés (latino ampliado)
8612	Cp1046	Árabe (grafías básicas únicamente) (o ASCII 420 y 8859_6)
9030	Cp874	Tailandés (SBCS ampliado de sistema principal)
13124	GBK	Parte SBCS del CCSID mixto EBCDIC para chino simplificado (se incluye GBK)
28709	Cp948	Parte SBCS del CCSID mixto EBCDIC para chino tradicional (CCSID 937)

Ejemplos: crear un programa Java internacionalizado

Si necesita personalizar un programa Java(TM) para una región concreta del mundo, puede crear un programa Java internacionalizado con los Entornos nacionales Java.

Entornos locales Java.

Crear un programa Java internacionalizado implica diversas tareas:

1. Aísle los datos y el código sensibles al entorno nacional. Por ejemplo, las series, las fechas y los números del programa.
2. Establezca u obtenga el entorno nacional con la clase Locale.
3. Formatee las fechas y los números con el fin de especificar un entorno nacional si no se quiere utilizar el entorno nacional por omisión.
4. Cree paquetes de recursos para manejar las series y demás datos sensibles al entorno nacional.

Revise los siguientes ejemplos, que ofrecen maneras de ayudarle a completar las tareas necesarias para crear un programa Java internacionalizado:

- "Ejemplo: internacionalización de las fechas con la clase java.util.DateFormat" en la página 449
- "Ejemplo: internacionalización de las presentaciones numéricas con la clase java.util.NumberFormat" en la página 449
- "Ejemplo: internacionalización de los datos específicos de entorno nacional con la clase java.util.ResourceBundle" en la página 450

Globalización

Internacionalización de Sun Microsystems, Inc.

Compatibilidad entre releases

En V6R1, el proceso directo ha dejado de estar soportado. Este cambio afecta a los parámetros OPTIMIZE y ENBPFRCOL.

En V6R1, el proceso directo ha dejado de estar soportado. Esto quiere decir que el nivel de optimización de los programas Java se ignora y que se emplea OPTIMIZE(*INTERPRET) cuando se crea un programa Java en V6R1 o releases posteriores.

Al igual que en los releases anteriores, el programa Java contiene una versión preverificada de uno o más archivos de clase Java. Sin embargo, en V6R1, el programa Java no contiene instrucciones de máquina. En tiempo de ejecución, el programa Java se interpreta a partir del bytecode o bien se ejecuta con el compilador Just-In-Time (JIT).

En el caso de los releases destinos anteriores a V6R1, el parámetro OPTIMIZE especifica el nivel de optimización del programa Java. Al crear un programa Java para un release destino anterior a V6R1, el valor del parámetro OPTIMIZE se encapsula dentro del programa Java, pero no se generan instrucciones de máquina. El valor encapsulado del parámetro OPTIMIZE se emplea durante la conversión del programa Java en el release destino del sistema operativo.

En el caso de los programas Java creados para el release destino V6R1 y posterior, el valor del parámetro ENBPFRCOL se ignora, y se utiliza ENDPFRCOL(*NONE). Para habilitar la recogida de rendimiento en las aplicaciones Java de los releases V6R1 y posteriores, debe utilizar la propiedad Java os400.enbpfrcol del sistema. Hallará más información en el tema "Lista de propiedades Java del sistema" en la página 16.

En el caso de un release destino anterior a V6R1, puede especificar valores distintos de *NONE, pero esos valores distintos de *NONE no entrarán en vigor mientras el programa Java no se vuelva a convertir en el release destino del sistema operativo.

Conceptos relacionados

"Novedades de la V6R1" en la página 1
Aquí encontrará la información nueva o la que ha cambiado notablemente en el temario de IBM Developer Kit para Java.

"Consideraciones sobre el rendimiento de Java" en la página 414
La total comprensión de las siguientes consideraciones puede ayudarle a mejorar el rendimiento de sus aplicaciones Java.

Acceso a base de datos con IBM Developer Kit para Java

Con IBM Developer Kit para Java, los programas Java pueden acceder a los archivos de base de datos de varias maneras.

Acceder a la base de datos de System i5 con el controlador JDBC de IBM Developer Kit para Java

El controlador JDBC de IBM Developer Kit para Java, que también se conoce como controlador nativo, proporciona acceso programático a los archivos de base de datos del System i5. Si se emplea la API de Java Database Connectivity (JDBC), las aplicaciones escritas en el lenguaje Java pueden acceder a las funciones de base de datos de JDBC con lenguaje de consulta estructurada (SQL) incorporado, ejecutar sentencias SQL, recuperar resultados y propagar los cambios de nuevo a la base de datos. La API de JDBC también se puede utilizar para interaccionar con múltiples orígenes de datos en un entorno distribuido heterogéneo.

La interfaz de línea de mandatos (CLI) de SQL99, en la que se basa la API de JDBC, es la base de ODBC. JDBC proporciona una correlación natural y fácil de usar desde el lenguaje de programación Java hasta las abstracciones y conceptos definidos en el estándar SQL.

Documentación de JDBC de Sun Microsystems, Inc.

Preguntas más frecuentes (P+F) sobre el controlador JDBC nativo

Especificación de la API de JDBC 4.0

Iniciación a JDBC

El controlador Java Database Connectivity (JDBC) que viene con IBM Developer Kit para Java también se llama controlador JDBC de IBM Developer Kit para Java. Este controlador también se conoce comúnmente como controlador JDBC nativo.

Para seleccionar el controlador JDBC que se ajuste a sus necesidades, tenga en cuenta las siguientes sugerencias:

- Los programas que se ejecutan directamente en un servidor en el que reside la base de datos deben utilizar el controlador JDBC nativo a efectos de rendimiento. Esto incluye la mayoría de las soluciones de servlets y JavaServer Pages (JSP) y las aplicaciones escritas para ejecutarse localmente en un sistema.
- Los programas que se deben conectar a un System i5 remoto emplean clases JDBC de IBM Toolbox para Java. El controlador JDBC de IBM Toolbox para Java es una sólida implementación de JDBC y se proporciona como parte de IBM Toolbox para Java. Por ser Java puro, el controlador JDBC de IBM Toolbox para Java es fácil de configurar en los clientes y requiere poca configuración del servidor.
- Los programas que se ejecutan en un System i5 y tienen que conectarse a una base de datos remoto no de System i5 emplean el controlador JDBC nativo y configuran una conexión de tipo Distributed Relational Database Architecture (DRDA) con dicho servidor remoto.

Tipos de controladores JDBC:

En este tema se definen los tipos de controladores JDBC (Java Database Connectivity). Los tipos de controladores se definen en categorías según la tecnología utilizada para conectarse a la base de datos. Los proveedores de controladores JDBC utilizan estos tipos para describir cómo operan sus productos. Algunos tipos de controladores JDBC son más adecuados que otros para algunas aplicaciones.

Tipo 1

Los controladores de tipo 1 son controladores "puente". Utilizan otra tecnología, como por ejemplo, ODBC (Open Database Connectivity), para comunicarse con la base de datos. Esto representa una ventaja, ya que existen controladores ODBC para muchas plataformas RDBMS (sistemas de gestión de bases de datos relacionales). Se emplea la interfaz Java nativa (JNI) para llamar a funciones ODBC desde el controlador JDBC.

Un controlador de tipo 1 debe tener el controlador puente instalado y configurado para poder utilizar JDBC con él. Esto puede representar un grave inconveniente para una aplicación de producción. Los controladores de tipo 1 no pueden utilizarse en un applet, ya que los applets no pueden cargar código nativo.

Tipo 2

Los controladores de tipo 2 utilizan una API nativa para comunicarse con un sistema de base de datos. Se utilizan métodos Java nativos para invocar las funciones de API que realizan operaciones de base de datos. Los controladores de tipo 2 son generalmente más rápidos que los controladores de tipo 1.

Los controladores de tipo 2 necesitan tener instalado y configurado código binario nativo para funcionar. Un controlador de tipo 2 siempre utiliza JNI. Los controladores de tipo 2 no pueden utilizarse en un applet, ya que los applets no pueden cargar código nativo. Un controlador JDBC de tipo 2 puede requerir la instalación de algún software de red DBMS (sistema de gestión de bases de datos).

El controlador JDBC de Developer Kit para Java es de tipo 2.

Tipo 3

Estos controladores utilizan un protocolo de red y middleware para comunicarse con un servidor. A continuación, el servidor convierte el protocolo a llamadas de función DBMS específicas de DBMS.

Los controladores de tipo 3 son la solución JDBC más flexible, ya que no requieren ningún código binario nativo en el cliente. Un controlador de tipo 3 no necesita ninguna instalación de cliente.

Tipo 4

El controlador de tipo 4 utiliza Java para implementar un protocolo de red de proveedores de DBMS. Puesto que los protocolos son generalmente de propiedad, los proveedores DBMS son generalmente las únicas empresas que suministran un controlador JDBC de tipo 4.

Los controladores de tipo 4 son todos ellos controladores Java. Esto significa que no existe ninguna instalación ni configuración de cliente. Sin embargo, un controlador de tipo 4 puede no ser adecuado para algunas aplicaciones si el protocolo subyacente no maneja adecuadamente cuestiones tales como la seguridad y la conectividad de red.

El controlador JDBC de IBM Toolbox para Java es de tipo 4, lo cual indica que la API es un controlador de protocolo red Java puro.

Requisitos de JDBC:

En este tema se indica qué requisitos hacen falta para acceder a JDBC núcleo y a la API de transacciones Java (JTA).

Antes de escribir y desplegar las aplicaciones JDBC, puede que sea necesario incluir archivos jar específicos en la vía de acceso de clase.

JDBC núcleo

En el caso del acceso JDBC (Java Database Connectivity) núcleo a la base de datos local, no hay ningún requisito. Todo el soporte se ha incorporado, preinstalado y configurado.

Compatibilidad de JDBC

El controlador JDBC nativo es compatible con todas las especificaciones JDBC relevantes. El nivel de compatibilidad del controlador JDBC no depende del release de i5/OS, sino del release de JDK que utilice. A continuación se ofrece una lista del nivel de compatibilidad del controlador JDBC nativo para las diversas versiones de JDK:

Versión	Nivel de compatibilidad del controlador JDBC
JDK 1.4 y versiones posteriores	Estas versiones del JDK están en conformidad con con JDBC 3.0.
J2SE 6 y versiones ulteriores	JDBC 4.0

Guía de aprendizaje de JDBC:

Esta guía sirve para aprender a escribir un programa JDBC (Java Database Connectivity) y a ejecutarlo en un System i5 con el controlador JDBC nativo. Está diseñada para mostrarle los pasos básicos necesarios para que el programa ejecute JDBC.

El ejemplo crea una tabla y la puebla con algunos datos. El programa procesa una consulta para obtener esos datos de la base de datos y visualizarlos en la pantalla.

Ejecutar el programa de ejemplo

Para ejecutar el programa de ejemplo, lleve a cabo los siguientes pasos:

1. Copie el programa en la estación de trabajo.
   a. Copie el ejemplo y péguelo en un archivo de la estación de trabajo.
   b. Guarde el archivo con el mismo nombre que la clase pública suministrada y con la extensión .java. En este caso, el nombre del archivo debe ser BasicJDBC.java en la estación de trabajo local.
2. Transfiera el archivo de la estación de trabajo al servidor. En un indicador de mandatos, entre los siguientes mandatos:

```
ftp <nombre de servidor>
<Entre el ID de usuario>
<Entre la contraseña>
cd /home/cujo
put BasicJDBC.java
quit
```

   Para que estos mandatos funcionen, debe tener un directorio en el que colocar el archivo. En el ejemplo, la ubicación es /home/cujo, pero puede utilizar la ubicación que desee.

   Nota: Nota: es posible que los mandatos FTP mencionados anteriormente sean diferentes en función de la configuración del servidor, pero deben ser parecidos. La forma de transferir el archivo al

servidor no tiene importancia, siempre y cuando lo transfiera al sistema de archivos integrado. Algunas herramientas, como VisualAge para Java, pueden automatizar totalmente este proveco.

3. Asegúrese de establecer la vía de acceso de clases en el directorio en el que ha colocado el archivo, para que los mandatos Java encuentren el archivo al ejecutarlas. Desde una línea de mandatos CL, puede utilizar el mandato WRKENVVAR para ver las variables de entorno establecidas para el perfil de usuario.
   - Si observa una variable de entorno denominada CLASSPATH, debe asegurarse de que la ubicación en la que ha colocado el archivo .java se encuentra en la serie de directorios indicados allí, o añádala si la ubicación no se ha especificado.
   - Si no existe ninguna variable de entorno CLASSPATH, debe añadir una. Esta operación puede realizarse con el siguiente mandato:

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

   Nota: Para compilar código Java desde el mandato CL, debe incluir el archivo tools.jar. Este archivo JAR incluye el mandato javac.
4. Compile el archivo Java en un archivo de clase. Entre el siguiente mandato desde la línea de mandatos CL:

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

   También puede compilar el archivo Java desde QSH:

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

   QSH se asegura automáticamente de que el archivo tools.jar pueda encontrarse. En consecuencia, no es necesario añadirlo a la vía de acceso de clases. El directorio actual también se encuentra en la vía de acceso de clases. Al emitir el mandato cambiar directorio (cd), también se encuentra el archivo BasicJDBC.java.

   Nota: También puede compilar el archivo en la estación de trabajo y utilizar FTP para enviar el archivo de clase al servidor en modalidad binaria. Este es un ejemplo de la capacidad de Java para ejecutarse en cualquier plataforma.

   Ejecute el programa Java mediante el siguiente mandato desde la línea de mandatos CL o desde QSH:

   ```
   java BasicJDBC
   ```

   La salida es la siguiente:

   ```
   ----------------------
   | 1 | Frank Johnson  |
   |                    |
   | 2 | Neil Schwartz  |
   |                    |
   | 3 | Ben Rodman     |
   |                    |
   | 4 | Dan Gloore     |
   ----------------------
   Se devuelven 4 filas.
   Salida completada.
   Programa Java completado.
   ```

   Sitio Web del controlador JDBC de IBM Toolbox para Java

   Página JDBC de Sun Microsystems, Inc.

Ejemplo: JDBC:

Este es un ejemplo de cómo utilizar el programa BasicJDBC. Este programa emplea un controlador JDBC nativo para IBM Developer Kit para Java para construir una tabla simple y procesar una consulta que visualiza los datos de esa tabla.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de BasicJDBC. Este programa utiliza el controlador JDBC nativo de
// Developer Kit para Java para crear una tabla simple y procesar una consulta
// que visualice los datos de dicha tabla.
//
// Sintaxis de mandato:
//    BasicJDBC
//
//////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo del controlador JDBC de IBM Developer Kit para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo solo con fines ilustrativos.
// Estos ejemplos no se han probado exhaustivamente bajo todas las
// condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas que contiene este ejemplo se suministran "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se rechazan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////

// Incluir las clases Java que deban utilizarse. En esta aplicación
// se utilizan muchas clases del paquete java.sql y también se utiliza
// la clase java.util.Properties como parte del proceso de obtención
// de una conexión con la base de datos.
import java.sql.*;
import java.util.Properties;

// Crear una clase pública para encapsular el programa.
public class BasicJDBC {

    // La conexión es una variable privada del objeto.
    private Connection connection = null;

    // Cualquier clase que deba ser un "punto de entrada" para ejecutar
    // un programa debe tener un método main. El método main
    // es donde empieza el proceso cuando se llama al programa.
    public static void main(java.lang.String[] args) {

        // Crear un objeto de tipo BasicJDBC. Esto
        // es fundamental en la programación orientada a objetos. Una vez
        // creado un objeto, llamar a diversos métodos de
        // ese objeto para realizar el trabajo.
        // En este caso, al llamar al constructor del objeto
        // se crea una conexión de base de datos que los otros
        // métodos utilizan para realizar el trabajo en la base de datos.
        BasicJDBC test = new BasicJDBC();

        // Llamar al método rebuildTable. Este método asegura que
        // la tabla utilizada en este programa existe y tiene el aspecto
```

```
    // correcto. El valor de retorno es un booleano para indicar
    // si la reconstrucción de la tabla se ha completado
    // satisfactoriamente. Si no es así, visualizar un mensaje
    // y salir del programa.
    if (!test.rebuildTable()) {
        System.out.println("Se produjo una anomalía al configurar " +
                           " para ejecutar la prueba.");
        System.out.println("La prueba no continuará.");
        System.exit(0);
    }

    // A continuación, se llama al método para ejecutar la consulta. Este método
    // procesa una sentencia SQL select con respecto a la tabla que
    // se creó en el método rebuildTable. La salida de
    // esa consulta va a la salida estándar de visualización.
    test.runQuery();

    // Por último, se llama al método cleanup. Este método
    // garantiza que la conexión de base de datos en la que el objeto
    // ha estado a la espera se ha cerrado.
    test.cleanup();
}


/**
Este es el constructor de la prueba básica JDBC. Crea una conexión
de base de datos que se almacena en una variable de instancia que se usará en
posteriores llamadas de método.
**/
public BasicJDBC() {

    // Una forma de crear una conexión de base de datos es pasar un URL
    // y un objeto java Properties al DriverManager. El siguiente
    // código construye un objeto Properties que contiene el ID de usuario y
    // la contraseña. Estos datos se emplean para establecer conexión
    // con la base de datos.
    Properties properties = new Properties ();
    properties.put("user", "cujo");
    properties.put("user", "newtiger");

    // Utilizar un bloque try/catch para capturar todas las excepciones que
    // puedan surgir del código siguiente.
    try {
        // DriverManager debe saber que existe un controlador JDBC disponible
        // para manejar una petición de conexión de usuario. La siguiente línea hace
        // que el controlador JDBC nativo se cargue y registre con DriverManager.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        // Crear el objeto Connection de base de datos que este programa utiliza
        // en las demás llamadas de método que se realicen. El código que sigue
        // especifica que debe establecerse una conexión con la base de datos local
        // y que dicha conexión debe ajustarse a las propiedades configuradas
        // anteriormente (es decir, debe utilizar el ID de usuario
        // y la contraseña especificados).
        connection = DriverManager.getConnection("jdbc:db2:*local", properties);

    } catch (Exception e) {
        // Si falla alguna de las líneas del bloque try/catch, el control pasa
        // a la siguiente línea de código. Una aplicación robusta intenta manejar
        // el problema o proporcionar más detalles. En este programa, se visualiza
        // el mensaje de error de la excepción, y la aplicación permite
        // el retorno del programa.
        System.out.println("Excepción capturada: " + e.getMessage());
    }
}
```

```
/**
Garantiza que la tabla qgpl.basicjdbc tiene el aspecto deseado al principio de
a prueba.

@returns boolean    Devuelve true si la tabla se ha reconstruido satisfactoriamente;
                    Devuelve false si se ha producido alguna anomalía.
**/
public boolean rebuildTable() {
    // Reiniciar todas las funciones de un bloque try/catch para que se realice
    // un intento de manejar los errores que puedan producirse dentro de este
    // método.
    try {

        // Se utilizan objetos Statement para procesar sentencias SQL en la
        // base de datos. El objeto Connection se utiliza para crear un
        // objeto Statement.
        Statement s = connection.createStatement();

        try {
            // Construir la tabla de prueba desde cero. Procesar una sentencia update
            // que intenta suprimir la tabla si existe actualmente.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // No realizar nada si se produjo una excepción. Presuponer
            // que el problema es que la tabla que se ha eliminado no
            // existe y que se puede crear a continuación.
        }

        // Utilizar el objeto sentencia para crear la tabla.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Utilizar el objeto sentencia para llenar la tabla con algunos
        // datos.
        s.executeUpdate("insert into qgpl.basicjdbc values(1, 'Frank Johnson')");
        s.executeUpdate("insert into qgpl.basicjdbc values(2, 'Neil Schwartz')");
        s.executeUpdate("insert into qgpl.basicjdbc values(3, 'Ben Rodman')");
        s.executeUpdate("insert into qgpl.basicjdbc values(4, 'Dan Gloore')");

        // Cerrar la sentencia SQL para indicar a la base de datos que ya no es
        // necesaria.
        s.close();

        // Si todo el método se ha procesado satisfactoriamente, devolver true. En este punto,
        // la tabla se ha creado o renovado correctamente.
        return true;

    } catch (SQLException sqle) {
        // Si ha fallado alguna de las sentencias SQL (que no sea la eliminación de la tabla
        // manejada en el bloque try/catch interno), el mensaje de error
        // se visualiza y se devuelve false al llamador, para indicar que la tabla
        // no puede completarse.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}



/**
Ejecuta una consulta a la tabla de muestra y los resultados se visualizan en
la salida estándar.
**/
public void runQuery() {
    // Reiniciar todas las funciones de un bloque try/catch para que se realice
    // un intento de manejar los errores que puedan producirse dentro de este
    // método.
    try {
```

```
            // Crear un objeto Statement.
            Statement s = connection.createStatement();

            // Usar el objeto Statement para ejecutar una consulta SQL. Las consultas devuelven
            // objetos ResultSet que se utilizan para observar los datos que la consulta
            // proporciona.
            ResultSet rs = s.executeQuery("select * from qgpl.basicjdbc");

            // Visualizar el principio de la 'tabla' e inicializar el contador del
            // número de filas devueltas.
            System.out.println("--------------------");
            int i = 0;

            // El método next de ResultSet se utiliza para procesar las filas de un
            // ResultSet. Hay que llamar al método next una vez antes de que
            // los primeros datos estén disponibles para verlos. Siempre que next devuelva
            // true, existe otra fila de datos que puede utilizarse.
            while (rs.next()) {

                // Obtener ambas columnas de la tabla para cada fila y escribir una fila en
                // nuestra tabla en pantalla con los datos. Luego, incrementar la cuenta
                // de filas que se han procesado.
                System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
                i++;
            }

            // Colocar un borde al final de la tabla y visualizar el número de filas
            // como salida.
            System.out.println("--------------------");
            System.out.println("Se han devuelto " + i + " filas.");
            System.out.println("Salida completada.");

        } catch (SQLException e) {
            // Visualizar más información acerca de las excepciones SQL
            // generadas como salida.
            System.out.println("Excepción SQLException: ");
            System.out.println("Mensaje:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Código proveedor:." + e.getErrorCode());
            e.printStackTrace();
        }
    }


    /**
    El siguiente método garantiza que se liberen los recursos JDBC que aún
    están asignados.
    **/
    public void cleanup() {
        try {
            if (connection != null)
                connection.close();
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Utilizar JNDI para los ejemplos de IBM Developer Kit para Java:

Los DataSources trabajan mano a mano con JNDI (Java Naming and Directory Interface). JNDI es una capa de abstracción Java para servicios de directorio, del mismo modo que JDBC (Java Database Connectivity) es una capa de abstracción para bases de datos.

JNDI se utiliza con mayor frecuencia con el protocolo ligero de acceso a directorio (LDAP), pero también puede utilizarse con los servicios de objetos CORBA (COS), con el registro de invocación de método remoto (RMI) Java o con el sistema de archivos subyacente. Esta utilización variada se lleva a cabo por medio de los diversos proveedores de servicios de directorio que convierten las peticiones JNDI comunes en peticiones de servicio de directorio específicas.

Nota: Tenga en cuenta que utilizar RMI puede resultar una tarea compleja. Antes de elegir RMI como solución, asegúrese de comprender las ramificaciones que puede conllevar. Un buen lugar para empezar a valorar RMI es en la invocación de método remoto (RMI) Java.

Los ejemplos de DataSource se han diseñado utilizando el proveedor de servicio del sistema de archivos JNDI. Si desea ejecutar los ejemplos suministrados, debe existir un proveedor de servicio JNDI.

Siga estas instrucciones para configurar el entorno para el proveedor de servicio del sistema de archivos:

1. Baje el soporte JNDI del sistema de archivos del sitio de JNDI de Sun Microsystems.
2. Transfiera (utilizando FTP u otro mecanismo) fscontext.jar y providerutil.jar al sistema y colóquelos en /QIBM/UserData/Java400/ext. Este es el directorio de extensiones, y los archivos JAR que coloque en él se encontrarán automáticamente cuando ejecute la aplicación (es decir, no es necesario especificarlos en la vía de acceso de clases).

Una vez que tiene soporte de un proveedor de servicio para JNDI, debe configurar la información de contexto para las aplicaciones. Esta acción puede realizarse colocando la información necesaria en un archivo SystemDefault.properties. Existen varios lugares del sistema en los que puede especificar propiedades por omisión, pero lo mejor es crear un archivo de texto denominado SystemDefault.properties en el directorio local (es decir, en /home/).

Para crear un archivo, utilice las siguientes líneas o añádalas al archivo existente:

```
# Valores de entorno necesarios para JNDI.
java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
java.naming.provider.url=file:/DataSources/jdbc
```

Estas líneas especifican que el proveedor de servicio del sistema de archivos maneja las peticiones JNDI y que /DataSources/jdbc es el directorio raíz para las tareas que utilizan JNDI. Puede cambiar esta ubicación, pero el directorio que especifique debe existir. La ubicación que especifique será el lugar de enlace y despliegue de los DataSources de ejemplo.

Conexiones

El objeto Connection representa una conexión con un origen de datos en Java Database Connectivity (JDBC). Los objetos Statement se crean a través de objetos Connection para procesar sentencias SQL en la base de datos. Un programa de aplicación puede tener varias conexiones simultáneas. Estos objetos Connection puede conectarse todos a la misma base de datos o a bases de datos diferentes.

La obtención de una conexión en JDBC puede realizarse de dos maneras:

- Mediante la clase DriverManager.
- Utilizando DataSources.

Es preferible utilizar DataSources para obtener una conexión, ya que mejora la portabilidad y la capacidad de mantenimiento de las aplicaciones. También permite que una aplicación utilice de forma transparente las agrupaciones de conexiones y sentencias y las transacciones distribuidas.

Conceptos relacionados

Crear diversos tipos de objetos Statement para interaccionar con la base de datos
El objeto Statement (sentencia) sirve para procesar una sentencia SQL estática y obtener los resultados producidos por ella. Solo puede haber un ResultSet abierto para cada objeto Statement en un momento dado. Todos los métodos statement que procesan una sentencia SQL cierran implícitamente el ResultSet actual de una sentencia si existe uno abierto.

Controlar transacciones con respecto a la base de datos
La transacción es una unidad de trabajo lógica. Para realizar una unidad de trabajo lógica, puede ser necesario llevar a cabo varias acciones con respecto a una base de datos.

Recuperar metadatos sobre la base de datos
El controlador JDBC de IBM Developer Kit para Java implementa la interfaz DatabaseMetaData para proporcionar información sobre los orígenes de datos subyacentes. La utilizan principalmente los servidores de aplicaciones y las herramientas para determinar cómo hay que interaccionar con un origen de datos dado. Las aplicaciones también pueden servirse de los métodos de DatabaseMetaData para obtener información sobre un origen de datos, pero esto ocurre con menos frecuencia.

Clase Java DriverManager:

DriverManager es una clase estática de Java 2 Plaform, Standard Edition (J2SE) y Java SE Development Kit (JDK). DriverManager gestiona el conjunto de controladores Java Database Connectivity (JDBC) que están disponibles para que los utilice una aplicación.

Las aplicaciones pueden utilizar varios controladores JDBC simultáneamente si es necesario. Cada aplicación especifica un controlador JDBC mediante la utilización de un URL (Localizador universal de recursos). Pasando un URL de un controlador JDBC específico a DriverManager, la aplicación informa a DriverManager acerca del tipo de conexión JDBC que debe devolverse a la aplicación.

Para poder realizar esta operación, DriverManager debe estar al corriente del los controladores JDBC disponibles para que pueda distribuir las conexiones. Efectuando una llamada al método Class.forName, carga una clase en la máquina virtual Java (JVM) que se está ejecutando en función del nombre de serie que se pasa en el método. A continuación figura un ejemplo del método class.forName utilizado para cargar el controlador JDBC nativo:

Ejemplo: cargar el controlador JDBC nativo

```
// Cargar el controlador JDBC nativo en DriverManager para hacerlo
// disponible para peticiones getConnection.

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

Los controladores JDBC están diseñados para informar a DriverManager acerca de sí mismos automáticamente cuando se carga su clase de implementación de controlador. Una vez que se ha procesado la línea de código mencionada anteriormente, el controlador JDBC nativo está disponible para la DriverManager con la que debe trabajar. La línea de código siguiente solicita un objeto Connection que utiliza el URL de JDBC nativo:

Ejemplo: solicitar un objeto Connection

```
// Obtener una conexión que utiliza el controlador JDBC nativo.

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

La forma más sencilla de URL JDBC es una lista de tres valores separados mediante dos puntos. El primer valor de la lista representa el protocolo, que es siempre `jdbc` para los URL JDBC. El segundo valor es el subprotocolo y se utiliza `db2` o `db2iSeries` para especificar el controlador JDBC nativo. El tercer valor es el nombre de sistema para establecer la conexión con un sistema específico. Existen dos valores especiales que pueden utilizarse para conectarse con la base de datos local. Son *LOCAL y localhost (ambos son sensibles a mayúsculas y minúsculas). También puede suministrarse un nombre de sistema específico, de la forma siguiente:

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

Así se crea una conexión con el sistema rchasmop. Si el sistema al que intenta conectarse es un sistema remoto (por ejemplo, a través de Distributed Relational Database architecture), debe utilizarse el nombre de sistema del directorio de bases de datos relacionales.

Nota: Si no se especifica lo contrario, el ID de usuario y la contraseña utilizados actualmente para iniciar la sesión también se utilizan para establecer la conexión con la base de datos.

Nota: El controlador IBM DB2 JDBC Universal también utiliza el subprotocolo db2. Para asegurarse de que el controlador JDBC nativo puede manejar el URL, las aplicaciones deben utilizar el URL jdbc:db2iSeries:xxxx en lugar del URL jdbc:db2:xxxx. Si la aplicación no desea que el controlador nativo acepte URLS con el subprotocolo db2, la aplicación deberá cargar la clase com.ibm.db2.jdbc.app.DB2iSeriesDriver, en lugar de com.ibm.db2.jdbc.app.DB2Driver. Al cargarse esta clase, el controlador nativo ya no tiene que manejar los URL que contienen el subprotocolo db2.

Propiedades

El método DriverManager.getConnection toma un URL de una sola serie indicado anteriormente, y solo es uno de los métodos de DriverManager destinado a obtener un objeto Connection. También existe otra versión del método DriverManager.getConnection que toma un ID de usuario y una contraseña. A continuación figura un ejemplo de esta versión:

Ejemplo: método DriverManager.getConnection que toma un ID de usuario y una contraseña

```
// Obtener una conexión que utiliza el controlador JDBC nativo.

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

La línea de código intenta conectarse con la base de datos local como usuario cujo con la contraseña newtiger sin importar quién ejecuta la aplicación. También existe una versión del método DriverManager.getConnection que toma un objeto java.util.Properties que permite una mayor personalización. A continuación se ofrece un ejemplo:

Ejemplo: método DriverManager.getConnection que toma un objeto java.util.Properties

```
// Obtener una conexión que utiliza el controlador JDBC nativo.

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

El código es funcionalmente equivalente a la versión mencionada anteriormente que ha pasado el ID de usuario y la contraseña como parámetros.

Consulte las Propiedades de Connection para obtener una lista completa de las propiedades de conexión del controlador JDBC nativo.

Propiedades de URL

Otra forma de especificar propiedades es colocarlas en una lista del propio objeto URL. Cada propiedad de la lista está separada mediante un signo de punto y coma, y la lista debe tener el formato `nombre propiedad=valor propiedad`. Solo existe un método abreviado que no cambia significativamente la forma en que se realiza el proceso, como muestra el ejemplo siguiente:

Ejemplo: especificar propiedades de URL

```
// Obtener una conexión que utiliza el controlador JDBC nativo.

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

De nuevo, el código es funcionalmente equivalente a los ejemplos mencionados anteriormente.

Si se especifica un valor de propiedad tanto en un objeto de propiedades como en el objeto URL, la versión de URL tiene preferencia sobre la versión del objeto de propiedades. A continuación se ofrece un ejemplo:

Ejemplo: propiedades de URL

```
// Obtener una conexión que utiliza el controlador JDBC nativo.
Properties prop = new java.util.Properties();
prop.put("user", "someone");
prop.put("password","something");
Connection c = DriverManager.getConnection("jdbc:db2:*local;user=cujo;password=newtiger",
prop);
```

El ejemplo utiliza el ID de usuario y la contraseña de la serie de URL en lugar de la versión del objeto Properties. Termina siendo funcionalmente equivalente al código mencionado anteriormente.

Ejemplo: ID de usuario y contraseña no válidos:

Este es un ejemplo de utilización de la propiedad Connection en la modalidad de denominación SQL.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de InvalidConnect.
//
// Este programa utiliza la propiedad Connection en modalidad de denominación SQL.
//
//////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo del controlador JDBC de IBM Developer para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo con fines ilustrativos
// solamente. Los ejemplos no se han probado minuciosamente bajo todas
// las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas contenidos aquí se proporcionan "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se declinan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// 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)
    {
        // Registrar el controlador.
        try {
```

```
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: el controlador JDBC no se ha cargado.");
            System.exit(0);
        }

        // Intento de obtener una conexión sin especificar ningún usuario o
        // la contraseña. El intento funciona y la conexión utiliza el
        // mismo perfil de usuario bajo el que se ejecuta el trabajo.
        try {
            Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
            c1.close();
        } catch (SQLException e) {
            System.out.println("Esta prueba no debe incluirse en esta vía de excepciones.");
            e.printStackTrace();
            System.exit(1);
        }

        try {
            Connection c2 = DriverManager.getConnection("jdbc:db2:*local",
                                                        "notvalid", "notvalid");
        } catch (SQLException e) {
            System.out.println("Este es un error esperado.");
            System.out.println("El mensaje es " + e.getMessage());
            System.out.println("SQLSTATE es " + e.getSQLState());
        }

    }
}
```

Propiedades de conexión del controlador JDBC:

En esta tabla figuran las propiedades válidas de conexión para el controlador JDBC, los valores que tienen y sus descripciones.

Propiedad	Valores	Significado
access (acceso)	all, read call, read only	Este valor permite restringir el tipo de operaciones que se pueden realizar con una determinada conexión. El valor predeterminado es ″all″, que básicamente significa que la conexión tiene pleno acceso a la API de JDBC. El valor ″read call″ (llamada de lectura) solo permite que la conexión haga consultas y llame a procedimientos almacenados. Se impide todo intento de actualizar la base de datos con una sentencia SQL. El valor solo de lectura ″read only″ permite restringir una conexión para que solo pueda hacer consultas. Se impiden las llamadas a procedimientos almacenados y las sentencias de actualización.
auto commit (compromiso automático)	true, false	Este valor se utiliza para establecer el compromiso automático de la conexión. El valor predeterminado es true a menos que se haya establecido la propiedad de aislamiento de transacción con un valor distino a ninguno. En ese caso, el valor predeterminado es false.

Propiedad	Valores	Significado
batch style (estilo de lotes)	2.0, 2.1	La especificación JDBC 2.1 define un segundo método para manejar las excepciones de una actualización por lotes. El controlador puede ajustarse a cualquiera de ellos. El valor predeterminado es trabajar según lo definido en la especificación JDBC 2.0.
block size (tamaño de bloque)	0, 8, 16, 32, 64, 128, 256, 512	Este es el número de filas que se extraen de una sola vez para un conjunto de resultados. En un proceso habitual solo hacia adelante de un conjunto de resultados, se obtiene un bloque de este tamaño. Entonces no es necesario acceder a la base de datos ya que la aplicación procesa cada fila. La base de datos solo solicitará otro bloque de datos cuando se haya llegado al final del bloque. Este valor solo se utiliza si la propiedad de habilitado para bloques (blocking enabled) se establece como true. Establecer la propiedad de tamaño de bloque en 0 tiene el mismo efecto que establecer la propiedad de habilitado para bloques como false. El valor predeterminado es utilizar la agrupación en bloques con un tamaño de bloque de 32. Esta decisión es completamente arbitraria, por lo que el valor predeterminado podría cambiar en el futuros. La agrupación en bloques no se utiliza en los conjuntos de resultados desplazables.
blocking enabled (habilitado para bloques)	true, false	Este valor sirve para determinar si la conexión utiliza bloques en la recuperación de filas de conjunto de resultados. La agrupación en bloques puede aumentar notablemente el rendimiento al procesar conjuntos de resultados. Por omisión, esta propiedad está establecida en true.

Propiedad	Valores	Significado
commit hold (retención de compromiso)	false, true	Este valor especifica si se emplea ″commit hold″ al llamar al método connection.commit(). Cuando se utiliza ″commit hold″, los cursores y otros recursos de base de datos no se cierran ni se liberan al llamar a commit. El valor predeterminado es false.
cursor hold (retención de cursor)	true, false	Este valor especifica si los conjuntos de resultados permanecen abiertos cuando se compromete una transacción. El valor true indica que una aplicación puede acceder a sus conjuntos de resultados abiertos una vez llamado el compromiso. El valor false indica que el compromiso cierra los cursores abiertos en la conexión. Por omisión, esta propiedad está establecida en true. Este valor de propiedad funciona como valor predeterminado para todos los conjuntos de resultados establecidos para la conexión. Con el soporte de retención de cursor añadido en JDBC 3.0, este valor predeterminado se sustituye simplemente si una aplicación especifica posteriormente una capacidad de retención diferente. Si se emigra de una versión anterior a la JDBC 3.0, hay que tener en cuenta que el soporte de retención del cursor se añade por primera vez en JDBC 3.0. En versiones anteriores, el valor predeterminado ″true″ se envia durante el tiempo de conexión pero la Máquina virtual Java todavía no lo reconoce. Por tanto, la propiedad de retención del cursor no impactará en la funcionalidad de la base de datos hasta JDBC 3.0.
cursor sensitivity (sensibilidad de cursor)	asensitive, sensitive	Especifica la sensibilidad empleada por los cursores de ResultSet.TYPE_SCROLL_SENSITIVE. Por defecto, el controlador JDBC nativo crea cursores no sensibles en el caso de los cursores ResultSet.TYPE_SCROLL_SENSITIVE.

Propiedad	Valores	Significado
data truncation (truncamiento de datos)	true, false	Este valor especifica si el truncamiento de datos de tipo carácter provoca avisos y excepciones (true) o si los datos deben truncarse de forma silenciosa (false). Si el valor predeterminado es true, debe aceptarse el truncamiento de datos en los campos de caracteres.
date format (formato de fecha)	juliano, mdy, dmy, ymd, usa, iso, eur, jis	Esta propiedad permite cambiar el formato de las fechas.
date separator (separador de fecha)	/(barra inclinada), -(guión), .(punto), ,(coma), blanco	Esta propiedad permite cambiar el separador de fecha. Solo es válida en combinación con algunos de los valores de dateFormat (según las normas del sistema).
decfloat rounding mode (modalidad de redondeo de flotante decimal)	round half even, round half up, round down, round ceiling, round floor, round half down, round up, round half even	Esta propiedad especifica la modalidad de redondeo que hay que utilizar en las operaciones de número decimales flotantes. El valor predeterminado es round half even.
decimal separator (separador de decimales)	.(punto), ,(coma)	Esta propiedad permite cambiar el separador de decimales.
direct map (correlación directa)	true, false	Esta propiedad especifica si se emplearán optimizaciones de correlación directa de base de datos al recuperar conjuntos de resultados de la base de datos. El valor predeterminado es true.

Propiedad	Valores	Significado
do escape processing (hacer proceso de escape)	true, false	Esta propiedad establece un distintivo que indica si las sentencias bajo la conexión deben hacer un proceso de escape. La utilización del proceso de escape es una manera de codificar las sentencias SQL para que sean genéricas y similares para todas las plataformas, pero luego la base de datos lee las cláusulas de escape y sustituye la debida versión específica del sistema para el usuario. Es una propiedad valiosa, salvo que implica hacer un trabajo adicional en el sistema. Si se sabe que solo se van a utilizar sentencias SQL que ya contienen sintaxis SQL válida de i5/OS, conviene establecer que este valor sea ″false″ para aumentar el rendimiento. El valor predeterminado de esta propiedad es ″true″, ya que debe estar en conformidad con la especificación JDBC (es decir, el proceso de escape está activo por omisión). Este valor se ha añadido debido a una deficiencia de la especificación JDBC. Solo se puede establecer que el proceso de escape está desactivado en la clase Statement. Eso funciona correctamente si se trata de sentencias simples. Basta con crear la sentencia, desactivar el proceso de escape y empezar a ejecutar las sentencias. Sin embargo, en el caso de las sentencias preparadas y de las sentencias invocables, este esquema no funciona. Se suministra la sentencia SQL en el momento de construir la sentencia preparada o la sentencia invocable y esta no cambia después de ello. Así que la sentencia queda preparada en primer lugar y el hecho de cambiar el proceso de escape más adelante no tiene ningún significado. Gracias a esta propiedad de conexión, existe un modo de soslayar la actividad general adicional.
errors (errores)	basic, full	Esta propiedad permite devolver el texto de errores de segundo nivel de todo el sistema en mensajes de objeto SQLException. El valor predeterminado es basic, que devuelve solo el texto de mensaje estándar.

Propiedad	Valores	Significado
extended metadata (metadatos ampliados)	true, false	Esta propiedad especifica si el controlador debe solicitar metadatos ampliados de la base de datos. Si esta propiedad es igual a true, aumenta la exactitud de la información que devuelven los siguientes métodos de ResultSetMetaData: • getColumnLabel(int) • getSchemaName(int) • getTableName(int) • isReadOnly(int) • isSearchable(int) • isWriteable(int) Cuando esta propiedad es true, el rendimiento puede disminuir, porque hay que recuperar más información de la base de datos.
ignore warnings (ignorar avisos)	Lista de estados SQL, separada por mandatos, que se deben ignorar.	Por defecto, el controlador JDBC nativo creará internamente un objeto java.sql.SQLWarning para cada aviso devuelto por la base de datos. Esta propiedad especifica una lista de los estados SQL para los que el controlador JDBC nativo no debe crear objetos de tipo aviso. Por ejemplo, se crea un aviso con el estado SQLSTATE 0100C cada vez que se devuelve un conjunto de resultados desde un procedimiento almacenado. Este aviso se puede ignorar sin problemas para mejorar el rendimiento de las aplicaciones que llaman a procedimientos almacenados.

Propiedad	Valores	Significado
libraries (bibliotecas)	Una lista de bibliotecas separadas mediante espacios. (Una lista de bibliotecas también puede separarse mediante signos de dos puntos o comas).	Esta propiedad permite colocar una lista de bibliotecas en la lista de bibliotecas del trabajo servidor o establecer una lista de bibliotecas específica. La propiedad de denominación, ″naming″, afecta al funcionamiento de esta propiedad. En el caso por omisión, en que ″naming″ está establecida en sql, JDBC funciona como ODBC. La lista de bibliotecas no tiene ningún efecto sobre el proceso que efectúa la conexión. Existe una biblioteca por omisión para todas las tablas no calificadas. Por omisión, la biblioteca tiene el mismo nombre que el perfil de usuario al que está conectado. Si se especifica la propiedad ″libraries″, la primera biblioteca de la lista pasa a ser la biblioteca por omisión. Si se especifica una biblioteca por omisión en el URL de conexión (como en jdbc:db2:*local/mibilioteca), se altera temporalmente cualquier valor de esta propiedad. Si ″naming″ se establece en ″system″, cada una de las bibliotecas especificadas para esta propiedad se añade a la parte del usuario de la lista de bibliotecas y se busca en la lista de bibliotecas para resolver las referencias de tabla no calificadas.

Propiedad	Valores	Significado
lob threshold (umbral de lob)	Cualquier valor por debajo de 500000	Esta propiedad indica al controlador que coloque los valores reales en el almacenamiento de conjunto de resultados en lugar de localizadores de columnas lob si la columna lob es inferior al tamaño del umbral. Esta propiedad actúa sobre el tamaño de columna, no sobre el tamaño de los datos lob propiamente. Por ejemplo, si la columna lob está definida para contener hasta 1 MB para cada lob, pero todos los valores de columna están por debajo de 500 MB, se siguen utilizando localizadores. Tenga en cuenta que el límite de tamaño se establece de forma que permita extraer los bloques de datos sin el riesgo de que los bloques de datos crezcan más allá de los 16 MB de máximo de tamaño de asignación. En conjuntos de resultados mayores, sigue siendo fácil sobrepasar este límite, lo cual provoca anomalías en las extracciones. Debe tener cuidado en la forma en que la propiedad block size y esta propiedad interactúan con el tamaño de un bloque de datos. El valor predeterminado es 0. Siempre se utilizan localizadores para datos lob.
maximum precision (precisión máxima)	31, 63	Este valor especifica la precisión máximo empleada para los datos de tipo decimal y numérico. El valor predeterminado es 31.
maximum scale (escala máxima)	0-63	Este valor especifica la escala máxima (número de posiciones decimales a la derecha de la coma decimal) que se devuelve para los datos de tipo decimal y numérico. El valor puede ir de 0 a la precisión máxima. El valor predeterminado es 31.
minimum divide scale (escala de división mínima)	0-9	Este valor especifica la escala de división mínima (número de posiciones decimales a la derecha de la coma decimal) que se devuelve para los tipos de datos intermedios y de resultados. El valor puede ir de 0 a 9, sin sobrepasar la escala máxima. Si se especifica 0, no se utiliza la escala de división mínima. El valor predeterminado es 0.

Propiedad	Valores	Significado
naming (denominación)	sql, system	Esta propiedad le permite utilizar la sintaxis de denominación tradicional de System i o la sintaxis de denominación estándar de SQL. La denominación del sistema significa que utilizará un carácter /(barra inclinada) para separar los valores de colección y de tabla, y la denominación SQL significa que utilizará un carácter .(punto) para separar los valores. El establecimiento de este valor tiene ramificaciones que afectan también a cuál es la biblioteca por omisión. Hallará más información al respecto en la propiedad libraries, más arriba. El valor predeterminado es utilizar la denominación SQL.
password (contraseña)	cualquier valor	Esta propiedad prevé la especificación de una contraseña para la conexión. Esta propiedad no funciona correctamente si no se especifica también la propiedad de usuario, "user". Estas propiedades permiten establecer conexiones con la base de datos en los casos en que el usuario no coincida con el que está ejecutando el trabajo de System i. Especificar las propiedades de usuario y contraseña tiene el mismo efecto que utilizar el método de conexión con la firma getConnection(String url, String userId, String password).
prefetch (captación previa)	true, false	Esta propiedad especifica si el controlador capta los primeros datos de un conjunto de resultados inmediatamente después del proceso o si espera hasta que se soliciten los datos. Si el valor predeterminado es true, hay que precaptar los datos. En las aplicaciones que utilizan el controlador JDBC nativo, esto rara vez representa un problema. La propiedad existe principalmente para uso interno con procedimientos almacenados Java y funciones definidas por usuario en las que es importante que el motor de bases de datos no extraiga ningún dato de los conjuntos de resultados en nombre del usuario antes de que este lo solicite.

Propiedad	Valores	Significado
qaqqinilib	library name (nombre de biblioteca)	Esta propiedad especifica la biblioteca que contiene el archivo qaqqini que hay que utilizar. En el archivo qaqqini están todos los atributos que pueden afectar potencialmente al rendimiento del motor de la base de datos DB2 para i5/OS.
query optimize goal (objetivo de optimización de consulta)	1, 2	Esta propiedad especifica el objetivo que debe tener el servidor con respecto a la optimización de las consultas. Este valor se corresponde con la opción, llamada OPTIMIZATION_GOAL, de QAQQINI del servidor. Los valores posibles son: **1** Optimizar la consulta para el primer bloque de datos (*FIRSTIO) **2** Optimizar la consulta para todo el conjunto de resultados (*ALLIO) El valor predeterminado es 2.
reuse objects (reutilizar objetos)	true, false	Esta propiedad especifica si el controlador intenta reutilizar algunos tipos de objetos después de que usted los haya cerrado. Esto representa una mejora en el rendimiento. El valor predeterminado es true.
time format (formato de hora)	hms, usa, iso, eur, jis	Esta propiedad permite cambiar el formato de los valores de hora.
time separator (separador de hora)	:(dos puntos), .(punto), ,(coma), blanco	Esta propiedad permite cambiar el separador de hora. Solo es válida en combinación con algunos de los valores de timeFormat (según las normas del sistema).
trace (rastreo)	true, false	Esta propiedad prevé la activación del rastreo de la conexión. Se puede utilizar como una simple ayuda para la depuración. El valor predeterminado es ″false″, que corresponde a no utilizar el rastreo.

Propiedad	Valores	Significado
transaction isolation (aislamiento de transacciones)	none, read committed, read uncommitted, repeatable read, serializable	Esta propiedad permite al usuario establecer el nivel de aislamiento de transacción para la conexión. No hay ninguna diferencia entre establecer esta propiedad en un nivel concreto y especificar un nivel en el método setTransactionIsolation() de la interfaz Connection. El valor predeterminado de esta propiedad es ″none″, El valor predeterminado de esta propiedad es ″none″, ya que JDBC toma por omisión la modalidad de compromiso automático.
translate binary (convertir binario)	true, false	Esta propiedad puede utilizarse para obligar al controlador JDBC a que trate los valores de datos de tipo binary y varbinary como si fuesen valores de datos de tipo char y varchar. El valor predeterminado de esta propiedad es ″false″, es decir, no tratar los datos de tipo binario como si fuesen datos de tipo carácter.
translate hex (convertir hexadecimal)	binario, carácter	Este valor se utiliza para seleccionar el tipo de datos utilizado por las constantes hex en expresiones SQL. El valor binario indica que las constantes hex utilizarán el tipo de datos BINARY FOR BIT DATA. El valor carácter indica que las constantes hex utilizarán el tipo de datos CHARACTER FOR BIT DATA. El valor predeterminado es carácter.

Propiedad	Valores	Significado
use block insert (utilizar inserción en bloque)	true, false	Esta propiedad permite al controlador JDBC nativo colocarse en modalidad de inserción en bloque para insertar bloques de datos en la base de datos. Esta es una versión optimizada de la actualización por lotes. Esta modalidad optimizada solo puede utilizarse en aplicaciones que garanticen no transgredir determinadas restricciones del sistema ni producir anomalías de inserción de datos, pudiendo dañar los datos. Las aplicaciones que activen esta propiedad solo deben conectarse al sistema local al intentar realizar actualizaciones por lotes. No deben utilizar DRDA para establecer conexiones remotas, ya que la inserción en bloque no puede gestionarse a través de DRDA. Las aplicaciones también deben asegurarse de que PreparedStatements con una sentencia SQL insert y una cláusula values indican todos los parámetros de valores de inserción. No se permiten contantes en la lista de valores. Este es un requisito del motor de inserción por bloques del sistema. El valor predeterminado es false.
user (usuario)	cualquier valor	Esta propiedad permite especificar un ID de usuario para la conexión. Esta propiedad no funciona correctamente si no se especifica también la propiedad de contraseña, "password". Estas propiedades permiten establecer conexiones con la base de datos en los casos en que el usuario no coincida con el que está ejecutando el trabajo de System i. Especificar las propiedades de usuario y contraseña tiene el mismo efecto que utilizar el método de conexión con la firma getConnection(String url, String userId, String password).

Utilizar DataSources con UDBDataSource:

Las interfaces DataSource proporcionan una flexibilidad adicional al utilizar controladores Java Database Connectivity (JDBC).

La utilización de DataSources puede dividirse en dos fases:

- **Despliegue**

  El despliegue es una fase de configuración que se produce antes de que una aplicación JDBC se ejecute realmente. En general, el despliegue implica configurar un DataSource con propiedades específicas y luego enlazarlo con un servicio de directorio mediante Java Naming and Directory Interface (JDNI). El servicio de directorio suele ser el protocolo ligero de acceso a directorio (LDAP), pero también podrías ser otros como los servicios de objeto de la arquitectura de intermediarios de peticiones de objetos comunes (CORBA), la invocación de método remoto (RMI) Java o el sistema de archivos subyacente.
- **Utilización**

  Al desasociar el despliegue de la utilización en tiempo de ejecución del DataSource, muchas aplicaciones pueden reutilizar la configuración de DataSource. Cambiando alguno de los aspectos del despliegue, todas las aplicaciones que utilizan ese DataSource recogen los cambios automáticamente.

Nota: Tenga en cuenta que utilizar RMI puede resultar una tarea compleja. Antes de elegir RMI como solución, asegúrese de comprender las ramificaciones que puede conllevar.

Una de las ventajas de los DataSources es que permiten a los controladores JDBC efectuar el trabajo en nombre de la aplicación sin influir directamente sobre el proceso de desarrollo de la misma. Para obtener más información, consulte lo siguiente:

- "Utilizar soporte de DataSource para la agrupación de objetos" en la página 132
- "Agrupación de sentencias basada en DataSource" en la página 136
- "Transacciones JDBC distribuidas" en la página 82

UDBDataSourceBind

El programa "Ejemplo: crear un UDBDataSource y enlazarlo con JNDI" en la página 59 es un ejemplo de cómo crear un UDBDataSource y enlazarlo con JNDI. Este programa realiza todas las tareas básicas solicitadas. Es decir, crea una instancia de un objeto UDBDataSource, establece las propiedades de este objeto, recupera un contexto JNDI y enlaza el objeto con un nombre del contexto JNDI.

El código de despliegue es específico del proveedor. La aplicación debe importar la implementación específica de DataSource con la que desea trabajar. En la lista de importación, se importa la clase UDBDataSource calificada por paquete. La parte menos conocida de esta aplicación es el trabajo que se realiza con JNDI (por ejemplo, la recuperación del objeto Context y la llamada a bind). Para obtener información adicional, vea: JNDI de Sun Microsystems, Inc.

Una vez que este programa se ha ejecutado y completado satisfactoriamente, existe una entrada nueva en un servicio de directorio JNDI denominada SimpleDS. Esta entrada se encuentra en la ubicación especificada por el contexto JNDI. Ahora, se despliega la implementación de DataSource. Un programa de aplicación puede utilizar este DataSource para recuperar conexiones de base de datos y trabajo relacionado con JDBC.

UDBDataSourceUse

El programa "Ejemplo: obtener un contexto inicial antes de enlazar UDBDataSource" en la página 60 es un ejemplo de una aplicación JDBC que utiliza la aplicación desplegada anteriormente.

La aplicación JDBC obtiene un contexto inicial al igual que hizo antes enlazando el UDBDataSource en el ejemplo anterior. A continuación, se utiliza el método lookup en ese contexto para devolver un objeto de tipo DataSource para que lo utilice la aplicación.

Nota: La aplicación de ejecución solo está interesada en los métodos de la interfaz DataSource, y por tanto no es necesario que esté al corriente de la clase de implementación. Esto hace que la aplicación sea portable.

Suponga que UDBDataSourceUse es una aplicación compleja que ejecuta una operación de grandes dimensiones dentro de la empresa. En la empresa existen una docena o más de aplicaciones similares de grandes dimensiones. Es necesario cambiar el nombre de uno de los sistemas de la red. Ejecutando una herramienta de despliegue y cambiando una sola propiedad de UDBDataSource, es posible conseguir este comportamiento nuevo en todas las aplicaciones sin cambiar el código de las mismas. Una de las ventajas de los DataSources es que permiten consolidar la información de configuración del sistema. Otra de las ventajas principales es que permiten a los controladores implementar funciones invisibles para la aplicación, como por ejemplo agrupación de conexiones, agrupación de sentencias y soporte para transacciones distribuidas.

Después de analizar detenidamente UDBDataSourceBind y UDBDataSourceUse, quizá se haya preguntado cómo es posible que el objeto DataSource sepa lo que debe hacer. No existe ningún código que especifique un sistema, un ID de usuario o una contraseña en ninguno de estos programas. La clase UDBDataSource tiene valores predeterminados para todas las propiedades; por defecto, se conecta al System i local con el perfil de usuario y la contraseña de la aplicación que se ejecuta. Si deseara asegurarse de que, en lugar de ello, la conexión se ha efectuado con el perfil de usuario cujo, podría hacerlo de dos maneras:

- Estableciendo el ID de usuario y la contraseña como propiedades de DataSource. Para saber cómo se utiliza esta técnica, vea: “Ejemplo: crear un UDBDataSourceBind y establecer las propiedades de DataSource” en la página 60.
- Utilizando el método getConnection de DataSource, que toma un ID de usuario y una contraseña durante la ejecución . Para saber cómo se utiliza esta técnica, vea: “Ejemplo: crear un UDBDataSource y obtener un ID de usuario y una contraseña” en la página 61“Ejemplo: crear un UDBDataSource y obtener un ID de usuario y una contraseña” en la página 61.

Existen diversas propiedades que pueden especificarse para UDBDataSource, al igual que existen propiedades que pueden especificarse para las conexiones creadas con DriverManager. Para obtener una lista de propiedades soportadas para el controlador JDBC nativo, consulte: “Propiedades de DataSource” en la página 62.

Aunque estas listas son similares, no es seguro que lo sean en releases futuros. Le animamos a que empiece la codificación en la interfaz DataSource.

Nota: El controlador JDBC nativo también tiene otras dos implementaciones de DataSource: DB2DataSource y DB2StdDataSource. Estas implementaciones han caído en desuso y no conviene utilizarlas directamente. En un futuro release se prescindirá de estas implementaciones.

Ejemplo: crear un UDBDataSource y enlazarlo con JNDI:

Este es un ejemplo de cómo crear un UDBDataSource y enlazarlo con JNDI.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
// Importar los paquetes necesarios. En el momento del despliegue,
// la clase específica del controlador JDBC que implementa
// DataSource se tiene que importar.
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
    {
        // Crear un nuevo objeto UDBDataSource y proporcionarle
        // una descripción.
        UDBDataSource ds = new UDBDataSource();
```

```
        ds.setDescription("Un UDBDataSource simple");

        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Enlazar el objeto UDBDataSource recién creado
        // con el servicio de directorios JNDI, dándole un nombre
        // que pueda utilizarse para buscar de nuevo este objeto
        // más adelante.
        ctx.rebind("SimpleDS", ds);
    }
}
```

Ejemplo: crear un UDBDataSourceBind y establecer las propiedades de DataSource:

Este es un ejemplo de cómo crear un UDBDataSource y establecer el ID de usuario y la contraseña como propiedades de DataSource.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Importar los paquetes necesarios. En el momento del despliegue,
// la clase específica del controlador JDBC que implementa
// DataSource se tiene que importar.
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
    {
        // Crear un nuevo objeto UDBDataSource y proporcionarle
        // una descripción.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("Un UDBDataSource simple"  +
                          "con cujo como perfil por" +
                          "defecto al que conectarse.");

        // Proporcionar un ID de usuario y una contraseña que se deban usar para
        // las propiedades de conexión.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Enlazar el objeto UDBDataSource recién creado
        // con el servicio de directorios JNDI, dándole un nombre
        // que pueda utilizarse para buscar de nuevo este objeto
        // más adelante.
        ctx.rebind("SimpleDS2", ds);
    }
}
```

Ejemplo: obtener un contexto inicial antes de enlazar UDBDataSource:

En el ejemplo siguiente se obtiene un contexto inicial antes de enlazar el UDBDataSource. A continuación, se utiliza el método lookup en ese contexto para devolver un objeto de tipo DataSource para que lo utilice la aplicación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Importar los paquetes necesarios. No hay ningún
// código específico del controlador que se necesite en las
// aplicaciones de tiempo de ejecución.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Recuperar el objeto UDBDataSource enlazado mediante el
        // nombre con el que estaba enlazado anteriormente. En tiempo de ejecución,
        // solo se utiliza la interfaz DataSource y, por lo tanto,
        // no hace falta convertir el objeto a la clase de
        // implementación de UDBDataSource. (No hace falta saber cuál
        // es la clase de implementación. El nombre JNDI lógico es
        // lo único necesario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una vez obtenido, el DataSource se puede usar para establecer
        // una conexión. Este objeto Connection es el mismo tipo
        // de objeto que el que se devuelve si se emplea el enfoque DriverManager
        // para establecer conexión. Por lo tanto, todo lo que hay desde
        // este punto en adelante es exactamente igual que en cualquier otra
        // aplicación JDBC.
        Connection connection = ds.getConnection();

        // La conexión puede utilizarse para crear objetos Statement y
        // actualizar la base de datos o procesar consultas de la forma siguiente.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // La conexión se cierra antes de que finalice la aplicación.
        connection.close();
    }
}
```

Ejemplo: crear un UDBDataSource y obtener un ID de usuario y una contraseña:

Este es un ejemplo de cómo crear un UDBDataSource y utilizar el método getConnection para obtener un ID de usuario y una contraseña durante la ejecución.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/// Importar los paquetes necesarios.
// No hay código específico del controlador que se necesite en las
// aplicaciones de tiempo de ejecución.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
```

```
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Recuperar el objeto UDBDataSource enlazado mediante el
        // nombre con el que estaba enlazado anteriormente. En tiempo de ejecución,
        // solo se utiliza la interfaz DataSource y, por lo tanto,
        // no hace falta convertir el objeto a la clase de
        // implementación de UDBDataSource. (No hace falta saber
        // cuál es la clase de implementación. El nombre JNDI lógico
        // es lo único necesario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una vez obtenido el DataSource, puede utilizarse para establecer
        // una conexión. El perfil de usuario cujo y la contraseña newtiger
        // se utilizan para crear la conexión en lugar del ID de usuario
        // y la contraseña por omisión para el DataSource.
        Connection connection = ds.getConnection("cujo", "newtiger");

        // La conexión puede utilizarse para crear objetos Statement y
        // actualizar la base de datos o procesar consultas de la forma siguiente.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // La conexión se cierra antes de que finalice la aplicación.
        connection.close();
    }
}
```

Propiedades de DataSource:

Para cada propiedad de conexión de controlador JDBC, existe un correspondiente método de origen de datos. En esta tabla figuran las propiedades válidas de los orígenes de datos.

En el caso de algunas propiedades, puede consultar la correspondiente propiedad de conexión de controlador para obtener más información.

Método set (tipo de datos)	Valores	Descripción
setAccess(String)	″all″, ″read call″, ″read only″	Consulte la propiedad de conexión access (acceso).
setAutoCommit(boolean)	″true″, ″false″	Consulte la propiedad de conexión auto commit (comprometer automático).
setBatchStyle(String)	″2.0″, ″2.1″	Consulte la propiedad de conexión batch style (estilo de por lotes).
setBlockSize(int)	″0″, ″8″, ″16″, ″32″, ″64″, ″128″, ″256″, ″512″	Consulte la propiedad de conexión block size (tamaño de bloque).
setCommitHold(boolean)	″true″, ″false″	Consulte la propiedad de conexión commit hold (retención de compromiso).
setCursorHold(boolean)	″true″, ″false″	Consulte la propiedad de conexión cursor hold (retención de cursor).

Método set (tipo de datos)	Valores	Descripción
setCursorSensitivity(String)	"sensitive", "asensitive"	Consulte la propiedad de conexión cursor sensitivity (sensibilidad de cursor).
setDataTruncation(boolean)	"true", "false"	Consulte la propiedad de conexión data truncation (truncamiento de datos).
setDatabaseName(String)	Cualquier nombre	Esta propiedad especifica la base de datos a la que DataSource intenta conectarse. El valor predeterminado es *LOCAL. El nombre de base de datos debe existir en el directorio de bases de datos relacionales del sistema que ejecuta la aplicación o ser el valor especial *LOCAL o localhost para especificar el sistema local.
setDataSourceName(String)	Cualquier nombre	Esta propiedad permite pasar un nombre JNDI (Java Naming and Directory Interface) ConnectionPoolDataSource para dar soporte a la agrupación de conexiones.
setDateFormat(String)	"julian", "mdy", "dmy", "ymd", "usa", "iso", "eur", "jis"	Consulte la propiedad de conexión date format (formato de fecha).
setDateSeparator(String)	"/", "-", ".", ",", "b"	Consulte la propiedad de conexión date separator (separador de fecha).
setDecimalSeparator(String)	".", ","	Consulte la propiedad de conexión decimal separator (separador de decimales).
setDecfloatRoundingMode(String)	"round half even", "round half up", "round down", "round ceiling", "round floor", "round half down", "round up", "round half even"	Consulte la propiedad de conexión decfloat rounding (redondeo de decimal flotante).
setDescription(String)	Cualquier nombre	Esta propiedad permite establecer el texto descriptivo de este objeto DataSource.
setDirectMap(boolean)	"true", "false"	Consulte la propiedad de conexión direct map (correlación directa).
setDoEscapeProcessing(boolean)	"true", "false"	Consulte la propiedad de conexión do escape (realizar proceso de escape).
setFullErrors(boolean)	"true", "false"	Consulte la propiedad de conexión errors (errores).
setIgnoreWarnings(String)	Lista de estados SQLSTATE, separados por comas.	Consulte la propiedad de conexión ignore warnings (ignorar avisos).
setLibraries(String)	Lista de bibliotecas separadas mediante espacios.	Consulte la propiedad de conexión libraries (bibliotecas).
setLobThreshold(int)	Cualquier valor por debajo de 500000	Consulte la propiedad de conexión lob threshold (umbral de lob).
setLoginTimeout(int)	Cualquier valor	Esta propiedad se pasa por alto actualmente; está prevista para uso futuro.

Método set (tipo de datos)	Valores	Descripción
setMaximumPrecision(int)	31, 63	Consulte la propiedad de conexión maximum precision (precisión máxima).
setMaximumScale(int)	0-63	Consulte la propiedad de conexión maximum scale (escala máxima).
setMinimumDivideScale(int)	0-9	Consulte la propiedad de conexión minimum divide scale (escala mínima de división).
setNetworkProtocol(int)	Cualquier valor	Esta propiedad se pasa por alto actualmente; está prevista para uso futuro.
setPassword(String)	Cualquier serie	Consulte la propiedad de conexión password (contraseña).
setPortNumber(int)	Cualquier valor	Esta propiedad se pasa por alto actualmente; está prevista para uso futuro.
setPrefetch(boolean)	″true″, ″false″	Consulte la propiedad de conexión prefetch (captación previa).
setQaqqinilib(String)	nombre de la biblioteca	Consulte la propiedad de conexión qaqqinilib.
setQueryOptimizeGoal(String)	1, 2	Consulte la propiedad de conexión query optimize goal (objetivo de optimización de consulta).
setReuseObjects(boolean)	″true″, ″false″	Consulte la propiedad de conexión reuse objects (reutilizar objetos).
setServerName(String)	Cualquier nombre	Esta propiedad se pasa por alto actualmente; está prevista para uso futuro.
setSystemNaming(boolean)	″true″, ″false″	Consulte la propiedad de conexión naming (denominación).
setTimeFormat(String)	″hms″, ″usa″, ″iso″, ″eur″, ″jis″	Consulte la propiedad de conexión time format (formato de hora).
setTimeSeparator(String)	″:″, ″.″, ″,″, ″b″	Consulte la propiedad de conexión time separator (separador de hora).
setTransactionIsolationLevel(String)	″none″, ″read committed″, ″read uncommitted″, ″repeatable read″, ″serializable″	Consulte la propiedad de conexión transaction isolation (aislamiento de transacción).
setTranslateBinary(Boolean)	″true″, ″false″	Consulte la propiedad de conexión translate binary (convertir binario).
setUseBlockInsert(boolean)	″true″, ″false″	Consulte la propiedad de conexión use block insert (usar inserción en bloque).
setUser(String)	cualquier valor	Consulte la propiedad de conexión user (usuario).

Propiedades de la JVM para JDBC

Algunos valores utilizados por el controlador JDBC nativo no pueden establecerse utilizando una propiedad de conexión. Estos valores deben establecerse para la JVM en la que se ejecuta el controlador JDBC nativo. Estos valores se utilizan para todas las conexiones creadas por el controlador JDBC nativo.

El controlador nativo reconoce las siguientes propiedades de la JVM:

Propiedad	Valores	Significado
jdbc.db2.job.sort.sequence	valor predeterminado = *HEX	Establecer esta propiedad como verdadera provoca que el controlador JDBC nativo utilice la Secuencia de ordenación de trabajos del usuario que inicia el trabajo en lugar de utilizar el valor predeterminado de *HEX. Establecerla con otro valor o dejarla en blanco provocará que JDBC continúe utilizando el valor predeterminado de *HEX. Tenga en cuenta lo que esto significa. Cuando las conexiones JDBC se pasan en perfiles de usuario distintos en peticiones de conexión, la secuencia de ordenación del perfil de usuario que inicia el servidor se utiliza para todas las conexiones. Este es un atributo de entorno que se establece en el momento del inicio, no un atributo de conexión dinámica.
jdbc.db2.trace	1 o error = error de información de rastreo 2 o info = información de rastreo e información de error 3 o verbose = Trace verboso, información, e información de error 4 o todo o verdadero = Rastrear toda la información posible	Esta propiedad activa el rastreo para el controlador JDBC. Deberá utilizarse al informar de un problema.
jdbc.db2.trace.config	stdout = Se envía información de rastreo a stdout (valor predeterminado) usrtrc = Se envía información de rastreo a un rastreo de usuario. El mandato CL Volcar almacenamiento intermedio de rastreo de usuario (DMPUSRTRC) puede utilizarse para obtener la información de rastreo. file://<pathtofile> = Se envía información de rastreo a un archivo. Si el nombre de archivo contiene "%j", se sustituirá "%j" por el nombre de trabajo. Un ejemplo de <pathtofile> es /tmp/jdbc.%j.trace.txt.	Este propiedad se utiliza para especificar a dónde debe ir la salida del rastreo.

Interfaz DatabaseMetaData de IBM Developer Kit para Java

El controlador JDBC de IBM Developer Kit para Java implementa la interfaz DatabaseMetaData para proporcionar información sobre los orígenes de datos subyacentes. La utilizan principalmente los servidores de aplicaciones y las herramientas para determinar cómo hay que interaccionar con un origen de datos dado. Las aplicaciones también pueden servirse de los métodos de DatabaseMetaData para obtener información sobre un origen de datos, pero esto ocurre con menos frecuencia.

La interfaz DatabaseMetaData incluye alrededor de 150 métodos, que se pueden clasificar en categorías en función de los tipos de información que proporcionan. Éstos están descritos abajo. La interfaz DatabaseMetaData también contiene alrededor de 40 campos, que son constantes empleadas como valores de retorno en los diversos métodos de DatabaseMetaData.

Para obtener información sobre los cambios realizados en los métodos de la interfaz DatabaseMetaData, vea ″Cambios en JDBC 3.0″ y ″Cambios en JDBC 4.0″, más abajo.

Crear un objeto DatabaseMetaData

Un objeto DatabaseMetaData se crea con el método getMetaData de Connection. Una vez creado el objeto, puede utilizarse para buscar dinámicamente información acerca del origen de datos subyacente. El ejemplo siguiente crea un objeto DatabaseMetaData y lo utiliza para determinar el número máximo de caracteres permitidos para un nombre de tabla:

Ejemplo: crear un objeto DatabaseMetaData

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

Recuperar información general

Algunos métodos de DatabaseMetaData se emplean para buscar dinámicamente información general acerca de un origen de datos, y también para obtener detalles sobre su implementación. Algunos de estos métodos son:

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

Determinar el soporte de características

Hay un gran grupo de métodos de DatabaseMetaData que permiten determinar si una característica concreta (o un conjunto concreto de características) está soportada por el controlador o el origen de datos subyacente. Aparte de esto, existen métodos que describen el nivel de soporte que se proporciona. Algunos de los métodos que describen el soporte de características individuales son:

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

Entre los métodos que describen un nivel de soporte de característica se incluyen los siguientes:

- supportsANSI92EntryLevelSQL
- supportsCoreSQLGrammar

Límites de origen de datos

Otro grupo de métodos proporciona los límites impuestos por un determinado origen de datos. Algunos de los métodos de esta categoría son:

- getMaxRowSize
- getMaxStatementLength

- getMaxTablesInSelect
- getMaxConnections
- getMaxCharLiteralLength
- getMaxColumnsInTable

Los métodos de este grupo devuelven el valor de límite como un entero (integer). Si el valor de retorno es cero, indica que no hay ningún límite o que el límite es desconocido.

Objetos SQL y sus atributos

Diversos métodos de DatabaseMetaData proporcionan información sobre los objetos SQL que llenan un determinado origen de datos. Estos métodos pueden determinar los atributos de los objetos SQL. Estos métodos también devuelven objetos ResultSet, en los que cada fila describe un objeto concreto. Por ejemplo, el método getUDTs devuelve un objeto ResultSet en el que hay una fila para cada tabla definida por usuario (UDT) que se haya definido en el origen de datos. Son ejemplos de esta categoría:

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

Soporte de transacciones

Hay un pequeño grupo de métodos que proporcionan información sobre la semántica de transacción soportada por el origen de datos. Son ejemplos de esta categoría:

- supportsMultipleTransactions
- getDefaultTransactionIsolation

En: “Ejemplo: devolver una lista de tablas utilizando la interfaz DatabaseMetaData de using the IBM Developer Kit para Java” en la página 70 encontrará un ejemplo de cómo usar la interfaz DatabaseMetaData.

Cambios en JDBC 3.0

En JDBC 3.0 existen cambios en los valores de retorno de algunos de los métodos. Los siguientes métodos se han actualizado en JDBC 3.0 para añadir campos a los ResultSets que devuelven.

- getTables
- getColumns
- getUDTs
- getSchemas

Nota: Si está desarrollando una implementación mediante Java Development Kit (JDK) 1.4, tal vez observe que se devuelve un determinado número de columnas al efectuar la prueba. Usted escribe la aplicación y espera acceder a todas estas columnas. Sin embargo, si la aplicación se está diseñando para que también funcione en releases anteriores de JDK, esta recibe una SQLException cuando intenta acceder a estos campos, que no existen en releases anteriores de JDK. “Ejemplo: utilizar ResultSets de metadatos que tienen más de una columna” en la página 71 es un ejemplo de cómo se puede escribir una aplicación para que funcione con varios releases de JDK.

Cambios en JDBC 4.0

En V6R1, la interfaz de línea de mandatos (CLI) cambia la implementación de las API de MetaData para que también llamen a los procedimientos almacenados de SYSIBM. Por ello, los métodos de MetaData de JDBC emplearán los procedimientos de SYSIBM directamente en V6R1, sea cual sea el nivel del JDK. Debido a este cambio, verá las siguientes diferencias:

- Anteriormente, el controlador JDBC nativo permitía el usuario `localhost` como nombre de catálogo para la mayoría de los métodos. En JDBC 4.0, el controlador JDBC nativo no devolverá información si se especifica `localhost`.
- El controlador JDBC nativo siempre devolvía un conjunto de resultados vacío cuando el valor del parámetro nullable de getBestRowIdentifier era igual a false. Esto se corregirá para que se devuelva el debido resultado.
- Los valores devueltos por getColumns, para las columnas BUFFER_LENGTH, SQL_DATA_TYPE y SQL_DATETIME_SUB, pueden ser distintos. Estos valores no se deben usar en una aplicación JDBC porque la especificación JDBC define estas columnas como ″unused″ (no utilizadas).
- Anteriormente, el controlador JDBC nativo reconocía los parámetros de esquema y tabla de los métodos getCrossReference, getExportedKeys, getImportedKeys y getPrimaryKeys como ″patrón″. Ahora los parámetros de esquema y tabla deben coincidir con el nombre tal como está almacenado en la base de datos.
- Las vistas empleadas para implementar vistas definidas por el sistema se describían anteriormente mediante getTables() como SYSTEM TABLES. Para ser coherentes con la familia DB2, ahora estas vistas se describen como VIEWS.
- Los nombres de columnas devueltos por getProcedures son diferentes. Estos nombres de columnas no están definidos por la especificación JDBC 4.0. Asimismo, la columna de observaciones que getProcedures empleaba para devolver ″″ si no había información disponible, ahora devuelve null.

Tabla 3. Nombres de columna devueltos por getProcedures en JDBC 4.0

Nº de columna	Nombre anterior	Nombre bajo JDBC 4.0
4	RESERVED1	NUM_INPUT_PARAMS
5	RESERVED2	NUM_OUTPUT_PARAMS
6	RESERVED3	NUM_RESULT_SETS

- Algunos valores devueltos por getProcedureColumns para diversos tipos de datos han cambiado, como se ve a continuación:

Tabla 4. Valores devueltos por getProcedureColumns en JDBC 4.0

Tipo de datos	Columna	Valor anterior	Valor en JDBC 4.0
ALL	Observaciones	″″	null
INTEGER	Longitud	Null	4
SMALLINT	Longitud	Null	2
BIGINT	Tipo de datos	19 (incorrecto)	-5
BIGINT	Longitud	Null	8
DECIMAL	Longitud	Null	precisión + escala
NUMERIC	Longitud	Null	precisión + escala
DOUBLE	Nombre de tipo	DOUBLE PRECISION	DOUBLE
DOUBLE	Longitud	Null	8
FLOAT	Nombre de tipo	DOUBLE PRECISION	DOUBLE
FLOAT	Longitud	Null	8
REAL	Longitud	Null	4

Tabla 4. Valores devueltos por getProcedureColumns en JDBC 4.0 (continuación)

Tipo de datos	Columna	Valor anterior	Valor en JDBC 4.0
DATE	Precisión	null	10
DATE	Longitud	10	6
TIME	Precisión	null	8
TIME	Longitud	8	6
TIME	Escala	null	0
TIMESTAMP	Precisión	null	26
TIMESTAMP	Longitud	26	16
TIMESTAMP	Escala	null	6
CHAR	Nombre de tipo	CHARACTER	CHAR
CHAR	Precisión	null	igual que longitud
VARCHAR	Nombre de tipo	CHARACTER VARYING	VARCHAR
VARCHAR	Precisión	null	igual que longitud
CLOB	Tipo de datos	null (incorrecto)	2005
CLOB	Nombre de tipo	CHARACTER LARGE OBJECT	CLOB
CLOB	Precisión	null	igual que longitud
CHAR FOR BIT DATA	Tipo de datos	1 (CHAR)	-2 (BINARY)
CHAR FOR BIT DATA	Nombre de tipo	CHARACTER	CHAR () FOR BIT DATA
CHAR FOR BIT DATA	Precisión	null	igual que longitud
BLOB	Tipo de datos	null (incorrecto)	2004
BLOB	Nombre de tipo	BINARY LARGE OBJECT	BLOB
BLOB	Precisión	null	igual que longitud
DATALINK	Tipo de datos	null (incorrecto)	70
DATALINK	Precisión	null	igual que longitud
VARCHAR FOR BIT DATA	Tipo de datos	12 (VARCHAR)	-3 (VARBINARY)
VARCHAR FOR BIT DATA	Nombre de tipo	CHARACTER VARYING	VARCHAR () FOR BIT DATA
VARCHAR FOR BIT DATA	Precisión	null	igual que longitud

Restricción sobre los procedimientos almacenados solo de lectura (READ ONLY)

En JDBC nativo se puede usar la propiedad `access = read only`. Esta propiedad entra en vigor a nivel de JDBC. Por ello, los procedimientos de MetaData deben seguir funcionando si se establece esta propiedad. Sin embargo, es posible emplear el controlador JDBC nativo de un procedimiento almacenado de base de datos que esté definido como solo de lectura (READ ONLY). En este caso, los procedimientos de MetaData no funcionarán.

Método nuevo: getClientInfoProperties()

El método getClientInfoProperties recupera una lista de las propiedades informativas del cliente soportadas por el controlador. Cada propiedad informativa del cliente se almacena en un registro SQL especial. El controlador JDBC nativo devolverá un conjunto de resultados con la siguiente información:

Tabla 5. Información devuelta por el método getClientInfoProperties

Nombre	Longitud máxima	Valor predeterminado	Descripción
ApplicationName	255	En blanco	Nombre de la aplicación que utiliza actualmente la conexión
ClientUser	255	En blanco	Nombre del usuario para el que está realizando trabajo la aplicación que utiliza la conexión. Puede no ser el mismo nombre de usuario que el empleado al establecer la conexión.
ClientHostname	255	En blanco	Nombre de host de la máquina en la que se ejecuta aplicación que utiliza la conexión.
ClientAccounting	255	En blanco	Información de contabilidad.

Los registros SQL especiales correspondientes a las propiedades informativas del cliente son:

Tabla 6. Registros SQL especiales

Nombre	Registro SQL especial
ApplicationName	CURRENT CLIENT_APPLNAME
ClientUser	CURRENT CLIENT_USERID
ClientHostname	CURRENT CLIENT_WRKSTNNAME
ClientAccounting	CURRENT CLIENT_ACCTNG

Las clientInfoProperties se pueden establecer mediante el método setClientInfo del objeto Connection.

Conceptos relacionados

"ResultSets" en la página 114
La interfaz ResultSet proporciona acceso a los resultados generados al ejecutar consultas. Conceptualmente, los datos de un ResultSet pueden considerarse como una tabla con un número específico de columnas y un número específico de filas. Por omisión, las filas de la tabla se recuperan por orden. Dentro de una fila, se puede acceder a los valores de columna en cualquier orden.

Ejemplo: devolver una lista de tablas utilizando la interfaz DatabaseMetaData de using the IBM Developer Kit para Java:

Este ejemplo muestra cómo devolver una lista de tablas.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

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

// Obtener los metadatos de base de datos de la conexión.
DatabaseMetaData dbMeta = c.getMetaData();

// Obtener una lista de tablas que cumplen estos criterios.
String catalog = "myCatalog";
String schema = "mySchema";
String table  = "myTable%"; // % indica el patrón de búsqueda
```

```
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"}:
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterar por ResultSet para obtener los valores.

// Cerrar la conexión.
c.close():
```

Ejemplo: utilizar ResultSets de metadatos que tienen más de una columna:

Este es un ejemplo de utilización de ResultSets de metadatos que tienen más una columna.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de SafeGetUDTs. Este programa muestra una forma de tratar con
// ResultSets de metadatos que tienen más columnas en JDK 1.4 que en
// releases anteriores.
//
// Sintaxis de mandato:
//    java SafeGetUDTs
//
//////////////////////////////////////////////////////////////////////////////
//
// Este código fuente es un ejemplo del controlador JDBC de IBM Developer para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo con fines ilustrativos
// solamente. Los ejemplos no se han probado minuciosamente bajo todas
// las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas que contiene este ejemplo se suministran "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se declinan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// 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: El bloque estático se ejecuta antes de que se inicie main.
    // Por tanto, existe acceso a jdbcLevel en
    // main.
    {
        try {
            Class.forName("java.sql.Blob");

            try {
                Class.forName("java.sql.ParameterMetaData");
                // Encontrada una interfaz JDBC 3.0. Debe soportar JDBC 3.0.
```

```
                jdbcLevel = 3;
            } catch (ClassNotFoundException ez) {
                // No se ha encontrado la clase ParameterMetaData de JDBC 3.0.
                // Debe estar en ejecución bajo una JVM solo con soporte
                // JDBC 2.0.
                jdbcLevel = 2;
            }

        } catch (ClassNotFoundException ex) {
            // No se ha encontrado la clase Blob de JDBC 2.0. Debe estar
            // en ejecución bajo una JVM solo con soporte para JDBC 1.0.
            jdbcLevel = 1;
        }
    }

    // Punto de entrada del programa.
    public static void main(java.lang.String[] args)
    {
        Connection c = null;

        try {
            // Obtener el controlador registrado.
            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 hay soporte para getUDTs. Solo retornar.");
                System.exit(1);
            }

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

                // Captar todas las columnas que han estado disponibles desde
                // el release JDBC 2.0.
                System.out.println("TYPE_CAT es " + rs.getString("TYPE_CAT"));
                System.out.println("TYPE_SCHEM es " + rs.getString("TYPE_SCHEM"));
                System.out.println("TYPE_NAME es " + rs.getString("TYPE_NAME"));
                System.out.println("CLASS_NAME es " + rs.getString("CLASS_NAME"));
                System.out.println("DATA_TYPE es " + rs.getString("DATA_TYPE"));
                System.out.println("REMARKS es " + rs.getString("REMARKS"));

                // Captar todas las columnas que se han añadido en JDBC 3.0.
                if (jdbcLevel > 2) {
                    System.out.println("BASE_TYPE es " + rs.getString("BASE_TYPE"));
                }
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            if (c != null) {
                try {
                    c.close();
                } catch (SQLException e) {
                    // Ignorar excepción de cierre.
                }
            }
        }
    }
}
```

Excepciones Java

El lenguaje Java utiliza excepciones para proporcionar posibilidades de manejo de errores para sus programas. Una excepción es un evento que se produce cuando se ejecuta el programa de forma que interrumpe el flujo normal de instrucciones.

El sistema de ejecución Java y muchas clases de paquetes Java lanzan excepciones en algunas circunstancias utilizando la sentencia throw. Puede utilizar el mismo mecanismo para lanzar excepciones en los programas Java.

Clase Java SQLException:

La clase SQLException y sus subtipos proporcionan información acerca de los errores y avisos que se producen mientras se está accediendo a un origen de datos.

A diferencia de la mayor parte de JDBC, que se define mediante interfaces, el soporte de excepciones se suministra en clases. La clase básica para las excepciones que se producen durante la ejecución de aplicaciones JDBC es SQLException. Todos los métodos de la API JDBC se declaran capaces de lanzar SQLExceptions. SQLException es una ampliación de java.lang.Exception y proporciona información adicional relacionada con las anomalías que se producen en un contexto de base de datos. Específicamente, en una SQLException está disponible la siguiente información:

- Texto descriptivo
- SQLState
- Código de error
- Una referencia a las demás excepciones que también se han producido

ExceptionExample es un programa que maneja adecuadamente la captura de una SQLException (esperada, en este caso), y el vuelco de toda la información que proporciona.

Nota: JDBC proporciona un mecanismo para encadenar las excepciones. Esto permite al controlador o a la base de datos informar de varios errores en una sola petición. Actualmente, no existen instancias en las que el controlador JDBC nativo realice esta acción. Sin embargo, esta información solo se proporciona como referencia y no como indicación clara de que el controlador nunca realizará esta acción en el futuro.

Como se ha indicado, los objetos SQLException se lanzan cuando se producen errores. Esta afirmación es correcta, pero es incompleta. En la práctica, el controlador JDBC nativo rara vez lanza SQLExceptions reales. Lanza instancias de sus propias subclases SQLException. Esto permite al usuario determinar más información acerca de lo que realmente ha fallado, como se indica a continuación.

DB2Exception.java

Los objetos DB2Exception no se lanzan directamente. Esta clase básica se utiliza para contener las funciones que son comunes a todas las excepciones JDBC. Existen dos subclases de esta clase que son las excepciones estándar que lanza JDBC. Estas subclases son DB2DBException.java y DB2JDBCException.java. Las DB2DBExceptions son excepciones de las que se informa al usuario que provienen directamente de la base de datos. Las DB2JDBCExceptions se lanzan cuando el controlador JDBC detecta problemas por su cuenta. Dividir la jerarquía de clases de excepción de esta forma permite manejar los dos tipos de excepciones de forma distinta.

DB2DBException.java

Como se ha indicado, las DB2DBExceptions son excepciones que provienen directamente de la base de datos. Se detectan cuando el controlador JDBC efectúa una llamada a CLI y obtiene un código de retorno SQLERROR. En estos casos, se llama al SQLError de la función CLI para obtener el texto del mensaje, SQLState, y el código del proveedor. También se recupera y se devuelve al usuario el texto de sustitución

del SQLMessage. La clase DatabaseException provoca un error que la base de datos reconoce e indica al controlador JDBC que cree el objeto de excepción para el mismo.

DB2JDBCException.java

Las DB2JDBCExceptions se generan para condiciones de error que provienen del propio controlador JDBC. El funcionamiento de esta clase de excepciones es fundamentalmente diferente; el propio controlador JDBC maneja la conversión de lenguaje del mensaje de la excepción y otras cuestiones que el sistema operativo y la base de datos manejan en el caso de las excepciones que se originan en la base de datos. Siempre que es posible, el controlador JDBC se ajusta a los SQLStates de la base de datos. El código de proveedor para las excepciones que lanza el controlador JDBC es siempre -99999. Las DB2DBExceptions reconocidas y devueltas por la capa CLI también tienen con frecuencia el código de error -99999. La clase JDBCException provoca un error que el controlador JDBC reconoce y crea la excepción por su cuenta. Durante el desarrollo de este release, se ha creado la siguiente salida. Observe que, al principio de la pila, se encuentra DB2JDBCException. Esto indica que se está informando del error desde el controlador JDBC antes de efectuar la petición a la base de datos.

Ejemplo: SQLException:

Este es un ejemplo de cómo capturar una SQLException y volcar toda la información que proporciona.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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("No se esperaba que la tabla existiera.");


        } catch (SQLException e) {
            System.out.println("Excepción SQLException: ");
            System.out.println("Mensaje:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Código proveedor:." + e.getErrorCode());
            System.out.println("-----------------------------------------------------");
            e.printStackTrace();
        } catch (Exception ex) {
            System.out.println("Se ha lanzado una excepción que no es una SQLException: ");
            ex.printStackTrace();
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                System.out.println("Excepción capturada al intentar concluir...");
            }
        }
    }
}
```

SQLWarning:

Los métodos de algunas interfaces generan un objeto SQLWarning si provocan un aviso de acceso a base de datos.

Los métodos de las siguientes interfaces pueden generar un SQLWarning:

- Connection
- Statement y sus subtipos, PreparedStatement y CallableStatement
- ResultSet

Cuando un método genera un objeto SQLWarning, no se informa al llamador de que se ha producido un aviso de acceso a base de datos. Debe llamarse al método getWarnings en el objeto adecuado para recuperar el objeto SQLWarning. Sin embargo, en algunas circunstancias puede que se lance la subclase DataTruncation de SQLWarning. Debe tenerse en cuenta que el controlador JDBC nativo opta por pasar por alto algunos avisos generados por la base de datos para aumentar la eficacia. Por ejemplo, el sistema genera un aviso cuando el usuario intenta recuperar datos más allá del final de un ResultSet mediante el método ResultSet.next. En este caso, el método next se define para devolver false en lugar de true, informando al usuario del error. Sería innecesario crear un objeto que avisara de nuevo, por lo que el aviso se pasa simplemente por alto.

Si se producen múltiples avisos de acceso a datos, los nuevos avisos se encadenan al primero y, para recuperarlos, se llama al método SQLWarning.getNextWarning. Si no hay más avisos en la cadena, el método getNextWarning devuelve `null`.

Los objetos SQLWarning subsiguientes se siguen añadiendo a la cadena hasta que se procesa la próxima sentencia o, en el caso de un objeto ResultSet, cuando vuelve a situarse el cursor. Como resultado, se eliminan todos los objetos SQLWarning de la cadena.

La utilización de objetos Connection, Statement y ResultSet puede provocar la generación de SQLWarnings. Los SQLWarnings son mensajes informativos que indican que, aunque una operación determinada se ha realizado satisfactoriamente, puede haber otra información sobre la que el usuario debe estar al corriente. Los SQLWarnings son una ampliación de la clase SQLException, pero no se lanzan. En lugar de ello, se conectan al objeto que provoca su generación. Cuando se genera un SQLWarning, no ocurre nada que indique a la aplicación que se ha generado el aviso. Las aplicaciones deben solicitar activamente la información de aviso.

Al igual que las SQLExceptions, los SQLWarnings pueden encadenarse entre sí. Puede llamar al método clearWarnings en un objeto Connection, Statement o ResultSet para borrar los avisos correspondientes a ese objeto.

Nota: Al llamar al método clearWarnings no se borran todos los avisos. Solo se borran los avisos asociados con un objeto determinado.

El controlador JDBC borra los objetos SQLWarning en momentos específicos si el usuario no los borra manualmente. Los objetos SQLWarning se borran cuando se realizan las siguientes acciones:

- En la interfaz Connection, los avisos se borran durante la creación de un objeto Statement, PreparedStatement o CallableStatement nuevo.
- En la interfaz Statement, los avisos se borran cuando se procesa la próxima sentencia (o cuando se procesa de nuevo la sentencia para PreparedStatements y CallableStatements).
- En la interfaz ResultSet, los avisos se borran cuando vuelve a situarse el cursor.

DataTruncation y truncamiento de silencioso:

DataTruncation es una subclase de SQLWarning. Aunque no se lanzan SQLWarnings, a veces se lanzan objetos DataTruncation y se conectan al igual que otros objetos SQLWarning. Un truncamiento silencioso

ocurre cuando el tamaño de una columna excede el tamaño especificado por el método de sentencia Silent setMaxFieldSize, pero no se produce ningún aviso o excepción.

Los objetos DataTruncation proporcionan información adicional más allá de lo que devuelve un SQLWarning. La información disponible es la siguiente:

- El número de bytes de datos que se han transferido.
- El índice de columna o parámetro que se ha truncado.
- Si el índice es para un parámetro o para una columna de ResultSet.
- Si el truncamiento se ha producido al leer en la base de datos o al escribir en ella.
- La cantidad de datos que se han transferido realmente.

En algunos casos, la información puede descifrarse, pero se producen situaciones que no son completamente intuitivas. Por ejemplo, si se utiliza el método setFloat de PreparedStatement para insertar un valor en una columna que contiene valores enteros, puede producirse una DataTruncation debido a que float puede ser mayor que el valor mayor que la columna puede contener. En estas situaciones, las cuentas de bytes para el truncamiento no tienen sentido, pero es importante para el controlador proporcionar la información de truncamiento.

Informar de los métodos set() y update()

Existe una ligera diferencia entre los controladores JDBC. Algunos controladores, como por ejemplo los controladores JDBC nativos y de IBM Toolbox para Java capturan e informan de las situaciones de truncamiento de datos en el momento de establecer el parámetro. Esta acción se realiza en el método set de PreparedStatement o en el método update de ResultSet. Otros controladores informan del problema en el momento de procesar la sentencia, y se realiza mediante los métodos execute, executeQuery o updateRow.

La notificación del problema en el momento de proporcionar datos incorrectos, en lugar de hacerlo en el momento en que el proceso no puede seguir adelante, ofrece las siguientes ventajas:

- La anomalía puede dirigirse a la aplicación cuando se produce un problema en lugar de dirigir el problema en el momento del proceso.
- Efectuando la comprobación al establecer los parámetros, el controlador JDBC puede asegurarse de que los valores que se pasan a la base de datos en el momento de procesar la sentencia son válidos. Esto permite a la base de datos optimizar su trabajo y el proceso puede realizarse con mayor rapidez.

Métodos ResultSet.update() que lanzan excepciones DataTruncation

En algunos releases anteriores, los métodos ResultSet.update() enviaban avisos cuando existían condiciones de truncamiento. Este caso se produce cuando el valor de datos va a insertarse en la base de datos. La especificación indica que los controladores JDBC deben lanzar excepciones en estos casos. Como resultado, el controlador JDBC funciona de esta forma.

No hay ninguna diferencia significativa entre manejar una función de actualización de ResultSet que recibe un error de truncamiento de datos y manejar un parámetro de sentencia preparada establecido para una sentencia update o insert que recibe un error. En ambos casos, el problema es idéntico; el usuario ha proporcionado datos que no caben donde se desea.

Los datos de tipo NUMERIC y DECIMAL se truncan a la derecha de una coma decimal de forma silenciosa. Así es como funciona JDBC para UDB NT y también SQL interactivo en una plataforma System i.

Nota: No se redondea ningún valor cuando se produce un truncamiento de datos. Cualquier parte fraccionaria de un parámetro que no quepa en una columna NUMERIC o DECIMAL se pierde sin aviso.

A continuación se ofrecen ejemplos, suponiendo que el valor de la cláusula values es realmente un parámetro que se establece en una sentencia preparada:

```
create table cujosql.test (col1 numeric(4,2))
a) insert into cujosql.test values(22.22)  // funciona - inserta 22.22
b) insert into cujosql.test values(22.223) // funciona - inserta 22.22
c) insert into cujosql.test values(22.227) // funciona - inserta 22.22
d) insert into cujosql.test values(322.22) // falla - Error de conversión en la asignación a columna COL1.
```

Diferencia entre un aviso de truncamiento de datos y una excepción de truncamiento de datos

La especificación indica que el truncamiento de datos en un valor que debe escribirse en la base de datos debe lanzar una excepción. Si el truncamiento de datos no se realiza en el valor que se escribe en la base de datos, se genera un aviso. Esto significa que, en el momento en que se identifica una situación de truncamiento de datos, el usuario también debe tener conocimiento del tipo de sentencia que el truncamiento de datos está procesando. Dado este requisito, a continuación figura una lista del comportamiento de varios tipos de sentencias SQL:

- En una sentencia SELECT, los parámetros de consulta nunca dañan el contenido de la base de datos. Por tanto, las situaciones de truncamiento de datos se manejan siempre enviando avisos.
- En las sentencias VALUES INTO y SET, los valores de entrada solo se emplean para generar valores de salida. En consecuencia, se emiten avisos.
- En una sentencia CALL, el controlador JDBC no puede determinar lo que un procedimiento almacenado hace con un parámetro. Se lanzan siempre excepciones cuando se trunca un parámetro de procedimiento almacenado.
- Todos los demás tipos de sentencias lanzan excepciones en lugar de enviar avisos.

Propiedad de truncamiento de datos para Connection y DataSource

Durante muchos releases ha existido una propiedad de truncamiento de datos disponible. El valor predeterminado de esa propiedad es true, que indica que se comprueban los problemas de truncamiento de datos y se envían avisos o se lanzan excepciones. La propiedad se suministra a efectos de conveniencia y rendimiento en los casos en que el hecho de que un valor no quepa en la columna de la base de datos no es preocupante. El usuario desea que el controlador coloque en la columna la mayor parte posible del valor.

La propiedad de truncamiento de datos solo afecta a tipos de datos de carácter y basados en binario

Hasta hace dos releases, la propiedad de truncamiento de datos determinaba si podían lanzarse excepciones de truncamiento de datos. La propiedad de truncamiento de datos se introdujo para que las aplicaciones JDBC no tuvieran que preocuparse de si se truncaba un valor si el truncamiento no era importante. Existen unos pocos casos en los que deseará almacenar el valor 00 o 10 en la base de datos cuando las aplicaciones intenten insertar 100 en un DECIMAL(2,0). Por tanto, la propiedad de truncamiento de datos del controlador JDBC se ha cambiado para prestar atención solo a las situaciones en las que el parámetro es para tipos basados en caracteres, como por ejemplo CHAR, VARCHAR, CHAR FOR BIT DATA y VARCHAR FOR BIT DATA.

La propiedad de truncamiento de datos solo se aplica a parámetros

La propiedad de truncamiento de datos es un valor del controlador JDBC y no de la base de datos. En consecuencia, no tiene ningún efecto sobre los literales de sentencia. Por ejemplo, las sentencias siguientes que se procesan para insertar un valor en una columna CHAR(8) de la base de datos siguen fallando con el identificador de truncamiento de datos establecido en false (suponiendo que la conexión sea un objeto java.sql.Connection asignado a cualquier otro lugar).

```
Statement stmt = connection.createStatement();
Stmt.executeUpdate("create table cujosql.test (col1 char(8))");
Stmt.executeUpdate("insert into cujosql.test values('dettinger')");
// Falla debido a que el valor no cabe en la columna de la base de datos.
```

El controlador JDBC nativo lanza excepciones por truncamientos de datos insignificantes

El controlador JDBC nativo no observa los datos que el usuario proporciona para los parámetros. Al hacerlo, solo ralentizaría el proceso. Sin embargo, pueden darse situaciones en las que no importa que un valor se trunque, pero no se ha establecido la propiedad de conexión de truncamiento de datos en false.

Por ejemplo, 'dettinger ', un valor char(10) que se pasa, lanza una excepción aunque quepan todas las partes importantes del valor. Este es el funcionamiento de JDBC para UDB NT; sin embargo, no es el comportamiento que obtendría si pasara el valor como un literal de una sentencia SQL. En este caso, el motor de base de datos eliminaría los espacios adicionales de forma silenciosa.

Los problemas con el controlador JDBC que no lanza una excepción son los siguientes:

- La actividad general de rendimiento es extensiva en todos los métodos set, sea o no necesaria. En la mayoría de los casos en los que no representa ninguna ventaja, se produce una actividad general de rendimiento considerable en una función tan común como setString().
- La solución alternativa es sencilla, por ejemplo, llamar a la función de ajuste (trim) en el valor de serie que se pasa.
- Existen situaciones en la columna de la base de datos que deben tenerse en cuenta. Un espacio del CCSID 37 no es en absoluto un espacio del CCSID 65535 o 13488.

Truncamiento silencioso

El método de sentencia setMaxFieldSize permite especificar un tamaño máximo de campo para cualquier columna. Si los datos se truncan debido a que el tamaño ha sobrepasado el valor de tamaño máximo de campo, no se informa de ningún aviso ni excepción. Este método, al igual que la propiedad de truncamiento de datos mencionada anteriormente, solo afecta a tipos basados en caracteres, como por ejemplo CHAR, VARCHAR, CHAR FOR BIT DATA y VARCHAR FOR BIT DATA.

Transacciones JDBC

La transacción es una unidad de trabajo lógica. Para realizar una unidad de trabajo lógica, puede ser necesario llevar a cabo varias acciones con respecto a una base de datos.

El soporte transaccional permite a las aplicaciones asegurarse de que:

- Se siguen todos los pasos para realizar una unidad de trabajo lógica.
- Cuando falla uno de los pasos de los archivos de la unidad de trabajo, todo el trabajo realizado como parte de esa unidad de trabajo lógica puede deshacerse y la base de datos puede volver a su estado anterior al inicio de la transacción.

Las transacciones se utilizan para proporcionar integridad de los datos, una semántica correcta de la aplicación y una vista coherente de los datos durante el acceso concurrente. Todos los controladores que estén en conformidad con Java Database Connectivity (JDBC) deben poder utilizar transacciones.

Nota: Esta sección solo describe las transacciones locales y el concepto de JDBC estándar con respecto a las transacciones. Java y el controlador JDBC nativo soportan la API de transacciones Java (JTA), las transacciones distribuidas y el protocolo de compromiso de dos fases (2PC).

Todo el trabajo transaccional se maneja a nivel de objetos Connection. Cuando finaliza el trabajo de una transacción, esta puede finalizarse llamando al método commit. Si la aplicación termina anormalmente la transacción, se llama al método rollback.

Todos los objetos Statement de una conexión forman parte de la transacción. Esto significa que, si una aplicación crea tres objetos Statement y utiliza cada uno de los objetos para efectuar cambios en la base de datos, cuando se produzca una llamada a commit o rollback, el trabajo de las tres sentencias se convierte en permanente o se descarta.

Las sentencias SQL commit y rollback se utilizan para finalizar transacciones al trabajar solo con SQL. Estas sentencias SQL no se pueden preparar dinámicamente, y no hay que tratar de utilizarlas en las aplicaciones JDBC para completar transacciones.

Modalidad JDBC dw compromiso automático:

Por defecto, JDBC utiliza una modalidad de operación denominada compromiso automático. Esto significa que todas las actualizaciones de la base de datos se convierten inmediatamente en permanentes.

Cualquier situación en la que una unidad lógica de trabajo requiera más de una actualización de la base de datos no puede gestionarse de forma segura en modalidad de compromiso automático. Si se produce alguna anomalía en la aplicación o en el sistema después de realizar una actualización y antes de que se realicen otras actualizaciones, el primer cambio no puede deshacerse en la ejecución por modalidad de compromiso automático.

Dado que en la modalidad de compromiso automático los cambios se convierten inmediatamente en permanentes, no es necesario que la aplicación llame al método commit ni al método rollback. Esto facilita la escritura de las aplicaciones.

La modalidad de compromiso automático puede habilitarse e inhabilitarse dinámicamente durante la existencia de una conexión. El compromiso automático se habilita de la siguiente forma, suponiendo que el origen de datos ya exista:

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

Connection.setAutoCommit(false);        // Inhabilita el compromiso automático.
```

Si se cambia el valor de compromiso automático en medio de una transacción, el trabajo pendiente se compromete automáticamente. Se genera una SQLException si se habilita el compromiso automático para una conexión que forma parte de una transacción distribuida.

Niveles de aislamiento de las transacciones:

Los niveles de aislamiento de las transacciones especifican qué datos son visibles para las sentencias dentro de una transacción. Estos niveles influyen directamente sobre el nivel de acceso concurrente al definir qué interacción es posible entre las transacciones en el mismo origen de datos destino.

Anomalías de base de datos

Las anomalías de base de datos son resultados generados que parecen incorrectos cuando se observan desde el ámbito de una sola transacción, pero que son correctos cuando se observan desde el ámbito de todas las transacciones. A continuación se describen los diversos tipos de anomalías de base de datos:

- Se producen lecturas **sucias** cuando:
  1. La transacción A inserta una fila en una tabla.
  2. La transacción B lee la fila nueva.
  3. La transacción A efectúa una retrotracción.

  La transacción B puede haber realizado trabajo en el sistema basándose en la fila insertada por la transacción A, pero la fila nunca ha pasado a formar parte permanente de la base de datos.
- Se producen lecturas **no repetibles** cuando:
  1. La transacción A lee una fila.
  2. La transacción B cambia la fila.
  3. La transacción A lee la misma fila por segunda vez y obtiene los resultados nuevos.
- Se producen lecturas **fantasma** cuando:
  1. La transacción A lee todas las filas que satisfacen una cláusula WHERE de una consulta SQL.

2. La transacción B inserta una fila adicional que satisface la cláusula WHERE.
3. La transacción A vuelve a evaluar la condición WHERE y recoge la fila adicional.

Nota: DB2 para i5/OS no siempre expone la aplicación a las anomalías de base de datos permitidas en los niveles prescritos debido a sus estrategias de bloqueo.

Niveles de aislamiento de las transacciones JDBC

Existen cinco niveles de aislamiento de transacciones en la API JDBC de IBM Developer Kit para Java. A continuación figuran los niveles ordenados del menos restrictivo al más restrictivo:

JDBC_TRANSACTION_NONE
Esta es una constante especial que indica que el controlador JDBC no da soporte a las transacciones.

JDBC_TRANSACTION_READ_UNCOMMITTED
Este nivel permite que las transacciones vean los cambios no comprometidos en los datos. En este nivel son posibles todas las anomalías de base de datos.

JDBC_TRANSACTION_READ_COMMITTED
Este nivel indica que los cambios efectuados dentro de una transacción no son visibles fuera de ella hasta que se compromete la transacción. Esto impide que las lecturas sucias sean posibles.

JDBC_TRANSACTION_REPEATABLE_READ
Este nivel indica que las filas que se leen retienen bloqueos para que otra transacción no pueda cambiarlas si la transacción no ha finalizado. Esto no permite las lecturas sucias y las lecturas no repetibles. Las lecturas fantasma siguen siendo posibles.

JDBC_TRANSACTION_SERIALIZABLE
Se bloquean las tablas de la transacción para que otras transacciones que añaden valores a la tabla o los eliminan de la misma no puedan cambiar condiciones WHERE. Esto impide todos los tipos de anomalías de base de datos.

Puede utilizar el método setTransactionIsolation para cambiar el nivel de aislamiento de las transacciones para una conexión.

Consideraciones

Una malinterpretación común es que la especificación JDBC define los cinco niveles transaccionales mencionados anteriormente. Se cree generalmente que el valor TRANSACTION_NONE representa el concepto de ejecución sin control de compromiso. La especificación JDBC no define TRANSACTION_NONE de la misma forma. TRANSACTION_NONE se define en la especificación JDBC como un nivel en el que el controlador no soporta las transacciones y que no es un controlador compatible con JDBC. El nivel NONE nunca se indica cuando se llama al método getTransactionIsolation.

La cuestión se complica tangencialmente por el hecho de que el nivel de aislamiento de transacciones por omisión del controlador JDBC queda definido por la implementación. El nivel por omisión de aislamiento de transacciones para el controlador JDBC nativo es NONE. Esto permite que el controlador trabaje con archivos que no tienen diarios, y no es necesario que el usuario realice especificaciones, como por ejemplo los archivos de la biblioteca QGPL.

El controlador JDBC nativo permite pasar JDBC_TRANSACTION_NONE al método setTransactionIsolation o especificar none como propiedad de conexión. Sin embargo, el método getTransactionIsolation siempre indica JDBC_TRANSACTION_READ_UNCOMMITTED cuando el valor es none. Es responsabilidad de la aplicación efectuar el seguimiento del nivel que se está ejecutando, si la aplicación lo necesita.

En releases anteriores, el controlador JDBC manejaba la especificación de true por parte del usuario para el compromiso automático cambiando el nivel de aislamiento de las transacciones a none, debido a que el sistema no tenía el concepto de modalidad de compromiso automático true. Esta era una aproximación bastante exacta a la funcionalidad, pero no proporcionaba los resultados correctos en todos los escenarios. Esto ya no se realiza; la base de datos desasocia el concepto de compromiso automático del concepto de nivel de aislamiento de las transacciones. Por tanto, es completamente válida la ejecución al nivel JDBC_TRANSACTION_SERIALIZABLE con el compromiso automático habilitado. El único escenario que no es válido es la ejecución al nivel JDBC_TRANSACTION_NONE sin modalidad de compromiso automático. La aplicación no puede tomar el control sobre los límites del compromiso si el sistema no se está ejecutando con un nivel de aislamiento de transacción.

Niveles de aislamiento de transacciones entre la especificación JDBC y la plataforma System i

La plataforma System i tiene nombres comunes para sus niveles de aislamiento de transacciones que no coinciden con los nombres que proporciona la especificación JDBC. En la tabla que sigue se proporciona una correspondencia con los nombres utilizados por la plataforma System i, pero no son equivalentes a los utilizados por la especificación JDBC.

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

* En esta tabla, por cuestión de claridad, el valor JDBC_TRANSACTION_NONE se hace corresponder con los niveles *NONE y *NC de System i. Esta no es una correspondencia directa entre especificación y nivel de System i.

Puntos de salvar:

Los puntos de salvar permiten establecer ″puntos intermedios″ en una transacción. Los puntos de salvar son puntos de comprobación a los que la aplicación puede retrotraerse sin eliminar toda la transacción.

Los puntos de salvar son una novedad de JDBC 3.0, lo que significa que, para poder utilizarlos, la aplicación se debe ejecutar en Java Development Kit (JDK) 1.4 o posterior. Además, los puntos de salvar son una novedad en Developer Kit para Java, es decir, que no están soportados si JDK 1.4 o superior no se utiliza con releases anteriores de Developer Kit para Java.

Nota: El sistema suministra sentencias SQL para trabajar con puntos de salvar. No es aconsejable que las aplicaciones JDBC utilicen estas sentencias directamente en una aplicación. Puede funcionar, pero el controlador JDBC pierde su capacidad para efectuar el seguimiento de los puntos de salvar cuando se realiza esta operación. Como mínimo, debe evitarse mezclar los dos modelos (es decir, utilizar sentencias SQL propias de puntos de salvar y utilizar la API JDBC).

Establecer puntos de salvar y retrotraerse a ellos

Pueden establecerse puntos de salvar a lo largo del trabajo de una transacción. A continuación, la aplicación puede retrotraerse a cualquiera de estos puntos de salvar si se produce algún error y continuar el proceso a partir de ese punto. En el ejemplo siguiente, la aplicación inserta el valor FIRST en una tabla de base de datos. Después de eso, se establece un punto de salvar y se inserta otro valor, SECOND, en la base de datos. Se emite una retrotracción al punto de salvar y se deshace el trabajo de insertar SECOND, pero se conserva FIRST como parte de la transacción pendiente. Finalmente, se inserta el valor THIRD y se compromete la transacción. La tabla de base de datos contiene los valores FIRST y THIRD.

Ejemplo: establecer puntos de salvar y retrotraerse a ellos

```
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);        // Deshace la inserción más reciente.
s.executeUpdate("insert into table1 values ('THIRD')");
connection.commit();
```

Aunque es poco probable que se produzcan problemas en los puntos de salvar establecidos si la modalidad es de compromiso automático, no pueden retrotraerse cuando sus vidas finalizan al final de la transacción.

Liberar un punto de salvar

La aplicación puede liberar puntos de salvar con el método releaseSavepoint del objeto Connection. Una vez liberado un punto de salvar, si se intenta retrotraer al mismo, se produce una excepción. Cuando una transacción se compromete o retrotrae, todos los puntos de salvar se liberan automáticamente. Cuando se retrotrae un punto de salvar, también se liberan los demás puntos de salvar que le siguen.

Transacciones JDBC distribuidas

Generalmente, en Java Database Connectivity (JDBC) las transacciones son locales. Esto significa que una sola conexión realiza todo el trabajo de la transacción y que la conexión solo puede trabajar en una transacción a la vez.

Cuando todo el trabajo para esa transacción ha finalizado o ha fallado, se llama al compromiso o a la retrotracción para convertir el trabajo en permanente y puede empezar una nueva transacción. Existe un soporte avanzado para transacciones disponible en Java que proporciona funciones que van más allá de las transacciones locales. Este soporte se especifica plenamente por la API de transacciones Java (JTA).

La API de transacciones Java (JTA) tiene soporte para transacciones complejas. También proporciona soporte para desasociar transacciones de objetos Connection. Al igual que JDBC se ha diseñado a partir de las especificaciones Object Database Connectivity (ODBC) y X/Open Call Level Interface (CLI), JTA se ha diseñado a partir de la especificación X/Open Extended Architecture (XA). JTA y JDBC funcionan conjuntamente para desasociar transacciones a partir de objetos Connection. El hecho de desasociar transacciones de objetos Connection permite que una sola conexión trabaje en varias transacciones simultáneamente. A la inversa, permite que varias conexiones trabajen en una sola transacción.

Nota: Si piensa trabajar con JTA, consulte el tema Iniciación a JDBC, donde hallará más información sobre los archivos Java de archivado (JAR) necesarios en la vía de acceso de clases de extensiones. Desea utilizar tanto el paquete opcional de JDBC 2.0 como los archivos JAR de JTA (JDK encuentra automáticamente estos archivos si ejecuta JDK 1.4 o otra versión posterior). No se encuentran por omisión.

Transacciones con JTA

Cuando JTA y JDBC se utilizan conjuntamente, existen una serie de pasos entre ellos para realizar el trabajo transaccional. Se proporciona soporte para XA mediante la clase XADataSource. Esta clase contiene soporte para configurar la agrupación de conexiones exactamente de la misma forma que su superclase ConnectionPoolDataSource.

Con una instancia de XADataSource, puede recuperar un objeto XAConnection. El objeto XAConnection funciona como contenedor tanto para el objeto JDBC Connection como para un objeto XAResource. El objeto XAResource está diseñado para manejar el soporte transaccional XA. XAResource maneja las transacciones mediante objetos denominados ID de transacción (XID).

XID es una interfaz que debe implementar. Representa una correlación Java de la estructura XID del identificador de transacción X/Open. Este objeto contiene tres partes:

- Un ID formato de transacción global
- Una transacción global
- Un calificador de rama

Consulte la especificación JTA para obtener detalles completos acerca de esta interfaz.

Utilizar el soporte de UDBXADataSource para agrupación y transacciones distribuidas

El soporte de API de transacciones Java (JTA) proporciona soporte directo para la agrupación de conexiones. UDBXADataSource es una ampliación de ConnectionPoolDataSource, que permite el acceso de las aplicaciones a objetos XAConnection agrupados. Puesto que UDBXADataSource es un ConnectionPoolDataSource, la configuración y utilización del UDBXADataSource es la misma que la descrita en el tema Utilizar el soporte de DataSource para agrupación de objetos.

Propiedades de XADataSource

Además de las propiedades que proporciona ConnectionPoolDataSource, la interfaz XADataSource proporciona las siguientes propiedades:

Método set (tipo de datos)	Valores	Descripción
setLockTimeout (int)	0 o cualquier valor positivo	Cualquier valor positivo es un tiempo de espera de bloqueo válido (en segundos) a nivel de transacción. Un tiempo de espera de bloqueo 0 significa que no se impone ningún valor de tiempo de espera de bloqueo a nivel de transacción, aunque puede imponerse uno a otros niveles (el trabajo o la tabla). El valor predeterminado es 0.
setTransactionTimeout (int)	0 o cualquier valor positivo	Cualquier valor positivo es un tiempo de espera de transacción válido (en segundos). Un tiempo de espera de transacción 0 significa que no se impone ningún tiempo de espera de transacción. Si la transacción permanece activa durante más tiempo del que indica el valor de tiempo de espera, se marca como solo de retrotracción y los intentos subsiguientes de realizar trabajo en ella provocan una excepción. El valor predeterminado es 0.

ResultSets y transacciones

Además de demarcar el inicio y la finalización de una transacción, como se ha mostrado en el ejemplo anterior, las transacciones pueden suspenderse durante un tiempo y reanudarse más tarde. Esto proporciona diversos escenarios para los recursos de ResultSet creados durante una transacción.

Fin de transacción simple

Al finalizar una transacción, todos los ResultSets abiertos que se crearon bajo esa transacción se cierran automáticamente. Es aconsejable cerrar explícitamente los ResultSets cuando haya terminado de utilizarlos para garantizar el máximo de proceso paralelo. Sin embargo, se produce una excepción si se accede a cualquier ResultSet que estaba abierto durante una transacción después de efectuar la llamada a XAResource.end.

Suspender y reanudar

Mientras una transacción está suspendida, el acceso a un ResultSet creado mientras la transacción estaba activa no está permitido y provoca una excepción. Sin embargo, una vez que la transacción se ha reanudado, el ResultSet está disponible de nuevo y permanece en el mismo estado en que estaba antes de que se suspendiera la transacción.

Efectuar ResultSets suspendidos

Mientras una transacción está suspendida, no puede accederse al ResultSet. Sin embargo, los objetos Statement pueden reprocesarse bajo otra transacción para realizar el trabajo. Debido a que los objetos JDBC Statement solo pueden tener un ResultSet a la vez (excluyendo el soporte de JDBC 3.0 para varios ResultSets simultáneos de una llamada de procedimiento almacenado), el ResultSet de la transacción suspendida debe cerrarse para satisfacer la petición de la nueva transacción. Esto es exactamente lo que ocurre.

Nota: Aunque JDBC 3.0 permite que un objeto Statement tenga varios ResultSets abiertos simultáneamente para una llamada de procedimiento almacenado, estos se tratan como una sola unidad y todos ellos se cierran si el objeto Statement se reprocesa bajo una transacción nueva. No es posible tener ResultSets procedentes de dos transacciones activas simultáneamente para una sola sentencia.

Multiplexado

La API JTA está diseñada para desasociar transacciones de conexiones JDBC. Esta API permite que varias conexiones trabajen en una sola transacción o que una sola conexión trabaje en varias transacciones simultáneamente. Esto se denomina **multiplexado**, que permite realizar muchas tareas complejas que no pueden realizarse solo con JDBC.

Para obtener más información acerca de la utilización de JTA, consulte la especificación JTA. La especificación JDBC 3.0 también contiene información acerca de cómo estas dos tecnologías funcionan conjuntamente para dar soporte a las transacciones distribuidas.

Compromiso de dos fases y anotación de transacciones

Las API JTA externalizan las responsabilidades del protocolo de compromiso de dos fases distribuido completamente en la aplicación. Como se ha mostrado en los ejemplos, al utilizar JTA y JDBC para acceder a una base de datos bajo una transacción JTA, la aplicación utiliza los métodos XAResource.prepare() y XAResource.commit() o solo el método XAResource.commit() para comprometer los cambios.

Además, al acceder a varias bases de datos distintas utilizando una sola transacción, es responsabilidad de la aplicación garantizar que se realicen el protocolo de compromiso de dos fases y las anotaciones asociadas necesarias para la atomicidad de la transacción en esas bases de datos. Generalmente, el proceso de compromiso de dos fases en varias bases de datos (es decir, XAResources) y sus anotaciones se realizan bajo el control de un servidor de aplicaciones o de un supervisor de transacciones para que la propia aplicación no tenga que preocuparse de estas cuestiones.

Por ejemplo, la aplicación puede llamar a algún método commit() o efectuar la devolución desde su proceso sin errores. El servidor de aplicaciones o supervisor de transacciones subyacente empezará entonces a procesar cada una de las bases de datos (XAResource) que ha participado en la única transacción distribuida.

El servidor de aplicaciones utilizará anotaciones extensivas durante el proceso de compromiso de dos fases. Llamará al método XAResource.prepare() para cada base de datos participante (XAResource), seguido de una llamada al método XAResource.commit() para cada base de datos participante (XAResource).

Si se produce una anomalía durante este proceso, las anotaciones del supervisor de transacciones del servidor de aplicaciones permiten que el propio servidor de aplicaciones utilice subsiguientemente las API de JTA para recuperar la transacción distribuida. Esta recuperación, bajo el control del servidor de aplicaciones o supervisor de transacciones, permite al servidor de aplicaciones situar la transacción en un estado conocido en cada base de datos participante (XAResource). Con ello garantiza un estado conocido de toda la transacción distribuida en todas las bases de datos participantes.

Conceptos relacionados

"Iniciación a JDBC" en la página 34
El controlador Java Database Connectivity (JDBC) que viene con IBM Developer Kit para Java también se llama controlador JDBC de IBM Developer Kit para Java. Este controlador también se conoce comúnmente como controlador JDBC nativo.

"Utilizar soporte de DataSource para la agrupación de objetos" en la página 132
Puede utilizar DataSources para que varias aplicaciones compartan una configuración común para acceder a una base de datos. Esta operación se realiza haciendo que cada una de las aplicaciones haga referencia al mismo nombre de DataSource.

Referencia relacionada

"Propiedades de ConnectionPoolDataSource" en la página 134
Puede configurar la interfaz ConnectionPoolDataSource utilizando el conjunto de propiedades que proporciona.

Información relacionada

Especificación de la API de transacciones Java (JTA) 1.0.1

Ejemplo: utilizar JTA para manejar una transacción:

Este es un ejemplo de cómo utilizar la API de transacciones Java para manejar una transacción en una aplicación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

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

public class JTACommit {

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

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

```
/**
 * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
 */
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) {
            // Ignorar... no existe
        }

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


/**
 * Esta prueba utiliza el soporte JTA para manejar transacciones.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presuponer que el origen de datos se apoya en un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Desde el DataSource, obtener un objeto XAConnection que
        // contenga un XAResource y un objeto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Para transacciones XA, es necesario un identificador de transacción.
        // No se incluye una implementación de la interfaz XID con el
        // controlador JDBC. En Transacciones con JTA hay una descripción de
        // esta interfaz para construir una clase para ella.
        Xid xid = new XidImpl();

        // La conexión de XAResource puede usarse como cualquier otra
        // conexión JDBC.
        Statement stmt = c.createStatement();

        // Hay que notificar al recurso XA antes de iniciar cualquier
        // trabajo transaccional.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Se realiza trabajo JDBC estándar.
        int count =
          stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA es bastante divertido.')");

        // Cuando el trabajo de transacción ha terminado, el recurso XA debe
        // recibir notificación de nuevo.
        xaRes.end(xid, XAResource.TMSUCCESS);
```

```
            // La transacción representada por el ID de transacción se prepara
            // para el compromiso.
            int rc = xaRes.prepare(xid);

            // La transacción se compromete mediante el XAResource.
            // No se utiliza el objeto JDBC Connection para comprometer
            // la transacción al utilizar JTA.
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: varias conexiones que funcionan en una transacción:

Este es un ejemplo de utilización de varias conexiones que trabajan en una sola transacción.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
   }
/**
* Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
*/
   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) {
         // Ignorar... no existe
         }
         s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
                         (50))");
               s.close();
      }
      finally {
```

```
         if (c != null) {
            c.close();
         }
      }
   }
/**
* Esta prueba utiliza el soporte JTA para manejar transacciones.
*/
   public void run() {
      Connection c1 = null;
      Connection c2 = null;
      Connection c3 = null;
      try {
         Context ctx = new InitialContext();
         // Presuponer que el origen de datos se apoya en un UDBXADataSource.
         UDBXADataSource ds = (UDBXADataSource)
                              ctx.lookup("XADataSource");
         // Desde el DataSource, obtener algunos objetos XAConnection que
         // contienen un XAResource y un objeto 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();
         // Para transacciones XA, es necesario un identificador de transacción.
         // El soporte para crear XID se deja de nuevo al programa
         // de aplicación.
         Xid xid = JDXATest.xidFactory();
         // Realizar algún trabajo transaccional bajo cada una de las tres
         // conexiones que se han creado.
         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);
         // Al terminar, comprometer la transacción como una sola unidad.
         // Es necesario prepare() y commit() o commit() de 1 fase para
         // cada base de datos independiente (XAResource) que ha participado en la
         // transacción. Dado que los recursos accedidos (xaRes1, xaRes2 y xaRes3)
         // se refieren todos a la misma base de datos, solo se necesita una
         // prepare o una commit.
         int rc = xaRes.prepare(xid);
         xaRes.commit(xid, false);
      }
      catch (Exception e) {
         System.out.println("Algo ha fallado.");
         e.printStackTrace();
      }
      finally {
         try {
            if (c1 != null) {
               c1.close();
            }
         }
```

```
         catch (SQLException e) {
            System.out.println("Nota: Excepción de limpieza " +
                               e.getMessage());
         }
         try {
            if (c2 != null) {
               c2.close();
            }
         }
         catch (SQLException e) {
            System.out.println("Nota: Excepción de limpieza " +
                               e.getMessage());
         }
         try {
            if (c3 != null) {
               c3.close();
            }
         }
         catch (SQLException e) {
            System.out.println("Nota: Excepción de limpieza " +
                               e.getMessage());
         }
      }
   }
}
```

Ejemplo: utlizar una conexión con múltiples transacciones:

Este es un ejemplo de utilización de una sola conexión con varias transacciones.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
```

```
        }

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

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


/**
 * Esta prueba utiliza el soporte JTA para manejar transacciones.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presuponer que el origen de datos se apoya en un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Desde el DataSource, obtener un objeto XAConnection que
        // contiene un XAResource y un objeto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();
        Statement     stmt   = c.createStatement();

        // Para transacciones XA, es necesario un identificador de transacción.
        // Esto no implica que todos los XID sean iguales.
        // Cada XID debe ser exclusivo para distinguir las diversas transacciones
        // que se producen.
        // El soporte para crear XID se deja de nuevo al programa
        // de aplicación.
        Xid xid1 = JDXATest.xidFactory();
        Xid xid2 = JDXATest.xidFactory();
        Xid xid3 = JDXATest.xidFactory();

        // Realizar el trabajo bajo tres transacciones para esta conexión.
        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);

        // Preparar todas las transacciones
        int rc1 = xaRes.prepare(xid1);
        int rc2 = xaRes.prepare(xid2);
        int rc3 = xaRes.prepare(xid3);

        // Dos de las transacciones se comprometen y la tercera se retrotrae.
        // El intento de insertar el segundo valor en la tabla
        // no se compromete.
        xaRes.commit(xid1, false);
        xaRes.rollback(xid2);
        xaRes.commit(xid3, false);
```

```
        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: ResultSets suspendidos:

Este es un ejemplo de cómo se reprocesa un objeto Statement bajo otra transacción para realizar el trabajo.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
            }

            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();
        }
    }
}


/**
 * Esta prueba utiliza el soporte JTA para manejar transacciones.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presuponer que el origen de datos se apoya en un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Desde el DataSource, obtener un objeto XAConnection que
        // contiene un XAResource y un objeto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Para transacciones XA, es necesario un identificador de transacción.
        // No se incluye una implementación de la interfaz XID con
        // el controlador JDBC. Vea: Transacciones con JTA
        // si desea obtener una descripción de esta interfaz para construir una
        // clase para ella.
        Xid xid = new XidImpl();

        // La conexión de XAResource puede usarse como cualquier otra
        // conexión JDBC.
        Statement stmt = c.createStatement();

        // Hay que notificar al recurso XA antes de iniciar cualquier
        // trabajo transaccional.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Crear un ResultSet durante el proceso de JDBC y captar una fila.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Se llama al método end con la opción suspend. Los
        // ResultSets asociados a la transacción actual quedan 'suspendidos'.
        // En este estado, no se eliminan ni son accesibles.
        xaRes.end(xid, XAResource.TMSUSPEND);


        // Mientras tanto, pueden realizarse otras tareas fuera de la
        // transacción.
        // Los ResultSets bajo la transacción pueden cerrarse si
        // se reutiliza el objeto Statement utilizado para crearlos.
        ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
        while (nonXARS.next()) {
            // Procesar aquí...
        }


        // Intento de volver a la transacción suspendida. El Resultset
        // de la transacción suspendida ha desaparecido porque la sentencia
        // se ha procesado de nuevo.
        xaRes.start(newXid, XAResource.TMRESUME);
        try {
            rs.next();
        } catch (SQLException ex) {
            System.out.println("Esta es una excepción esperada. " +
```

```
                                "El ResultSet se ha cerrado debido a otro proceso.");
            }


            // Cuando la transacción termine, finalizarla
            // y comprometer el trabajo que contenga.
            xaRes.end(xid, XAResource.TMNOFLAGS);
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: finalizar una transacción:

Este es un ejemplo de cómo finalizar una transacción en la aplicación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
```

```
        }

        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();
        }
    }
}


/**
 * Esta prueba utiliza el soporte JTA para manejar transacciones.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presuponer que el origen de datos se apoya en un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Desde el DataSource, obtener un objeto XAConnection que
        // contiene un XAResource y un objeto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Para transacciones XA, es necesario un identificador de transacción.
        // No se incluye una implementación de la interfaz XID con
        // el controlador JDBC. Vea: Transacciones con JTA
        // para obtener una descripción de esta interfaz para crear una clase para ella.
        Xid xid = new XidImpl();

        // La conexión de XAResource puede usarse como cualquier otra
        // conexión JDBC.
        Statement stmt = c.createStatement();

        // Hay que notificar al recurso XA antes de iniciar cualquier
        // trabajo transaccional.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Crear un ResultSet durante el proceso de JDBC y captar una fila.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Cuando se llama al método end, se cierran todos los cursores de ResultSet.
        // El intento de acceder a ResultSet después de este punto provoca
        // el lanzamiento de una excepción.
        xaRes.end(xid, XAResource.TMNOFLAGS);

        try {
            String value = rs.getString(1);
            System.out.println("Algo ha fallado si recibe ese mensaje.");
        } catch (SQLException e) {
            System.out.println("Se ha lanzado la excepción esperada.");
        }

        // Comprometer la transacción para asegurarse que todos los bloqueos
        // se liberan.
        int rc = xaRes.prepare(xid);
        xaRes.commit(xid, false);
```

```
        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

"Transacciones JDBC distribuidas" en la página 82
Generalmente, en Java Database Connectivity (JDBC) las transacciones son locales. Esto significa que una sola conexión realiza todo el trabajo de la transacción y que la conexión solo puede trabajar en una transacción a la vez.

Ejemplo: suspender y reanudar una transacción:

Este es un ejemplo de una transacción que se suspende y luego se reanuda.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
            }

            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();
        }
    }
}


/**
 * Esta prueba utiliza el soporte JTA para manejar transacciones.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Presuponer que el origen de datos se apoya en un UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // Desde el DataSource, obtener un objeto XAConnection que
        // contiene un XAResource y un objeto Connection.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // Para transacciones XA, es necesario un identificador de transacción.
        // No se incluye una implementación de la interfaz XID con
        // el controlador JDBC. Vea Transacciones con JTA
        // para obtener una descripción de esta interfaz para crear una clase para ella.
        Xid xid = new XidImpl();

        // La conexión de XAResource puede usarse como cualquier otra
        // conexión JDBC.
        Statement stmt = c.createStatement();

        // Hay que notificar al recurso XA antes de iniciar cualquier
        // trabajo transaccional.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Crear un ResultSet durante el proceso de JDBC y captar una fila.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // Se llama al método end con la opción suspend. Los
        // ResultSets asociados a la transacción actual quedan 'suspendidos'.
        // En este estado, no se eliminan ni son accesibles.
        xaRes.end(xid, XAResource.TMSUSPEND);


        // Pueden realizarse otras tareas con la transacción.
        // Como ejemplo, puede crear una sentencia y procesar una consulta.
        // Este trabajo, y cualquier otro transaccional que la transacción pueda
        // realizar, está separado del trabajo realizado anteriormente bajo XID.
        Statement nonXAStmt = conn.createStatement();
        ResultSet nonXARS = nonXAStmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
        while (nonXARS.next()) {
            // Procesar aquí...
        }
        nonXARS.close();
        nonXAStmt.close();


        // Si se intenta utilizar recursos de transacciones
```

```
            // suspendidas, se produce una excepción.
            try {
                rs.getString(1);
                System.out.println("El valor de la primera fila es " + rs.getString(1));
            } catch (SQLException e) {
                System.out.println("Esta es una excepción esperada - " +
                                   "se ha utilizado ResultSet suspendido.");
            }


            // Reanudar la transacción suspendida y terminar el trabajo en ella.
            // El ResultSet es exactamente igual que antes de la suspensión.
            xaRes.start(newXid, XAResource.TMRESUME);
            rs.next();
            System.out.println("El valor de la segunda fila es " + rs.getString(1));


            // Cuando la transacción termine, finalizarla
            // y comprometer el trabajo que contenga.
            xaRes.end(xid, XAResource.TMNOFLAGS);
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

Tipos de sentencia

La interfaz Statement y sus subclases PreparedStatement y CallableStatement se utilizan para procesar mandatos de lenguaje de consulta estructurada (SQL) en la base de datos. Las sentencias SQL provocan la generación de objetos ResultSet.

Las subclases de la interfaz Statement se crean con diversos métodos de la interfaz Connection. Bajo un solo objeto Connection se pueden crear simultáneamente muchos objetos Statement. En releases anteriores, era posible dar números exactos de objetos Statement que podían crearse. Esto es imposible en este release, ya que los diversos tipos de objetos Statement toman números diferentes de ″handles″ dentro del motor de bases de datos. Por tanto, los tipos de objetos Statement que utilice influyen sobre el número de sentencias que pueden estar activas bajo una conexión en un momento dado.

Una aplicación llama al método Statement.close para indicar que ha terminado el proceso de una sentencia. Todos los objetos Statement se cierran cuando se cierra la conexión que los ha creado. Sin embargo, este comportamiento no es del todo fiable en lo que a cerrar objetos Statement se refiere. Por ejemplo, si la aplicación cambia de forma que se utiliza una agrupación de conexiones en lugar de cerrar explícitamente las conexiones, la aplicación se queda sin handles de sentencia, ya que la conexión nunca se cierra. El hecho de cerrar los objetos Statement en cuanto ya no son necesarios permite liberar inmediatamente los recursos externos de base de datos utilizados por la sentencia.

El controlador JDBC nativo intenta detectar fugas de sentencia y las maneja en nombre del usuario. Sin embargo, basarse en ese soporte provoca un rendimiento inferior.

Debido a la jerarquía de herencia, según la cual CallableStatement amplía PreparedStatement, que a su vez amplía Statement, las características de cada una de las interfaces están disponibles en la clase que amplía la interfaz. Por ejemplo, las características de la clase Statement también están soportadas en las clases PreparedStatement y CallableStatement. La excepción principal la constituyen los métodos executeQuery, executeUpdate y execute de la clase Statement. Estos métodos toman una sentencia SQL para procesarla dinámicamente y provocan excepciones si el usuario intenta utilizarlos con objetos PreparedStatement o CallableStatement.

Objetos Statement:

El objeto Statement (sentencia) sirve para procesar una sentencia SQL estática y obtener los resultados producidos por ella. Solo puede haber un ResultSet abierto para cada objeto Statement en un momento dado. Todos los métodos statement que procesan una sentencia SQL cierran implícitamente el ResultSet actual de una sentencia si existe uno abierto.

Crear sentencias

Los objetos Statement se crean a partir de objetos Connection con el método createStatement. Por ejemplo, suponiendo que ya exista un objeto Connection denominado conn, la siguiente línea de código crea un objeto Statement para pasar sentencias SQL a la base de datos:

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

Especificar características de ResultSet

Las características de los ResultSets están asociadas con la sentencia que finalmente los crea. El método Connection.createStatement permite especificar estas características de ResultSet. A continuación se ofrecen algunos ejemplos de llamadas válidas al método createStatement:

Ejemplo: método createStatement

```
// El siguiente código es nuevo en JDBC 2.0

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

// El siguiente código es nuevo en JDBC 3.0

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

Para obtener más información acerca de estas características, consulte la sección ResultSets.

Procesar sentencias

El proceso de sentencias SQL con un objeto Statement se realiza mediante los métodos executeQuery(), executeUpdate() y execute().

Devolver resultados desde consultas SQL

Si debe procesarse una sentencia de consulta SQL que devuelva un objeto ResultSet, debe utilizarse el método executeQuery(). Puede consultar el programa de ejemplo que utiliza el método executeQuery de un objeto Statement para obtener un ResultSet.

Nota: si una sentencia SQL que se procesa con el método executeQuery no devuelve un ResultSet, se lanza una SQLException.

Devolver cuentas de actualización para sentencias SQL

Si se sabe que el código SQL es una sentencia de lenguaje de definición de datos (DDL) o una sentencia de lenguaje de manipulación de datos (DML) que devuelve una cuenta de actualización, debe utilizarse el método executeUpdate(). El programa StatementExample utiliza el método executeUpdate de un objeto Statement.

Procesar sentencias SQL en las que el valor de retorno esperado es desconocido

Si no se sabe cuál es el tipo de sentencia SQL, debe utilizarse el método execute. Una vez que se ha procesado este método, el controlador JDBC puede indicar a la aplicación qué tipos de resultados ha generado la sentencia SQL mediante las llamadas de API. El método execute devuelve true si el resultado es un ResultSet como mínimo, y false si el valor de retorno es una cuenta de actualización. Con esta información, las aplicaciones pueden utilizar los métodos de sentencia getUpdateCount o getResultSet para recuperar el valor de retorno del proceso de la sentencia SQL. El programa StatementExecute utiliza el método execute en un objeto Statement. Este programa espera que se pase un parámetro que sea una sentencia SQL. Sin mirar el texto de la sentencia SQL que el usuario proporciona, el programa procesa la sentencia y determina la información relativa a lo que se ha procesado.

Nota: la llamada al método getUpdateCount cuando el resultado es un ResultSet devuelve -1. La llamada al método getResultSet cuando el resultado es una cuenta de actualización devuelve nulo.

El método cancel

Los métodos del controlador JDBC nativo están sincronizados para evitar que dos hebras que se ejecutan en el mismo objeto provoquen daños en el mismo. Una excepción a esta norma la representa el método cancel. Una hebra puede utilizar el método cancel para detener una sentencia SQL de larga ejecución en otra hebra que opera sobre el mismo objeto. El controlador JDBC nativo no puede obligar a la hebra a detenerse; solo puede solicitarle que detenga la tarea que estaba realizando. Por esta razón, una sentencia cancelada tarda tiempo en detenerse. El método cancel puede utilizarse para detener consultas SQL incontroladas en el sistema.

Ejemplo: utilizar el método executeUpdate del objeto Statement:

Este es un ejemplo de utilización del método executeUpdate del objeto Statement.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

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

public class StatementExample {

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

        // Sugerencia: cargarlos desde un objeto de propiedades.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Registrar el controlador JDBC nativo. Si el controlador no puede
        // registrarse, la prueba no puede continuar.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("No se ha podido registrar el controlador.");
            System.out.println(e.getMessage());
            System.exit(1);
        }
```

```
        Connection c = null;
        Statement s = null;

        try {
            // Crear las propiedades de conexión.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Conectar con la base de datos local de System i5.
            c = DriverManager.getConnection(URL, properties);

            // Crear un objeto Statement.
            s = c.createStatement();
            // Suprimir la tabla de prueba, si existe. Nota: en este
            // ejemplo se presupone que la colección MYLIBRARY
            // existe en el sistema.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Continuar simplemente... es probable que la tabla no exista.
            }

            // Ejecutar una sentencia SQL que crea una tabla en la base de datos.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

            // Ejecutar algunas sentencias SQL que insertan registros en la tabla.
            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)");

            // Ejecutar una consulta SQL en la tabla.
            ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

            // Visualizar todos los datos de la tabla.
            while (rs.next()) {
                System.out.println("El empleado " + rs.getString(1) + " tiene el ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("El proceso de base de datos ha fallado.");
            System.out.println("Razón: " + sqle.getMessage());
        } finally {
            // Cerrar los recursos de base de datos.
            try {
                if (s != null) {
                    s.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Statement.");
            }
             }

            try {
                if (c != null) {
                    c.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Connection.");
            }
        }
    }
}
```

PreparedStatements:

Las PreparedStatements amplían la interfaz Statement y proporcionan soporte para añadir parámetros a sentencias SQL.

Las sentencias SQL que se pasan a la base de datos pasan por un proceso de dos pasos al devolver los resultados al usuario. Primero se preparan y, a continuación, se procesan. Con los objetos Statement, estas dos fases aparecen como una sola en las aplicaciones. Las PreparedStatements permiten independizar estos dos procesos. El paso de preparación se produce cuando se crea el objeto, y el paso de proceso se produce cuando se llama a los métodos executeQuery, executeUpdate o execute en el objeto PreparedStatement.

La posibilidad de dividir el proceso SQL en fases independientes no tiene sentido sin la adición de marcadores de parámetro. Los marcadores de parámetro se colocan en una aplicación para que esta pueda indicar a la base de datos que no tiene un valor específico durante la preparación, pero que proporciona uno durante el proceso. En las sentencias SQL, los marcadores de parámetro se representan mediante signos de interrogación.

Los marcadores de parámetro posibilitan la creación de sentencias SQL generales que se utilizan para peticiones específicas. Por ejemplo, considere la siguiente sentencia de consulta SQL:

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

Se trata de una sentencia SQL específica que devuelve un solo valor; es decir, la información relativa a un empleado llamado Dettinger. Si se añade un marcador de parámetro, la sentencia puede ser más flexible:

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

Estableciendo simplemente el marcador de parámetro en un valor, puede obtenerse información acerca de cualquier empleado de la tabla.

Las PreparedStatements proporcionan mejoras de rendimiento significativas con respecto a las Statements; el ejemplo de Statement anterior puede pasar por la fase de preparación una sola vez y, a continuación, procesarse repetidamente con valores diferentes para el parámetro.

Nota: La utilización de PreparedStatements es obligatoria para dar soporte a la agrupación de sentencias del controlador JDBC nativo.

Para obtener más información sobre cómo utilizar sentencias preparadas, incluida la creación de sentencias preparadas, especificar características del conjunto de resultados, trabajar con claves generadas automáticamente y establecer marcas de parámetros, consulte las páginas siguientes:

Crear y utilizar PreparedStatements:

Para crear objetos PreparedStatement nuevos, se utiliza el método prepareStatement. A diferencia de en el método createStatement, la sentencia SQL debe suministrarse al crear el objeto PreparedStatement. En ese momento, la sentencia SQL se precompila para su utilización.

Por ejemplo, suponiendo que ya exista un objeto Connection, el ejemplo siguiente crea un objeto PreparedStatement y prepara la sentencia SQL para que se procese en la base de datos:

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

Epecificar las características de ResultSet y el soporte de claves generadas automáticamente

Al igual que el método createStatement, el método prepareStatement se ha cargado a posteriori para proporcionar soporte para la especificación de características de ResultSet. El método prepareStatement también presenta variantes para trabajar con claves generadas automáticamente. A continuación se ofrecen algunos ejemplos de llamadas válidas al método prepareStatement:

Ejemplo: método prepareStatement

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Nuevo en JDBC 2.0

PreparedStatement ps2 = conn.prepareStatement("SELECT * FROM
    EMPLOYEE_TABLE WHERE LASTNAME = ?",

ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATEABLE);

// Nuevo en 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);
```

Manejar los parámetros

Para poder procesar un objeto PreparedStatement, debe establecerse un valor en cada uno de los marcadores de parámetro. El objeto PreparedStatement proporciona varios métodos para establecer parámetros. Todos los métodos tienen el formato set<Tipo>, siendo <Tipo> un tipo de datos Java. Son ejemplos de estos métodos setInt, setLong, setString, setTimestamp, setNull y setBlob. Casi todos estos métodos toman dos parámetros:

- El primero es el índice que el parámetro tiene dentro de la sentencia. Los marcadores de parámetro están numerados, empezando por el 1.
- El segundo parámetro es el valor que debe establecerse en el parámetro. Hay un par de métodos set<Tipo> que tienen parámetros adicionales, como el parámetro longitud del método setBinaryStream.

Consulte el Javadoc del paquete java.sql para obtener más información. Tomando la sentencia SQL preparada en los ejemplos anteriores del objeto ps, el código siguiente muestra cómo se especifica el valor de parámetro antes del proceso:

```
ps.setString(1,'Dettinger');
```

Si se intenta procesar una PreparedStatement con marcadores de parámetro que no se han establecido, se lanza una SQLException.

Nota: Una vez establecidos, los marcadores de parámetro conservan el mismo valor entre los procesos, a menos que se produzca una de las siguientes situaciones:

- Otra llamada a un método set cambia el valor.
- El valor se elimina cuando se llama al método clearParameters.

El método clearParameters identifica todos los parámetros como no establecidos. Una vez efectuada la llamada al método clearParameters, es necesario llamar de nuevo al método set en todos los parámetros antes del próximo proceso.

Soporte de ParameterMetaData

Una nueva interfaz ParameterMetaData permite recuperar información acerca de un parámetro. Este soporte es complementario de ResultSetMetaData, y es parecido. Se suministra información acerca de precisión, escala, tipos de datos, nombre de tipo de datos y si el parámetro permite el valor nulo.

Ejemplo: ParameterMetaData:

Este es un ejemplo de utilización de la interfaz ParameterMetaData para recuperar información acerca de los parámetros.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de ParameterMetaData. Este programa muestra
// el nuevo soporte de JDBC 3.0 para obtener información
// acerca de los parámetros de una PreparedStatement.
//
// Sintaxis de mandato:
//    java PMD
//
//////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo del controlador JDBC de IBM Developer para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo con fines ilustrativos
// solamente. Los ejemplos no se han probado minuciosamente bajo todas
// las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas contenidos aquí se proporcionan "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se declinan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// 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 de entrada del programa.
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Obtener configuración.
        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("Número parámetro " + i);
            System.out.println("  El nombre de clase es " + pmd.getParameterClassName(i));
            // Nota: la modalidad hace referencia a entrada, salida y entrada/salida
            System.out.println("  La modalidad es " + pmd.getParameterClassName(i));
            System.out.println("  El tipo es " + pmd.getParameterType(i));
            System.out.println("  El nombre de tipo es " + pmd.getParameterTypeName(i));
            System.out.println("  La precisión es " + pmd.getPrecision(i));
            System.out.println("  La escala es " + pmd.getScale(i));
```

```
            System.out.println("  ¿Posibilidad de nulos? es " + pmd.isNullable(i));
            System.out.println("  ¿Firmado? es " + pmd.isSigned(i));
        }
    }
}
```

Procesar PreparedStatements:

El proceso de sentencias SQL con un objeto PreparedStatement se realiza mediante los métodos executeQuery, executeUpdate y execute, al igual que el proceso de objetos Statement. A diferencia de las versiones de Statement, no se pasan parámetros en estos métodos debido a que la sentencia SQL ya se ha suministrado al crear el objeto. Dado que PreparedStatement amplía Statement, las aplicaciones pueden intentar llamar a versiones de los métodos executeQuery, executeUpdate y execute que toman una sentencia SQL. Esta operación provoca el lanzamiento de una excepción SQLException.

Devolver resultados desde consultas SQL

Si debe procesarse una sentencia de consulta SQL que devuelva un objeto ResultSet, debe utilizarse el método executeQuery. El programa PreparedStatementExample utiliza el método executeQuery de un objeto PreparedStatement para obtener un ResultSet.

Nota: Si una sentencia SQL se procesa con el método executeQuery y no devuelve un ResultSet, se lanza una SQLException.

Devolver cuentas de actualización para sentencias SQL

Si se sabe que el código SQL es una sentencia de lenguaje de definición de datos (DDL) o una sentencia de lenguaje de manipulación de datos (DML) que devuelve una cuenta de actualización, debe utilizarse el método executeUpdate. El programa PreparedStatementExample utiliza el método executeUpdate de un objeto PreparedStatement.

Procesar sentencias SQL en las que se desconoce el valor de retorno esperado

Si no se sabe cuál es el tipo de sentencia SQL, debe utilizarse el método execute. Una vez que se ha procesado este método, el controlador JDBC puede indicar a la aplicación qué tipos de resultados ha generado la sentencia SQL mediante las llamadas de API. El método execute devuelve true si el resultado es un ResultSet como mínimo, y false si el valor de retorno es una cuenta de actualización. Con esta información, las aplicaciones pueden utilizar los métodos de sentencia getUpdateCount o getResultSet para recuperar el valor de retorno del proceso de la sentencia SQL.

Nota: Si se llama al método getUpdateCount cuando el resultado es un ResultSet, el valor de retorno es -1. Si se llama al método getResultSet cuando el resultado es una cuenta de actualización, el valor de retorno es null.

Ejemplo: utilizar PreparedStatement para obtener un ResultSet:

Este es un ejemplo de utilización del método executeQuery de un objeto PreparedStatement para obtener un ResultSet.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

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

public class PreparedStatementExample {

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

```
// Cargar lo siguiente desde un objeto de propiedades.
String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
String URL    = "jdbc:db2://*local";

// Registrar el controlador JDBC nativo. Si el controlador no puede
// registrarse, la prueba no puede continuar.
try {
    Class.forName(DRIVER);
} catch (Exception e) {
    System.out.println("No se ha podido registrar el controlador.");
    System.out.println(e.getMessage());
    System.exit(1);
}

Connection c = null;
Statement s = null;

//    Este programa crea una tabla que
//    las sentencias preparadas utilizan más tarde.
try {
    // Crear las propiedades de conexión.
    Properties properties = new Properties ();
    properties.put ("user", "userid");
    properties.put ("password", "password");

    // Conectar con la base de datos local.
    c = DriverManager.getConnection(URL, properties);

    // Crear un objeto Statement.
    s = c.createStatement();
    // Suprimir la tabla de prueba, si existe. Observar que
    // en todo este ejemplo se presupone que la colección
    // MYLIBRARY existe en el sistema.
    try {
        s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
    } catch (SQLException e) {
        // Continuar simplemente... es probable que la tabla no exista.
    }

    // Ejecutar una sentencia SQL que cree una tabla en la base de datos.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

} catch (SQLException sqle) {
    System.out.println("El proceso de base de datos ha fallado.");
    System.out.println("Razón: " + sqle.getMessage());
} finally {
    // Cerrar los recursos de base de datos.
    try {
        if (s != null) {
            s.close();
        }
    } catch (SQLException e) {
        System.out.println("El borrado no ha podido cerrar Statement.");
    }
}


// Luego, este programa usa una sentencia preparada para insertar muchas
// filas en la base de datos.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Crear un objeto PreparedStatement utilizado para insertar datos en la
    // tabla.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");
```

```
            for (int i = 0; i < nameArray.length; i++) {
                ps.setString(1, nameArray[i]);  // Establecer el nombre a partir de nuestra matriz.
                ps.setInt(2, i+1);              // Establecer el ID.
                ps.executeUpdate();
            }

        } catch (SQLException sqle) {
            System.out.println("El proceso de base de datos ha fallado.");
            System.out.println("Razón: " + sqle.getMessage());
        } finally {
            // Cerrar los recursos de base de datos.
            try {
                if (ps != null) {
                    ps.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Statement.");
            }
        }


        //    Utilizar una sentencia preparada para consultar la tabla de
        //    base de datos creada y devolver datos desde ella. En
        //    este ejemplo, el parámetro usado se ha establecido arbitrariamente
        //    en 5, es decir, devolver todas las filas en que el campo ID sea menor
        //    o igual a 5.
        try {
            ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
                                    "WHERE ID <= ?");

            ps.setInt(1, 5);

            // Ejecutar una consulta SQL en la tabla.
            ResultSet rs = ps.executeQuery();
            // Visualizar todos los datos de la tabla.
            while (rs.next()) {
                System.out.println("El empleado " + rs.getString(1) + " tiene el ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("El proceso de base de datos ha fallado.");
            System.out.println("Razón: " + sqle.getMessage());
        } finally {
            // Cerrar los recursos de base de datos.
            try {
                if (ps != null) {
                    ps.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Statement.");
            }

            try {
                if (c != null) {
                    c.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Connection.");
;
            }

        }
    }
}
```

CallableStatements:

La interfaz CallableStatement de JDBC amplía PreparedStatement y proporciona soporte para parámetros de salida y de entrada/salida. La interfaz CallableStatement tiene también soporte para parámetros de entrada, que proporciona la interfaz PreparedStatement.

La interfaz CallableStatement permite la utilización de sentencias SQL para llamar a procedimientos almacenados. Los procedimientos almacenados son programas que tienen una interfaz de base de datos. Estos programas tienen lo siguiente:

- Pueden tener parámetros de entrada y de salida, o parámetros que son tanto de entrada como de salida.
- Pueden tener un valor de retorno.
- Tienen la capacidad de devolver varios ResultSets.

Conceptualmente, en JDBC una llamada de procedimiento almacenado es una sola llamada a la base de datos, pero el programa asociado con el procedimiento almacenado puede procesar cientos de peticiones de base de datos. El programa de procedimiento almacenado también puede realizar otras diversas tareas programáticas que no realizan habitualmente las sentencias SQL.

Las CallableStatements, debido a que siguen el modelo de PreparedStatement (que consiste en desacoplar las fases de preparación y proceso), permiten la reutilización optimizada (los detalles están en: “PreparedStatements” en la página 100). Dado que las sentencias SQL de un procedimiento almacenado están enlazadas a un programa, se procesan como SQL estático, y con ello puede obtenerse un rendimiento superior. La encapsulación de gran cantidad de trabajo de base de datos en una sola llamada de base de datos reutilizable es un ejemplo de utilización óptima de procedimientos almacenados. Solo esta llamada se transmite por la red al otro sistema, pero la petición puede realizar gran cantidad de trabajo en el sistema remoto.

Crear CallableStatements

El método prepareCall se utiliza para crear objetos CallableStatement nuevos. Al igual que en el método prepareStatement, la sentencia SQL debe suministrarse en el momento de crear el objeto CallableStatement. En ese momento, se precompila la sentencia SQL. Por ejemplo, suponiendo que ya exista un objeto Connection denominado conn, el siguiente código crea un objeto CallableStatement y realiza la fase de preparación de la sentencia SQL para que se procese en la base de datos:

```
PreparedStatement ps = conn.prepareStatement("? = CALL ADDEMPLOYEE(?, ?, ?");
```

El procedimiento almacenado ADDEMPLOYEE toma parámetros de entrada para un nuevo nombre de empleado, su número de seguridad social y el ID de usuario de su administrador. A partir de esta información, pueden actualizarse varias tablas de base de datos de la empresa con información relativa al empleado, como por ejemplo la fecha en que ha empezado a trabajar, la división, el departamento, etc. Además, un procedimiento almacenado es un programa que puede generar ID de usuario estándar y direcciones de correo electrónico para ese empleado. El procedimiento almacenado también puede enviar un correo electrónico al administrador que ha contratado al empleado con nombres de usuario y contraseñas iniciales; a continuación, el administrador puede proporcionar la información al empleado.

El procedimiento almacenado ADDEMPLOYEE está configurado para tener un valor de retorno. El código de retorno puede ser un código de éxito o error que el programa que efectúa la llamada puede utilizar cuando se produce una anomalía. El valor de retorno también puede definirse como el número de ID de empresa del nuevo empleado. Finalmente, el programa de procedimiento almacenado puede haber procesado consultas internamente y haber dejado los ResultSets de esas consultas abiertos y disponibles para el programa que efectúa la llamada. Consultar toda la información del nuevo empleado y ponerla a disposición del llamador mediante un ResultSet devuelto es una operación que tiene sentido.

En las secciones siguientes se describe cómo realizar cada uno de estos tipos de tareas.

Especificar las características de ResultSet y el soporte de claves generadas automáticamente

Al igual que en createStatement y prepareStatement, existen varias versiones de prepareCall que proporcionan soporte para especificar características de ResultSet. A diferencia de prepareStatement, el método prepareCall no proporciona variantes para trabajar con claves generadas automáticamente a partir de CallableStatements (JDBC 3.0 no soporta este concepto). A continuación se ofrecen algunos ejemplos de llamadas válidas al método prepareCall:

Ejemplo: método prepareCall

```
// El siguiente código es nuevo en JDBC 2.0

CallableStatement cs2 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE);

// Nuevo en JDBC 3.0

CallableStatement cs3 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,
    ResultSet.HOLD_CURSOR_OVER_COMMIT);
```

Manejar los parámetros

Como se ha indicado, los objetos CallableStatement pueden tomar tres tipos de parámetros:

- **IN**

  Los parámetros IN se manejan de la misma forma que PreparedStatements. Los diversos métodos set de la clase PreparedStatement heredada se utilizan para establecer los parámetros.
- **OUT**

  Los parámetros OUT se manejan con el método registerOutParameter. En la forma más común de registerOutParameter, el primer parámetro es un índice de parámetro y el segundo es un tipo de SQL. Este indica al controlador JDBC qué tipo de datos debe esperar del parámetro cuando se procesa la sentencia. Existen otras dos variantes del método registerOutParameter que pueden encontrarse en el Javadoc del paquete java.sql.
- **INOUT**

  Los parámetros INOUT requieren que se realice el trabajo tanto para parámetros IN como para parámetros OUT. Para cada parámetro INOUT, debe llamarse a un método set y al método registerOutParameter para que pueda procesarse la sentencia. Si alguno de los parámetros no se establece o registra, se lanza una SQLException cuando se procesa la sentencia.

Hallará más información en: “Ejemplo: crear un procedimiento con parámetros de entrada y salida” en la página 112.

Al igual que en PreparedStatements, los valores de parámetro de CallableStatement permanecen estables entre los procesos, a menos que llame de nuevo a un método set. El método clearParameters no afecta a los parámetros registrados para la salida. Después de llamar a clearParameters, todos los parámetros IN deben establecerse de nuevo en un valor, pero ninguno de los parámetros OUT tiene que registrarse de nuevo.

Nota: El concepto de parámetro no debe confundirse con el de índice de un marcador de parámetro. Una llamada de procedimiento almacenado espera que se le pase un determinado número de parámetros. Una sentencia SQL determinada contiene caracteres ″?″ (marcadores de parámetro) para representar los valores que se suministrarán en el momento de la ejecución. El siguiente ejemplo muestra la diferencia que existe entre los dos conceptos:

```
CallableStatement cs = con.prepareCall("CALL PROC(?, "SECOND", ?)");

cs.setString(1, "First");     //Marcador de parámetro 1, parámetro de procedimiento almacenado 1

cs.setString(2, "Third");     //Marcador de parámetro 2, parámetro de procedimiento almacenado 3
```

Acceder a los parámetros de procedimientos almacenados por su nombre

Los parámetros de procedimientos almacenados tienen nombres asociados a ellos, como se ve en esta declaración de procedimiento almacenado:

Ejemplo: parámetros de procedimiento almacenado

```
CREATE
PROCEDURE MYLIBRARY.APROC
    (IN PARM1 INTEGER)
LANGUAGE SQL SPECIFIC MYLIBRARY.APROC
BODY: BEGIN
    <Realizar aquí una tarea...>
END BODY
```

Existe un solo parámetro de tipo integer con el nombre PARM1. En JDBC 3.0, existe soporte para especificar parámetros de procedimiento almacenado por el nombre y por índice. El código para configurar una CallableStatement para este procedimiento es el siguiente:

```
CallableStatement cs = con.prepareCall("CALL APROC(?)");

cs.setString("PARM1", 6);     //Establece el parámetro de entrada del índice 1
                              //(PARM1) en 6.
```

Procesar CallableStatements:

Al procesar llamadas de procedimiento almacenado SQL con un objeto CallableStatement de JDBC se emplean los mismos métodos que con un objeto PreparedStatement.

Devolver resultados para procedimientos almacenados

Si una sentencia SQL se procesa dentro de un procedimiento almacenado, los resultados de la consulta pueden ponerse a disposición del programa que llama al procedimiento almacenado. También puede llamarse a varias consultas dentro del procedimiento almacenado, y el programa que efectúa la llamada puede procesar todos los ResultSets disponibles.

Consulte “Ejemplo: crear un procedimiento con múltiples ResultSets” en la página 110 para obtener más información.

Nota: Si un procedimiento almacenado se procesa con executeQuery y no devuelve un ResultSet, se lanza una SQLException.

Acceder a ResultSets de forma concurrente

El tema Devolver resultados para procedimientos almacenados trata sobre los ResultSets y los procedimientos almacenados y proporciona un ejemplo que funciona en todos los releases de Java Development Kit (JDK). En el ejemplo, los ResultSets se procesan por orden, empezando por el primer ResultSet que el procedimiento almacenado ha abierto hasta el último ResultSet abierto. El ResultSet anterior se cierra antes de utilizar el siguiente.

En JDK 1.4 y versiones posteriores, existe soporte para trabajar con ResultSets de procedimientos almacenados de forma concurrente.

Nota: Esta característica se añadió al soporte de sistema subyacente mediante la interfaz de línea de mandatos (CLI) en V5R2. En consecuencia, JDK 1.4 y posteriores versionesejecutado en un sistema anterior a V5R2 no tiene disponible este soporte.

Devolver cuentas de actualización para procedimientos almacenados

La devolución de cuentas de actualización para procedimientos almacenados es una característica que se describe en la especificación JDBC, pero no está soportada actualmente en la plataforma System i. No existe ninguna forma de devolver varias cuentas de actualización desde una llamada de procedimiento almacenado. Si es necesaria una cuenta de actualización de una sentencia SQL procesada en un procedimiento almacenado, existen dos formas de devolver el valor:

- Devolver el valor como parámetro de salida.
- Pasar de nuevo el valor como valor de retorno del parámetro. Este es un caso especial de parámetro de salida. Consulte la sección Procesar procedimientos almacenados que tienen retorno para obtener más información.

Procesar procedimientos almacenados en los que el valor esperado es desconocido

Si el resultado de una llamada de procedimiento almacenado no es conocido, debe utilizarse el método execute. Una vez que se ha procesado este método, el controlador JDBC puede indicar a la aplicación qué tipos de resultados ha generado el procedimiento almacenado mediante las llamadas de API. El método execute devuelve true si el resultado es uno o varios ResultSets. Las cuentas de actualización no provienen de llamadas de procedimiento almacenado.

Procesar procedimientos almacenados que tienen valor de retorno

La plataforma System i soporta los procedimientos almacenados que tienen un valor de retorno parecido al valor de retorno de una función. El valor de retorno de un procedimiento almacenado se etiqueta como otras marcas de parámetro, según lo asignado por la llamada de procedimiento almacenado. A continuación se ofrece un ejemplo de esta descripción:

```
? = CALL MYPROC(?, ?, ?)
```

El valor de retorno de una llamada de procedimiento almacenado es siempre un tipo integer y debe registrarse igual que cualquier otro parámetro de salida.

Consulte “Ejemplo: crear un procedimiento con valores de retorno” en la página 113 para obtener más información.

Ejemplo: crear un procedimiento con múltiples ResultSets:

En este ejemplo se muestra cómo acceder a una base de datos y luego crear un procedimiento con múltiples ResultSets utilizando JDBC.

Nota: Lea la Declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

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

public class CallableStatementExample1 {

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

        // Registrar el controlador JDBC nativo. Si no podemos
        // registrar el controlador, la prueba no puede continuar.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

```
// Crear las propiedades de la conexión
Properties properties = new Properties ();
properties.put ("user", "userid");
properties.put ("password", "password");

// Conectar con la base de datos del servidor local
Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

Statement s = c.createStatement();

// Crear un procedimiento con múltiples 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: Aquí ignoramos el error. Hacemos la
    //       suposición de que el único motivo de que esto falle
    //       se debe a que el procedimiento ya existe. Otros
    //       motivos de que falle son que el compilador C
    //       no se encuentre para para compilar el procedimiento
    //       o que la colección MYLIBRARY no existe en el sistema.
}
s.close();

// Ahora, utilizar JDBC para ejecutar el procedimiento y obtener los resultados. En
// este caso, vamos a obtener información sobre los procedimientos de'MYLIBRARY'
// (que es también donde creamos este procedimiento, para así
// estar seguros de que haya algo que obtener.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX1");

ResultSet rs = cs.executeQuery();

// Ahora tenemos el primer objeto ResultSet que el procedimiento almacenado
// dejó abierto. Hay que utilizarlo.
int i = 1;
while (rs.next()) {
    System.out.println("Procedimiento almacenado de MYLIBRARY
                       " + i + " es " + rs.getString(1) + "." +
                       rs.getString(2));
    i++;

}
System.out.println("");


// Ahora, obtener el siguiente objeto ResultSet del sistema - el anterior
// se ha cerrado automáticamente.
if (!cs.getMoreResults()) {
    System.out.println("Algo ha fallado. Debería haber
                        habido otro ResultSet, hay que salir.");
    System.exit(0);
}
rs = cs.getResultSet();

 // Ahora tenemos el segundo objeto ResultSet que el procedimiento almacenado
// dejó abierto. Hay que utilizarlo.
```

```
            i = 1;
            while (rs.next()) {
                System.out.println("Procedimiento de MYLIBRARY " + rs.getString(1)
                                    + "." + rs.getString(2) +
                                   " parámetro:  " + rs.getInt(3) + " dirección:
                                   " + rs.getString(4) +
                                   " tipo de datos: " + rs.getString(5));
                i++;

            }

            if (i == 1) {
                System.out.println("Ninguno de los procedimientos almacenados tiene parámetros.");
            }


            if (cs.getMoreResults()) {
                System.out.println("Algo ha fallado,
                                    no debe haber otro ResultSet.");
                System.exit(0);
            }

            cs.close();  // cerrar el objeto CallableStatement.
            c.close();   // cerrar el objeto Connection.

        } catch (Exception e) {
            System.out.println("Algo ha fallado...");
            System.out.println("Razón: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Ejemplo: crear un procedimiento con parámetros de entrada y salida:

En este ejemplo se muestra como acceder a una base de datos utilizando JDBC y luego crear un procedimiento con parámetros de entrada y de salida.

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

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

public class CallableStatementExample2 {

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

        // Registrar el controlador JDBC nativo. Si no podemos
        // registrar el controlador, la prueba no puede continuar.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Crear las propiedades de la conexión
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Conectar con la base de datos del servidor local
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Crear un procedimiento con parámetros de entrada, de salida y de entrada/salida.
            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: Aquí ignoramos el error. Hacemos la
                //       suposición de que el único motivo de que esto falle
                //       se debe a que el procedimiento ya existe. Otros
                //       motivos de que falle son que el compilador C
                //       no se encuentre para para compilar el procedimiento
                //       o que la colección MYLIBRARY no existe en el sistema.
            }
            s.close();

            // Preparar una sentencia a la que puede llamarse para ejecutar
            // el procedimiento.
            CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX2(?, ?, ?)");

            // Todos los parámetros de entrada deben estar establecidos y todos los de salida deben
            // estar registrados. Esto implica que tenemos que hacer dos llamadas
            // para un parámetro de entrada y salida.
            cs.setInt(1, 5);
            cs.setInt(3, 10);
            cs.registerOutParameter(2, Types.INTEGER);
            cs.registerOutParameter(3, Types.INTEGER);

            // Ejecutar el procedimiento
            cs.executeUpdate();

            // Verificar que los parámetros de salida tienen los valores deseados.
            System.out.println("El valor de P2 debe ser P1 (5) + 1 = 6.  -->  " + cs.getInt(2));
            System.out.println("El valor de P3 debe ser P3 (10) + 1 = 11.  -->  " + cs.getInt(3));

            cs.close();  // cerrar el objeto CallableStatement.
            c.close();   // cerrar el objeto Connection.

        } catch (Exception e) {
            System.out.println("Algo ha fallado...");
            System.out.println("Razón: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Ejemplo: crear un procedimiento con valores de retorno:

En este ejemplo se muestra como acceder a una base de datos utilizando JDBC y luego crear un procedimiento con valores de retorno .

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

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

public class CallableStatementExample3 {

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

        // Registrar el controlador JDBC nativo. Si el controlador no puede
        // registrarse, la prueba no puede continuar.
        try {
```

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

            // Crear las propiedades de la conexión
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Conectar con la base de datos del servidor local
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Crear un procedimiento con un valor de retorno.
            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: Aquí se ignora el error. Se presupone que
                //       la única razón de este error es que
                //       el procedimiento ya existe. Otros
                //       motivos de que falle son que el compilador C
                //       no se encuentre para para compilar el procedimiento
                //       o que la colección MYLIBRARY no existe en el sistema.
            }
            s.close();

            // Preparar una sentencia a la que puede llamarse para ejecutar
            // el procedimiento.
            CallableStatement cs = c.prepareCall("? = CALL MYLIBRARY.SQLSPEX3");

            // Aún se tiene que registrar el parámetro de salida.
            cs.registerOutParameter(1, Types.INTEGER);

            // Ejecutar el procedimiento.
            cs.executeUpdate();

            // Mostrar que se devuelve el valor correcto.
            System.out.println("El valor de retorno
                               siempre debe ser 1976 en este ehjemplo:
                               -->  " + cs.getInt(1));

            cs.close();  // cerrar el objeto CallableStatement.
            c.close();   // cerrar el objeto Connection.

        } catch (Exception e) {
            System.out.println("Algo ha fallado...");
            System.out.println("Razón: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

ResultSets

La interfaz ResultSet proporciona acceso a los resultados generados al ejecutar consultas. Conceptualmente, los datos de un ResultSet pueden considerarse como una tabla con un número específico de columnas y un número específico de filas. Por omisión, las filas de la tabla se recuperan por orden. Dentro de una fila, se puede acceder a los valores de columna en cualquier orden.

Características de ResultSet:

Este tema trata de las características de los ResultSets, como son los tipos de ResultSet, la concurrencia, la capacidad para cerrar el ResultSet comprometiendo el objeto conexión, y la especificación de las características de ResultSet.

Por defecto, el tipo de todos los ResultSets creados es solo de reenvío, la concurrencia es solo de lectura y los cursores se retienen en los límites del compromiso. Una excepción de ello la presenta WebSphere, que actualmente cambia el valor predeterminado de la capacidad de retención de cursores para que los cursores se cierren implícitamente al comprometerse. Estas características pueden configurarse mediante los métodos accesibles en objetos Statement, PreparedStatement y CallableStatement.

Tipos de ResultSet

El tipo de un ResultSet especifica los siguiente acerca del ResultSet:

- Si el ResultSet es desplazable.
- Los tipos de los ResultSets de Java Database Connectivity (JDBC) definidos por constantes en la interfaz ResultSet.

Las definiciones de estos tipos de ResultSet son las siguientes:

TYPE_FORWARD_ONLY
: Un cursor que solo puede utilizarse para procesar desde el principio de un ResultSet hasta el final del mismo. Este es el tipo por omisión.

TYPE_SCROLL_INSENSITIVE
: Un cursor que se puede emplear para desplazares a través de un ResultSet. Este tipo de cursor es insensible a los cambios efectuados en la base de datos mientras está abierto. Contiene filas que satisfacen la consulta cuando esta se procesa o cuando se extraen datos.

TYPE_SCROLL_SENSITIVE
: Un cursor que puede utilizarse para el desplazamiento en diversas formas a través de un ResultSet. Este tipo de cursor es sensible a los cambios efectuados en la base de datos mientras está abierto. Los cambios en la base de datos tienen un impacto directo sobre los datos del ResultSet.

Los ResultSets de JDBC 1.0 son siempre solo hacia adelante. Los cursores desplazables se añadieron en JDBC 2.0.

Nota: las propiedades de agrupación por bloques habilitada y de conexión de tamaño de bloque afectan al grado de sensibilidad de un cursor TYPE_SCROLL_SENSITIVE. La agrupación por bloques mejora el rendimiento al almacenar en caché datos de la propia capa del controlador JDBC.

Concurrencia

La concurrencia determina si el ResultSet puede actualizarse. Los tipos se definen de nuevo mediante constantes de la interfaz ResultSet. Los valores de concurrencia disponibles son los siguientes:

CONCUR_READ_ONLY
: Un ResultSet que solo puede utilizarse para leer datos de la base de datos. Este es el valor predeterminado.

CONCUR_UPDATEABLE
: Un ResultSet que permite efectuar cambios en el mismo. Estos cambios pueden colocarse en la base de datos subyacente.

Los ResultSets de JDBC 1.0 son siempre solo hacia adelante. Los ResultSets actualizables se añadieron en JDBC 2.0.

Nota: Según la especificación JDBC, el controlador JDBC puede cambiar el tipo de ResultSet del valor de concurrencia de ResultSet si los valores no pueden utilizarse conjuntamente. En tales casos, el controlador JDBC sitúa un aviso en el objeto Connection.

Existe una situación en la que la aplicación especifica un ResultSet TYPE_SCROLL_INSENSITIVE, CONCUR_UPDATEABLE. La insensibilidad se implementa en el motor de bases de datos efectuando una copia de los datos. A continuación, no se permite al usuario realizar actualizaciones mediante esa copia en la base de datos subyacente. Si especifica esta combinación, el controlador cambia la sensibilidad a TYPE_SCROLL_SENSITIVE y crea el aviso, que indica que la petición se ha cambiado.

Capacidad de retención

La característica de capacidad de retención determina si la llamada al compromiso en el objeto Connection cierra el ResultSet. La API de JDBC destinada a trabajar con la característica de capacidad de retención es nueva en la versión 3.0. Sin embargo, el controlador JDBC nativo ha proporcionado una propiedad de conexión para varios releases que le permite especificar ese valor predeterminado para todos los ResultSets creados bajo la conexión. El soporte de API altera temporalmente cualquier valor de la propiedad de conexión. Los valores de la característica de capacidad de retención se definen mediante constantes de ResultSet y son los siguientes:

HOLD_CURSOR_OVER_COMMIT
Todos los cursores abiertos permanecen así cuando se llama a la cláusula commit. Este es el valor predeterminado del controlador JDBC nativo.

CLOSE_CURSORS_ON_COMMIT
Todos los cursores abiertos se cierran cuando se llama a la cláusula commit.

Nota: Al llamar a la retrotracción en una conexión, siempre se cierran todos los cursores abiertos. Este es un hecho poco conocido, pero es una forma común de que las bases de datos manejen los cursores.

Según la especificación JDBC, el valor predeterminado de la capacidad de retención está definida por la implementación. Algunas plataformas optan por utilizar CLOSE_CURSORS_ON_COMMIT como valor predeterminado. Esto no representa generalmente un problema para la mayoría de las aplicaciones, pero el usuario debe estar al corriente de lo que realiza el controlador que utiliza si está trabajando con cursores en los límites del compromiso. El controlador JDBC de IBM Toolbox para Java también utiliza el valor predeterminado de HOLD_CURSORS_ON_COMMIT, pero el controlador JDBC de UDB para Windows NT tiene el valor predeterminado CLOSE_CURSORS_ON_COMMIT.

Especificar características de ResultSet

Las características de un ResultSet no cambian una vez que se ha creado el objeto ResultSet. Por tanto, las características se han especificado antes de crear el objeto. Puede especificar estas características mediante variantes cargadas a posteriori de los métodos createStatement, prepareStatement y prepareCall.

Nota: Existen métodos de ResultSet para obtener el tipo de ResultSet y la concurrencia del ResultSet, pero no existe ningún método para obtener la capacidad de retención del ResultSet.

Conceptos relacionados

"Objetos Statement" en la página 98
El objeto Statement (sentencia) sirve para procesar una sentencia SQL estática y obtener los resultados producidos por ella. Solo puede haber un ResultSet abierto para cada objeto Statement en un momento dado. Todos los métodos statement que procesan una sentencia SQL cierran implícitamente el ResultSet actual de una sentencia si existe uno abierto.

"CallableStatements" en la página 106
La interfaz CallableStatement de JDBC amplía PreparedStatement y proporciona soporte para parámetros de salida y de entrada/salida. La interfaz CallableStatement tiene también soporte para parámetros de entrada, que proporciona la interfaz PreparedStatement.

"PreparedStatements" en la página 100
Las PreparedStatements amplían la interfaz Statement y proporcionan soporte para añadir parámetros a sentencias SQL.

"Movimiento de cursores" en la página 121
Los controladores Java Database Connectivity (JDBC) de System i admiten ResultSets desplazables. Con un ResultSet desplazable, puede procesar filas de datos en cualquier orden mediante diversos métodos de posicionamiento de cursor.

Tareas relacionadas

"Cambiar ResultSets" en la página 124
Con los controladores JDBC de System i, puede cambiar los ResultSets realizando varias tareas.

Referencia relacionada

"Propiedades de conexión del controlador JDBC" en la página 46
En esta tabla figuran las propiedades válidas de conexión para el controlador JDBC, los valores que tienen y sus descripciones.

"Propiedades de DataSource" en la página 62
Para cada propiedad de conexión de controlador JDBC, existe un correspondiente método de origen de datos. En esta tabla figuran las propiedades válidas de los orígenes de datos.

Ejemplo: ResultSets sensibles e insensibles:

El ejemplo siguiente muestra la diferencia entre los ResultSets sensibles y los insensibles cuando se insertan filas en una tabla.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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) {
                // Ignorado.
            }

            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("Excepción capturada: " + e.getMessage());
            if (e instanceof SQLException) {
                SQLException another = ((SQLException) e).getNextException();
                System.out.println("Otra:  " + another.getMessage());
            }
        }
    }


    public void run(String sensitivity) {
        try {
            Statement s = null;
            if (sensitivity.equalsIgnoreCase("insensitive")) {
                System.out.println("creando un cursor TYPE_SCROLL_INSENSITIVE");
                s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
            } else {
                System.out.println("creando un cursor TYPE_SCROLL_SENSITIVE");
                s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
            }

            ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

            // Captar los cinco valores que se encuentran allí.
            rs.next();
            System.out.println("el valor es " + rs.getInt(1));
            rs.next();
            System.out.println("el valor es " + rs.getInt(1));
            rs.next();
            System.out.println("el valor es " + rs.getInt(1));
            rs.next();
            System.out.println("el valor es " + rs.getInt(1));
            rs.next();
            System.out.println("el valor es " + rs.getInt(1));
            System.out.println("captadas las cinco filas...");

            // Nota:  Si capta la última fila, el ResultSet interpreta
            //        que las filas cerradas y las nuevas que se añadan
            //        no se reconocen.

            // Permitir que otra sentencia inserte un valor nuevo.
            Statement s2 = connection.createStatement();
            s2.executeUpdate("insert into cujosql.sensitive values(6)");
            s2.close();

            // El hecho de que una fila se reconozca se basa en el valor
            // de sensibilidad.
            if (rs.next()) {
                System.out.println("Ahora existe una fila: " + rs.getInt(1));
            } else {
                System.out.println("No hay más filas.");
            }
```

```
        } catch (SQLException e) {
            System.out.println("Excepción SQLException: ");
            System.out.println("Mensaje:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Código proveedor:." + e.getErrorCode());
            System.out.println("-----------------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("Se ha lanzado una excepción que no es una SQLException: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Ejemplo: sensibilidad de ResultSet:

El ejemplo siguiente muestra cómo un cambio puede afectar a una cláusula where de una sentencia SQL en función de la sensibilidad del ResultSet.

Puede que el formato de este ejemplo no sea del todo correcto como consecuencia de haber adaptado este ejemplo a una página impresa.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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("Se utiliza controlador JDBC nativo");
            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) {
            // Ignorado.
        }

        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) {
            // Ignorado.
        }

        s.executeUpdate("create table cujosql.sensitive2(col2 int)");
        s.executeUpdate("insert into cujosql.sensitive2 values(1)");
        s.executeUpdate("insert into cujosql.sensitive2 values(2)");
        s.executeUpdate("insert into cujosql.sensitive2 values(3)");
        s.executeUpdate("insert into cujosql.sensitive2 values(4)");
        s.executeUpdate("insert into cujosql.sensitive2 values(5)");

        s.close();

    } catch (Exception e) {
        System.out.println("Excepción capturada: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Otra:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creando cursor TYPE_SCROLL_INSENSITIVE");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creando cursor TYPE_SCROLL_SENSITIVE");
            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("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        rs.next();
```

```
            System.out.println("el valor es " + rs.getInt(1));
            rs.next();
            System.out.println("el valor es " + rs.getInt(1));

            System.out.println("captadas las cuatro filas...");

            // Otra sentencia crea un valor que no se ajusta a la cláusula 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("Todavía hay una fila: " + rs.getInt(1));
            } else {
                System.out.println("No hay más filas.");
            }

        } catch (SQLException e) {
            System.out.println("Excepción SQLException: ");
            System.out.println("Mensaje:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Código proveedor:." + e.getErrorCode());
            System.out.println("----------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("Se ha lanzado una excepción que no es una SQLException: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Movimiento de cursores:

Los controladores Java Database Connectivity (JDBC) de System i admiten ResultSets desplazables. Con un ResultSet desplazable, puede procesar filas de datos en cualquier orden mediante diversos métodos de posicionamiento de cursor.

El método ResultSet.next se utiliza para desplazarse por filas en un ResultSet de una en una. Con Java Database Connectivity (JDBC) 2.0, los controladores JDBC de System i admiten ResultSets desplazables. Los ResultSets desplazables permiten procesar las filas de datos en cualquier orden mediante los métodos previous, absolute, relative, first y last.

Por omisión, los ResultSets JDBC son siempre solo hacia adelante, lo cual significa que el único método de posicionamiento de cursor válido al que puede llamarse es next(). Un ResultSet desplazable debe solicitarse explícitamente. Consulte la sección Tipo de ResultSet para obtener más información.

Con un ResultSet desplazable, puede utilizar los siguientes métodos de posicionamiento de cursor:

Método	**Descripción**
Next	Este método adelanta el cursor una fila del ResultSet. El método devuelve true si el cursor está situado en una fila válida, y false si no es así.
Previous	El método hace retroceder el cursor una fila del ResultSet. El método devuelve true si el cursor está situado en una fila válida, y false si no es así.
First	El método mueve el cursor a la primera fila del ResultSet. El método devuelve true si el cursor está situado en la primera fila, y false si el ResultSet está vacío.
Last	El método mueve el cursor a la última fila del ResultSet. El método devuelve true si el cursor está situado en la última fila, y false si el ResultSet está vacío.
BeforeFirst	El método mueve el cursor inmediatamente antes de la primera fila del ResultSet. En un ResultSet vacío, este método no tiene ningún efecto. No existe ningún valor de retorno de este método.
AfterLast	El método mueve el cursor inmediatamente después de la última fila del ResultSet. En un ResultSet vacío, este método no tiene ningún efecto. No existe ningún valor de retorno de este método.
Relative (int rows)	El método mueve el cursor en relación a su posición actual. • Si rows es 0, este método no tiene ningún efecto. • Si rows es positive, el cursor se mueve hacia adelante el número de filas indicado. Si entre la posición actual y el final del ResultSet existen menos filas que las indicadas por los parámetros de entrada, este método opera igual que afterLast. • Si rows es negative, el cursor se mueve hacia atrás el número de filas indicado. Si entre la posición actual y el final del ResultSet existen menos filas que las indicadas por el parámetro de entrada, este método opera igual que beforeFirst. El método devuelve true si el cursor está situado en una fila válida, y false si no es así.
Absolute (int row)	El método mueve el cursor a la fila especificada por el valor de fila (row). Si el valor de fila es positivo, el cursor se coloca en el número de filas indicado a partir del principio de ResultSet. El número de la primera fila es 1, el de la segunda es 2, etc. Si en el ResultSet existen menos filas que las especificadas por el valor de fila, este método opera igual que afterLast. Si el valor de fila es negativo, el cursor se coloca en el número de filas indicado a partir del final de ResultSet. El número de la última fila es -1, el de la penúltima es -2, etc. Si en el ResultSet existen menos filas que las especificadas por el valor de fila, este método opera igual que beforeLast. Si el valor de fila es 0, este método opera igual que beforeFirst. El método devuelve true si el cursor está situado en una fila válida, y false si no es así.

Recuperar datos de ResultSet:

El objeto ResultSet proporciona varios métodos para obtener los datos de columna correspondientes a un fila. Todos ellos tienen el formato get<*Tipo*>, siendo <*Tipo*> un tipo de datos Java. Algunos ejemplos de estos métodos son getInt, getLong, getString, getTimestamp y getBlob. Casi todos estos métodos toman un solo parámetro, que es el índice que la columna tiene dentro del ResultSet o bien el nombre de la columna.

Las columnas de ResultSet están numeradas, empezando por el 1. Si se emplea el nombre de la columna y hay más de una columna que tenga ese mismo nombre en el ResultSet, se devuelve la primera. Algunos de los métodos get<Tipo> tienen parámetros adicionales, como el objeto opcional Calendar, que se puede pasar a los métodos getTime, getDate y getTimestamp. Consulte el Javadoc del paquete java.sql para obtener todos los detalles.

En los métodos get que devuelven objetos, el valor de retorno es null cuando la columna del ResultSet es nula. En tipos primitivos, no puede devolverse null. En estos casos, el valor es 0 o false. Si una aplicación debe distinguir entre null, y 0 o false, puede utilizarse el método wasNull inmediatamente después de la llamada. A continuación, este método puede determinar si el valor era un valor 0 o false real o si ese valor se ha devuelto debido a que el valor de ResultSet era de hecho null.

Soporte de ResultSetMetaData

Cuando se llama al método getMetaData en un objeto ResultSet, el método devuelve un objeto ResultSetMetaData que describe las columnas de ese objeto ResultSet. En los casos en que la sentencia SQL que se va a procesar no se conoce hasta el momento de la ejecución, puede utilizarse ResultSetMetaData para determinar cuál de los métodos get hay que emplear para recuperar los datos. El ejemplo de código siguiente utiliza ResultSetMetaData para determinar cada uno de los tipos de columna del conjunto de resultados.

```
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);
```

Ejemplo: interfaz ResultSetMetaData de IBM Developer Kit para Java:

Este programa hace una demostración de cómo utilizar un ResultSetMetaData y un ResultSet para visualizar todos los metadatos de ResultSet creado al consultar una tabla. El usuario pasa el valor de la tabla y la biblioteca.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.sql.*;

/**
ResultSetMetaDataExample.java

Este programa muestra la utilización de ResultSetMetaData y
ResultSet para visualizar todos los metadatos relacionados con un ResultSet
creado al consultar una tabla. El usuario pasa el valor correspondiente a
la tabla y a la biblioteca.
**/
public class ResultSetMetaDataExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Uso:  java ResultSetMetaDataExample <biblioteca> <tabla>");
            System.out.println("siendo <biblioteca> la biblioteca que contiene la <tabla>");
            System.exit(0);
        }
```

```
        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Obtener una conexión a base de datos y preparar una sentencia.
            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("Información acerca de la columna " + i);
                System.out.println("   Nombre........: " + rsmd.getColumnName(i));
                System.out.println("   Tipo de datos.....: " + rsmd.getColumnType(i) +
                                   " ( " + rsmd.getColumnTypeName(i) + " )");
                System.out.println("   Precisión.....: " + rsmd.getPrecision(i));
                System.out.println("   Escala........: " + rsmd.getScale(i));
                System.out.print  ("   Permitir nulos: ");
                if (rsmd.isNullable(i)==0)
                    System.out.println("false");
                else
                    System.out.println("true");
            }

        } catch (Exception e) {
            // Manejar los errores.
            System.out.println("Tenemos un error... ");
            e.printStackTrace();
        } finally {
            // Hay que asegurarse de que siempre se haga
            // el borrado. Si la conexión se cierra, la
            // sentencia que hay debajo de ella también se cerrará.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Error grave: no se puede cerrar el objeto conexión");
                }
            }
        }
    }
}
```

Cambiar ResultSets:

Con los controladores JDBC de System i, puede cambiar los ResultSets realizando varias tareas.

El valor predeterminado de los ResultSets es solo de lectura. Sin embargo, con Java Database Connectivity (JDBC) 2.0, los controladores JDBC de System i proporcionan soporte completo para los ResultSets actualizables.

Para saber cómo se actualizan los ResultSets, consulte: “Características de ResultSet” en la página 114.

Actualizar filas

Pueden actualizarse las filas de una tabla de base de datos mediante la interfaz ResultSet. Los pasos que implica este proceso son los siguientes:

1. Cambie los valores de una fila concreta utilizando los diversos métodos update<*Tipo*>, siendo <*Tipo*> un tipo de datos Java. Estos métodos update<*Tipo*> se corresponden con los métodos get<*Tipo*> disponibles para recuperar valores.
2. Aplique las filas a la base de datos subyacente.

La propia base de datos no se actualiza hasta el segundo paso. Si se actualizan columnas de un ResultSet sin llamar al método updateRow, no se realizan cambios en la base de datos.

Las actualizaciones planificadas en una fila pueden eliminarse con el método cancelUpdates. Una vez que se ha llamado al método updateRow, los cambios de la base de datos son finales y no pueden deshacerse.

Nota: El método rowUpdated siempre devuelve false, ya que la base de datos no tiene forma alguna de señalar qué filas se han actualizado. En consecuencia, el método updatesAreDetected devuelve false.

Suprimir filas

Pueden suprimirse las filas de una tabla de base de datos mediante la interfaz ResultSet. Se suministra el método deleteRow, que suprime la fila actual.

Insertar filas

Pueden insertarse filas en una tabla de base de datos mediante la interfaz ResultSet. Este proceso utiliza una operación ″insert row″ (insertar fila), a la que las aplicaciones mueven específicamente el cursor y crean los valores que desean insertar en la base de datos. Los pasos que implica este proceso son los siguientes:

1. Sitúe el cursor en la operación insert row.
2. Establezca cada uno de los valores para las columnas de la fila nueva.
3. Inserte la fila en la base de datos y, opcionalmente, mueva el cursor de nuevo a la fila actual de ResultSet.

Nota: No se insertan filas nuevas en la tabla en la que está situado el cursor. Generalmente, se añaden al final del espacio de datos de la tabla. Por omisión, una base de datos relacional no depende de la posición. Por ejemplo, no debe esperar mover el cursor a la tercera fila e insertar algo que aparezca antes de la cuarta fila si usuarios subsiguientes extraen los datos.

Soporte para actualizaciones posicionadas

Además del método destinado a actualizar la base de datos mediante un ResultSet, pueden utilizarse sentencias SQL para emitir actualizaciones posicionadas. Este soporte se basa en la utilización de cursores con nombre. JDBC suministra el método setCursorName de Statement y el método getCursorName de ResultSet para proporcionar acceso a estos valores.

Dos métodos de DatabaseMetaData, supportsPositionedUpdated y supportsPositionedDelete, devuelven true, ya que esta característica está soportada por el controlador JDBC nativo.

Ejemplo: eliminar valores de una tabla mediante el cursor de otra sentencia:

Este ejemplo Java enseña a eliminar valores de una tabla mediante el cursor de otra sentencia.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
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();
    }



/**
Manejar todo el trabajo de configuración necesario.
**/
    public void setup() {
        try {
            // Registrar el controlador 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) {
                // Aquí, pasar por alto los problemas.
            }

            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("Excepción capturada: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
En esta sección, debe añadirse todo el código destinado a realizar
la prueba. Si solo se necesita una conexión con la base de datos,
se puede utilizar la variable global 'connection'.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Actualizar cada valor utilizando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("El nombre de cursor es " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement
                                      ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
                                      rs.getCursorName ());
```

```
            // Repetir en bucle el ResultSet y actualizar las demás entradas.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Borrar los recursos una vez utilizados.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
En esta sección, colocar todo el trabajo de borrado para la prueba.
**/
    public void cleanup() {
        try {
            // Cerrar la conexión global abierta en setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
Visualizar el contenido de la tabla.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Índice " + rs.getInt(1) + " valor " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Ejemplo: cambiar valores con una sentencia mediante el cursor de otra sentencia:

Este ejemplo Java enseña a cambiar valores con una sentencia mediante el cursor de otra sentencia.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }



/**
Manejar todo el trabajo de configuración necesario.
**/
    public void setup() {
        try {
            // Registrar el controlador 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) {
                // Aquí, pasar por alto los problemas.
            }

            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("Excepción capturada: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
En esta sección, debe añadirse todo el código destinado a realizar
la prueba. Si solo se necesita una conexión con la base de datos,
se puede utilizar la variable global 'connection'.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Actualizar cada valor utilizando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
```

```
                                        "FOR UPDATE OF COL_VALUE");

            System.out.println("El nombre de cursor es " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
                                                                   + " CUJOSQL.WHERECUREX
                                                                       SET COL_VALUE = 'CHANGED'
                                                                       WHERE CURRENT OF "
                                                                   + rs.getCursorName ());


            // Repetir en bucle el ResultSet y actualizar las demás entradas.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Borrar los recursos una vez utilizados.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
En esta sección, colocar todo el trabajo de borrado para la prueba.
**/
    public void cleanup() {
        try {
            // Cerrar la conexión global abierta en setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
Visualizar el contenido de la tabla.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Índice " + rs.getInt(1) + " valor " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
```

```
            e.printStackTrace();
        }
    }
}
```

Crear ResultSets:

Para crear un objeto ResultSet, puede utilizar los métodos executeQuery u otros métodos. En este tema se describen las opciones para crear ResultSets.

Estos métodos proceden de las interfaces Statement, PreparedStatement o CallableStatement. Sin embargo, existen otros métodos disponibles. Por ejemplo, los métodos de DatabaseMetaData como getColumns, getTables, getUDTs, getPrimaryKeys, etcétera, devuelven ResultSets. También es posible que una sola sentencia SQL devuelva varios ResultSets para el proceso. También puede utilizar el método getResultSet para recuperar un objeto ResultSet después de llamar al método execute suministrado por las interfaces Statement, PreparedStatement o CallableStatement.

Consulte “Ejemplo: crear un procedimiento con múltiples ResultSets” en la página 110 para obtener más información.

Cerrar ResultSets

Aunque un objeto ResultSet se cierra automáticamente cuando se cierra el objeto Statement con el que está asociado, es aconsejable cerrar los objetos ResultSet cuando haya terminado de utilizarlos. Al hacerlo, liberará inmediatamente los recursos internos de base de datos que pueden aumentar el rendimiento de las aplicaciones.

También es importante cerrar los objetos ResultSet generados por llamadas a DatabaseMetaData. Debido a que el usuario no tiene acceso directo al objeto Statement utilizado para crear estos ResultSets, no se llama directamente a close en el objeto Statement. Estos objetos están enlazados conjuntamente de tal forma que el controlador JDBC cierra el objeto Statement interno cuando el usuario cierra el objeto ResultSet externo. Si estos objetos no se cierran manualmente, el sistema sigue en funcionamiento; sin embargo, utiliza más recursos de los necesarios.

Nota: La característica de capacidad de retención de ResultSets puede cerrar también los ResultSets automáticamente en nombre del usuario. Se permite llamar a close varias veces en un objeto ResultSet.

“Objetos Statement” en la página 98
El objeto Statement (sentencia) sirve para procesar una sentencia SQL estática y obtener los resultados producidos por ella. Solo puede haber un ResultSet abierto para cada objeto Statement en un momento dado. Todos los métodos statement que procesan una sentencia SQL cierran implícitamente el ResultSet actual de una sentencia si existe uno abierto.

“PreparedStatements” en la página 100
Las PreparedStatements amplían la interfaz Statement y proporcionan soporte para añadir parámetros a sentencias SQL.

“CallableStatements” en la página 106
La interfaz CallableStatement de JDBC amplía PreparedStatement y proporciona soporte para parámetros de salida y de entrada/salida. La interfaz CallableStatement tiene también soporte para parámetros de entrada, que proporciona la interfaz PreparedStatement.

“Interfaz DatabaseMetaData de IBM Developer Kit para Java” en la página 65
El controlador JDBC de IBM Developer Kit para Java implementa la interfaz DatabaseMetaData para proporcionar información sobre los orígenes de datos subyacentes. La utilizan principalmente los servidores de aplicaciones y las herramientas para determinar cómo hay que interaccionar con un origen de datos dado. Las aplicaciones también pueden servirse de los métodos de DatabaseMetaData para obtener información sobre un origen de datos, pero esto ocurre con menos frecuencia.

Ejemplo: interfaz ResultSet de IBM Developer Kit para Java:

Este es un ejemplo de cómo utilizar la interfaz ResultSet.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.sql.*;

/**
ResultSetExample.java

Este programa muestra la utilización de ResultSetMetaData y
ResultSet para visualizar todos los datos de una tabla aunque
el programa que obtiene los datos no sabe cuál es el aspecto
que tendrá la tabla (el usuario pasa los valores correspondientes
a la tabla y a la biblioteca).
**/
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Uso:  java ResultSetExample <biblioteca> <tabla>");
            System.out.println(" siendo <biblioteca> la biblioteca que contiene la <tabla>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Obtener una conexión a base de datos y preparar una sentencia.
            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("Datos para la fila " + rowCount);
                for (int i = 1; i <= colCount; i++)
                    System.out.println("   Fila " + i + ": " + rs.getString(i));
            }

        } catch (Exception e) {
            // Manejar los errores.
            System.out.println("Tenemos un error... ");
            e.printStackTrace();
        } finally {
            // Hay que asegurarse de que siempre se haga
            // el borrado. Si la conexión se cierra, la
            // sentencia que hay debajo de ella también se cerrará.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Error grave: no se puede cerrar el objeto conexión");
                }
```

```
            }
        }
    }
}
```

Agrupación de objetos JDBC

La agrupación de objetos es una importante consideración a tener en cuenta para Java Database Connectivity (JDBC) y para el rendimiento. Debido a que la creación de muchos objetos utilizados en JDBC es costosa, como por ejemplo objetos Connection, Statement y ResultSet, pueden obtenerse ventajas significativas de rendimiento reutilizando estos objetos en lugar de crearlos cada vez que son necesarios.

Muchas aplicaciones ya manejan la agrupación de objetos por su cuenta. Por ejemplo, WebSphere tiene un amplio soporte para agrupar objetos JDBC, y le permite controlar cómo se gestiona la agrupación. Debido a ello, el usuario puede obtener las funciones que desee sin necesidad de preocuparse de sus propios mecanismos de agrupación. Sin embargo, en las ocasiones en que no se suministra soporte, es necesario encontrar una solución para todas las aplicaciones, excepto las triviales.

Utilizar soporte de DataSource para la agrupación de objetos:

Puede utilizar DataSources para que varias aplicaciones compartan una configuración común para acceder a una base de datos. Esta operación se realiza haciendo que cada una de las aplicaciones haga referencia al mismo nombre de DataSource.

Mediante la utilización de DataSources, pueden cambiarse muchas aplicaciones desde una ubicación central. Por ejemplo, si cambia el nombre de una biblioteca por omisión utilizada por todas las aplicaciones y ha utilizado un solo DataSource para obtener conexiones para todas ellas, puede actualizar el nombre de la colección en ese dataSource. A continuación, todas las aplicaciones empezarán a utilizar la nueva biblioteca por omisión.

Al utilizar DataSources para obtener conexiones para una aplicación, puede utilizar el soporte incorporado del controlador JDBC nativo para la agrupación de conexiones. Este soporte se suministra como implementación de la interfaz ConnectionPoolDataSource.

La agrupación se realiza pasando objetos lógicos Connection en lugar de objetos físicos Connection. Un **objeto lógico Connection** es un objeto de conexión devuelto por un objeto Connection de la agrupación. Cada objeto lógico de conexión actúa como handle temporal para la conexión física representada por el objeto de conexión de la agrupación. Con respecto a la aplicación, cuando se devuelve el objeto Connection no existe ninguna diferencia apreciable entre los dos. La ligera diferencia surge cuando se llama el método close en el objeto Connection. Esta llamada invalida la conexión lógica y devuelve la conexión física a la agrupación, donde otra aplicación puede utilizarla. Esta técnica permite que muchos objetos de conexión lógica reutilicen un solo objeto de conexión física.

Configurar la agrupación de conexiones

La agrupación de conexiones se realiza creando un objeto DataSource que hace referencia a un objeto ConnectionPoolDataSource. Los objetos ConnectionPoolDataSource tienen propiedades que pueden establecerse para manejar diversos aspectos del mantenimiento de la agrupación.

Encontrará más detalles en el ejemplo de cómo configurar la agrupación de conexiones con UDBDataSource y UDBConnectionPoolDataSource. También puede ver la interfaz Java Naming and Directory Interface (JNDI) si desea obtener detalles sobre el papel que desempeña JNDI en este ejemplo.

En el ejemplo, el enlace que conecta los dos objetos DataSource es dataSourceName. El enlace indica al objeto DataSource que retrase el establecimiento de las conexiones con el objeto ConnectionPoolDataSource que gestiona la agrupación automáticamente.

Aplicaciones con agrupación y sin agrupación

No existe ninguna diferencia entre una aplicación que utiliza la agrupación de conexiones y una que no lo hace. Por tanto, el soporte de agrupación puede añadirse una vez completado el código de la aplicación, sin efectuar cambios en el mismo.

A continuación se muestra la salida de la ejecución local del programa anterior durante el desarrollo.

Iniciar temporización de la versión de DataSource sin agrupación...tiempo invertido: 6410

Iniciar temporización de la versión con agrupación...tiempo invertido: 282

Programa Java completado.

Por omisión, un objeto UDBConnectionPoolDataSource agrupa una sola conexión. Si una aplicación necesita una conexión varias veces y solo necesita una conexión a la vez, la utilización de UDBConnectionPoolDataSource es una solución perfecta. Si necesita muchas conexiones simultáneas, debe configurar la ConnectionPoolDataSource ("Propiedades de ConnectionPoolDataSource" en la página 134) para que se ajuste a sus necesidades y recursos.

Conceptos relacionados

"Java Naming and Directory Interface" en la página 581
Java Naming and Directory Interface (JNDI) forma parte de la interfaz de programa de aplicación(API) de la plataforma JavaSoft. Gracias a JNDI, se pueden establecer conexiones sin fisuras con varios servicios de directorio y denominación. Con esta interfaz, se pueden construir aplicaciones Java habilitadas para directorio que sean potentes y portables.

Referencia relacionada

"Ejemplo: configurar una agrupación de conexiones con UDBDataSource y UDBConnectionPoolDataSource"
Este es un ejemplo de cómo utilizar una agrupación de conexiones con UDBDataSource y UDBConnectionPoolDataSource.

"Propiedades de ConnectionPoolDataSource" en la página 134
Puede configurar la interfaz ConnectionPoolDataSource utilizando el conjunto de propiedades que proporciona.

Ejemplo: configurar una agrupación de conexiones con UDBDataSource y UDBConnectionPoolDataSource:

Este es un ejemplo de cómo utilizar una agrupación de conexiones con UDBDataSource y UDBConnectionPoolDataSource.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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
    {
        // Crear una implementación de ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Objeto DataSource de agrupación de conexiones");

        // Establecer un contexto JNDI y enlazar el origen de datos de agrupación
        // de conexiones
```

```
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Crear un origen de datos estándar que haga referencia al mismo.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource que soporta la agrupación");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
  }
}
```

Ejemplo: probar el rendimiento de una agrupación de conexiones:

Este es un ejemplo de cómo probar el rendimiento del ejemplo de agrupación en comparación con el rendimiento del ejemplo sin agrupación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
        // Realizar el trabajo sin una agrupación:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("\nIniciar temporización de versión de DataSource sin agrupación...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Tiempo transcurrido: " + (endTime - startTime));

        // Realizar el trabajo con agrupación:
        ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nIniciar temporización de la versión con agrupación...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Tiempo transcurrido: " + (endTime - startTime));
    }
}
```

Propiedades de ConnectionPoolDataSource:

Puede configurar la interfaz ConnectionPoolDataSource utilizando el conjunto de propiedades que proporciona.

En la tabla siguiente figuran las descripciones de estas propiedades.

Propiedad	Descripción
initialPoolSize	Cuando se crea la primera instancia de la agrupación, esta propiedad determina cuántas conexiones se colocan en la agrupación. Si este valor se especifica fuera del rango de minPoolSize y maxPoolSize, se utiliza minPoolSize o maxPoolSize como número inicial de conexiones que deben crearse.
maxPoolSize	A medida que se utiliza la agrupación, pueden solicitarse más conexiones que las que se encuentran en la agrupación. Esta propiedad especifica el número máximo de conexiones permitidas que pueden crearse en la agrupación. Las aplicaciones no se ″bloquean″ esperando a que se devuelva una conexión a la agrupación cuando esta se encuentra en el tamaño máximo y todas las conexiones se están utilizando. En lugar de ello, el controlador JDBC crea una conexión nueva basada en las propiedades de DataSource y devuelve la conexión. Si se especifica un valor 0 en maxPoolSize, se permite que la agrupación crezca ilimitadamente siempre y cuando el sistema tenga recursos disponibles.
minPoolSize	Los picos de utilización de la agrupación pueden provocar que aumente el número de conexiones que se encuentran en ella. Si el nivel de actividad disminuye hasta el punto que algunas conexiones nunca se extraen de la agrupación, los recursos se están ocupando sin ninguna razón. En tales casos, el controlador JDBC tiene la capacidad de liberar algunas de las conexiones que ha acumulado. Esta propiedad permite indicar a JDBC que libere conexiones, asegurando que siempre exista un número determinado de conexiones disponibles para el uso. Si se especifica un valor 0 en minPoolSize, es posible que la agrupación libere todas sus conexiones y que la aplicación ″pague″ realmente por el tiempo de conexión en cada petición de conexión.
maxIdleTime	Las conexiones realizan el seguimiento del tiempo que han permanecido en la agrupación sin que se las utilice. Esta propiedad especifica el tiempo durante el que una aplicación permite que las conexiones permanezcan sin utilizar antes de liberarlas (es decir, existan más conexiones de las necesarias). Esta propiedad expresa un tiempo en segundos y no especifica cuándo se produce el cierre real. Especifica cuándo ha pasado el tiempo suficiente para que la conexión se libere.
propertyCycle	Esta propiedad representa el número de segundos que pueden transcurrir entre la entrada en vigor de estas normas.

Nota: Si se establece en 0 el tiempo de maxIdleTime o propertyCycle, significa que el controlador JDBC no comprueba que las conexiones se eliminen de la agrupación por su cuenta. Las normas especificadas para el tamaño inicial, mínimo y máximo siguen en vigor.

Si maxIdleTime y propertyCycle no son 0, se utiliza una hebra de gestión para efectuar la observación de la agrupación. La hebra está a la escucha de cada segundo de propertyCycle y comprueba todas las conexiones de la agrupación para ver cuáles han permanecido en ella sin utilizarse durante más segundos que los indicados en maxIdleTime. Las conexiones que se ajustan a estos criterios se eliminan de la agrupación hasta que se alcanza el valor indicado en minPoolSize.

Agrupación de sentencias basada en DataSource:

La propiedad maxStatements, disponible en la interfaz UDBConnectionPoolDataSource, permite la agrupación de sentencias dentro de la agrupación de conexiones. La agrupación de sentencias solo tiene efecto sobre PreparedStatements y CallableStatements. Los objetos Statement no se agrupan.

La implementación de la agrupación de sentencias es parecida a la de la agrupación de conexiones. Cuando la aplicación llama a Connection.prepareStatement("select * from tablex"), el módulo de agrupación comprueba si el objeto Statement ya se ha preparado bajo la conexión. Si es así, se pasa al usuario un objeto lógico PreparedStatement en lugar del objeto físico. Al llamar al método close, el objeto Connection se devuelve a la agrupación, el objeto lógico Connection se elimina y el objeto Statement puede reutilizarse.

La propiedad maxStatements permite a DataSource especificar cuántas sentencias pueden agruparse en una conexión. El valor 0 indica que no debe utilizarse la agrupación de sentencias. Cuando la agrupación de sentencias está llena, se aplica un algoritmo de menor utilización reciente para determinar qué sentencia debe eliminarse.

En el ejemplo que sigue se prueba un DataSource que solo utiliza la agrupación de conexiones y otro DataSource que utiliza la agrupación de sentencias y de conexiones.

El ejemplo siguiente muestra la salida de la ejecución local de este programa durante el desarrollo.

Desplegando origen de datos de agrupación de sentencia Iniciar temporización de la única versión de agrupación de conexiones...Tiempo invertido: 26312

Iniciando temporización de la versión de agrupación de sentencias... Tiempo invertido: 2292 El programa Java se ha completado

Ejemplo: probar el rendimiento de dos orígenes de datos (objeto DataSource):

En este ejemplo se ve cómo probar un DataSource que solo utiliza la agrupación de conexiones y otro DataSource que utiliza la agrupación de sentencias y conexiones.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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("desplegando origen de datos de agrupación de sentencias");
        deployStatementPoolDataSource();

        // Realizar el trabajo solo con la agrupación de conexiones.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nIniciar temporización de la versión solo de agrupación de conexiones...");

        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("Tiempo transcurrido: " + (endTime - startTime));


        // Realizar el trabajo añadiendo la agrupación de sentencias.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("\nIniciar temporización de la versión de agrupación de sentencias...");

        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("Tiempo transcurrido: " + (endTime - startTime));
    }



    private static void deployStatementPoolDataSource()
    throws Exception
    {
        // Crear una implementación de ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Objeto DataSource de agrupación de conexiones con agrupación de sentencias");
        cpds.setMaxStatements(10);

        // Establecer un contexto JNDI y enlazar el origen de datos de agrupación
        // de conexiones
        Context ctx = new InitialContext();
        ctx.rebind("StatementSupport", cpds);


        // Crear un datasource estándar que haga referencia a él.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource que soporta la agrupación de sentencias");
        ds.setDataSourceName("StatementSupport");
        ctx.rebind("StatementPoolingDataSource", ds);

    }
}
```

Construir una agrupación de conexiones propia:

Puede desarrollar su propia agrupación de conexiones y sentencias sin necesidad de contar con soporte para DataSources ni basarse en otro producto.

Sin agrupación de conexiones, se produce demasiado trabajo en la base de datos para cada petición. Es decir, se obtiene una conexión, se obtiene una sentencia, se procesa la sentencia, se cierra la sentencia y se

cierra la conexión. En lugar de descartar todos los elementos después de cada petición, existe una forma de reutilizar algunas partes de este proceso. La **agrupación de conexiones** sustituye el código de crear conexión por código destinado a obtener una conexión de la agrupación y luego sustituye el código cerrar conexión por código destinado a devolver la conexión a la agrupación para reutilizarla.

El constructor de la agrupación de conexiones crea las conexiones y las coloca en la agrupación. La clase de agrupación contiene métodos take y put para localizar una conexión que debe utilizarse y para devolver la conexión a la agrupación cuando se ha terminado de trabajar con ella. Estos métodos están sincronizados debido a que el objeto agrupación es un recurso compartido, pero no conviene que varias hebras intenten manipular simultáneamente los recursos agrupados.

Construir una agrupación de sentencias propia

Al utilizar la agrupación de conexiones, se emplea tiempo en la creación y cierre de la sentencia cuando se proceso cada una de ellas. Este es un ejemplo de desperdicio de un objeto que puede se reutilizar.

Para reutilizar un objeto, puede utilizar la clase de sentencia preparada. En la mayoría de aplicaciones, las mismas sentencias SQL se reutilizan con cambios secundarios. Por ejemplo, una iteración a través de una aplicación puede generar la consulta siguiente:

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

La próxima iteración puede generar la consulta siguiente:

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

Se trata de la misma consulta, pero utiliza un parámetro diferente. Ambas consultas pueden realizarse con la consulta siguiente:

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

A continuación, puede establecer el marcador de parámetro (indicado por el signo de interrogación) en 100000 al procesar la primera consulta y en 50000 al procesar la segunda. Con ello mejorará el rendimiento por tres razones, más allá de lo que la agrupación de conexiones puede ofrecer:

- Se crean menos objetos. Se crea un objeto PreparedStatement y se reutiliza, en lugar de crear un objeto Statement para cada petición. Por tanto, se ejecutan menos constructores.
- El trabajo de la base de datos destinado a definir la sentencia SQL (denominado **preparación**) puede reutilizarse. La preparación de sentencias SQL es considerablemente costosa, ya que implica determinar lo que indica el texto de la sentencia SQL y la forma en que el sistema debe realizar la tarea solicitada.
- Al eliminar las creaciones de objetos adicionales, se produce una ventaja que con frecuencia no se tiene en cuenta. No es necesario destruir lo que no se ha creado. Este modelo facilita la función del recolector de basura Java y también aumenta el rendimiento a lo largo de tiempo cuando existen muchos usuarios.

Consideraciones

El rendimiento mejora mediante la replicación. Si un elemento no se reutiliza, el hecho de colocarlo en la agrupación significa malgastar recursos.

La mayoría de las aplicaciones contienen secciones de código de vital importancia. Generalmente, una aplicación utiliza entre el 80 y el 90 por ciento de su tiempo de proceso en solo entre el 10 y el 20 por ciento del código. Si en una aplicación pueden utilizarse potencialmente 10.000 sentencias SQL, no todas ellas se colocan en una agrupación. El objetivo es identificar y agrupar las sentencias SQL que se utilizan en las secciones de código de la aplicación que son de vital importancia.

La creación de objetos en una implementación Java puede conllevar un importante coste. La solución de agrupación puede utilizarse ventajosamente. Los objetos utilizados en el proceso se crean al principio, antes de que otros usuarios intenten utilizar el sistema. Estos objetos se reutilizan con la frecuencia que

sea necesaria. El rendimiento es excelente, y es posible ajustar el rendimiento de la aplicación a lo largo del tiempo para facilitar su utilización por parte de un número mayor de usuarios. Como resultado, se agrupa un mayor número de objetos. Más aún, permite una ejecución multihebra más eficaz del acceso a base de datos de la aplicación para mejorar el rendimiento.

Java (mediante JDBC) se basa en SQL dinámico y tiende a ser lento. La agrupación puede minimizar este problema. Al preparar las sentencias durante el inicio, al acceso a la base de datos puede convertirse en estático. Una vez preparada la sentencia, existe poca diferencia en cuanto al rendimiento entre el SQL estático y el dinámico.

El rendimiento del acceso a bases de datos en Java puede ser eficaz y puede realizarse sin sacrificar el diseño orientado a objetos o la capacidad de mantenimiento del código. Escribir código para crear una agrupación de sentencias y conexiones no es difícil. Además, el código puede cambiarse y mejorarse para dar soporte a varias aplicaciones y tipos de aplicaciones (basadas en la Web, cliente/servidor), etc.

Actualizaciones por lotes

El soporte de actualización por lotes permite pasar las actualizaciones de la base de datos como una sola transacción entre el programa de usuario y la base de datos. Este procedimiento puede mejorar significativamente el rendimiento cuando deben realizarse muchas actualizaciones simultáneamente.

Por ejemplo, si una empresa grande necesita que sus nuevos empleados empiecen a trabajar un Lunes, este requisito hace necesario procesar muchas actualizaciones (en este caso, inserciones) en la base de datos de empleados simultáneamente. Creando un lote de actualizaciones y sometiéndolas a la base de datos como una unidad, puede ahorrar tiempo de proceso.

Existen dos tipos de actualizaciones por lotes:

- Actualizaciones por lotes que utilizan objetos Statement.
- Actualizaciones por lotes que utilizan objetos PreparedStatement.

Actualización por lotes de Statement:

Para realizar una actualización por lotes de Statement, debe desactivar el compromiso automático. En Java Database Connectivity (JDBC), el compromiso automático está activado por omisión. El compromiso automático significa que las actualizaciones de la base de datos se comprometen después de procesar cada sentencia SQL. Si desea tratar un grupo de sentencias que se manejan en la base de datos como un grupo funcional, no deseará que la base de datos comprometa cada sentencia individualmente. Si no desactiva el compromiso automático y falla una sentencia situada en medio del lote, no podrá retrotraer la totalidad del lote e intentarlo de nuevo, ya que la mitad de las sentencias se han convertido en finales. Además, el trabajo adicional de comprometer cada sentencia de un lote crea una gran cantidad de carga de trabajo.

Encontrará más detalles en: “Transacciones JDBC” en la página 78.

Después de desactivar el compromiso automático, puede crear un objeto Statement estándar. En lugar de procesar sentencias con métodos como por ejemplo executeUpdate, se añaden al lote con el método addBatch. Una vez añadidas todas las sentencias que desea al lote, puede procesarlas todas con el método executeBatch. Puede vaciar el lote en cualquier momento con el método clearBatch.

El ejemplo siguiente muestra cómo puede utilizar estos métodos:

Ejemplo: actualización por lotes de Statement

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
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();
```

En este ejemplo, se devuelve una matriz de enteros desde el método executeBatch. Esta matriz tiene un valor entero para cada sentencia que se procesa en el lote. Si se insertan valores en la base de datos, el valor de cada sentencia es 1 (suponiendo que el proceso sea satisfactorio). Sin embargo, algunas de las sentencias pueden ser sentencias de actualización (update) que afectan a varias filas. Si coloca en el lote sentencias que no son INSERT, UPDATE o DELETE, se produce una excepción.

Actualización por lotes de PreparedStatement:

Un lote preparedStatement es parecido al lote Statement; sin embargo, un lote preparedStatement funciona siempre sobre la misma sentencia preparada y solo se cambian los parámetros de esa sentencia.

A continuación se ofrece un ejemplo que utiliza un lote preparedStatement.

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
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();
```

Excepción BatchUpdateException de JDBC:

Una consideración importante acerca de las actualizaciones por lotes es la acción que debe realizarse cuando falla una llamada al método executeBatch. En este caso, se lanza un nuevo tipo de excepción, denominada BatchUpdateException. BatchUpdateException es una subclase de SQLException que permite llamar a los mismos métodos a los que ha llamado siempre para recibir el mensaje, SQLState y el código del proveedor.

BatchUpdateException también proporciona el método getUpdateCounts, que devuelve una matriz de enteros. La matriz de enteros contiene cuentas de actualización de todas las sentencias del lote que se han procesado hasta el punto en el que se ha producido la anomalía. La longitud de la matriz indica qué sentencia del lote ha fallado. Por ejemplo, si la matriz devuelta en la excepción tiene una longitud de tres, la cuarta sentencia del lote ha fallado. Por tanto, a partir del único objeto BatchUpdateException devuelto puede determinar las cuentas de actualización para todas las sentencias que han sido satisfactorias, qué sentencia ha fallado y toda la información acerca de la anomalía.

El rendimiento estándar del proceso de actualizaciones por lotes es equivalente al rendimiento del proceso de cada sentencia por separado. Para obtener más información sobre el soporte optimizado de las actualizaciones por lotes, consulte: Soporte de inserción en bloque. Debe seguir utilizando el nuevo modelo al codificar y sacar provecho de futuras optimizaciones del rendimiento.

Nota: En la especificación JDBC 2.1, se suministra una opción diferente para el manejo de condiciones de excepción para actualizaciones por lotes. JDBC 2.1 presenta un modelo en el que el proceso por lotes continúa después de que falle una entrada del lote. Se sitúa una cuenta de actualización especial en la matriz de enteros de cuenta de actualización que se devuelve por cada entrada que falla. Esto permite que lotes de gran tamaño se continúen procesando aunque falle una de sus entradas. Consulte la especificación JDBC 2.1 o JDBC 3.0 para obtener detalles acerca de estas dos modalidades de operación. Por omisión, el controlador JDBC nativo utiliza la definición de JDBC 2.0. El controlador proporciona una propiedad Connection que se utiliza al utilizar DriverManager para establecer las conexiones. También proporciona una propiedad DataSource que se utiliza al utilizar DataSources para establecer las conexiones. Estas propiedades permiten a las aplicaciones elegir cómo desean que las operaciones por lotes manejen las anomalías.

Inserciones en bloque con JDBC:

La inserción en bloque es una operación que inserta varias filas a la vez en una tabla de base de datos.

La inserción en bloque es un tipo de operación especial del System i que proporciona una manera muy optimizada de insertar varias filas a la vez en una tabla de base de datos. Las inserciones en bloque pueden considerarse como un subconjunto de actualizaciones por lotes. Las actualizaciones por lotes pueden ser cualquier forma de petición de actualización, pero las inserciones en bloque son específicas. Sin embargo, los tipos de inserción en bloque de actualizaciones por lotes son comunes; el controlador JDBC nativo se ha modificado para sacar partido de esta característica.

Debido a las restricciones del sistema al utilizar el soporte de inserción en bloque, el valor predeterminado para el controlador JDBC nativo es tener la inserción en bloque inhabilitada. Puede habilitarse mediante una propiedad Connection de una propiedad DataSource. La mayoría de las restricciones de utilización de una inserción en bloque pueden comprobarse y manejarse en nombre del usuario, pero no así algunas restricciones; esta es la razón por la que el soporte de inserción en bloque esté desactivado por omisión. La lista de restricciones es la siguiente:

- La sentencia SQL utilizada debe ser una sentencia INSERT con una cláusula VALUES, indicando que no es una sentencia INSERT con SUBSELECT. El controlador JDBC reconoce esta restricción y realiza las acciones adecuadas.
- Debe utilizarse una PreparedStatement, indicando que no existe soporte optimizado para objetos Statement. El controlador JDBC reconoce esta restricción y realiza las acciones adecuadas.
- La sentencia SQL debe especificar marcadores de parámetro para todas las columnas de la tabla. Esto significa que no puede utilizar valores de constante para una columna ni permitir que la base de datos inserte valores predeterminados para ninguna de las columnas. El controlador JDBC no tiene ningún mecanismo para manejar las pruebas de marcadores de parámetro específicos en la sentencia SQL. Si establece la propiedad para realizar inserciones en bloque optimizadas y no evita los valores predeterminados o constantes en las sentencias SQL, los valores que terminen en la tabla de base de datos no serán correctos.
- La conexión debe establecerse con el sistema local. Esto quiere decir que no se puede usar una conexión mediante DRDA para acceder a un sistema remoto, porque DRDA no permite las operaciones de inserción en bloque. El controlador JDBC no tiene ningún mecanismo para manejar las pruebas de una conexión con un sistema local. Si establece la propiedad para realizar una inserción en bloque optimizada e intenta conectarse con un sistema remoto, el proceso de la actualización por lotes fallará.

Este ejemplo de código muestra cómo habilitar el soporte para el proceso de inserción en bloque. La única diferencia entre este código y una versión que no utilizara el soporte de inserción en bloque es `use block insert=true` que se añade al URL de conexión.

Ejemplo: Proceso de inserción en bloque

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
// Crear una conexión de base de datos
Connection c = DriverManager.getConnection("jdbc:db2:*local;use block insert=true");
BigDecimal bd = new BigDecimal("123456");

// Crear una PreparedStatement para insertarla en una tabla con 4 columnas
PreparedStatement ps =
  c.prepareStatement("insert into cujosql.xxx values(?, ?, ?, ?)");

// Iniciar temporización...
for (int i = 1; i <= 10000; i++) {
  ps.setInt(1, i);                 // Establecer todos los parámetros para una fila
  ps.setBigDecimal(2, bd);
  ps.setBigDecimal(3, bd);
  ps.setBigDecimal(4, bd);
  ps.addBatch();                   // Añadir los parámetros al proceso por lotes
}

// Procesar los lotes
int[] counts = ps.executeBatch();

// Finalizar temporización...
```

En pruebas similares, una inserción en bloque es varias veces más rápida que realizar las mismas operaciones sin utilizar una inserción en bloque. Por ejemplo, la prueba realizada en el código anterior es nueve veces más rápida con la utilización de inserciones en bloque. En los casos en que solo se utilizan tipos nativos en lugar de objetos, la velocidad puede ser hasta dieciséis veces mayor. En las aplicaciones en las que existe una cantidad significativa de trabajo en ejecución, las expectativas deben modificarse adecuadamente.

Tipos de datos avanzados

Los tipos de datos SQL3 proporcionan un enorme nivel de flexibilidad. Son idóneos para almacenar objetos Java serializados, documentos de lenguaje de códigos extensible (XML) y datos multimedia como canciones, imágenes de productos, fotografías de empleados y clips de películas. Java Database Connectivity (JDBC) 2.0 y superior proporciona soporte para trabajar con estos tipos de datos que forman parte del estándar SQL99.

Tipos diferenciados

El tipo diferenciado es un tipo definido por usuario que se basa en un tipo de base de datos estándar. Por ejemplo, puede definir un tipo Número de Seguridad Social, SSN, que internamente sea CHAR(9). La siguiente sentencia SQL crea un tipo DISTINCT de esa clase.

```
CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)
```

El tipo diferenciado siempre se correlaciona con un tipo de datos incorporado. Para obtener más información sobre cuándo y cómo utilizar los tipos diferenciados en el contexto de SQL, vea los manuales de consulta de SQL.

Para utilizar tipos diferenciados en JDBC, acceda a ellos de la misma forma que a un tipo subyacente. El método getUDTs es un nuevo método que le permite consultar qué tipos diferenciados están disponibles en el sistema. En el ejemplo: Programa de tipos diferenciados se muestra lo siguiente:

- La creación de un tipo diferenciado.
- La creación de una tabla que lo utiliza.
- La utilización de una PreparedStatement para establecer un parámetro de tipo diferenciado.
- La utilización de un ResultSet para devolver un tipo diferenciado.
- La utilización de la llamada de la interfaz de programación de aplicaciones (API) de metadatos a getUDTs para obtener información acerca de un tipo diferenciado.

Hallará más información en el subtema Ejemplo: tipos diferenciados, donde encontrará diversas tareas comunes que se pueden realizar utilizando tipos diferenciados.

Objetos grandes

Existen tres tipos de Objetos grandes (LOB):

- Objetos grandes binarios (BLOB)
- Objetos grandes de tipo carácter (CLOB)
- Objetos grandes de caracteres de doble byte (DBCLOB)

Los DBCLOB son parecidos a los CLOB, excepto en su representación de almacenamiento interno de los datos de tipo carácter. Dado que Java y JDBC externalizan todos los datos de tipo carácter como Unicode, en JDBC solo se pueden usar los CLOB. Los DBCLOB funcionan de forma intercambiable con el soporte de CLOB desde una perspectiva JDBC.

Objetos grandes binarios

En muchos aspectos, una columna de Objeto grande binario (BLOB) es parecida a una columna CHAR FOR BIT DATA que puede convertirse en grande. En estas columnas puede almacenar cualquier objeto que pueda representarse como una corriente de bytes no convertidos. A menudo, se emplean columnas BLOB para almacenar objetos Java serializados, imágenes, canciones y otros datos binarios.

Puede utilizar los BLOB de la misma forma que otros tipos de base de datos estándar. Puede pasarlos a procedimientos almacenados, utilizarlos en sentencias preparadas y actualizarlos en conjuntos de resultados. La clase PreparedStatement contiene un método setBlob para pasar BLOB a la base de datos, y la clase ResultSet añade una clase getBlob para recuperarlos de la base de datos. Un BLOB se representa en un programa Java program mediante un objeto BLOB que es una interfaz JDBC.

Objetos grandes de tipo carácter

Los Objetos grandes de tipo carácter (CLOB) son el complemento de datos de tipo carácter de los BLOB. En lugar de almacenar datos en la base de datos sin conversión, los datos se almacenan en la base de datos en forma de texto y se procesan de la misma forma que una columna CHAR. Al igual que para los BLOB, JDBC 2.0 proporciona funciones para tratar directamente con los CLOB. La interfaz PreparedStatement contiene un método setClob y la interfaz ResultSet contiene un método getClob.

Aunque las columnas BLOB y CLOB funcionan como las columnas CHAR FOR BIT DATA y CHAR, así es como funcionan conceptualmente desde la perspectiva del usuario externo. Internamente, son distintos; debido al tamaño potencialmente enorme de las columnas de Objeto grande (LOB), generalmente se trabaja indirectamente con los datos. Por ejemplo, cuando se extrae un bloque de filas de la base de datos, no se mueve un bloque de LOB al ResultSet. En lugar de ello, se mueven punteros denominados localizadores de LOB (es decir, enteros de cuatro bytes) a ResultSet. Sin embargo, no es necesario, tener conocimientos acerca de los localizadores al trabajar con los LOB en JDBC.

Enlaces de datos (Datalinks)

Los **enlaces de datos** son valores encapsulados que contienen una referencia lógica de la base de datos a un archivo almacenado fuera de la misma. Los enlaces de datos se representan y utilizan desde una perspectiva JDBC de dos maneras diferentes, dependiendo de si se utiliza JDBC 2.0 o anterior, o si se utiliza JDBC 3.0 o posterior.

Tipos de datos SQL3 no soportados

Existen otros tipos de datos SQL3 que se han definido y para los que la API de JDBC proporciona soporte. Son ARRAY, REF y STRUCT. Actualmente, estos tipos no se pueden usar en System i. Por tanto,

el controlador JDBC no proporciona ninguna forma de soporte para ellos.

Referencia relacionada

"Ejemplo: tipos distinct" en la página 153
Este es un ejemplo de utilización de tipos distinct.

Escribir código que utilice objetos BLOB:

Existen diversas tareas que pueden realizarse con columnas BLOB (Gran objeto binario) de base de datos mediante la API (Interfaz de programación de aplicaciones) de Java Database Connectivity (JDBC). Los temas que siguen describen brevemente estas tareas e incluyen ejemplos de cómo realizarlas.

Leer los BLOB de la base de datos e insertar BLOB en la base de datos

Con la API de JDBC, existen formas de extraer los BLOB de la base de datos y formas de colocar BLOB en la base de datos. Sin embargo, no existe ninguna forma estandarizada para crear un objeto Blob. Esto no representa ningún problema si la base de datos ya está llena de BLOB, pero sí lo es si desea trabajar con los BLOB desde cero mediante JDBC. En lugar de definir un constructor para las interfaces Blob y Clob de la API de JDBC, se proporciona soporte para colocar los BLOB en la base de datos y extraerlos de la base de datos directamente como otros tipos. Por ejemplo, el método setBinaryStream puede funcionar con una columna de base de datos de tipo Blob. En el tema Ejemplo: objetos Blob se muestran algunas de las formas comunes en que se puede poner un BLOB en la base de datos o en que se puede recuperar de la base de datos.

Trabajar con la API de objetos Blob

Los BLOB se definen en JDBC como una interfaz de la que los diversos controladores proporcionan implementaciones. Esta interfaz contiene una serie de métodos que se pueden utilizar para interaccionar con el objeto Blob. En el tema Ejemplo: utilizar objetos Blob se muestran algunas de las tareas comunes que se pueden realizar mediante esta API. Consulte el Javadoc de JDBC para obtener una lista completa de los métodos disponibles en el objeto Blob.

Utilizar el soporte de JDBC 3.0 para actualizar BLOB

En JDBC 3.0 existe soporte para efectuar cambios en objetos LOB. Estos cambios se pueden almacenar en columnas BLOB de la base de datos. En el tema Ejemplo: actualizar objetos Blob se muestran algunas de las tareas que se pueden realizar con el soporte de BLOB en JDBC 3.0.

Referencia relacionada

"Ejemplo: BLOB"
Este es un ejemplo de cómo puede colocarse un BLOB en la base de datos o recuperarse de la misma.

"Ejemplo: actualizar objetos BLOB" en la página 146
Este es un ejemplo de cómo actualizar objetos BLOB en las aplicaciones Java.

Ejemplo: BLOB:

Este es un ejemplo de cómo puede colocarse un BLOB en la base de datos o recuperarse de la misma.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////
// PutGetBlobs es una aplicación de ejemplo
// que muestra cómo trabajar con la API de JDBC
// para colocar objetos BLOB en columnas de base
// de datos y obtenerlos desde ellas.
//
// Los resultados de la ejecución de este programa
// provocan la inserción de dos valores de BLOB
```

```
// en una tabla nueva. Ambos son idénticos
// y contienen 500k de datos de byte
// aleatorios.
//////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      // Establecer una conexión y una sentencia con las que trabajar.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Borrar ejecuciones anteriores de esta aplicación.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
      } catch (SQLException e) {
          // Ignorarlo y suponer que la tabla no existía.
      }

      // Crear una tabla con una columna BLOB. El tamaño de columna BLOB
      // predeterminado es 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

      // Crear un objeto PreparedStatement que permita colocar
      // un nuevo objeto Blob en la base de datos.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

      // Crear un valor BLOB grande...
      Random random = new Random ();
      byte [] inByteArray = new byte[500000];
      random.nextBytes (inByteArray);

      // Establecer el parámetro PreparedStatement. Nota: esto no es
      // portable a todos los controladores JDBC. Estos no tienen
      // soporte al utilizar setBytes para columnas BLOB. Esto se usa para
      // permitirle generar nuevos BLOB. También permite a
      // los controladores JDBC 1.0 trabajar con columnas que contengan datos BLOB.
      ps.setBytes(1, inByteArray);

      // Procesar la sentencia, insertando el BLOB en la base de datos.
      ps.executeUpdate();

      // Procesar una consulta y obtener el BLOB que se acaba de insertar
      // a partir de la base de datos como objeto Blob.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
      rs.next();
      Blob blob = rs.getBlob(1);


      // Colocar de nuevo ese Blob en la base de datos mediante
      // la PreparedStatement.
      ps.setBlob(1, blob);
      ps.execute();

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: actualizar objetos BLOB:

Este es un ejemplo de cómo actualizar objetos BLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// UpdateBlobs es una aplicación de ejemplo
// que muestra algunas de las API que proporcionan
// soporte para cambiar objetos Blob
// y reflejar esos cambios en la
// base de datos.
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetBlobs.
////////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      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);


      // Truncar un BLOB.
      blob1.truncate((long) 150000);
      System.out.println("La nueva longitud de Blob1 es " + blob1.length());

      // Actualizar parte del BLOB con una matriz de bytes nueva.
      // El código siguiente obtiene los bytes que están en
      // las posiciones 4000-4500 y los establece en las posiciones 500-1000.

      // Obtener parte del BLOB como matriz de bytes.
      byte[] bytes = blob1.getBytes(4000L, 4500);

      int bytesWritten = blob2.setBytes(500L, bytes);

      System.out.println("Los bytes escritos son " + bytesWritten);

      // Los bytes se encuentran ahora en la posición 500 de blob2
      long startInBlob2 = blob2.position(bytes, 1);

      System.out.println("encontrado patrón que empieza en la posición " + startInBlob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: utilizar objetos BLOB:

Este es un ejemplo de cómo utilizar objetos BLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// UseBlobs es una aplicación de ejemplo
// que muestra algunas de las API asociadas
// con objetos Blob.
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetBlobs.
////////////////////////////////////////
import java.sql.*;

public class UseBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      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);


      // Determinar la longitud de un LOB.
      long end = blob1.length();
      System.out.println("La longitud de Blob1 es " + blob1.length());

      // Al trabajar con objetos LOB, toda la indexación relacionada con ellos
      // se basa en 1, y no en 0 como en las series y matrices.
      long startingPoint = 450;
      long endingPoint = 500;

      // Obtener parte del BLOB como matriz de bytes.
      byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

      // Buscar donde se encuentra primero un sub-BLOB o matriz de bytes en un
      // BLOB. La configuración de este programa ha colocado dos copias idénticas
      // de un BLOB aleatorio en la base de datos. Por tanto, la posición inicial
      // de la matriz de bytes extraída de blob1 se puede encontrar en la
      // posición inicial de blob2. La excepción sería si hubiera 50
      // bytes aleatorios idénticos en los LOB anteriormente.
      long startInBlob2 = blob2.position(outByteArray, 1);

      System.out.println("encontrado patrón que empieza en la posición " + startInBlob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Escribir código que utilice objetos CLOB:

Existen diversas tareas que pueden realizarse con columnas CLOB y DBCLOB de base de datos mediante la API (Interfaz de programación de aplicaciones) de Java Database Connectivity (JDBC). Los temas que siguen describen brevemente estas tareas e incluyen ejemplos de cómo realizarlas.

Leer los CLOB de la base de datos e insertar CLOB en la base de datos

Con la API de JDBC, existen formas de extraer los CLOB de la base de datos y formas de colocar CLOB en la base de datos. Sin embargo, no existe ninguna forma estandarizada para crear un objeto Clob. Esto no representa ningún problema si la base de datos ya está llena de CLOB, pero sí lo es si desea trabajar con los CLOB desde cero mediante JDBC. En lugar de definir un constructor para las interfaces Blob y Clob de la API de JDBC, se proporciona soporte para colocar los CLOB en la base de datos y extraerlos de la base de datos directamente como otros tipos. Por ejemplo, el método setCharacterStream puede funcionar con una columna de base de datos de tipo Clob. En el tema Ejemplo: objetos CLOB se muestran algunas de las formas comunes en que se puede poner un CLOB en la base de datos o en que se puede recuperar de la base de datos.

Trabajar con la API de objetos Clob

Los CLOB se definen en JDBC como una interfaz de la que los diversos controladores proporcionan implementaciones. Esta interfaz contiene una serie de métodos que pueden utilizarse para interactuar con el objeto Clob. En el tema Ejemplo: utilizar objetos Clob se muestran algunas de las tareas comunes que se pueden realizar mediante esta API. Consulte el Javadoc de JDBC para obtener una lista completa de los métodos disponibles en el objeto Clob.

Utilizar el soporte de JDBC 3.0 para actualizar CLOB

En JDBC 3.0 existe soporte para efectuar cambios en objetos LOB. Estos cambios pueden almacenarse en columnas CLOB de la base de datos. En el tema Ejemplo: actualizar objetos Clob se muestran algunas de las tareas que se pueden realizar con el soporte de CLOB en JDBC 3.0.

Referencia relacionada

"Ejemplo: CLOB"
Este es un ejemplo de cómo puede colocarse un CLOB en la base de datos o recuperarse de la misma.

"Ejemplo: utilizar objetos CLOB" en la página 150
Este es un ejemplo de cómo utilizar objetos CLOB en las aplicaciones Java.

"Ejemplo: actualizar objetos CLOB" en la página 149
Este es un ejemplo de cómo actualizar objetos CLOB en las aplicaciones Java.

Ejemplo: CLOB:

Este es un ejemplo de cómo puede colocarse un CLOB en la base de datos o recuperarse de la misma.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////
// PutGetClobs es una aplicación de ejemplo
// que muestra cómo trabajar con la API de JDBC
// para colocar objetos CLOB en columnas de base
// de datos y obtenerlos desde ellas.
//
// Los resultados de la ejecución de este programa
// son que existan dos valores de CLOB
// en una tabla nueva. Ambos son idénticos
// y contienen alrededor de 500k de datos
// de texto de repetición.
//////////////////////////////////////////
import java.sql.*;
```

```
public class PutGetClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      // Establecer una conexión y una sentencia con las que trabajar.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Borrar ejecuciones anteriores de esta aplicación.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
      } catch (SQLException e) {
          // Ignorarlo y suponer que la tabla no existía.
      }

      // Crear una tabla con una columna CLOB. El tamaño de columna CLOB
      // predeterminado es 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

      // Crear un objeto PreparedStatement que permita colocar
      // un nuevo objeto Clob en la base de datos.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

      // Crear un valor 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();

      // Establecer el parámetro PreparedStatement. Esto no es
      // portable a todos los controladores JDBC. Estos no tienen
      // que soportar setBytes para columnas CLOB. Esto se hace para
      // permitirle que genere nuevos CLOB. También
      // permite a los controladores JDBC 1.0 trabajar con columnas que contengan
      // datos Clob.
      ps.setString(1, clobValue);

      // Procesar la sentencia, insertando el clob en la base de datos.
      ps.executeUpdate();

      // Procesar una consulta y obtener el CLOB que se acaba de insertar
      // a partir de la base de datos como objeto Clob.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
      rs.next();
      Clob clob = rs.getClob (1);


      // Colocar de nuevo ese Clob en la base de datos mediante
      // la PreparedStatement.
      ps.setClob(1, clob);
      ps.execute();

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: actualizar objetos CLOB:

Este es un ejemplo de cómo actualizar objetos CLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// UpdateClobs es una aplicación de ejemplo
// que muestra algunas de las API que proporcionan
// soporte para cambiar objetos Clob
// y reflejar esos cambios en la
// base de datos.
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetClobs.
////////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      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);


      // Truncar un CLOB.
      clob1.truncate((long) 150000);
      System.out.println("La nueva longitud de Clob1 es " + clob1.length());

      // Actualizar una parte del CLOB con un nuevo valor de tipo String.
      String value = "Some new data for once";
      int charsWritten = clob2.setString(500L, value);

      System.out.println("Los caracteres escritos son " + charsWritten);

      // Los bytes se encuentran en la posición 500 de clob2
      long startInClob2 = clob2.position(value, 1);

      System.out.println("encontrado patrón que empieza en la posición " + startInClob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: utilizar objetos CLOB:

Este es un ejemplo de cómo utilizar objetos CLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////
// UpdateClobs es una aplicación de ejemplo
// que muestra algunas de las API que proporcionan
// soporte para cambiar objetos Clob
// y reflejar esos cambios en la
// base de datos.
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetClobs.
//////////////////////////////////////
import java.sql.*;

public class UseClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      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);


      // Determinar la longitud de un LOB.
      long end = clob1.length();
      System.out.println("La longitud de Clob1 es " + clob1.length());

      // Al trabajar con objetos LOB, toda la indexación relacionada con ellos
      // se basa en 1, y no en 0 como en las series y matrices.
      long startingPoint = 450;
      long endingPoint = 50;

      // Obtener parte del CLOB como matriz de bytes.
      String outString = clob1.getSubString(startingPoint, (int)endingPoint);
      System.out.println("La subserie de Clob es " + outString);

      // Buscar donde se encuentra primero un sub-CLOB o serie en un
      // CLOB. La configuración de este programa ha colocado dos copias idénticas
      // de un CLOB repetido en la base de datos. Por tanto, la posición inicial
      // de la serie extraída de clob1 se puede encontrar en la
      // posición inicial de clob2 si la búsqueda empieza cerca de la posición
      // en la que empieza la serie.
      long startInClob2 = clob2.position(outString, 440);

      System.out.println("encontrado patrón que empieza en la posición " + startInClob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Escribir código que utilice enlaces de datos:

La forma de trabajar con enlaces de datos depende del release con el que se trabaja. En JDBC 3.0, existe soporte para trabajar directamente con columnas Datalink mediante los métodos getURL y putURL.

En las versiones anteriores de JDBC, era necesario trabajar con columnas Datalink como si fueran columnas String. Actualmente, la base de datos no soporta las conversiones automáticas entre Datalink y tipos de datos de carácter. En consecuencia, es necesario realizar una conversión temporal de tipos en las sentencias SQL.

Ejemplo: Datalink:

En esta aplicación de ejemplo se enseña a utilizar la API de JDBC para manejar columnas de base de datos de tipo datalink.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////
// PutGetDatalinks es una aplicación de ejemplo
// que muestra cómo utilizar la API de JDBC
// para manejar columnas de base de datos de tipo datalink.
//////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrar el controlador JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Error de configuración.
        }

        // Establecer una conexión y una sentencia con las que trabajar.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Borrar ejecuciones anteriores de esta aplicación.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
        } catch (SQLException e) {
            // Ignorarlo y suponer que la tabla no existía.
        }

        // Crear una tabla con una columna datalink.
        s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

        // Crear un objeto PreparedStatement que permita añadir
        // un nuevo objeto datalink en la base de datos. Dado que la conversión
        // a un datalink no puede realizarse directamente en la base de datos,
        // puede codificar la sentencia SQL para realizar la conversión explícita.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
                                                   VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

        // Establecer el datalink. Este URL señala hacia un tema sobre
        // las nuevas características de JDBC 3.0.
        ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

        // Procesar la sentencia, insertando el CLOB en la base de datos.
        ps.executeUpdate();

        // Procesar una consulta y obtener el CLOB que se acaba de insertar
        // a partir de la base de datos como objeto Clob.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        String datalink = rs.getString(1);
```

```
        // Colocar ese valor de datalink en la base de datos mediante
        // la PreparedStatement. Nota: para esta función se necesita
        // soporte de JDBC 3.0.
        /*
        try {
            URL url = new URL(datalink);
            ps.setURL(1, url);
            ps.execute();
        } catch (MalformedURLException mue) {
            // Manejar aquí este problema.
        }

        rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        URL url = rs.getURL(1);
        System.out.println("el valor de URL es " + url);
        */

        c.close(); // El cierre de la conexión también cierra stmt y rs.
    }
}
```

Ejemplo: tipos distinct:

Este es un ejemplo de utilización de tipos distinct.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////
// Este programa de ejemplo muestra ejemplos de
// diversas tareas comunes que pueden realizarse
// con tipos distinct.
//////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrar el controlador JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Error de configuración.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Borrar ejecuciones antiguas.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignorarlo y suponer que la tabla no existía.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
        } catch (SQLException e) {
            // Ignorarlo y suponer que la tabla no existía.
        }

        // Crear el tipo, crear la tabla e insertar un valor.
```

```
        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();

        // Puede obtener detalles sobre los tipos disponibles con nuevos metadatos en
        // 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("Nombre de tipo " + rs.getString(3) +
                           " tiene el tipo " + rs.getString(4));


        // Acceder a los datos que ha insertado.
        rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
        rs.next();
        System.out.println("SSN es " + rs.getString(1));

        c.close(); // El cierre de la conexión también cierra stmt y rs.
    }
}
```

RowSets de JDBC

Los RowSets se añadieron originariamente al paquete opcional Java Database Connectivity (JDBC) 2.0. A diferencia de algunas de las interfaces más conocidas de la especificación JDBC, la especificación RowSet está diseñada más como infraestructura que como implementación real. Las interfaces RowSet definen un conjunto de funciones centrales que comparten todos los RowSets. Los proveedores de la implementación RowSet tienen una libertad considerable para definir las funciones necesarias para satisfacer sus necesidades en un espacio de problemas específico.

Características de RowSet:

Puede solicitar determinadas propiedades que los rowsets deben satisfacer. Las propiedades comunes incluyen el conjunto de interfaces a las que el rowset resultante debe dar soporte.

Los RowSets son ResultSets

La interfaz RowSet amplía la interfaz ResultSet, lo que significa que los RowSets tienen la capacidad de realizar todas las funciones que los ResultSets pueden efectuar. Por ejemplo, los RowSets pueden ser desplazables y actualizables.

Los RowSets pueden desconectarse de la base de datos

Existen dos categorías de RowSets:

Conectado
: Aunque los RowSets conectados se pueblan de datos, siempre tienen conexiones internas con la base de datos subyacente abierta y funcionan como envolturas en torno a una implementación de ResultSet.

Desconectado
: No es necesario que los RowSets desconectados mantengan siempre conexiones con su origen de

datos. Los RowSets desconectados pueden desconectarse de la base de datos, utilizarse de diversas formas y, a continuación, reconectarse a la base de datos para reflejar los cambios efectuados en ellos.

Los RowSets son componentes JavaBeans

Los RowSets tienen soporte para el manejo de eventos basado en el modelo de manejo de eventos de los JavaBeans. También tienen propiedades que pueden establecerse. El RowSet puede utilizar estas propiedades para realizar las siguientes operaciones:

- Establecer una conexión con la base de datos.
- Procesar una sentencia SQL.
- Determinar características de los datos que el RowSet representa y manejar otras características internas del objeto RowSet.

Los RowSets son serializables

Los RowSets pueden serializarse y deserializarse para permitirles fluir a través de una conexión de red, escribirlos en un archivo plano (es decir, en un documento de texto sin proceso de texto ni otros caracteres de estructura), etc.

DB2CachedRowSet:

El objeto DB2CachedRowSet es un RowSet desconectado, lo que significa que puede utilizarse sin estar conectado a la base de datos. Su implementación es muy parecida a la descripción de un CachedRowSet. DB2CachedRowSet es un contenedor para filas de datos de un ResultSet. DB2CachedRowSet contiene la totalidad de sus propios datos, de forma que no necesita mantener una conexión con la base de datos aparte de la explícitamente necesaria para leer o escribir datos en la misma.

Utilizar DB2CachedRowSet:

Debido a que el objeto DB2CachedRowSet puede desconectarse y serializarse, resulta de utilidad en entornos donde no siempre es práctico ejecutar un controlador JDBC completo (por ejemplo, en asistentes digitales personales (PDA) y teléfonos móviles habilitados para Java).

Dado que el objeto DB2CachedRowSet se encuentra en memoria y sus datos siempre se conocen, puede funcionar como una forma altamente optimizada de ResultSet desplazable para las aplicaciones. Mientras que los DB2ResultSets desplazables pagan habitualmente un precio de rendimiento debido a que sus movimientos aleatorios interfieren con la capacidad del controlador JDBC para almacenar en caché filas de datos, los RowSets no presentan este problema.

En DB2CachedRowSet se suministran dos métodos que crean nuevos RowSets:

- El método createCopy crea un nuevo RowSet que es idéntico al copiado.
- El método createShared crea un nuevo RowSet que comparte los mismos datos subyacentes que el original.

Puede utilizar el método createCopy para entregar ResultSets comunes a los clientes. Si los datos de tabla no cambian, crear una copia de un RowSet y pasarla a cada cliente es más eficaz que ejecutar cada vez una consulta en la base de datos.

Puede utilizar el método createShared para mejorar el rendimiento de la base de datos permitiendo que varios usuarios utilicen los mismos datos. Por ejemplo, suponga que dispone de un sitio Web que muestra los veinte productos más vendidos en la página de presentación cuando un cliente se conecta. Desea que la información de la página principal se actualice periódicamente, pero ejecutar la consulta para obtener los productos adquiridos con mayor frecuencia cada vez que un cliente visita la página principal no resulta práctico. Mediante el método createShared, puede de hecho crear ″cursores″ para

cada cliente sin necesidad de procesar de nuevo la consulta ni almacenar una enorme cantidad de información en la memoria. Cuando proceda, puede ejecutarse de nuevo la consulta para buscar los productos adquiridos con mayor frecuencia. Los datos nuevos pueden llenar el RowSet utilizado para crear los cursores compartidos y los servlets pueden utilizarlos.

Los DB2CachedRowSets proporcionan una característica de proceso retardado. Esta característica permite agrupar varias peticiones de consulta y procesarlas en la base de datos como una sola petición. Vea el tema "Crear y poblar un DB2CachedRowSet" para eliminar parte de la sobrecarga de cálculo a la que, de lo contrario, estaría sometida la base de datos.

Debido a que RowSet debe efectuar un cuidadoso seguimiento de los cambios que se producen en sí mismo para que se reflejen de nuevo en la base de datos, existe soporte para funciones que deshacen los cambios o permiten al usuario ver todos los cambios que se han efectuado. Por ejemplo, existe un método showDeleted que puede utilizarse para indicar al RowSet que permita al usuario extraer filas suprimidas. También existen los métodos cancelRowInsert y cancelRowDelete para deshacer inserciones y supresiones de filas, respectivamente, después de efectuarlas.

El objeto DB2CachedRowSet ofrece una mejor interoperatividad con otras API Java debido a su soporte de manejo de eventos y a sus métodos toCollection, que permiten convertir un RowSet o parte de él en una colección Java.

El soporte de manejo de eventos de DB2CachedRowSet puede utilizarse en aplicaciones de interfaz gráfica de usuario (GUI) para controlar pantallas, anotar información acerca de los cambios de RowSet a medida que se realizan o buscar información relativa a los cambios en orígenes que no sean RowSets. Encontrará los detalles en: "Eventos de DB2JdbcRowSet" en la página 175.

Para obtener información sobre el modelo de eventos y el manejo de eventos, vea: "DB2JdbcRowSet" en la página 172, ya que este soporte funciona de manera idéntica para ambos tipos de RowSets.

Crear y poblar un DB2CachedRowSet:

Hay varias maneras de colocar datos en un DB2CachedRowSet: el método populate, propiedades de DB2CachedRowSet con DataSources, propiedades de DB2CachedRowSet y los URL JDBC, el método setConnection(Connection), el método execute(Connection) y el método execute(int).

Utilizar el método populate

Los DB2CachedRowSets tienen un método populate que puede utilizarse para colocar datos en el RowSet desde un objeto DB2ResultSet. A continuación se ofrece un ejemplo de este método.

Ejemplo: utilizar el método populate

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Establecer una conexión con la base de datos.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Crear una sentencia y utilizarla para realizar una consulta.
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");

// Crear y llenar un DB2CachedRowSet desde ella.
DB2CachedRowSet crs = new DB2CachedRowSet();
crs.populate(rs);

// Nota: desconectar los objetos ResultSet, Statement
// y Connection utilizados para crear el RowSet.
rs.close();
```

```
stmt.close();
conn.close();

// Repetir en bucle los datos del RowSet.
while (crs.next()) {
  System.out.println("v1 es " + crs.getString(1));
}

crs.close();
```

Utilizar propiedades de DB2CachedRowSet y DataSources

Los DB2CachedRowSets tienen propiedades que permiten a los DB2CachedRowSets aceptar una consulta SQL y un nombre DataSource. Luego utilizan la consulta SQL y el nombre DataSource para crear datos para sí mismos. A continuación se ofrece un ejemplo de este método. Se supone que la referencia al DataSource denominado BaseDataSource es un DataSource válido que se ha configurado anteriormente.

Ejemplo: utilizar propiedades DB2CachedRowSet y DataSources

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Crear un nuevo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Establecer las propiedades necesarias para
// que el RowSet utilice un DataSource para poblarse.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet. Esto hace
// que el RowSet utilice el DataSource y la consulta SQL
// especificada para poblarse de datos. Una vez
// poblado, el RowSet se desconecta de la base de datos.
crs.execute();

// Repetir en bucle los datos del RowSet.
while (crs.next()) {
  System.out.println("v1 es " + crs.getString(1));
}

// Finalmente, cerrar el RowSet.
crs.close();
```

Utilizar propiedades de DB2CachedRowSet y los URL de JDBC

Los DB2CachedRowSets tienen propiedades que permiten a los DB2CachedRowSets aceptar una consulta SQL y un URL de JDBC. Luego utilizan la consulta y el URL de JDBC para crear datos para sí mismos. A continuación se ofrece un ejemplo de este método.

Ejemplo: utilizar propiedades DB2CachedRowSet y los URL de JDBC

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Crear un nuevo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Establecer las propiedades necesarias para
// que el RowSet utilice un URL de JDBC para poblarse.
crs.setUrl("jdbc:db2:*local");
crs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet. Esto hace
```

```
// que el RowSet utilice el DataSource y la consulta SQL
// especificada para poblarse de datos. Una vez
// poblado, el RowSet se desconecta de la base de datos.
crs.execute();

// Repetir en bucle los datos del RowSet.
while (crs.next()) {
  System.out.println("v1 es " + crs.getString(1));
}

// Finalmente, cerrar el RowSet.
crs.close();
```

Utilizar el método setConnection(Connection) para emplear una conexión de base de datos existente

Para promocionar la reutilización de objetos JDBC Connection, DB2CachedRowSet proporciona un mecanismo para pasar un objeto Connection establecido al DB2CachedRowSet utilizado para llenar el RowSet. Si se pasa un objeto Connection suministrado por usuario, el DB2CachedRowSet no lo desconecta después de llenarse.

Ejemplo: utilizar el método setConnection(Connection) para utilizar una conexión de base de datos existente

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Establecer una conexión JDBC con la base de datos.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Crear un nuevo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Establecer las propiedades necesarias para que
//el RowSet utilice una conexión ya establecida
// para llenarse.
crs.setConnection(conn);
crs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet. Esto hace
// que el RowSet utilice la conexión que se le suministró
// con anterioridad. Una vez poblado, el RowSet no
// cierra la conexión suministrada por el usuario.
crs.execute();

// Repetir en bucle los datos del RowSet.
while (crs.next()) {
  System.out.println("v1 es " + crs.getString(1));
}

// Finalmente, cerrar el RowSet.
crs.close();
```

Utilizar el método execute(Connection) para emplear una conexión de base de datos existente

Para promocionar la reutilización de objetos JDBC Connection, DB2CachedRowSet proporciona un mecanismo para pasar un objeto Connection establecido al DB2CachedRowSet cuando se llama al método execute. Si se pasa un objeto Connection suministrado por usuario, el DB2CachedRowSet no lo desconecta después de llenarse.

Ejemplo: utilizar el método execute(Connection) para utilizar una conexión de base de datos existente

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Establecer una conexión JDBC con la base de datos.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Crear un nuevo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Establecer la sentencia SQL que debe utilizarse para
// llenar el RowSet.
crs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet, pasando la conexión
// que hay que utilizar. Una vez poblado, el RowSet no
// cierra la conexión suministrada por el usuario.
crs.execute(conn);

// Repetir en bucle los datos del RowSet.
while (crs.next()) {
  System.out.println("v1 es " + crs.getString(1));
}

// Finalmente, cerrar el RowSet.
crs.close();
```

Utilizar el método execute(int) para agrupar peticiones de base de datos

Para reducir la carga de trabajo de la base de datos, DB2CachedRowSet proporciona un mecanismo para agrupar sentencias SQL de varias hebras en una sola petición de proceso de la base de datos.

Ejemplo: utilizar el método execute(int) para agrupar peticiones de base de datos

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Crear un nuevo DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Establecer las propiedades necesarias para
// que el RowSet utilice un DataSource para poblarse.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet. Esto hace
// que el RowSet utilice el DataSource y la consulta SQL
// especificada para poblarse de datos. Una vez
// poblado, el RowSet se desconecta de la base de datos.
// Esta versión del método execute acepta el número de segundos
// que sean necesarios para esperar los resultados. Al
// permitir un retardo, el RowSet puede agrupar las peticiones
// de varios usuarios y procesar la petición con respecto a
// la base de datos subyacente una sola vez.
crs.execute(5);

// Repetir en bucle los datos del RowSet.
while (crs.next()) {
  System.out.println("v1 es " + crs.getString(1));
}

// Finalmente, cerrar el RowSet.
crs.close();
```

Acceso a datos de DB2CachedRowSet y manipulación del cursor:

Este tema proporciona información de cómo aceder a datos de DB2CachedRowSet y a diversas funciones de manipulación del cursor.

Los RowSets dependen de métodos de ResultSet. En muchas operaciones, como por ejemplo acceso a datos DB2CachedRowSet y movimiento de cursores, no hay ninguna diferencia a nivel de aplicación entre utilizar un ResultSet y utilizar un RowSet.

Acceso a datos de DB2CachedRowSet

RowSets y ResultSets acceden a los datos de la misma manera. En el ejemplo siguiente, el programa crea una tabla y la llena con diversos tipos de datos mediante JDBC. Una vez que la tabla está preparada, se crea un DB2CachedRowSet y se llena con la información de la tabla. El ejemplo también utiliza diversos métodos get de la clase RowSet.

Ejemplo: acceso a datos de DB2CachedRowSet

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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[])
  {
    // Registrar el controlador.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No es necesario continuar.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Borrar ejecuciones anteriores
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Capturado drop table: " + ex.getMessage());
      }

      // Crear tabla de prueba
      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 indicación de la hora)");
      System.out.println("Tabla creada.");

      // Insertar algunas filas de prueba
      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("Filas insertadas");
```

```
      ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
      System.out.println("Consulta ejecutada");

      // Crear un nuevo conjunto de filas (rowset) y llenarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet lleno.");

      conn.close();
      System.out.println("RowSet se desconecta...");

      System.out.println("Probar con getObject");
      int count = 0;
      while (crs.next()) {
        System.out.println("Fila " + (++count));
        for (int i = 1; i <= 13; i++) {
          System.out.println("  Col " + i + " value " + crs.getObject(i));
        }
      }

      System.out.println("Probar con getXXX... ");
      crs.first();
      System.out.println("Fila 1");
      System.out.println("  Valor Col 1 " + crs.getShort(1));
      System.out.println("  Valor Col 2 " + crs.getInt(2));
      System.out.println("  Valor Col 3 " + crs.getLong(3));
      System.out.println("  Valor Col 4 " + crs.getFloat(4));
      System.out.println("  Valor Col 5 " + crs.getDouble(5));
      System.out.println("  Valor Col 6 " + crs.getDouble(6));
      System.out.println("  Valor Col 7 " + crs.getBigDecimal(7));
      System.out.println("  Valor Col 8 " + crs.getBigDecimal(8));
      System.out.println("  Valor Col 9 " + crs.getString(9));
      System.out.println("  Valor Col 10 " + crs.getString(10));
      System.out.println("  Valor Col 11 " + crs.getDate(11));
      System.out.println("  Valor Col 12 " + crs.getTime(12));
      System.out.println("  Valor Col 13 " + crs.getTimestamp(13));
      crs.next();
      System.out.println("Fila 2");
      System.out.println("  Valor Col 1 " + crs.getShort(1));
      System.out.println("  Valor Col 2 " + crs.getInt(2));
      System.out.println("  Valor Col 3 " + crs.getLong(3));
      System.out.println("  Valor Col 4 " + crs.getFloat(4));
      System.out.println("  Valor Col 5 " + crs.getDouble(5));
      System.out.println("  Valor Col 6 " + crs.getDouble(6));
      System.out.println("  Valor Col 7 " + crs.getBigDecimal(7));
      System.out.println("  Valor Col 8 " + crs.getBigDecimal(8));
      System.out.println("  Valor Col 9 " + crs.getString(9));
      System.out.println("  Valor Col 10 " + crs.getString(10));
      System.out.println("  Valor Col 11 " + crs.getDate(11));
      System.out.println("  Valor Col 12 " + crs.getTime(12));
      System.out.println("  Valor Col 13 " + crs.getTimestamp(13));

      crs.close();
    }
    catch (Exception ex) {
      System.out.println("SQLException: " + ex.getMessage());
                 ex.printStackTrace();
    }
  }
}
```

Manipulación del cursor

Los RowSets son desplazables y actúan exactamente igual que un ResultSet desplazable. En el ejemplo siguiente, el programa crea una tabla y la llena con datos mediante JDBC. Una vez que la tabla está

preparada, se crea un objeto DB2CachedRowSet y se llena con la información de la tabla. El ejemplo también utiliza diversas funciones de manipulación de cursores.

Ejemplo: manipulación de cursores

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample1
{
  public static void main(String args[])
  {
    // Registrar el controlador.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No es necesario continuar.
      System.exit(1);
  }

  try {
    Connection conn = DriverManager.getConnection("jdbc:db2:*local");

    Statement stmt = conn.createStatement();

    // Borrar ejecuciones anteriores
    try {
      stmt.execute("drop table cujosql.test_table");
    }
    catch (SQLException ex) {
      System.out.println("Capturado drop table: " + ex.getMessage());
    }

    // Crear una tabla de prueba
    stmt.execute("Create table cujosql.test_table (col1 smallint)");
    System.out.println("Tabla creada.");

    // Insertar algunas filas de prueba
    for (int i = 0; i < 10; i++) {
      stmt.execute("insert into cujosql.test_table values (" + i + ")");
    }
    System.out.println("Filas insertadas");

    ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
    System.out.println("Consulta ejecutada");

    // Crear un nuevo conjunto de filas (rowset) y llenarlo...
    DB2CachedRowSet crs = new DB2CachedRowSet();
    crs.populate(rs);
    System.out.println("RowSet lleno.");

    conn.close();
    System.out.println("RowSet se desconecta...");

    System.out.println("Utilizar next()");
    while (crs.next()) {
      System.out.println("v1 es " + crs.getShort(1));
    }

    System.out.println("Utilizar previous()");
    while (crs.previous()) {
      System.out.println("el valor es " + crs.getShort(1));
    }
```

```
    System.out.println("Utilizar relative()");
    crs.next();
    crs.relative(9);
    System.out.println("el valor es " + crs.getShort(1));

    crs.relative(-9);
    System.out.println("el valor es " + crs.getShort(1));

    System.out.println("Utilizar absolute()");
    crs.absolute(10);
    System.out.println("el valor es " + crs.getShort(1));
    crs.absolute(1);
    System.out.println("el valor es " + crs.getShort(1));
    crs.absolute(-10);
    System.out.println("el valor es " + crs.getShort(1));
    crs.absolute(-1);
    System.out.println("el valor es " + crs.getShort(1));

    System.out.println("Probar beforeFirst()");
    crs.beforeFirst();
    System.out.println("isBeforeFirst es " + crs.isBeforeFirst());
    crs.next();
    System.out.println("mover uno... isFirst es " + crs.isFirst());

    System.out.println("Probar afterLast()");
    crs.afterLast();
    System.out.println("isAfterLast es " + crs.isAfterLast());
    crs.previous();
    System.out.println("mover uno... isLast es " + crs.isLast());

    System.out.println("Probar getRow()");
    crs.absolute(7);
    System.out.println("la fila debe ser (7) y es " + crs.getRow() +
                       " el valor debe ser (6) y es " + crs.getShort(1));

    crs.close();
  }
  catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
  }
 }
}
```

Cambiar datos de DB2CachedRowSet y reflejar los cambios de nuevo en el origen de datos:

Este tema proporciona información acerca de cómo hacer cambios de filas en un DB2CachedRowSet y luego actualizar la base de datos subyacente.

DB2CachedRowSet utiliza los mismos métodos que la interfaz ResultSet estándar para efectuar cambios en los datos del objeto RowSet. No existe ninguna diferencia a nivel de aplicación entre cambiar los datos de un RowSet y cambiar los datos de un ResultSet. DB2CachedRowSet proporciona el método acceptChanges, que se utiliza para reflejar de nuevo los cambios de RowSet en la base de datos de donde proceden los datos.

Suprimir, insertar y actualizar filas en un DB2CachedRowSet

Los DB2CachedRowSets pueden actualizarse. En el ejemplo siguiente, el programa crea una tabla y la llena con datos mediante JDBC. Una vez que la tabla está preparada, se crea un DB2CachedRowSet y se llena con la información de la tabla. En el ejemplo también se utilizan diversos métodos que pueden utilizarse para actualizar el RowSet y muestra cómo utilizar la propiedad showDeleted, que permite a la aplicación extraer filas incluso después de que se hayan suprimido. Además, en el ejemplo se utilizan los métodos cancelRowInsert y cancelRowDelete para permitir deshacer la inserción o la supresión de filas.

Ejemplo: suprimir, insertar y actualizar filas en un DB2CachedRowSet

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample2
{
  public static void main(String args[])
  {
    // Registrar el controlador.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
              ex.getMessage());

      // No es necesario continuar.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Borrar ejecuciones anteriores
      try {
        stmt.execute("drop table cujosql.test_table");
      }

      catch (SQLException ex) {
        System.out.println("Capturado drop table: " + ex.getMessage());
      }

      // Crear tabla de prueba
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Tabla creada.");

      // Insertar algunas filas de prueba
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Filas insertadas");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Consulta ejecutada");

      // Crear un nuevo conjunto de filas (rowset) y llenarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet lleno.");

      conn.close();
      System.out.println("RowSet se desconecta...");

      System.out.println("Suprimir las tres primeras filas");
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();
```

```
  crs.beforeFirst();
  System.out.println("Insertar el valor -10 en el RowSet");
  crs.moveToInsertRow();
  crs.updateShort(1, (short)-10);
  crs.insertRow();
  crs.moveToCurrentRow();

  System.out.println("Actualizar las filas para que sean el contrario de lo que son ahora");
  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 es ahora (value - inserted - updated - deleted)");
  crs.beforeFirst();
  while (crs.next()) {
    System.out.println("el valor es " + crs.getShort(1) + " " +
                crs.rowInserted() + " " +
                crs.rowUpdated() + " " +
                crs.rowDeleted());
  }

  System.out.println("getShowDeleted es " + crs.getShowDeleted());

  System.out.println("Ahora, deshacer las inserciones y supresiones");
  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 es ahora (value - inserted - updated - deleted)");
  crs.beforeFirst();
  while (crs.next()) {
    System.out.println("el valor es " + crs.getShort(1) + " " +
                crs.rowInserted() + " " +
                crs.rowUpdated() + " " +
                crs.rowDeleted());
  }

  System.out.println("finalmente, mostrar que el cancelRowUpdates llamante funciona);
  crs.first();
  crs.updateShort(1, (short) 1000);
  crs.cancelRowUpdates();
  crs.updateRow();
  System.out.println("el valor de la fila es " + crs.getShort(1));
  System.out.println("getShowDeleted es " + crs.getShowDeleted());

  crs.close();

}
```

```
    catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

Reflejar de nuevo los cambios de un DB2CachedRowSet en la base de datos subyacente

Una vez efectuados los cambios en un DB2CachedRowSet, estos solo existen mientras exista el objeto RowSet. Es decir, realizar cambios en RowSet desconectado no tiene ningún efecto sobre la base de datos. Para reflejar los cambios de un RowSet en la base de datos subyacente, se utiliza el método acceptChanges. Este método indica al RowSet desconectado que debe volver a establecer una conexión con la base de datos e intentar efectuar en la base de datos subyacente los cambios que se han realizado en el RowSet. Si los cambios no pueden realizarse de forma segura en la base de datos debido a conflictos con otros cambios de base de datos después de crear el RowSet, se lanza una excepción y la transacción se retrotrae.

Ejemplo: reflejar de nuevo los cambios de un DB2CachedRowSet en la base de datos subyacente

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample3
{
  public static void main(String args[])
  {
    // Registrar el controlador.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No es necesario continuar.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Borrar ejecuciones anteriores
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Capturado drop table: " + ex.getMessage());
      }

      // Crear tabla de prueba
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Tabla creada.");

      // Insertar algunas filas de prueba
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Filas insertadas");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
```

```
      System.out.println("Consulta ejecutada");

      // Crear un nuevo conjunto de filas (rowset) y llenarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet lleno.");

      conn.close();
      System.out.println("RowSet se desconecta...");

      System.out.println("Suprimir las tres primeras filas");
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();

      crs.beforeFirst();
      System.out.println("Insertar el valor -10 en el RowSet");
      crs.moveToInsertRow();
      crs.updateShort(1, (short)-10);
      crs.insertRow();
      crs.moveToCurrentRow();

      System.out.println("Actualizar las filas para que sean el contrario de lo que son ahora");
      crs.beforeFirst();
      while (crs.next()) {
        short value = crs.getShort(1);
        value = (short)-value;
        crs.updateShort(1, value);
        crs.updateRow();
      }

      System.out.println("Ahora, aceptar los cambios de la base de datos");

      crs.setUrl("jdbc:db2:*local");
      crs.setTableName("cujosql.test_table");

      crs.acceptChanges();
      crs.close();

      System.out.println("La tabla de base de datos tendrá este aspecto:");
      conn = DriverManager.getConnection("jdbc:db2:localhost");
      stmt = conn.createStatement();
      rs = stmt.executeQuery("select col1 from cujosql.test_table");
      while (rs.next()) {
        System.out.println("El valor de la tabla es"  + rs.getShort(1));
      }

      conn.close();

    }
    catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

Características de DB2CachedRowSet:

Además de funcionar igual que un ResultSet, la clase DB2CachedRowSet tiene algunas funciones adicionales que hacen más flexible su utilización. Se proporcionan métodos para convertir todo el RowSet de Java Database Connectivity (JDBC) o tan solo una parte de él en una colección Java. Más aún, debido a su naturaleza desconectada, DB2CachedRowSets no tiene una relación estricta de uno a uno con ResultSets.

Además de funcionar como un ResultSet como se ha mostrado en varios ejemplos, la clase DB2CachedRowSet tiene algunas funciones adicionales que hacen más flexible su utilización. Se proporcionan métodos para convertir todo el RowSet de Java Database Connectivity (JDBC) o tan solo una parte de él en una colección Java. Más aún, debido a su naturaleza desconectada, DB2CachedRowSets no tiene una relación estricta de uno a uno con ResultSets.

Con los métodos suministrados por DB2CachedRowSet, puede realizar las siguientes tareas:

Obtener colecciones a partir de DB2CachedRowSets

Existen tres métodos que devuelven alguna forma de colección desde un objeto DB2CachedRowSet. Son los siguientes:

- **toCollection** devuelve una ArrayList (es decir, una entrada por cada fila) de vectores (es decir, una entrada por cada columna).
- **toCollection(int columnIndex)** devuelve un vector que contiene el valor para cada fila a partir de la columna dada.
- **getColumn(int columnIndex)** devuelve una matriz que contiene el valor para cada columna para una columna determinada.

La diferencia principal entre toCollection(int columnIndex) y getColumn(int columnIndex) consiste en que el método getColumn puede devolver una matriz de tipos primitivos. Por tanto, si columnIndex representa una columna que tiene datos enteros, se devuelve una matriz de enteros y no una matriz que contiene objetos java.lang.Integer.

El ejemplo siguiente muestra cómo puede utilizar estos métodos.

Ejemplo: obtener colecciones a partir de DB2CachedRowSets

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
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[])
  {
    // Registrar el controlador.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No es necesario continuar.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");
      Statement stmt = conn.createStatement();

      // Borrar ejecuciones anteriores
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Capturado drop table: " + ex.getMessage());
```

```
      }

      // Crear tabla de prueba
      stmt.execute("Create table cujosql.test_table (col1 smallint, col2 smallint)");
      System.out.println("Tabla creada.");

      // Insertar algunas filas de prueba
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ", " + (i + 100) + ")");
      }
      System.out.println("Filas insertadas");

      ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
      System.out.println("Consulta ejecutada");

      // Crear un nuevo conjunto de filas (rowset) y llenarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet lleno.");

      conn.close();
      System.out.println("RowSet se desconecta...");

      System.out.println(Probar el método toCollection());
      Collection collection = crs.toCollection();
      ArrayList map = (ArrayList) collection;

      System.out.println("el tamaño es"  + map.size());
      Iterator iter = map.iterator();
      int row = 1;
      while (iter.hasNext()) {
        System.out.print("fila [" + (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("Probar el método toCollection(int)");
      collection = crs.toCollection(2);
      Vector vector = (Vector) collection;

      iter = vector.iterator();

      while (iter.hasNext()) {
        System.out.println("Iterar: el valor es"  + iter.next());
      }

      System.out.println("Probar el método getColumn(int)");
      Object values = crs.getColumn(2);
      short[] shorts = (short [])values;

      for (int i =0; i < shorts.length; i++) {
        System.out.println("Matriz: el valor es"  + shorts[i]));
      }
    }
    catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

Crear copias de RowSets

El método createCopy crea una copia de DB2CachedRowSet. Se copian todos los datos asociados con el RowSet, junto con todas las estructuras de control, propiedades e identificadores de estado.

El ejemplo siguiente muestra cómo puede utilizar este método.

Ejemplo: crear copias de RowSets

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;

public class RowSetSample5
{
  public static void main(String args[])
  {
    // Registrar el controlador.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No es necesario continuar.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Borrar ejecuciones anteriores
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Capturado drop table: " + ex.getMessage());
      }

      // Crear tabla de prueba
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Tabla creada.");

      // Insertar algunas filas de prueba
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Filas insertadas");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Consulta ejecutada");

      // Crear un nuevo conjunto de filas (rowset) y llenarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet lleno.");

      conn.close();
      System.out.println("RowSet se desconecta...");
```

```
      System.out.println("Ahora, algunos RowSets nuevos a partir de uno.");;
      DB2CachedRowSet crs2 = crs.createCopy();
      DB2CachedRowSet crs3 = crs.createCopy();

      System.out.println("Cambiar el segundo a valores negados");
      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("Ahora, observar los tres de nuevo");

      while (crs.next()) {
        crs2.next();
        crs3.next();
        System.out.println("Valores: crs: " + crs.getShort(1) + ", crs2: " + crs2.getShort(1) +
                   ", crs3: " + crs3.getShort(1));
      }
    }
    catch (Exception ex) {
      System.out.println("SQLException: " + ex.getMessage());
      ex.printStackTrace();
    }

  }

}
```

Crear compartimientos para RowSets

El método createShared crea un nuevo objeto RowSet con información de estado de alto nivel y permite que dos objetos RowSet compartan los mismos datos físicos subyacentes.

El ejemplo siguiente muestra cómo puede utilizar este método.

Ejemplo: crear compartimientos de RowSets

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;

public class RowSetSample5
{
  public static void main(String args[])
  {
    // Registrar el controlador.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No es necesario continuar.
      System.exit(1);
    }
```

```
    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Borrar ejecuciones anteriores
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Capturado drop table: " + ex.getMessage());
      }

      // Crear tabla de prueba
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Tabla creada.");

      // Insertar algunas filas de prueba
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Filas insertadas");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Consulta ejecutada");

      // Crear un nuevo conjunto de filas (rowset) y llenarlo...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet lleno.");

      conn.close();
      System.out.println("RowSet se desconecta...");

      System.out.println("Probar la función createShared (crear 2 compartimientos)");
      DB2CachedRowSet crs2 = crs.createShared();
      DB2CachedRowSet crs3 = crs.createShared();

      System.out.println("Utilizar el original para actualizar el valor 5 de la tabla");
      crs.absolute(5);
      crs.updateShort(1, (short)-5);
      crs.updateRow();

      crs.beforeFirst();
      crs2.afterLast();

      System.out.println("Ahora, mover los cursores en direcciones opuestas de los mismos datos.");;

      while (crs.next()) {
        crs2.previous();
        crs3.next();
        System.out.println("Valores: 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:

DB2JdbcRowSet es un RowSet conectado, lo que significa que solo puede utilizarse con el soporte de un objeto Connection subyacente, un objeto PreparedStatement o un objeto ResultSet. Su implementación es muy parecida a la descripción de un JdbcRowSet.

Utilizar DB2JdbcRowSet

Debido a que el objeto DB2JdbcRowSet soporta los eventos descritos en la especificación Java Database Connectivity (JDBC) 3.0 para todos los RowSets, puede actuar como un objeto intermediario entre una base de datos local y otros objetos a los que debe informarse de los cambios efectuados en los datos de la base de datos.

Como ejemplo, suponga que está trabajando en un entorno en el que tiene una base de datos principal y varios asistentes digitales personales (PDA) que utilizan un protocolo inalámbrico para conectarse a ella. Puede utilizarse un objeto DB2JdbcRowSet para mover a una fila y actualizarla utilizando una aplicación maestra que se ejecuta en el servidor. La actualización de fila provoca la generación de un evento por parte del componente RowSet. Si existe un servicio en ejecución que es responsable de enviar actualizaciones a los PDA, puede registrarse a sí mismo como ″escucha″ del RowSet. Cada vez que recibe un evento RowSet, puede generar la actualización adecuada y enviarla a los dispositivos inalámbricos.

Consulte el Ejemplo: eventos DB2JdbcRowSet para obtener más información.

Crear JDBCRowSets

Existen varios métodos suministrados para crear un objeto DB2JDBCRowSet. Cada uno de ellos está diseñado de la siguiente forma.

Utilizar propiedades de DB2JdbcRowSet y DataSources

Los DB2JdbcRowSets tienen propiedades que aceptan una consulta SQL y un nombre DataSource. Con ello, los DB2JdbcRowSets quedan preparados para utilizarse. A continuación se ofrece un ejemplo de este método. Se supone que la referencia al DataSource denominado BaseDataSource es un DataSource válido que se ha configurado anteriormente.

Ejemplo: utilizar propiedades de DB2JdbcRowSet DataSources

Nota: lea el apartado Declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Crear un nuevo DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Establecer las propiedades necesarias para
// procesar el RowSet.
jrs.setDataSourceName("BaseDataSource");
jrs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet. Este método hace
// que el RowSet utilice el DataSource y la consulta SQL
// especificada para prepararse para el proceso de datos.
jrs.execute();

// Repetir en bucle por los datos del RowSet.
while (jrs.next()) {
    System.out.println("v1 es " + jrs.getString(1));
}

// Finalmente, cerrar el RowSet.
jrs.close();
```

Utilizar propiedades de DB2JdbcRowSet y URL de JDBC

Los DB2JdbcRowSets tienen propiedades que aceptan una consulta SQL y un URL de JDBC. Con ello, los DB2JdbcRowSets quedan preparados para utilizarse. A continuación se ofrece un ejemplo de este método:

Ejemplo: Utilizar propiedades DB2JdbcRowSet y los URL de JDBC

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Crear un nuevo DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Establecer las propiedades necesarias para
// procesar el RowSet.
jrs.setUrl("jdbc:db2:*local");
jrs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet. Esto hace
// que el RowSet utilice el URL y la consulta SQL especificada
// anteriormente para prepararse para el proceso de datos.
jrs.execute();

// Repetir en bucle por los datos del RowSet.
while (jrs.next()) {
    System.out.println("v1 es " + jrs.getString(1));
}

// Finalmente, cerrar el RowSet.
jrs.close();
```

Utilizar el método setConnection(Connection) para utilizar una conexión de base de datos existente

Para promocionar la reutilización de objetos JDBC Connection, DB2JdbcRowSet permite pasar un objeto connection establecido al DB2JdbcRowSet. DB2JdbcRowSet utiliza esta conexión para prepararse para la utilización cuando se llama al método execute.

Ejemplo: utilizar el método setConnection

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
// Establecer una conexión JDBC con la base de datos.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Crear un nuevo DB2JdbcRowSet.
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Establecer las propiedades necesarias para
// que el RowSet utilice una conexión establecida.
jrs.setConnection(conn);
jrs.setCommand("select col1 from cujosql.test_table");

// Llamar al método execute de RowSet. Esto hace
// que el RowSet utilice la conexión que se le suministró
// anteriormente para prepararse para el proceso de datos.
jrs.execute();

// Repetir en bucle por los datos del RowSet.
while (jrs.next()) {
    System.out.println("v1 es " + jrs.getString(1));
}

// Finalmente, cerrar el RowSet.
jrs.close();
```

Acceso a datos y movimiento de cursores

La manipulación de la posición del cursor y el acceso a los datos de la base de datos a través de un DB2JdbcRowSet corre a cargo del objeto ResultSet subyacente. Las tareas que pueden realizarse con un objeto ResultSet también se aplican al objeto DB2JdbcRowSet.

Cambiar datos y reflejarlos en la base de datos subyacente

El objeto ResultSet subyacente maneja completamente el soporte para actualizar la base de datos a través de un DB2JdbcRowSet. Las tareas que pueden realizarse con un objeto ResultSet también se aplican al objeto DB2JdbcRowSet.

Eventos de DB2JdbcRowSet:

Todas las implementaciones de RowSet soportan el manejo de eventos para situaciones que son de interés para otros componentes. Este soporte permite a los componentes de aplicación ″conversar″ entre sí cuando se producen eventos en ellos. Por ejemplo, la actualización de una fila de base de datos mediante un RowSet puede provocar que se muestre al usuario una tabla de interfaz gráfica de usuario (GUI) para que se actualice automáticamente.

En el ejemplo siguiente, el método main efectúa la actualización en RowSet y es la aplicación central. El escucha forma parte del servidor inalámbrico utilizado por los clientes desconectados en el campo. Es posible enlazar estos dos aspectos de gestión sin necesidad de mezclar el código de los dos procesos. Aunque el soporte de eventos de RowSets se ha diseñado principalmente para actualizar GUI con datos de base de datos, funciona perfectamente para este tipo de problema de aplicación.

Ejemplo: eventos DB2JdbcRowSet

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2JdbcRowSet;

public class RowSetEvents {
    public static void main(String args[])
    {
        // Registrar el controlador.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                               ex.getMessage());
            // No es necesario continuar.
            System.exit(1);
        }


        try {
            // Obtener la conexión JDBC y la sentencia necesarias para configurar
            // este ejemplo.
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");
            Statement stmt = conn.createStatement();

            // Borrar ejecuciones anteriores.
            try {
                stmt.execute("drop table cujosql.test_table");
            } catch (SQLException ex) {
                System.out.println("Capturado drop table: " + ex.getMessage());
            }
```

```
            // Crear tabla de prueba
            stmt.execute("Create table cujosql.test_table (col1 smallint)");
            System.out.println("Tabla creada.");

            // Llenar la tabla con datos.
            for (int i = 0; i < 10; i++) {
                stmt.execute("insert into cujosql.test_table values (" + i + ")");
            }
            System.out.println("Filas insertadas");

            // Eliminar los objetos de configuración.
            stmt.close();
            conn.close();

            // Crear un nuevo rowset y establecer las propiedades necesarias para
            // procesarlo.
            DB2JdbcRowSet jrs = new DB2JdbcRowSet();
            jrs.setUrl("jdbc:db2:*local");
            jrs.setCommand("select col1 from cujosql.test_table");
            jrs.setConcurrency(ResultSet.CONCUR_UPDATEABLE);

            // Dar un escucha al objeto RowSet. Este objeto maneja
            // el proceso especial cuando se realizan determinadas acciones en
            // el RowSet.
            jrs.addRowSetListener(new MyListener());

            // Procesar el RowSet para proporcionar acceso a los datos de la base
            // de datos.
            jrs.execute();

            // Provocar eventos de cambio de cursor. Estos eventos hacen que cursorMoved,
            // método del objeto escucha, obtenga en control.
            jrs.next();
            jrs.next();
            jrs.next();

            // Provocar un evento de cambio de fila. Este evento hace que rowChanged,
            // método del objeto escucha, obtenga en control.
            jrs.updateShort(1, (short)6);
            jrs.updateRow();

            // Finalmente, provocar un evento de cambio de RowSet. Esto hace que
            // el método rowSetChanged del objeto escuchado obtenga el control.
            jrs.execute();

            // Al terminar, cerrar el RowSet.
            jrs.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}


/**
 * Este es un ejemplo de un escucha. Este ejemplo imprime mensajes que muestran
 * cómo se mueve el flujo de control a través de la aplicación y ofrece algunas
 * sugerencias acerca de lo que puede hacerse si la aplicación se ha implementado plenamente.
 */
class MyListener
implements RowSetListener {
    public void cursorMoved(RowSetEvent rse) {
        System.out.println("Evento a realizar:  Posición de cursor cambiada.");;
        System.out.println("  Para el sistema remoto, no hacer nada ");
        System.out.println("  cuando este evento ha sucedido. La vista remota de los datos");
        System.out.println("  puede controlarse independientemente de la vista local.");;
        try {
```

```
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("la fila es " + rs.getRow() + . \n\n); \n\n");
        } catch (SQLException e) {
            System.out.println("Hacer:  Manejar adecuadamente los posibles problemas.");;
        }
    }

    public void rowChanged(RowSetEvent rse) {
        System.out.println("Evento a realizar:  Fila cambiada.");;
        System.out.println("  Indicar al sistema remoto que una fila ha cambiado. A continuación,");
        System.out.println("  pasar todos los valores solo de esa fila al ");
        System.out.println("  sistema remoto.");;
        try {
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("los nuevos valores son"  + rs.getShort(1) + ". \n\n");
        } catch (SQLException e) {
              System.out.println("Hacer:  Manejar adecuadamente los posibles problemas.");;
        }
    }

    public void rowSetChanged(RowSetEvent rse) {
        System.out.println("Evento a realizar:  RowSet cambiado.");;
        System.out.println("  Si existe un RowSet remoto ya establecido, ");
        System.out.println("  indicar al sistema remoto que los valores que ");
        System.out.println("  tiene que haberse lanzado. Luego, pasar todo ");
        System.out.println("  los valores actuales.\n\n");
    }
}
```

Consejos sobre el rendimiento del controlador JDBC de IBM Developer Kit para Java

El controlador JDBC de IBM Developer Kit para Java se ha diseñado para ser una interfaz Java de alto rendimiento que permita trabajar con la base de datos. Sin embargo, para obtener el mejor rendimiento posible, es necesario crear las aplicaciones de manera que aprovechen el potencial que puede ofrecer el controlador JDBC. Los siguientes consejos pretenden llevar a la práctica las mejores técnicas de programación JDBC. La mayoría de ellos no son específicos del controlador JDBC nativo. Por tanto, las aplicaciones que se escriban de acuerdo con estas directrices también tendrán un buen rendimiento si se emplean con controladores JDBC distintos del nativo.

Evitar consultas SQL SELECT *

SELECT * FROM... es una forma muy habitual de establecer una consulta en SQL. Sin embargo, muchas veces no es necesario consultar todos los campos. Para cada columna que hay que devolver, el controlador JDBC tiene que hacer el trabajo adicional de enlazar y devolver la fila. Aún en el caso de que la aplicación no llegue a utilizar nunca una columna concreta, el controlador JDBC debe tenerla presente y reservar espacio por si se utiliza. Esta cuestión no supone una actividad general significativa si son pocas las columnas que no se utilizan de las tablas. Sin embargo, si son numerosas las columnas no utilizadas, la actividad general puede llegar a ser significativa. En tal caso, sería mejor listar individualmente las columnas que a la aplicación le interesa consultar, como en este ejemplo:

```
SELECT COL1, COL2, COL3 FROM...
```

Utilizar getXXX(int) en vez de getXXX(String)

Utilice los métodos getXXX de ResultSet que toman valores numéricos, en vez de las versiones que toman nombres de columna. Si bien la libertad que supone utilizar nombres de columna en vez de constantes numéricas parece ser una ventaja, la base de datos propiamente dicha solo sabe manejar los índices de las columnas. Por ello, cada vez que se llama a un método getXXX utilizando el nombre de una columna, el controlador JDBC lo debe resolver para que el método se pueda pasar a la base de datos. Normalmente, a los métodos getXXX se les suele llamar dentro de bucles que se pueden ejecutar millones de veces, y ello

provocaría rápidamente la acumulación de la pequeña actividad general que supone la resolución de cada uno de los nombres de columna.

Evitar llamadas a getObject para tipos Java primitivos

Cuando de la base de datos se obtienen valores de tipos primitivos (int, long, float, etc.), es más rápido utilizar el método get específico del tipo primitivo (getInt, getLong, getFloat) que utilizar el método getObject. La llamada a getObject realiza el trabajo de obtener el tipo primitivo y luego crea un objeto para devolverlo. Normalmente, esto se hace en los bucles, lo que supone crear millones de objetos de corta vida. Si se emplea getObject para los mandatos de primitivos, existe el inconveniente adicional de que se activa con frecuencia el colector de basura, lo que aún reduce más el rendimiento.

Utilizar PreparedStatement más que Statement

Si escribe una sentencia SQL que se va a utilizar más de una vez, el rendimiento será mayor si la sentencia es un objeto PreparedStatement que si es un objeto Statement. Cada vez que se ejecuta una sentencia, se realiza un proceso de dos pasos: la sentencia se prepara y luego se procesa. Si se emplea una sentencia preparada, el paso de preparar la sentencia solo tiene lugar en el momento de construir la sentencia, y no se repite cada vez que se ejecuta la sentencia. Los programadores, aunque saben que el rendimiento de PreparedStatement es mayor que el de Statement, suelen desaprovechar esta ventaja en muchas ocasiones. Debido a la mejora de rendimiento que proporcionan las sentencias PreparedStatement, conviene utilizarlas en el diseño de las aplicaciones siempre que sea posible (consulte la sección “Considerar la posibilidad de utilizar la agrupación de sentencias PreparedStatement” en la página 179).

Evitar llamadas a DatabaseMetaData

Conviene tener en cuenta que algunas de las llamadas a DatabaseMetaData pueden ser costosas. En particular, los métodos getBestRowIdentifier, getCrossReference, getExportedKeys y getImportedKeys pueden resultar costosos. Algunas llamadas a DataBaseMetaData implican complejas condiciones de unión sobre tablas a nivel del sistema. Únicamente se deben emplear cuando se necesita la información que proporcionan, no porque resulte más práctico.

Utilizar el nivel de compromiso correcto para la aplicación

JDBC proporciona varios niveles de compromiso, que determinan cómo se afectan mutuamente varias transacciones con respecto al sistema (consulte la sección Transacciones para obtener más detalles). El valor predeterminado es utilizar el nivel de compromiso más bajo. Esto implica que las transacciones pueden ver parte del trabajo de cada una de ellas a través de los límites del compromiso. También implica la posibilidad de que se produzcan ciertas anomalías de base de datos. Algunos programadores aumentan el nivel de compromiso para no tener que preocuparse de si se produce este tipo de anomalías. Tenga en cuenta que los niveles de compromiso más altos implican que la base de datos se cuelgue en bloqueos más bastos. Esto limita la cantidad de concurrencia que el sistema puede tener, disminuyendo en gran medida el rendimiento de algunas aplicaciones. A menudo, las condiciones de anomalía no se pueden producir debido en primer lugar al diseño de la aplicación. Tómese su tiempo para comprender lo que está tratando de lograr y limite el nivel de aislamiento de las transacciones al mínimo que pueda emplear sin arriesgarse.

Considerar la posibilidad de almacenar datos en Unicode

En Java, todos los datos de tipo carácter con los que se trabaja (objetos String) deben tener el formato Unicode. Por lo tanto, para las tablas cuyos datos no tengan el formato Unicode, se necesitará que el controlador JDBC convierta los datos a ese formato y desde él al ponerlos en la base de datos y al recuperarlos de la base de datos. Si la tabla ya tiene los datos en Unicode, el controlador JDBC no tendrá que convertirlos, por lo que será más rápido colocar en ella los datos de la base de datos. Fíjese, sin embargo, que los datos en Unicode pueden no funcionar con las aplicaciones no Java, ya que estas no

saben cómo manejar el formato Unicode. Tenga presente también que el rendimiento no se ve afectado para los datos que no son de tipo carácter, ya que esos datos nunca se tienen que convertir. Aún hay que tener en cuenta otra particularidad, y es que los datos almacenados en Unicode ocupan el doble de espacio que los datos de un solo byte. Sin embargo, si son numerosas las columnas de tipo carácter que se leen muchas veces, puede llegar a ser notable el aumento de rendimiento que supone almacenar los datos en Unicode.

Utilizar procedimientos almacenados

El uso de procedimientos almacenados está permitido en Java. El rendimiento de los procedimientos almacenados puede ser mayor al permitir que el controlador JDBC ejecute SQL estático en vez de SQL dinámico. No cree procedimientos almacenados para cada sentencia SQL individual que ejecute en el programa. No obstante, cuando sea posible, cree un procedimiento almacenado que ejecute un grupo de sentencias SQL.

Utilizar BigInt en lugar de Numérico o Decimal

En vez de utilizar campos numéricos o decimales cuya escala sea 0, utilice el tipo de datos BigInt. BigInt se convierte directamente al tipo Java primitivo Long, mientras que los tipos de datos numéricos o decimales se convierten en objetos String o BigDecimal. Como se ha indicado en la sección “Evitar llamadas a DatabaseMetaData” en la página 178, es preferible utilizar tipos de datos primitivos a utilizar tipos que requieren la creación de objetos.

Cerrar explícitamente los recursos JDBC cuando ya no se necesitan

La aplicación debe cerrar explícitamente los objetos ResultSet, Statement y Connection cuando ya no se necesitan. Así se hace una limpieza de recursos del modo más eficaz posible, y el rendimiento puede aumentar. Además, los recursos de base de datos que no se cierran de manera explícita pueden provocar fugas de recursos, y los bloqueos de base de datos se pueden prolongar más de lo debido. Ello puede producir anomalías de aplicación o reducir la concurrencia en las aplicaciones.

Utilizar agrupación de conexiones

La agrupación de conexiones es una estrategia que permite reutilizar los objetos Connection de JDBC por parte de múltiples usuarios, en vez de dejar que cada usuario solicite crear su propio objeto Connection. La creación de objetos Connection es costosa. En vez de hacer que cada usuario cree una conexión nueva, conviene que las aplicaciones sensibles al rendimiento compartan una agrupación de conexiones. Muchos productos (como WebSphere) proporcionan soporte de agrupación de conexiones, que puede utilizarse con poco esfuerzo adicional por parte del usuario. Si no desea utilizar un producto que tenga soporte para la agrupación de conexiones o si prefiere construir una propia con objeto de controlar mejor su funcionamiento y su ejecución, piense que es relativamente fácil hacerlo.

Considerar la posibilidad de utilizar la agrupación de sentencias PreparedStatement

La agrupación de sentencias funciona de manera muy parecida a la agrupación de conexiones. En vez de poner en la agrupación tan solo las conexiones, en ella se pone un objeto que contenga la conexión y las sentencias PreparedStatement. Luego se recupera ese objeto y se accede a la sentencia concreta que se desea utilizar. Esta estrategia puede aumentar drásticamente el rendimiento.

Utilizar SQL eficaz

Dado que JDBC se construye encima de SQL, cualquier técnica que mejore la eficacia de SQL mejorará también la eficacia de JDBC. Por tanto, JDBC se beneficia de las consultas optimizadas, de los índices acertadamente elegidos y de otros aspectos que mejoren el diseño de SQL.

Acceder a bases de datos utilizando el soporte SQLJ de DB2 de IBM Developer Kit para Java

El soporte de lenguaje de consulta estructurada para Java (SQLJ) de DB2 para Java se basa en el estándar ANSI de SQLJ. El soporte SQLJ de DB2 se encuentra en IBM Developer Kit para Java. El soporte SQLJ de DB2 le permite crear, construir y ejecutar SQL intercalado para aplicaciones Java.

El soporte SQLJ proporcionado por IBM Developer Kit para Java incluye las clases SQLJ de tiempo de ejecución y está disponible en /QIBM/ProdData/Java400/ext/runtime.zip.

Puesta a punto de SQLJ

Para poder utilizar SQLJ de las aplicaciones Java en el servidor, debe preparar el servidor para que utilice SQLJ. Hallará más información en el tema Configuración de SQLJ.

Herramientas SQLJ

En el soporte SQLJ proporcionado por IBM Developer Kit para Java también se incluyen las siguientes herramientas:

- El conversor SQLJ, `sqlj`, sustituye las sentencias SQL intercaladas del programa SQLJ por sentencias fuente Java y genera un perfil serializado que contiene información sobre las operaciones SQLJ que se encuentran en el programa SQLJ.
- El personalizador de perfiles SQLJ de DB2, `db2profc`, precompila las sentencias SQL almacenadas en el perfil generado y genera un paquete en la base de datos DB2.
- El impresor de perfiles SQLJ de DB2, `db2profp`, imprime el contenido de un perfil personalizado de DB2 en texto sin formato.
- El instalador de auditores de perfiles SQLJ, `profdb`, instala y desinstala auditores de clases de depuración en un conjunto existente de perfiles binarios.
- La herramienta de conversión de perfiles SQLJ, `profconv`, convierte una instancia de perfil serializado al formato de clase Java.

Nota: Estas herramientas deben ejecutarse en el intérprete Qshell.

Restricciones de SQLJ de DB2

Cuando cree aplicaciones DB2 con SQLJ, debe tener presentes las siguientes restricciones:

- El soporte SQLJ de DB2 se ajusta a las restricciones de DB2 Universal Database estándar relacionadas con la emisión de sentencias SQL.
- El personalizador de perfil SQLJ de DB2 solo se debe ejecutar en los perfiles asociados a conexiones con la base de datos local.
- Para la implementación de referencia SQLJ se necesita JDK 1.1 o superior. Vea el tema: Soporte para múltiples Java Development Kits (JDKs), donde hallará más información sobre cómo ejecutar múltiples versiones de Java Development Kit.

Conceptos relacionados

"Perfiles de lenguaje de consulta estructurada para Java (SQLJ)" en la página 181
El conversor de SQLJ, sqlj, genera perfiles cuando se convierte el archivo fuente SQLJ. Los perfiles son archivos binarios serializados. Esta es la razón por la que estos archivos tienen la extensión .ser. Estos archivos contienen las sentencias SQL del archivo fuente SQLJ asociado.

"Soporte para múltiples opciones del LP 5761-JV1" en la página 6
La plataforma del System i5 admite múltiples versiones de los Java Development Kits (JDK) y de la plataforma Java 2, Standard Edition.

"Intercalar sentencias SQL en la aplicación Java" en la página 186
Las sentencias de SQL estático en SQLJ están en las cláusula SQLJ. Las cláusulas SQLJ empiezan por `#sql` y terminan en punto y coma (;).

Tareas relacionadas

"Configurar el sistema para que utilice SQLJ" en la página 193
Antes de ejecutar un programa Java que contenga sentencias SQLJ intercaladas, no olvide configurar el servidor para que dé soporte a SQLJ. El soporte de SQLJ requiere que modifique la variable de entorno CLASSPATH para el servidor.

"Compilar y ejecutar programas SQLJ" en la página 190
Si el programa Java tiene sentencias SQLJ intercaladas, debe seguir un procedimiento especial para compilarlo y ejecutarlo.

Perfiles de lenguaje de consulta estructurada para Java (SQLJ)

El conversor de SQLJ, sqlj, genera perfiles cuando se convierte el archivo fuente SQLJ. Los perfiles son archivos binarios serializados. Esta es la razón por la que estos archivos tienen la extensión .ser. Estos archivos contienen las sentencias SQL del archivo fuente SQLJ asociado.

Para generar perfiles a partir del código fuente SQLJ, ejecute el "Conversor de lenguaje de consulta estructurada para Java (SQLJ) (sqlj)" en el archivo .sqlj.

Para obtener más información, consulte "Compilar y ejecutar programas SQLJ" en la página 190.

Conversor de lenguaje de consulta estructurada para Java (SQLJ) (sqlj)

El conversor de SQL para Java, sqlj, genera un perfil serializado que contiene información sobre las operaciones SQL que se encuentran en el programa SQLJ. El conversor SQLJ utiliza el archivo /QIBM/ProdData/Java400/ext/translator.zip.

Para obtener más información sobre el perfil, consulte este enlace: Profile.

Precompilar sentencias SQL en un perfil mediante el personalizador de perfiles SQLJ de DB2, db2profc

Puede utilizar el personalizador de perfiles SQLJ de DB2, db2profc, para conseguir que su aplicación Java funcione de manera más eficaz con la base de datos.

El personalizador de perfiles SQLJ de DB2 hace lo siguiente:

- Precompila las sentencias SQL almacenadas en un perfil y genera un paquete en la base de datos DB2.
- Personaliza el perfil SQLJ sustituyendo las sentencias SQL por referencias a la sentencia asociada en el paquete que se ha creado.

Para precompilar las sentencias SQL de un perfil, escriba lo siguiente en el indicador de mandatos de Qshell:

```
db2profc MyClass_SJProfile0.ser
```

Donde *MyClass_SJProfile0.ser* es el nombre del perfil que desea precompilar.

Utilización y sintaxis del personalizador de perfiles SQLJ de DB2

db2profc[*opciones*] <*nombre_perfil_SQLJ*>

Donde *nombre_perfil_SQLJ* es el nombre del perfil que debe imprimirse, y *opciones* es la lista de opciones que desea.

Las opciones disponibles para db2profp son las siguientes:

- `-URL=<URL_JDBC>`

- -user=<*nombreusuario*>
- -password=<*contraseña*>
- -package=<*nombre_biblioteca/nombre_paquete*>
- -commitctrl=<*control_compromiso*>
- -datefmt=<*formato_fecha*>
- -datesep=<*separador_fecha*>
- -timefmt=<*formato_hora*>
- -timesep=<*separador_hora*>
- -decimalpt=<*separador_decimal*>
- -stmtCCSID=<*CCSID*>
- -sorttbl=<*nombre_biblioteca/nombre_tabla_secuencia_ordenación*>
- -langID=<*identificar_idioma*>

A continuación se describen estas opciones:

-URL=<*URL_JDBC***>**
Donde *URL_JDBC* es el URL de la conexión JDBC. La sintaxis del URL es:

```
jdbc:db2:nombresistema
```

Para obtener más información, consulte "Acceder a la base de datos de System i5 con el controlador JDBC de IBM Developer Kit para Java" en la página 34.

-user=<*nombreusuario***>**
Donde *nombreusuario* es el nombre de usuario. El valor predeterminado es el ID del usuario actual que ha iniciado la sesión en la conexión local.

-password=<*contraseña***>**
Donde *contraseña* es su contraseña. El valor predeterminado es la contraseña del usuario actual que ha iniciado la sesión en la conexión local.

-package=<*nombre_biblioteca/nombre_paquete***>**
Donde *nombre_biblioteca* es la biblioteca donde se encuentra el paquete y *nombre_paquete* es el nombre del paquete que debe generarse. El nombre de biblioteca por omisión es QUSRSYS. El nombre de paquete por omisión se genera a partir del nombre del perfil. La longitud máxima del nombre de paquete es de 10 caracteres. Debido a que el nombre del perfil SQLJ siempre es superior a los 10 caracteres, el nombre de paquete por omisión que se crea es diferente del nombre de perfil. El nombre de paquete por omisión se crea concatenando las primeras letras del nombre del perfil con el número de clave del perfil. Si el número de clave del perfil es superior a 10 caracteres, se utilizan los 10 últimos caracteres del número de clave del perfil como nombre de paquete por omisión. Por ejemplo, el siguiente diagrama muestra algunos nombres de perfil y sus nombres de paquete por omisión:

Nombre de perfil	Nombre de paquete por omisión
App_SJProfile0	App_SJPro0
App_SJProfile01234	App_S01234
App_SJProfile012345678	A012345678
App_SJProfile01234567891	1234567891

-commitctrl=<*control_compromiso***>**
Donde *control_compromiso* es el nivel de control de compromiso que se desea. El control de compromiso puede tener cualquiera de los siguientes valores de tipo carácter:

Valor	Definición
C	*CHG. Son posibles las lecturas sucias, las lecturas no repetibles y las lecturas fantasma.
S	*CS. No son posibles las lecturas sucias, pero sí las lecturas no repetibles y las lecturas fantasma.
A	*ALL. No son posibles las lecturas sucias ni las lecturas no repetibles, pero sí las lecturas fantasma.
N	*NONE. No son posibles las lecturas sucias, las lecturas no repetibles ni las lecturas fantasma. Este es el valor predeterminado.

-datefmt=<*formato_fecha***>**

Donde *formato_fecha* es el tipo de formato de fecha que se desea. El formato de fecha puede tener cualquiera de los siguientes valores:

Valor	Definición
USA	Estándar IBM de Estados Unidos (mm.dd.aaaa,hh:mm a.m., hh:mm p.m.)
ISO	International Standards Organization (aaaa-mm-dd, hh.mm.ss). Este es el valor predeterminado.
EUR	Estándar IBM Europeo (dd.mm.aaaa, hh.mm.ss)
JIS	Era cristiana estándar japonesa (aaaa-mm-dd, hh:mm:ss)
MDY	Mes/Día/Año (mm/d/aa)
DMY	Día/Mes/Año (dd/mm/aa)
YMD	Año/Mes/Día (aa/mm/dd)
JUL	Juliana (aa/ddd)

El formato de fecha se utiliza al acceder a las columnas de resultados de tipo fecha. Todos los campos de fecha de salida se devuelven en el formato especificado. Para las series de fecha de entrada, se utiliza el valor indicado para determinar si la fecha se ha especificado con un formato válido. El valor predeterminado es ISO.

-datesep=<*separador_fecha***>**

Donde *separador_fecha* es el tipo de separador que desea utilizar. El separador de fecha se utiliza al acceder a las columnas de resultados de tipo fecha. El separador de fecha puede ser cualquiera de los siguientes:

Valor	Definición
/	Se utiliza una barra inclinada.
.	Se utiliza un punto.
,	Se utiliza una coma.
-	Se utiliza un guión. Este es el valor predeterminado.
blank	Se utiliza un espacio.

-timefmt=<*formato_hora***>**

Donde *formato_hora* es el formato que desea utilizar para visualizar campos de hora. El formato de hora se utiliza al acceder a las columnas de resultados de tipo hora. Para las series de hora de entrada, se utiliza el valor indicado para determinar si la hora se ha especificado con un formato válido. El formato de hora puede tener cualquiera de los siguientes valores:

Valor	Definición
USA	Estándar IBM de Estados Unidos (mm.dd.aaaa,hh:mm a.m., hh:mm p.m.)
ISO	International Standards Organization (aaaa-mm-dd, hh.mm.ss). Este es el valor predeterminado.
EUR	Estándar IBM Europeo (dd.mm.aaaa, hh.mm.ss)
JIS	Era cristiana estándar japonesa (aaaa-mm-dd, hh:mm:ss)
HMS	Hora/Minutos/Segundos (hh:mm:ss)

-timesep=<*separador_hora***>**

Donde *separador_hora* es el carácter que desea utilizar para acceder a las columnas de resultado de hora. El separador de hora puede ser cualquiera de los siguientes:

Valor	Definición
:	Se utilizan dos puntos.
.	Se utiliza un punto. Este es el valor predeterminado.
,	Se utiliza una coma.
blank	Se utiliza un espacio.

-decimalpt=<*separador_decimal***>**

Donde *separador_decimal* es el separador decimal que desea utilizar. El separador decimal se utiliza para las constantes numéricas en las sentencias SQL. El separador decimal puede ser cualquiera de los siguientes:

Valor	Definición
.	Se utiliza un punto. Este es el valor predeterminado.
,	Se utiliza una coma.

-stmtCCSID=<*CCSID***>**

Donde *CCSID* es el identificador de juego de caracteres para las sentencias SQL preparadas en el paquete. El valor predeterminado es el valor del trabajo durante el tiempo de personalización.

-sorttbl=<*nombre_biblioteca/nombre_tabla_secuencia_ordenación***>**

Donde *nombre_biblioteca/nombre_tabla_secuencia_ordenación* es la ubicación y el nombre de tabla de la tabla de secuencia de ordenación que desea utilizar. La tabla de secuencia de ordenación se utiliza para las comparaciones de series en las sentencias SQL. Los dos nombres, el de la biblioteca y el de la tabla de secuencia de ordenación, no pueden tener más de 10 caracteres. El valor predeterminado se toma del trabajo durante el tiempo de personalización.

-langID=<*identificar_idioma***>**

Donde *identificador_idioma* es el identificador de idioma que desea utilizar. El valor predeterminado para el identificador de idioma se toma del trabajo actual durante el tiempo de personalización. El identificador de idioma se utiliza junto con la tabla de secuencia de ordenación.

Información relacionada

Programación SQL

Imprimir el contenido de los perfiles SQLJ de DB2 (db2profp y profp)

El impresor de perfiles SQLJ de DB2, db2profp, imprime el contenido de un perfil personalizado de DB2 en texto sin formato. El impresor de perfiles, profp, imprime el contenido de los perfiles generados por el conversor SQLJ en texto sin formato.

Para imprimir el contenido de los perfiles generados por el conversor SQLJ en texto sin formato, utilice el programa de utilidad profp de la manera siguiente:

```
profp MyClass_SJProfile0.ser
```

Donde *MyClass_SJProfile0.ser* es el nombre del perfil que desea imprimir.

Para imprimir el contenido de una versión personalizada por DB2 del perfil en texto sin formato, utilice el programa de utilidad db2profp como se indica a continuación:

```
db2profp MyClass_SJProfile0.ser
```

Donde *MyClass_SJProfile0.ser* es el nombre del perfil que desea imprimir.

Nota: si ejecuta db2profp en un perfil no personalizado, el programa le indicará esta circunstancia. Si ejecuta profp en un perfil personalizado, el programa visualiza el contenido del perfil sin la personalización.

Utilización y sintaxis del impresor de perfiles SQLJ de DB2:

```
db2profp [opciones] <nombre_perfil_SQLJ>
```

Donde *nombre_perfil_SQLJ* es el nombre del perfil que debe imprimirse, y *opciones* es la lista de opciones que desea.

Las opciones disponibles para db2profp son las siguientes:

-URL=<*URL_JDBC***>**
Donde *JDBC_URL* es el URL al que desea conectarse. Para obtener más información, consulte "Acceder a la base de datos de System i5 con el controlador JDBC de IBM Developer Kit para Java" en la página 34.

-user=<*nombreusuario***>**
Donde *nombreusuario* es el nombre de usuario del perfil de usuario.

-password=<*contraseña***>**
Donde *contraseña* es la contraseña del perfil de usuario.

Instalador de auditores de perfiles SQLJ (profdb)

El instalador de auditores de perfiles SQLJ (profdb) instala y desinstala auditores de clase de depuración. Los auditores de clase de depuración se instalan en un conjunto existente de perfiles binarios. Una vez instalados los auditores de clase de depuración, se anotan todas las llamadas a RTStatement y RTResultSet efectuadas durante la ejecución de la aplicación. Pueden anotarse en un archivo o en la salida estándar. Luego se pueden inspeccionar las anotaciones con objeto de verificar el comportamiento y rastrear los errores de la aplicación. Tenga en cuenta que solo se auditan las llamadas efectuadas en tiempo de ejecución a las interfaces RTStatement y RTResultSetcall subyacentes.

Para instalar auditores de clase de depuración, entre lo siguiente en el indicador de mandatos de Qshell:

```
profdb MyClass_SJProfile0.ser
```

Donde *MyClass_SJProfile0.ser* es el nombre del perfil generado por el conversor SQLJ.

Para desinstalar auditores de clase de depuración, entre lo siguiente en el indicador de mandatos de Qshell:

```
profdb -Cuninstall MyClass_SJProfile.ser
```

Donde *MyClass_SJProfile0.ser* es el nombre del perfil generado por el conversor SQLJ.

Convertir una instancia de perfil serializado al formato de clase Java mediante la herramienta de conversión de perfiles SQLJ (profconv)

La herramienta de conversión de perfiles SQLJ (profconv) convierte una instancia de perfil serializado al formato de clase Java. La herramienta profconv es necesaria debido a que algunos navegadores no dan soporte a cargar un objeto serializado desde un archivo de recurso asociado a un applet. Ejecute el programa de utilidad profconv para realizar la conversión.

Para ejecutar el programa de utilidad profconv, escriba lo siguiente en la línea de mandatos de Qshell:

```
profconv MyApp_SJProfile0.ser
```

donde *MyApp_SJProfile0.ser* es el nombre de la instancia de perfil que desea convertir.

La herramienta `profconv` invoca `sqlj -ser2class`. Las opciones de línea de mandatos están en sqlj.

Intercalar sentencias SQL en la aplicación Java

Las sentencias de SQL estático en SQLJ están en las cláusula SQLJ. Las cláusulas SQLJ empiezan por `#sql` y terminan en punto y coma (;).

Antes de crear cláusulas SQLJ en la aplicación Java, importe estos paquetes:

- import java.sql.*;
- import sqlj.runtime.*;
- import sqlj.runtime.ref.*;

Las cláusulas SQLJ más sencillas son aquellas que se pueden procesar y que constan del símbolo `#sql` seguido de una sentencia SQL entre llaves. Por ejemplo, la siguiente cláusula SQLJ puede aparecer en cualquier lugar donde esté permitido que aparezca una sentencia Java:

```
#sql { DELETE FROM TAB };
```

El ejemplo anterior suprime todas las filas de una tabla denominada TAB.

En una cláusula de proceso SQLJ, los símbolos que aparecen dentro de las llaves son símbolos SQL o variables del lenguaje principal. Todas las variables del lenguaje principal se distinguen mediante el carácter de dos puntos (:). Los símbolos SQL nunca aparecen fuera de las llaves de una cláusula de proceso SQLJ. Por ejemplo, el siguiente método Java inserta sus argumentos en una tabla SQL:

```
public void insertIntoTAB1 (int x, String y, float z) throws SQLException
{
    #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

El cuerpo del método consta de una cláusula de proceso SQLJ que contiene las variables de lenguaje principal x, y, y z.

En general, los símbolos SQL son sensibles a mayúsculas y minúsculas (excepto los identificadores delimitados mediante comillas dobles) y pueden escribirse con mayúsculas, minúsculas o con una combinación de ellas. Sin embargo, los símbolos *Java* son sensibles a mayúsculas/minúsculas. A efectos de claridad en los ejemplos, los símbolos SQL no sensibles a mayúsculas/minúsculas están en mayúsculas, y los símbolos Java están en minúsculas o combinadas. A lo largo de este tema, se utilizan las minúsculas `null` para representar el valor ″null″ de Java y las mayúsculas `NULL` para representar el valor ″null″ de SQL.

En los programas SQLJ pueden aparecer los siguientes tipos de construcciones SQL:

- Consultas. Por ejemplo, expresiones y sentencias SELECT.
- Sentencias de cambio de datos SQL (DML). Por ejemplo, INSERT, UPDATE, DELETE.
- Sentencias de datos. Por ejemplo, FETCH, SELECT..INTO.

- Sentencias de control de transacción. Por ejemplo, COMMIT, ROLLBACK, etc.
- Sentencias de lenguaje de definición de datos (DDL, también conocido como lenguaje de manipulación de esquemas o SML). Por ejemplo, CREATE, DROP, ALTER.
- Llamadas a procedimientos almacenados. Por ejemplo, CALL MYPROC(:x, :y, :z)
- Invocaciones de las funciones almacenadas. Por ejemplo, VALUES(MYFUN(:x))

Variables de lenguaje principal del lenguaje de consulta estructurada para Java (SQLJ):

Los argumentos de las sentencias SQL intercaladas se pasan por medio de las variables del lenguaje principal. Las variables del lenguaje principal son las que se utilizan en el lenguaje principal y pueden aparecer en las sentencias SQL.

Las variables de lenguaje principal tienen tres partes como máximo:
- Un signo de dos puntos (:) como prefijo.
- Una variable de lenguaje principal Java que es un identificador Java de un parámetro, una variable o un campo.
- Un identificador de modalidad de parámetro opcional.

  Este identificador de modalidad puede ser uno de los siguientes:
  IN, OUT o INOUT.

La evaluación de un identificador Java no tiene efectos colaterales en un programa Java, por lo que puede aparecer múltiples veces en el código Java generado para sustituir una cláusula SQLJ.

La consulta siguiente contiene la variable de lenguaje principal `:x`. Esta variable de lenguaje principal es la variable, campo o parámetro `x` Java que es visible en el ámbito que contiene la consulta.

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

Ejemplo: intercalar sentencias SQL en la aplicación Java:

La siguiente aplicación SQLJ de ejemplo, App.sqlj, utiliza SQL estático para recuperar y actualizar datos de la tabla EMPLOYEE de la base de datos de ejemplo DB2.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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
{

  /**********************
   **  Controlador de registro **
   **********************/

 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;

    // El URL es jdbc:db2:nombredb
    String url = "jdbc:db2:sample";

    DefaultContext ctx = DefaultContext.getDefaultContext();
    if (ctx == null)
    {
      try
      {
        // Conectar con id/contraseña predeterminados.
        Connection con = DriverManager.getConnection(url);
        con.setAutoCommit(false);
        ctx = new DefaultContext(con);
      }
      catch (SQLException e)
      {
        System.out.println("Error: no se ha podido obtener un contexto predeterminado");
        System.err.println(e) ;
        System.exit(1);
      }
      DefaultContext.setDefaultContext(ctx);
    }

    // Recuperar datos de la base de datos.
    System.out.println("Recuperar algunos datos de la base de datos.");
    #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

    // Visualizar el conjunto de resultados.
    // cursor1.next() devuelve false cuando no hay más filas.
    System.out.println("Resultados recibidos:");
    while (cursor1.next()) // 3
    {
      str1 = cursor1.empno(); // 4
      str2 = cursor1.firstnme();

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor1.close(); // 9

    // Recuperar el número de empleado de la base de datos.
    #sql { SELECT count(*) into :count1 FROM employee }; // 5
    if (1 == count1)
      System.out.println ("Hay una fila en la tabla de empleados");
    else
      System.out.println ("Hay " + count1
                          + " filas en la tabla de empleados");

    // Actualizar la base de datos.
    System.out.println("Actualizar la base de datos.");
```

```
      #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

      // Recuperar los datos actualizados de la base de datos.
      System.out.println("Recuperar los datos actualizados de la base de datos.");
      str1 = "000010";
      #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

      // Visualizar el conjunto de resultados.
      // cursor2.next() devuelve false cuando no hay más filas.
      System.out.println("Resultados recibidos:");
      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

      // Retrotraer la actualización.
      System.out.println("Retrotraer la actualización.");
      #sql { ROLLBACK work };
      System.out.println("Retrotracción terminada.");
    }
    catch( Exception e )
    {
      e.printStackTrace();
    }
  }
}
```

Declarar iteradores. Esta sección declara dos tipos de iteradores:

- App_Cursor1: Declara nombres y tipos de datos de columna, y devuelve los valores de las columnas de acuerdo con el nombre de columna (enlace por nombre con columnas).
- App_Cursor2: Declara tipos de datos de columna, y devuelve los valores de las columnas por posición de columna (enlace posicional con columnas).

Inicializar el iterador. El objeto iterador cursor1 se inicializa utilizando el resultado de una consulta. La consulta almacena el resultado en cursor1.

Adelantar el iterador a la próxima fila. El método cursor1.next() devuelve un Boolean `false` si no existen más filas para recuperar.

[4]Mover los datos. El método de acceso por nombre empno() devuelve el valor de la columna denominada empno en la fila actual. El método de acceso por nombre firstnme() devuelve el valor de la columna denominada firstnme en la fila actual.

[5]Aplicar SELECT a los datos de una variable del lenguaje principal. La sentencia SELECT pasa el número de filas de la tabla a la variable de lenguaje principal count1.

[6] Inicializar el iterador. El objeto iterador cursor2 se inicializa utilizando el resultado de una consulta. La consulta almacena el resultado en cursor2.

[7]Recuperar los datos. La sentencia FETCH devuelve el valor actual de la primera columna declarada en el cursor ByPos desde la tabla de resultados a la variable de lenguaje principal str2.

[8]Comprobar el éxitode una sentencia FETCH.INTO. El método endFetch() devuelve Boolean true si el iterador no está situado en una fila, es decir, si el último intento de captar una fila ha fallado. El método

endFetch() devuelve false si el último intento de captar una fila ha sido satisfactorio. DB2 intenta captar una fila cuando se llama al método next(). Una sentencia FETCH...INTO llama implícitamente al método next().

[9]Cerrar los iteradores. El método close() libera los recursos retenidos por los iteradores. Los iteradores se deben cerrar explícitamente para asegurar que se liberan los recursos del sistema de forma oportuna.

Compilar y ejecutar programas SQLJ

Si el programa Java tiene sentencias SQLJ intercaladas, debe seguir un procedimiento especial para compilarlo y ejecutarlo.

Si el programa Java tiene sentencias SQLJ intercaladas, debe seguir un procedimiento especial para compilarlo y ejecutarlo.

1. Configure el servidor para que utilice SQLJ.
2. Utilice el conversor SQLJ, `sqlj` en el código fuente Java con SQL intercalado para generar código fuente Java y perfiles asociados. Se genera un perfil para cada conexión.

   Por ejemplo, escriba el siguiente mandato:

   ```
   sqlj MyClass.sqlj
   ```

   donde *MyClass.sqlj* es el nombre del archivo SQLJ.

   En este ejemplo, el conversor de SQLJ genera el archivo de código fuente MyClass.java y los perfiles asociados. Los perfiles asociados se denominan MyClass_SJProfile0.ser, MyClass_SJProfile1.ser, MyClass_SJProfile2.ser, etc.

   Nota: El conversor de SQLJ compila automáticamente el código fuente Java convertido en un archivo de clase, a menos que desactive explícitamente la opción de compilación con la cláusula `-compile=false`.
3. Utilice la herramienta Personalizador de perfiles SQLJ, `db2profc`, para instalar personalizadores de SQLJ de DB2 en los perfiles generados y para crear los paquetes DB2 en el sistema local.

   Por ejemplo, escriba el mandato:

   ```
   db2profc MyClass_SJProfile0.ser
   ```

   siendo *MyClass_SJProfile0.ser* el nombre del perfil en el que se ejecuta el personalizador de SQLJ de DB2.

   Nota: Este paso es opcional, pero se lo recomendamos porque aumenta el rendimiento en tiempo de ejecución.
4. Ejecute el archivo de clase Java igual que cualquier otro archivo de clase Java.

   Por ejemplo, escriba el mandato:

   ```
   java MyClass
   ```

   siendo *MyClass* el nombre del archivo de clase Java.

Conceptos relacionados

"Intercalar sentencias SQL en la aplicación Java" en la página 186
Las sentencias de SQL estático en SQLJ están en las cláusula SQLJ. Las cláusulas SQLJ empiezan por `#sql` y terminan en punto y coma (;).

Rutinas SQL Java

El sistema proporciona capacidad para acceder a programas Java desde programas y sentencias SQL. Esta operación puede realizarse mediante procedimientos almacenados Java y funciones definidas por usuario (UDF) Java. El System i5 permite utilizar ambos convenios, el de DB2 y el de SQLJ, para llamar a procedimientos almacenados Java y a funciones UDF Java. Tanto los procedimientos almacenados Java

como las UDF Java pueden utilizar clasesJava almacenadas en archivos JAR. El System i5 emplea procedimientos almacenados definidos por el estándar *SQLJ Parte 1* para registrar archivos JAR en la base de datos.

Utilizar rutinas SQL Java

Puede acceder a programas Java desde sentencias y programas SQL. Esta operación puede realizarse mediante procedimientos almacenados Java y funciones definidas por usuario (UDF)Java.

Para utilizar rutinas SQL Java, realice estas tareas:

1. Habilitar SQLJ

   Dado que las rutinas SQL Java pueden utilizar SQLJ, haga que el soporte de tiempo de ejecución SQLJ esté siempre disponible al ejecutar la plataforma Java 2, Standard Edition (J2SE). Para habilitar el soporte de tiempo de ejecución para SQLJ en J2SE, añada un enlace al archivo runtime.zip de SQLJ desde el directorio de extensiones. Hallará más información en: Configurar el sistema para que utilice SQLJ.

2. Escribir los métodos Java para las rutinas

   La rutina SQL Java procesa un método Java desde SQL. Este método debe haberse escrito utilizando los convenios de paso de parámetros de DB2 para i5/OS o SQLJ. Vea los procedimientos almacenados Java, las funciones definidas por usuario Java y las funciones de tabla definidas por usuario Java para obtener más información sobre cómo escribir el código de un método empleado por una rutina SQL Java.

3. Compilar las clases Java

   Las rutinas SQL Java escritas utilizando el estilo de los parámetros Java se pueden compilar sin ninguna configuración adicional. Sin embargo, las rutinas SQL Java que utilizan el estilo de los parámetros de DB2GENERAL deben ampliar la clase com.ibm.db2.app.UDF o la clase com.ibm.db2.app.StoredProc. Estas clases se encuentran en el archivo JAR /QIBM/ProdData/Java400/ext/db2routines_classes.jar. Si se utiliza javac para compilar estas rutinas, este archivo JAR debe existir en la CLASSPATH. Por ejemplo, el mandato que sigue compila un archivo fuente Java que contiene una rutina que emplea el estilo de los parámetros de DB2GENERAL:

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

4. Hacer que las clases compiladas sean accesible para la máquina virtual Java (JVM) que emplea la base de datos

   Las clases definidas por usuario utilizadas por la máquina virtual Java (JVM) de la base de datos pueden residir en el directorio /QIBM/UserData/OS400/SQLLib/Function o en un archivo JAR registrado para la base de datos.

   /QIBM/UserData/OS400/SQLLib/Function es el equivalente System i5 de /sqllib/function, directorio en el que DB2 para i5/OS almacena los procedimientos almacenados Java y las UDF Java en otras plataformas. La clase, si forma parte de un paquete Java, debe residir en el subdirectorio pertinente. Por ejemplo, si se crea la clase runit como parte del paquete foo.bar, el archivo runnit.class debe estar en el directorio del sistema de archivos integrado, /QIBM/ProdData/OS400/SQLLib/Function/foo/bar.

   El archivo de clase también puede colocarse en un archivo JAR registrado para la base de datos. El archivo JAR se registra mediante el procedimiento almacenado SQLJ.INSTALL_JAR. Este procedimiento almacenado se utiliza para asignar un ID de JAR a un archivo JAR. Este ID de JAR se utiliza para identificar el archivo JAR en el que reside el archivo de clase. Vea: Procedimientos SQLJ que manipulan archivos JAR, para obtener más información sobre SQLJ.INSTALL_JAR y sobre otros procedimientos almacenados para manipular archivos JAR.

5. Registrar la rutina en la base de datos.

   Las rutinas SQL Java se registran en la base de datos utilizando las sentencias SQL CREATE PROCEDURE y CREATE FUNCTION. Estas sentencias contienen los siguientes elementos:

Palabras clave CREATE
Las sentencias SQL que crean una rutina SQL Java empiezan por CREATE PROCEDURE o por CREATE STATEMENT.

Nombre de la rutina
A continuación, la sentencia SQL identifica el nombre de la rutina conocido por la base de datos. Es el nombre utilizado para acceder a la rutina Java desde SQL.

Parámetros y valores de retorno
Luego, la sentencia SQL identifica los parámetros y los valores de retorno, si procede, de la rutina Java.

LANGUAGE JAVA
La sentencia SQL utiliza las palabras clave LANGUAGE JAVA para indicar que la rutina se escribió en Java.

Palabras clave PARAMETER STYLE
A continuación, la sentencia SQL identifica el estilo de parámetro mediante las palabras clave PARAMETER STYLE JAVA o PARAMETER STYLE DB2GENERAL.

Nombre externo
Entonces, la sentencia SQL identifica el método Java que hay que procesar como rutinas SQL Java. El nombre externo puede tener dos formatos:

- Si el método se encuentra en un archivo de clase ubicado en el directorio /QIBM/UserData/OS400/SQLLib/Function, se identifica mediante el formato *nombreclase.nombremétodo*, donde *nombreclase* es el nombre totalmente calificado de la clase y *nombremétodo* es el nombre del método.
- Si el método se encuentra en un archivo JAR registrado para la base de datos, se identifica mediante el formato *jarid*:*nombreclase.nombremétodo*, donde *jarid* es el ID de JAR del archivo JAR registrado, *nombreclase* es el nombre de la clase y *nombremétodo* es el nombre del método.

Se puede utilizar System i Navigator para crear un procedimiento almacenado o una función definida por usuario que emplee el estilo de los parámetros Java.

6. Utilizar el procedimiento Java

Para llamar a un procedimiento almacenado Java se utiliza la sentencia SQL CALL. La UDF Java es una función a la que se llama como parte de ora sentencia SQL.

"Configurar el sistema para que utilice SQLJ" en la página 193
Antes de ejecutar un programa Java que contenga sentencias SQLJ intercaladas, no olvide configurar el servidor para que dé soporte a SQLJ. El soporte de SQLJ requiere que modifique la variable de entorno CLASSPATH para el servidor.

"Procedimientos almacenados Java" en la página 193
Cuando se utiliza Java para escribir procedimientos almacenados, se pueden utilizar dos estilos posibles para pasar parámetros.

"Funciones escalares Java definidas por usuario" en la página 197
La función escalar Java devuelve un valor a la base de datos desde un programa Java. Por ejemplo, se podría crear una función escalar que devolviera la suma de dos números.

"Funciones de tabla definidas por usuario Java" en la página 202
DB2 proporciona capacidad para que una función devuelva una tabla. Esto resulta de utilidad para exponer información externa a la base de datos en formato de tabla. Por ejemplo, se puede crear una tabla que exponga las propiedades establecidas en la máquina virtual Java (JVM) empleada para procedimientos almacenados Java y funciones definidas por usuario Java (tanto de tabla como escalares).

"Procedimientos SQLJ que manipulan archivos JAR" en la página 204
Tanto los procedimientos almacenados Java como las funciones definidas por usuario (UDF) Java pueden utilizar clases Java almacenadas en archivos JAR Java.

Configurar el sistema para que utilice SQLJ:

Antes de ejecutar un programa Java que contenga sentencias SQLJ intercaladas, no olvide configurar el servidor para que dé soporte a SQLJ. El soporte de SQLJ requiere que modifique la variable de entorno CLASSPATH para el servidor.

Para obtener más información sobre cómo trabajar con vías de acceso de clases Java, vea la siguiente página:

Vía de acceso de clases Java

Utilizar SQLJ y J2SE

Para configurar SQLJ en un servidor que ejecute cualquier versión soportada de J2SE, complete los siguientes pasos:

1. Añada los siguientes archivos a la variable de entorno CLASSPATH para el servidor:
   - /QIBM/ProdData/Os400/Java400/ext/sqlj_classes.jar
   - /QIBM/ProdData/Os400/Java400/ext/translator.zip

   Nota: Solo debe añadir el archivo translator.zip cuando desee ejecutar el conversor de SQLJ (mandato sqlj). No hace falta que añada el archivo translator.zip si solo quiere ejecutar programas Java compilados que empleen SQLJ. Hallará más información en: El conversor de SQLJ (sqlj).
2. En un indicador de mandatos de i5/OS, emita el siguiente mandato para añadir un enlace que le lleve al archivo runtime.zip desde el directorio de extensiones. Teclee el mandato en una sola línea y pulse **Intro**.

   ```
   ADDLNK OBJ('/QIBM/ProdData/Os400/Java400/ext/runtime.zip')
   NEWLNK('/QIBM/UserData/Java400/ext/runtime.zip')
   ```

   Para obtener más información sobre cómo instalar extensiones, consulte la siguiente página:

   Instalar extensiones para IBM Developer Kit para Java

Procedimientos almacenados Java

Cuando se utiliza Java para escribir procedimientos almacenados, se pueden utilizar dos estilos posibles para pasar parámetros.

El estilo recomendado es el de los parámetros de JAVA, que coincide con el estilo especificado en el estándar de rutinas SQLj: SQL. El segundo estilo, DB2GENERAL, está definido por DB2 UDB. El estilo de los parámetros también determina los convenios que hay que utilizar al codificar un procedimiento almacenado Java.

Además, también tener presentes algunas restricciones que se aplican a los procedimientos almacenados Java.

Estilo de los parámetros de JAVA:

Al codificar un procedimiento almacenado Java que utilice el estilo de los parámetros de JAVA, debe ajustarse a estos convenios.

- El método Java debe ser un método público void estático (no de instancia).
- Los parámetros del método Java deben ser tipos compatibles con SQL.
- Un método Java puede probar un valor SQL NULL cuando el parámetro es un tipo con posibilidad de nulos (como en el caso de String).
- Los parámetros de salida se devuelven utilizando matrices de un solo elemento.
- El método Java puede acceder a la base de datos actual utilizando el método getConnection.

Los procedimientos almacenados Java que utilizan el estilo de los parámetros de JAVA son métodos públicos estáticos. Dentro de las clases, los procedimientos almacenados se identifican mediante el nombre y la firma de método. Al llamar a un procedimiento almacenado, su firma se genera automáticamente, en función de los tipos de variable definidos por la sentencia CREATE PROCEDURE.

Si se pasa un parámetro en un tipo Java que permite el valor nulo, un método Java puede comparar el parámetro con null para determinar si un parámetro de entrada es SQL NULL.

Los tipos Java que no permiten usar el valor null son:

- short
- int
- long
- float
- double

Si se pasa un valor nulo a un tipo Java que no permite usar el valor nulo, se devolverá una excepción SQL con un código de error -20205.

Los parámetros de salida se pasan como matrices que contienen un elemento. El procedimiento almacenado Java puede establecer el primer elemento de la matriz para establecer el parámetro de salida.

Se accede a una conexión con el contexto de aplicación de incorporación utilizando la siguiente llamada Java Database Connectivity (JDBC):

```
connection=DriverManager.getConnection("jdbc:default:connection");
```

Luego, esta conexión ejecuta sentencias SQL con las API JDBC.

A continuación se ofrece un pequeño procedimiento almacenado con un parámetro de entrada y dos parámetros de salida. Ejecuta la consulta SQL dada y devuelve el número de filas del resultado y SQLSTATE.

Ejemplo: procedimiento almacenado con una entrada y dos salidas

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
   }
  }
 }
```

En el estándar SQLj, para devolver un conjunto de resultados en rutinas que utilizan el estilo de parámetro JAVA, el conjunto de resultados debe establecerse explícitamente. Cuando se crea un procedimiento que devuelve conjuntos de resultados, se añaden parámetros adicionales de conjunto de resultados al final de la lista de parámetros. Por ejemplo, la sentencia

```
CREATE PROCEDURE RETURNTWO()
DYNAMIC RESULT SETS 2
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'javaClass!returnTwoResultSets'
```

llamará a un método Java con la signatura `public static void returnTwoResultSets(ResultSet[] rs1, ResultSet[] rs2)`.

Los parámetros de salida del conjunto de resultados deben establecerse explícitamente, como se muestra en el ejemplo siguiente. Al igual que en el estilo DB2GENERAL, los conjuntos de resultados y las sentencias correspondientes no deben cerrarse.

Ejemplo: procedimiento almacenado que devuelve dos conjuntos de resultados

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.sql.*;
public class javaClass {
  /**
    * Procedimiento almacenado Java, con parámetros de estilo JAVA,
    * que procesa dos sentencias predefinidas
    * y devuelve dos conjuntos de resultados
    *
    * @param ResultSet[] rs1    primer ResultSet
    * @param ResultSet[] rs2    segundo ResultSet
    */
public static void returnTwoResultSets (ResultSet[] rs1, ResultSet[] rs2) throws Exception
{
  // obtener conexión del llamador con la base de datos; heredado de StoredProc
  Connection con = DriverManager.getConnection("jdbc:default:connection");

  //definir y procesar la primera sentencia select
  Statement stmt1 = con.createStatement();
  String sql1 = "select value from table01 where index=1";
  rs1[0] = stmt1.executeQuery(sql1);

  //definir y procesar la segunda sentencia select
  Statement stmt2 = con.createStatement();
  Stringsql2 = "select value from table01 where index=2";
  rs2[0] = stmt2.executeQuery(sql2);
  }
}
```

En el servidor, los parámetros adicionales de conjunto de resultados no se examinan para determinar el orden de los resultados. Los conjuntos de resultados del servidor se devuelven en el orden en el que se han abierto. Para garantizar la compatibilidad con el estándar SQLj, el resultado debe asignarse en el orden en que se abre, como se ha mostrado anteriormente.

Estilo de los parámetros de DB2GENERAL:

Al codificar un procedimiento almacenado Java que utilice el estilo de los parámetros de DB2GENERAL, debe ajustarse a estos convenios.

- La clase que define un procedimiento almacenado Java debe *ampliar*, o ser una subclase de, la clase Java com.ibm.db2.app.StoredProc.
- El método Java debe ser un método público de instancia void.

- Los parámetros del método Java deben ser tipos compatibles con SQL.
- Un método Java puede probar un valor SQL NULL utilizando el método isNull.
- El método Java debe establecer explícitamente los parámetros de retorno mediante el método set.
- El método Java puede acceder a la base de datos actual utilizando el método getConnection.

Una clase que incluya un procedimiento almacenado Java debe ampliar la clase com.ibm.db2.app.StoredProc. Los procedimientos almacenados Java son métodos públicos de instancia. Dentro de las clases, los procedimientos almacenados se identifican mediante el nombre y la firma de método. Al llamar a un procedimiento almacenado, su firma se genera automáticamente, en función de los tipos de variable definidos por la sentencia CREATE PROCEDURE.

La clase com.ibm.db2.app.StoredProc proporciona el método isNull, que permite que un método Java determine si un parámetro de entrada es SQL NULL. La clase com.ibm.db2.app.StoredProc también proporciona métodos set...( ) que establecen parámetros de salida. Debe utilizar estos métodos para establecer parámetros de salida. Si no establece un parámetro de salida, el parámetro de salida devuelve el valor SQL NULL.

La clase com.ibm.db2.app.StoredProc proporciona la rutina siguiente para captar una conexión JDBC con el contexto de aplicación de incorporación. Se accede a una conexión con el contexto de aplicación de incorporación utilizando la siguiente llamada JDBC:

```
public Java.sql.Connection getConnection( )
```

Luego, esta conexión ejecuta sentencias SQL con las API JDBC.

A continuación se ofrece un pequeño procedimiento almacenado con un parámetro de entrada y dos parámetros de salida. Procesa la consulta SQL dada y devuelve el número de filas del resultado y SQLSTATE.

Ejemplo: procedimiento almacenado con una entrada y dos salidas

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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());
    }
  }
}
```

Para devolver un conjunto de resultados en procedimientos que utilizan el estilo de los parámetros de DB2GENERAL, el conjunto de resultados y la sentencia que responde deben dejarse abiertos al final del procedimiento. El conjunto de resultados que se devuelve debe cerrarlo la aplicación cliente. Si se

devuelven varios conjuntos de resultados, se devuelven en el orden en el que se han abierto. Por ejemplo, el siguiente procedimiento almacenado devuelve dos conjuntos de resultados.

Ejemplo: procedimiento almacenado que devuelve dos conjuntos de resultados

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
public void returnTwoResultSets() throws Exception
{
  // obtener conexión del llamador con la base de datos; heredado de 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);
}
```

Restricciones de los procedimientos almacenados Java:

Estas restricciones se aplican a los procedimientos almacenados Java.

- El procedimiento almacenado Java no debe crear hebras adicionales. Solo puede crearse una hebra adicional en un trabajo si el trabajo tiene capacidad multihebra. Puesto que no existe ninguna garantía de que un trabajo que llama a un procedimiento almacenado SQL tenga capacidad multihebra, el procedimiento almacenado Java no debe crear hebras adicionales.
- No puede utilizar una autorización adoptada para acceder a archivos de clase Java.
- El procedimiento almacenado Java emplea la misma versión predeterminada del JDK que el mandato `java`. Si es necesario, la versión del JDK empleado por un procedimiento almacenado Java se puede cambiar mediante un archivo SystemDefault.properties.
- Dado que las clases Blob y Clob residen en los paquetes java.sql y com.ibm.db2.app, el programador debe utilizar el nombre completo de estas clases si ambas clases se utilizan en el mismo programa. El programa debe garantizar que las clases Blob y Clob de com.ibm.db2.app se utilizan como parámetros pasados al procedimiento almacenado.
- Cuando se crea un procedimiento almacenado Java, el sistema genera un programa en la biblioteca. Este programa de servicio se utiliza para almacenar la definición de procedimiento. El programa de servicio tiene un nombre generado por el sistema. Este nombre puede obtenerse examinando las anotaciones del trabajo que ha creado el procedimiento almacenado. Si el objeto de programa se guarda y luego se restaura, se restaura también la definición de procedimiento. Si hay que trasladar un procedimiento almacenado Java de un sistema a otro, deberá encargarse de trasladar el programa que contiene la definición del procedimiento, así como el archivo del sistema de archivos integrado, que contiene la clase Java.
- El procedimiento almacenado Java no puede establecer las propiedades (por ejemplo, la denominación del sistema) de la conexión JDBC empleada para establecer conexión con la base de datos. Siempre se utilizan las propiedades de conexión JDBC por omisión, excepto cuando la preextracción está inhabilitada.

Funciones escalares Java definidas por usuario

La función escalar Java devuelve un valor a la base de datos desde un programa Java. Por ejemplo, se podría crear una función escalar que devolviera la suma de dos números.

Al igual que los procedimientos almacenados Java, las funciones escalares Java utilizan uno de los dos estilos de parámetro, Java y DB2GENERAL. Cuando escriba el código de una función definida por usuario (UDF) Java, debe tener presentes las restricciones que afectan a la la creación de funciones escalares Java.

Estilo de parámetro Java

El estilo de los parámetros Java es el especificado por el estándar *SQLJ Part 1: SQL Routines*. Cuando escriba el código fr una UDF Java, siga estos convenios.

- El método Java debe ser público estático.
- El método Java debe devolver un tipo compatible con SQL. El valor de retorno es el resultado del método.
- Los parámetros del método Java deben ser tipos compatibles con SQL.
- El método Java podría comprobar la presencia de un SQL NULL para los tipos Java que permiten el valor nulo.

Por ejemplo, dada una UDF que se llama sample!test3 y devuelve INTEGER y toma argumentos de tipo CHAR(5), BLOB(10K) y DATE, DB2 espera que la implementación Java de la UDF tenga esta signatura:

```
import com.ibm.db2.app.*;
public class sample {
   public static int test3(String arg1, Blob arg2, Date arg3) { ... }
}
```

Los parámetros de un método Java deben ser tipos compatibles con SQL. Por ejemplo, si se declara que una UDF toma argumentos de los tipos SQL t1, t2 y t3, y que devuelve el tipo t4, se la llama como a un método Java con la signatura Java que cabe esperar:

```
public static T4 nombre (T1 a, T2 b, T3 c)  { .....}
```

donde:

- *nombre* es el nombre del método
- Los tipos de T1 a T4 son los tipos Java que se corresponden con los tipos SQL de t1 a t4.
- *a*, *b* y *c* son nombres de variable arbitrarios para los argumentos de entrada.

La correlación entre tipos SQL y tipos Java se encuentra en: Convenios de pase de parámetros para procedimientos almacenados y funciones definidas por usuario (UDF).

Los valores SQL NULL se representan mediante variables Java que no están inicializadas. Estas variables tiene un valor Java nulo si son tipos de objeto. Si se pasa un SQL NULL a datos Java de tipo escalar (como int), se produce una condición de excepción.

Para devolver un resultado desde una UDF Java al utilizar el estilo de parámetros JAVA, basta con devolver el resultado del método.

```
{ ....
  return value;
}
```

Al igual que los módulos C utilizados en las UDF y en los procedimientos almacenados, no puede utilizar las corrientes de E/S estándar Java (System.in, System.out y System.err) en las UDF Java.

Estilo de los parámetros DB2GENERAL

El estilo de los parámetros DB2GENERAL se emplea en las funciones definidas por usuario (UDF) Java. En este estilo de parámetro, el valor de retorno se pasa como el último parámetro de la función y debe establecerse mediante un método *set* de la clase com.ibm.db2.app.UDF.

Cuando escriba el código de una UDF Java, hay que seguir estos convenios:

- La clase, que incluye la UDF Java, debe ampliar (*extend*) la clase Java com.ibm.db2.app.UDF o ser una subclase de ella.

- En el estilo de los parámetros DB2GENERAL, el método Java debe ser un método de instancia público vacío (void).
- Los parámetros del método Java deben ser tipos compatibles con SQL.
- El método Java podría comprobar la presencia de un valor SQL NULL mediante el método isNull.
- En el caso del estilo de los parámetros DB2GENERAL, el método Java debe establecer explícitamente el parámetro de retorno utilizando el método set().

Una clase que incluya una UDF Java debe ampliar la clase Java com.ibm.db2.app.UDF. Una UDF Java que utilice el estilo de parámetro DB2GENERAL debe ser un método de instancia void de la clase Java. Por ejemplo, en el caso de una UDF que se llame sample!test3 y devuelva INTEGER y tome argumentos de tipo CHAR(5), BLOB(10K) y DATE, DB2 espera que la implementación Java de la UDF tenga esta signatura:

```
import com.ibm.db2.app.*;
public class sample extends UDF {
   public void test3(String arg1, Blob arg2, String arg3, int result) { ... }
}
```

Los parámetros de un método Java deben ser tipos SQL. Por ejemplo, si se declara que una UDF toma argumentos de los tipos SQL t1, t2 y t3, y que devuelve el tipo t4, se la llama como a un método Java con la signatura Java que cabe esperar:

```
public void nombre (T1 a, T2 b, T3 c, T4 d)  { .....}
```

donde:

- *nombre* es el nombre del método
- Los tipos de T1 a T4 son los tipos Java que se corresponden con los tipos SQL de t1 a t4.
- *a*, *b* y *c* son nombres de variable arbitrarios para los argumentos de entrada.
- *d* es un nombre de variable arbitrario que representa el resultado de la UDF que se calcula.

La correlación entre tipos SQL y tipos Java se proporciona en: Convenios de pase de parámetros para procedimientos almacenados y funciones definidas por usuario (UDF).

Los valores SQL NULL se representan mediante variables Java que no están inicializadas. Estas variables tienen el valor cero si son tipos primitivos, y tienen el valor nulo Java si son tipos de objeto, según las reglas de Java. Para indicar un valor SQL NULL que no sea un cero ordinario, puede llamarse al método isNull para un argumento de entrada:

```
{ ....
  if (isNull(1)) { /* el argumento nº 1 era un SQL NULL */ }
  else           { /* no NULL */ }
}
```

En el ejemplo anterior, los números de argumento empiezan por el uno. La función isNull(), al igual que las demás funciones que siguen, se hereda de la clase com.ibm.db2.app.UDF. Para devolver un resultado desde una UDF Java cuando se utiliza el estilo de parámetro DB2GENERAL, emplee el método set() de la UDF, como se indica a continuación:

```
{ ....
  set(2, valor);
}
```

Donde 2 es el índice de un argumento de salida y *valor* es un literal o variable de un tipo compatible. El número de argumento es el índice de la lista de argumentos de la salida seleccionada. En el primer ejemplo de esta sección, la variable de resultado int tiene un índice 4. Un argumento de salida que no se establezca antes de que la UDF retorne tiene el valor NULL.

Al igual que los módulos C utilizados en las UDF y en los procedimientos almacenados, no puede utilizar las corrientes de E/S estándar Java (System.in, System.out y System.err) en las UDF Java.

Por lo general, DB2 llama a una UDF muchas veces, una para cada fila de una entrada o conjunto de resultados de una consulta. Si se especifica SCRATCHPAD en la sentencia CREATE FUNCTION de la UDF, DB2 reconoce que es necesaria alguna "continuidad" entre las sucesivas invocaciones de la UDF y, por tanto, en el caso de las funciones de estilo de parámetro DB2GENERAL, no se crea una instancia de la clase Java de implementación para cada llamada, sino que generalmente se crea una vez por cada referencia a UDF y por cada sentencia. Sin embargo, si se especifica NO SCRATCHPAD para una UDF, se crea una instancia pura por cada llamada a la UDF por medio de una llamada al constructor de la clase.

Puede ser de utilidad tener un área reutilizable para guardar información a lo largo de las llamadas a una UDF. Las UDF Java pueden utilizar variables de instancia o bien establecer que el área reutilizable consiga continuidad entre las llamadas. Las UDF Java acceden al área reutilizable con los métodos getScratchPad y setScratchPad disponibles en com.ibm.db2.app.UDF. Al final de una consulta, si especifica la opción FINAL CALL en la sentencia CREATE FUNCTION, se llama al método public void close() del objeto (para las funciones del estilo de parámetro DB2GENERAL). Si no define este método, una función de apéndice toma el control y el evento se pasa por alto. La clase com.ibm.db2.app.UDF contiene variables y métodos útiles que pueden utilizarse con una UDF del estilo de parámetro DB2GENERAL. En la tabla siguiente se describen estas variables y métodos.

Variables y métodos	**Descripción**
• 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;	En las UDF escalares, se trata de constantes para determinar si la llamada es una primera llamada o una llamada normal. En las UDF de tabla, se trata de constantes para determinar si la llamada es una primera llamada, una llamada abierta, una llamada de extracción, una llamada de cierre o una llamada final.
public Connection getConnection();	El método contiene el handle de conexión JDBC para esta llamada de procedimiento almacenado y devuelve un objeto JDBC que representa la conexión de la aplicación que efectúa la llamada con la base de datos. Es análogo al resultado de una llamada a SQLConnect() nula de un procedimiento almacenado C.
public void close();	La base de datos llama a este método al final de una evaluación de UDF, si la UDF se ha creado con la opción FINAL CALL. Es análogo a la llamada final de una UDF C. Si una clase Java UDF no implementa este método, el evento se ignora.
public boolean isNull(int i)	Este método comprueba si un argumento de entrada con el índice dado es SQL NULL.
• 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);	Estos métodos establecen un argumento de salida en el valor dado. Se lanza una excepción si se produce alguna anomalía, incluyendo las siguientes: • La llamada de UDF no progresa • El índice no hace referencia a un argumento de salida válido • El tipo de datos no coincide • La longitud de los datos no coincide • Se produce un error de conversión de página de códigos

Variables y métodos	Descripción
public void setSQLstate(String string);	Puede llamarse a este método desde una UDF para establecer el SQLSTATE que debe devolverse desde esta llamada. Si la serie no es aceptable como SQLSTATE, se lanza una excepción. El usuario puede establecer SQLSTATE en el programa externo para que devuelva un error o un aviso desde la función. En este caso, SQLSTATE debe contener uno de los siguientes elementos: • '00000' para indicar el éxito • '01Hxx', donde xx son dos dígitos o letras mayúsculas cualesquiera, para indicar un aviso • '38yxx', donde y es una letra mayúscula entre 'I' y 'Z' y xx son dos dígitos o letras mayúsculas cualesquiera, para indicar un error
public void setSQLmessage(String string);	Este método es parecido al método setSQLstate. Establece el resultado del mensaje SQL. Si la serie no es aceptable (por ejemplo, es superior a los 70 caracteres), se lanza una excepción.
public String getFunctionName();	Este método devuelve el nombre de la UDF de proceso.
public String getSpecificName();	Este método devuelve el nombre específico de la UDF de proceso.
public byte[] getDBinfo();	Este método devuelve una estructura DBINFO sin procesar para la UDF de proceso, como matriz de bytes. La UDF debe haberse registrado (mediante CREATE FUNCTION) con la opción DBINFO.
• public String getDBname(); • public String getDBauthid(); • public String getDBver_rel(); • public String getDBplatform(); • public String getDBapplid(); • public String getDBapplid(); • public String getDBtbschema(); • public String getDBtbname(); • public String getDBcolname();	Estos métodos devuelven el valor del campo adecuado de la estructura DBINFO de la UDF de proceso. La UDF debe haberse registrado (mediante CREATE FUNCTION) con la opción DBINFO. Los métodos getDBtbschema(), getDBtbname() y getDBcolname() solo devuelven información que tenga sentido si se ha especificado una función definida por usuario a la derecha de una cláusula SET en una sentencia UPDATE.
public int getCCSID();	Este método devuelve el CCSID del trabajo.
public byte[] getScratchpad();	Este método devuelve una copia del área reutilizable de la UDF de proceso actual. Primero debe declarar la UDF con la opción SCRATCHPAD.
public void setScratchpad(byte ab[]);	Este método sobrescribe el área reutilizable de la UDF de proceso actual con el contenido de la matriz de bytes dada. Primero debe declarar la UDF con la opción SCRATCHPAD. La matriz de bytes debe tener el mismo tamaño que el devuelto por getScratchpad().

Variables y métodos	Descripción
public int getCallType();	Este método devuelve el tipo de llamada que se está efectuando actualmente. Estos valores corresponden a los valores C definidos en sqludf.h. Los valores de retorno posibles son los siguientes: • SQLUDF_FIRST_CALL • SQLUDF_NORMAL_CALL • SQLUDF_TF_FIRST • SQLUDF_TF_OPEN • SQLUDF_TF_FETCH • SQLUDF_TF_CLOSE • SQLUDF_TF_FINAL

Restricciones impuestas a las funciones definidas por usuario Java:

Aquí se indican las restricciones impuestas a las funciones definidas por usuario (UDF) Java.

- Las UDF Java no deben crear hebras adicionales. Solo se puede crear una hebra adicional en un trabajo si este tiene capacidad multihebra. Puesto que no existe ninguna garantía de que un trabajo que llame a un procedimiento almacenado SQL tenga capacidad multihebra, el procedimiento almacenado Java no debe crear hebras adicionales.
- El nombre completo del procedimiento almacenado Java definido en la base de datos está limitado a 279 caracteres. Este límite es consecuencia de la columna EXTERNAL_NAME, que tiene una anchura máxima de 279 caracteres.
- No se puede utilizar la autorización adoptada para acceder a archivos de clase Java.
- Las UDF Java siempre utilizan la versión más reciente del JDK instalada en el sistema.
- Dado que las clases Blob y Clob residen en los paquetes java.sql y com.ibm.db2.app, el programador debe utilizar el nombre completo de estas clases si ambas clases se utilizan en el mismo programa. El programa debe garantizar que las clases Blob y Clob de com.ibm.db2.app se utilizan como parámetros pasados al procedimiento almacenado.
- Al igual que las funciones con origen, cuando se crea una UDF Java, se utiliza un programa de servicio de la biblioteca para almacenar la definición de la función. El sistema genera el nombre del programa de servicio, que puede encontrarse en las anotaciones del trabajo que ha creado la función. Si este objeto se guarda y luego se restaura en otro sistema, se restaura también la definición de la función. Si hay que trasladar una UDF Java de un sistema a otro, usted se debe encargar de trasladar el programa de servicio que contiene la definición de la función, así como el archivo del sistema de archivos integrado que contiene la clase Java.
- Las UDF Java no pueden establecer las propiedades (por ejemplo, la denominación del sistema) de la conexión JDBC utilizada para conectarse a la base de datos. Siempre se utilizan las propiedades de conexión JDBC por omisión, excepto cuando la precaptación está inhabilitada.

Funciones de tabla definidas por usuario Java:

DB2 proporciona capacidad para que una función devuelva una tabla. Esto resulta de utilidad para exponer información externa a la base de datos en formato de tabla. Por ejemplo, se puede crear una tabla que exponga las propiedades establecidas en la máquina virtual Java (JVM) empleada para procedimientos almacenados Java y funciones definidas por usuario Java (tanto de tabla como escalares).

El estándar *SQLJ Part 1: SQL Routines* no da soporte a las funciones de tabla. En consecuencia, las funciones de tabla solo están disponibles mediante el estilo de parámetro DB2GENERAL.

A una función de tabla se efectúan cinco tipos diferentes de llamadas. La tabla siguiente describe estas llamadas. Se presupone que se ha especificado scratchpad en la sentencia SQL de creación de función.

Punto en tiempo de exploración	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
Antes del primer OPEN de la función de tabla	Sin llamadas	Se llama al constructor de la clase (indica nuevo scratchpad). Se llama al método de la UDF con llamada FIRST.
En cada OPEN de la función de tabla.	Se llama al constructor de la clase (indica nuevo scratchpad). Se llama al método de la UDF con llamada OPEN.	Se llama al método de la UDF con llamada OPEN.
En cada FETCH de una fila nueva de datos de la función de tabla.	Se llama al método de la UDF con llamada FETCH.	Se llama al método de la UDF con llamada FETCH.
En cada CLOSE de la función de tabla.	Se llama al método de la UDF con llamada CLOSE. También se llama al método close(), si existe.	Se llama al método de la UDF con llamada CLOSE.
Después de la última CLOSE de la función de tabla.	Sin llamadas	Se llama al método de la UDF con llamada FINAL. También se llama al método close(), si existe.

Ejemplo: función de tabla Java

A continuación figura un ejemplo de una función de tabla Java que determina las propiedades establecidas en la JVM empleada para ejecutar la función de tabla Java definida por usuario.

Nota: Lea la Declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
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 vez compilada la función de tabla y después de haberse copiado el archivo de clase en /QIBM/UserData/OS400/SQLLib/Function, la función se puede registrar en la base de datos mediante la siguiente sentencia 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
```

Después de registrar la función, se la puede utilizar como parte de una sentencia SQL. Por ejemplo, la siguiente sentencia SELECT devuelve la tabla generada por la función de tabla.

```
SELECT * FROM TABLE(PROPERTIES())
```

Procedimientos SQLJ que manipulan archivos JAR

Tanto los procedimientos almacenados Java como las funciones definidas por usuario (UDF) Java pueden utilizar clases Java almacenadas en archivos JAR Java.

Para utilizar un archivo JAR, debe haber un *id-jar* asociado al archivo JAR. El sistema proporciona procedimientos almacenados en el esquema SQLJ que permiten manipular *id-jar* y archivos JAR. Estos procedimientos permiten instalar, sustituir y eliminar archivos JAR. También proporcionan la posibilidad de utilizar y actualizar los catálogos SQL asociados con archivos JAR.

SQLJ.INSTALL_JAR:

El procedimiento almacenado SQLJ.INSTALL_JAR instala un archivo JAR en el sistema de bases de datos. Este archivo JAR puede utilizarse en sentencias CREATE FUNCTION y CREATE PROCEDURE subsiguientes.

Autorización

El privilegio que contiene el ID de autorización de la sentencia CALL debe incluir como mínimo uno de los siguientes valores para las tablas de catálogo SYSJAROBJECTS y SYSJARCONTENTS:

- Las siguientes autorizaciones de sistema:
  - Los privilegios INSERT y SELECT en la tabla
  - La autorización de sistema *EXECUTE sobre la biblioteca QSYS2
- Autorización administrativa

El privilegio que contiene el ID de autorización de la sentencia CALL también debe incluir las siguientes autorizaciones:

- Acceso de lectura (*R) sobre el archivo JAR especificado en el parámetro *url-jar* que se instala.
- Acceso de escritura, ejecución y lectura (*RWX) sobre el directorio donde está instalado el archivo JAR. Este directorio es /QIBM/UserData/OS400/SQLLib/Function/jar/*esquema*, donde *esquema* es el esquema de *id-jar*.

No puede utilizarse autorización adoptada para estas autorizaciones.

Sintaxis SQL

```
>>-CALL--SQLJ.INSTALL_JAR-- (--'url-jar'--,--'id-jar'--,--despliegue--)-->

>-------------------------------------------------------------><
```

Descripción

url-jar El URL que contiene el archivo JAR que debe instalarse o sustituirse. El único esquema de URL soportado es 'file:'.

id-jar El identificador de JAR de la base de datos que debe asociarse con el archivo especificado por el *url-jar*. El *id-jar* utiliza la denominación SQL y el archivo JAR se instala en el esquema o biblioteca especificados por el calificador implícito o explícito.

despliegue
Valor utilizado para describir la acción de instalación (install_action) del archivo del descriptor de despliegue. Si este entero es un valor distinto de cero, las acciones de instalación (install_actions) de un archivo de descriptor de despliegue deben realizarse al final del procedimiento (install_jar). La versión actual de DB2 para i5/OS solo permite el valor cero.

Notas de utilización

Cuando se instala un archivo JAR, installed, DB2 para i5/OS registra el archivo JAR en el catálogo SYSJAROBJECTS del sistema. También extrae los nombres de los archivos de clase Java del archivo JAR y registra cada clase en el catálogo SYSJARCONTENTS del sistema. DB2 para i5/OS copia el archivo JAR en un subdirectorio jar/schema del directorio /QIBM/UserData/OS400/SQLLib/Function. DB2 para i5/OS da a la nueva copia del archivo JAR el nombre que figura en la cláusula *id-jar*. Los archivos JAR que DB2 para i5/OS haya instalado en un subdirectorio de /QIBM/UserData/OS400/SQLLib/Function/jar no se deben cambiar. En lugar de ello, deben utilizarse los mandatos SQL CALL SQLJ.REMOVE_JAR y CALL SQLJ.REPLACE_JAR para eliminar o sustituir un archivo JAR instalado.

Ejemplo

El mandato siguiente se emite desde una sesión interactiva SQL.

```
CALL SQLJ.INSTALL_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar', 0)
```

El archivo Proc.jar situado en el directorio file:/home/db2inst/classes/ se instala en DB2 para i5/OS con el nombre myproc_jar. Los mandatos SQL subsiguientes que utilizan el archivo Procedure.jar hacen referencia a él con el nombre myproc_jar.

SQLJ.REMOVE_JAR:

El procedimiento almacenado SQLJ.REMOVE_JAR elimina un archivo JAR del sistema de bases de datos.

Autorización

El privilegio que contiene el ID de autorización de la sentencia CALL debe incluir como mínimo uno de los siguientes valores para las tablas de catálogo SYSJARCONTENTS y SYSJAROBJECTS:

- Las siguientes autorizaciones de sistema:
  - Los privilegios SELECT y DELETE en la tabla
  - La autorización de sistema *EXECUTE sobre la biblioteca QSYS2
- Autorización administrativa

El privilegio que contiene el ID de autorización de la sentencia CALL también debe incluir la siguiente autorización:

- La autorización *OBJMGT sobre el archivo JAR que se elimina. El archivo JAR se denomina /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile.

No puede utilizarse autorización adoptada para esta autorización.

Sintaxis

```
>>-CALL--SQLJ.REMOVE_JAR--(--'id-jar'--,--deshacer despliegue--)----------><
```

Descripción

id-jar Identificador JAR del archivo JAR que debe eliminarse de la base de datos.

deshacer despliegue
Valor utilizado para describir la acción de eliminación (remove_action) del archivo del descriptor de despliegue. Si este entero es un valor distinto de cero, las acciones de eliminación (remove_actions) de un archivo de descriptor de despliegue deben realizarse al final del procedimiento install_jar. La versión actual de DB2 para i5/OS solo permite el valor cero.

Ejemplo

El mandato siguiente se emite desde una sesión interactiva SQL:

```
CALL SQLJ.REMOVE_JAR('myProc_jar', 0)
```

El archivo JAR myProc_jar se elimina de la base de datos.

SQLJ.REPLACE_JAR:

El procedimiento almacenado SQLJ.REPLACE_JAR sustituye un archivo JAR en el sistema de bases de datos.

Autorización

El privilegio que contiene el ID de autorización de la sentencia CALL debe incluir como mínimo uno de los siguientes valores para las tablas de catálogo SYSJAROBJECTS y SYSJARCONTENTS:

- Las siguientes autorizaciones de sistema:
  - Los privilegios SELECT, INSERT y DELETE en la tabla
  - La autorización de sistema *EXECUTE sobre la biblioteca QSYS2
- Autorización administrativa

El privilegio que contiene el ID de autorización de la sentencia CALL también debe incluir las siguientes autorizaciones:

- Acceso de lectura (*R) sobre el archivo JAR especificado en el parámetro *url-jar* que se instala.
- La autorización *OBJMGT sobre el archivo JAR que se elimina. El archivo JAR se denomina /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile.

No puede utilizarse autorización adoptada para estas autorizaciones.

Sintaxis

```
>>-CALL--SQLJ.REPLACE_JAR--(--'url-jar'--,--'id-jar'--)--------><
```

Descripción

url-jar El URL que contiene el archivo JAR que debe sustituirse. El único esquema de URL soportado es ′file:′.

id-jar El identificador de JAR de la base de datos que debe asociarse con el archivo especificado por el *url-jar*. El *id-jar* utiliza la denominación SQL y el archivo JAR se instala en el esquema o biblioteca especificados por el calificador implícito o explícito.

Notas de utilización

El procedimiento almacenado SQLJ.REPLACE_JAR sustituye un archivo JAR que se ha instalado anteriormente en la base de datos mediante SQLJ.INSTALL_JAR.

Ejemplo

El mandato siguiente se emite desde una sesión interactiva SQL:

```
CALL SQLJ.REPLACE_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar')
```

El archivo JAR actual al que hace referencia el *id-jar* myproc_jar se sustituye por el archivo Proc.jar ubicado en el directorio file:/home/db2inst/classes/.

SQLJ.UPDATEJARINFO:

SQLJ.UPDATEJARINFO actualiza la columna CLASS_SOURCE de la tabla de catálogo SYSJARCONTENTS. Este procedimiento no forma parte del estándar SQLJ, pero se utiliza en el constructor de procedimientos almacenados de the DB2 para i5/OS.

Autorización

El privilegio que contiene el ID de autorización de la sentencia CALL debe incluir como mínimo uno de los siguientes valores para la tabla de catálogo SYSJARCONTENTS:

- Las siguientes autorizaciones de sistema:
  - Los privilegios SELECT y UPDATEINSERT en la tabla
  - La autorización de sistema *EXECUTE sobre la biblioteca QSYS2
- Autorización administrativa

El usuario que ejecuta la sentencia CALL también debe tener las siguientes autorizaciones:

- Acceso de lectura (*R) sobre el archivo JAR especificado en el parámetro *url-jar*. Acceso de lectura (*R) sobre el archivo JAR que se instala.
- Acceso de escritura, ejecución y lectura (*RWX) sobre el directorio donde está instalado el archivo JAR. Este directorio es /QIBM/UserData/OS400/SQLLib/Function/jar/*esquema*, donde *esquema* es el esquema de *id-jar*.

No puede utilizarse autorización adoptada para estas autorizaciones.

Sintaxis

```
>>-CALL--SQLJ.UPDATEJARINFO--(--'id-jar'--,--'id-clase'--,--'url-jar'--)-->

>-------------------------------------------------------------><
```

Descripción

id-jar El identificador JAR de la base de datos que debe actualizarse.

id-clase
El nombre de clase calificado por paquete de la clase que debe actualizarse.

url-jar El URL que contiene el archivo de clase con el que debe actualizarse el archivo JAR. El único esquema de URL soportado es 'file:'.

Ejemplo

El mandato siguiente se emite desde una sesión interactiva SQL:

```
CALL SQLJ.UPDATEJARINFO('myproc_jar', 'mypackage.myclass',
                        'file:/home/user/mypackage/myclass.class')
```

El archivo JAR asociado con el *id-jar* myproc_jar se actualiza con una versión nueva de la clase mypackage.myclass. La versión nueva de la clase se obtiene del archivo /home/user/mypackage/myclass.class.

SQLJ.RECOVERJAR:

El procedimiento SQLJ.RECOVERJAR toma el archivo JAR almacenado en el catálogo SYSJAROBJECTS y lo restaura en el archivo /QIBM/UserData/OS400/SQLLib/Function/jar/*jarschema*/*jar_id*.jar.

Autorización

El privilegio que contiene el ID de autorización de la sentencia CALL debe incluir como mínimo uno de los siguientes valores para la tabla de catálogo SYSJAROBJECTS:

- Las siguientes autorizaciones de sistema:
  - Los privilegios SELECT y UPDATEINSERT en la tabla
  - La autorización de sistema *EXECUTE sobre la biblioteca QSYS2
- Autorización administrativa

El usuario que ejecuta la sentencia CALL también debe tener las siguientes autorizaciones:

- Acceso de escritura, ejecución y lectura (*RWX) sobre el directorio donde está instalado el archivo JAR. Este directorio es /QIBM/UserData/OS400/SQLLib/Function/jar/*esquema*, donde *esquema* es el esquema de *id-jar*.
- La autorización *OBJMGT sobre el archivo JAR que se elimina. El archivo JAR se denomina /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile.

Sintaxis

```
>>-CALL--SQLJ.RECOVERJAR--(--'id-jar'--)----------------------><
```

Descripción

id-jar El identificador JAR de la base de datos que debe convertirse.

Ejemplo

El mandato siguiente se emite desde una sesión interactiva SQL:

```
CALL SQLJ.UPDATEJARINFO('myproc_jar')
```

El archivo JAR asociado con myproc_jar se actualiza con el contenido de la tabla SYSJARCONTENT. El archivo se copia en /QIBM/UserData/OS400/SQLLib/Function/jar/jar_schema myproc_jar.jar.

SQLJ.REFRESH_CLASSES:

El procedimiento almacenado SQLJ.REFRESH_CLASSES hace que se vuelvan a cargar las clases definidas por usuario utilizadas por los procedimientos almacenados Java o por las funciones definidas por usuario (UDF) Java en la conexión de base de datos actual. A este procedimiento almacenado deben llamarlo las conexiones de base de datos existentes para obtener cambios realizados por una llamada al procedimiento almacenado SQLJ.REPLACE_JAR.

Autorización

NONE

Sintaxis

```
>>-CALL--SQLJ.REFRESH_CLASSES-- ()-->
>-------------------------------------------------------------><
```

Ejemplo

Llame al procedimiento almacenado Java MYPROCEDURE que utiliza una clase de un archivo jar registrado con el id de jar MYJAR:

```
CALL MYPROCEDURE()
```

Sustituya el archivo jar utilizando la siguiente llamada:

```
CALL SQLJ.REPLACE_JAR('MYJAR', '/tmp/newjarfile.jar')
```

Para hacer que las siguientes llamadas al procedimiento almacenado MYPROCEDURE utilicen el archivo jar actualizado, hay que llamar a SQLJ.REFRESH_CLASSES:

```
CALL SQLJ.REFRESH_CLASSES()
```

Vuelva a llamar al procedimiento almacenado. Se utilizan los archivos de clase actualizados al llamar al procedimiento.

```
CALL MYPROCEDURE()
```

Convenios de pase de parámetros para procedimientos almacenados y funciones definidas por usuario (UDF) Java

En la tabla que sigue figura una lista de cómo se representan los tipos de datos SQL en los procedimientos almacenados y las UDF Java.

Tipo de datos SQL	Parámetro Java de estilo JAVA	Parámetro Java de estilo 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, FLOAT o FLOAT(p)	double	double
CHARACTER(n)	Serie	Serie
CHARACTER(n) FOR BIT DATA	byte[]	com.ibm.db2.app.Blob
VARCHAR(n)	Serie	Serie
VARCHAR(n) FOR BIT DATA	byte[]	com.ibm.db2.app.Blob
GRAPHIC(n)	Serie	Serie
VARGRAPHIC(n)	Serie	Serie
DATE	Fecha	Serie
TIME	Hora	Serie
TIMESTAMP	Indicación de la hora	Serie
Variable de indicador	-	-
CLOB	-	com.ibm.db2.app.Clob

Tipo de datos SQL	Parámetro Java de estilo JAVA	Parámetro Java de estilo DB2GENERAL
BLOB	-	com.ibm.db2.app.Blob
DBCLOB	-	com.ibm.db2.app.Clob
DataLink	-	-

Java con otros lenguajes de programación

Con Java, existen varias maneras de llamar al código escrito en un lenguaje distinto de Java.

Interfaz Java nativa (JNI)

Una de las formas en que puede llamar a código escrito en otro lenguaje consiste en implementar como 'métodos nativos' los métodos Java seleccionados. Los métodos nativos son procedimientos, escritos en otro lenguaje, que proporcionan la implementación real de un método Java. Los métodos nativos pueden acceder a la máquina virtual Java mediante la interfaz Java nativa (JNI). Estos métodos nativos se ejecutan bajo la hebra Java, que es una hebra del kernel, por lo que han de ser seguros en ejecución multihebra. Una función es segura en ejecución multihebra si puede iniciarse simultáneamente en varias hebras dentro de un mismo proceso. Asimismo, lo es si, y solo si, lo son también todas las funciones a las que llama.

Los métodos nativos constituyen el "puente" que permite acceder a las funciones del sistema no soportadas directamente en Java o que permite intercambiar información con el código de usuario existente. A la hora de utilizar métodos nativos, conviene ser precavido porque el código al que se llama puede no ser seguro en ejecución multihebra.

API de invocación Java

El hecho de utilizar la API de invocación Java, que también forma parte de la especificación Interfaz Java nativa (JNI), permite que una aplicación no Java pueda emplear la máquina virtual Java. Permite asimismo emplear código Java como extensión de la aplicación.

Métodos nativos de i5/OS PASE

La máquina virtual i5/OS Java permite que se utilicen métodos nativos en ejecución en el entorno i5/OS PASE. Los métodos nativos de i5/OS PASE para Java le permiten transportar fácilmente las aplicaciones Java que se ejecutan en AIX al servidor. Puede copiar los archivos de clase y las bibliotecas de métodos nativos de AIX en el sistema de archivos integrado del servidor y ejecutarlos desde cualquier indicador de mandatos de lenguaje de control (CL), de Qshell o de sesión de terminal i5/OS PASE.

Métodos nativos de teraespacio

La máquina virtual Java (JVM) de i5/OS permite utilizar métodos nativos del modelo de almacenamiento de teraespacio. El modelo de almacenamiento de teraespacio proporciona un entorno de procesos extensos con dirección local para programas ILE. Utilizar el teraespacio le permite llevar código de método nativo desde otros sistemas operativos a i5/OS con pocos o ningún cambio en el código fuente.

java.lang.Runtime.exec()

El método java.lang.Runtime.exec() le permite llamar a programas o mandatos desde dentro de un programa Java. El método exec() inicia otro proceso en el que se puede ejecutar cualquier mandato o programa de System i5. En este modelo, para la comunicación entre procesos se puede utilizar la corriente de entrada, de salida y de error estándar del proceso hijo.

Comunicación entre procesos

Una opción es utilizar sockets para la comunicación entre el proceso padre y el proceso hijo.

También se pueden utilizar archivos continuos para la comunicación entre programas. Si desea obtener una visión general de las opciones existentes para comunicar con programas que se ejecuten en otro proceso, vea el tema de comunicaciones entre procesos.

Para llamar a Java desde otros lenguajes, vea los temas Ejemplo: llamar a Java desde C o Ejemplo: llamar a Java desde RPG.

También puede utilizar IBM Toolbox para Java para llamar a programas y mandatos existentes en el servidor. Las colas de datos y los mensajes de System i5 se suelen utilizar para la comunicación entre procesos (IPC) con IBM Toolbox para Java.

Nota: La utilización de Runtime.exec(), IBM Toolbox para Java, o JNI puede comprometer la portabilidad del programa Java. Debe evitar el uso de estos métodos en un entorno Java "puro".

Conceptos relacionados

"API de invocación Java" en la página 213
La API de invocación, que forma parte de la interfaz Java nativa (JNI), permite al código no crear una máquina virtual Java, así como cargar y utilizar clases Java Esta función permite a un programa multihebra utilizar las clases Java que se ejecutan en múltiples hebras de una sola máquina virtual Java.

"Utilizar sockets para la comunicación entre procesos" en la página 234
Las corrientes por sockets comunican entre sí programas que se están ejecutando en procesos aparte.

"Utilizar corrientes de entrada y de salida para la comunicación entre procesos" en la página 238
Las corrientes de entrada y de salida comunican entre sí programas que se ejecutan en procesos aparte.

Referencia relacionada

"Ejemplo: llamar a Java desde C" en la página 239
Este es un ejemplo de programa C que utiliza la función system() para llamar al programa Java Hello.

"Ejemplo: llamar a Java desde RPG" en la página 239
Este es un ejemplo de un programa RPG que utiliza la API QCMDEXC para llamar al programa Java Hello.

Información relacionada

IBM Toolbox para Java

Utilizar la interfaz Java nativa (JNI) para métodos nativos

Los métodos nativos solo se deben usar en los casos en que Java puro no pueda responder a sus necesidades de programación.

Debe limitar el uso de métodos nativos y emplearlos solo en las circunstancias siguientes:

- Para acceder a funciones del sistema que no están disponibles por medio de Java puro.
- Para implementar métodos extremadamente sensibles al rendimiento que pueden beneficiarse en gran medida de una implementación nativa.
- Para intercambiar información con las interfaces de programación de aplicaciones (API) existentes que permitan a Java llamar a otras API.

Las siguientes instrucciones corresponden al uso de la interfaz Java nativa (JNI) con el lenguaje C. Para obtener información sobre cómo utilizar JNI con el lenguaje RPG, vea el capítulo 11 del manual WebSphere Development Studio: ILE RPG Programmer's Guide, SC09-2507.

Si desea utilizar la interfaz Java nativa (JNI) para métodos nativos, siga estos pasos:

1. Diseñe la clase especificando qué métodos son nativos con la sintaxis del lenguaje Java estándar.
2. Escoja un nombre de biblioteca y programa para el programa de servicio (*SRVPGM) que contiene las implementaciones de método nativo. Cuando codifique la llamada a método System.loadLibrary() en el inicializador estático de la clase, especifique el nombre del programa de servicio.
3. Utilice la herramienta `javac` para compilar el fuente Java y obtener un archivo de clase.
4. Utilice la herramienta `javah` para crear el archivo de cabecera (.h). Este contiene los prototipos exactos para crear las implementaciones de método nativo. La opción -d especifica el directorio en el que debe crearse el archivo de cabecera.
5. Copie el archivo de cabecera del sistema de archivos integrado en un miembro de un archivo fuente; para ello, utilice el mandato Copiar desde archivo continuo (CPYFRMSTMF). Debe copiar el archivo de cabecera en un miembro de archivo fuente para que el compilador C pueda utilizarlo. Emplee el nuevo soporte de archivos continuos del mandato Crear programa ILE C/400 enlazado (CRTCMOD) para dejar los archivos fuente y de cabecera C en el sistema de archivos integrado. Para obtener más información acerca del mandato CRTCMOD y la utilización de archivos continuos, vea el manual WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712.
6. Escriba el código de método nativo. Vea el tema de consideraciones en torno a los métodos nativos y hebras Java para obtener detalles sobre los lenguajes y funciones empleados para los métodos nativos.
   a. Incluya el archivo de cabecera creado en los pasos anteriores.
   b. Correlacione de manera exacta los prototipos del archivo de cabecera.
   c. Convierta las series a ASCII (American Standard Code for Information Interchange) si hay que pasarlas a la máquina virtual Java. Hallará más información en el tema sobre codificación de caracteres Java.
7. Si el método nativo debe interaccionar con la máquina virtual Java, utilice las funciones que se proporcionan con JNI.
8. Compile el código fuente C, utilizando el mandato CRTCMOD, en un objeto módulo (*MODULE).
9. Enlace uno o varios objetos módulo para crear un programa de servicio (*SRVPGM); para ello, utilice el mandato Crear programa de servicio (CRTSRVPGM). El nombre de este programa de servicio debe coincidir con el nombre que ha proporcionado en el código Java que se halla en las llamadas a función System.load() o System.loadLibrary().
10. Si ha utilizado la llamada a System.loadLibrary() en el código Java, realice una de las siguientes tareas, la que sea pertinente para el J2SE que esté ejecutando:
    - Incluya la lista de bibliotecas necesarias en la variable de entorno LIBPATH. Puede cambiar la variable de entorno LIBPATH en QShell y desde la línea de mandatos de i5/OS.
      – En el indicador de mandatos de Qshell, escriba:

      ```
      exportar LIBPATH=/QSYS.LIB/MYLIB.LIB
      java -Djava.version=1.5 myclass
      ```

      – O bien, en la línea de mandatos, escriba:

      ```
      ADDENVVAR LIBPATH '/QSYS.LIB/MYLIB.LIB'
      JAVA PROP((java.version 1.5)) myclass
      ```

    - También puede suministrar la lista en la propiedad **java.library.path**. Puede cambiar la propiedad java.library.path en QShell y desde la línea de mandatos de i5/OS.
      – En el indicador de mandatos de Qshell, entre:

      ```
      java -Djava.library.path=/QSYS.LIB/MYLIB.LIB -Djava.version=1.5 myclass
      ```

      – O bien, en la línea de mandatos de i5/OS, teclee:

      ```
      JAVA PROP((java.library.path '/QSYS.LIB/MYLIB.LIB') (java.version '1.5')) myclass
      ```

Aquí, */QSYS.LIB/MYLIB.LIB* es la biblioteca que desea cargar utilizando la llamada a loadLibrary(), y *myclass* es el nombre de la aplicación Java.

11. La sintaxis de la vía de acceso para System.load(String path) puede ser cualquiera de las siguientes:
    - /qsys.lib/sysNMsp.srvpgm (para *SRVPGM QSYS/SYSNMSP)
    - /qsys.lib/mylib.lib/myNMsp.srvpgm (para *SRVPGM MYLIB/MYNMSP)
    - un enlace simbólico, por ejemplo /home/mydir/myNMsp.srvpgm que enlace con /qsys.lib/mylib.lib/myNMsp.srvpgm

    Nota: Esto equivale a utilizar el método System.loadLibrary(″myNMsp″).

    Nota: El nombre de vía de acceso suele ser un literal de serie entre comillas. Por ejemplo, podría utilizar el siguiente código:

    ```
    System.load("/qsys.lib/mylib.lib/myNMsp.srvpgm")
    ```

12. El parámetro libname para System.loadLibrary(String libname) suele ser un literal de serie entre comillas que identifica la biblioteca de métodos nativos. El sistema utiliza la lista de bibliotecas actual y las variables de entorno LIBPATH y PASE_LIBPATH para buscar un programa de servicio o un ejecutable i5/OS PASE PASE que coincida con el nombre de biblioteca. Por ejemplo, loadLibrary(″myNMsp″) da como resultado la búsqueda de un *SRVPGM denominado MYNMSP o un ejecutable i5/OS PASE denominado libmyNMsp.a o libmyMNsp.so.

En Ejemplos: utilizar la interfaz Java nativa (JNI) para métodos nativos, encontrará un ejemplo de cómo utilizar JNI para los métodos nativos.

Websphere Development Studio: ILE RPG Programmer's Guide, SC09-2507.

“Consideraciones sobre los métodos nativos Java y las hebras” en la página 218
Puede utilizar métodos nativos para acceder a funciones que no están disponibles en Java. Para utilizar mejor Java junto con los métodos nativos, conviene aclarar unos cuantos conceptos.

Interfaz Java nativa (JNI) de Sun Microsystems, Inc.

“Ejemplos: utilizar la interfaz Java nativa (JNI) para métodos nativos” en la página 561
Este programa es un ejemplo sencillo de la interfaz Java nativa (JNI) en el que se utiliza un método nativo C para visualizar ″Hello, World.″ Para generar el archivo NativeHello.h, se utiliza la herramienta `javah` con el archivo de clase NativeHello. En este ejemplo, se supone que la implementación C de NativeHello forma parte de un programa de servicio llamado NATHELLO.

“Las series en los métodos nativos” en la página 220
Muchas funciones de la interfaz Java nativa (JNI) aceptan series de estilo de lenguaje C como parámetros. Por ejemplo, la función FindClass() de JNI acepta un parámetro de tipo serie que especifica el nombre totalmente calificado de un archivo de clase. Si se encuentra el archivo de clase, la función FindClass lo cargará, y al llamador de FindClass se le devolverá una referencia al archivo de clase.

“Codificaciones de caracteres de Java” en la página 26
Los programas Java pueden convertir datos en distintos formatos, permitiendo a las aplicaciones transferir y utilizar información de muchas clases de juegos de caracteres internacionales.

API de invocación Java

La API de invocación, que forma parte de la interfaz Java nativa (JNI), permite al código no crear una máquina virtual Java, así como cargar y utilizar clases Java Esta función permite a un programa multihebra utilizar las clases Java que se ejecutan en múltiples hebras de una sola máquina virtual Java.

IBM Developer Kit para Java permite utilizar la API de de invocación Java para los siguientes tipos de llamadores:

- Un programa o programa de servicio ILE creado para `STGMDL(*SNGLVL)` y `DTAMDL(*P128)`
- Un programa o programa de servicio ILE creado para `STGMDL(*TERASPACE)` y `DTAMDL(*LLP64)`

- Un ejecutable i5/OS PASE creado para AIX de 32 bits o de 64 bits.

La aplicación controla la máquina virtual Java. La aplicación puede crear la máquina virtual Java, llamar a métodos Java (de forma parecida a cómo llama una aplicación a las subrutinas) y destruir la máquina virtual Java. Una vez creada, la máquina virtual Java está preparada para ejecutarse dentro del proceso hasta que la aplicación la destruye de manera explícita. Mientras se destruye, la máquina virtual Java realiza operaciones de borrado como, por ejemplo, ejecutar finalizadores, finalizar las hebras de la máquina virtual Java y liberar los recursos de la máquina virtual Java.

Con una máquina virtual Java preparada para ejecutarse, una aplicación escrita en lenguajes ILE, tales como C y RPG, puede llamar a la máquina virtual Java para que realice cualquier función. También puede retornar de la máquina virtual Java a la aplicación C, llamar de nuevo a la máquina virtual Java, y así sucesivamente. La máquina virtual Java se crea una sola vez y y no hace falta crearla nuevamente antes de llamarla para que ejecute código Java, ya sea poco o mucho.

Cuando se utiliza la API de invocación para ejecutar programas Java, el destino de STDOUT y STDERR se controla por medio de una variable de entorno llamada QIBM_USE_DESCRIPTOR_STDIO. Si está establecida en Y o I (por ejemplo, QIBM_USE_DESCRIPTOR_STDIO=Y), la máquina virtual Java utiliza descriptores de archivo para STDIN (fd 0), STDOUT (fd 1) y STDERR (fd 2). En este caso, el programa debe establecer los descriptores de archivo en valores válidos abriéndolos como los tres primeros archivos o conductos del trabajo. Al primer archivo abierto en el trabajo se le da 0 como fd, al segundo 1 y al tercero 2. Para los trabajos iniciados con la API de engendramiento, estos descriptores se pueden preasignar mediante una correlación de descriptores de archivo (consulte la documentación de la API de engendramiento). Si la variable de entorno QIBM_USE_DESCRIPTOR_STDIO no está establecida o bien lo está en cualquier otro valor, no se utilizan descriptores de archivo para STDIN, STDOUT y STDERR. En lugar de ello, se direcciona STDOUT y STDERR a un archivo en spool propiedad del trabajo actual y la utilización de STDIN da como resultado una excepción de E/S.

Funciones de la API de invocación:

IBM Developer Kit para Java permite utilizar estas funciones de la API de invocación.

Nota: Antes de utilizar esta API, debe asegurarse de que está en un trabajo con capacidad multihebra. Consulte la sección Aplicaciones multihebra para obtener más información acerca de los trabajos con capacidad multihebra.

- **JNI_GetCreatedJavaVMs**

  Devuelve información sobre todas las máquina virtuales Java que se han creado. Aunque esta API está diseñada para devolver información para múltiples máquinas virtuales Java (JVM), solamente puede existir una JVM para un proceso. Por consiguiente, esta API devolverá un máximo de una JVM.

  Firma:

```
jint JNI_GetCreatedJavaVMs(JavaVM **vmBuf,
                                          jsize bufLen,
                                          jsize *nVMs);
```

  vmBuf es un área de salida cuyo tamaño viene determinado por bufLen, que es el número de punteros. Cada máquina virtual Java tiene una estructura JavaVM asociada que está definida en java.h. Esta API almacena un puntero que señala hacia la estructura JavaVM que está asociada a cada una de las máquinas virtuales Java creadas en vmBuf, a menos que vmBuf sea 0. Los punteros que señalan a estructuras JavaVM se almacenan en el orden de las máquinas virtuales Java correspondientes que se crean. nVMs devuelve el número de máquinas virtuales que hay creadas actualmente. El servidor permite crear más de una máquina virtual Java, por lo que cabe esperar un valor superior a uno. Esta información, junto con el tamaño de vmBuf, determina si se devuelven o no los punteros que señalan hacia las estructuras JavaVM de cada una de las máquinas virtuales Java creadas.

- **JNI_CreateJavaVM**

  Le permite crear una máquina virtual Java y utilizarla después en una aplicación.

Firma:

```
jint JNI_CreateJavaVM(JavaVM **p_vm,
                      void **p_env,
                      void *vm_args);
```

p_vm es la dirección de un puntero de JavaVM para la máquina virtual Java que se ha creado. Hay otras API de invocación de JNI que utilizan p_vm para identificar la máquina virtual Java. p_env es la dirección de un puntero de Entorno JNI para la máquina virtual Java de nueva creación. Señala hacia una tabla de funciones de JNI que inician dichas funciones. vm_args es una estructura que contiene los parámetros de inicialización de la máquina virtual Java.

Si inicia un mandato Ejecutar Java (RUNJVA) o un mandato JAVA y especifica una propiedad que tenga un parámetro de mandato equivalente, este parámetro tiene preferencia. Se hace caso omiso de la propiedad.

Para obtener una lista de las propiedades exclusivas de 0S/400 soportadas por la API JNI_CreateJavaVM, vea: "Propiedades Java del sistema" en la página 15.

Nota: Java en System i5 permite crear solo una máquina virtual Java (JVM) dentro de un único trabajo o proceso. Hallará más información en: "Spoorte para múltiples máquinas virtuales Java"

- **DestroyJavaVM**

  Destruye la máquina virtual Java.

  Firma:

  ```
  jint DestroyJavaVM(JavaVM *vm)
  ```

  Cuando se crea la máquina virtual Java, vm es el puntero de JavaVM devuelto.

- **AttachCurrentThread**

  Conecta una hebra con una máquina virtual Java para que la hebra pueda utilizar los servicios de la máquina virtual Java.

  Firma:

  ```
  jint AttachCurrentThread(JavaVM *vm,
                                     void **p_env,
                                     void *thr_args);
  ```

  El puntero de JavaVM, vm, identifica la máquina virtual Java a la que se conecta la hebra. p_env es el puntero que señala hacia la ubicación en la que está situado el puntero de interfaz JNI de la hebra actual. thr_args contiene argumentos de conexión de hebra específicos de la VM.

- **DetachCurrentThread**

  Firma:

  ```
  jint DetachCurrentThread(JavaVM *vm);
  ```

  vm identifica la máquina virtual Java de la que se desconecta la hebra.

Interfaces Java nativas de Sun Microsystems, Inc.

Spoorte para múltiples máquinas virtuales Java:

Java en la plataforma del System i5 ya no permite crear más de una máquina virtual Java dentro de un solo trabajo o proceso. Esta restricción afecta solamente a aquellos usuarios que crean máquinas JVM utilizando la API de invocación de interfaz Java nativa (JNI). Este cambio en el soporte no afecta a cómo se utiliza el mandato java para ejecutar los programas Java.

No puede llamar a JNI_CreateJavaVM() satisfactoriamente más de una vez en un trabajo, y JNI_GetCreatedJavaVMs() no puede devolver más de una JVM en una lista de resultados.

El soporte para crear solamente una JVM individual dentro de un solo trabajo o proceso sigue los estándares de la implementación de referencia de Java de Sun Microsystems, Inc.

Ejemplo: API de invocación Java:

Este ejemplo sigue el paradigma de la API de invocación estándar.

Hace lo siguiente:
- Se crea una máquina virtual Java mediante JNI_CreateJavaVM.
- Se utiliza la máquina virtual Java para buscar el archivo de clase que se desea ejecutar.
- Se busca el ID del método main de la clase.
- Se llama al método main de la clase.
- Se notifican los errores si se produce una excepción.

Al crear el programa, el programa de servicio QJVAJNI o QJVAJNI64 proporciona la función de API de invocación JNI_CreateJavaVM. JNI_CreateJavaVM crea la máquina virtual Java.

Nota: QJVAJNI64 es un nuevo programa de servicio para teraespacio/método nativo LLP64 y soporte de API de invocación.

Estos programas de servicio residen en el directorio de enlace del sistema y no es necesario identificarlos explícitamente en un mandato crear de lenguaje de control (CL). Por ejemplo, no identificaría explícitamente los programas de servicio mencionados anteriormente al utilizar el mandato Crear programa (CRTPGM) o el mandato Crear programa de servicio (CRTSRVPGM).

Una manera de ejecutar este programa es utilizar el siguiente mandato de lenguaje de control:

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```

Todo trabajo que cree una máquina virtual Java debe tener capacidad multihebra. La salida del programa principal, así como cualquier salida del programa, va a parar a los archivos en spool QPRINT. Estos archivos en spool resultan visibles si se utiliza el mandato de lenguaje de control (CL) Trabajar con trabajos sometidos (WRKSBMJOB) y se visualiza el trabajo iniciado utilizando el mandato CL Someter trabajo (SBMJOB).

Ejemplo: Utilizar la API de invocación Java

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
#define OS400_JVM_15
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

/* Especificar el pragma que provoca que todas las series literales
 * del código fuente se almacenen en ASCII (que, para las series
 * utilizadas, es equivalente a UTF-8)
 */

#pragma convert(819)

/* Procedimiento: Oops
 *
 * Descripción: Rutina de ayuda a la que se llama cuando una función JNI
 *              devuelve un valor cero, indicando un error grave.
 *              Esta rutina informa de la excepción a la salida de error estándar y
 *              finaliza la JVM abruptamente con una llamada a FatalError.
 *
 * Parámetros:  env -- JNIEnv* que debe utilizarse para llamadas JNI
 *              msg -- char* que señala hacia la descripción del error en UTF-8
 *
 * Nota:        El control no se devuelve después de la llamada a FatalError
 *              y no se devuelve desde este procedimiento.
```

```
|   */
|
|  void Oops(JNIEnv* env, char *msg) {
|      if ((*env)->ExceptionOccurred(env)) {
|          (*env)->ExceptionDescribe(env);
|      }
|      (*env)->FatalError(env, msg);
|  }
|
|  /* Esta es la rutina "main" del programa. */
|  int main (int argc, char *argv[])
|  {
|
|      JavaVMInitArgs initArgs; /* Estructura de inicialización de máquina virtual, pasada por
|                               * referencia a JNI_CreateJavaVM(). Los detalles están en jni.h
|                               */
|      JavaVM* myJVM;           /* Puntero de JavaVM establecido por llamada a
|                               * JNI_CreateJavaVM */
|      JNIEnv* myEnv;           /* Puntero de JNIEnv establecido por llamada a
|                               * JNI_CreateJavaVM */
|       char*   myClasspath;     /* 'Serie' de vía de acceso de clases modificable */
|      jclass  myClass;         /* Clase a la que hay que llamar, 'NativeHello'. */
|      jmethodID mainID;        /* ID de método de esta rutina 'main' routine. */
|      jclass  stringClass;     /* Necesaria para crear la String[] arg para main */
|      jobjectArray args;       /* La propia String[] */
|      JavaVMOption options[1]; /* Matriz de opciones -- utilizar opciones para
|                               * establecer vía de acceso de clases */
|      int     fd0, fd1, fd2;   /* Descriptores de archivo para IO */
|
|      /* Abrir descriptores de archivo para que IO funcione. */
|      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);
|
|      /*  Establecer campo versión de argumentos de inicialización para J2SE v1.5. */
|      initArgs.version = 0x00010005;
|   /*  Para utilizar J2SDK v1.4, establezca initArgs.version = 0x00010004; */
|
|
|      /* Ahora, interesa especificar el directorio para que la clase
|       * se ejecute en la vía de acceso de clases.
|       * Con Java2, la vía de acceso de clases se pasa como opción.
|       * Nota: debe especificar el nombre de directorio en formato UTF-8. Envuelva
|       " los bloques de código con sentencias #pragma convert.
|       */
|      options[0].optionString="-Djava.class.path=/CrtJvmExample";
|   /*para utilizar J2SDK v1.4, sustituya '1.5' por '1.4'.
|   options[1].optionString="-Djava.version=1.5" */
|
|      initArgs.options=options;  /* Pasar la vía de acceso de clases configurada. */
|      initArgs.nOptions = 1;     /* Pasar vía de acceso de clases y opciones de versión */
|
|      /*  Crear la JVM -- un código de retorno no cero indica que se produjo
|       *  un error. Vuelva a EBCDIC y escriba un mensaje en stderr
|       *  antes de salir del programa.
|       */
|      if (JNI_CreateJavaVM(myJVM, (void **)myEnv, (void *)initArgs)) {
|    #pragma convert(0)
|           fprintf(stderr, "No se ha podido crear la JVM\n");
|    #pragma convert(819)
|            exit(1);
|      }
|
|      /*  Utilizar la JVM recién creada para localizar la clase de ejemplo,
|       *  denominada 'NativeHello'.
|       */
|      myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
```

```
    if (! myClass) {
        Oops(myEnv, "No se ha encontrado la clase 'NativeHello'");
    }

    /* Ahora, obtener el identificador de método del punto de entrada 'main'
     * de la clase.
     * Nota: la firma de 'main' siempre es la misma para cualquier
     *       clase llamada mediante el siguiente mandato java:
     *           "main" , "([Ljava/lang/String;)V"
     */
    mainID = (*myEnv)->GetStaticMethodID(myEnv,myClass,"main",
                                         "([Ljava/lang/String;)V");
    if (! mainID) {
        Oops(myEnv, "No se ha encontrado jmethodID de 'main'");
    }

    /* Obtener la jclass para String para crear la matriz
     * de String que debe pasarse a 'main'.
     */
    stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
    if (! stringClass) {
        Oops(myEnv, "No se ha encontrado java/lang/String");
    }

    /*  Ahora, es necesario crear una matriz de series vacía,
     *  dado que main requiere una matriz de este tipo como parámetro.
     */
    args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
    if (! args) {
        Oops(myEnv, "No se ha podido crear la matriz de args");
    }

    /* Ahora, ya tiene el methodID de main y la clase, así que puede
     * llamar al método main.
     */
    (*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);

    /* Comprobar si hay errores. */
    if ((*myEnv)->ExceptionOccurred(myEnv)) {
        (*myEnv)->ExceptionDescribe(myEnv);
    }

    /* Finalmente, destruir la JavaVM que creó. */
    (*myJVM)->DestroyJavaVM(myJVM);

    /* Eso es todo. */
    return 0;
}
```

Para obtener más información, consulte “API de invocación Java” en la página 213.

Consideraciones sobre los métodos nativos Java y las hebras

Puede utilizar métodos nativos para acceder a funciones que no están disponibles en Java. Para utilizar mejor Java junto con los métodos nativos, conviene aclarar unos cuantos conceptos.

- Una hebra Java, tanto si la ha creado Java como si es una hebra nativa conectada, tiene inhabilitadas todas las excepciones de coma flotante. Si la hebra ejecuta un método nativo que vuelve a habilitar las excepciones de coma flotante, Java no las desactivará por segunda vez. Si la aplicación de usuario no las inhabilita antes de regresar para ejecutar el código Java, es posible que el comportamiento del código Java no sea el correcto si se produce una excepción de coma flotante. Cuando una hebra nativa se desconecta de la máquina virtual Java, su máscara de excepción de coma flotante se restaura en el valor que estaba en vigor en el momento de conectarse.
- Cuando una hebra nativa se conecta a la máquina virtual Java, la máquina virtual Java cambia la prioridad de las hebras, si conviene, para ajustarse a los esquemas de prioridad de uno a diez que

define Java. Cuando la hebra se desconecta, la prioridad se restaura. Después de conectarse, la hebra puede cambiar la prioridad de hebra utilizando una interfaz de método nativo (por ejemplo, una API POSIX). Java no cambiará la prioridad de hebra en las transiciones de regreso a la máquina virtual Java.

- El componente API de invocación de la interfaz Java nativa (JNI) permite que un usuario intercale una máquina virtual Java en su aplicación. Si una aplicación crea una máquina virtual Java y resulta que la máquina virtual Java finaliza de manera anómala, se indica la excepción MCH74A5 de System i5 "Máquina virtual Java terminada" a la hebra inicial del proceso si dicha hebra estaba conectada a la máquina virtual Java en el momento de finalizar esta. Los motivos por lo que puede finalizar de forma anómala la máquina virtual Java son los siguientes:
  - El usuario llama al método java.lang.System.exit().
  - Finaliza una hebra que la máquina virtual Java necesita.
  - Se produce un error interno en la máquina virtual Java.

  Este comportamiento es distinto del de la mayoría de las plataformas Java. En la mayoría de las demás plataformas, el proceso que crea automáticamente la máquina virtual Java finaliza de manera brusca tan pronto como finaliza la máquina virtual Java. La aplicación, si supervisa y maneja una excepción MCH74A5 indicada, puede seguir ejecutándose. De lo contrario, el proceso finaliza si la excepción queda sin manejar. Si se añade el código que se encarga de la excepción MCH74A5 específica del sistema System i5, podría verse mermado el grado de portabilidad de la aplicación a otras plataformas.

Dado que los métodos nativos se ejecutan siempre en un proceso multihebra, el código que contienen debe ser seguro en ejecución multihebra. Este hecho impone a los lenguajes y a las funciones que se utilizan para los métodos nativos las siguientes restricciones:

- No se debe utilizar ILE CL para métodos nativos, ya que este lenguaje no es seguro en ejecución multihebra. Para ejecutar mandatos CL seguros en ejecución multihebra, puede utilizar la función system() del lenguaje C o el método java.lang.Runtime.exec().
  - Para ejecutar mandatos CL seguros en ejecución multihebra desde un método nativo C o C++, utilice la función system() del lenguaje C.
  - Para ejecutar mandatos CL seguros en ejecución multihebra directamente desde Java, utilice el método java.lang.Runtime.exec().
- Se puede utilizar ILE C, ILE C++, ILE COBOL e ILE RPG para escribir un método nativo, pero todas las funciones a las que se llame desde dentro del método nativo deben ser seguras en ejecución multihebra.

  Nota: El soporte en tiempo de compilación para escribir métodos nativos solo se suministra actualmente para los lenguajes C, C++ y RPG. Escribir métodos nativos en otros lenguajes, si bien es posible, puede resultar mucho más complicado.

  Atención: ***no todas las funciones estándar C, C++, COBOL o RPG son seguras en ejecución multihebra.***
- Las funciones exit() y abort() de C y C++ no deben utilizarse nunca dentro de un método nativo. Estas funciones provocan la detención de todo el proceso que se ejecuta en la máquina virtual Java. Esto incluye todas las hebras del proceso, se hayan originado o no mediante Java.

  Nota: La función exit() a la que se hace referencia es la función de C y C++, y no es igual que el método java.lang.Runtime.exit().

Hallará más información sobre las hebras del servidor en: Aplicaciones multihebra.

Métodos nativos y la interfaz Java nativa

Los métodos nativos son métodos Java que se inician en un lenguaje distinto de Java. los métodos nativos pueden acceder a interfaces API y a funciones específicas del sistema que no están disponibles directamente en Java.

La utilización de métodos nativos limita la portabilidad de una aplicación porque en ellos interviene código específico del sistema. Un método nativo puede consistir en sentencias de código nativo nuevas o en sentencias de código nativo que llaman a código nativo existente.

Una vez que haya decidido que se necesita un método nativo, es posible que este tenga que interaccionar con la máquina virtual Java en la que se ejecuta. La interfaz Java nativa (JNI) hace que resulte más fácil esta interoperatividad de una manera neutral por lo que a la plataforma se refiere.

JNI es un conjunto de interfaces que permiten que un método nativo interaccione con la máquina virtual Java de muchas maneras. Por ejemplo, JNI incluye interfaces que crean objetos nuevos y llaman a métodos, que obtienen campos y los establecen, que procesan excepciones y manipulan series y matrices.

Para obtener una descripción completa de JNI, consulte: Interfaz Java nativa (JNI) de Sun Microsystems, Inc.

Información relacionada

Interfaz Java nativa (JNI) de Sun Microsystems, Inc.

Las series en los métodos nativos

Muchas funciones de la interfaz Java nativa (JNI) aceptan series de estilo de lenguaje C como parámetros. Por ejemplo, la función FindClass() de JNI acepta un parámetro de tipo serie que especifica el nombre totalmente calificado de un archivo de clase. Si se encuentra el archivo de clase, la función FindClass lo cargará, y al llamador de FindClass se le devolverá una referencia al archivo de clase.

Todas las funciones de JNI esperan que los parámetros de tipo serie estén codificados en UTF-8. Para obtener detalles acerca de UTF-8 puede consultar la especificación JNI, pero en la mayoría de los casos es suficiente con observar que los caracteres ASCII (American Standard Code for Information Interchange) de 7 bits son equivalentes a su representación en UTF-8. Los caracteres ASCII de 7 bits son en realidad caracteres de 8 bits, pero su primer bit siempre es 0. Por lo tanto, la mayoría de las series ASCII C ya tienen en realidad el formato UTF-8.

El compilador C del servidor opera en EBCDIC (extended binary-coded decimal interchange code) por defecto, lo que le permite proporcionar series a las funciones de JNI en UTF-8. Existen dos maneras de hacerlo. Se pueden utilizar series literales o bien series dinámicas. Las series literales son aquellas cuyo valor es conocido en el momento de compilar el código fuente. Las series dinámicas son aquellas cuyo valor no se conoce durante la compilación, sino que en realidad se calcula durante la ejecución.

Series literales en métodos nativos:

Resulta más fácil codificar las series literales en UTF-8 si la serie está compuesta por caracteres con una representación ASCII (American Standard for Information Interchange) de 7 bits.

Si la serie puede representarse en ASCII, como ocurre con la mayoría, puede ir entre sentencias 'pragma' que modifiquen la página de códigos actual del compilador. Entonces, el compilador almacenará internamente la serie en el formato UTF-8 que JNI requiere. Si la serie no puede representarse en ASCII, es más fácil tratar la serie EBCDIC original como si fuese una serie dinámica y procesarla con iconv() antes de pasarla a JNI.

Por ejemplo, para buscar una clase denominada `java/lang/String`, el código será el siguiente:

```
#pragma convert(819)
myClass = (*env)->FindClass(env,"java/lang/String");
#pragma convert(0)
```

La primera sentencia pragma, con el número 819, informa al compilador que debe almacenar todas las series entrecomilladas posteriores (series literales) en formato ASCII. La segunda sentencia pragma, con el número 0, indica al compilador que para las series entrecomilladas debe volver a la página de códigos

por omisión del compilador, que es normalmente la página de códigos EBCDIC 37. Así, incluyendo la llamada entre sentencias pragma, se cumple el requisito de JNI de que todos los parámetros estén codificados en UTF-8.

Atención: tenga cuidado con las sustituciones de texto. Por ejemplo, si el código es:

```
#pragma convert(819)
#define MyString "java/lang/String"
#pragma convert(0)
myClass = (*env)->FindClass(env,MyString);
```

La serie resultante es EBCDIC porque durante la compilación se sustituye el valor de MyString en la llamada a FindClass. En el momento de producirse esta sustitución, la sentencia pragma número 819 no ha entrado en vigor. Por tanto, las series literales no se almacenan en ASCII.

Convertir series dinámicas a y desde EBCDIC, Unicode y UTF-8:

Para manipular variables de tipo serie calculadas en tiempo de ejecución, puede ser necesario convertir las series a, o desde, EBCDIC, Unicode y UTF-8.

La API del sistema que proporciona la función de conversión de página de códigos es iconv(). Para utilizar iconv(), siga estos pasos:

1. Cree un descriptor de conversión con `QtqIconvOpen()`.
2. Llame a `iconv()` para que utilice el descriptor con el fin de convertir en una serie.
3. Cierre el descriptor mediante `iconv_close`.

En el ejemplo 3 de la utilización de Java Native Interface para ejemplos de métodos nativos, la rutina crea, utiliza y a continuación destruye el descriptor de conversión iconv dentro de la rutina. Esta estrategia evita los problemas que plantea la utilización multihebra del descriptor iconv_t, pero en el caso de código sensible al rendimiento es mejor crear un descriptor de conversión en almacenamiento estático y moderar el acceso múltiple a él utilizando una exclusión mutua (mutex) u otro recurso de sincronización.

Métodos nativos IBM i5/OS PASE para Java

La máquina virtual i5/OS Java permite que se utilicen métodos nativos en ejecución en el entorno i5/OS PASE. Antes de la V5R2, la JVM nativa de i5/OS solo empleaba métodos nativos ILE.

El soporte para métodos nativos i5/OS PASE incluye:

- Pleno uso de la interfaz System i5 Java nativa (JNI) desde los métodos nativos i5/OS PASE
- Capacidad para llamar a los métodos nativos i5/OS PASE desde la JVM nativa de i5/OS

Este nuevo soporte permite transportar fácilmente al servidor las aplicaciones Java que se ejecutan en AIX. Puede copiar los archivos de clase y las bibliotecas de métodos nativos de AIX en el sistema de archivos integrado del servidor y ejecutarlos desde cualquier indicador de mandatos de lenguaje de control (CL), de Qshell o de sesión de terminal i5/OS PASE.

Información relacionada

i5/OS PASE
En esta información se presupone que usted ya está familiarizado con i5/OS PASE. Si todavía no está familiarizado con PASE, vea este tema para aprender más sobre la utilización de los métodos nativos IBM i5/OS PASE con Java.

Variables de entorno Java de i5/OS PASE

La máquina virtual Java (JVM) utiliza las siguientes variables para iniciar los entornos i5/OS PASE. Debe establecer la variable QIBM_JAVA_PASE_STARTUP para poder ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java.

QIBM_JAVA_PASE_STARTUP

Debe establecer esta variable de entorno si se cumplen las dos condiciones siguientes:

- Se utilizan los métodos nativos i5/OS PASE
- Se propone iniciar Java desde un indicador de mandatos de i5/OS o desde un indicador de mandatos de Qshell

La JVM utiliza esta variable de entorno para iniciar un entorno PASE. El valor de la variable identifica un programa de arranque de i5/OS PASE. El servidor incluye dos programas de arranque de i5/OS PASE:

- /usr/lib/start32: inicia un entorno i5/OS PASE de 32 bits
- /usr/lib/start64: inicia un entorno i5/OS PASE de 64 bits

El formato de bits de todos los objetos de biblioteca compartida empleados por un entorno i5/OS PASE debe coincidir con el formato de bits del entorno i5/OS PASE.

No puede utilizar esta variable cuando inicie Java desde una sesión de terminal de i5/OS PASE. Una sesión de terminal de i5/OS PASE siempre utiliza un entorno i5/OS PASE de 32 bits. Las JVM iniciadas desde una sesión de terminal de i5/OS PASE utilizan el mismo tipo de entorno PASE que la sesión de terminal.

QIBM_JAVA_PASE_CHILD_STARTUP

Establezca esta variable de entorno opcional si el entorno i5/OS PASE de una JVM secundaria debe ser diferente del entorno i5/OS PASE de la JVM primaria. Una llamada a Runtime.exec() en Java inicia una JVM secundaria (o hija).

QIBM_JAVA_PASE_ALLOW_PREV

Establezca esta variable de entorno opcional cuando desee utilizar el entorno i5/OS PASE actual, si ya existe uno. En ciertas situaciones, resulta difícil determinar si ya existe un entorno i5/OS PASE. El hecho de utilizar QIBM_JAVA_PASE_ALLOW_PREV y QIBM_JAVA_PASE_STARTUP combinadas permite a la JVM utilizar un entorno i5/OS PASE existente o iniciar un nuevo entorno i5/OS PASE.

Referencia relacionada

"Ejemplos: variables de entorno para el ejemplo de IBM i5/OS PASE"
Para emplear el ejemplo de métodos nativos de IBM i5/OS PASE para Java, debe establecer variables de entorno.

Ejemplos: variables de entorno para el ejemplo de IBM i5/OS PASE:

Para emplear el ejemplo de métodos nativos de IBM i5/OS PASE para Java, debe establecer variables de entorno.

PASE_LIBPATH

El servidor utiliza esta variable de entorno de i5/OS PASE para identificar la ubicación de las bibliotecas de métodos nativos de i5/OS PASE. Puede establecer la vía de acceso a un único directorio o a varios directorios. En el caso de especificar varios directorios, utilice un carácter de dos puntos (:) para separar las entradas. El servidor también puede emplear la variable de entorno LIBPATH.

Para obtener más información sobre cómo utilizar Java, bibliotecas de métodos nativos y PASE_LIBPATH con este ejemplo, vea: "Gestionar bibliotecas de métodos nativos" en la página 225.

PASE_THREAD_ATTACH

Al establecer esta variable de entorno de i5/OS PASE en Y se hace que una hebra ILE no iniciada por i5/OS PASE se conecte automáticamente a OS/400 PASE al llamar a un procedimiento de i5/OS PASE.

Para obtener más información sobre las variables de entorno de i5/OS PASE, vea las entradas pertinentes en: Trabajar con variables de entorno de i5/OS PASE.

QIBM_JAVA_PASE_STARTUP

La JVM utiliza esta variable de entorno para iniciar un entorno i5/OS PASE. El valor de la variable identifica un programa de arranque de i5/OS PASE.

Para obtener más información, consulte "Variables de entorno Java de i5/OS PASE" en la página 221.

Utilizar QIBM_JAVA_PASE_CHILD_STARTUP:

La variable de entorno QIBM_JAVA_PASE_CHILD_STARTUP indica el programa de arranque de i5/OS PASE para las JVM secundarias.

Utilice QIBM_JAVA_PASE_CHILD_STARTUP cuando se cumplan todas las condiciones siguientes:

- La aplicación Java que desea ejecutar crea máquinas virtuales Java mediante llamadas Java a Runtime.exec().
- Las JVM primaria y secundarias utilizan métodos nativos i5/OS PASE.
- El entorno i5/OS PASE de las JVM secundarias debe ser distinto del entorno i5/OS PASE de la JVM primaria.

Si todas las condiciones indicadas anteriormente son ciertas, lleve a cabo las acciones siguientes:

- Establezca que la variable de entorno QIBM_JAVA_PASE_CHILD_STARTUP sea igual al programa de arranque de i5/OS PASE de las JVM secundarias.
- Al iniciar la JVM primaria desde un indicador de mandatos de i5/OS o desde Qshell, establezca que la variable de entorno QIBM_JAVA_PASE_STARTUP sea igual al programa de arranque de i5/OS PASE de la JVM primaria.

Nota: Al iniciar la JVM primaria desde una sesión de terminal i5/OS PASE, no establezca QIBM_JAVA_PASE_STARTUP.

El proceso de la JVM secundaria hereda la variable de entorno QIBM_JAVA_PASE_CHILD_STARTUP. Además, i5/OS establece la variable de entorno QIBM_JAVA_PASE_STARTUP del proceso de la JVM secundaria en el valor de la variable de entorno QIBM_JAVA_PASE_CHILD_STARTUP del proceso padre.

La siguiente tabla identifica los entornos i5/OS PASE resultantes (si los hay) de las diversas combinaciones de entornos y definiciones de mandatos de QIBM_JAVA_PASE_STARTUP y QIBM_JAVA_PASE_CHILD_STARTUP:

Tabla 7. Entornos PASE resultantes para QIBM_JAVA_PASE_STARTUP y QIBM_JAVA_PASE_CHILD_STARTUP

Entorno de inicio			**Comportamiento resultante**	
Entorno de mandatos	**QIBM_JAVA _PASE_STARTUP**	**QIBM_JAVA_PASE_ CHILD_STARTUP**	Arranque de i5/OS PASE de JVM primaria	Arranque de i5/OS PASE de JVM secundaria
CL o QSH	startX definido	startY definido	Utilizar startX	Utilizar startY
CL o QSH	startX definido	No definido	Utilizar startX	Utilizar startX
CL o QSH	No definido	startY definido	Ningún entorno i5/OS PASE	Utilizar startY
CL o QSH	No definido	No definido	Ningún entorno i5/OS PASE	Ningún entorno i5/OS PASE
Sesión de terminal i5/OS PASE	startX definido	startY definido	No permitido*	No permitido*
Sesión de terminal i5/OS PASE	startX definido	No definido	No permitido*	No permitido*

Tabla 7. Entornos PASE resultantes para QIBM_JAVA_PASE_STARTUP y QIBM_JAVA_PASE_CHILD_STARTUP (continuación)

Entorno de inicio			Comportamiento resultante	
Sesión de terminal i5/OS PASE	No definido	startY definido	Utilizar entorno de sesión de terminal i5/OSPASE	Utilizar startY
Sesión de terminal i5/OS PASE	No definido	No definido	Utilizar entorno de sesión de terminal i5/OSPASE	Ningún entorno i5/OS PASE

* Las filas marcadas como No permitido indican situaciones en las que la variable de entorno QIBM_JAVA_PASE_STARTUP podría entrar en conflicto con la sesión de terminal i5/OS PASE. Debido al posible conflicto, el uso de QIBM_JAVA_PASE_STARTUP no está permitido desde una sesión de terminal i5/OSPASE.

Utilizar QIBM_JAVA_PASE_ALLOW_PREV:

A veces resulta difícil determinar si ya existe un entorno i5/OS PASE. Utilizar QIBM_JAVA_PASE_ALLOW_PREV en combinación con QIBM_JAVA_PASE_STARTUP permite a la JVM determinar si debe utilizarse el entorno i5/OSPASE actual (si existe uno) o iniciar un nuevo entorno i5/OSPASE.

Para utilizar estas dos variables de entorno en combinación, establézcalas con los siguientes valores:

- Establezca QIBM_JAVA_PASE_STARTUP en el programa de arranque por omisión
- Establezca QIBM_JAVA_PASE_ALLOW_PREV en 1

Por ejemplo, una aplicación que inicie un entorno i5/OS PASE opcionalmente, llamará al programa que inicia la JVM. En este caso, al utilizar los valores anteriores, el programa puede utilizar el entorno i5/OS PASE actual, si existe uno, o iniciar un nuevo entorno i5/OSPASE.

La siguiente tabla identifica los entornos i5/OS PASE que sean resultado de las diversas combinaciones de entorno i5/OSPASE y definiciones de QIBM_JAVA_PASE_STARTUP y QIBM_JAVA_PASE_ALLOW_PREV:

Tabla 8. Entornos i5/OS PASE resultantes de combinaciones de entorno i5/OS PASE y de definiciones de QIBM_JAVA_PASE_STARTUP y QIBM_JAVA_PASE_ALLOW_PREV

Entorno de inicio			Comportamiento resultante
i5/OS Entorno PASE	**QIBM_JAVA _PASE_STARTUP**	**QIBM_JAVA_PASE _ALLOW_PREV**	**Arranque de i5/OS PASE de JVM**
Ninguno	No definido	No definido*	Ningún entorno i5/OS PASE
Ninguno	No definido	Definido '1'	Ningún entorno i5/OS PASE
Ninguno	startX definido	No definido*	Utilizar startX
Ninguno	startX definido	Definido '1'	Utilizar startX
Iniciado	No definido	No definido*	Utilizar entorno i5/OS PASE existente
Iniciado	No definido	Definido '1'	Utilizar entorno i5/OS PASE existente
Iniciado	startX definido	No definido*	No permitido: error de JVM durante el arranque
Iniciado	startX definido	Definido '1'	Utilizar entorno i5/OS PASE existente

* ″No definido″ significa que no se ha incluido QIBM_JAVA_PASE_ALLOW_PREV o que tiene un valor que no es 1.

Las dos últimas filas de la tabla anterior indican situaciones en las que es de utilidad establecer QIBM_JAVA_PASE_ALLOW_PREV. La JVM comprueba QIBM_JAVA_PASE_ALLOW_PREV cuando ya existe un entorno i5/OS PASE y se ha definido QIBM_JAVA_PASE_STARTUP. De lo contrario, la JVM ignora QIBM_JAVA_PASE_ALLOW_PREV.

Las variables de entorno QIBM_JAVA_PASE_ALLOW_PREV y QIBM_JAVA_PASE_CHILD_STARTUP son independientes la una de la otra.

Códigos de error Java de i5/OS PASE

Para ayudarle a resolver problemas relacionados con los métodos nativos de i5/OS PASE, este tema describe las condiciones de error indicadas por los mensajes de las anotaciones de trabajo de i5/OS y las excepciones Java de tiempo de ejecución. En la lista que sigue se describen los errores con los que se puede encontrar en el momento del arranque o de la ejecución al utilizar métodos nativos de i5/OS PASE para Java.

Errores de arranque

Para los errores de arranque, examine los mensajes de las anotaciones de trabajo pertinentes.

Errores de ejecución

Además de los errores de arranque, pueden aparecer las excepciones Java PaseInternalError o PaseExit en los datos de salida de Qshell de la JVM:

- PaseInternalError - Indica un error interno del sistema. Compruebe las entradas de las anotaciones del código interno bajo licencia.

  Para obtener más información sobre el código de error de PaseInternalError, consulte Qp2CallPase.
- PaseExit - La aplicación i5/OS PASE ha llamado a la función exit() o el entorno i5/OS PASE ha finalizado de forma anormal. Compruebe las anotaciones de trabajo y las anotaciones del código interno bajo licencia para obtener más información.

Gestionar bibliotecas de métodos nativos

Para emplear bibliotecas de métodos nativos, especialmente si desea gestionar múltiples versiones de una biblioteca de métodos nativos en el servidor iSeries, debe conocer los convenios de denominación de bibliotecas Java y el algoritmo de búsqueda de bibliotecas.

El i5/OS utiliza la primera biblioteca de métodos nativos que coincide con el nombre de la biblioteca que carga la máquina virtual Java (JVM). Para asegurarse de que el i5/OS localiza los métodos nativos correctos, debe evitar los conflictos de nombres de biblioteca y toda confusión acerca de qué biblioteca de métodos nativos utiliza la JVM.

Convenios de denominación de bibliotecas Java de i5/OS PASE y AIX

Si el código Java carga una biblioteca denominada Sample, el correspondiente archivo ejecutable se debe llamar libSample.a o libSample.so.

Orden de búsqueda de las bibliotecas Java

Cuando se habilitan los métodos nativos i5/OS PASE para la JVM, el servidor utiliza tres listas distintas (en el orden siguiente) para crear una única vía de búsqueda de bibliotecas de métodos nativos:

1. Lista de bibliotecas de i5/OS
2. Variable de entorno LIBPATH
3. Variable de entorno PASE_LIBPATH

Para efectuar la búsqueda, i5/OS convierte la lista de bibliotecas al formato del sistema de archivos integrado. Los objetos del sistema de archivos QSYS tienen nombres equivalentes en el sistema de archivos integrado, pero algunos objetos del sistema de archivos integrado no tienen nombres del sistema de archivos QSYS equivalentes. Como el cargador de bibliotecas busca los objetos tanto en el sistema de archivos QSYS como el sistema de archivos integrado, i5/OS utiliza el formato del sistema de archivos integrado para buscar las bibliotecas de métodos nativos.

La tabla siguiente muestra cómo i5/OS convierte las entradas de la lista de bibliotecas al formato del sistema de archivos integrado:

Entrada de lista de bibliotecas	Formato del sistema de archivos integrado
QSYS	/qsys.lib
QSYS2	/qsys.lib/qsys2.lib
QGPL	/qsys.lib/qgpl.lib
QTEMP	/qsys.lib/qtemp.lib

Ejemplo: buscar la biblioteca Sample2

En el ejemplo siguiente, LIBPATH se establece en /home/user1/lib32:/samples/lib32 y PASE_LIBPATH se establece en /QOpenSys/samples/lib.

La tabla siguiente, cuando se lee de principio a fin, indica la vía de acceso de búsqueda completa:

Origen	Directorios del sistema de archivos integrado
Lista de bibliotecas	/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: Los caracteres en mayúsculas y minúsculas solo son significativos en la vía de acceso /QOpenSys.

Para buscar la biblioteca Sample2, el cargador de bibliotecas Java busca los candidatos de archivos en el orden siguiente:

1. /qsys.lib/sample2.srvpgm
2. /qsys.lib/libSample2.a
3. /qsys.lib/libSample2.so
4. /qsys.lib/qsys2.lib/sample2.srvpgm
5. /qsys.lib/qsys2.lib/libSample2.a
6. /qsys.lib/qsys2.lib/libSample2.so
7. /qsys.lib/qgpl.lib/sample2.srvpgm
8. /qsys.lib/qgpl.lib/libSample2.a
9. /qsys.lib/qgpl.lib/libSample2.so
10. /qsys.lib/qtemp.lib/sample2.srvpgm
11. /qsys.lib/qtemp.lib/libSample2.a
12. /qsys.lib/qtemp.lib/libSample2.so
13. /home/user1/lib32/sample2.srvpgm

14. /home/user1/lib32/libSample2.a
15. /home/user1/lib32/libSample2.so
16. /samples/lib32/sample2.srvpgm
17. /samples/lib32/libSample2.a
18. /samples/lib32/libSample2.so
19. /QOpenSys/samples/lib/SAMPLE2.srvpgm
20. /QOpenSys/samples/lib/libSample2.a
21. /QOpenSys/samples/lib/libSample2.so

El i5/OS carga el primer candidato de la lista que existe realmente en la JVM como una biblioteca de métodos nativos. Aunque en la búsqueda se encuentren candidatos como '/qsys.lib/libSample2.a' y '/qsys.lib/libSample2.so', no es posible crear archivos del sistema de archivos integrado o enlaces simbólicos en los directorios /qsys.lib. Por consiguiente, aunque i5/OS busque estos archivos candidatos, nunca los encontrará en los directorios del sistema de archivos integrado que empiezan por /qsys.lib.

Sin embargo, puede crear enlaces simbólicos arbitrarios de otros directorios del sistema de archivos integrado a objetos i5/OS del sistema de archivos QSYS. En consecuencia, entre los candidatos de archivos válidos se encuentran archivos como /home/user1/lib32/sample2.srvpgm.

Ejemplo: método nativo IBM i5/OS PASE para Java

El ejemplo de método nativo IBM i5/OS PASE para Java llama a una instancia de un método nativo C que luego utiliza la interfaz Java nativa (JNI) para llamar de nuevo al código Java. En lugar de acceder a la serie directamente desde el código Java, el ejemplo llama a un método nativo que luego llama de nuevo a Java mediante JNI para obtener el valor de la serie.

Para ver las versiones HTML de los archivos fuente de ejemplo, utilice los enlaces siguientes:

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

- "Ejemplo: PaseExample1.java" en la página 557
- "Ejemplo: PaseExample1.c" en la página 558

Para poder ejecutar el ejemplo de método nativo i5/OS PASE, primero debe llevar a cabo las tareas de los siguientes temas:

1. "Ejemplo: descargar el código fuente de ejemplo a la estación de trabajo AIX" en la página 559
2. "Ejemplo: preparar el código fuente de ejemplo" en la página 559
3. "Ejemplo: preparar el System i5 para que ejecute el ejemplo de método nativo PASE para Java" en la página 560

Ejecutar el ejemplo de método nativo i5/OS PASE para Java

Tras completar las tareas anteriores, puede ejecutar el ejemplo. Utilice cualquiera de los mandatos siguientes para ejecutar el programa de ejemplo:

- Desde un indicador de mandatos de i5/OS:

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- Desde un indicador de mandatos de Qshell o desde una sesión de terminal i5/OS PASE:

```
cd /home/example
java PaseExample1
```

Métodos nativos del modelo de almacenamiento de teraespacio para Java

Ahora la máquina virtual i5/OS Java (JVM) permite usar métodos nativos del modelo de almacenamiento de teraespacio. El modelo de almacenamiento de teraespacio proporciona un entorno de direcciones locales de proceso de gran tamaño para programas ILE. El uso del modelo de almacenamiento de teraespacio le permite transportar código de métodos nativos de otros sistemas operativos a i5/OS sin tener que realizar ningún cambio en el código fuente (o realizando muy pocos cambios).

Nota: El modelo de almacenamiento de teraespacio proporciona un entorno en el que el almacenamiento estático, las variables locales y las asignaciones de memoria dinámica se localizan automáticamente en el almacenamiento de teraespacio. Si todo lo que necesita es habilitar el acceso al almacenamiento de teraespacio, **no** hace falta que utilice el modelo de almacenamiento de teraespacio. Basta con que habilite el código del método nativo para el teraespacio. Para habilitar el método nativo de cara al teraespacio, utilice el parámetro TERASPACE(*YES) en el mandato Crear módulo C (CRTCMOD), Crear módulo C++ (CRTCPPMOD) u otro mandato de creación de módulos.

Para conocer detalles sobre la programación con el modelo de almacenamiento de teraespacio, consulte la siguiente información:

- Capítulo 4 de ILE Concepts
- Capítulo 17 de WebSphere Development Studio ILE C/C++ Programmer's Guide

El concepto de métodos nativos Java creados para el modelo de almacenamiento de teraespacio es muy similar al de los métodos nativos que utilizan almacenamiento de un solo nivel. La JVM pasa a los métodos nativos del modelo de almacenamiento de teraespacio un puntero que señala hacia el entorno de la interfaz Java nativa (JNI) e Interface (JNI) que los métodos pueden utilizar para llamar a funciones JNI.

Para los métodos nativos del modelo de almacenamiento de teraespacio, la JVM proporciona implementaciones de funciones JNI que utilizan el modelo de almacenamiento de teraespacio y punteros de 8 bytes.

Crear métodos nativos del modelo de almacenamiento de teraespacio

Para crear satisfactoriamente un método nativo del modelo de almacenamiento de teraespacio, el mandato de creación del módulo del modelo de almacenamiento de teraespacio debe utilizar las siguientes opciones:

```
TERASPACE(*YES) STGMDL(*TERASPACE) DTAMDL(*LLP64)
```

La siguiente opción (*TSIFC), para utilizar funciones de almacenamiento de teraespacio, es opcional:

```
TERASPACE(*YES *TSIFC)
```

Nota: Si no utiliza DTAMDL(*LLP64), cuando emplee métodos nativos Java del modelo de almacenamiento de teraespacio y llame a un método nativo, se lanzará una excepción de tiempo de ejecución.

Crear programas de servicio del modelo de almacenamiento de teraespacio que utilicen métodos nativos

Para poder crear un programa de servicio del modelo de almacenamiento de teraespacio, utilice la siguiente opción en el mandato de lenguaje de control (CL) Crear programa de servicio (CRTSRVPGM):

```
CRTSRVPGM STGMDL(*TERASPACE)
```

Además, debería utilizar la opción ACTGRP(*CALLER), que permite a la JVM activar todos los programas de servicio de métodos nativos del modelo de almacenamiento de teraespacio en el mismo grupo de

activación de teraespacio. El uso de un grupo de activación de teraespacio de este modo puede ser importante para que los métodos nativos puedan manejar las excepciones de forma eficaz.

Para obtener más detalles sobre la activación de programas y los grupos de activación, vea el capítulo 3 de ILE Concepts.

Utilizar interfaces de invocación Java con los métodos nativos del modelo de almacenamiento de teraespacio

Utilice la función GetEnv de la API de invocación cuando el puntero del entorno JNI no coincida con el modelo de almacenamiento del programa de servicio. La función GetEnv de la API de invocación siempre devuelve el puntero de entorno JNI correcto.

La JVM da soporte a métodos nativos del modelo de almacenamiento de teraespacio y de un solo nivel, pero los dos modelos de almacenamiento utilizan entornos JNI distintos. Dado que los dos modelos de almacenamiento utilizan entornos JNI distintos, no pase el puntero de entorno JNI como un parámetro entre métodos nativos en los dos modelos de almacenamiento.

Conceptos relacionados

"API de invocación Java" en la página 213
La API de invocación, que forma parte de la interfaz Java nativa (JNI), permite al código no crear una máquina virtual Java, así como cargar y utilizar clases Java Esta función permite a un programa multihebra utilizar las clases Java que se ejecutan en múltiples hebras de una sola máquina virtual Java.

Información relacionada

Interfaz Java nativa (JNI) de Sun Microsystems, Inc.

Mandato CL Crear módulo C (CRTCMOD)

Mandato CL Crear módulo C++ (CRTCPPMOD)

Capítulo 3 de ILE Concepts

Websphere Development Studio ILE C/C++ Programmer's Guide

Comparación entre Integrated Language Environment y Java

En un System i5, el entorno Java está separado del entorno de lenguajes integrados (ILE). Java no es un lenguaje ILE y no se pueden enlazar con módulos de objeto ILE para crear programas o programas de servicio en un System i5.

ILE	Java
Los miembros que forman parte de la biblioteca o de la estructura de archivos de un servidor System i5 almacenan los archivos de código fuente.	Los archivos continuos del sistema de archivos integrado contienen el código fuente.
El programa de utilidad para entrada del fuente (SEU) edita archivos fuente EBCDIC.	Los archivos fuente ASCII (American Standard Code for Information Interchange) se editan normalmente con un editor de estación de trabajo.
El resultado de la compilación de archivos fuente son módulos de código objeto, que se almacenan en bibliotecas de un servidor System i5.	El resultado de la compilación del código fuente son archivos de clase, que se almacenan en el sistema de archivos integrado.
Los módulos objeto están unidos estáticamente entre sí en programas o programas de servicio.	Las clases se cargan dinámicamente según convenga en tiempo de ejecución.
Se puede llamar directamente a funciones escritas en otros lenguajes de programación ILE.	Para llamar a otros lenguajes desde Java, se debe utilizar la interfaz Java nativa (JNI).

ILE	Java
Los lenguajes ILE se compilan y ejecutan siempre en forma de instrucciones de lenguaje máquina.	Los programas Java pueden ser interpretados o compilados.

Utilizar java.lang.Runtime.exec()

Utilice el método java.lang.Runtime.exec para llamar a programas o mandatos desde dentro de un programa Java. Al utilizar el método java.lang.Runtime.exec() se crean uno o varios trabajos adicionales habilitados para hebras. Los trabajos adicionales procesan la serie de mandatos que se pasa en el método.

Nota: El método java.lang.Runtime.exec ejecuta los programas en un trabajo aparte, que es distinto a la función C system(). La función C system ejecuta programas en el mismo trabajo.

El proceso real que se produce depende de los siguientes elementos:

- La clase de mandato que pase en java.lang.Runtime.exec()
- El valor de la propiedad del sistema os400.runtime.exec

Procesar distintos tipos de mandatos

La tabla siguiente indica cómo java.lang.Runtime.exec() procesa distintas clases de mandatos y muestra los efectos de la propiedad del sistema os400.runtime.exec.

	Valor de la propiedad del sistema os400.runtime.exec	
Tipo de mandato	**EXEC (valor predeterminado)**	**QSHELL**
mandato java	Inicia un segundo trabajo que ejecuta la JVM. La JVM inicia un tercer trabajo que ejecuta la aplicación Java.	Inicia un segundo trabajo que ejecuta Qshell, el intérprete de shell. Qshell inicia un tercer trabajo para ejecutar la aplicación, programa o mandato Java.
programa	Inicia un segundo trabajo que ejecuta un programa ejecutable (programa i5/OS o programa i5/OSPASE).	
mandato CL	Inicia un segundo trabajo que ejecuta un programa i5/OS. El programa i5/OS ejecuta el mandato CL en el segundo trabajo.	

Nota: Al llamar a un mandato CL o un programa CL, asegúrese de que el CCSID del trabajo contiene los caracteres que pasa como parámetros al mandato llamado.

El proceso del segundo o del tercer trabajo se ejecuta concurrentemente con cualquier máquina virtual Java (JVM) del trabajo original. Cualquier proceso de salida o conclusión que tenga lugar en dichos trabajos no afecta a la JVM original.

propiedad del sistema os400.runtime.exec

Puede establecer el valor de la propiedad del sistema os400.runtime.exec como EXEC (el valor predeterminado) o QSHELL. El valor de os400.runtime.exec determina si java.lang.Runtime.exec() utiliza la interfaz EXEC o Qshell.

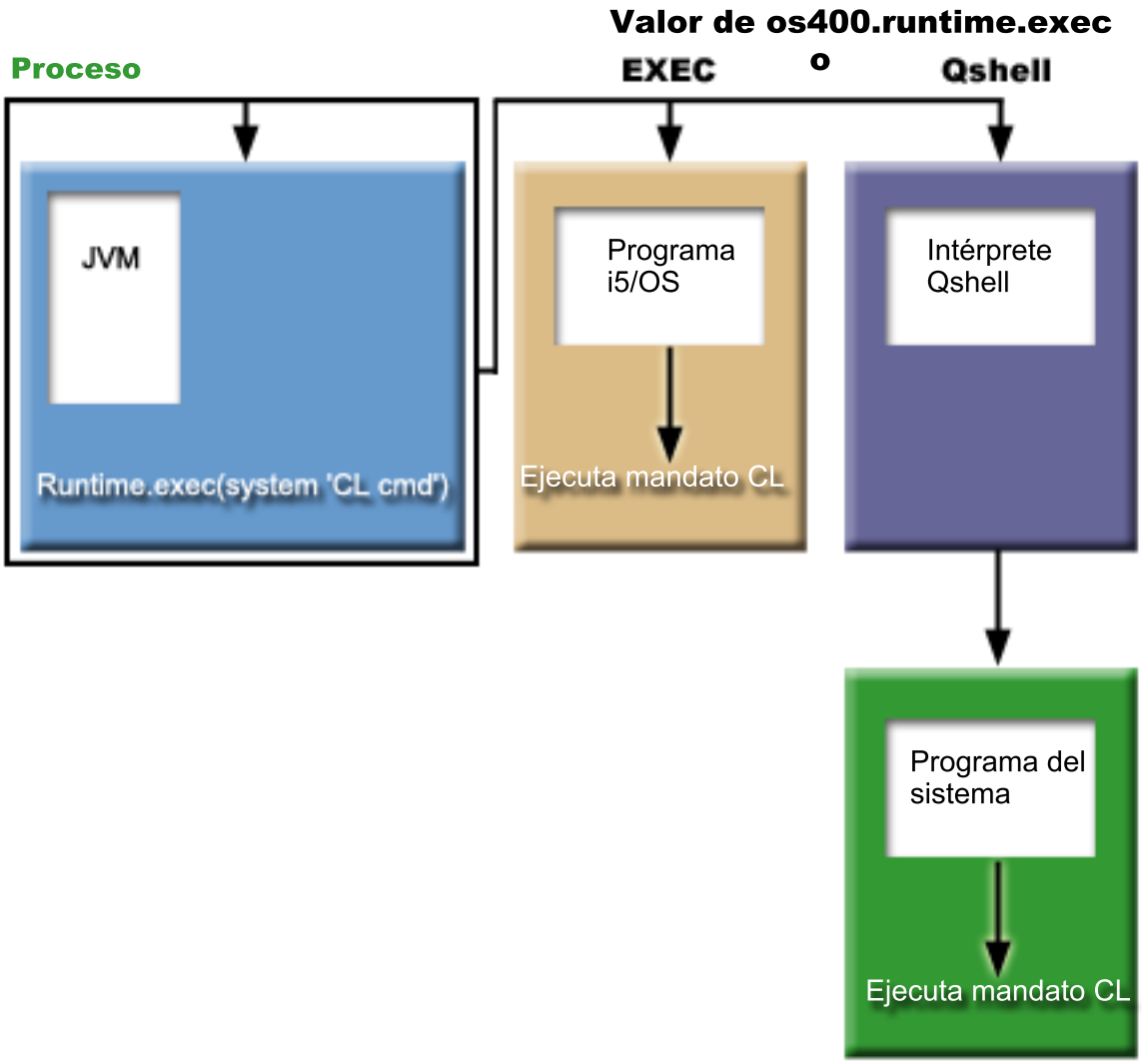
Utilizar un valor de EXEC en lugar de QSHELL tiene las siguientes ventajas:

- El programa Java que llama a java.lang.Runtime.exec() es más portable.
- Utilizar java.lang.Runtime.exec() para llamar a un mandato CL emplea menos recursos del sistema

Deberá utilizar java.lang.Runtime.exec() para ejecutar Qshell solamente cuando lo requiera la compatibilidad a la inversa. Utilizar java.lang.Runtime.exec() para ejecutar Qshell requiere que establezca os400.runtime.exec como QSHELL.

La siguiente ilustración muestra cómo utilizar un valor de QSHELL lanza un tercer trabajo, que consume recursos del sistema adicionales. Tenga presente que el hecho de utiliza el valor QSHELL disminuye la portabilidad del programa Java.

Figura 1. Utilizando un valor de QSHELL para la propiedad de sistema os400.runtime.exec



Además, al utilizar un valor de QSHELL, pasar un mandato CL a java.lang.Runtime.exec() requiere una sintaxis específica. Para obtener más información, consulte el ejemplo siguiente para llamar a un mandato CL.

Para obtener información sobre cómo establecer os400.runtime.exec, vea: Lista de propiedades Java del sistema.

Ejemplo: llamar a otros programa Java con java.lang.Runtime.exec()

En este ejemplo se enseña a llamar a otro programa Java con java.lang.Runtime.exec(). Esta clase llama al programa Hello que viene como parte de IBM Developer Kit para Java. Cuando la clase Hello escribe en System.out, este programa obtiene un handle para acceder a la corriente y puede leer en ella.

Nota: Para llamar al programa, utilice el intérprete Qshell.

Ejemplo 1: clase CallHelloPgm

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.io.*;

public class CallHelloPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallHelloPgm.main() invocado");

      // llamar a la clase Hello.
      try
      {
          theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
      }
      catch(IOException e)
      {
         System.err.println("Error en el método exec()");
         e.printStackTrace();
      }

      // leer en la corriente de salida estándar del programa llamado.
      try
      {
         inStream = new BufferedReader(
                              new InputStreamReader( theProcess.getInputStream() ));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error en inStream.readLine()");
         e.printStackTrace();
      }

   } // Finalizar el método.

} // Finalizar la clase
```

Ejemplo: llamar a un programa CL con java.lang.Runtime.exec()

En este ejemplo se muestra cómo ejecutar programas CL desde dentro de un programa Java. En este ejemplo, la clase Java CallCLPgm ejecuta un programa CL.

El programa CL utiliza el mandato Visualizar programa Java (DSPJVAPGM) para visualizar el programa asociado al archivo de clase Hello. En este ejemplo, se supone que el programa CL se ha compilado y está en una biblioteca que se llama JAVSAMPLIB. La salida del programa CL está en el archivo en spool QSYSPRT.

Para obtener un ejemplo de cómo llamar a un programa CL desde dentro de un programa Java, vea: "Ejemplo: llamar a un mandato CL con java.lang.Runtime.exec()" en la página 233.

Nota: La biblioteca JAVSAMPLIB no se crea como parte del proceso de instalación del programa bajo licencia (LP) IBM Developer número 5761-JV1. Tendrá que crear la biblioteca de manera explícita.

Ejemplo 1: clase CallCLPgm

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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 en el método exec()");
         e.printStackTrace();
      }
   } // Finalizar el método main()
} // Finalizar la clase
```

Ejemplo 2: visualizar un programa CL Java

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
               OUTPUT(*PRINT)
ENDPGM
```

Ejemplo: llamar a un mandato CL con java.lang.Runtime.exec()

Este ejemplo muestra cómo ejecutar un mandato de lenguaje de control (CL) desde un programa Java.

En este ejemplo, la clase Java ejecuta un mandato CL. El mandato CL utiliza el mandato CL Visualizar programa Java (DSPJVAPGM) para visualizar el programa asociado al archivo de clase Hello. La salida del mandato CL está en el archivo en spool QSYSPRT.

Al establecer la propiedad del sistema os400.runtime.exec como EXEC (que es el valor predeterminado), los mandatos que pase a la función `Runtime.getRuntime().exec()` utilizarán el siguiente formato:

```
Runtime.getRuntime()Exec("system CLCOMMAND");
```

donde *CLCOMMAND* es el mandato CL que desea ejecutar.

Nota: Al establecer os400.runtime.exec como QSHELL, debe añadir una barra inclinada y comillas (\"). Por ejemplo, el mandato anterior tendrá este aspecto:

```
Runtime.getRuntime()Exec("system \"CLCOMMAND\"");
```

Ejemplo: Clase para llamar a un mandato CL

El código siguiente presupone que utilizará el valor predeterminado de EXEC para la propiedad del sistema os400.runtime.exec.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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 en el método exec()");
         e.printStackTrace();
      }
   } // Finalizar el método main()
} // Finalizar la clase
```

Conceptos relacionados

"Utilizar java.lang.Runtime.exec()" en la página 230
Utilice el método java.lang.Runtime.exec para llamar a programas o mandatos desde dentro de un programa Java. Al utilizar el método java.lang.Runtime.exec() se crean uno o varios trabajos adicionales habilitados para hebras. Los trabajos adicionales procesan la serie de mandatos que se pasa en el método.

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Comunicaciones entre procesos

A la hora de comunicar con programas que se ejecutan en otro proceso, existen diversas opciones.

Una opción es utilizar sockets para la comunicación entre procesos. Un programa puede actuar a modo de programa servidor, a la escucha de una entrada procedente del programa cliente en una conexión por socket. El programa cliente se conecta al servidor por medio de un socket. Una vez establecida la conexión por socket, cualquiera de los dos programas puede enviar o recibir información.

Otra opción es utilizar archivos continuos para la comunicación entre programas. Para ello, se utilizan las clases System.in, System.out y System.err.

Una tercera opción consiste en utilizar IBM Toolbox para Java, que proporciona colas de datos y objetos de tipo mensaje de System i5.

También puede llamar a Java desde otros lenguajes, como se muestra en los ejemplos que siguen.

Información relacionada

IBM Toolbox para Java

Utilizar sockets para la comunicación entre procesos

Las corrientes por sockets comunican entre sí programas que se están ejecutando en procesos aparte.

Los programas se pueden iniciar por separado o mediante el método java.lang.Runtime.exec() desde el programa Java principal (main). Si un programa está escrito en un lenguaje distinto de Java, hay que asegurarse de que tiene lugar la conversión ASCII (American Standard Code for Information Interchange) o EBCDIC (extended binary-coded decimal interchange code). Encontrará más detalles en los temas que describen las codificaciones de caracteres Java.

Conceptos relacionados

"Utilizar java.lang.Runtime.exec()" en la página 230
Utilice el método java.lang.Runtime.exec para llamar a programas o mandatos desde dentro de un

programa Java. Al utilizar el método java.lang.Runtime.exec() se crean uno o varios trabajos adicionales habilitados para hebras. Los trabajos adicionales procesan la serie de mandatos que se pasa en el método.

"Codificaciones de caracteres de Java" en la página 26
Los programas Java pueden convertir datos en distintos formatos, permitiendo a las aplicaciones transferir y utilizar información de muchas clases de juegos de caracteres internacionales.

Ejemplo: utilizar sockets para la comunicación entre procesos:

En este ejemplo se utilizan sockets para la comunicación entre un programa Java y un programa C.

El programa C, que está a la escucha en un socket, debe iniciarse primero. Una vez que el programa Java se haya conectado al socket, el programa C le envía una serie utilizando la conexión por socket. La serie que se envía desde el programa C es una serie ASCII (American Standard Code for Information Interchange) de la página de códigos 819.

El programa Java se debe iniciar con el mandato `java TalkToC xxxxx nnnn` en la línea de mandatos del intérprete Qshell o en otra plataforma Java. También puede teclear `JAVA TALKTOC PARM(xxxxx nnnn)` en la línea de mandatos de i5/OS para iniciar el programa Java. `xxxxx` es el nombre de dominio o la dirección de protocolo Internet (IP) del sistema en el que se ejecuta el programa C. `nnnn` es el número de puerto del socket utilizado por el programa C. Este número de puerto también se tiene que utilizar como primer parámetro de la llamada al programa C.

Ejemplo 1: clase de cliente TalkToC

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();

   } // Finalizar el método main()

   public void setUp()
   {
      System.out.println("TalkToC.setUp() invocado");

      try
      {
         socket = new Socket(host, port);
         inStream = new BufferedReader(new InputStreamReader(
                                     socket.getInputStream()));
      }
      catch(UnknownHostException e)
      {
```

```
            System.err.println("No se puede encontrar el host llamado: " + host);
            e.printStackTrace();
            System.exit(-1);
         }
         catch(IOException e)
         {
            System.err.println("No se ha podido establecer conexión para " + host);
            e.printStackTrace();
            System.exit(-1);
         }
      } // Finalizar el método setUp().

      public void converse()
      {
         System.out.println("TalkToC.converse() invocado");

         if (socket != null && inStream != null)
         {
            try
            {
               System.out.println(inStream.readLine());
            }
            catch(IOException e)
            {
               System.err.println("Error de conversación con host " + host);
               e.printStackTrace();
            }

         } // Finalizar if.

      } // Finalizar el método converse()

      public void cleanUp()
      {
         try
         {

            if(inStream != null)
            {
               inStream.close();
            }
            if(socket != null)
            {
               socket.close();
            }
         } // Finalizar try.
         catch(IOException e)
         {
            System.err.println("Error de borrado");
            e.printStackTrace();
            System.exit(-1);
         }
      } // Finalizar el método cleanUp().

} // Finalizar la clase TalkToC.
```

SockServ.C se inicia pasando un parámetro correspondiente al número de puerto. Por ejemplo, `CALL SockServ '2001'`.

Ejemplo 2: programa servidor SockServ.C

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
#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 = "Este es un mensaje del servidor de sockets C.";
   #pragma convert (0)

   /*  Asignar el socket.  */
   if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
   {
      printf("anomalía en la asignación de socket\n");
      perror(NULL);
      exit(-1);
   }

   /* Realizar el enlace (bind). */
   if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
   {
     printf("Enlace fallido\n");
     perror(NULL);
     exit(-1);
   }

   /* Invocar el método listen. */
   listen(server, 1);

   /* Esperar la petición del cliente. */
   if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
   {
     printf("aceptar ha fallado\n");
     perror(NULL);
     exit(-1);
   }


   /* Envía un saludo (greeting) al cliente. */
   if((sendrc = send(client, greeting, strlen(greeting),0))<0)
   {
      printf("Envío fallido\n");
      perror(NULL);
      exit(-1);
   }
```

```
   close(client);
   close(server);

}
```

Utilizar corrientes de entrada y de salida para la comunicación entre procesos

Las corrientes de entrada y de salida comunican entre sí programas que se ejecutan en procesos aparte.

El método java.lang.Runtime.exec() ejecuta un programa. El programa padre puede obtener handles para acceder a las corrientes de entrada y de salida del proceso hijo y escribir o leer en ellas. Si el programa hijo está escrito en un lenguaje distinto de Java, es preciso asegurarse de que tiene lugar la conversión ASCII (American Standard Code for Information Interchange) o EBCDIC (Extended Binary-Coded Decimal Interchange Code). Vea las codificaciones de caracteres Java para obtener más detalles.

Conceptos relacionados

"Utilizar java.lang.Runtime.exec()" en la página 230
Utilice el método java.lang.Runtime.exec para llamar a programas o mandatos desde dentro de un programa Java. Al utilizar el método java.lang.Runtime.exec() se crean uno o varios trabajos adicionales habilitados para hebras. Los trabajos adicionales procesan la serie de mandatos que se pasa en el método.

"Codificaciones de caracteres de Java" en la página 26
Los programas Java pueden convertir datos en distintos formatos, permitiendo a las aplicaciones transferir y utilizar información de muchas clases de juegos de caracteres internacionales.

Ejemplo: utilizar corrientes de entrada y de salida para la comunicación entre procesos:

Este ejemplo muestra cómo llamar a un programa C desde Java y utilizar corrientes de entrada y salida para la comunicación entre procesos.

En este ejemplo, el programa C escribe una serie en su corriente de salida estándar y el programa Java la lee y la visualiza. En este ejemplo, se supone que se ha creado una biblioteca llamada JAVSAMPLIB y que en ella se ha creado el programa CSAMP1.

Nota: La biblioteca JAVSAMPLIB no se crea como parte del proceso de instalación del programa bajo licencia (LP) IBM Developer número 5761-JV1. Debe crearse de manera explícita.

Ejemplo 1: clase CallPgm

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.io.*;

public class CallPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallPgm.main() invocado");

      // llamar al programa CSAMP1
      try
      {
          theProcess = Runtime.getRuntime().exec(
                       "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
      }
      catch(IOException e)
      {
         System.err.println("Error en el método exec()");
```

```
            e.printStackTrace();
         }

         // leer en la corriente de salida estándar del programa llamado.
         try
         {
            inStream = new BufferedReader(new InputStreamReader
                               (theProcess.getInputStream()));
            System.out.println(inStream.readLine());
         }
         catch(IOException e)
         {
            System.err.println("Error en inStream.readLine()");
            e.printStackTrace();
         }

      } // Finalizar el método.

   } // Finalizar la clase
```

Ejemplo 2: programa C CSAMP1

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
     /* Convertir la serie a ASCII en tiempo de compilación */
#pragma convert(819)
     printf("Se ha invocado el programa JAVSAMPLIB/CSAMP1\n");
#pragma convert(0)
     /* es posible que Stdout esté en memoria intermedia, */
     /* así que hay que vaciar la memoria intermedia */

     fflush(stdout);

}
```

Ejemplo: llamar a Java desde C

Este es un ejemplo de programa C que utiliza la función system() para llamar al programa Java Hello.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
#include <stdlib.h>

 int main(void)
 {
   int result;

  /* La función system pasa la serie dada al procesador de mandatos CL
    para su proceso. */

   result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
 }
```

Ejemplo: llamar a Java desde RPG

Este es un ejemplo de un programa RPG que utiliza la API QCMDEXC para llamar al programa Java Hello.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
D*             DEFINIR LOS PARÁMETROS DE LA API QCMDEXC
D*
DCMDSTRING        S             25    INZ('JAVA CLASS(''com.ibm.as400.system.Hello'')')
DCMDLENGTH        S             15P 5 INZ(25)
D*             AHORA SE LLAMA A QCMDEXC CON EL MANDATO CL 'JAVA'
C                   CALL      'QCMDEXC'
C                   PARM                    CMDSTRING
C                   PARM                    CMDLENGTH
C*       La siguiente línea visualiza 'DID IT' después de salir de la
C*       shell Java por medio de F3 o F12.
C     'DID IT'      DSPLY
C*       Activar LR para salir del programa RPG
C                   SETON                                        LR
C
```

Plataforma Java

La plataforma Java es el entorno para desarrollar y gestionar applets y aplicaciones Java. Consta de tres componentes principales: el lenguaje Java, los paquetes Java y la máquina virtual Java.

El lenguaje y los paquetes Java son parecidos a C++ y a sus bibliotecas de clases. Los paquetes Java contienen clases, que están disponibles en cualquier implementación compatible con Java. La interfaz de programación de aplicaciones (API) debe ser la misma en cualquier sistema que soporte Java.

Java difiere de un lenguaje tradicional, como C++, en la forma en que se compila y ejecuta. En un entorno de programación tradicional, usted escribe y compila el código fuente de un programa en código objeto para un sistema operativo y un hardware específicos. El código objeto se enlaza con otros módulos de código objeto para crear un programa en ejecución. El código es específico con respecto a un conjunto determinado de hardware de sistema y no funciona en otros sistemas si no se realizan cambios. Esta figura muestra el entorno de despliegue de un lenguaje tradicional.

Applets y aplicaciones Java

El applet es un programa Java diseñado para incluirse en un documento Web HTML. Podrá escribir un applet Java y luego incluirlo en una página HTML de una manera muy parecida a cómo se incluye una imagen. Al utilizar un navegador habilitado para Java para ver una página HTML que contiene un applet, el código del applet se transfiere al sistema y la máquina virtual Java del navegador lo ejecuta.

El documento HTML contiene códigos que especifican el nombre del applet Java y su localizador uniforme de recursos (URL). El URL es la ubicación en la que residen los bytecodes del applet en Internet. Cuando se visualiza un documento HTML que contiene un código de applet Java, un navegador Web habilitado para Java descarga los bytecodes Java de Internet y utiliza la máquina virtual Java para procesar el código desde el documento Web. Estos applets Java son los que permiten que las páginas Web contengan gráficos animados o información interactiva.

También puede escribir una aplicación Java que no requiera la utilización de un navegador Web.

Para obtener más información, consulte Writing Applets, la guía de aprendizaje de Sun Microsystems' para applets Java. Incluye una visión general de los applets, instrucciones para escribir applets y algunos problemas comunes acerca de los applets.

Las **aplicaciones** son programas autónomos que no requieren la utilización de un navegador. Las aplicaciones Java se ejecutan iniciando el intérprete de Javadesde la línea de mandatos y especificando el archivo que contiene la aplicación compilada. Generalmente, las aplicaciones residen en el sistema en el que se despliegan. Las aplicaciones acceden a recursos del sistema y están restringidas por el modelo de seguridad Java.

Máquina virtual Java virtual machine

La máquina virtual Java es un entorno de tiempo de ejecución que se puede añadir a un navegador Web o a un sistema operativo, como IBM i5/OS. La máquina virtual Java ejecuta instrucciones generadas por un compilador Java. Consta de un intérprete de bytecode y un entorno de tiempo de ejecución que permiten ejecutar los archivos de clase Java en cualquier plataforma, sea cual sea la plataforma en la que se desarrollaron originariamente.

El cargador de clases y el gestor de seguridad, que forman parte del entorno de tiempo de ejecución Java, aíslan el código que proviene de otra plataforma. También pueden restringir a qué recursos del sistema puede acceder cada una de las clases que se cargan.

Nota: Las aplicaciones Java no están restringidas; tan solo lo están los applets. Las aplicaciones pueden acceder libremente a los recursos del sistema y utilizar métodos nativos. La mayoría de los programas de IBM Developer Kit para Java son aplicaciones.

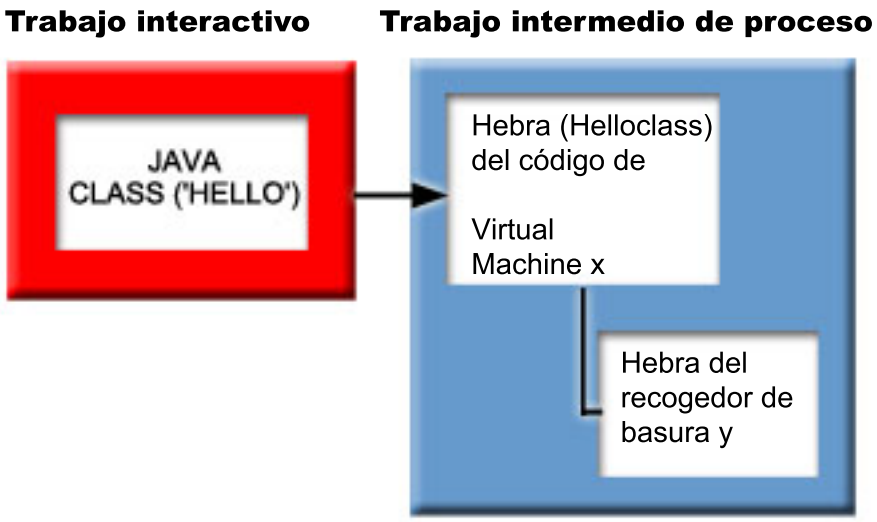
Puede utilizar el mandato Crear programa Java (CRTJVAPGM) para asegurarse de que el código cumple los requisitos de seguridad impuestos por el entorno de tiempo de ejecución Java para verificar el bytecode. Esto incluye forzar restricciones de tipos, comprobar conversiones de datos, garantizar que no se produzcan desbordamientos o subdesbordamientos de la pila de parámetros y comprobar las violaciones de acceso. Sin embargo, muchas veces no es necesario comprobar explícitamente los bytecodes. Si no utiliza el mandato CRTJVAPGM de antemano, las comprobaciones se producen durante la primera utilización de una clase. Una vez verificados los bytecodes, el intérprete decodifica los bytecodes y ejecuta las instrucciones de máquina necesarias para realizar las operaciones deseadas.

Además de cargar y ejecutar el bytecode, la máquina virtual Java incluye un recogedor de basura que gestiona la memoria. La “Recogida de basura en Java” en la página 418 se ejecuta al mismo tiempo que la carga y la interpretación del bytecode.

Entorno Java de tiempo de ejecución (JRE)

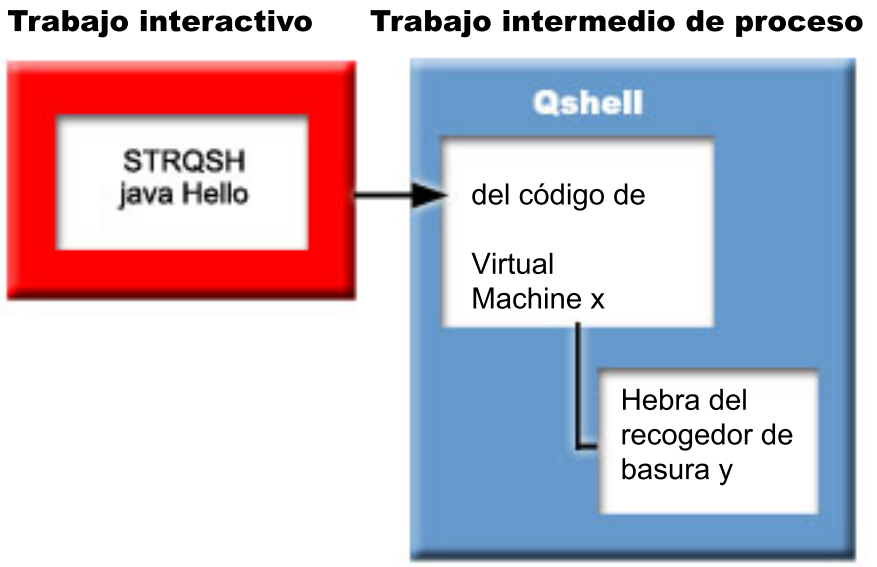
El entorno Java de tiempo de ejecución (JRE) se inicia siempre que se entra el mandato Ejecutar Java (RUNJVA) o el mandato JAVA en la línea de mandatos del i5/OS. Dado que el entorno Java es multihebra, es necesario ejecutar la máquina virtual Java en un trabajo que permita hebras, como puede ser un trabajo inmediato por lotes (BCI). Como se ilustra en la siguiente figura, una vez iniciada la máquina virtual Java, pueden iniciarse más hebras en el trabajo en que se ejecutará el recogedor de basura.

Figura 1: El típico entorno Java cuando se utiliza el mandato CL RUNJVA o JAVA



También es posible iniciar el entorno Java de tiempo de ejecución (JRE) utilizando el mandato `java` en Qshell desde el intérprete Qshell. En este entorno, el intérprete Qshell se ejecuta en un trabajo BCI asociado a un trabajo interactivo. El entorno Java de tiempo de ejecución (JRE) se inicia en el trabajo que ejecuta el intérprete Qshell.

Figura 2: el entorno Java cuando se utiliza el mandato java en Qshell



Cuando el entorno Java de tiempo de ejecución (JRE) se inicia desde un trabajo interactivo, aparece la pantalla de la shell Java. Esta pantalla proporciona una línea de entrada para incluir datos en la corriente System.in, así como para visualizar los datos que se escriben en la corriente System.out y en la corriente System.err.

Intérprete de Java

El intérprete de Java es el componente de la máquina virtual Java que interpreta los archivos de clase Java para una plataforma de hardware determinada. El intérprete Java decodifica cada bytecode y realiza la correspondiente operación.

Ejecutar un programa PASE de i5/OS con QP2TERM()

"Funciones de la API de invocación" en la página 214
IBM Developer Kit para Java permite utilizar estas funciones de la API de invocación.

Archivos JAR y de clase Java

El archivo Java de ARchivado (JAR) tiene un formato que combina muchos archivos en uno solo. El entorno Java difiere de otros entornos de programación en que el compilador Java no genera código de máquina para un conjunto de instrucciones específicas de hardware. En lugar de ello, el compilador Java convierte el código fuente Java en instrucciones de máquina virtual Java, almacenadas en los archivos de clase Java. Puede utilizar archivos JAR para almacenar los archivos de clase. El archivo de clase no está destinado a una plataforma de hardware específica, sino a la arquitectura de máquina virtual Java.

Puede utilizar los archivos JAR como herramienta de archivado general y también para distribuir programas Java de todos los tipos, incluidos los applets. Los applets Java se descargan en un navegador en una sola transacción del protocolo de transferencia de hipertexto (HTTP), en lugar de hacerlo abriendo una conexión nueva para cada componente. Este método de descarga aumenta la velocidad con la que el applet se carga en una página Web y empieza a funcionar.

JAR es el único formato de archivado independiente de plataforma. JAR también es el único formato que maneja archivos de audio e imagen, así como archivos de clase. JAR es un formato estándar abierto, totalmente ampliable, escrito en Java.

El formato JAR también da soporte a la compresión, lo cual reduce el tamaño del archivo y disminuye el tiempo de bajada. Además, el autor de un applet puede firmar digitalmente las entradas individuales de un archivo JAR para autenticar su origen.

Para actualizar clases en los archivos JAR, utilice la herramienta `jar`.

Los **archivos de clase Java** son archivos continuos que se producen cuando el compilador Java compila un archivo fuente. El archivo de clase contiene tablas que describen cada uno de los campos y métodos de la clase. El archivo también contiene el bytecode de cada método, datos estáticos y descripciones que se emplean para representar los objetos Java.

Información relacionada

Herramienta Java jar de Sun Microsytems, Inc.

Hebras Java

La hebra es una única corriente independiente que se ejecuta dentro de un programa. Java es un lenguaje de programación multihebra, por lo que en un momento dado puede haber más de una hebra que se ejecute en la máquina virtual Java. Las hebras Java proporcionan un medio de que el programa Java realice varias tareas al mismo tiempo. Una hebra es esencialmente un flujo de control de un programa.

Las hebras son construcciones de programación moderna que se utilizan para dar soporte a programas concurrentes y para mejorar el rendimiento y la escalabilidad de las aplicaciones. La mayoría de los lenguajes de programación soportan las hebras mediante bibliotecas de programación de complementos. Java soporta las hebras como interfaces de programación de aplicaciones (API) incorporadas.

Nota: La utilización de hebras proporciona el soporte necesario para aumentar la interactividad, consiguiendo con ello reducir la espera en el teclado al ejecutarse más tareas en paralelo. Sin embargo, el programa no es necesariamente más interactivo solo porque utilice hebras.

Las hebras son el mecanismo de espera en interacciones de larga ejecución, mientras permiten que el programa maneje otras tareas. Las hebras tienen la capacidad de dar soporte a varios flujos mediante la

misma corriente de código. A veces se las denomina **procesos ligeros**. El lenguaje Java incluye soporte directo para las hebras. Sin embargo, por diseño, no soporta la entrada y salida asíncrona sin bloques con interrupciones ni la espera múltiple.

Las hebras permiten el desarrollo de programas paralelos que se escalan adecuadamente en un entorno en el que una máquina tenga varios procesadores. Si se construyen apropiadamente, también proporcionan un modelo para el manejo de varias transacciones y usuarios.

Puede utilizar hebras en un programa Java en diversas situaciones. Algunos programas deben ser capaces de sumarse a varias actividades y continuar siendo capaces de responder a la entrada adicional por parte del usuario. Por ejemplo, un navegador Web debe ser capaz de responder a la entrada del usuario mientras reproduce un sonido.

Las hebras también pueden utilizar métodos asíncronos. Cuando el usuario llama a un segundo método, no es necesario que espere a que finalice el primer método para que el segundo continúe con su propia actividad.

Existen también muchas razones para no utilizar hebras. Si un programa utiliza de manera inherente una lógica secuencial, una sola hebra puede realizar la secuencia completa. En este caso, la utilización de varias hebras produce un programa complejo sin ninguna ventaja. La creación e inicio de una hebra conlleva un trabajo considerable. Si una operación solo implica unas pocas sentencias, resulta más rápido manejarla con una sola hebra. Esto puede ser cierto aún en el caso de que la operación sea conceptualmente asíncrona. Cuando varias hebras comparten objetos, estos deben sincronizarse para coordinar el acceso de las hebras y conservar la coherencia. La sincronización añade complejidad a un programa, resulta difícil ajustarlo para un rendimiento óptimo y puede ser una fuente de errores de programación.

Para obtener más información acerca de las hebras, vea: Desarrollar aplicaciones multihebra.

Java Development Kit de Sun Microsystems, Inc.

El Java Development Kit (JDK) es un software distribuido por Sun Microsystems, Inc., para los desarrolladores de Java. Incluye el intérprete Java, clases Java y herramientas de desarrollo Java (JDT): compilador, depurador, desensamblador, visor de applete, generador de archivos de apéndice y generador de documentación.

El JDK le permite escribir aplicaciones que se desarrollan una sola vez y se ejecutan en cualquier lugar de cualquier máquina virtual Java. Las aplicaciones Java desarrolladas con el JDK en un sistema se pueden usar en otro sistema sin tener que cambiar ni recompilar el código. Los archivos de clase Java son portables a cualquier máquina virtual Java estándar.

Para obtener más información sobre el JDK actual, consulte la versión del IBM Developer Kit para Java en su servidor.

Puede consultar la versión del IBM Developer Kit para Java predeterminado de la máquina virtual Java (JVM) en el servidor, entrando uno de estos mandatos:

- `java -version` en el indicador de mandatos de Qshell.
- `RUNJVA CLASS(*VERSION)` en la línea de mandatos CL.

Luego busque la misma versión del JDK de Sun Microsystems, Inc., en The Source for Java Technology java.sun.com, para localizar documentación específica. El IBM Developer Kit para Java es una implementación compatible de la tecnología Java de Sun Microsystems, Inc., por lo que debe de estar familiarizado con la correspondiente documentación del JDK.

Paquetes Java

El paquete Java es una forma de agrupar clases e interfaces relacionadas en Java. Los paquetes Java son similares a las bibliotecas de clases que están disponibles en otros lenguajes.

Los paquetes Java, que proporcionan las interfaces API Java, están disponibles como parte del Java Development Kit (JDK) de Sun Microsystems, Inc. Para obtener una lista completa de los paquetes Java e información sobre las API Java, vea: Paquetes de la plataforma Java 2.

Herramientas Java

Para obtener una lista completa de las herramientas suministradas por el Java Development Kit de Sun Microsystems, Inc., vea en manual de consulta de herramientas de Sun Microsystems, Inc. Para obtener más información sobre cada herramienta individual soportada por IBM Developer Kit para Java, vea: Herramientas Java soportadas por IBM Developer Kit para Java.

"Soporte para múltiples opciones del LP 5761-JV1" en la página 6
La plataforma del System i5 admite múltiples versiones de los Java Development Kits (JDK) y de la plataforma Java 2, Standard Edition.

"Métodos nativos y la interfaz Java nativa" en la página 219
Los métodos nativos son métodos Java que se inician en un lenguaje distinto de Java. los métodos nativos pueden acceder a interfaces API y a funciones específicas del sistema que no están disponibles directamente en Java.

Paquetes de la plataforma Java 2

Manual de consulta de herramientas de Sun Microsystems, Inc.

"Herramientas Java soportadas por IBM Developer Kit para Java" en la página 422
El entorno Qshell incluye las herramientas de desarrollo Java (JDT) que se suelen necesitar para el desarrollo de programas.

Temas avanzados

Este tema proporciona instrucciones para ejecutar Java en un trabajo por lotes y describe las autorizaciones de archivo Java necesarias en el sistema de archivos integrado para visualizar, ejecutar o depurar un programaJava.

Clases, paquetes y directorios Java

Cada clase Java forma parte de un paquete. La primera sentencia de un archivo fuente Java indica qué clase hay en cada paquete. Si el archivo fuente no contiene la sentencia package, significa que la clase forma parte de un paquete por omisión cuyo nombre no se indica.

El nombre del paquete está relacionado con la estructura de directorios en la que reside la clase. El sistema de archivos integrado da soporte a las clases Java en una estructura jerárquica de archivos parecida a la que existe en la mayoría de los sistemas UNIX y PC. Las clases Java deben almacenarse en un directorio que tenga una vía de acceso relativa que coincida con el nombre de paquete de la clase. Por ejemplo, observe la siguiente clase Java:

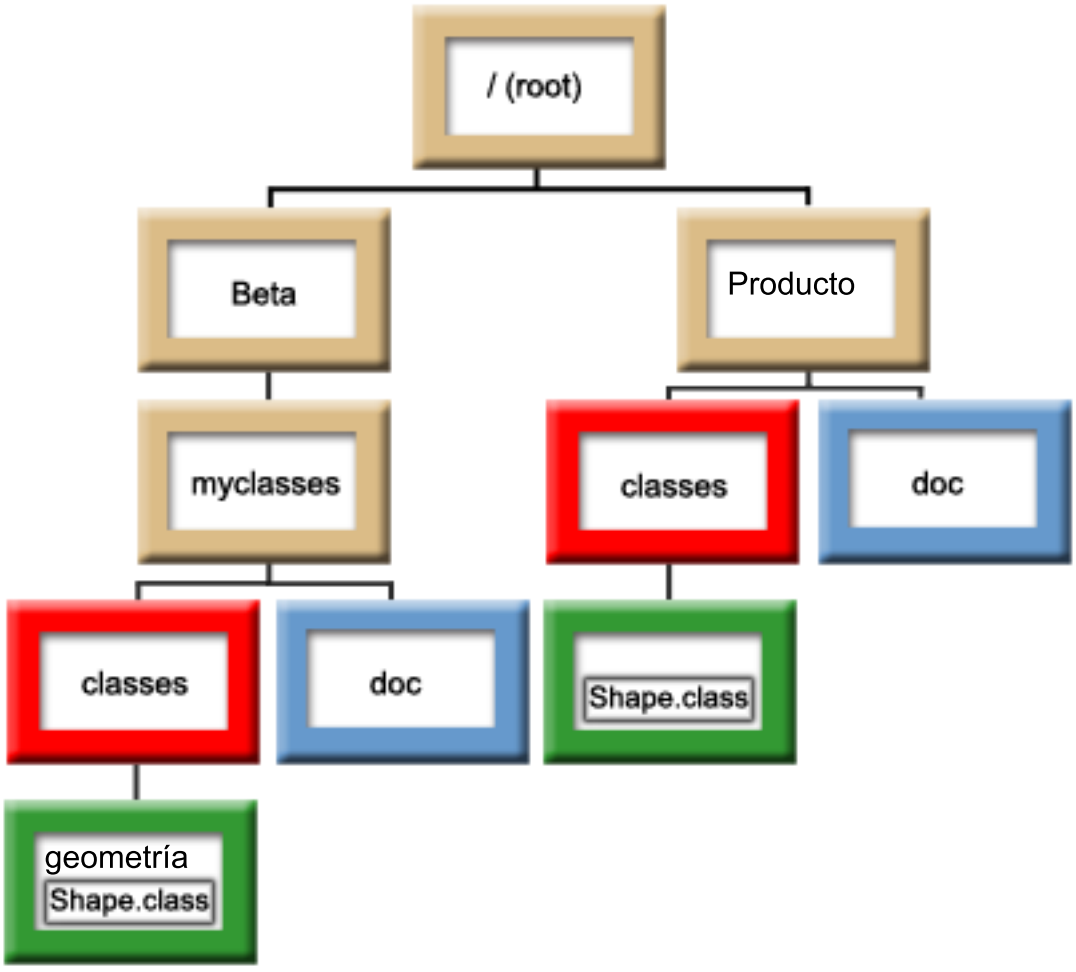
```
package classes.geometry;
import java.awt.Dimension;

public class Shape {

Dimension metrics;

   // El código de la implementación de la clase Shape estaría aquí ...

}
```

La sentencia package del código anterior indica que la clase Shape forma parte del paquete classes.geometry. Para que el entorno de tiempo de ejecución Java encuentre la clase Shape, almacene la clase Shape en la estructura de directorios relativa classes/geometry.

Nota: El nombre de paquete se corresponde con el nombre de directorio relativo en el que reside la clase. El cargador de clases de la máquina virtual Java busca la clase agregando el nombre de vía de acceso relativa a cada uno de los directorios que especifique en la vía de acceso de clases. El cargador de clases de la máquina virtual Java también puede encontrar la clase realizando una búsqueda en los archivos ZIP o JAR especificados en la vía de acceso de clases.

Por ejemplo, si almacena la clase Shape en el directorio /Product/classes/geometry del sistema de archivos "root" (/), deberá especificar /Product en la vía de acceso de clases.

Figura 1: Ejemplo de estructura de directorios para clases Java del mismo nombre en paquetes distintos



Nota: En la estructura de directorios pueden existir varias versiones de la clase Shape. Para utilizar la versión Beta de la clase Shape, coloque /Beta/myclasses dentro de la vía de acceso de clases antes de cualquier otro directorio o archivo ZIP que contenga la clase Shape.

El compilador Java utiliza la vía de acceso de clases Java, el nombre de paquete y la estructura de directorios para buscar los paquetes y las clases cuando compila el código fuente Java. Para obtener más información, consulte "Vía de acceso de clases Java" en la página 13.

Archivos relacionados con Java en el IFS

El sistema de archivos integrado (IFS) almacena los archivos JAR, los archivos ZIP, los archivos fuente y los archivos de clase relacionados con Java en una estructura jerárquica de archivos. IBM Developer Kit para Java permite utilizar los sistemas de archivos con seguridad multihebra en el IFS para almacenar y trabajar con los archivos JAR, archivos ZIP, archivos fuente y archivos de clase relacionados con Java.

Información relacionada

Consideraciones sobre sistemas de archivos para programación multihebra

Comparación de sistemas de archivos

Autorizaciones de archivo Java en el sistema de archivos integrado

Para ejecutar o depurar un programa Java, es necesario que el archivo de clase, JAR o ZIP tenga autorización de lectura (*R). Los directorios necesitan la autorización de lectura y la de ejecución (*RX).

Para utilizar el mandato Crear programa Java (CRTJVAPGM) con el fin de optimizar un programa, el archivo de clase, el archivo JAR o el archivo ZIP debe tener autorización de lectura (*R) y el directorio debe tener autorización de ejecución (*X). Si utiliza un patrón dentro del nombre de archivo de clase, el directorio debe tener autorización de lectura y de ejecución (*RX).

Para suprimir un programa Java con el mandato Suprimir programa Java (DLTJVAPGM), hay que tener autorización de lectura y escritura (*RW) sobre el archivo de clase, y el directorio debe tener autorización de ejecución (*X). Si utiliza un patrón dentro del nombre de archivo de clase, el directorio debe tener autorización de lectura y de ejecución (*RX).

Para visualizar un programa Java con el mandato Visualizar programa Java (DSPJVAPGM), hay que tener autorización de lectura (*R) sobre el archivo de clase, y el directorio debe tener la autorización de ejecución (*X).

Nota: Para un usuario que posea la autorización QSECOFR, siempre parecerá que los archivos y los directorios que carecen de autorización de ejecución (*X) tienen dicha autorización. Distintos usuarios pueden obtener resultados diferentes en determinadas situaciones, aunque aparentemente todos ellos tengan el mismo tipo de acceso a los mismos archivos. Es importante saberlo cuando se ejecutan scripts de shell mediante el intérprete Qshell o java.Runtime.exec().

Por ejemplo, un usuario escribe un programa Java que utiliza java.Runtime.exec() para llamar a un script de shell y lo prueba utilizando un ID de usuario que tiene la autorización QSECOFR. Si la modalidad de archivo del script de shell tiene autorización de lectura y de escritura (*RW), el sistema de archivos integrado permitirá que lo ejecute el ID de usuario que tiene la autorización QSECOFR. Sin embargo, podría ser que un usuario sin autorización QSECOFR intentase ejecutar el mismo programa Java y que el sistema de archivos integrado le indicase al código de java.Runtime.exec() que el script de shell no puede ejecutarse porque falta *X. En este caso, java.Runtime.exec() lanza una excepción de entrada y salida.

También puede asignar autorizaciones a archivos nuevos creados por programas Java en un sistema de archivos integrado. Utilizando la propiedad del sistema os400.file.create.authy para los archivos y os400.dir.create.auth para los directorios, puede utilizarse cualquier combinación de autorizaciones de lectura, escritura y ejecución.

Para obtener más información, consulte las secciones Las API de programa y mandato CL o Sistema de archivos integrado.

Ejecutar Java en un trabajo por lotes

Los programas Java se ejecutan en un trabajo por lotes mediante el mandato Someter trabajo (SBMJOB). En esta modalidad, la pantalla Entrada de mandato de Qshell de Java no está disponible para manejar las corrientes System.in, System.out y System.err.

Puede redirigir estas corrientes a otros archivos. Por omisión, las corrientes System.out y System.err se envían a un archivo en spool. El trabajo por lotes, que da como resultado una excepción de entrada y salida para las peticiones de lectura procedentes de System.in, es el propietario del archivo en spool. System.in, System.out y System.err se pueden redirigir dentro del programa Java. Para ello también se pueden utilizar las propiedades os400.stdin, os400.stdout y os400.stderr del sistema.

Nota: SBMJOB establece el directorio de trabajo actual (CWD) en el directorio HOME especificado en el perfil de usuario.

Ejemplo: ejecutar Java Java en un trabajo por lotes

```
SBMJOB CMD(JAVA Hello OPTION(*VERBOSE)) CPYENVVAR(*YES)
```

La ejecución del mandato JAVA en el ejemplo anterior genera un segundo trabajo. Por lo tanto, el subsistema en el que se ejecuta el trabajo debe tener capacidad para ejecutar más de un trabajo.

Para verificar que el trabajo por lotes tiene capacidad para ejecutar más de un trabajo, siga estos pasos:

1. En la línea de mandatos CL, escriba `DSPSBSD(MYSBSD)`, donde `MYSBSD` es la descripción de subsistema del trabajo por lotes.
2. Elija la opción 6, Entradas de cola de trabajos.
3. Observe el campo Máx. activo correspondiente a la cola de trabajos.

Ejecutar la aplicación Java en un host que no tenga una interfaz gráfica de usuario

Si desea ejecutar la aplicación Java en un host que no tenga una interfaz gráfica de usuario (GUI), como puede ser un System i5, puede utilizar Native Abstract Windowing Toolkit (NAWT).

Utilice NAWT para proporcionar a las aplicaciones y servlets Java las prestaciones completas de las funciones de gráficos AWT de Java 2 Platform, Standard Edition (J2SE).

NAWT (Native Abstract Windowing Toolkit)

Native Abstract Windowing Toolkit (NAWT) no es en realidad un kit de herramientas, sino más bien un término que ha evolucionado para referirse al soporte de i5/OS nativo que proporciona a las aplicaciones y servlets Java capacidad para utilizar las funciones gráficas de use the Abstract Windowing Toolkit (AWT) que ofrece la plataforma Java 2, Standard Edition (J2SE).

Nota: En general, la información de este apartado solo atañe a la JVM clásica original, no a la máquina virtual de tecnología IBM para Java. La excepción es que la información que sigue para iniciar y utilizar un servidor X de VNC atañe a ambas JVM si la aplicación emplea la API de AWT para interaccionar directamente con un usuario, o si de lo contrario utiliza componentes AWT pesados.

Nota: Cuando se utiliza JDK 1.4, NAWT no admite fonts y juegos de caracteres específicos del entorno local ni del idioma. Al utilizar NAWT, asegúrese de que está en conformidad con los siguientes requisitos:

- Utilice solamente caracteres que estén definidos en el juego de caracteres ISO8859-1.
- Utilice el archivo font.properties. El archivo font.properties reside en el directorio /QIBM/ProdData/Java400/jdknn/lib, donde *nn* es el número de versión de J2SE que está utilizando. No utilice ninguno de los archivos font.properties.xxx, donde *xxx* es un idioma u otro calificador.

Al utilizar JDK versión 1.5 o posterior, los fonts y juegos de caracteres empleados se determinan en el momento del arranque de Java mediante el valor de la propiedad file.encoding. Si file.encoding no tiene un valor explícito, se le da un valor predeterminado pertinente tomando como base el

CCSID del trabajo. Hallará más información en: “Valores de codificación de archivo predeterminada y de codificación Java” en la página 252.

Prerrequisitos para los programas bajo licencia

Para poder usar la API de AWT Java, hay que tener instalado IBM i5/OS Portable Application Solutions Environment (PASE). Hallará más información en: Instalar i5/OS PASE.

Instalar i5/OS PASE

Seleccionar una modalidad de AWT

Al utilizar AWT, se puede elegir entre dos modalidades: acéfala y normal. Uno de los factores que debe tener en cuenta al elegir la modalidad es si se propone utilizar componentes AWT pesados.

Modalidad acéfala

La modalidad acéfala está permitida si la aplicación Java no interacciona directamente con un usuario. Esto quiere decir que la aplicación Java no visualiza ventanas ni diálogos, no acepta entrada da teclado ni de ratón, ni tampoco utiliza componentes AWT pesados. Para seleccionar esta modalidad, se especifica la propiedad Java *java.awt.headless=true* en la invocación Java. Con la modalidad acéfala se evita la necesidad de tener un servidor VNC/X.

Algunas aplicaciones que pueden utilizar la modalidad acéfala son, por ejemplo:

- Servlets u otros programas basados en servidor que solo emplean la API de AWT para crear imágenes que deban incluirse en una corriente de datos devuelta a un usuario remoto.
- Cualquier programa que solo cree o manipule imágenes o archivo de imágenes sin visualizarlos realmente con componente AWT pesados.

Modalidad normal

La modalidad normal es necesaria si la aplicación utiliza la API de AWT Java para visualizar ventanas, marcos, diálogos u otros componentes pesados similares. Elija la modalidad normal si espera que la aplicación va a recibir eventos de operación del ratón o entrada del teclado. Si se propone utilizar la JVM clásica de IBM, hay que habilitar la modalidad normal especificando la propiedad Java *os400.awt.native=true* en la invocación Java. Si se propone utilizar la JVM de tecnología IBM para Java, la propiedad os400.awt.native no es necesaria y se ignora si se especifica.

Componentes AWT pesados

Los elementos que se consideran componentes AWT pesados son los que figuran a continuación. Si ve que su aplicación los necesita, debe elegir la modalidad normal:

Tabla 9. Componentes AWT pesados

Componentes AWT pesados			
Applet	Marco	Lista	Robot
Botón	Applet J	Menú	Barra de desplazamiento
Recuadro de selección	Diálogo J	Parra de menús	Panel de desplazamiento
Elección	Marco J	Componente de menú	Área de texto
Diálogo	Ventana J	Elemento de menú	Componente de texto
Diálogo de archivo	Etiqueta	Menú emergente	Ventana

Utilizar AWT en modalidad normal con soporte completo de interfaz gráfica de usuario:

Para poder utilizar una interfaz gráfica de usuario, se necesita un sistema de gestión de ventanas. En el caso de Java en i5/OS, la opción soportada es el servidor de informática de redes virtuales (VNC). El servidor VNC es el más indicado para el sistema porque no necesita un ratón, teclado y monitor con capacidad para gráficos que estén dedicados. IBM proporciona una versión del servidor VNC que se ejecuta en PASE. Siga estas instrucciones para asegurarse de que VNC está instalado, que se ha iniciado y que la sesión Java está configurada para utilizarlo.

Para poder probar AWT o empezar a usarlo, siga estos pasos necesarios y opcionales:

- Cree una contraseña VNC. Hay que hacerlo una vez para cada perfil de usuario que se empleará para iniciar un servidor VNC.
- Inicie el servidor VNC, normalmente después de cada IPL del sistema.
- Configure las variables de entorno de AWT, una vez en cada sesión y antes de ejecutar Java y utilizar la API de AWT por primera vez.
- Configure las propiedades Java del sistema. Hay que hacerlo cada vez que se ejecute Java.
- Opcional, para uso interactivo: configure el gestor de ventanas iceWM.
- Opcional, para interacción directa con un usuario: utilice un visor VNC o un navegador Web para establecer conexión con VNC.
- Opcional: verifique la configuración de AWT.

Crear un archivo de contraseñas VNC:

Para utilizar el Native Abstract Windowing Toolkit (NAWT) con un servidor de informática de redes virtuales (VNC), debe crear un archivo de contraseñas VNC.

El servidor VNC requiere por omisión un archivo de contraseñas que utiliza para proteger la pantalla de VNC contra el acceso de usuarios no autorizados. El archivo de contraseñas VNC se debe crear bajo el perfil empleado para iniciar el servidor VNC. En un indicador de mandatos de i5/OS, teclee:

1. `MKDIR DIR('/home/VNCprofile/.vnc')`
2. `QAPTL/VNCPASSWD USEHOME(*NO) PWDFILE('/home/VNCprofile/.vnc/passwd')`, siendo *VNCprofile* el perfil que inició el servidor VNC.

Para obtener acceso interactivo al servidor VNC utilizando un visor VNC o un navegador Web desde un sistema remoto, los usuarios deben utilizar la contraseña que usted especifique en este paso.

Iniciar el servidor VNC:

Para iniciar el servidor de informática de redes virtuales (VNC), siga estos pasos.

donde *n* es el número de pantalla que desea utilizar. Los números de pantalla pueden ser cualquier entero en el rango de 1 a 99.

El archivo .Xauthority

El proceso de iniciar el servidor VNC crea un nuevo archivo .Xauthority o modifica un archivo .Xauthority ya existente. La autorización de servidor VNC utiliza el archivo .Xauthority, que contiene información de clave cifrada, con lo que se impide que aplicaciones de otros usuarios intercepten las peticiones del servidor X. Las comunicaciones seguras entre la máquina virtual Java (JVM) y VNC **REQUIEREN** que tanto la JVM como VNC tengan acceso a la información de clave cifrada en el archivo .Xauthority.

El archivo .Xauthority pertenece al perfil que ha iniciado VNC. La manera más sencilla de permitir que tanto la JVM como el servidor VNC compartan el acceso al archivo .Xauthority es ejecutar el servidor

VNC y la JVM bajo el mismo perfil de usuario. Si no puede ejecutar el servidor VNC y la máquina virtual JVM bajo el mismo perfil de usuario, puede configurar la variable de entorno XAUTHORITY para señalar al archivo .Xauthority correcto.

Para iniciar el servidor de informática de redes virtuales (VNC), teclee el siguiente mandato en la línea de mandatos y pulse **Intro**: `CALL PGM(QSYS/QP2SHELL) PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n'))`, siendo *n* el número de pantalla que desea utilizar. Los números de pantalla pueden ser cualquier entero en el rango de 1 a 99.

Cuando se inicia el servidor VNC, aparece un mensaje que identifica el nombre de sistema de System i5 y su número de pantalla; por ejemplo, `El nuevo 'X'desktop es nombre_sistema:1`. Tome nota del nombre del sistema y del número de la pantalla, porque los empleará más adelante para configurar la variable de entorno DISPLAY cuando ejecute la aplicación Java que emplea AWT.

Si tiene más de un servidor VNC ejecutándose a la vez, se necesita un número de pantalla exclusivo para cada servidor VNC. Si especifica explícitamente el valor de pantalla al iniciar el servidor VNC como se ha indicado, podrá controlar qué número de pantalla utiliza cada aplicación. Por otro lado, si no quiere especificar el número de pantalla, quite `':n'` del mandato anterior, deje que el programa vncserver_java localice un número de pantalla disponible y tome nota de ese número.

Archivo .Xauthority

El proceso de iniciar el servidor VNC crea un nuevo archivo .Xauthority o modifica un archivo .Xauthority existente en el directorio inicial del usuario que inicia el servidor. El archivo .Xauthority contiene información de autorización de clave cifrada, que el servidor VNC emplea para impedir que las aplicaciones de otros usuarios intercepten las peticiones del servidor X. Las comunicación seguras entre la máquina virtual Java (JVM) y VNC **requiere** que tanto la JVM como VNC tengan acceso al mismo archivo .Xauthority.

El archivo .Xauthority pertenece al perfil que ha iniciado VNC. Para permitir que la JVM y el servidor VNC compartan el acceso, ejecútelos bajo el mismo perfil de usuario. Si no resulta posible, puede configurar la variable de entorno XAUTHORITY para que señale hacia el archivo .Xauthority correcto.

Configurar variables de entorno de NAWT:

Cuando ejecuta Java y desea utiliza pleno soporte de interfaz gráfica de usuario de AWT, debe tener definidas las variables de entorno DISPLAY y XAUTHORITY para indicar a Java qué pantalla de servidor X hay que utilizar y dónde está el archivo .Xauthority correcto.

Variable de entorno DISPLAY

En la sesión en la que desea ejecutar programas Java, establezca que la variable de entorno DISPLAY sea igual al nombre de sistema y al número de pantalla. En un indicador de mandatos de i5/OS, teclee el siguiente mandato y pulse Intro:

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('nombre_sistema:n')
```

Aquí, *nombre_sistema* es el nombre de host o la dirección IP de su sistema, y *n* es el número de pantalla del servidor VNC que hay que usar.

Variable de entorno XAUTHORITY

En la sesión en la que desea ejecutar programas Java, establezca que la variable de entorno XAUTHORITY sea igual a `/home/VNCprofile/.Xauthority`, siendo *VNCprofile* el perfil que inició el servidor VNC. Desde un indicador de mandatos de i5/OS, ejecute el mandato:

```
ADDENVVAR ENVVAR(XAUTHORITY) VALUE('/home/VNCprofile/.Xauthority')
```

Donde pone *VNCprofile*, debe escribir el nombre de perfil pertinente.

Valores de codificación de archivo predeterminada y de codificación Java:

En la tabla que sigue figuran las codificaciones Java para los valores de file.encoding o CCSID de JOB específicos. La codificación Java determina los fonts que se emplearán para visualizar las series de caracteres.

CCSID del trabajo	Codificación de archivo predeterminada	Codificación Java
00037	ISO8859_1	ISO-8859-1
00420	Cp1046	IBM-1046
00424	ISO8859_8	ISO-8859-8
00870	ISO8859_2	ISO-8859-2
00875	ISO8859_7	ISO-8859-7
00933	Cp970	EUC-KR
00935	Cp1383	IBM-1383
00937	Cp950	IBM-950
00939	Cp943C	IBM-943C
01026	ISO8859_9	ISO-8859-9
01399	Cp943C	IBM-943C
Predeterminado para los demás valores	ISO8859_1	ISO-8859-1

Configurar las propiedades Java del sistema necesarias para AWT:

Para poder utilizar la API Java, para AWT, deberá establecer algunas propiedades en la invocación Java.

Las propiedades que hay que establecer para cada versión de JDK y para cada tipo de JVM son las siguientes:

Tabla 10. Propiedades Java del sistema para AWT

	JVM clásica, JDK 1.4 y posterior	JVM de tecnología IBM para Java, JDK 1.5 y posterior
Modalidad normal (soporte completo de interfaz gráfica de usuario)	`os400.awt.native=true`	Ninguna[1]
Modalidad acéfala	`java.awt.headless=true`	`java.awt.headless=true`

[1] Aunque no se necesitan propiedades adicionales para la JVM de tecnología IBM para Java en modalidad normal, sí que debe realizar la configuración de variables de entorno y del servidor VNC descrita en: "Utilizar AWT en modalidad normal con soporte completo de interfaz gráfica de usuario" en la página 249.

Configurar el gestor de ventanas iceWM:

Configure iceWM window manager (iceWM), como un paso opcional durante la puesta a punto de NAWT, cuando desee utilizar interactivamente el servidor de informática de redes virtuales (VNC). Por ejemplo, puede interesarle ejecutar una aplicación Java que incluya una interfaz gráfica de usuario (GUI). iceWM es un pequeño pero potente gestor de ventanas incluido en la PRPQ de System i Tools For Developers.

Ejecutándose en un segundo plano, iceWM controla el aspecto de las ventanas que se ejecutan dentro del entorno X Window del servidor VNC. iceWM proporciona una interfaz y un conjunto de características similares a muchos gestores de ventanas populares. El comportamiento por omisión del script vncserver_java incluido inicia el servidor VNC y ejecuta iceWM.

Completando este paso se crean varios archivos de configuración que iceWM necesita. Si lo desea, también puede inhabilitar iceWM.

Configurar iceWM

Para configurar el gestor de ventanas iceWM, complete los pasos siguientes en un indicador de mandatos i5/OS. Asegúrese de llevar a cabo estos pasos bajo el perfil que utilice para iniciar el servidor VNC.

1. Teclee el siguiente mandato y pulse **Intro** para iniciar la instalación:

   ```
   STRPTL CLIENT(IGNORE)
   ```

   El valor IGNORE funciona como espacio reservado que asegura que el mandato activa solamente las características de configuración de STRPTL que NAWT necesita.
2. Teclee el siguiente mandato y pulse **INTRO** para finalizar la sesión:

   ```
   SIGNOFF
   ```

   Finalizar la sesión asegura que los resultados específicos de la sesión derivados del mandato STRPTL no afectarán a las acciones subsiguientes que realice para utilizar o configurar NAWT.

Nota: Ejecute el mandato STRPTL una sola vez para cada perfil que inicie un servidor VNC. NAWT no requiere ninguno de los argumentos opcionales disponibles para el mandato. Estas sentencias alteran temporalmente las posibles instrucciones de instalación de STRPTL asociadas a la PRPQ de 5799-PTL System i Tools para Developers.

Inhabilitar iceWM

Iniciar el servidor VNC crea o modifica un archivo script existente denominado xstartup_java que contiene el mandato para ejecutar iceWM. El archivo script xstartup_java reside en el siguiente directorio del sistema de archivos integrado:

```
/home/VNCprofile/.vnc/
```

donde *VNCprofile* es el nombre del perfil que ha iniciado el servidor VNC.

Para inhabilitar iceWM por completo, utilice un editor de texto para poner la línea como comentario o eliminarla del script que inicia iceWM. Para poner la línea como comentario, inserte un signo almohadilla (#) al principio de la línea.

Utilizar un VNCviewer o un navegador Web:

Para ejecutar una aplicación que presente una interfaz gráfica de usuario (GUI) en un System i, debe utilizar un visor VNC o un navegador Web para establecer conexión con el servidor de informática de redes virtuales (VNC). El visor VNC o el navegador Web se deben ejecutar en una plataforma con capacidad para gráficos, como un PC.

Nota: Para realizar los pasos que siguen, debe saber cuál es el número de pantalla y la contraseña de VNC. Al iniciar el servidor de informática de redes virtuales (VNC) se determina el valor del número de pantalla. Al crear un archivo de contraseñas de VNC se establece la contraseña de VNC.

Utilizar un VNCviewer para acceder al servidor VNC

Para utilizar un VNCviewer para conectarse al servidor VNC, complete los pasos siguientes:

1. Descargar e instalar la aplicación VNCviewer:
   - Los visores VNC están disponibles para la mayoría de las plataformas en el sitio Web RealVNC.
2. Inicie el VNCviewer que ha bajado. En el indicador de solicitud, entre el nombre de sistema y el número de pantalla y pulse **Aceptar**.
3. En el indicador de solicitud de contraseña, teclee la contraseña de VNC para obtener acceso a la pantalla del servidor VNC.

Utilizar un navegador Web para acceder al servidor VNC

Para utilizar un navegador Web para conectarse al servidor VNC, complete los pasos siguientes:

1. Inicie el navegador y acceda al siguiente URL:

   ```
   http://systemname:58nn
   ```

   donde:
   - *systemname* es el nombre o dirección IP del sistema que ejecuta el servidor VNC
   - *nn* es la representación de 2 dígitos del número de pantalla del servidor VNC

   Por ejemplo, cuando el nombre de sistema es `system_one` y el número de pantalla es `2`, el URL es:

   ```
   http://system_one:5802
   ```
2. Al acceder al URL satisfactoriamente se visualiza un indicador para la contraseña del servidor VNC. En el indicador de solicitud de contraseña, teclee la contraseña de VNC para obtener acceso a la pantalla del servidor VNC.

Consejos para la utilización de VNC:

Utilice los mandatos de lenguaje de control (CL) de i5/OS para iniciar y detener un servidor VNC y para visualizar información sobre los servidores VNC en ejecución actualmente.

Iniciar un servidor de pantalla VNC desde un programa CL

El ejemplo siguiente presenta una forma de establecer la variable de entorno DISPLAY e iniciar VNC automáticamente utilizando mandatos de lenguaje de control (CL):

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
ADDENVVAR ENVVAR(DISPLAY) VALUE('nombresistema:n')
```

donde:

- *nombresistema* es el nombre de host o la dirección IP del sistema en el que se ejecuta VNC
- *n* es el valor numérico que representa el número de pantalla que desea iniciar

Nota: En el ejemplo se presupone que no está ejecutando la pantalla: *n* y que ha creado satisfactoriamente el archivo de contraseñas de VNC necesario. Para obtener más información sobre cómo crear un archivo de contraseña, vea: Crear un archivo de contraseña VNC

Detener un servidor de pantalla VNC desde un programa CL

El siguiente código muestra una manera de detener un servidor VNC desde un programa CL:

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' '-kill' ':n')
```

donde *n* es el valor numérico que representa el número de pantalla que desea terminar.

Buscar servidores de pantalla VNC en ejecución

Para determinar qué servidores VNC (si los hay) se están ejecutando en este momento en el sistema, siga estos pasos:

1. Desde una línea de mandatos de i5/OS, inicie una shell PASE:

```
CALL QP2TERM
```

2. Desde el indicador de la shell PASE, utilice el mandato ps de PASE para listar los servidores VNC:

```
ps gaxuw | grep Xvnc
```

La salida resultante de este mandato revelará qué servidores VNC están en ejecución, con el siguiente formato:

```
    john  418  0.9  0.0  5020  0  - A  Jan 31 222:26
    /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/Xvnc :1 -desktop X -httpd
  jane   96  0.2  0.0   384  0  - A  Jan 30 83:54
    /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/Xvnc :2 -desktop X -httpd
```

Donde:

- La primera columna es el perfil que ha iniciado el servidor.
- La segunda columna es el ID de proceso PASE del servidor.
- La información que empieza por */QOpensys/* es el mandato que ha iniciado el servidor VNC (incluidos los argumentos). El número de pantalla suele ser el primer elemento de la lista de argumentos del mandato Xvnc.

Nota: El proceso Xvnc, que aparece en los datos de salida del ejemplo anterior, es el nombre del programa servidor VNC real. Xvnc se inicia cuando se ejecuta el script vncserver_java, el cual prepara el entorno y los parámetros para Xvnc y, a continuación, inicia Xvnc.

Consejos para utilizar AWT con WebSphere Application Server:

Si tiene que ejecutar las aplicaciones basadas en WebSphere en modalidad de GUI completa, en lugar de en la modalidad acéfala, tenga en cuenta estos consejos para evitar que surjan problemas relacionados con la conexión entre WebSphere y el servidor VNC.

Garantizar las comunicaciones seguras

El servidor VNC emplea un método, llamado comprobación de autorización X, que garantiza las comunicaciones seguras entre el propio servidor y la aplicación que lo utiliza, como puede ser WebSphere.

El proceso de iniciar el servidor VNC crea un archivo .Xauthority que contiene información de clave cifrada. WebSphere Application Server, para poder acceder a VNC, **debe** tener acceso al (y debe utilizar el) mismo archivo .Xauthority que el que utiliza el servidor VNC.

Para logralo, siga uno de estos métodos:

Ejecutar WebSphere Application Server y VNC utilizando el mismo perfil

Si ambos, WebSphere Application Server y el servidor VNC que hay que utilizar, se inician mediante el mismo perfil de usuario, ambos utilizarán por defecto el mismo archivo .Xauthority. Para ello, deberá elegir entre iniciar el servidor VNC desde el usuario predeterminado de e WebSphere (que es QEJBSVR) o cambiar el usuario predeterminado de WebSphere para que sea el perfil empleado para iniciar el servidor VNC.

Para cambiar el perfil de usuario del servidor de aplicaciones desde el perfil de usuario predeterminado (QEJBSVR) a un perfil distinto, debe realizar las siguientes acciones:

1. Utilice la consola administrativa de WebSphere Application Server para cambiar la configuración del servidor de aplicaciones.
2. Utilice System i Navigator para habilitar el nuevo perfil.

Ejecutar WebSphere Application Server y VNC utilizando perfiles distintos

En este caso, el WebSphere Application Server se inicia mediante un perfil de usuario y el archivo .Xauthority es propiedad de un perfil de usuario distinto. Para que WebSphere Application Server pueda iniciar el servidor VNC, tendrá que seguir estos pasos:

1. Cree un nuevo archivo .Xauthority (o actualice un archivo .Xauthority ya existente) iniciando el servidor VNC desde el perfil de usuario deseado. Por ejemplo, en una línea de mandatos de lenguaje de control (CL) de i5/OS, teclee este mandato y pulse Intro:

   ```
   CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
   ```

   donde *n* es el número de pantalla (un valor numérico en el rango de 1 a 99).

   Nota: El archivo .Xauthority reside en el directorio del perfil bajo el que se ejecuta el servidor VNC.
2. Utilice los siguientes mandatos CL para otorgar al perfil bajo el que ejecuta WebSphere Application Server la autorización para leer el archivo .Xauthority:

   ```
   CHGAUT OBJ('/home') USER(WASprofile) DTAAUT(*RX)
   CHGAUT OBJ('/home/VNCprofile') USER(WASprofile) DTAAUT(*RX)
   CHGAUT OBJ('/home/VNCprofile/.Xauthority') USER(WASprofile) DTAAUT(*R)
   ```

   Aquí, *VNCprofile* y *WASprofile* son los perfiles adecuados bajo los que ejecuta el servidor VNC y WebSphere Application Server.

   Nota: Solo debe seguir estos pasos cuando los perfiles *VNCprofile* y *WASprofile* sean diferentes. Si sigue estos pasos cuando los perfiles *VNCprofile* y *WASprofile* sean iguales, puede provocar que VCN no funcione correctamente.
3. En la consola administrativa de WebSphere Application Server, defina las variables de entorno DISPLAY y XAUTHORITY para su aplicación:
   - Para DISPLAY, utilice: system:n o localhost:n

     donde *system* es el nombre o la dirección IP del sistema y *n* es el número de pantalla que ha utilizado para iniciar el servidor VNC.
   - Para XAUTHORITY, utilice: /home/VNCprofile/.Xauthority

     donde *VNCprofile* es el perfil que ha iniciado el servidor VNC.
4. Active los cambios de configuración reiniciando WebSphere Application Server.

   WebSphere Application Server para i5/OS

   Gestionar usuarios y grupos con Management Central

Variables de entorno i5/OS PASE relacionadas con AWT:

Si se propone ejecutar la JVM clásica de IBM y utilizar la API de AWT, necesitará el entorno i5/OS PASE, que se inicia automáticamente. Sin embargo, en función de sus requisitos, tal vez tenga que establecer variables de entorno.

Esta información no le concierne cuando se utiliza la JVM de tecnología IBM para Java, porque ya se está ejecutando en el entorno PASE.

QIBM_JAVA_PASE_STARTUP
Si se especifica *java.awt.headless=true* o *os400.awt.native=true* en una invocación de la JVM clásica de IBM, el entorno i5/OS PASE de inicia automáticamente, pero lo hace siempre en modalidad de 32 bites. Si su aplicación exige que el entorno PASE se ejecute en la modalidad de 64 bits, debe establecer que la variable de entorno QIBM_JAVA_PASE_STARTUP tenga el valor */usr/lib/start64* antes de invocar Java. El valor predeterminado es */usr/lib/start32*.

QIBM_JAVA_PASE_ALLOW_PREV
Por defecto, la JVM clásica espera iniciar ella misma el entorno i5/OS PASE si es necesario, y fallará con un mensaje de error si el entorno PASE ya está activo. Si la aplicación necesita que

AWT utilice un entorno PASE iniciado con anterioridad (quizá debido a que la aplicación Java se inicia desde dentro de una shell QP2TERM), antes de invocar Java, hay que establecer que la variable de entorno QIBM_JAVA_PASE_ALLOW_PREV tenga el valor *1*, que indica que hay que usar el entorno PASE existente.

PASE_THREAD_ATTACH
Esta variable de entorno indica al entorno i5/OS PASE que las hebras adicionales iniciadas en el proceso (a menudo por la JVM) se tienen que conectar implícitamente al entorno PASE para así tener acceso a los métodos nativos de AWT, etcétera. PASE_THREAD_ATTACH se establece automáticamente mediante el mandato Java en un indicador de mandatos i5/OS o en las shells QSH o QP2TERM. Si la aplicación crea directamente una JVM desde un programa C/C++ utilizando la interfaz de invocación JNI, y las hebras secundarias creadas por la JVM tienen que acceder a PASE (por ejemplo, los métodos nativos de AWT), habrá que establecer que la variable de entorno PASE_THREAD_ATTACH tenga el valor *Y*. De lo contrario, se produciría un UnsatisfiedLinkError u otros errores similares.

Conceptos relacionados

"Variables de entorno Java de i5/OS PASE" en la página 221
La máquina virtual Java (JVM) utiliza las siguientes variables para iniciar los entornos i5/OS PASE. Debe establecer la variable QIBM_JAVA_PASE_STARTUP para poder ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java.

Verificar la configuración de AWT:

Puede verificar la configuración de AWT ejecutando un programa de prueba Java.

Para ejecutar el programa de prueba desde una línea de mandatos de i5/OS, teclee uno de los siguientes mandatos, en función de la modalidad que desea someter a prueba:

```
JAVA CLASS(NAWTtest) CLASSPATH('/QIBM/ProdData/Java400') PROP((os400.awt.native true))
```

O bien

```
JAVA CLASS (NAWTtest) CLASSPATH('/QIBM/ProdData/Java400') PROP((java.awt.headless true))
```

El programa de prueba crea una imagen codificada en JPEG y la guarda en la siguiente vía de acceso en el sistema de archivos integrado:

```
/tmp/NAWTtest.jpg
```

Cuando lo haya ejecutado, asegúrese de que el programa de prueba ha creado el archivo sin producir excepciones Java. Para visualizar la imagen, utilice la modalidad binaria para subir el archivo de imagen a un sistema con capacidad para gráficos y visualizarlo con un navegador, con un programa de pintura o con otra herramienta similar.

Seguridad Java

Este tema proporciona detalles sobre la autorización adoptada y describe cómo se puede utilizar SSL para hacer que las corrientes por sockets sean seguras en la aplicación Java.

Las aplicaciones Java están sujetas a las mismas restricciones de seguridad que cualquier otro programa de la plataforma System i5. Para ejecutar un programa Java en un System i5, debe poseer autorización sobre el archivo de clase en el sistema de archivos integrado. El programa, una vez iniciado, se ejecuta con la autorización del usuario.

En los releases anteriores a V6R1, se podía utilizar la autorización adoptada para acceder a los objetos con la autorización del usuario que ejecuta el programa y con la autorización del propietario del programa. La autorización adoptada otorga temporalmente a un usuario autorización sobre objetos a los

que, en principio, no habría tenido permiso para acceder. Vea la información sobre el mandato Crear programa Java (CRTJVAPGM) para obtener los detalles de los dos parámetros de la autorización adoptada, que son USRPRF y USEADPAUT.

En la V6R1, necesitará una solicitud de oferta de precio para programación (PRPQ) especial, que es un programa producto IBM construido de forma personalizada, para seguir utilizando la autorización adoptada en las aplicaciones Java. El los futuros releases, se retirará el soporte de autorización adoptada. Si desea más información sobre la autorización adoptada, sobre cómo obtener la PRPQ y sobre cómo realizar la transición de las aplicaciones para que dejen la autorización adoptada, vea: Cambios realizados en la autorización adoptada en V6R1.

La mayoría de los programas Java que se ejecutan en un System i5 son aplicaciones, no applets, por lo que el modelo de seguridad de tipo "cajón de arena" no representa ninguna restricción para ellas.

Nota: Para J2SDK, versión 1.4 y versiones posteriores, JAAS, JCE, JGSS y JSSE forman parte del JDK básico y no se consideran extensiones. En las versiones anteriores de JDK, estos elementos de seguridad eran extensiones.

Descripción del mandato CL Crear programa Java (CRTJVAPGM)

"Cambios realizados en la autorización adoptada en V6R1"
El soporte de la autorización adoptada por perfil de usuario mediante programas Java se retira en V6R1. En este tema se explica cómo determinar si las aplicaciones emplean la autorización adoptada y cómo modificarlas para que se ajusten a este cambio.

Cambios realizados en la autorización adoptada en V6R1

El soporte de la autorización adoptada por perfil de usuario mediante programas Java se retira en V6R1. En este tema se explica cómo determinar si las aplicaciones emplean la autorización adoptada y cómo modificarlas para que se ajusten a este cambio.

En V6R1, las aplicaciones Java ya no podrán adoptar la autorización del perfil de usuario mediante programas Java, a menos que se instale la solicitud de oferta de precio para programación (PRPQ) 5799-AAJ de IBM Adopt Authority for Java for i5/OS. El soporte de autorización adoptada Java y 5799-AAJ se retirarán en un futuro release. Por lo tanto, debe utilizar V6R1 como release de transición de la autorización adoptada Java. Le animamos encarecidamente a que elimine todas las dependencias de la autorización adoptada Java lo más pronto posible.

En estos temas se describen algunas situaciones comunes en las que se utiliza la autorización adoptada Java y se explica cómo se pueden modificar las aplicaciones Java para quitar la dependencia de la autorización adoptada Java. Estas modificaciones permitirán que las aplicaciones Java afectadas sigan funcionando como cabría esperar en los futuros releases. Estos cambios permitirán asimismo que las aplicaciones Java afectadas funcionen como cabe esperar en el release actual sin tener instalada la PRPQ 5799-AAJ.

Cómo averiguar si las aplicaciones emplean la autorización adoptada

En los releases V5R3 y V5R4, puede utilizar una herramienta que le ayude a determinar si tiene aplicaciones Java que quedarán afectadas por los cambios realizados en la autorización adoptada. La herramienta funciona con la JVM para informar de los programas Java que la JVM ha anotado como que utilizan autorización adoptada. Esta herramienta se adquiere por separado de la PRPQ 5799-AAJ y está disponible mediante los siguientes arreglos PTF:

V5R3

La herramienta se proporciona en el PTF SI27769. En la V5R3, la prestación de anotaciones de la JVM la proporciona el PTF de grupo Java SF99269, nivel 15, y está limitada a los programas Java que emplean el JDK 5.0.

V5R4

La herramienta se proporciona en el PTF SI27772. En la V5R4, la prestación de anotaciones de la JVM está habilitada para todos los JDK y no se necesitan más arreglos PTF.

Esta herramienta también sirve de ayuda para identificar las aplicaciones Java que podrían estar basadas en la autorización adoptada, explorando el sistema en busca de programas Java creados con el soporte de autorización adoptada.

Por defecto, esta herramienta visualiza los usos de autorización adoptada anotados por la JVM y también explora todo el sistema. Sin embargo, también admite varias opciones de Qshell:

```
Uso: /qsys.lib/qjava.lib/qjvaadptl.pgm [opción]...
        Las opciones válidas son
          -h             : mostrar esta sentencia de uso.
          -o <archivo>   : escribir datos de salida en el archivo especificado.
          -d <directorio>: explorar solo el árbol de directorio especificado.
          -noscan        : no explorar el sistema. Informar solo de los usos anotados.
```

Los datos de salida de la herramienta pueden ayudarle a determinar qué aplicaciones Java del sistema utilizan autorización adoptada. Mediante esta información, es posible que deba hacer lo siguiente:

- Si el uso está en el código que ha adquirido, póngase en contacto con el distribuidor para averiguar qué planes tienen en relación con la autorización adoptada.
- Si el uso está en su propio código, lea las soluciones indicadas en este documento y mire a ver si le interesa y puede modificar el código para que deje de utilizar la autorización adoptada.
- Si, después de considerar las alternativas, piensa que necesitará utilizar la autorización adoptada en V6R1, póngase en contacto con el representante del servicio técnico de IBM. Se le pondrá en contacto con el Rochester Development Lab, y se le consultará sobre procedimientos para llevar a cabo su tarea sin utilizar la autorización adoptada. Si es necesario, se le dará instrucciones sobre cómo obtener 5799-AAJ de IBM Adopt Authority for Java for i5/OS.

Uso de la autorización adoptada

Dado que la autorización adoptada solo resulta de utilidad para realizar operaciones en los objetos de i5/OS, así como en los registros de base de datos, o para acceder a métodos nativos, los ejemplos de este temario se centrarán en estas áreas. Para obtener una explicación básica de la autorización adoptada, vea: Objetos que adoptan la autorización del propietario, en el tema de consulta: Seguridad.

Mediante la autorización adoptada, un método podría adoptar la autorización del propietario del programa que se esté ejecutando, en lugar de adoptar la autorización de la persona que ejecuta el programa para llevar a cabo alguna operación. Desde el punto de vista conceptual, esto es muy similar a `Set UID` y `Set GID` en UNIX, y el ejemplo de uso canónico es `Change Password` tal como se ha construido en UNIX. Si bien no es razonable que cada usuario tenga autorización para cambiar el archivo de contraseña, resulta conviente fiarse del programa y permitirle que cambie la contraseña del usuario.

System i podía proporcionar la característica de autorización adoptada para los programas Java porque la JVM formaba parte de la base informática de confianza (TCB) del código interno bajo licencia del sistema (SLIC). Como i5/OS va hacia una implementación Java donde la JVM es un programa a nivel de usuario, Java no puede seguir proporcionando esta característica.

Para poder lograr la misma función, el enfoque más común será añadir al programa Java un método nativo ILE que adopte una autorización de perfil de usuario equivalente y realice la operación necesaria. También sería posible lograr el mismo efecto que la autorización adoptada sin añadir métodos nativos si se ejecuta la función en un proceso aparte que tenga más autorización, y se envían las peticiones a ese programa según se necesite.

Atributos de la adopción

Cada invocación de método se representa mediante un marco de la pila de tiempo de ejecución. En el caso de los métodos Java de ejecución directa (DE) y los métodos nativos, el marco de pila puede estar correlacionado con un objeto programa de i5/OS que identifique los atributos de adopción de la invocación. Los atributos son:

Adoptar perfil de usuario
Esto se corresponde con la opción USRPRF(*USER/*OWNER) de los mandatos CRTJVAPGM, CRTPGM y CRTSRVPGM. El valor *USER indica que los métodos del programa **no** adoptan las autorizaciones del perfil de usuario propietario en tiempo de ejecución. El valor *OWNER indica que los métodos del programa sí adoptan las autorizaciones del perfil de usuario propietario en tiempo de ejecución.

En los diagramas que siguen, los marcos de pila que adoptan las autorizaciones del perfil de usuario propietario se indican con un fondo sombreado.

Cada marco de pila se puede describir como que **adopta** o **no adopta** la autorización del perfil de usuario propietario.

Permitir/descartar autorización adoptada
Esto se corresponde con la opción USEADPAUT(*YES/*NO) en los mandatos CRTJVAPGM, CHGPGM y CHGSRVPGM. El valor *YES indica que la invocación del marco de pila permitirá que las autorizaciones adoptadas del perfil de usuario de los marcos de pila anteriores se utilicen en la invocación actual. El valor *NO quiere decir que se descarta la autorización adoptada de las invocaciones anteriores.

La autorización adoptada, cuando se descarta de las invocaciones anteriores, no se utiliza en la invocación actual ni en ninguno de los métodos a los que se llame en el marco actual.

Cada marco de pila se puede describir como que **permite** o **descarta** la autorización adoptada por las invocaciones anteriores de la pila.

Objetos que adoptan la autorización del propietario

Ejemplos: alternativas de la autorización adoptada

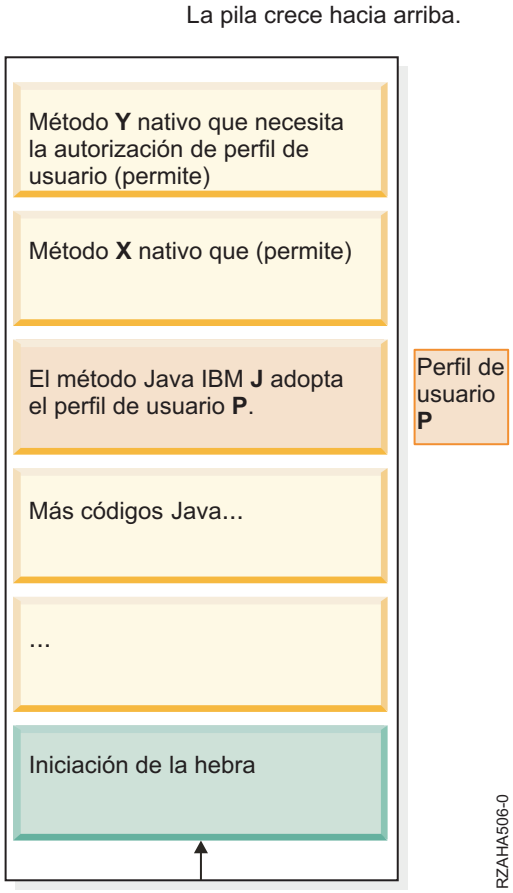
En este tema figuran algunos ejemplos del uso de la autorización adoptada Java y de algunas alternativas sugeridas. En las alternativas se emplean métodos nativos de los programas de servicio ILE para que adopten la autorización de manera parecida a los ejemplos de Java.

Existen otras posibles alternativas que pueden ser preferibles en determinadas circunstancias. Una posibilidad consiste en intercambiar el perfil de usuario del proceso para que adquiera autorización adicional. En este tema no se explica cómo intercambiar el perfil de usuario, que conlleva su propio conjunto de dificultades y riesgos. Los ejemplos incluidos en este temario harán una demostración de dos situaciones comunes en las que se utiliza la autorización adoptada Java y se ofrecen posibles alternativas. En este documento se da por sentado que no se tiene instalada la PRPQ 5799-AAJ.

- "Ejemplo 1: un método Java adopta autorización justo antes de llamar a un método nativo" en la página 261
- "Alternativa 1A: volver a empaquetar el método nativo X" en la página 263
- "Alternativa 1B: método nativo N nuevo" en la página 265
- "Ejemplo 2: un método Java adopta autorización y llama a otros métodos Java antes de llamar a un método nativo" en la página 267
- "Alternativa 2: método nativo N nuevo" en la página 269
- "Mandatos de compilación para los ejemplos" en la página 271

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

Ejemplo 1: un método Java adopta autorización justo antes de llamar a un método nativo



En este ejemplo, el método IBM Java **J** está contenido en un programa Java, el cual adopta el perfil de usuario **P** y llama directamente al método nativo **X**. El método nativo **X** llama al método ILE **Y**, que necesita la autorización adoptada.

JA61Example1.java

```
public class JA61Example1 {
    public static void main(String args[]) {
 int returnVal = J();
 if (returnVal > 0) {
     System.out.println("Autorización adoptada satisfactoriamente.");
 }
 else {
     System.out.println("ERROR: No se puede adoptar autorización.");
 }
    }

    static int J() {
 return X();
    }

    // Devuelve: 1 si puede acceder satisfactoriamente a *DTAARA JADOPT61/DATAAREA
    static native int X();

    static {
 System.loadLibrary("EX1");
    }
}
```

JA61Example1.h

```
/* NO EDITE ESTE ARCHIVO - está generado por máquina */
#include <jni.h>
/* Cabecera de la clase JA61Example1 */

#ifndef _Included_JA61Example1
#define _Included_JA61Example1
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Clase:     JA61Example1
 * Método:    X
 * Signatura: ()I
 */
JNIEXPORT jint JNICALL Java_JA61Example1_X
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

JA61Example1.c

```
/* Contiene el código fuente del método nativo Java_JA61Example1_X. Este
   módulo se enlaza al programa de servicio JADOPT61/EX1. */

#include "JA61Example1.h"
#include "JA61ModuleY.h"

/*
 * Clase:     JA61Example1
 * Método:    X
 * Signatura: ()I
 */
JNIEXPORT jint JNICALL Java_JA61Example1_X(JNIEnv* env, jclass klass) {
   return methodY();
}
```

JA61ModuleY.h

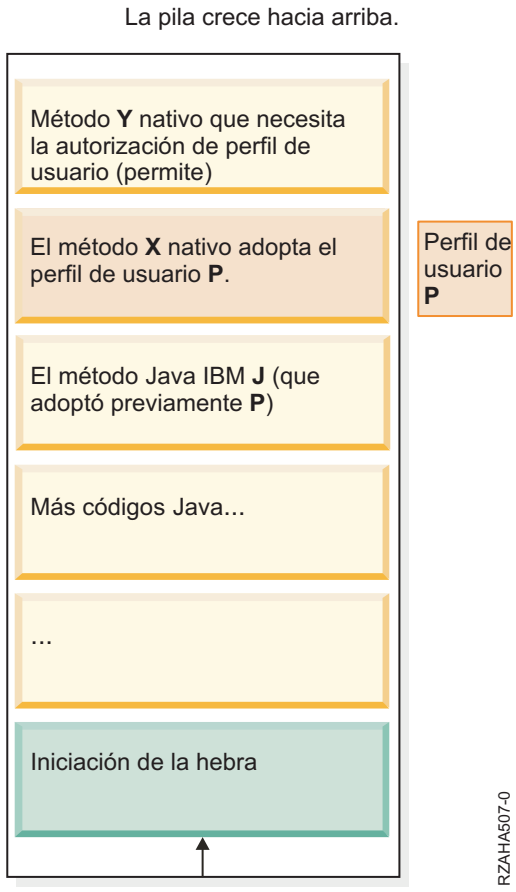
```
/* Este método intenta cambiar *DTAARA JADOPT61/DATAAREA. Esto
   solo será posible si se ha adoptado el perfil de usuario JADOPT61UP. */
int methodY(void);
```

JA61ModuleY.c

```
#include <except.h>
#include <stdio.h>
#include "JA61ModuleY.h"
#include <xxdtaa.h>

#define START 1
#define LENGTH 8


/* Este método intenta operar en *DTAARA JADOPT61/DATAAREA. Esto
 solo será posible si se ha adoptado el perfil de usuario JADOPT61UP. */
int methodY(void) {
   int returnValue;
   volatile int com_area;
   char newdata[LENGTH] = "new data";
   _DTAA_NAME_T dtaname = {"DATAAREA  ", "JADOPT61  "};

   /* Supervisar por si se produce una excepción en este rango */
#pragma exception_handler(ChangeFailed, 0, _C1_ALL, _C2_MH_ESCAPE)
```

```
    /* cambiar la *DTAARA JADOPT61/DATAAREA */
    QXXCHGDA(dtaname, START, LENGTH, newdata);
#pragma disable_handler

ChangeCompleted:
    printf("Área de datos actualizada satisfactoriamente\n");
    returnValue = 1;
    goto TestComplete;

ChangeFailed:      /* Aquí se controla si se produce una excepción */
    printf("Se ha obtenido una excepción.\n");
    returnValue = 0;

TestComplete:
    printf("methodY completed\n");
    return returnValue;
}
```

Alternativa 1A: volver a empaquetar el método nativo X



Una manera de conservar la adopción de la autorización consiste en separar el método nativo **X** colocándolo en un programa de servicio nuevo. Luego, ese nuevo programa de servicio puede adoptar el perfil de usuario **P**, adoptado anteriormente por el método Java **J**.

JA61Alternative1A.java

```
public class JA61Alternative1A {
    public static void main(String args[]) {
 int returnVal = J();
 if (returnVal > 0) {
```

```
|       System.out.println("Autorización adoptada satisfactoriamente.");
|   }
|   else {
|       System.out.println("ERROR: No se puede adoptar autorización.");
|   }
|      }
|
|      static int J() {
|   return X();
|      }
|
|      // Devuelve: 1 si puede acceder satisfactoriamente a *DTAARA JADOPT61/DATAAREA
|      static native int X();
|
|      static {
|   System.loadLibrary("ALT1A");
|      }
|  }
```

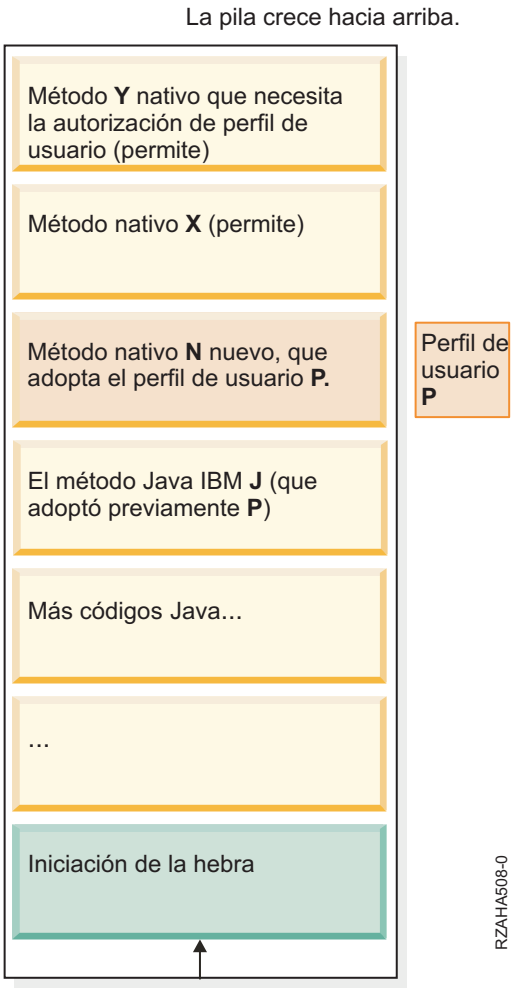
JA61Alternative1A.h

```
|  /* NO EDITE ESTE ARCHIVO - está generado por máquina */
|  #include <jni.h>
|  /* Cabecera de la clase JA61Alternative1A */
|
|  #ifndef _Included_JA61Alternative1A
|  #define _Included_JA61Alternative1A
|  #ifdef __cplusplus
|  extern "C" {
|  #endif
|  /*
|   * Clase:     JA61Alternative1A
|   * Método:    X
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative1A_X
|    (JNIEnv *, jclass);
|
|  #ifdef __cplusplus
|  }
|  #endif
|  #endif
```

JA61Alternative1A.c

```
|  /* Contiene el código fuente del método nativo Java_JA61Alternative1A_X. Este
|     módulo se enlaza al programa de servicio JADOPT61/ALT1A.*/
|
|  #include "JA61Alternative1A.h"
|  #include "JA61ModuleY.h"
|
|  /*
|   * Clase:     JA61Alternative1A
|   * Método:    X
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative1A_X(JNIEnv* env, jclass klass) {
|     return methodY();
|  }
```

Alternativa 1B: método nativo N nuevo



Otra manera de conservar la adopción de perfil de usuario consiste en crear un método nativo **N** completamente nuevo contenido en un programa de servicio que adopte el perfil de usuario **P**. El método Java **J** llama a este nuevo método, el cual llama al método nativo **X**. El método Java **J** se tendrá que cambiar para que llame a **N** en lugar de a **X**, pero el método nativo **X** no se tendrá que cambiar ni volver a empaquetar.

JA61Alternative1B.java

```
public class JA61Alternative1B {
    public static void main(String args[]) {
 int returnVal = J();
 if (returnVal > 0) {
     System.out.println("Autorización adoptada satisfactoriamente.");
 }
 else {
     System.out.println("ERROR: No se puede adoptar autorización.");
 }
    }

    static int J() {
 return N();
    }

    // Devuelve: 1 si puede acceder satisfactoriamente a *DTAARA JADOPT61/DATAAREA
```

```
|      static native int N();
|
|      static {
|   System.loadLibrary("ALT1B");
|      }
|  }
```

JA61Alternative1B.h

```
|  /* NO EDITE ESTE ARCHIVO - está generado por máquina */
|  #include <jni.h>
|  /* Cabecera de la clase JA61Alternative1B */
|
|  #ifndef _Included_JA61Alternative1B
|  #define _Included_JA61Alternative1B
|  #ifdef __cplusplus
|  extern "C" {
|  #endif
|  /*
|   * Clase:     JA61Alternative1B
|   * Método:    N
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative1B_N
|    (JNIEnv *, jclass);
|
|  #ifdef __cplusplus
|  }
|  #endif
|  #endif
```

JA61Alternative1B.c

```
|  /* Contiene el código fuente del método nativo Java_JA61Alternative1B_N. Este
|     módulo se enlaza al programa de servicio JADOPT61/ALT1B. */
|
|  #include "JA61Alternative1B.h"
|  #include "JA61Example1.h"
|
|  /*
|   * Clase:     JA61Alternative1B
|   * Método:    N
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative1B_N(JNIEnv* env, jclass klass) {
|     return Java_JA61Example1_X(env, klass);  /* de JA61Example1.h */
|  }
```

Ejemplo 2: un método Java adopta autorización y llama a otros métodos Java antes de llamar a un método nativo

En este ejemplo, un método IBM Java **J1** se encuentra dentro de un programa Java que adopta el perfil de usuario **P**. **J1** llama al método Java **J2**, que llama a **J3**, que a su vez llama al método nativo **X**. El método nativo **X** llama al método ILE **Y**, que necesita la autorización adoptada.

JA61Example2.java

```
public class JA61Example2 {
    public static void main(String args[]) {
 int returnVal = J1();
 if (returnVal > 0) {
     System.out.println("Autorización adoptada satisfactoriamente.");
 }
 else {
     System.out.println("ERROR: No se puede adoptar autorización.");
 }
    }
```

```
|
|      static int J1() {
|   return JA61Example2Allow.J2();
|      }
|
|  }
```

JA61Example2Allow.java

```
|  public class JA61Example2Allow {
|      public static int J2() {
|   return J3();
|      }
|
|      static int J3() {
|   return X();
|      }
|
|      // Devuelve: 1 si puede acceder satisfactoriamente a *DTAARA JADOPT61/DATAAREA
|      static native int X();
|
|      static {
|   System.loadLibrary("EX2ALLOW");
|      }
|  }
```

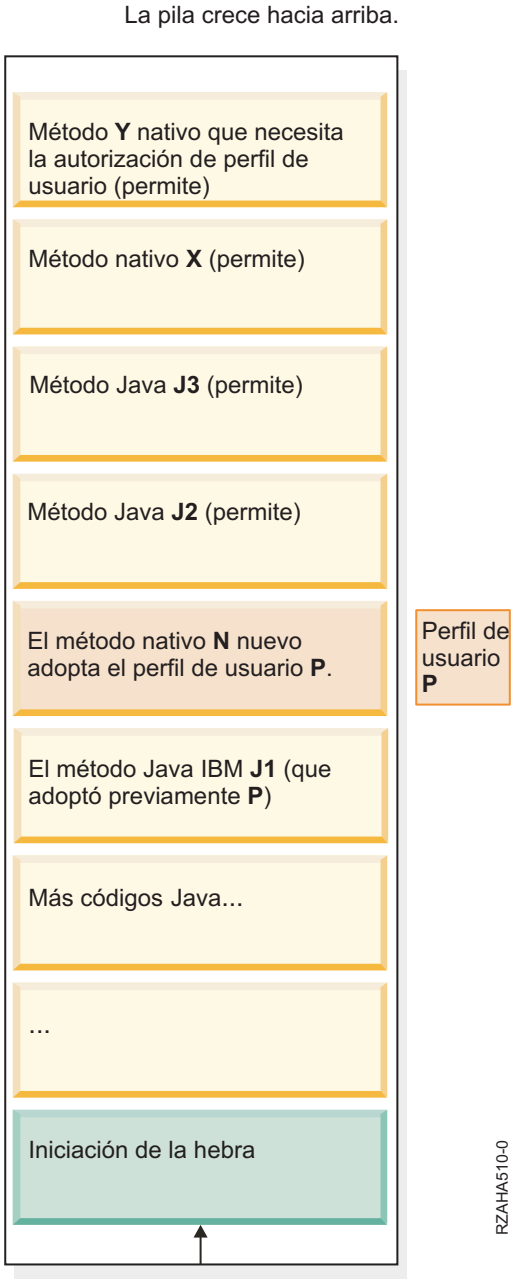
JA61Example2Allow.h

```
|  /* NO EDITE ESTE ARCHIVO - está generado por máquina */
|  #include <jni.h>
|  /* Cabecera de la clase JA61Example2Allow */
|
|  #ifndef _Included_JA61Example2Allow
|  #define _Included_JA61Example2Allow
|  #ifdef __cplusplus
|  extern "C" {
|  #endif
|  /*
|   * Clase:     JA61Example2Allow
|   * Método:    X
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Example2Allow_X
|    (JNIEnv *, jclass);
|
|  #ifdef __cplusplus
|  }
|  #endif
|  #endif
```

JA61Example2Allow.c

```
|  /* Contiene el código fuente del método nativo Java_JA61Example2Allow_X. Este
|     módulo se enlaza al programa de servicio JADOPT61/EX2ALLOW. */
|
|  #include "JA61Example2Allow.h"
|  #include "JA61ModuleY.h"
|
|  /*
|   * Clase:     JA61Example2Allow
|   * Método:    X
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Example2Allow_X(JNIEnv* env, jclass klass) {
|     return methodY();
|  }
```

Alternativa 2: método nativo N nuevo



Para poder conservar la autorización adoptada en este caso, se puede crear un nuevo método nativo **N**. Este método nativo se encuentra en un programa de servicio que adopta el perfil de usuario **P**.

Entonces el método nativo **N** utiliza JNI para llamar el método Java **J2**, que no presenta cambios. El método Java **J1** se tiene que cambiar para que llame al método nativo **N** en lugar de llamar al método Java **J2**.

JA61Alternative2.java

```
public class JA61Alternative2 {
    public static void main(String args[]) {
 int returnVal = J1();
```

```
|   if (returnVal > 0) {
|       System.out.println("Autorización adoptada satisfactoriamente.");
|   }
|   else {
|       System.out.println("ERROR: No se puede adoptar autorización.");
|   }
|      }
|
|      static native int N();
|
|      static int J1() {
|   return N();
|      }
|
|      static {
|   System.loadLibrary("ALT2");
|      }
|
|  }
```

JA61Alternative2.h

```
|  /* NO EDITE ESTE ARCHIVO - está generado por máquina */
|  #include <jni.h>
|  /* Cabecera de la clase JA61Alternative2 */
|
|  #ifndef _Included_JA61Alternative2
|  #define _Included_JA61Alternative2
|  #ifdef __cplusplus
|  extern "C" {
|  #endif
|  /*
|   * Clase:     JA61Alternative2
|   * Método:    N
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative2_N
|    (JNIEnv *, jclass);
|
|  #ifdef __cplusplus
|  }
|  #endif
|  #endif
```

JA61Alternative2.C

```
|  include "JA61Alternative2.h"
|
|  /*
|   * Clase:     JA61Alternative2
|   * Método:    N
|   * Signatura: ()I
|   */
|  JNIEXPORT jint JNICALL Java_JA61Alternative2_N(JNIEnv* env, jclass klass) {
|     #pragma convert (819)
|      char* className = "JA61Example2Allow";
|      char* methodName = "J2";
|      char* methodSig = "()I";
|  #pragma convert(0)
|
|      // Localizar la clase JA61Example2Allow
|      jclass cls = env->FindClass(className);
|      // Obtener el ID del método J2()I y llamarlo.
|      jmethodID methodID = env->GetStaticMethodID(cls, methodName, methodSig);
|      int result = env->CallStaticIntMethod(cls, methodID);
|      return result;
|  }
```

Mandatos de compilación para los ejemplos

Las siguientes instrucciones se basan en el código fuente situado en un directorio que se llama `/home/javatests/adoptup/v6r1mig` en una máquina System i.

En Qshell:

```
> cd /home/javatests/adoptup/v6r1mig
> javac -g *.java
```

Desde CL:

```
> CRTLIB JADOPT61
> CRTUSRPRF USRPRF(JADOPT61UP) STATUS(*DISABLED)
> CRTUSRPRF USRPRF(JADOPT61) PASSWORD(j61adopt) INLPGM(QSYS/QCMD) SPCAUT(*NONE)

> CRTDTAARA DTAARA(JADOPT61/DATAAREA) TYPE(*CHAR) LEN(50) VALUE('Valor inicial')
> GRTOBJAUT OBJ(JADOPT61/DATAAREA) OBJTYPE(*DTAARA) USER(JADOPT61UP) AUT(*ALL)
> RVKOBJAUT OBJ(JADOPT61/DATAAREA) OBJTYPE(*DTAARA) USER(*PUBLIC) AUT(*ALL)
> RVKOBJAUT OBJ(JADOPT61/DATAAREA) OBJTYPE(*DTAARA) USER(YOUR_USER_ID) AUT(*ALL)
```

Crear SRVPGMY, que se utiliza en todos los ejemplos:

```
> CRTCMOD MODULE(JADOPT61/MODULEY) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61ModuleY.c')
DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/SRVPGMY) MODULE(JADOPT61/MODULEY) EXPORT(*ALL)
```

Crear "Ejemplo 1: un método Java adopta autorización justo antes de llamar a un método nativo" en la página 261:

```
> CRTCMOD MODULE(JADOPT61/EX1) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Example1.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/EX1) EXPORT(*ALL) BNDSRVPGM(JADOPT61/SRVPGMY)
> QSH CMD('chown JADOPT61UP /home/javatests/adoptup/v6r1mig/JA61Example1.class')
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Example1.class') USRPRF(*OWNER)
```

Crear "Alternativa 1A: volver a empaquetar el método nativo X" en la página 263:

```
> CRTCMOD MODULE(JADOPT61/ALT1A) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Alternative1A.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/ALT1A) EXPORT(*ALL) BNDSRVPGM(JADOPT61/SRVPGMY) USRPRF(*OWNER)
> CHGOBJOWN OBJ(JADOPT61/ALT1A) OBJTYPE(*SRVPGM) NEWOWN(JADOPT61UP)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Alternative1A.class')
```

Crear "Alternativa 1B: método nativo N nuevo" en la página 265:

```
> CRTCMOD MODULE(JADOPT61/ALT1B) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Alternative1B.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/ALT1B) EXPORT(*ALL) BNDSRVPGM(JADOPT61/EX1) USRPRF(*OWNER)
> CHGOBJOWN OBJ(JADOPT61/ALT1B) OBJTYPE(*SRVPGM) NEWOWN(JADOPT61UP)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Alternative1B.class')
```

Crear "Ejemplo 2: un método Java adopta autorización y llama a otros métodos Java antes de llamar a un método nativo" en la página 267

```
> CRTCMOD MODULE(JADOPT61/EX2ALLOW) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Example2Allow.c')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/EX2ALLOW) EXPORT(*ALL) BNDSRVPGM(JADOPT61/SRVPGMY)
> QSH CMD('chown JADOPT61UP /home/javatests/adoptup/v6r1mig/JA61Example2.class')
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Example2.class') USRPRF(*OWNER)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Example2Allow.class') USEADPAUT(*YES)
```

Crear "Alternativa 2: método nativo N nuevo" en la página 269:

```
> CRTCPPMOD MODULE(JADOPT61/ALT2) SRCSTMF('/home/javatests/adoptup/v6r1mig/JA61Alternative2.C')
INCDIR('/home/javatests/adoptup/v6r1mig') DBGVIEW(*ALL)
> CRTSRVPGM SRVPGM(JADOPT61/ALT2) EXPORT(*ALL) USRPRF(*OWNER)
> CHGOBJOWN OBJ(JADOPT61/ALT2) OBJTYPE(*SRVPGM) NEWOWN(JADOPT61UP)
> CRTJVAPGM CLSF('/home/javatests/adoptup/v6r1mig/JA61Alternative2.class')
```

Para ejecutar los ejemplos, siga estos pasos:

```
> inicie sesión como JADOPT61
> ADDLIBLE JADOPT61
> ADDENVVAR ENVVAR(CLASSPATH) VALUE('/home/javatests/adoptup/v6r1mig')
> JAVA JA61Example1
> JAVA JA61Alternative1A
> JAVA JA61Alternative1B
> JAVA JA61Example2
> JAVA JA61Alternative2
```

Modelo de seguridad Java

Puede descargar applets Java desde cualquier sistema; por lo tanto, en la máquina virtual Java (JVM) existen mecanismos de seguridad para proteger contra los applets malignos. El sistema Java de tiempo de ejecución verifica los bytecodes a medida que la máquina virtual Java los va cargando. Ello garantiza que son bytecodes válidos y que el código no viola ninguna de las restricciones que la máquina virtual Java impone a los applets Java.

Al igual que sucede con los applets, el cargador y el verificador de bytecodes comprueban que los bytecodes sean válidos y que los tipos de datos se utilicen correctamente. También comprueban que se acceda correctamente a los registros y a la memoria y que la pila no sufra desbordamientos ni subdesbordamientos. Estas comprobaciones garantizan que la máquina virtual Java puede ejecutar con total seguridad la clase sin poner en peligro la integridad del sistema.

Las restricciones afectan a las operaciones que pueden realizar los applets Java, a la forma en que estos pueden acceder a la memoria y a la manera en que pueden utilizar la máquina virtual Java. Se imponen restricciones con el fin de impedir que un applet Java pueda acceder al sistema operativo subyacente o a los datos del sistema. Este modelo de seguridad se denomina ″cajón de arena″, pues según este modelo los applets Java solo tienen libertad para ″jugar″ en el cajón de arena reservado para ellos.

El modelo de seguridad ″cajón de arena″ es una combinación formada por el cargador de clases, el verificador de archivos de clase y la clase java.lang.SecurityManager.


Seguridad, de Sun Microsystems, Inc.

Proteger aplicaciones con SSL

Extensión de criptografía Java (JCE)

La extensión de criptografía Java (JCE) 1.2 es una extensión estándar de la plataforma Java 2, Standard Edition (J2SE). La implementación de JCE en System i es compatible con la implementación de Sun Microsystems, Inc. Esta documentación cubre los aspectos exclusivos de la implementación de System i.

Para poder entender esta información, debe estar familiarizado con la documentación general de las extensiones JCE. En la documentación de JCE de Sun hallará más información sobre las extensiones JCE.

El proveedor de JCE de IBM proveedor un generador de números aleatorios.

Existe también un proveedor de JCE de IBMJCEFIPS. Este proveedor se ha validado y se ha observado que está en conformidad con los estándares federales de proceso de la información (FIPS) 140-2, ″Requisitos de seguridad para los módulos criptográficos.″

El proveedor de JCE de IBMJCEFIPS da soporte a los siguientes algoritmos:

Tabla 11. Algoritmos soportados por el proveedor JCE de IBMJCEFIPS

Algoritmos de firma	Algoritmos de cifra	Códigos de autenticación de mensaje (MAC)	Algoritmos digest de mensaje
SHA1withDSA SHA1withRSA	AES TripleDES RSA	HmacSHA1	MD5 SHA-1 SHA-256 SHA-384 SHA-512

El proveedor JCE de IBMJCEFIPS también da soporte al algoritmo IBMSecureRandom para la generación de números aleatorios.

Para utilizar IBMJCEFIPS, necesita añadir un enlace simbólico al directorio de extensiones emitiendo el siguiente mandato:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjcefips.jar')
NEWLNK(< su directorio de extensiones >)
```

También debe añadir el proveedor a la lista de proveedores añadiendo una entrada en el archivo java.security file (por ejemplo, `security.provider.4=com.ibm.crypto.fips.provider.IBMJCEFIPS`), o utilizando el método Security.addProvider().

Utilizar la criptografía por hardware

La implementación de IBMJCECAI5OS amplía la extensión de criptografía Java (JCE) y la arquitectura de criptografía Java (JCA) para añadir la capacidad de utilizar criptografía por hardware por medio de las interfaces de la arquitectura criptográfica común (CCA) de IBM.

El proveedor de IBMJCECAI5OS saca partido de la criptografía por hardware en la arquitectura JCE existente y proporciona a los programadores de Java 2 las notables ventajas de seguridad y rendimiento de la criptografía por hardware con un mínimo de cambios en las aplicaciones Java existentes. Dado que de las complejidades de la criptografía por hardware se encarga la JCE normal, resulta fácil disponer de seguridad y rendimiento avanzados al utilizar dispositivos criptográfico de hardware. El proveedor de IBMJCECCAI5OS se adhiere a la infraestructura JCE de la misma manera que los proveedores actuales. Para las peticiones por hardware, se invocan las API de CCA por medio de nuevos métodos nativos. El proveedor de IBMJCECCAI5OS almacena las etiquetas de claves RSA de CCA en un nuevo tipo de almacén de claves Java, llamado JCECCAI5OSKS.

Requisitos de la criptografía por hardware

Para utilizar la criptografía por hardware, debe tener instalados los siguientes productos en el sistema:

- Un coprocesador criptográfico modelo 4764
- IBM i5/OS (5761-SS1) Opción 35 - Proveedor de servicio criptográfico (CSP) de CCA
- Oferta de programa bajo licencia (LPO) 5733-CY1 - IBM eServer iSeries Cryptographic Device Manager

Características del proveedor de criptografía por hardware de IBM

El proveedor de IBMJCECCAI5OS admite los siguientes algoritmos:

Tabla 12. Algoritmos soportados por el proveedor de IBMJCECCAI5OS

Algoritmos de firma	Algoritmos de cifrado	Códigos de autenticación de mensaje (MAC)	Algoritmos digest de mensaje
SHA1withRSA MD2WithRSA MD5WithRSA	RSA	HmacMD2 HmacMD5 HmacSHA1	MD2 MD5 SHA-1

El proveedor de IBMJCECCAI5OS también incluye un potente generador de números seudoaleatorios (PRNG), que permite la generación de claves por medio de fábricas de claves, y la generación de claves/certificados y la gestión de claves/certificados por medio de una aplicación `keytool`.

El proveedor de acceso criptográfico por hardware está disponible mediante la aplicación `hwkeytool`.

Nota: El proveedor de IBMJCECCAI5OS no se puede añadir a la JVM con los métodos insertProviderAt() y addProviderAt().

Propiedades del sistema de criptografía

Para trabajar con dispositivos criptográficos, puede utilizar las siguientes propiedades del sistema:

i5os.crypto.device
Especifica el dispositivo criptográfico que hay que utilizar. Si no se establece esta propiedad, se emplea el dispositivo predeterminado CRP01.

i5os.crypto.keystore
Especifica el archivo de almacén de claves CCA que hay que utilizar. Si no se establece esta propiedad, se emplea el archivo de almacén de claves mencionado en la descripción de dispositivo criptográfico.

Coprocesador criptográfico 4764

"Aplicación Java hwkeytool" en la página 424
La aplicación `hwkeytool` le permite utilizar las prestaciones de criptografía del coprocesador criptográfico modelo 4764 con la extensión de criptografía Java (JCE) y la arquitectura de criptografía Java (JCA).

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Pares de claves y su utilización por hardware:

En el entorno de criptografía por hardware, se puede utilizar el coprocesador criptográfico con dos tipos de pares de claves, RETAINED y conjunto de datos de clave pública (PKDS).

Cualquier aplicación puede utilizar ambos tipos de pares de claves por hardware soportados por el proveedor IBMJCECAI5OS. Los dos tipos de pares de claves, RETAINED y PKDS, devuelven una etiqueta, que el proveedor IBMJCECAI5OS trata como una clave. Podrá elegir el mecanismo por el que la aplicación almacenará la etiqueta.

El proveedor IBMJCECAI5OS almacena las claves RSA en un archivo de almacén de claves de arquitectura criptográfica común (CCA), cifrado bajo una clave maestra que se aloja en el coprocesador criptográfico 4764. El almacén de claves JCECCAI5OSKS almacena las etiquetas de los registros de clave en el almacén de claves CCA.

Pares de claves por hardware RETAINED

La implementación de criptografía más segura soportada por el proveedor IBMJCECAI5OS consiste en almacenar las claves en el dispositivo criptográfico de hardware real y no permitir que se recupere o vea nunca la clave privada (que es la parte confidencial del par de claves). Por eso se le llama par de claves *retenido*, ya que la clave privada queda retenida en el dispositivo de hardware y nunca está permitido verla ni recuperarla como texto sin cifrar. Lo que se devuelve a la aplicación o lo que se almacena en el almacén de claves al generarse el par de claves es solo una referencia a la clave privada, es decir, una *etiqueta*.

Cuando se necesita la clave, la petición se envía a la tarjeta de hardware en la está retenida la clave, junto con la etiqueta de la clave. La operación criptográfica se lleva a cabo en esa tarjeta utilizando la clave retenida, y luego se devuelve el resultado. Las claves retenidas son el tipo de clave más seguro. El inconveniente de utilizar claves retenidas es que no se puede hacer copia de seguridad de ellas ni se pueden recuperar. Si la tarjeta falla, las claves se pierden.

Pares de claves por hardware de conjunto de datos de clave pública (PKDS)

La otra opción para utilizar claves RSA es mediante el par de claves PKDS. Cuando se genera este tipo de par de claves, la clave privada se cifra con la clave maestra del coprocesador, para que nunca exista la posibilidad de ver ni recuperar la versión en texto sin cifrar de esta clave. El par de claves se almacena en un archivo de base de datos DB2. Lo que se devuelve a la aplicación o lo que se almacena en el almacén de claves al generarse el par de claves es solo una referencia a la clave privada, es decir, una *etiqueta*. Existe una manera de hacer copia de seguridad de las claves y de recuperarlas si la tarjeta falla, y consiste en almacenar el par de claves en un archivo.

Extensión de sockets seguros Java (JSSE)

La extensión de sockets seguros Java (JSSE) es como una infraestructura que abstrae los mecanismos subyacentes de la capa de sockets segura (SSL) y de la seguridad de capa de transporte (TLS). Al abstraer la complejidad y las peculiaridades de los protocolos subyacentes, JSSE permite a los programadores utilizar comunicaciones cifradas seguras al tiempo que se minimizan las posibles vulneraciones de seguridad. La extensión de sockets seguros Java (JSSE) utiliza ambos protocolos, SSL y TLS, para proporcionar comunicaciones cifradas seguras entre los clientes y los sevidores.

SSL/TLS proporciona los medios para autenticar un servidor y un cliente con el fin de ofrecer privacidad e integridad de datos. Todas las comunicaciones SSL/TLS comienzan con un "establecimiento de enlace" entre el servidor y el cliente. Durante el establecimiento de enlace, SSL/TLS negocia la suite de cifrado que el cliente y el servidor utilizarán para comunicarse entre sí. La suite de cifrado es una combinación de diversas funciones de seguridad disponibles a través de SSL/TLS.

Para mejorar la seguridad de la aplicación, JSSE hace lo siguiente:

- Protege los datos de comunicación mediante cifrado.
- Autentica los ID de usuarios remotos.
- Autentica los nombres de sistemas remotos.

Nota: JSSE utiliza un certificado digital para cifrar la comunicación por sockets de la aplicación Java. Los certificados digitales son un estándar de Internet para identificar aplicaciones, usuarios y sistemas seguros. Puede controlar los certificados digitales mediante IBM Digital Certificate Manager (el gestor de certificados digitales, DCM). Para obtener más información, vea: IBM Digital Certificate Manager.

Para conseguir que la aplicación Java sea más segura con JSSE:

- Prepare el System i5 para que soporte JSSE.
- Diseñe la aplicación Java para que utilice JSSE; para ello:
  - Si todavía no utiliza fábricas de sockets, cambie el código de sockets Java para que utilice las fábricas de sockets.

– Cambie el código Java para que utilice JSSE.
- Utilice un certificado digital para hacer que la aplicación Java sea más segura; para ello:
  1. Seleccione un tipo de certificado digital que deba utilizarse.
  2. Utilice el certificado digital al ejecutar la aplicación.

También puede registrar la aplicación Java como aplicación segura mediante la API QsyRegisterAppForCertUse.

Preparar el sistema para el soporte de capa de sockets segura (SSL)

Para preparar el System i5 con el fin de que utilice la capa de sockets segura (SSL), tendrá que instalar el programa bajo licencia (LP) Digital Certificate Manager (Gestor de certificados digitales o DCM).

Instale el LP Digital Certificate Manager, 5761-SS1 i5/OS - Digital Certificate Manager.

Además, debe asegurarse de que puede acceder a un certificado digital, o bien crear uno, en el sistema.

Información relacionada

Digital Certificate Manager

Cambiar el código Java para que utilice fábricas de sockets

Para utilizar la capa de sockets segura (SSL) con el código existente, primero debe cambiar el código de forma que utilice fábricas de sockets.

Para cambiar el código de forma que utilice fábricas de sockets, lleve a cabo los siguientes pasos:

1. Añada la línea siguiente al programa con el fin de importar la clase SocketFactory:
   ```
   import javax.net.*;
   ```
2. Añada una línea que declare una instancia de un objeto SocketFactory. Por ejemplo:
   ```
   SocketFactory socketFactory
   ```
3. Inicialice la instancia de SocketFactory estableciendo que sea igual al método SocketFactory.getDefault(). Por ejemplo:
   ```
   socketFactory = SocketFactory.getDefault();
   ```
   La declaración completa de SocketFactory debe ser así:
   ```
   SocketFactory socketFactory = SocketFactory.getDefault();
   ```
4. Inicialice los sockets existentes. Llame al método createSocket(host,port) de SocketFactory en la fábrica de sockets para cada socket que declare.

   Las declaraciones de sockets deben ser así:
   ```
   Socket s = socketFactory.createSocket(host,port);
   ```

   Donde:
   - *s* es el socket que se está creando.
   - *socketFactory* es la instancia de SocketFactory creada en el paso 2.
   - *host* es una variable de tipo serie que representa el nombre de un servidor de hospedaje.
   - *port* es una variable de tipo entero que representa el número de puerto de la conexión por socket.

Una vez realizados todos los pasos anteriores, el código utilizará fábricas de sockets. No hay que hacer más cambios en el código. Todos los métodos a los que se llama y la sintaxis con sockets seguirán funcionando.

Ejemplos: cambiar el código Java para que utilice fábricas de sockets de servidor:

Estos ejemplos muestran cómo cambiar una clase de socket simple, denominada simpleSocketServer, de forma que utilice fábricas de sockets para crear todos los sockets. El primer ejemplo muestra la clase

simpleSocketServer sin fábricas de sockets. El segundo muestra la clase simpleSocketServer con fábricas de sockets. En el segundo ejemplo, simpleSocketServer cambia de nombre y pasa a llamarse factorySocketServer.

Ejemplo 1: programa Socket Server sin fábricas de sockets

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/* Archivo 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    ServerSocket    serverSocket =
      new ServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Este servidor tan solo se hace eco de lo que se le envía...

    byte buffer[] = new byte[4096];

    int bytesRead;

    // Leer hasta que se devuelva "eof".
    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead); // Escribir lo recibido.
      os.flush();    // Vaciar la memoria intermedia de salida.
    }


    s.close();
    serverSocket.close();
  }// Finalizar main().

}       // Finalizar la definición de clase.
```

Ejemplo 2: programa Simple Socket Server con fábricas de sockets

```
/* Archivo factorySocketServer.java */

// Hay que importar javax.net para tomar la clase 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    // Cambiar la clase simpleSocketServer original para que utilice una
    // ServerSocketFactory con el fin de crear sockets de servidor.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ahora, la fábrica ha de crear el socket de servidor. Es el último
    // cambio que se realiza en el programa original.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Este servidor tan solo se hace eco de lo que se le envía...

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }


    s.close();
    serverSocket.close();
  }

}
```

Ejemplos: cambiar el código Java para que utilice fábricas de sockets de cliente:

Estos ejemplos muestran cómo cambiar una clase de socket simple, denominada simpleSocketClient, de forma que utilice fábricas de sockets para crear todos los sockets. El primer ejemplo muestra la clase simpleSocketClient sin fábricas de sockets. El segundo muestra la clase simpleSocketClient con fábricas de sockets. En el segundo ejemplo, simpleSocketClient cambia de nombre y pasa a llamarse factorySocketClient.

Ejemplo 1: programa Socket Client sin fábricas de sockets

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
/* Programa Simple Socket Client */

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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Conectando a host " + args[0] + " en el puerto " +
                       serverPort);

    // Crear el socket y conectar con el servidor.
    Socket  s = new Socket(args[0], serverPort);
    .
    .
    .

    // El resto del programa sigue a partir de aquí.
```

Ejemplo 2: programa Simple Socket Client con fábricas de sockets

```
/* Programa Simple Socket Factory Client */

// Observe que se importa javax.net.* para tomar la clase 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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Conectando a host " + args[0] + " en el puerto " +
                       serverPort);

    // Cambiar el programa simpleSocketClient original para crear una
    // fábrica de sockets y luego utilizarla para crear sockets.

    SocketFactory socketFactory = SocketFactory.getDefault();

    // Ahora, la fábrica crea el socket. Es el último cambio
    // que se realiza en el programa simpleSocketClient original.

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // El resto del programa sigue a partir de aquí.
```

Cambiar el código Java para que utilice la capa de sockets segura (SSL)

Si el código utiliza fábricas de sockets para crear los sockets, puede añadir el soporte de capa de sockets segura (SSL) al programa.

Si el código todavía no utiliza fábricas de sockets, vea: Cambiar el código Java para que utilice fábricas de sockets.

Para cambiar el código de forma que utilice SSL, lleve a cabo los siguientes pasos:

1. Importe javax.net.ssl.* para añadir el soporte SSL:
   ```
   import javax.net.ssl.*;
   ```
2. Declare una fábrica de sockets utilizando SSLSocketFactory para inicializarla:
   ```
   SocketFactory newSF = SSLSocketFactory.getDefault();
   ```
3. Utilice la nueva fábrica de sockets para inicializar los sockets de la misma manera que ha utilizado la anterior:
   ```
   Socket s = newSF.createSocket(args[0], serverPort);
   ```

Ahora el código ya utiliza el soporte SSL. No hay que hacer más cambios en el código.

Ejemplos: cambiar el servidor Java para que utilice la capa de sockets segura (SSL):

Estos ejemplos muestran cómo cambiar una clase, denominada factorySocketServer, de forma que utilice la capa de sockets segura (SSL).

El primer ejemplo muestra la clase factorySocketServer sin SSL. El segundo muestra la misma clase, que cambia de nombre y pasa a llamarse factorySSLSocketServer, con SSL.

Ejemplo 1: clase factorySocketServer simple sin soporte SSL

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/* Archivo factorySocketServer.java */
// Hay que importar javax.net para tomar la clase 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    // Cambiar la clase simpleSocketServer original para que utilice una
    // ServerSocketFactory con el fin de crear sockets de servidor.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ahora, la fábrica ha de crear el socket de servidor. Es el último
    // cambio que se realiza en el programa original.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());
```

```
    // Este servidor tan solo se hace eco de lo que se le envía.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

Ejemplo 2: clase factorySocketServer simple con soporte SSL

```
/* Archivo factorySocketServer.java */

// Hay que importar javax.net para tomar la clase 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    // Cambiar la clase simpleSocketServer original para que utilice una
    // ServerSocketFactory con el fin de crear sockets de servidor.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ahora, la fábrica ha de crear el socket de servidor. Es el último
    // cambio que se realiza en el programa original.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Este servidor tan solo se hace eco de lo que se le envía.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }
```

```
    s.close();
    serverSocket.close();
  }
}
```

Ejemplos: cambiar el cliente Java para que utilice la capa de sockets segura (SSL):

Estos ejemplos muestran cómo cambiar una clase, denominada factorySocketClient, de forma que utilice SSL (capa de sockets segura). El primer ejemplo muestra la clase factorySocketClient sin SSL. El segundo muestra la misma clase, que cambia de nombre y pasa a llamarse factorySSLSocketClient, con SSL.

Ejemplo 1: clase factorySocketClient simple sin soporte SSL

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/* Programa Simple Socket Factory Client */

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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Conectando a host " + args[0] + " en el puerto " +
                       serverPort);

    SocketFactory socketFactory = SocketFactory.getDefault();

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // El resto del programa sigue a partir de aquí.
```

Ejemplo 2: clase factorySocketClient simple con soporte SSL

```
// Observe que importamos javax.net.ssl.* para tomar el soporte 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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
```

```
  serverPort = new Integer(args[1]).intValue();

System.out.println("Conectando a host " + args[0] + " en el puerto " +
                   serverPort);

// Cambiar esto para crear una SSLSocketFactory en lugar de una SocketFactory.
SocketFactory socketFactory = SSLSocketFactory.getDefault();

// No es necesario cambiar nada más.
// ¡Ahí está la gracia de utilizar fábricas!
Socket  s = socketFactory.createSocket(args[0], serverPort);
.
.
.

// El resto del programa sigue a partir de aquí.
```

Seleccionar un certificado digital

A la hora de decidir cuál es el certificado digital que va a utilizar, debe tomar en consideración diversos factores. Se puede utilizar el certificado predeterminado del sistema o bien especificar otro certificado.

Le interesa utilizar el certificado predeterminado del sistema si:

- No tiene requisitos de seguridad específicos para la aplicación Java.
- Desconoce el tipo de seguridad que se necesita para la aplicación Java.
- El certificado predeterminado del sistema cumple los requisitos de seguridad de la aplicación Java.

Nota: Si decide que desea utilizar el certificado predeterminado del sistema, consulte con el administrador del sistema para asegurarse de que se ha creado un certificado predeterminado del sistema.

Si no desea utilizar el certificado predeterminado del sistema, tendrá que elegir otro certificado. Puede optar por dos tipos de certificados:

- Un **certificado de usuario**, que identifica al usuario de la aplicación.
- Un **certificado del sistema**, que identifica el sistema en el que se ejecuta la aplicación.

Debe utilizar un certificado de usuario si:

- la aplicación se ejecuta como aplicación cliente.
- desea que el certificado identifique al usuario que trabaja con la aplicación.

Debe utilizar un certificado de sistema si:

- la aplicación se ejecuta como aplicación de servidor.
- desea que el certificado identifique en qué sistema se ejecuta la aplicación.

Una vez que sepa cuál es el tipo de certificado que necesita, podrá elegir cualquiera de los certificados digitales que haya en los contenedores de certificados a los que pueda acceder.

Información relacionada

Digital Certificate Manager

Utilizar el certificado digital al ejecutar la aplicación Java

Para utilizar la capa de sockets segura (SSL), debe ejecutar la aplicación Java con un certificado digital.

Para especificar qué certificado hay que utilizar, emplee las siguientes propiedades:

- os400.certificateContainer
- os400.certificateLabel

Por ejemplo, si desea ejecutar la aplicación Java MyClass.class con el certificado digital MYCERTIFICATE y MYCERTIFICATE está en el contenedor YOURDCC, el mandato `java` sería como este:

```
java -Dos400.certificateContainer=YOURDCC
     -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

Si todavía no ha decido qué certificado digital desea utilizar, vea: “Seleccionar un certificado digital” en la página 283. También puede decidir que va a utilizar el certificado predeterminado del sistema, que está almacenado en el contenedor de certificados predeterminados del sistema.

Para utilizar el certificado digital predeterminado del sistema, no es necesario que especifique ningún certificado ni ningún contenedor de certificados en ninguna parte. La aplicación Java utilizará automáticamente el certificado digital predeterminado del sistema.

Los certificados digitales y la propiedad -os400.certificateLabel

Los certificados digitales son un estándar de Internet para identificar aplicaciones, usuarios y sistemas seguros. Los certificados digitales se almacenan en contenedores de certificados digitales. Si desea utilizar el certificado predeterminado de un contenedor de certificados digitales, no es necesario que especifique una etiqueta de certificado. Si desea utilizar un certificado digital determinado, es necesario que especifique la etiqueta de dicho certificado en el mandato `java` utilizando esta propiedad:

```
os400.certificateLabel=
```

Por ejemplo, si el nombre del certificado que desea utilizar es MYCERTIFICATE, el mandato `java` que entre será así:

```
java -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

En este ejemplo, la aplicación Java MyClass utiliza el certificado MYCERTIFICATE. MYCERTIFICATE debe estar en el contenedor de certificados predeterminado del sistema para que MyClass pueda utilizarlo.

Los contenedores de certificados digitales y la propiedad -os400.certificateContainer

Los contenedores de certificados digitales almacenan certificados digitales. Si desea utilizar el contenedor de certificados predeterminado del sistema System i5, no hace falta que especifique un contenedor de certificados. Si desea utilizar un determinado contenedor de certificados digitales, debe especificar el contenedor de certificados digitales en el mandato `java`, mediante esta propiedad:

```
os400.certificateContainer=
```

Por ejemplo, si el nombre del contenedor de certificados que contiene el certificado digital que desea utilizar se denomina MYDCC, el mandato `java` que debe entrar sería:

```
java -Dos400.certificateContainer=MYDCC MyClass
```

En este ejemplo, la aplicación Java MyClass.class se ejecuta en el sistema utilizando el certificado digital predeterminado que hay en el contenedor de certificados digitales MYDCC. Los sockets que cree en la aplicación utilizarían el certificado predeterminado de MYDCC para identificarse y hacer que las comunicaciones sean seguras.

Si deseara utilizar el certificado digital MYCERTIFICATE del contenedor de certificados digitales, entraría un mandato `java` como el siguiente:

```
java -Dos400.certificateContainer=MYDCC
          -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

Información relacionada

Digital Certificate Manager

Utilizar la extensión de sockets seguros Java 1.4

Esta información solo es válida cuando se utiliza JSSE en un System i5 que ejecute J2SDK, versión 1.4. JSSE es como una infraestructura que abstrae los mecanismos subyacentes de SSL y TLS. Mediante la abstracción de la complejidad y las peculiaridades de los protocolos subyacentes, JSSE permite a los programadores utilizar comunicaciones cifradas seguras al tiempo que se minimizan las posibles vulneraciones de seguridad. La extensión de sockets seguros Java utiliza ambos protocolos, el seguridad de capa de transporte (TLS) de capa de sockets segura (SSL) y el protocolo de seguridad de capa de transporte (TLS), para proporcionar comunicaciones cifradas seguras entre los clientes y los sevidores.

A la implementación IBM de JSSE se le llama IBM JSSE. IBM JSSE incluye un proveedor de JSSE nativo de System i5 y un proveedor de JSSE Java puro.

Configurar el sistema para que soporte JSSE 1.4:

Configure el sistema para que utilice IBM JSSE. Este tema incluye los requisitos de software, cómo cambiar los proveedores de JSSE y las propiedades de seguridad y las propiedades del sistema necesarias.

Cuando utiliza el Java 2 Software Development Kit (J2SDK), versión 1.4 o posterior, en el System i5, JSSE ya está configurado. En la configuración predeterminada se emplea el proveedor de JGSS nativo de System i5.

Cambiar los proveedores de JSSE

Puede configurar JSSE para que utilice el proveedor de JSSE Java puro en lugar del proveedor de JSSE nativo de System i5. Si cambia algunas propiedades específicas de seguridad JSSE y algunas propiedades Java del sistema, podrá conmutar entre los dos proveedores.

Gestores de seguridad

Si se propone ejecutar la aplicación JSSE teniendo habilitado un gestor de seguridad Java, es posible que tenga que establecer los permisos de red disponibles. Hallará más información en: Permisos SSL, en Permisos de Java 2 SDK.

Proveedores de JSSE 1.4:

En la extensión JSSE de IBM, hay un proveedor de JSSE nativo de System i5 y dos proveedores de JSSE Java puro. El proveedor que elija utilizar dependerá de las necesidades de la aplicación.

Los tres proveedores cumplen la especificación de la interfaz JSSE. Pueden comunicarse entre sí y con cualquier otra implementación de SSL o TLS, incluso con implementaciones que no sean Java.

Proveedor de JSSE Java puro

El proveedor de JSSE Java puro ofrece las siguientes características:

- Funciona con cualquier tipo de objeto KeyStore para controlar y configurar certificados digitales (por ejemplo, JKS, PKCS12, etc.).
- Le permite utilizar juntos cualquier combinación de componentes JSSE de diversas implementaciones.

El nombre del proveedor de la implementación Java puro es IBMJSSE. Debe pasar este nombre de proveedor, utilizando las mayúsculas y minúsculas correctas, al método java.security.Security.getProvider() o a los diversos métodos getInstance() para varias de las clases de JSSE.

Proveedor de JSSE FIPS 140-2 Java puro

El proveedor de JSSE FIPS 140-2 Java puro ofrece las siguientes características:
- Está en conformidad con los estándares federales de proceso de la información (FIPS) 140-2 para módulos criptográficos.
- Funciona con cualquier tipo de objeto KeyStore para controlar y configurar certificados digitales.

Nota: El proveedor de JSSE FIPS 140-2 Java puro no permite que se conecten a su implementación componentes de cualquier otra implementación.

El nombre del proveedor de JSSE FIPS 140-2 de la implementación Java puro es IBMJSSEFIPS. Debe pasar este nombre de proveedor, utilizando las mayúsculas y minúsculas correctas, al método java.security.Securirty.getProvider() o a los diversos métodos getInstance() para varias de las clases de JSSE.

Proveedor de JSSE nativo de System i5

El proveedor de JSSE nativo de System i5 ofrece las siguientes características:
- Utiliza el soporte SSL nativo del System i5.
- Permite usar el gestor de certificados digitales para configurar y controlar certificados digitales. Esto se proporciona por medio de un tipo de KeyStore (almacén de claves) exclusivo del System i5 (IbmISeriesKeyStore).
- Ofrece un rendimiento óptimo.
- Le permite utilizar juntos cualquier combinación de componentes JSSE de diversas implementaciones. Sin embargo, para conseguir el mejor rendimiento posible, utilice solo componentes JSSE nativos de System i5.

El nombre del proveedor de la implementación nativa de System i5 es IbmISeriesSslProvider. Debe pasar este nombre de proveedor, utilizando las mayúsculas y minúsculas correctas, al método java.security.Security.getProvider() o a los diversos métodos getInstance() para varias de las clases de JSSE.

Cambiar el proveedor de JSSE por omisión

Puede cambiar el proveedor de JSSE por omisión efectuando las modificaciones adecuadas en las propiedades de seguridad. Hallará más información en: Propiedades de la seguridad de JSSE.

Tras cambiar el proveedor de JSSE, compruebe que las propiedades del sistema especifican la configuración correcta para la información de certificados digitales (almacén de claves) que requiere el nuevo proveedor. Hallará más información en:, Propiedades Java del sistema.

Propiedades de la seguridad de JSSE 1.4:

La máquina virtual Java (JVM) emplea importantes propiedades de seguridad que se establecen editando el archivo de propiedades Java maestro de seguridad.

Este archivo, que se llama java.security, suele residir en el directorio /QIBM/ProdData/Java400/jdk14/lib/security directory del servidor iSeries.

La lista siguiente describe varias propiedades de seguridad relevantes para utilizar JSSE. Utilice las descripciones a modo de guía para editar el archivo java.security.

security.provider.<entero>

El proveedor de JSSE que desea utilizar. Estáticamente también registra las clases de proveedor criptográfico. Especifique los distintos proveedores de JSSE exactamente igual que en el ejemplo siguiente:

```
security.provider.5=com.ibm.as400.ibmonly.net.ssl.Provider
security.provider.6=com.ibm.jsse.IBMJSSEProvider
security.provider.7=com.ibm.fips.jsse.IBMJSSEFIPSProvider
```

ssl.KeyManagerFactory.algorithm

Especifica el algoritmo de KeyManagerFactory por omisión. En el caso del proveedor de JSSE nativo del iSeries, utilice:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

En el caso del proveedor de JSSE Java puro, utilice:

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

Hallará más información en el javadoc de javax.net.ssl.KeyManagerFactory.

ssl.TrustManagerFactory.algorithm

Especifica el algoritmo de TrustManagerFactory por omisión. En el caso del proveedor de JSSE nativo del iSeries, utilice:

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

En el caso del proveedor de JSSE Java puro, utilice:

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

Hallará más información en el javadoc de javax.net.ssl.TrustManagerFactory.

ssl.SocketFactory.provider

Especifica la fábrica de sockets SSL por omisión. En el caso del proveedor de JSSE nativo del iSeries, utilice:

```
ssl.SocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLSocketFactoryImpl
```

En el caso del proveedor de JSSE Java puro, utilice:

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
```

Hallará más información en el javadoc de javax.net.ssl.SSLSocketFactory.

ssl.ServerSocketFactory.provider

Especifica la fábrica de sockets de servidor SSL por omisión. En el caso del proveedor de JSSE nativo del iSeries, utilice:

```
ssl.ServerSocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLServerSocketFactoryImpl
```

En el caso del proveedor de JSSE Java puro, utilice:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

Hallará más información en el javadoc de javax.net.ssl.SSLServerSocketFactory.

Propiedades Java del sistema en JSSE 1.4:

Si desea emplear JSSE en las aplicaciones, debe especificar varias propiedades del sistema que los objetos SSLContext predeterminados necesitan para proporcionar una confirmación de la configuración. Algunas de las propiedades son válidas para ambos proveedores, mientras que otras solo son válidas para el proveedor nativo de System i5.

Cuando se utiliza el proveedor de JSSE nativo del System i5, si no especifica ninguna de las propiedades, os400.certificateContainer toma por defecto el valor *SYSTEM, lo que significa que JSSE utiliza la entrada predeterminada del almacén de certificados del sistema.

Propiedades que funcionan para ambos proveedores

Las propiedades siguientes son válidas para ambos proveedores JSSE. En cada descripción se indica la propiedad predeterminada, si procede.

javax.net.ssl.trustStore

El nombre del archivo que contiene el objeto KeyStore que desea que utilice el TrustManager predeterminado. El valor predeterminado es jssecacerts, o cacerts (si no existe jssecacerts).

javax.net.ssl.trustStoreType

El tipo de objeto KeyStore que desea que utilice el TrustManager predeterminado. El valor predeterminado es el valor devuelto por el método KeyStore.getDefaultType.

javax.net.ssl.trustStorePassword

La contraseña del objeto KeyStore que desea que utilice el TrustManager predeterminado.

javax.net.ssl.keyStore

El nombre del archivo que contiene el objeto KeyStore que desea que utilice el KeyManager predeterminado.

javax.net.ssl.keyStoreType

El tipo de objeto KeyStore que desea que utilice el KeyManager predeterminado. El valor predeterminado es el valor devuelto por el método KeyStore.getDefaultType.

javax.net.ssl.keyStorePassword

La contraseña del objeto KeyStore que desea que utilice el KeyManager predeterminado.

Propiedades que solo funcionan con el proveedor de JSSE nativo del System i5

Las propiedades que solo son válidas para el proveedor de JSSE nativo de System i5 son las siguientes:

os400.secureApplication

El identificador de la aplicación. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword

- javax.ssl.net.trustStoreType

os400.certificateContainer

El nombre del archivo de claves que desea utilizar. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

os400.certificateLabel

La etiqueta de archivo de claves que desea utilizar. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

Conceptos relacionados

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Información relacionada

Propiedades del sistema en el sitio Web de Java de Sun

Utilizar el proveedor de JSSE 1.4 nativo de System i5:

El proveedor de JSSE nativo de System i5 ofrece la suite completa de clases e interfaces JSSE, incluidas las implementaciones de la clase KeyStore y la clase SSLConfiguration de JSSE.

Para emplear eficazmente el proveedor nativo de System i5, utilice la información de este tema y vea también la información del Javadoc SSLConfiguration para JSSE 1.4.

Valores de protocolo para el método SSLContext.getInstance

En la tabla que sigue se identifican y describen los valores de protocolo del método SSLContext.getInstance del proveedor de JSSE nativo de System i5.

Los protocolos SSL soportados pueden estar limitados por los valores de sistema establecidos en su sistema. Encontrará más detalles en el subtema: Valores del sistema de seguridad: protocolos de la capa de sockets segura (SSL), en la información de gestión de sistemas.

Valor de protocolo	Protocolos SSL soportados
SSL	SSL versión 2, SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.
SSLv2	SSL versión 2
SSLv3	Protocolo SSL versión 3. Aceptará hello SSLv3 encapsulado en un hello de formato SSLv2.
TLS	SSL versión 2, SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.
TLSv1	Protocolo TLS versión 1, definido en la petición de comentarios (RFC) 2246. Aceptará hello TLSv1 encapsulado en un hello de formato SSLv2.
SSL_TLS	SSL versión 2, SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.

Implementación de KeyStore nativo de System i5

El proveedor nativo de System i5 ofrece una implementación de la clase KeyStore de tipo IbmISeriesKeyStore. Esta implementación de almacén de claves proporciona una envoltura alrededor del soporte de gestor de certificados digitales. El contenido del almacén de claves se basa en un identificador de aplicación específico o un archivo de claves, una contraseña y una etiqueta. JSSE carga las entradas del almacén de claves del gestor de certificados digitales. Para cargar las entradas, JSSE utiliza el identificador de aplicación adecuado o la información del archivo de claves cuando la aplicación intenta por primera vez acceder a las entradas del almacén de claves o la información del archivo de claves. No es posible modificar el almacén de claves y todos los cambios de configuración deben efectuarse mediante el Gestor de certificados digitales.

Recomendaciones al utilizar el proveedor nativo de System i5

A continuación figuran algunas recomendaciones para hacer que el proveedor nativo de System i5 se ejecute de la manera más eficaz posible.

- Para que el proveedor de JSSE nativo de System i5 funcione, la aplicación JSSE solo debe emplear componentes de la implementación nativa. Por ejemplo, la aplicación habilitada para JSSE nativo de System i5 no puede emplear un objeto X509KeyManager creado con el proveedor de JSSE Java puro para inicializar satisfactoriamente un objeto SSLContext creado con el proveedor de JSSE nativo de System i5.
- Además, hay que inicializar las implementaciones de X509KeyManager y X509TrustManager en el proveedor nativo de System i5, utilizando un objeto IbmISeriesKeyStore o un objeto com.ibm.as400.SSLConfiguration.

Nota: Las recomendaciones mencionadas pueden cambiar en futuros releases, con lo que el proveedor de JSSE nativo de System i5 podría permitirle que conectase componentes no nativos (por ejemplo, KeyStore de JKS o TrustManagerFactory de IbmX509).

Referencia relacionada

"Información Javadoc de SSLConfiguration para JSSE 1.4"

Información relacionada

Digital Certificate Manager

Valores del sistema de seguridad: protocolos de la capa de sockets segura (SSL)

Información Javadoc de SSLConfiguration para JSSE 1.4:

com.ibm.as400

Clase SSLConfiguration

```
java.lang.Object
  |
  +--com.ibm.as400.SSLConfiguration
```

Todas las interfaces implementadas:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

public final class **SSLConfiguration**
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable

Esta clase corresponde a la especificación de la configuración necesaria para la implementación JSSE nativa de System i5.

La implementación JSSE nativa de System i5 funciona de manera más eficaz con un objeto `KeyStore` de tipo ″IbmISeriesKeyStore″. Este tipo de objeto `KeyStore` contiene entradas de clave y entradas de certificado de confianza basadas ya sea en un identificador de aplicación registrado en el gestor de certificado digital (DCM) o en un archivo de conjunto de claves (contenedor de certificados digitales). Luego, un objeto `KeyStore` de este tipo puede servir para inicializar un objeto `X509KeyManager` y un objeto `X509TrustManager` del `Provider` ″IbmISeriesSslProvider″. Después, los objetos `X509KeyManager` y `X509TrustManager` se pueden usar para inicializar un objeto `SSLContext` del ″IbmISeriesSslProvider″. Entonces, el objeto `SSLContext` proporciona acceso a la implementación JSSE nativa de System i5 basándose en la información de configuración especificada para el objeto `KeyStore`. Cada vez que se realiza una carga en un `KeyStore` de ″IbmISeriesKeyStore″, el `KeyStore` se inicializa basándose en la configuración actual especificada por el identificador de la aplicación o el archivo de conjunto de claves.

Esta clase también puede utilizarse para generar un objeto `KeyStore` de cualquier tipo válido. El `KeyStore` se inicializa basándose en la configuración actual especificada por el identificador de la aplicación o el archivo de conjunto de claves. Para realizar cualquier cambio en la configuración especificada por un identificador de aplicación o por un archivo de conjunto de claves, es necesario regenerar el objeto `KeyStore`. Observe que se tiene que especificar un contraseña de conjunto de claves (para el almacén de certificados *SYSTEM cuando se utiliza el identificador de la aplicación) para poder crear con éxito otro tipo de `KeyStore` diferente a ″IbmISeriesKeyStore″. La contraseña de conjunto de claves debe especificarse para conseguir el acceso a cualquier clave privada de cualquier `KeyStore` de tipo ″IbmISeriesKeyStore″.

Desde:
SDK 1.4

Consulte también:
KeyStore, X509KeyManager, X509TrustManager, SSLContext

Resumen del constructor

`SSLConfiguration()` crea una nueva SSLConfiguration. Consulte “Detalles del constructor” en la página 292 para obtener más información.

Tabla 13. **Resumen de los métodos**

`void`	“clear” en la página 296() Borra toda la información en el objeto para que todos los métodos get devuelvan un valor nulo.
`java.lang.Object`	“clone” en la página 297() Genera una nueva copia de esta configuración SSL.

Tabla 13. **Resumen de los métodos** *(continuación)*

`boolean`	“equals” en la página 297(java.lang.Objectobj) Indica si algún otro objeto es ″igual a″ este.
`protected void`	“finalize” en la página 295() llamado por el recogedor de basura de un objeto cuando la recogida de basura determina que no hay más referencias al objeto.
`java.lang.String`	“getApplicationId” en la página 295() Devuelve el ID de la aplicación.
`java.lang.String`	“getKeyringLabel” en la página 295() Devuelve la etiqueta de conjunto de claves.
`java.lang.String`	“getKeyringName” en la página 295() Devuelve el nombre de conjunto de claves.
`char[]`	“getKeyringPassword” en la página 295() Devuelve la contraseña de conjunto de claves.
`java.security.KeyStore`	“getKeyStore” en la página 297(char[]password) Devuelve un almacén de claves de tipo ″IbmISeriesKeyStore″ utilizando la contraseña especificada.
`java.security.KeyStore`	“ getKeyStore” en la página 298(java.lang.Stringtype, char[]password) Devuelve un almacén de claves del tipo solicitado utilizando la contraseña especificada.
`int`	“hashCode” en la página 297() Devuelve un valor de código hash para el objeto.
`staticvoid`	(java.lang.String[]args) Ejecuta las funciones de SSLConfiguration.
`void`	(java.lang.String[]args, java.io.PrintStreamout) Ejecuta las funciones de SSLConfiguration.
`void`	“setApplicationId” en la página 296(java.lang.StringapplicationId) Establece el ID de la aplicación.
`void`	“setApplicationId” en la página 296(java.lang.StringapplicationId, char[]password) Establece el identificador de la aplicación y la contraseña del conjunto de claves.
`void`	“setKeyring” en la página 296(java.lang.Stringname,java.lang.Stringlabel, char[]password) Establece la información de conjunto de claves.

Métodos heredados de la clase java.lang.Object
`getClass, notify, notifyAll, toString, wait, wait, wait`

Detalles del constructor

SSLConfiguration

```
public SSLConfiguration()
```

Crea una nueva SSLConfiguration. El identificador de aplicación y la información del conjunto de claves se inicializa con valores de omisión.

El valor predeterminado para el identificador de aplicación es el valor especificado para la propiedad ″os400.secureApplication″.

Los valores predeterminados para la información de conjunto de claves son igual a nulo si se especifica la propiedad ″os400.secureApplication″. Si no se especifica la propiedad ″os400.secureApplication″, el valor

para el nombre del conjunto de claves es el valor especificado para la propiedad ″os400.certificateContainer″. Si no se especifica la propiedad ″os400.secureApplication″, la etiqueta de conjunto de claves se inicializa con el valor de la ″os400.certificateLabel″. Si no se establece la propiedad ″os400.secureApplication″ ni la ″os400.certificateContainer″, el nombre de conjunto de claves se inicializará con ″*SYSTEM″.

Detalles del método

---

main

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

Ejecuta las funciones de SSLConfiguration. Hay cuatro mandatos que se pueden ejecutar: -help, -create, -display y -update. El mandato debe ser el primer parámetro especificado.

Las opciones que se pueden especificar son las siguientes (en cualquier orden):

-keystore **nombre-archivo-almacén-claves**
Especifica el nombre del archivo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para todos los mandatos.

-storepass **contraseña-archivo-almacén-claves**
Especifica la contraseña asociada con el archivo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para todos los mandatos.

-storetype **tipo-almacén-claves**
Especifica el tipo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para cualquier mandato. Si no se especifica esta opción, se utiliza un valor de ″IbmISeriesKeyStore″.

-varname>-appid identificador-aplicación
Especifica el identificador de aplicación que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-keyring **nombre-archivo-conjunto-claves**
Especifica el nombre de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-keyringpass **contraseña-archivo-conjunto-claves**
Especifica la contraseña de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Puede especificarse esta opción para los mandatos *-create* y *-update* y es necesaria cuando se especifica un tipo de almacén de claves distinto de ″IbmISeriesKeyStore″. Si no se especifica esta opción, se utiliza la contraseña oculta de conjunto de claves.

-keyringlabel **etiqueta-archivo-conjunto-claves**
Especifica la etiqueta de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción solo se especifica cuando también se especifica la opción *-keyring*. Si no es especifica esta opción cuando se especifica la opción *keyring*, se utiliza la etiqueta por omisión en el conjunto de claves.

-systemdefault
Especifica el valor predeterminado del sistema que se utiliza para inicializar un almacén de claves

que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-v Especifica que se debe generar salida verbosa. Esta opción se puede especificar para cualquier mandato.

El mandato ayuda visualiza la información de uso para especificar los parámetros de este método. Los parámetros para invocar la función de ayuda se especifican de esta manera:

```
-help
```

El mandato create crea un nuevo archivo de almacén de datos. Existen tres variantes del mandato create. Una variante para crear un almacén de claves basado en un identificador de aplicación particular, otra variante para crear un almacén de claves basado en un nombre, una etiqueta y una contraseña de conjunto de claves, y la tercera variante para crear un conjunto de claves basado en la configuración por omisión del sistema.

Para crear un almacén de claves basado en un identificador de aplicación particular, se debe especificar la opción *-appid*. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en el identificador de aplicación ″APPID″:

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -appid APPID
```

Para crear un almacén de claves basado en un archivo de conjunto de claves particular, se debe especificar la opción *-keyring*. Las opciones *-keyringpass* y *keyringlabel* también pueden especificarse. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en el archivo de almacén de claves ″keyring.file″, basándose en la contraseña ″ringpass″ y en la etiqueta de conjunto de claves ″keylabel″:

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

Para crear un almacén de claves basado en la configuración por omisión del sistema, hay que especificar la opción *-systemdefault*. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en la configuración por omisión del sistema :

```
-create -keystore keystore.file -storepass keypass -systemdefault
```

El mandato update actualiza un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″. Existen tres variantes del mandato update que son idénticas a las variantes del mandato create. Las opciones del mandato update son idénticas a las opciones utilizadas para el mandato create. El mandato display visualiza la configuración especificada de un archivo de almacén de claves existente. Los siguientes parámetros visualizarían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″:

```
-display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

Parámetros:

`args` - los argumentos de la línea de mandatos

---

run

```
public void run(java.lang.String[]args,
                java.io.PrintStreamout)
```

Ejecuta las funciones de SSLConfiguration. Los parámetros y la funcionalidad de este método son idénticos a los del método main().

Parámetros:

`args` - los argumentos de mandato

`out` - la corriente de salida en que se deben escribir los resultados

Consulte también: `com.ibm.as400.SSLConfiguration.main()`

---

getApplicationId

```
public java.lang.String getApplicationId()
```

Devuelve el ID de la aplicación.

Devuelve:

el ID de la aplicación.

---

getKeyringName

```
public java.lang.String getKeyringName()
```

Devuelve el nombre de conjunto de claves.

Devuelve:

el nombre de conjunto de claves.

---

getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

Devuelve la etiqueta de conjunto de claves.

Devuelve:

la etiqueta de conjunto de claves.

---

getKeyringPassword

```
public final char[] getKeyringPassword()
```

Devuelve la contraseña de conjunto de claves.

Devuelve:

la contraseña de conjunto de claves.

---

finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Llamado por el recogedor de basura de un objeto cuando la recogida de basura determina que no hay más referencias al objeto.

Alteraciones temporales:
finalize en la clase java.lang.Object

Lanza: java.lang.Throwable - la excepción lanzada por este método.

---

clear

```
public void clear()
```

Borra toda la información en el objeto para que todos los métodos get devuelvan un valor nulo.

---

setKeyring

```
public void setKeyring(java.lang.Stringname,
                       java.lang.Stringlabel,
                       char[]password)
```

Establece la información de conjunto de claves.

Parámetros:
name - el nombre de conjunto de claves

label - la etiqueta de conjunto de claves, o el valor nulo si se utiliza la entrada de conjunto de claves por omisión.

password - la contraseña de conjunto de claves, o el valor nulo si se utiliza la contraseña oculta.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId)
```

Establece el ID de la aplicación.

Parámetros:
applicationId - el ID de la aplicación.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId,
                             char[]password)
```

Establece el ID de la aplicación y la contraseña del conjunto de claves. Al especificar la contraseña de conjunto de claves se permite cualquier conjunto de claves creado para permitir el acceso a la clave privada.

Parámetros:
applicationId - el ID de la aplicación.

password - la contraseña de conjunto de claves.

---

equals

```
public boolean equals(java.lang.Objectobj)
```

Indica si algún otro objeto es ″igual a″ este.

Alteraciones temporales:
equals de la clase java.lang.Object

Parámetros:
obj - objeto para comparar

Devuelve:
el indicador de si los objetos especifican la misma información de configuración

---

hashCode

```
public int hashCode()
```

Devuelve un valor de código hash para el objeto.

Alteraciones temporales:
hashCode de la clase java.lang.Object

Devuelve:
un valor de código hash para el objeto.

---

clone

```
public java.lang.Object clone()
```

Genera una nueva copia de esta configuración SSL. Los cambios posteriores en los componentes de esta configuración no afectarán a la copia nueva y viceversa.

Alteraciones temporales:
clone de la clase java.lang.Object

Devuelve:
una copia de esta configuración SSL

---

getKeyStore

```
public java.security.KeyStore getKeyStore(char[]password)
                                  throws java.security.KeyStoreException
```

Devuelve un almacén de claves de tipo ″IbmISeriesKeyStore″ utilizando la contraseña especificada. Se inicializa el almacén de claves basándose en la información de configuración actual almacenada en el objeto.

Parámetros:
password - se utiliza para inicializar el almacén de claves

Devuelve:
el almacén de claves KeyStore inicializado basándose en la información de configuración actual almacenada en el objeto.

Lanza: java.security.KeyStoreException - si no se puede crear el almacén de claves

---

getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                           char[]password)
                                    throws java.security.KeyStoreException
```

Devuelve un almacén de claves del tipo solicitado utilizando la contraseña especificada. Se inicializa el almacén de claves basándose en la información de configuración actual almacenada en el objeto.

Parámetros:

`type` - tipo de almacén de claves a devolver

`password` - se utiliza para inicializar el almacén de claves

Devuelve:

el almacén de claves KeyStore inicializado basándose en la información de configuración actual almacenada en el objeto.

Lanza: `java.security.KeyStoreException` - si no se puede crear el almacén de claves

Ejemplos: IBM Java Secure Sockets Extension 1.4:

Los ejemplos de JSSE muestran cómo pueden un cliente y un servidor emplear el proveedor de JSSE nativo de System i5 para crear un contexto que habilite las comunicaciones seguras.

Nota: Ambos ejemplos utilizan el proveedor de JSSE nativo de System i5, sean cuales sean las propiedades especificadas por el archivo java.security.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

Ejemplo: cliente SSL que utiliza un objeto SSLContext de la versión 1.4:

Este programa cliente de ejemplo utiliza un objeto SSLContext, que inicializa para emplear el ID de aplicación ″MY_CLIENT_APP″. Este programa utilizará la implementación nativa de System i5 con independencia de lo que se haya especificado en el archivo java.security.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
| ////////////////////////////////////////////////////////////////////////////
| //
| // Este programa cliente de ejemplo utiliza un objeto SSLContext, que inicializa
| // para emplear el ID de aplicación "MY_CLIENT_APP".
| //
| // El ejemplo utiliza el proveedor de JSSE nativo de iSeries, sean cuales sean
| // las propiedades especificadas por el archivo java.security.
| //
| // Sintaxis de mandato:
| //     java -Djava.version=1.4 SslClient
| //
| // Tenga en cuenta que "-Djava.version=1.4" es innecesario si se ha configurado
| // que hay que usar J2SDK versión 1.4 por defecto.
| //
| ////////////////////////////////////////////////////////////////////////////
|
| import java.io.*;
| import javax.net.ssl.*;
| import java.security.*;
| import com.ibm.as400.SSLConfiguration;
|
| /**
```

```
| * Programa cliente SSL.
| */
| public class SslClient {
|
|   /**
|    * Método main de SslClient.
|    *
|    * @param args argumentos de línea de mandatos (no se utiliza)
|    */
|   public static void main(String args[]) {
|       /*
|        * Configurar para capturar las excepciones lanzadas.
|        */
|       try {
|           /*
|            * Inicializar un objeto SSLConfiguration para especificar un ID de
|            * aplicación.  MY_CLIENT_APP se debe registrar y configurar
|            * correctamente con el Gestor de certificados digitales (DCM).
|            */
|           SSLConfiguration config = new SSLConfiguration();
|           config.setApplicationId("MY_CLIENT_APP");
|           /*
|            * Obtener un objeto KeyStore del objeto SSLConfiguration.
|            */
|           char[] password = "password".toCharArray();
|           KeyStore ks = config.getKeyStore(password);
|           /*
|            * Asignar e inicializar una KeyManagerFactory.
|            */
|           KeyManagerFactory kmf =
|                             KeyManagerFactory.getInstance("IbmISeriesX509");
|           kmf.init(ks, password);
|           /*
|            * Asignar e inicializar una TrustManagerFactory.
|            */
|           TrustManagerFactory tmf =
|                           TrustManagerFactory.getInstance("IbmISeriesX509");
|           tmf.init(ks);
|           /*
|            * Asignar e inicializar un SSLContext.
|            */
|           SSLContext c =
|                      SSLContext.getInstance("SSL", "IbmISeriesProvider");
|           c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
|           /*
|            * Obtener la SSLSocketFactory del SSLContext.
|            */
|           SSLSocketFactory sf = c.getSocketFactory();
|           /*
|            * Crear un SSLSocket.
|            *
|            * Cambiar la dirección IP codificada por programa por la dirección IP o el
|            * nombre de host del servidor.
|            */
|           SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
|           /*
|            * Enviar un mensaje al servidor mediante la sesión segura.
|            */
|           String sent = "Probar la escritura SSL java";
|           OutputStream os = s.getOutputStream();
|           os.write(sent.getBytes());
|           /*
|            * Escribir los resultados en la pantalla.
|            */
|           System.out.println("Escritos " + sent.length() + " bytes...");
|           System.out.println(sent);
|           /*
```

```
|              * Recibir un mensaje del servidor mediante la sesión segura.
|              */
|             InputStream is = s.getInputStream();
|             byte[] buffer = new byte[1024];
|             int bytesRead = is.read(buffer);
|             if (bytesRead == -1)
|                 throw new IOException("Recibido fin de archivo inesperado");
|             String received = new String(buffer, 0, bytesRead);
|             /*
|              * Escribir los resultados en la pantalla.
|              */
|             System.out.println("Leídos " + received.length() + " bytes...");
|             System.out.println(received);
|         } catch (Exception e) {
|             System.out.println("Excepción inesperada: "+;
|                                e.getMessage());
|             e.printStackTrace();
|         }
|     }
|
| }
```

Ejemplo: servidor SSL que utiliza un objeto SSLContext de la versión 1.4:

El siguiente programa servidor utiliza un objeto SSLContext que inicializa con un archivo de almacén de claves creado anteriormente.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////////////////////////////////////////
//
// El siguiente programa servidor utiliza un objeto SSLContext que
// inicializa con un archivo de almacén de claves creado anteriormente.
//
// El archivo de almacén de claves tiene el nombre y la contraseña siguientes:
//   Nombre de archivo: /home/keystore.file
//   Contraseña: password
//
// El programa de ejemplo necesita el archivo de almacén de claves para crear un
// objeto IbmISeriesKeyStore. El objeto KeyStore debe especificar MY_SERVER_APP como
// identificador de la aplicación.
//
// Para crear el archivo de almacén de claves, puede emplear el siguiente mandato de Qshell:
//
//   java com.ibm.as400.SSLConfiguration  -create  -keystore /home/keystore.file
//     -storepass password  -appid MY_SERVER_APP
//
// Sintaxis de mandato:
//     java -Djava.version=1.4 JavaSslServer
//
// Tenga en cuenta que "-Djava.version=1.4" es innecesario si se ha configurado
// que hay que usar J2SDK versión 1.4 por defecto.
//
// También puede crear el archivo de almacén de claves entrando este mandato en un indicador de mandatos de i5/OS:
//
//  RUNJVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
//      '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
/**
* Programa servidor Java SSL que utiliza el ID de aplicación.
```

```
*/
public class JavaSslServer {

   /**
    * Método main de JavaSslServer.
    *
    * @param args argumentos de línea de mandatos (no se utiliza)
    */
   public static void main(String args[]) {
       /*
        * Configurar para capturar las excepciones lanzadas.
        */
       try {
           /*
            * Asignar e inicializar un objeto KeyStore.
            */
           char[] password = "password".toCharArray();
           KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
           FileInputStream fis = new FileInputStream("/home/keystore.file");
           ks.load(fis, password);
           /*
            * Asignar e inicializar una KeyManagerFactory.
            */
           KeyManagerFactory kmf =
                            KeyManagerFactory.getInstance("IbmISeriesX509");
           kmf.init(ks, password);
           /*
            * Asignar e inicializar una TrustManagerFactory.
            */
           TrustManagerFactory tmf =
                           TrustManagerFactory.getInstance("IbmISeriesX509");
           tmf.init(ks);
           /*
            * Asignar e inicializar un SSLContext.
            */
           SSLContext c =
                      SSLContext.getInstance("SSL", "IbmISeriesProvider");
           c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
           /*
            * Obtener una SSLServerSocketFactory del SSLContext.
            */
           SSLServerSocketFactory sf = c.getServerSocketFactory();
           /*
            * Crear un SSLServerSocket.
            */
           SSLServerSocket ss =
               (SSLServerSocket) sf.createServerSocket(13333);
           /*
            * Efectuar un accept() para crear un SSLSocket.
            */
           SSLSocket s = (SSLSocket) ss.accept();
           /*
            * Recibir un mensaje del cliente mediante la sesión segura.
            */
           InputStream is = s.getInputStream();
           byte[] buffer = new byte[1024];
           int bytesRead = is.read(buffer);
           if (bytesRead == -1)
               throw new IOException("Recibido fin de archivo inesperado");
           String received = new String(buffer, 0, bytesRead);
           /*
            * Escribir los resultados en la pantalla.
            */
           System.out.println("Leídos " + received.length() + " bytes...");
           System.out.println(received);
           /*
            * Volver a enviar como un eco el mensaje al cliente mediante la sesión segura.
```

```
             */
            OutputStream os = s.getOutputStream();
            os.write(received.getBytes());
            /*
             * Escribir los resultados en la pantalla.
             */
            System.out.println("Escritos " + received.length() + " bytes...");
            System.out.println(received);
        } catch (Exception e) {
            System.out.println("Excepción inesperada: "+;
                               e.getMessage());
            e.printStackTrace();
        }
    }

}
```

Utilizar la extensión de sockets seguros Java 1.5

Esta información solo es válida cuando se utiliza JSSE en un sistema que ejecute la plataforma Java 2, Standard Edition (J2SE), versión 1.5 o posterior. JSSE es como una infraestructura que abstrae los mecanismos subyacentes de SSL y TLS. Mediante la abstracción de la complejidad y las peculiaridades de los protocolos subyacentes, JSSE permite a los programadores utilizar comunicaciones cifradas seguras al tiempo que se minimizan las posibles vulneraciones de seguridad. La extensión de sockets seguros Java utiliza ambos protocolos, el seguridad de capa de transporte (TLS) de capa de sockets segura (SSL) y el protocolo de seguridad de capa de transporte (TLS), para proporcionar comunicaciones cifradas seguras entre los clientes y los sevidores.

A la implementación IBM de JSSE se le llama IBM JSSE. En la extensión JSSE de IBM se incluye un proveedor de JSSE nativo de iSeries y un proveedor de JSSE Java puro de IBM. Además, la extensión JSSE de Sun Microsystems, Inc., venía inicialmente en el System i, y seguirá viniendo con él.

Configurar el servidor para que soporte JSSE 1.5:

Configure el System i5 para que utilice implementaciones distintas de JSSE. Este tema incluye los requisitos de software, cómo cambiar los proveedores de JSSE y las propiedades de seguridad y las propiedades del sistema necesarias. En la configuración predeterminada se emplea el proveedor de JSSE Java puro de IBM, conocido como IBMJSSE2.

Cuando utiliza la plataforma Java 2, Standard Edition (J2SE), versión 1.5, en el System i5, JSSE ya está configurado. En la configuración predeterminada se emplea el proveedor de JGSS nativo de System i5.

Cambiar los proveedores de JSSE

Puede configurar JSSE para que utilice el proveedor de JSSE nativo de System i5 o el proveedor de JSEE de Sun Microsystems, Inc., en lugar del proveedor de JSEE Java puro de IBM. Si cambia algunas propiedades específicas de seguridad JSSE y algunas propiedades Java del sistema, podrá conmutar entre proveedores.

Gestores de seguridad

Si se propone ejecutar la aplicación JSSE teniendo habilitado un gestor de seguridad Java, es posible que tenga que establecer los permisos de red disponibles. Hallará más información en: Permisos SSL, en Permisos de Java 2 SDK.

Proveedores de JSSE 1.5:

En la extensión JSEE de IBM hay un proveedor de JSSE nativo de System i5 y un proveedor de JSSE Java puro de IBM. El System i5 también viene con la extensión JSSE de Sun Microsystems, Inc. El proveedor que elija utilizar dependerá de las necesidades de la aplicación.

Todos los proveedores cuplen la especificación de la interfaz JSSE. Pueden comunicarse entre sí y con cualquier otra implementación de SSL o TLS, incluso con implementaciones que no sean Java.

Proveedor de JSSE Java puro de Sun Microsystems, Inc.

Esta es la implementación Java Sun de la extensión JSSE. Venía inicialmente en el System i, y seguirá viendo con él. Para obtener más información sobre la extensión JSSE de Sun Microsystems, Inc., vea el manual de consulta de la extensión de sockets seguros Java (JSSE) de Sun Microsystems, Inc.

Proveedor de JSSE Java puro de IBM

El proveedor de JSSE Java puro de IBM ofrece las siguientes características:

- Funciona con cualquier tipo de objeto KeyStore para controlar y configurar certificados digitales (por ejemplo, JKS, PKCS12, etc.).
- Le permite utilizar juntos cualquier combinación de componentes JSSE de diversas implementaciones.

El nombre del proveedor de la implementación Java puro es IBMJSSEProvider2. Debe pasar este nombre de proveedor, utilizando las mayúsculas y minúsculas correctas, al método java.security.Security.getProvider() o a los diversos métodos getInstance() para varias de las clases de JSSE.

Proveedor de JSSE nativo de System i5

El proveedor de JSSE nativo de System i5 ofrece las siguientes características:

- Utiliza el soporte SSL nativo del System i5.
- Permite usar el gestor de certificados digitales para configurar y controlar certificados digitales. Esto se proporciona por medio de un tipo de KeyStore (almacén de claves) exclusivo del System i5 (IbmISeriesKeyStore).
- Le permite utilizar juntos cualquier combinación de componentes JSSE de diversas implementaciones.

El nombre de proveedor de la implementación nativa de System i5 es IBMi5OSJSSEProvider. Debe pasar este nombre de proveedor, utilizando las mayúsculas y minúsculas correctas, al método java.security.Security.getProvider() o a los diversos métodos getInstance() para varias de las clases de JSSE.

Cambiar el proveedor de JSSE por omisión

Puede cambiar el proveedor de JSSE por omisión efectuando las modificaciones adecuadas en las propiedades de seguridad. Para obtener más información, consulte “Propiedades de la seguridad de JSSE 1.5”.

Tras cambiar el proveedor de JSSE, compruebe que las propiedades del sistema especifican la configuración correcta para la información de certificados digitales (almacén de claves) que requiere el nuevo proveedor. Para obtener más información, consulte “Propiedades Java del sistema en JSSE 1.5” en la página 305.

Propiedades de la seguridad de JSSE 1.5:

La máquina virtual Java (JVM) emplea importantes propiedades de seguridad que se establecen editando el archivo de propiedades Java maestro de seguridad.

Este archivo, que se llama java.security, suele residir en el directorio /QIBM/ProdData/Java400/jdk15/lib/security del servidor.

La lista siguiente describe varias propiedades de seguridad relevantes para utilizar JSSE. Utilice las descripciones a modo de guía para editar el archivo java.security.

security.provider.<entero>

El proveedor de JSSE que desea utilizar. Estáticamente también registra las clases de proveedor criptográfico. Especifique los distintos proveedores de JSSE exactamente igual que en el ejemplo siguiente:

```
security.provider.5=com.ibm.i5os.jsse.JSSEProvider
security.provider.6=com.ibm.jsse2.IBMJSSEProvider2
security.provider.7=com.sun.net.ssl.internal.ssl.Provider
```

ssl.KeyManagerFactory.algorithm

Especifica el algoritmo de KeyManagerFactory por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

En el caso del proveedor de JSSE Java puro de IBM, utilice:

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

En el caso del proveedor de JSSE Java puro de Sun Microsystems, Inc., utilice:

```
ssl.KeyManagerFactory.algorithm=SunX509
```

Hallará más información en el Javadoc de javax.net.ssl.KeyManagerFactory.

ssl.TrustManagerFactory.algorithm

Especifica el algoritmo de TrustManagerFactory por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

En el caso del proveedor de JSSE Java puro de IBM, utilice:

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

Hallará más información en el Javadoc de javax.net.ssl.TrustManagerFactory.

ssl.SocketFactory.provider

Especifica la fábrica de sockets SSL por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.SocketFactory.provider=com.ibm.i5os.jsse.JSSESocketFactory
```

En el caso del proveedor de JSSE Java puro de IBM, utilice:

```
ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
```

Hallará más información en el Javadoc de javax.net.ssl.SSLSocketFactory.

ssl.ServerSocketFactory.provider

Especifica la fábrica de sockets de servidor SSL por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.ServerSocketFactory.provider=com.ibm.i5os.jsse.JSSEServerSocketFactory
```

En el caso del proveedor de JSSE Java puro, utilice:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
```

Hallará más información en el Javadoc de javax.net.ssl.SSLServerSocketFactory.

Información relacionada

- Javadoc de javax.net.ssl.KeyManagerFactory
- Javadoc de javax.net.ssl.TrustManagerFactory
- Javadoc de javax.net.ssl.SSLSocketFactory
- Javadoc de javax.net.ssl.SSLServerSocketFactory

Propiedades Java del sistema en JSSE 1.5:

Si desea emplear JSSE en las aplicaciones, debe especificar varias propiedades del sistema que los objetos SSLContext predeterminados necesitan para proporcionar una confirmación de la configuración. Algunas de las propiedades son válidas para todos los proveedores, mientras que otras solo son válidas para el proveedor nativo de System i5.

Cuando se utiliza el proveedor de JSSE nativo del System i5, si no especifica las propiedades, os400.certificateContainer toma por defecto el valor *SYSTEM, lo que significa que JSSE utiliza la entrada predeterminada del almacén de certificados del sistema.

Propiedades válidas para el proveedor de JSSE nativo de System i5 y para el proveedor de JSSE Java puro de IBM

Las propiedades siguientes son válidas para ambos proveedores de JSSE. En cada descripción se indica la propiedad predeterminada, si procede.

javax.net.ssl.trustStore

El nombre del archivo que contiene el objeto KeyStore que desea que utilice el TrustManager predeterminado. El valor predeterminado es jssecacerts, o cacerts (si no existe jssecacerts).

javax.net.ssl.trustStoreType

El tipo de objeto KeyStore que desea que utilice el TrustManager predeterminado. El valor predeterminado es el valor devuelto por el método KeyStore.getDefaultType.

javax.net.ssl.trustStorePassword

La contraseña del objeto KeyStore que desea que utilice el TrustManager predeterminado.

javax.net.ssl.keyStore

El nombre del archivo que contiene el objeto KeyStore que desea que utilice el KeyManager predeterminado.El valor predeterminado es jssecacerts, o cacerts (si no existe jssecacerts).

javax.net.ssl.keyStoreType

El tipo de objeto KeyStore que desea que utilice el KeyManager predeterminado. El valor predeterminado es el valor devuelto por el método KeyStore.getDefaultType.

javax.net.ssl.keyStorePassword

La contraseña del objeto KeyStore que desea que utilice el KeyManager predeterminado.

Propiedades que solo funcionan para el proveedor de JSSE nativo del System i5

Las propiedades que solo son válidas para el proveedor de JSSE nativo de System i5 son las siguientes:

os400.secureApplication

El identificador de la aplicación. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType

os400.certificateContainer

El nombre del archivo de claves que desea utilizar. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

os400.certificateLabel

La etiqueta de archivo de claves que desea utilizar. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

Conceptos relacionados

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Información relacionada

Propiedades del sistema de Sun Microsystems, Inc.

Utilizar el proveedor de JSSE 1.5 nativo de System i5:

El proveedor de JSSE nativo de System i5 ofrece la suite completa de clases e interfaces JSSE, incluidas las implementaciones de la clase KeyStore y la clase SSLConfiguration de JSSE.

Valores de protocolo para el método SSLContext.getInstance

En la tabla que sigue se identifican y describen los valores de protocolo del método SSLContext.getInstance del proveedor de JSSE nativo de System i5.

Los protocolos SSL soportados pueden estar limitados por los valores de sistema establecidos en su sistema. Encontrará más detalles en el subtema: Valores del sistema de seguridad: protocolos de la capa de sockets segura (SSL), en la información de gestión de sistemas.

Valor de protocolo	Protocolos SSL soportados
SSL	SSL versión 2, SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.
SSLv2	SSL versión 2
SSLv3	Protocolo SSL versión 3. Aceptará hello SSLv3 encapsulado en un hello de formato SSLv2.
TLS	SSL versión 2, SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.
TLSv1	Protocolo TLS versión 1, definido en la petición de comentarios (RFC) 2246. Aceptará hello TLSv1 encapsulado en un hello de formato SSLv2.
SSL_TLS	SSL versión 2, SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.

Implementaciones de KeyStore nativo de System i5

El proveedor nativo de System i5 ofrece dos implementaciones de la clase KeyStore, que son IbmISeriesKeyStore o IBMi5OSKeyStore. Ambas implementaciones de KeyStore (almacén de claves) proporcionan una envoltura alrededor del soporte de gestor de certificados digitales (DCM).

IbmISeriesKeyStore
El contenido del almacén de claves se basa en un identificador de aplicación específico o un archivo de claves, una contraseña y una etiqueta. JSSE carga las entradas del almacén de claves del gestor de certificados digitales. Para cargar las entradas, JSSE utiliza el identificador de aplicación adecuado o la información del archivo de claves cuando la aplicación intenta por primera vez acceder a las entradas del almacén de claves o la información del archivo de claves. No es posible modificar el almacén de claves y todos los cambios de configuración deben efectuarse mediante el Gestor de certificados digitales.

IBMi5OSKeyStore
El contenido de este almacén de claves se basa en un archivo de almacén de certificados i5OS y la contraseña para acceder a ese archivo. Esta clase KeyStore permite modificar el almacén de certificados. Podrá hacer cambios sin tener que utilizar el gestor de certificados digitales (DCM).

La implementación de IBMi5OSKeyStore está en conformidad con la especificación de Sun Microsystems, Inc., para la API Java KeyStore. Encontrará más detalles en la información Javadoc de Keystore de Sun Microsystems, Inc.

Para obtener más información sobre cómo gestionar almacenes de claves mediante el DCM, vea el tema Gestor de certificados digitales (DCM).

Información relacionada

Valores del sistema de seguridad: protocolos de la capa de sockets segura (SSL)

Información Javadoc de la clase i5OSLoadStoreParameter:

com.ibm.i5os.keystore

Clase i5OSLoadStoreParameter

```
java.lang.Object
  |
  +--com.ibm.i5os.keystore.i5OSLoadStoreParameter
```

Todas las interfaces implementadas:

java.security.Keystore.LoadStoreParameter

```
public class i5OSLoadStoreParameter
extends java.lang.Object
implements java.security.Keystore.LoadStoreParameter
```

Esta clase crea un objeto KeyStore.ProtectionParameter que sirve para cargar/almacenar almacenes de certificados de i5OS. Una vez creado, la clase proporciona información sobre el almacén de certificados al que hay que acceder y sobre la contraseña empleada para proteger el almacén de certificados.

Ejemplo de uso de esta clase:

```
//inicializar el almacén de claves (keystore)
    KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");

 //Cargar un almacén de claves existente
    File kdbFile = new File("/tmp/certificateStore.kdb");
    i5OSLoadStoreParameter lsp =
   new i5OSLoadStoreParameter (kdbFile, "password".toCharArray());
    ks.load(lsp);

 //Obtener y añadir entradas al almacén de certificados
    ...

 //Guardar el almacén de certificados
    Ks.store(lsp);
```

Desde:
SDK 1.5

Resumen del constructor

i5OSLoadStoreParameter(java.io.File ksFile, char[] password)

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la contraseña que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

i5OSLoadStoreParameter(java.io.File ksFile, java.security.KeyStore.PasswordProtection pwdProtParam)

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la PasswordProtection que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

Tabla 14. **Resumen de los métodos**

java.security.KeyStore. ProtectionParameter	*"getProtectionParameter" en la página 310*() Devuelve el objeto KeyStore.KeyStoreParameter ascociado a este LoadStoreParameter

---

Métodos heredados de la clase java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Detalles del constructor

i5OSLoadStoreParameter

```
public i5OSLoadStoreParameter(java.io.File ksFile,
                              char[] password)
                       throws java.lang.IllegalArgumentException
```

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la contraseña que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

Parámetros:

ksFile - Objeto Archivo del almacén de claves.

Si se ha utilizado keystore.load() con un método i5OSLoadStoreParameter(ksFile = null, password), se crea un almacén de claves nuevo.

Si se ha utilizado keystore.store() con un método i5OSLoadStoreParameter(ksFile = null, password), se lanza una IllegalArgumentException.

password - Contraseña para acceder al almacén de certificados de i5OS. No puede ser null ni está vacía.

Lanza:

java.lang.IllegalArgumentException - Si la contraseña es nula o está vacía

---

i5OSLoadStoreParameter

```
public i5OSLoadStoreParameter(java.io.File ksFile,
                              java.security.KeyStore.PasswordProtection pwdProtParam)
                       throws java.lang.IllegalArgumentException
```

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la PasswordProtection que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

Si se ha utilizado keystore.load() con un método i5OSLoadStoreParameter(ksFile = null, password), se crea un almacén de claves nuevo.

Si se ha utilizado keystore.store() con un método i5OSLoadStoreParameter(ksFile = null, password), se lanza una IllegalArgumentException.

Parámetros:

ksFile - Objeto Archivo del almacén de claves.

pwdProtParam - Instancia de PasswordProtection que se empleará para adquirir la contraseña. No puede ser null.

Lanza:

java.lang.IllegalArgumentException - Si KeyStore.PasswordProtection es null, o si la contraseña contenida en pwdProtParam es null o está vacía.

Detalles del método

---

getProtectionParameter

```
public java.security.KeyStore.ProtectionParameter getProtectionParameter()
```

Devuelve el objeto KeyStore.KeyStoreParameter ascociado a este LoadStoreParameter.

Especificado por:

```
getProtectionParameter in interface java.security.KeyStore.LoadStoreParameter
```

Devuelve:

Una instancia que implementa la interfaz KeyStore.KeyStoreParameter

Vea también:

```
java.security.KeyStore.ProtectionParameter#getProtectionParameter()
```

Información Javadoc de la clase i5OSSystemCertificateStoreFile:

com.ibm.i5os.keystore

Class i5OSSystemCertificateStoreFile

```
java.lang.Object
   java.io.File
       com.ibm.i5os.keystore.i5OSSystemCertificateStoreFile
```

Todas las interfaces implementadas:

java.io.Serializable, java.lang.Comparable<java.io.File>

```
public class i5OSSystemCertificateStoreFile
extends java.io.File
```

Esta clase proporciona una nueva implementación de archivo que señala hacia el archivo de almacén de certificados *SYSTEM. Proporciona un mecanismo para que el usuario cargue el almacén de certificados *SYSTEM sin obligarle a saber cuál es la vía real de acceso al almacén.

Para cargar el almacén de certificados *SYSTEM en un almacén de claves, cree primero un i5OSSystemCertificateStoreFile.

A partir de aquí, el almacén de claves se puede cargar de dos maneras:

- Mediante un i5OSLoadStoreParameter:

```
//crear un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();

  //utilizar ese archivo para crear un i5OSLoadStoreParameter
     i5OSLoadStoreParameter lsp = new i5OSLoadStoreParameter(starSystemFile, pwd);

  //cargar el almacén de certificados en un almacén de claves
     KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
     ks.load(lsp);
```

- Mediante una FileInputStream:

```
//crear un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();
```

```
//crear una corriente de entrada al starSystemFile
   FileInputStream fis = new FileInputStream(starSystemFile);

//cargar el almacén de certificados en un almacén de claves
   KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
   ks.load(fis, pwd);
```

Desde:
SDK 1.5

Vea también:
Forma serializada

Resumen de los campos

Campos heredados de la clase java.io.File
pathSeparator, pathSeparatorChar, separator, separatorChar

Resumen del constructor

i5OSSystemCertificateStoreFile()

Crea un File() que señala hacia el archivo de almacén de certificados *System.

Resumen de los métodos

Métodos heredados de la clase java.io.File
canRead, canWrite, compareTo, createNewFile, createTempFile, createTempFile, delete, deleteOnExit, equals, exists, getAbsoluteFile, getAbsolutePath, getCanonicalFile, getCanonicalPath, getName, getParent, getParentFile, getPath, hashCode, isAbsolute, isDirectory, isFile, isHidden, lastModified, length, list, list, listFiles, listFiles, listFiles, listRoots, mkdir, mkdirs, renameTo, setLastModified, setReadOnly, toString, toURI, toURL

Métodos heredados de la clase java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Detalles del constructor

i5OSSystemCertificateStoreFile

public **i5OSSystemCertificateStoreFile**()

Crea un File() que señala hacia el archivo de almacén de certificados *System.

Información Javadoc de SSLConfiguration para la versión 1.5:

com.ibm.i5os.jsse

Clase SSLConfiguration

```
java.lang.Object
  |
  +--com.ibm.i5os.jsse.SSLConfiguration
```

Todas las interfaces implementadas:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

public final class **SSLConfiguration**
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable

Esta clase corresponde a la especificación de la configuración necesaria para la implementación JSSE nativa de System i5.

La implementación JSSE nativa de System i5 funciona de manera más eficaz con un objeto `KeyStore` de tipo ″IbmISeriesKeyStore″. Este tipo de objeto `KeyStore` contiene entradas de clave y entradas de certificado de confianza basadas ya sea en un identificador de aplicación registrado en el gestor de certificado digital (DCM) o en un archivo de conjunto de claves (contenedor de certificados digitales). Luego, un objeto `KeyStore` de este tipo puede servir para inicializar un objeto `X509KeyManager` y un objeto `X509TrustManager` del `Provider` ″IBMi5OSJSSEProvider″. Después, los objetos `X509KeyManager` y `X509TrustManager` se pueden usar para inicializar un objeto `SSLContext` del ″IBMi5OSJSSEProvider″. Entonces, el objeto `SSLContext` proporciona acceso a la implementación JSSE nativa de System i5 basándose en la información de configuración especificada para el objeto `KeyStore`. Cada vez que se realiza una carga en un `KeyStore` de ″IbmISeriesKeyStore″, el `KeyStore` se inicializa basándose en la configuración actual especificada por el identificador de la aplicación o el archivo de conjunto de claves.

Esta clase también puede utilizarse para generar un objeto `KeyStore` de cualquier tipo válido. El `KeyStore` se inicializa basándose en la configuración actual especificada por el identificador de la aplicación o el archivo de conjunto de claves. Para realizar cualquier cambio en la configuración especificada por un identificador de aplicación o por un archivo de conjunto de claves, es necesario regenerar el objeto `KeyStore`. Observe que se tiene que especificar un contraseña de conjunto de claves (para el almacén de certificados *SYSTEM cuando se utiliza el identificador de la aplicación) para poder crear con éxito otro tipo de `KeyStore` diferente a ″IbmISeriesKeyStore″. La contraseña de conjunto de claves debe especificarse para conseguir el acceso a cualquier clave privada de cualquier `KeyStore` de tipo ″IbmISeriesKeyStore″.

Desde:
SDK 1.5

Consulte también:
KeyStore, X509KeyManager, X509TrustManager, SSLContext

Resumen del constructor

`SSLConfiguration()` crea una nueva SSLConfiguration. Consulte “Detalles del constructor” en la página 313 para obtener más información.

Tabla 15. **Resumen de los métodos**

`void`	“clear” en la página 316() Borra toda la información en el objeto para que todos los métodos get devuelvan un valor nulo.
`java.lang.Object`	“clone” en la página 318() Genera una nueva copia de esta configuración SSL.
`boolean`	“equals” en la página 317(java.lang.Objectobj) Indica si algún otro objeto es ″igual a″ este.
`protected void`	“finalize” en la página 316() llamado por el recogedor de basura de un objeto cuando la recogida de basura determina que no hay más referencias al objeto.
`java.lang.String`	“getApplicationId” en la página 316() Devuelve el ID de la aplicación.
`java.lang.String`	“getKeyringLabel” en la página 316() Devuelve la etiqueta de conjunto de claves.

Tabla 15. **Resumen de los métodos** *(continuación)*

`java.lang.String`	“getKeyringName” en la página 316() Devuelve el nombre de conjunto de claves.
`char[]`	“getKeyringPassword” en la página 316() Devuelve la contraseña de conjunto de claves.
`java.security.KeyStore`	“getKeyStore” en la página 318(char[]password) Devuelve un almacén de claves de tipo ″IbmISeriesKeyStore″ utilizando la contraseña especificada.
`java.security.KeyStore`	“ getKeyStore” en la página 318(java.lang.Stringtype, char[]password) Devuelve un almacén de claves del tipo solicitado utilizando la contraseña especificada.
`int`	“hashCode” en la página 318() Devuelve un valor de código hash para el objeto.
`staticvoid`	(java.lang.String[]args) Ejecuta las funciones de SSLConfiguration.
`void`	(java.lang.String[]args, java.io.PrintStreamout) Ejecuta las funciones de SSLConfiguration.
`void`	“setApplicationId” en la página 317(java.lang.StringapplicationId) Establece el identificador de aplicación.
`void`	“setApplicationId” en la página 317(java.lang.StringapplicationId, char[]password) Establece el identificador de la aplicación y la contraseña del conjunto de claves.
`void`	“setKeyring” en la página 317(java.lang.Stringname,java.lang.Stringlabel, char[]password) Establece la información de conjunto de claves.

Métodos heredados de la clase java.lang.Object
`getClass, notify, notifyAll, toString, wait, wait, wait`

Detalles del constructor

SSLConfiguration

```
public SSLConfiguration()
```

Crea una nueva SSLConfiguration. El identificador de aplicación y la información del conjunto de claves se inicializa con valores de omisión.

El valor predeterminado para el identificador de aplicación es el valor especificado para la propiedad ″os400.secureApplication″.

Los valores predeterminados para la información de conjunto de claves son igual a nulo si se especifica la propiedad ″os400.secureApplication″. Si no se especifica la propiedad ″os400.secureApplication″, el valor para el nombre del conjunto de claves es el valor especificado para la propiedad ″os400.certificateContainer″. Si no se especifica la propiedad ″os400.secureApplication″, la etiqueta de conjunto de claves se inicializa con el valor de la ″os400.certificateLabel″. Si no se establece la propiedad ″os400.secureApplication″ ni la ″os400.certificateContainer″, el nombre de conjunto de claves se inicializará con ″*SYSTEM″.

Detalles del método

---

main

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

Ejecuta las funciones de SSLConfiguration. Hay cuatro mandatos que se pueden ejecutar: -help, -create, -display y -update. El mandato debe ser el primer parámetro especificado.

Las opciones que se pueden especificar son las siguientes (en cualquier orden):

-keystore **nombre-archivo-almacén-claves**
Especifica el nombre del archivo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para todos los mandatos.

-storepass **contraseña-archivo-almacén-claves**
Especifica la contraseña asociada con el archivo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para todos los mandatos.

-storetype **tipo-almacén-claves**
Especifica el tipo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para cualquier mandato. Si no se especifica esta opción, se utiliza un valor de "IbmISeriesKeyStore".

-varname>-appid identificador-aplicación
Especifica el identificador de aplicación que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-keyring **nombre-archivo-conjunto-claves**
Especifica el nombre de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-keyringpass **contraseña-archivo-conjunto-claves**
Especifica la contraseña de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Puede especificarse esta opción para los mandatos *-create* y *-update* y es necesaria cuando se especifica un tipo de almacén de claves distinto de "IbmISeriesKeyStore". Si no se especifica esta opción, se utiliza la contraseña oculta de conjunto de claves.

-keyringlabel **etiqueta-archivo-conjunto-claves**
Especifica la etiqueta de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción solo se especifica cuando también se especifica la opción *-keyring*. Si no es especifica esta opción cuando se especifica la opción *keyring*, se utiliza la etiqueta por omisión en el conjunto de claves.

-systemdefault
Especifica el valor predeterminado del sistema que se utiliza para inicializar un almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos-*create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-v Especifica que se debe generar salida verbosa. Esta opción se puede especificar para cualquier mandato.

El mandato ayuda visualiza la información de uso para especificar los parámetros de este método. Los parámetros para invocar la función de ayuda se especifican de esta manera:

```
-help
```

El mandato create crea un nuevo archivo de almacén de datos. Existen tres variantes del mandato create. Una variante para crear un almacén de claves basado en un identificador de aplicación particular, otra variante para crear un almacén de claves basado en un nombre, una etiqueta y una contraseña de conjunto de claves, y la tercera variante para crear un conjunto de claves basado en la configuración por omisión del sistema.

Para crear un almacén de claves basado en un identificador de aplicación particular, se debe especificar la opción *-appid*. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en el identificador de aplicación ″APPID″:

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -appid APPID
```

Para crear un almacén de claves basado en un archivo de conjunto de claves particular, se debe especificar la opción *-keyring*. Las opciones *-keyringpass* y *keyringlabel* también pueden especificarse. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en el archivo de almacén de claves ″keyring.file″, basándose en la contraseña ″ringpass″ y en la etiqueta de conjunto de claves ″keylabel″:

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

Para crear un almacén de claves basado en la configuración por omisión del sistema, se debe especificar la opción-*systemdefault*. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en la configuración por omisión del sistema :

```
-create -keystore keystore.file -storepass keypass -systemdefault
```

El mandato update actualiza un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″. Existen tres variantes del mandato update que son idénticas a las variantes del mandato create. Las opciones del mandato update son idénticas a las opciones utilizadas para el mandato create. El mandato display visualiza la configuración especificada de un archivo de almacén de claves existente. Los siguientes parámetros visualizarían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″:

```
-display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

Parámetros:

`args` - los argumentos de la línea de mandatos

---

run

```
public void run(java.lang.String[]args,
                java.io.PrintStreamout)
```

Ejecuta las funciones de SSLConfiguration. Los parámetros y la funcionalidad de este método son idénticos a los del método main().

Parámetros:

`args` - los argumentos de mandato

`out` - la corriente de salida en que se deben escribir los resultados

Vea también: `com.ibm.i5os.jsse.SSLConfiguration.main()`

---

getApplicationId

```
public java.lang.String getApplicationId()
```

Devuelve el ID de la aplicación.

Devuelve:
el ID de la aplicación.

---

getKeyringName

```
public java.lang.String getKeyringName()
```

Devuelve el nombre de conjunto de claves.

Devuelve:
el nombre de conjunto de claves.

---

getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

Devuelve la etiqueta de conjunto de claves.

Devuelve:
la etiqueta de conjunto de claves.

---

getKeyringPassword

```
public final char[] getKeyringPassword()
```

Devuelve la contraseña de conjunto de claves.

Devuelve:
la contraseña de conjunto de claves.

---

finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Llamado por el recogedor de basura de un objeto cuando la recogida de basura determina que no hay más referencias al objeto.

Alteraciones temporales:
`finalize` en la clase `java.lang.Object`

Lanza: `java.lang.Throwable` - la excepción lanzada por este método.

---

clear

```
public void clear()
```

Borra toda la información en el objeto para que todos los métodos get devuelvan un valor nulo.

---

setKeyring

```
public void setKeyring(java.lang.Stringname,
                       java.lang.Stringlabel,
                       char[]password)
```

Establece la información de conjunto de claves.

Parámetros:

`name` - el nombre de conjunto de claves

`label` - la etiqueta de conjunto de claves, o el valor nulo si se utiliza la entrada de conjunto de claves por omisión.

`password` - la contraseña de conjunto de claves, o el valor nulo si se utiliza la contraseña oculta.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId)
```

Establece el ID de la aplicación.

Parámetros:

`applicationId` - el ID de la aplicación.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId,
                             char[]password)
```

Establece el ID de la aplicación y la contraseña del conjunto de claves. Al especificar la contraseña de conjunto de claves se permite cualquier conjunto de claves creado para permitir el acceso a la clave privada.

Parámetros:

`applicationId` - el ID de la aplicación.

`password` - la contraseña de conjunto de claves.

---

equals

```
public boolean equals(java.lang.Objectobj)
```

Indica si algún otro objeto es ″igual a″ este.

Alteraciones temporales:

`equals` de la clase `java.lang.Object`

Parámetros:

`obj` - objeto para comparar

Devuelve:

el indicador de si los objetos especifican la misma información de configuración

---

hashCode

```
public int hashCode()
```

Devuelve un valor de código hash para el objeto.

Alteraciones temporales:
hashCode de la clase java.lang.Object

Devuelve:
un valor de código hash para el objeto.

---

clone

```
public java.lang.Object clone()
```

Genera una nueva copia de esta configuración SSL. Los cambios posteriores en los componentes de esta configuración no afectarán a la copia nueva y viceversa.

Alteraciones temporales:
clone de la clase java.lang.Object

Devuelve:
una copia de esta configuración SSL

---

getKeyStore

```
public java.security.KeyStore getKeyStore(char[]password)
                                  throws java.security.KeyStoreException
```

Devuelve un almacén de claves de tipo "IbmISeriesKeyStore" utilizando la contraseña especificada. Se inicializa el almacén de claves basándose en la información de configuración actual almacenada en el objeto.

Parámetros:
password - se utiliza para inicializar el almacén de claves

Devuelve:
el almacén de claves KeyStore inicializado basándose en la información de configuración actual almacenada en el objeto.

Lanza: java.security.KeyStoreException - si no se puede crear el almacén de claves

---

getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                           char[]password)
                                  throws java.security.KeyStoreException
```

Devuelve un almacén de claves del tipo solicitado utilizando la contraseña especificada. Se inicializa el almacén de claves basándose en la información de configuración actual almacenada en el objeto.

Parámetros:
type - tipo de almacén de claves a devolver

password - se utiliza para inicializar el almacén de claves

Devuelve:
el almacén de claves KeyStore inicializado basándose en la información de configuración actual almacenada en el objeto.

Lanza: `java.security.KeyStoreException` - si no se puede crear el almacén de claves

Ejemplos: IBM Java Secure Sockets Extension 1.5:

Los ejemplos de JSSE muestran cómo pueden un cliente y un servidor emplear el proveedor de JSSE nativo de System i5 para crear un contexto que habilite las comunicaciones seguras.

Nota: Ambos ejemplos utilizan el proveedor de JSSE nativo de System i5, sean cuales sean las propiedades especificadas por el archivo java.security.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

Ejemplo: cliente SSL que utiliza un objeto SSLContext de la versión 1.5:

Este programa cliente de ejemplo utiliza un objeto SSLContext, que inicializa para emplear el ID de aplicación "MY_CLIENT_APP". Este programa utilizará la implementación nativa de System i5 con independencia de lo que se haya especificado en el archivo java.security.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////////////////////////////////////////
//
// Este programa cliente de ejemplo utiliza un objeto SSLContext, que inicializa
// para emplear el ID de aplicación "MY_CLIENT_APP".
//
// El ejemplo usa el proveedor de JSSE nativo de System i5, sean cuales sean
// las propiedades especificadas por el archivo java.security.
//
// Sintaxis de mandato:
//     java -Djava.version=1.5 SslClient
//
// Fíjese en que "-Djava.version=1.5" es innecesario si se ha configurado
// que hay que usar JDK versión 1.5 por defecto.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
import com.ibm.i5os.jsse.SSLConfiguration;
/**
 * Programa cliente SSL.
 */
public class SslClient {

   /**
    * Método main de SslClient.
    *
    * @param args argumentos de línea de mandatos (no se utiliza)
    */
   public static void main(String args[]) {
       /*
        * Configurar para capturar las excepciones lanzadas.
        */
       try {
           /*
            * Inicializar un objeto SSLConfiguration para especificar un ID de
            * aplicación.  MY_CLIENT_APP se debe registrar y configurar
```

```
     * correctamente con el Gestor de certificados digitales (DCM).
     */
    SSLConfiguration config = new SSLConfiguration();
    config.setApplicationId("MY_CLIENT_APP");
    /*
     * Obtener un objeto KeyStore del objeto SSLConfiguration.
     */
    char[] password = "password".toCharArray();
    KeyStore ks = config.getKeyStore(password);
    /*
     * Asignar e inicializar una KeyManagerFactory.
     */
    KeyManagerFactory kmf =
                      KeyManagerFactory.getInstance("IbmISeriesX509");
    kmf.init(ks, password);
    /*
     * Asignar e inicializar una TrustManagerFactory.
     */
    TrustManagerFactory tmf =
                    TrustManagerFactory.getInstance("IbmISeriesX509");
    tmf.init(ks);
    /*
     * Asignar e inicializar un SSLContext.
     */
    SSLContext c =
               SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
    c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
    /*
     * Obtener una SSLSocketFactory del SSLContext.
     */
    SSLSocketFactory sf = c.getSocketFactory();
    /*
     * Crear un SSLSocket.
     *
     * Cambiar la dirección IP codificada por programa por la dirección IP o el
     * nombre de host del servidor.
     */
    SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
    /*
     * Enviar un mensaje al servidor mediante la sesión segura.
     */
    String sent = "Probar la escritura SSL java";
    OutputStream os = s.getOutputStream();
    os.write(sent.getBytes());
    /*
     * Escribir los resultados en la pantalla.
     */
    System.out.println("Escritos " + sent.length() + " bytes...");
    System.out.println(sent);
    /*
     * Recibir un mensaje del servidor mediante la sesión segura.
     */
    InputStream is = s.getInputStream();
    byte[] buffer = new byte[1024];
    int bytesRead = is.read(buffer);
    if (bytesRead == -1)
        throw new IOException("Recibido fin de archivo inesperado");
    String received = new String(buffer, 0, bytesRead);
    /*
     * Escribir los resultados en la pantalla.
     */
    System.out.println("Leídos " + received.length() + " bytes...");
    System.out.println(received);
} catch (Exception e) {
    System.out.println("Excepción inesperada: "+;
                       e.getMessage());
    e.printStackTrace();
```

```
        }
    }

}
```

Ejemplo: servidor SSL que utiliza un objeto SSLContext de la versión 1.5:

El siguiente programa servidor utiliza un objeto SSLContext que inicializa con un archivo de almacén de claves creado anteriormente.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// El siguiente programa servidor utiliza un objeto SSLContext que
// inicializa con un archivo de almacén de claves creado anteriormente.
//
// El archivo de almacén de claves tiene el nombre y la contraseña siguientes:
//   Nombre de archivo: /home/keystore.file
//   Contraseña: password
//
// El programa de ejemplo necesita el archivo de almacén de claves para crear un
// objeto IbmISeriesKeyStore. El objeto KeyStore debe especificar MY_SERVER_APP como
// identificador de la aplicación.
//
// Para crear el archivo de almacén de claves, puede emplear el siguiente mandato de Qshell:
//
//   java com.ibm.i5os.SSLConfiguration  -create  -keystore /home/keystore.file
//     -storepass password  -appid MY_SERVER_APP
//
// Sintaxis de mandato:
//     java -Djava.version=1.5 JavaSslServer
//
// Fíjese en que "-Djava.version=1.5" es innecesario si se ha configurado
// que hay que usar JDK versión 1.5 por defecto.
//
// También puede crear el archivo de almacén de claves entrando este mandato en un indicador de mandatos de i5/OS:
//
//  RUNJVA CLASS(com.ibm.i5os.SSLConfiguration) PARM('-create' '-keystore'
//      '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
//
//////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
/**
* Programa servidor Java SSL que utiliza el ID de aplicación.
*/
public class JavaSslServer {

   /**
    * Método main de JavaSslServer.
    *
    * @param args argumentos de línea de mandatos (no se utiliza)
    */
   public static void main(String args[]) {
       /*
        * Configurar para capturar las excepciones lanzadas.
        */
       try {
           /*
            * Asignar e inicializar un objeto KeyStore.
            */
           char[] password = "password".toCharArray();
```

```
            KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
            FileInputStream fis = new FileInputStream("/home/keystore.file");
            ks.load(fis, password);
            /*
             * Asignar e inicializar una KeyManagerFactory.
             */
            KeyManagerFactory kmf =
                            KeyManagerFactory.getInstance("IbmISeriesX509");
            kmf.init(ks, password);
            /*
             * Asignar e inicializar una TrustManagerFactory.
             */
            TrustManagerFactory tmf =
                          TrustManagerFactory.getInstance("IbmISeriesX509");
            tmf.init(ks);
            /*
             * Asignar e inicializar un SSLContext.
             */
            SSLContext c =
                       SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
            c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
            /*
             * Obtener una SSLServerSocketFactory del SSLContext.
             */
            SSLServerSocketFactory sf = c.getServerSocketFactory();
            /*
             * Crear un SSLServerSocket.
             */
            SSLServerSocket ss =
                (SSLServerSocket) sf.createServerSocket(13333);
            /*
             * Efectuar un accept() para crear un SSLSocket.
             */
            SSLSocket s = (SSLSocket) ss.accept();
            /*
             * Recibir un mensaje del cliente mediante la sesión segura.
             */
            InputStream is = s.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead = is.read(buffer);
            if (bytesRead == -1)
                throw new IOException("Recibido fin de archivo inesperado");
            String received = new String(buffer, 0, bytesRead);
            /*
             * Escribir los resultados en la pantalla.
             */
            System.out.println("Leídos " + received.length() + " bytes...");
            System.out.println(received);
            /*
             * Volver a enviar como un eco el mensaje al cliente mediante la sesión segura.
             */
            OutputStream os = s.getOutputStream();
            os.write(received.getBytes());
            /*
             * Escribir los resultados en la pantalla.
             */
            System.out.println("Escritos " + received.length() + " bytes...");
            System.out.println(received);
        } catch (Exception e) {
            System.out.println("Excepción inesperada: "+;
                               e.getMessage());
            e.printStackTrace();
        }
    }

}
```

Utilizar la extensión de sockets seguros Java 6

Esta información solo es válida cuando se utiliza JSSE en un sistema que ejecute la plataforma Java 2, Standard Edition (J2SE), versión 6 o posterior. JSSE es como una infraestructura que abstrae los mecanismos subyacentes de SSL y TLS. Mediante la abstracción de la complejidad y las peculiaridades de los protocolos subyacentes, JSSE permite a los programadores utilizar comunicaciones cifradas seguras al tiempo que se minimizan las posibles vulneraciones de seguridad. La extensión de sockets seguros Java utiliza ambos protocolos, el seguridad de capa de transporte (TLS) de capa de sockets segura (SSL) y el protocolo de seguridad de capa de transporte (TLS), para proporcionar comunicaciones cifradas seguras entre los clientes y los sevidores.

A la implementación IBM de JSSE se le llama IBM JSSE. En la extensión JSEE de IBM hay un proveedor de JSSE nativo de i5/OS y un proveedor de JSSE Java puro de IBM. Además, la extensión JSSE de Sun Microsystems, Inc., venía inicialmente en el System i, y seguirá viniendo con él.

Configurar el servidor para que soporte JSSE 6:

Configure el System i5 para que utilice implementaciones distintas de JSSE. Este tema incluye los requisitos de software, cómo cambiar los proveedores de JSSE y las propiedades de seguridad y las propiedades del sistema necesarias. En la configuración predeterminada se emplea el proveedor de JSSE Java puro de IBM, conocido como IBMJSSE2.

Cuando utiliza la plataforma Java 2, Standard Edition (J2SE) versión 6 en el System i5, JSSE ya está configurado. En la configuración predeterminada se emplea el proveedor de JGSS nativo de System i5.

Cambiar los proveedores de JSSE

Puede configurar JSSE para que utilice el proveedor de JSSE nativo de System i5 o el proveedor de JSEE de Sun Microsystems, Inc., en lugar del proveedor de JSEE Java puro de IBM. Si cambia algunas propiedades específicas de seguridad JSSE y algunas propiedades Java del sistema, podrá conmutar entre proveedores.

Gestores de seguridad

Si se propone ejecutar la aplicación JSSE teniendo habilitado un gestor de seguridad Java, es posible que tenga que establecer los permisos de red disponibles. Hallará más información en: Permisos SSL, en Permisos de Java 2 SDK.

Proveedores de JSSE 6:

En la extensión JSEE de IBM hay un proveedor de JSSE nativo de System i5 y un proveedor de JSSE Java puro de IBM. El System i5 también viene con la extensión JSSE de Sun Microsystems, Inc. El proveedor que elija utilizar dependerá de las necesidades de la aplicación.

Todos los proveedores cuplen la especificación de la interfaz JSSE. Pueden comunicarse entre sí y con cualquier otra implementación de SSL o TLS, incluso con implementaciones que no sean Java.

Proveedor de JSSE Java puro de Sun Microsystems, Inc.

Esta es la implementación Java Sun de la extensión JSSE. Venía inicialmente en el System i, y seguirá viendo con él. Para obtener más información sobre la extensión JSSE de Sun Microsystems, Inc., vea el manual de consulta de la extensión de sockets seguros Java (JSSE) de Sun Microsystems, Inc.

Proveedor de JSSE Java puro de IBM

El proveedor de JSSE Java puro de IBM ofrece las siguientes características:

- Funciona con cualquier tipo de objeto KeyStore para controlar y configurar certificados digitales (por ejemplo, JKS, PKCS12, etc.).
- Le permite utilizar juntos cualquier combinación de componentes JSSE de diversas implementaciones.

El nombre del proveedor de la implementación Java puro es IBMJSSEProvider2. Debe pasar este nombre de proveedor, utilizando las mayúsculas y minúsculas correctas, al método java.security.Security.getProvider() o a los diversos métodos getInstance() para varias de las clases de JSSE.

Proveedor de JSSE nativo de System i5

El proveedor de JSSE nativo de System i5 ofrece las siguientes características:

- Utiliza el soporte SSL nativo del System i5.
- Permite usar el gestor de certificados digitales para configurar y controlar certificados digitales. Esto se proporciona por medio de un tipo de KeyStore (almacén de claves) exclusivo del System i5 (IbmISeriesKeyStore).
- Le permite utilizar juntos cualquier combinación de componentes JSSE de diversas implementaciones.

El nombre de proveedor de la implementación nativa de System i5 es IBMi5OSJSSEProvider. Debe pasar este nombre de proveedor, utilizando las mayúsculas y minúsculas correctas, al método java.security.Security.getProvider() o a los diversos métodos getInstance() para varias de las clases de JSSE.

Cambiar el proveedor de JSSE por omisión

Puede cambiar el proveedor de JSSE por omisión efectuando las modificaciones adecuadas en las propiedades de seguridad.

Tras cambiar el proveedor de JSSE, compruebe que las propiedades del sistema especifican la configuración correcta para la información de certificados digitales (almacén de claves) que requiere el nuevo proveedor.

Hallará más información en: Propiedades de la seguridad de JSSE 6.

 Manual de consulta de JSSE de Sun Microsystems, Inc.

"Propiedades de la seguridad de JSSE 6"
La máquina virtual Java (JVM) emplea importantes propiedades de seguridad que se establecen editando el archivo de propiedades Java maestro de seguridad.

Propiedades de la seguridad de JSSE 6:

La máquina virtual Java (JVM) emplea importantes propiedades de seguridad que se establecen editando el archivo de propiedades Java maestro de seguridad.

Este archivo, que se llama java.security, suele residir en el directorio /QIBM/ProdData/Java400/jdk6/lib/security del servidor.

La lista siguiente describe varias propiedades de seguridad relevantes para utilizar JSSE. Utilice las descripciones a modo de guía para editar el archivo java.security.

security.provider.<entero>

El proveedor de JSSE que desea utilizar. Estáticamente también registra las clases de proveedor criptográfico. Especifique los distintos proveedores de JSSE exactamente igual que en el ejemplo siguiente:

```
security.provider.5=com.ibm.i5os.jsse.JSSEProvider
security.provider.6=com.ibm.jsse2.IBMJSSEProvider2
security.provider.7=com.sun.net.ssl.internal.ssl.Provider
```

ssl.KeyManagerFactory.algorithm

Especifica el algoritmo de KeyManagerFactory por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

En el caso del proveedor de JSSE Java puro de IBM, utilice:

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

En el caso del proveedor de JSSE Java puro de Sun Microsystems, Inc., utilice:

```
ssl.KeyManagerFactory.algorithm=SunX509
```

Hallará más información en el Javadoc de javax.net.ssl.KeyManagerFactory.

ssl.TrustManagerFactory.algorithm

Especifica el algoritmo de TrustManagerFactory por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

En el caso del proveedor de JSSE Java puro de IBM, utilice:

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

Hallará más información en el Javadoc de javax.net.ssl.TrustManagerFactory.

ssl.SocketFactory.provider

Especifica la fábrica de sockets SSL por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.SocketFactory.provider=com.ibm.i5os.jsse.JSSESocketFactory
```

En el caso del proveedor de JSSE Java puro de IBM, utilice:

```
ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
```

Hallará más información en el Javadoc de javax.net.ssl.SSLSocketFactory.

ssl.ServerSocketFactory.provider

Especifica la fábrica de sockets de servidor SSL por omisión. En el caso del proveedor de JSSE nativo de System i5, utilice:

```
ssl.ServerSocketFactory.provider=com.ibm.i5os.jsse.JSSEServerSocketFactory
```

En el caso del proveedor de JSSE Java puro, utilice:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
```

Hallará más información en el Javadoc de javax.net.ssl.SSLServerSocketFactory.

Información relacionada

Javadoc de javax.net.ssl.KeyManagerFactory

Javadoc de javax.net.ssl.TrustManagerFactory

Javadoc de javax.net.ssl.SSLSocketFactory

Javadoc de javax.net.ssl.SSLServerSocketFactory

Propiedades Java del sistema en JSSE 6:

Si desea emplear JSSE en las aplicaciones, debe especificar varias propiedades del sistema que los objetos SSLContext predeterminados necesitan para proporcionar una confirmación de la configuración. Algunas de las propiedades son válidas para todos los proveedores, mientras que otras solo son válidas para el proveedor nativo de System i5.

Cuando se utiliza el proveedor de JSSE nativo del System i5, si no especifica las propiedades, os400.certificateContainer toma por defecto el valor *SYSTEM, lo que significa que JSSE utiliza la entrada predeterminada del almacén de certificados del sistema.

Propiedades válidas para el proveedor de JSSE nativo de System i5 y para el proveedor de JSSE Java puro de IBM

Las propiedades siguientes son válidas para ambos proveedores de JSSE. En cada descripción se indica la propiedad predeterminada, si procede.

javax.net.ssl.trustStore

El nombre del archivo que contiene el objeto KeyStore que desea que utilice el TrustManager predeterminado. El valor predeterminado es jssecacerts, o cacerts (si no existe jssecacerts).

javax.net.ssl.trustStoreType

El tipo de objeto KeyStore que desea que utilice el TrustManager predeterminado. El valor predeterminado es el valor devuelto por el método KeyStore.getDefaultType.

javax.net.ssl.trustStorePassword

La contraseña del objeto KeyStore que desea que utilice el TrustManager predeterminado.

javax.net.ssl.keyStore

El nombre del archivo que contiene el objeto KeyStore que desea que utilice el KeyManager predeterminado.El valor predeterminado es jssecacerts, o cacerts (si no existe jssecacerts).

javax.net.ssl.keyStoreType

El tipo de objeto KeyStore que desea que utilice el KeyManager predeterminado. El valor predeterminado es el valor devuelto por el método KeyStore.getDefaultType.

javax.net.ssl.keyStorePassword

La contraseña del objeto KeyStore que desea que utilice el KeyManager predeterminado.

Propiedades que solo funcionan para el proveedor de JSSE nativo del System i5

Las propiedades que solo son válidas para el proveedor de JSSE nativo de System i5 son las siguientes:

os400.secureApplication

El identificador de la aplicación. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType

os400.certificateContainer

El nombre del archivo de claves que desea utilizar. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

os400.certificateLabel

La etiqueta de archivo de claves que desea utilizar. JSSE solo utiliza esta propiedad cuando no se ha especificado ninguna de las propiedades siguientes:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

Conceptos relacionados

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Información relacionada

Propiedades del sistema de Sun Microsystems, Inc.

Utilizar el proveedor de JSSE 6 nativo de System i5:

El proveedor de JSSE nativo de System i5 ofrece la suite completa de clases e interfaces JSSE, incluidas las implementaciones de la clase KeyStore y la clase SSLConfiguration de JSSE.

Valores de protocolo para el método SSLContext.getInstance

En la tabla que sigue se identifican y describen los valores de protocolo del método SSLContext.getInstance del proveedor de JSSE nativo de System i5.

Los protocolos SSL soportados pueden estar limitados por los valores de sistema establecidos en su sistema. Encontrará más detalles en el subtema: Valores del sistema de seguridad: protocolos de la capa de sockets segura (SSL), en la información de gestión de sistemas.

Valor de protocolo	Protocolos SSL soportados
SSL	SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.
SSLv2	SSL versión 2
SSLv3	Protocolo SSL versión 3. Aceptará hello SSLv3 encapsulado en un hello de formato SSLv2.
TLS	Protocolo TLS versión 1, definido en la petición de comentarios (RFC) 2246. Aceptará hello TLSv1 encapsulado en un hello de formato SSLv2.
TLSv1	Protocolo TLS versión 1, definido en la petición de comentarios (RFC) 2246. Aceptará hello TLSv1 encapsulado en un hello de formato SSLv2.
SSL_TLS	SSL versión 3 y TLS versión 1. Aceptará hello SSLv3 o TLSv1 encapsulado en un hello de formato SSLv2.

Implementaciones de KeyStore nativo de System i5

El proveedor nativo de System i5 ofrece dos implementaciones de la clase KeyStore, que son IbmISeriesKeyStore o IBMi5OSKeyStore. Ambas implementaciones de KeyStore (almacén de claves) proporcionan una envoltura alrededor del soporte de gestor de certificados digitales (DCM).

IbmISeriesKeyStore
El contenido del almacén de claves se basa en un identificador de aplicación específico o un archivo de claves, una contraseña y una etiqueta. JSSE carga las entradas del almacén de claves del gestor de certificados digitales. Para cargar las entradas, JSSE utiliza el identificador de aplicación adecuado o la información del archivo de claves cuando la aplicación intenta por primera vez acceder a las entradas del almacén de claves o la información del archivo de claves. No es posible modificar el almacén de claves y todos los cambios de configuración deben efectuarse mediante el Gestor de certificados digitales.

IBMi5OSKeyStore
El contenido de este almacén de claves se basa en un archivo de almacén de certificados i5OS y la contraseña para acceder a ese archivo. Esta clase KeyStore permite modificar el almacén de certificados. Podrá hacer cambios sin tener que utilizar el gestor de certificados digitales (DCM).

La implementación de IBMi5OSKeyStore está en conformidad con la especificación de Sun Microsystems, Inc., para la API Java KeyStore. Encontrará más detalles en la información Javadoc de Keystore de Sun Microsystems, Inc.

Para obtener más información sobre cómo gestionar almacenes de claves mediante el DCM, vea el tema Gestor de certificados digitales (DCM).

Información relacionada

Valores del sistema de seguridad: protocolos de la capa de sockets segura (SSL)

Información Javadoc de la clase i5OSLoadStoreParameter:

com.ibm.i5os.keystore

Clase i5OSLoadStoreParameter

```
java.lang.Object
  |
  +--com.ibm.i5os.keystore.i5OSLoadStoreParameter
```

Todas las interfaces implementadas:

java.security.Keystore.LoadStoreParameter

```
public class i5OSLoadStoreParameter
extends java.lang.Object
implements java.security.Keystore.LoadStoreParameter
```

Esta clase crea un objeto KeyStore.ProtectionParameter que sirve para cargar/almacenar almacenes de certificados de i5OS. Una vez creado, la clase proporciona información sobre el almacén de certificados al que hay que acceder y sobre la contraseña empleada para proteger el almacén de certificados.

Ejemplo de uso de esta clase:

```
//inicializar el almacén de claves (keystore)
    KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");

 //Cargar un almacén de claves existente
    File kdbFile = new File("/tmp/certificateStore.kdb");
    i5OSLoadStoreParameter lsp =
   new i5OSLoadStoreParameter (kdbFile, "password".toCharArray());
    ks.load(lsp);

 //Obtener y añadir entradas al almacén de certificados
    ...

 //Guardar el almacén de certificados
    Ks.store(lsp);
```

Desde:
SDK 1.5

Resumen del constructor

i5OSLoadStoreParameter(java.io.File ksFile, char[] password)

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la contraseña que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

i5OSLoadStoreParameter(java.io.File ksFile, java.security.KeyStore.PasswordProtection pwdProtParam)

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la PasswordProtection que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

Tabla 16. **Resumen de los métodos**

java.security.KeyStore. ProtectionParameter	*"getProtectionParameter" en la página 331*() Devuelve el objeto KeyStore.KeyStoreParameter ascociado a este LoadStoreParameter

---

Métodos heredados de la clase java.lang.Object

| clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Detalles del constructor

i5OSLoadStoreParameter

```
public i5OSLoadStoreParameter(java.io.File ksFile,
                              char[] password)
                       throws java.lang.IllegalArgumentException
```

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la contraseña que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

Parámetros:

`ksFile` - Objeto Archivo del almacén de claves.

Si se ha utilizado `keystore.load()` con un método `i5OSLoadStoreParameter(ksFile = null, password)`, se crea un almacén de claves nuevo.

Si se ha utilizado `keystore.store()` con un método `i5OSLoadStoreParameter(ksFile = null, password)`, se lanza una IllegalArgumentException.

`password` - Contraseña para acceder al almacén de certificados de i5OS. No puede ser null ni está vacía.

Lanza:

`java.lang.IllegalArgumentException` - Si la contraseña es nula o está vacía

---

i5OSLoadStoreParameter

```
public i5OSLoadStoreParameter(java.io.File ksFile,
                              java.security.KeyStore.PasswordProtection pwdProtParam)
                       throws java.lang.IllegalArgumentException
```

Crea una instancia de ProtectionParameter a partir del archivo de almacén de claves (KeyStore) y la PasswordProtection que hay que utilizar para cargar/almacenar un almacén de certificados de i5OS.

Si se ha utilizado `keystore.load()` con un método `i5OSLoadStoreParameter(ksFile = null, password)`, se crea un almacén de claves nuevo.

Si se ha utilizado `keystore.store()` con un método `i5OSLoadStoreParameter(ksFile = null, password)`, se lanza una IllegalArgumentException.

Parámetros:

`ksFile` - Objeto Archivo del almacén de claves.

`pwdProtParam` - Instancia de PasswordProtection que se empleará para adquirir la contraseña. No puede ser null.

Lanza:

`java.lang.IllegalArgumentException` - Si KeyStore.PasswordProtection es null, o si la contraseña contenida en pwdProtParam es null o está vacía.

Detalles del método

---

getProtectionParameter

```
public java.security.KeyStore.ProtectionParameter getProtectionParameter()
```

Devuelve el objeto KeyStore.KeyStoreParameter ascociado a este LoadStoreParameter.

Especificado por:

```
getProtectionParameter in interface java.security.KeyStore.LoadStoreParameter
```

Devuelve:

Una instancia que implementa la interfaz KeyStore.KeyStoreParameter

Vea también:

```
java.security.KeyStore.ProtectionParameter#getProtectionParameter()
```

Información Javadoc de la clase i5OSSystemCertificateStoreFile:

com.ibm.i5os.keystore

Class i5OSSystemCertificateStoreFile

```
java.lang.Object
   java.io.File
       com.ibm.i5os.keystore.i5OSSystemCertificateStoreFile
```

Todas las interfaces implementadas:

java.io.Serializable, java.lang.Comparable<java.io.File>

```
public class i5OSSystemCertificateStoreFile
extends java.io.File
```

Esta clase proporciona una nueva implementación de archivo que señala hacia el archivo de almacén de certificados *SYSTEM. Proporciona un mecanismo para que el usuario cargue el almacén de certificados *SYSTEM sin obligarle a saber cuál es la vía real de acceso al almacén.

Para cargar el almacén de certificados *SYSTEM en un almacén de claves, cree primero un i5OSSystemCertificateStoreFile.

A partir de aquí, el almacén de claves se puede cargar de dos maneras:

- Mediante un i5OSLoadStoreParameter:

```
//crear un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();

  //utilizar ese archivo para crear un i5OSLoadStoreParameter
     i5OSLoadStoreParameter lsp = new i5OSLoadStoreParameter(starSystemFile, pwd);

  //cargar el almacén de certificados en un almacén de claves
     KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
     ks.load(lsp);
```

- Mediante una FileInputStream:

```
//crear un i5OSSystemCertificateStoreFile
     File starSystemFile = new i5OSSystemCertificateStoreFile();
```

```
//crear una corriente de entrada al starSystemFile
   FileInputStream fis = new FileInputStream(starSystemFile);

//cargar el almacén de certificados en un almacén de claves
   KeyStore ks = KeyStore.getInstance("IBMi5OSKeyStore");
   ks.load(fis, pwd);
```

Desde:
SDK 1.5

Vea también:
Forma serializada

Resumen de los campos

Campos heredados de la clase java.io.File
pathSeparator, pathSeparatorChar, separator, separatorChar

Resumen del constructor

i5OSSystemCertificateStoreFile()

Crea un File() que señala hacia el archivo de almacén de certificados *System.

Resumen de los métodos

Métodos heredados de la clase java.io.File
canRead, canWrite, compareTo, createNewFile, createTempFile, createTempFile, delete, deleteOnExit, equals, exists, getAbsoluteFile, getAbsolutePath, getCanonicalFile, getCanonicalPath, getName, getParent, getParentFile, getPath, hashCode, isAbsolute, isDirectory, isFile, isHidden, lastModified, length, list, list, listFiles, listFiles, listFiles, listRoots, mkdir, mkdirs, renameTo, setLastModified, setReadOnly, toString, toURI, toURL

Métodos heredados de la clase java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Detalles del constructor

i5OSSystemCertificateStoreFile

public **i5OSSystemCertificateStoreFile**()

Crea un File() que señala hacia el archivo de almacén de certificados *System.

Información Javadoc de SSLConfiguration para la versión 6:

com.ibm.i5os.jsse

Clase SSLConfiguration

```
java.lang.Object
  |
  +--com.ibm.i5os.jsse.SSLConfiguration
```

Todas las interfaces implementadas:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

public final class **SSLConfiguration**
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable

Esta clase corresponde a la especificación de la configuración necesaria para la implementación JSSE nativa de System i5.

La implementación JSSE nativa de System i5 funciona de manera más eficaz con un objeto `KeyStore` de tipo ″IbmISeriesKeyStore″. Este tipo de objeto `KeyStore` contiene entradas de clave y entradas de certificado de confianza basadas ya sea en un identificador de aplicación registrado en el gestor de certificado digital (DCM) o en un archivo de conjunto de claves (contenedor de certificados digitales). Luego, un objeto `KeyStore` de este tipo puede servir para inicializar un objeto `X509KeyManager` y un objeto `X509TrustManager` del `Provider` ″IBMi5OSJSSEProvider″. Después, los objetos `X509KeyManager` y `X509TrustManager` se pueden usar para inicializar un objeto `SSLContext` del ″IBMi5OSJSSEProvider″. Entonces, el objeto `SSLContext` proporciona acceso a la implementación JSSE nativa de System i5 basándose en la información de configuración especificada para el objeto `KeyStore`. Cada vez que se realiza una carga en un `KeyStore` de ″IbmISeriesKeyStore″, el `KeyStore` se inicializa basándose en la configuración actual especificada por el identificador de la aplicación o el archivo de conjunto de claves.

Esta clase también puede utilizarse para generar un objeto `KeyStore` de cualquier tipo válido. El `KeyStore` se inicializa basándose en la configuración actual especificada por el identificador de la aplicación o el archivo de conjunto de claves. Para realizar cualquier cambio en la configuración especificada por un identificador de aplicación o por un archivo de conjunto de claves, es necesario regenerar el objeto `KeyStore`. Observe que se tiene que especificar un contraseña de conjunto de claves (para el almacén de certificados *SYSTEM cuando se utiliza el identificador de la aplicación) para poder crear con éxito otro tipo de `KeyStore` diferente a ″IbmISeriesKeyStore″. La contraseña de conjunto de claves debe especificarse para conseguir el acceso a cualquier clave privada de cualquier `KeyStore` de tipo ″IbmISeriesKeyStore″.

Desde:
SDK 1.5

Consulte también:
KeyStore, X509KeyManager, X509TrustManager, SSLContext

Resumen del constructor

`SSLConfiguration()` crea una nueva SSLConfiguration. Consulte “Detalles del constructor” en la página 334 para obtener más información.

Tabla 17. **Resumen de los métodos**

`void`	“clear” en la página 337() Borra toda la información en el objeto para que todos los métodos get devuelvan un valor nulo.
`java.lang.Object`	“clone” en la página 339() Genera una nueva copia de esta configuración SSL.
`boolean`	“equals” en la página 338(java.lang.Objectobj) Indica si algún otro objeto es ″igual a″ este.
`protected void`	“finalize” en la página 337() llamado por el recogedor de basura de un objeto cuando la recogida de basura determina que no hay más referencias al objeto.
`java.lang.String`	“getApplicationId” en la página 337() Devuelve el ID de la aplicación.
`java.lang.String`	“getKeyringLabel” en la página 337() Devuelve la etiqueta de conjunto de claves.

Tabla 17. **Resumen de los métodos** *(continuación)*

`java.lang.String`	“getKeyringName” en la página 337() Devuelve el nombre de conjunto de claves.
`char[]`	“getKeyringPassword” en la página 337() Devuelve la contraseña de conjunto de claves.
`java.security.KeyStore`	“getKeyStore” en la página 339(char[]password) Devuelve un almacén de claves de tipo ″IbmISeriesKeyStore″ utilizando la contraseña especificada.
`java.security.KeyStore`	“ getKeyStore” en la página 339(java.lang.Stringtype, char[]password) Devuelve un almacén de claves del tipo solicitado utilizando la contraseña especificada.
`int`	“hashCode” en la página 339() Devuelve un valor de código hash para el objeto.
`staticvoid`	(java.lang.String[]args) Ejecuta las funciones de SSLConfiguration.
`void`	(java.lang.String[]args, java.io.PrintStreamout) Ejecuta las funciones de SSLConfiguration.
`void`	“setApplicationId” en la página 338(java.lang.StringapplicationId) Establece el identificador de aplicación.
`void`	“setApplicationId” en la página 338(java.lang.StringapplicationId, char[]password) Establece el identificador de la aplicación y la contraseña del conjunto de claves.
`void`	“setKeyring” en la página 338(java.lang.Stringname,java.lang.Stringlabel, char[]password) Establece la información de conjunto de claves.

Métodos heredados de la clase java.lang.Object
`getClass, notify, notifyAll, toString, wait, wait, wait`

Detalles del constructor

SSLConfiguration

```
public SSLConfiguration()
```

Crea una nueva SSLConfiguration. El identificador de aplicación y la información del conjunto de claves se inicializa con valores de omisión.

El valor predeterminado para el identificador de aplicación es el valor especificado para la propiedad ″os400.secureApplication″.

Los valores predeterminados para la información de conjunto de claves son igual a nulo si se especifica la propiedad ″os400.secureApplication″. Si no se especifica la propiedad ″os400.secureApplication″, el valor para el nombre del conjunto de claves es el valor especificado para la propiedad ″os400.certificateContainer″. Si no se especifica la propiedad ″os400.secureApplication″, la etiqueta de conjunto de claves se inicializa con el valor de la ″os400.certificateLabel″. Si no se establece la propiedad ″os400.secureApplication″ ni la ″os400.certificateContainer″, el nombre de conjunto de claves se inicializará con ″*SYSTEM″.

Detalles del método

---

main

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

Ejecuta las funciones de SSLConfiguration. Hay cuatro mandatos que se pueden ejecutar: -help, -create, -display y -update. El mandato debe ser el primer parámetro especificado.

Las opciones que se pueden especificar son las siguientes (en cualquier orden):

-keystore **nombre-archivo-almacén-claves**
: Especifica el nombre del archivo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para todos los mandatos.

-storepass **contraseña-archivo-almacén-claves**
: Especifica la contraseña asociada con el archivo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para todos los mandatos.

-storetype **tipo-almacén-claves**
: Especifica el tipo de almacén de claves para crearlo, actualizarlo o visualizarlo. Esta opción es necesaria para cualquier mandato. Si no se especifica esta opción, se utiliza un valor de "IbmISeriesKeyStore".

-varname>-appid identificador-aplicación
: Especifica el identificador de aplicación que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-keyring **nombre-archivo-conjunto-claves**
: Especifica el nombre de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-keyringpass **contraseña-archivo-conjunto-claves**
: Especifica la contraseña de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Puede especificarse esta opción para los mandatos *-create* y *-update* y es necesaria cuando se especifica un tipo de almacén de claves distinto de "IbmISeriesKeyStore". Si no se especifica esta opción, se utiliza la contraseña oculta de conjunto de claves.

-keyringlabel **etiqueta-archivo-conjunto-claves**
: Especifica la etiqueta de archivo de conjunto de claves que se debe utilizar para inicializar un archivo de almacén de claves que se esté creando o actualizando. Esta opción solo se especifica cuando también se especifica la opción *-keyring*. Si no es especifica esta opción cuando se especifica la opción *keyring*, se utiliza la etiqueta por omisión en el conjunto de claves.

-systemdefault
: Especifica el valor predeterminado del sistema que se utiliza para inicializar un almacén de claves que se esté creando o actualizando. Esta opción es opcional para los mandatos *-create* y *-update*. Solo se puede especificar una de las opciones *-appid*, *keyring* y *-systemdefault*.

-v
: Especifica que se debe generar salida verbosa. Esta opción se puede especificar para cualquier mandato.

El mandato ayuda visualiza la información de uso para especificar los parámetros de este método. Los parámetros para invocar la función de ayuda se especifican de esta manera:

```
    -help
```

El mandato create crea un nuevo archivo de almacén de datos. Existen tres variantes del mandato create. Una variante para crear un almacén de claves basado en un identificador de aplicación particular, otra variante para crear un almacén de claves basado en un nombre, una etiqueta y una contraseña de conjunto de claves, y la tercera variante para crear un conjunto de claves basado en la configuración por omisión del sistema.

Para crear un almacén de claves basado en un identificador de aplicación particular, se debe especificar la opción *-appid*. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en el identificador de aplicación ″APPID″:

```
    -create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
            -appid APPID
```

Para crear un almacén de claves basado en un archivo de conjunto de claves particular, se debe especificar la opción *-keyring*. Las opciones *-keyringpass* y *keyringlabel* también pueden especificarse. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en el archivo de almacén de claves ″keyring.file″, basándose en la contraseña ″ringpass″ y en la etiqueta de conjunto de claves ″keylabel″:

```
    -create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
            -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

Para crear un almacén de claves basado en la configuración por omisión del sistema, se debe especificar la opción-*systemdefault*. Los siguientes parámetros crearían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″ que se inicializaría basándose en la configuración por omisión del sistema :

```
    -create -keystore keystore.file -storepass keypass -systemdefault
```

El mandato update actualiza un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″. Existen tres variantes del mandato update que son idénticas a las variantes del mandato create. Las opciones del mandato update son idénticas a las opciones utilizadas para el mandato create. El mandato display visualiza la configuración especificada de un archivo de almacén de claves existente. Los siguientes parámetros visualizarían un archivo de almacén de claves de tipo ″IbmISeriesKeyStore″ de nombre ″keystore.file″ con una contraseña ″keypass″:

```
    -display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

Parámetros:

`args` - los argumentos de la línea de mandatos

---

run

```
public void run(java.lang.String[]args,
                java.io.PrintStreamout)
```

Ejecuta las funciones de SSLConfiguration. Los parámetros y la funcionalidad de este método son idénticos a los del método main().

Parámetros:

`args` - los argumentos de mandato

`out` - la corriente de salida en que se deben escribir los resultados

Vea también: `com.ibm.i5os.jsse.SSLConfiguration.main()`

---

getApplicationId

```
public java.lang.String getApplicationId()
```

Devuelve el ID de la aplicación.

Devuelve:
el ID de la aplicación.

---

getKeyringName

```
public java.lang.String getKeyringName()
```

Devuelve el nombre de conjunto de claves.

Devuelve:
el nombre de conjunto de claves.

---

getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

Devuelve la etiqueta de conjunto de claves.

Devuelve:
la etiqueta de conjunto de claves.

---

getKeyringPassword

```
public final char[] getKeyringPassword()
```

Devuelve la contraseña de conjunto de claves.

Devuelve:
la contraseña de conjunto de claves.

---

finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Llamado por el recogedor de basura de un objeto cuando la recogida de basura determina que no hay más referencias al objeto.

Alteraciones temporales:
`finalize` en la clase `java.lang.Object`

Lanza: `java.lang.Throwable` - la excepción lanzada por este método.

---

clear

```
public void clear()
```

Borra toda la información en el objeto para que todos los métodos get devuelvan un valor nulo.

---

setKeyring

```
public void setKeyring(java.lang.Stringname,
                       java.lang.Stringlabel,
                       char[]password)
```

Establece la información de conjunto de claves.

Parámetros:

`name` - el nombre de conjunto de claves

`label` - la etiqueta de conjunto de claves, o el valor nulo si se utiliza la entrada de conjunto de claves por omisión.

`password` - la contraseña de conjunto de claves, o el valor nulo si se utiliza la contraseña oculta.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId)
```

Establece el ID de la aplicación.

Parámetros:

`applicationId` - el ID de la aplicación.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId,
                             char[]password)
```

Establece el ID de la aplicación y la contraseña del conjunto de claves. Al especificar la contraseña de conjunto de claves se permite cualquier conjunto de claves creado para permitir el acceso a la clave privada.

Parámetros:

`applicationId` - el ID de la aplicación.

`password` - la contraseña de conjunto de claves.

---

equals

```
public boolean equals(java.lang.Objectobj)
```

Indica si algún otro objeto es ″igual a″ este.

Alteraciones temporales:

`equals` de la clase `java.lang.Object`

Parámetros:

`obj` - objeto para comparar

Devuelve:

el indicador de si los objetos especifican la misma información de configuración

---

hashCode

```
public int hashCode()
```

Devuelve un valor de código hash para el objeto.

Alteraciones temporales:
hashCode de la clase java.lang.Object

Devuelve:
un valor de código hash para el objeto.

---

clone

```
public java.lang.Object clone()
```

Genera una nueva copia de esta configuración SSL. Los cambios posteriores en los componentes de esta configuración no afectarán a la copia nueva y viceversa.

Alteraciones temporales:
clone de la clase java.lang.Object

Devuelve:
una copia de esta configuración SSL

---

getKeyStore

```
public java.security.KeyStore getKeyStore(char[]password)
                                  throws java.security.KeyStoreException
```

Devuelve un almacén de claves de tipo "IbmISeriesKeyStore" utilizando la contraseña especificada. Se inicializa el almacén de claves basándose en la información de configuración actual almacenada en el objeto.

Parámetros:
password - se utiliza para inicializar el almacén de claves

Devuelve:
el almacén de claves KeyStore inicializado basándose en la información de configuración actual almacenada en el objeto.

Lanza: java.security.KeyStoreException - si no se puede crear el almacén de claves

---

getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                           char[]password)
                                  throws java.security.KeyStoreException
```

Devuelve un almacén de claves del tipo solicitado utilizando la contraseña especificada. Se inicializa el almacén de claves basándose en la información de configuración actual almacenada en el objeto.

Parámetros:
type - tipo de almacén de claves a devolver

password - se utiliza para inicializar el almacén de claves

Devuelve:
el almacén de claves KeyStore inicializado basándose en la información de configuración actual almacenada en el objeto.

Lanza: `java.security.KeyStoreException` - si no se puede crear el almacén de claves

Ejemplos: IBM Java Secure Sockets Extension 6:

Los ejemplos de JSSE muestran cómo pueden un cliente y un servidor emplear el proveedor de JSSE nativo de System i5 para crear un contexto que habilite las comunicaciones seguras.

Nota: Ambos ejemplos utilizan el proveedor de JSSE nativo de System i5, sean cuales sean las propiedades especificadas por el archivo java.security.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

Ejemplo: cliente SSL que utiliza un objeto SSLContext de la versión 6:

Este programa cliente de ejemplo utiliza un objeto SSLContext, que inicializa para emplear el ID de aplicación ″MY_CLIENT_APP″. Este programa utilizará la implementación nativa de System i5 con independencia de lo que se haya especificado en el archivo java.security.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////////////////////////////////////////
//
// Este programa cliente de ejemplo utiliza un objeto SSLContext, que inicializa
// para emplear el ID de aplicación "MY_CLIENT_APP".
//
// El ejemplo usa el proveedor de JSSE nativo de System i5, sean cuales sean
// las propiedades especificadas por el archivo java.security.
//
// Sintaxis de mandato:
//     java -Djava.version=1.6 SslClient
//
// Fíjese en que "-Djava.version=1.6" es innecesario si se ha configurado
// que hay que usar JDK versión 6 por defecto.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
import com.ibm.i5os.jsse.SSLConfiguration;
/**
 * Programa cliente SSL.
 */
public class SslClient {

   /**
    * Método main de SslClient.
    *
    * @param args argumentos de línea de mandatos (no se utiliza)
    */
   public static void main(String args[]) {
       /*
        * Configurar para capturar las excepciones lanzadas.
        */
       try {
           /*
            * Inicializar un objeto SSLConfiguration para especificar un ID de
            * aplicación.  MY_CLIENT_APP se debe registrar y configurar
```

```
| * correctamente con el Gestor de certificados digitales (DCM).
| */
| SSLConfiguration config = new SSLConfiguration();
| config.setApplicationId("MY_CLIENT_APP");
| /*
| * Obtener un objeto KeyStore del objeto SSLConfiguration.
| */
| char[] password = "password".toCharArray();
| KeyStore ks = config.getKeyStore(password);
| /*
| * Asignar e inicializar una KeyManagerFactory.
| */
| KeyManagerFactory kmf =
|                   KeyManagerFactory.getInstance("IbmISeriesX509");
| kmf.init(ks, password);
| /*
| * Asignar e inicializar una TrustManagerFactory.
| */
| TrustManagerFactory tmf =
|                  TrustManagerFactory.getInstance("IbmISeriesX509");
| tmf.init(ks);
| /*
| * Asignar e inicializar un SSLContext.
| */
| SSLContext c =
|            SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
| c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
| /*
| * Obtener una SSLSocketFactory del SSLContext.
| */
| SSLSocketFactory sf = c.getSocketFactory();
| /*
| * Crear un SSLSocket.
| *
| * Cambiar la dirección IP codificada por programa por la dirección IP o el
| * nombre de host del servidor.
| */
| SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
| /*
| * Enviar un mensaje al servidor mediante la sesión segura.
| */
| String sent = "Probar la escritura SSL java";
| OutputStream os = s.getOutputStream();
| os.write(sent.getBytes());
| /*
| * Escribir los resultados en la pantalla.
| */
| System.out.println("Escritos " + sent.length() + " bytes...");
| System.out.println(sent);
| /*
| * Recibir un mensaje del servidor mediante la sesión segura.
| */
| InputStream is = s.getInputStream();
| byte[] buffer = new byte[1024];
| int bytesRead = is.read(buffer);
| if (bytesRead == -1)
|     throw new IOException("Recibido fin de archivo inesperado");
| String received = new String(buffer, 0, bytesRead);
| /*
| * Escribir los resultados en la pantalla.
| */
| System.out.println("Leídos " + received.length() + " bytes...");
| System.out.println(received);
| } catch (Exception e) {
| System.out.println("Excepción inesperada: "+;
|                    e.getMessage());
| e.printStackTrace();
```

```
        }
    }

}
```

Ejemplo: servidor SSL que utiliza un objeto SSLContext de la versión 6:

El siguiente programa servidor utiliza un objeto SSLContext que inicializa con un archivo de almacén de claves creado anteriormente.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////////////////////////////////////////
//
// El siguiente programa servidor utiliza un objeto SSLContext que
// inicializa con un archivo de almacén de claves creado anteriormente.
//
// El archivo de almacén de claves tiene el nombre y la contraseña siguientes:
//   Nombre de archivo: /home/keystore.file
//   Contraseña: password
//
// El programa de ejemplo necesita el archivo de almacén de claves para crear un
// objeto IbmISeriesKeyStore. El objeto KeyStore debe especificar MY_SERVER_APP como
// identificador de la aplicación.
//
// Para crear el archivo de almacén de claves, puede emplear el siguiente mandato de Qshell:
//
//   java com.ibm.i5os.SSLConfiguration  -create  -keystore /home/keystore.file
//     -storepass password  -appid MY_SERVER_APP
//
// Sintaxis de mandato:
//     java -Djava.version=1.6 JavaSslServer
//
// Fíjese en que "-Djava.version=1.6" es innecesario si se ha configurado
// que hay que usar JDK versión 6 por defecto.
//
// También puede crear el archivo de almacén de claves entrando este mandato en un indicador de mandatos de i5/OS:
//
//  RUNJVA CLASS(com.ibm.i5os.SSLConfiguration) PARM('-create' '-keystore'
//      '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;
import java.security.*;
/**
* Programa servidor Java SSL que utiliza el ID de aplicación.
*/
public class JavaSslServer {

    /**
     * Método main de JavaSslServer.
     *
     * @param args argumentos de línea de mandatos (no se utiliza)
     */
    public static void main(String args[]) {
        /*
         * Configurar para capturar las excepciones lanzadas.
         */
        try {
            /*
             * Asignar e inicializar un objeto KeyStore.
             */
            char[] password = "password".toCharArray();
```

```
|             KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
|             FileInputStream fis = new FileInputStream("/home/keystore.file");
|             ks.load(fis, password);
|             /*
|              * Asignar e inicializar una KeyManagerFactory.
|              */
|             KeyManagerFactory kmf =
|                             KeyManagerFactory.getInstance("IbmISeriesX509");
|             kmf.init(ks, password);
|             /*
|              * Asignar e inicializar una TrustManagerFactory.
|              */
|             TrustManagerFactory tmf =
|                           TrustManagerFactory.getInstance("IbmISeriesX509");
|             tmf.init(ks);
|             /*
|              * Asignar e inicializar un SSLContext.
|              */
|             SSLContext c =
|                       SSLContext.getInstance("SSL", "IBMi5OSJSSEProvider");
|             c.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
|             /*
|              * Obtener una SSLServerSocketFactory del SSLContext.
|              */
|             SSLServerSocketFactory sf = c.getServerSocketFactory();
|             /*
|              * Crear un SSLServerSocket.
|              */
|             SSLServerSocket ss =
|                 (SSLServerSocket) sf.createServerSocket(13333);
|             /*
|              * Efectuar un accept() para crear un SSLSocket.
|              */
|             SSLSocket s = (SSLSocket) ss.accept();
|             /*
|              * Recibir un mensaje del cliente mediante la sesión segura.
|              */
|             InputStream is = s.getInputStream();
|             byte[] buffer = new byte[1024];
|             int bytesRead = is.read(buffer);
|             if (bytesRead == -1)
|                 throw new IOException("Recibido fin de archivo inesperado");
|             String received = new String(buffer, 0, bytesRead);
|             /*
|              * Escribir los resultados en la pantalla.
|              */
|             System.out.println("Leídos " + received.length() + " bytes...");
|             System.out.println(received);
|             /*
|              * Volver a enviar como un eco el mensaje al cliente mediante la sesión segura.
|              */
|             OutputStream os = s.getOutputStream();
|             os.write(received.getBytes());
|             /*
|              * Escribir los resultados en la pantalla.
|              */
|             System.out.println("Escritos " + received.length() + " bytes...");
|             System.out.println(received);
|         } catch (Exception e) {
|             System.out.println("Excepción inesperada: "+;
|                              e.getMessage());
|             e.printStackTrace();
|         }
|     }
|
|  }
```

Servicio de autenticación y autorización Java

El servicio de autenticación y autorización Java (JAAS) es una extensión estándar de la plataforma Java 2, Standard Edition (J2SE). J2SE proporciona controles de acceso que se basan en dónde se originó el código y en quién lo firmó (controles de acceso basados en el origen del código). Sin embargo, a J2SDK le falta la capacidad de forzar controles de acceso adicionales basados en quién ejecuta el código. JAAS proporciona una infraestructura que añade este soporte al modelo de seguridad de Java 2.

La implementación de JAAS en System i5 es compatible con la implementación de Sun Microsystems, Inc. Esta documentación cubre los aspectos exclusivos de la implementación de System i5. Se presupone que está usted familiarizado con la documentación general de las extensiones JAAS. Para que le resulta más fácil trabajar con esa información y con la nuestra sobre System i5, le proporcionamos estos enlaces.

Información relacionada

Javadoc JAAS específico del servidor System i5

Especificación de la API de JAAS
Contiene información Javadoc sobre JAAS.

LoginModule de JAAS
Se centra en los aspectos de autenticación de JAAS.

Preparar y configurar un System i5 para el servicio de autenticación y autorización Java (JAAS)

Debe cumplir los requisitos de software y configurar el System i5 para que utilice el servicio de autenticación y autorización Java (JAAS).

Requisitos de software para ejecutar JAAS 1.0 en un System i5

Instale los siguientes programas bajo licencia:

- Java 2 SDK, versión 1.4 (J2SDK)o superiores
- Para cambiar la identidad de la hebra de OS, se necesita el programa bajo licencia IBM Toolbox para Java (mod 4, 5761-JC1). Contiene las clases ProfileTokenCredential necesarias para poder cambiar la identidad de hebra de OS de System i5 y las clases de implementación nativas.

Configurar el sistema

Para configurar el sistema con objeto de que utilice JAAS, siga estos pasos:

1. En ${java.home}/lib/security se proporciona un archivo login.config por omisión que invoca com.ibm.as400.security.auth.login.BasicAuthenticationLoginModule. Este archivo login.config conecta al sujeto autenticado una credencial ProfileTokenCredential de un solo uso. Si desea emplear un archivo login.config propio que tenga distintas opciones, puede incluir la siguiente propiedad del sistema cuando invoque su aplicación:

   ```
   -Djava.security.auth.login.config=su archivo login.config
   ```

2. Añada al directorio de extensiones (ext) un enlace simbólico para el archivo jt400Native.jar. Así el cargador de clases de extensión podrá cargar este archivo.

   El hecho de enlazar simbólicamente el archivo jt400Native.jar con el directorio /QIBM/ProdData/Java400/jdk14/lib/ext obliga a todos los usuarios de J2SDK 1.4 del servidor a ejecutarse con esta versión de jt400Native.jar. Esto puede no ser conveniente si diversos usuarios necesitan distintas versiones de las clases de IBM Toolbox para Java. Existe la alternativa de colocar el archivo jt400Native.jar en la CLASSPATH de la aplicación, como ya se ha indicado. Aún hay otra opción, que consiste en añadir el enlace simbólico a su propio directorio y luego incluir ese directorio en la vía de acceso de clases del directorio de extensiones, especificando la propiedad java.ext.dirs del sistema en el momento de invocar la aplicación.

   Para enlazar el archivo jt400Native.jar al directorio /QIBM/ProdData/Java400/jdk14/lib/ext, ejecute este mandato en la línea de mandatos de i5/OS y así añadir el enlace:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
     NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')
```

Para enlazar el archivo jt400Native.jar a su propio directorio, haga lo siguiente:

a. Para añadir el enlace, ejecute este mandato en la línea de mandatos de i5/OS:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
  NEWLNK('su directorio de extensiones/jt400Native.jar')
```

b. Cuando llame al programa Java, emplee este patrón:

```
java -Djava.ext.dirs=su directorio de extensiones:directorios de
extensiones por omisión
```

Nota: En IBM Toolbox para Java encontrará información sobre las clases de credenciales de System i5. Pulse **Clases de seguridad**. Pulse **Servicios de autenticación**. Pulse la clase **ProfileTokenCredential**. Pulse **Paquete**.

3. Actualice los archivos de política de Java 2 para otorgar los debidos permisos sobre las ubicaciones reales de los archivos JAR de IBM Toolbox para Java. Aunque estos archivos pueden estar simbólicamente enlazados a los directorios de extensiones y a estos directorios se les otorgue java.security.AllPermission en el archivo ${java.home}/lib/security/java.policy, la autorización se basa en la ubicación real de los archivos JAR.

   Para utilizar satisfactoriamente las clases de credenciales de IBM Toolbox para Java, añada este fragmento de código al archivo de política Java 2 de su aplicación:

```
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";
 }
```

   También tendrá que añadir estos permisos para codeBase de la aplicación, ya que las operaciones efectuadas por los archivos JAR de IBM Toolbox para Java no se ejecutan en modalidad privilegiada.

   En “Servicio de autenticación y autorización Java (JAAS)” hallará información relacionada con los archivos de política de Java 2.

4. Asegúrese de que los servidores de hospedaje de System i5 se han iniciado y están funcionando. Las clases ProfileTokenCredential que residen en Toolbox (por ejemplo, jt400Native.jar) se emplean como credenciales conectadas al sujeto autenticado. Las clases de credenciales necesitan acceder a los servidores de sistema principal. Para verificar que los servidores se han iniciado y están funcionando, teclee lo siguiente en el indicador de mandatos de i5/OS:

```
StrHostSVR *all
StrTcpSvr *DDM
```

   Si los servidores ya se habían iniciado, estos pasos no hacen nada. Si los servidores no se habían iniciado, lo harán ahora.

Servicio de autenticación y autorización Java (JAAS)

El servicio de autenticación y autorización Java (JAAS) es una extensión estándar del Java 2 Software Development Kit. Actualmente, Java 2 proporciona controles de acceso basados en el código fuente (controles de acceso basados en *dónde* se originó el código y en *quién firmó* el código). Sin embargo, carece de capacidad para aplicar controles de acceso adicionales basados en *quién ejecuta* el código. JAAS proporciona una infraestructura que añade este soporte al modelo de seguridad de Java 2.

Guía del desarrollador

- **Visión general**
- **Quién debe leer este documento**
- **Documentación relacionada**

- **Introducción**
- **Clases núcleo**
- **Clases comunes**
  - **Sujeto**
  - **Principales**
  - **Credenciales**
- **Clases de autenticación**
- **LoginContext**
- **LoginModule**
- **CallbackHandler**
- **Callback**
- **Clases de autorización**
- **Política**
- **AuthPermission**
- **PrivateCredentialPermission**

Referencias

- Implementación
- ¡"Hello World", estilo JAAS!
- Apéndice A: Valores de JAAS en el archivo de propiedades de seguridad java.security
- Apéndice B: Archivo de configuración de inicio de sesión
- Apéndice C: Archivo de política de autorización

Visión general

El servicio de autenticación y autorización Java (JAAS) es una extensión estándar del Java 2 Software Development Kit. Actualmente, Java 2 proporciona controles de acceso basados en el código fuente (controles de acceso basados en *dónde* se originó el código y en *quién firmó* el código). Sin embargo, carece de capacidad para aplicar controles de acceso adicionales basados en *quién ejecuta* el código. JAAS proporciona una infraestructura que añade este soporte al modelo de seguridad de Java 2.

La última actualización de este documento fue el 17 de marzo de 2000.

Quién debe leer este documento

Este documento va dirigido a programadores con experiencia que deseen crear aplicaciones restringidas por un modelo de seguridad basado en la fuente de código y en el sujeto.

Documentación relacionada

En este documento se presupone que ya se ha leído la documentación siguiente:

- Especificación de la API de Java 2 Software Development Kit
- Especificación de la API de JAAS
- La seguridad y la plataforma Java

El maual LoginModule Developer's Guide, suministrado por Sun Microsystems, Inc., es un suplemento de la presente guía.

Introducción

La infraestructura de JAAS puede dividirse en dos componentes principales: un componente de **autenticación** y un componente de **autorización**. El componente de autenticación de JAAS proporciona capacidad para determinar con fiabilidad y seguridad quién está procesando el código Java en ese momento, independientemente de si el código se ejecuta como una aplicación, un applet, un bean o un servlet. El componente de autorización de JAAS complementa la infraestructura de seguridad existente de Java 2, proporcionando los medios para evitar que se procese código Java para realizar tareas confidenciales, en función del origen del código (como se hace en Java 2) y en función de la persona que se autentique.

La autenticación JAAS se realiza de manera *pluggable*. Esto permite a las aplicaciones Java seguir siendo independientes de las tecnologías de autenticación subyacentes. Así, las tecnologías de autenticación nuevas o actualizadas pueden conectarse a una aplicación sin necesidad de realizar modificaciones en la propia aplicación. Las aplicaciones permiten el proceso de autenticación al crear una instancia de un

objeto
`LoginContext`

, el cual a su vez hace referencia a un objeto
`Configuration`

para determinar la tecnología de autenticación, o
`LoginModule`

, que hay que utilizar al realizar la autenticación. Los LoginModules típicos pueden solicitar un nombre de usuario y una contraseña y verificarlos. Existen otros que pueden leer muestras de huellas dactilares y de voz, que luego verifican.

Una vez autenticado el usuario que está procesando el código, el componente de autorización de JAAS trabaja conjuntamente con el modelo de control de acceso existente de Java 2 para proteger el acceso a recursos confidenciales. A diferencia de Java 2, donde las decisiones de control de acceso se basan solamente en la ubicación del código y en los firmantes del código (un
`CodeSource`

), en JAAS las decisiones de control de acceso se basan en el
`CodeSource`

del código de proceso, así como en el usuario que ejecuta el código, o el
`Subject`

. Tenga en cuenta que la política de JAAS tan solo amplía la política de Java 2 con la información relevante basada en Subject. Por lo tanto, los permisos que se reconocen y comprenden en Java 2 (por ejemplo,
`java.io.FilePermission`

y
`java.net.SocketPermission`

) también se reconocen y comprenden en JAAS. Además, aunque la política de seguridad de JAAS esté físicamente separada de la política de seguridad Java 2 existente, las dos políticas, juntas, forman una sola política lógica.

Clases núcleo

Las clases núcleo de JAAS pueden dividirse en 3 categorías: comunes, de autenticación y de autorización.

- Clases Comunes
  - Subject, Principals, Credentials
- Clases de autenticación
  - LoginContext, LoginModule, CallbackHandler, Callback
- Clases de autorización
  - Policy, AuthPermission, PrivateCredentialPermission

Clases Comunes

Las clases comunes se comparten en los ambos componentes, el de autenticación y el de autorización de JAAS.

La clase clave de JAAS es
`Subject`

, que representa un agrupamiento de información relacionada para una entidad individual, como puede ser una persona. Abarca los Principales, las credenciales públicas y las credenciales privadas de la entidad.

Tenga en cuenta que JAAS utiliza la interfaz
`java.security.Principal`

Java 2

existente para un Principal. Tenga también en cuenta que JAAS no introduce una interfaz o clase de credencial aparte. Una credencial, como se describe en JAAS, puede ser cualquier objeto.

Subject

Para autorizar el acceso a los recursos, las aplicaciones necesitan primero autenticar el origen de la petición. La infraestructura JAAS define el término Subject para representar el origen de una petición. El Subject puede ser cualquier entidad, como una persona o un servicio. Cuando se ha autenticado, el Subject se puebla con identidades asociadas, o Principals. Un Subject puede tener muchos Principals. Por ejemplo, una persona puede tener un nombre Principal (″John Doe″) y un SSN Principal (″123-45-6789″) lo que lo distingue de otros Subjects.

Un
`Subject`

también puede ser propietario de atributos relacionados con la seguridad, a los que llamamos credenciales. Las credenciales sensibles que requieren una protección especial (como las claves criptográficas privadas), se almacenan en un
`Set`

de credenciales privadas. Las credenciales que se comparten (como los certificados de clave pública o los tickets Kerberos) se almacenan en un
`Set`

de credenciales públicas. Se necesitan diferentes permisos para acceder y modificar los diferentes Sets de credenciales.

Los Subjects se crean con estos constructores:

```
public Subject();

public Subject(boolean readOnly, Set principals,
               Set pubCredentials, Set privCredentials);
```

El primer constructor crea un Subject con Sets de Principals vacíos (no nulos) y credenciales. El segundo constructor crea un Subject con los Sets de Principals y credenciales especificados. También tiene un argumento booleano que puede crear un Subject solo de lectura (Sets de Principales y credenciales inmutables).

Encontrará una forma alternativa de obtener una referencia para un Subject autenticado sin utilizar estos constructores en el apartado LoginContext.

Si no se ha creado instancia de un Subject en un estado solo de lectura, puede establecerse en un estado solo de lectura llamando a este método:

```
public void setReadOnly();
```

Se necesita un

```
AuthPermission("setReadOnly")
```

para invocar este método. Una vez que esté en estado solo de lectura, todo intento de añadir o eliminar Principals o credenciales puede hacer que se lance una excepción

```
IllegalStateException
```

.

Para probar el estado solo de lectura de un Subject, se puede llamar a este método:

```
public boolean isReadOnly();
```

Para recuperar los Principals asociados a un Subject, se dispone de dos métodos:

```
public Set getPrincipals();
public Set getPrincipals(Class c);
```

El primer método devuelve todo los Principals que contiene el Subject, mientras que el segundo método solo devuelve los Principals que son una instancia de la clase c especificada, o una instancia de una subclase de la clase c. Se devolverá un conjunto vacío si el Subject no tiene ningún Principal asociado.

Para recuperar las credenciales públicas asociadas a un Subject, se dispone de estos métodos:

```
public Set getPublicCredentials();
public Set getPublicCredentials(Class c);
```

El comportamiento observado de estos métodos es idéntico al del método

```
getPrincipals
```

.

Para acceder a las credenciales privadas asociadas a un Subject, se dispone de los siguientes métodos:

```
public Set getPrivateCredentials();
public Set getPrivateCredentials(Class c);
```

El comportamiento observado de estos métodos es idéntico al de los métodos

```
getPrincipals
```

y

getPublicCredentials

.

Para modificar o actuar sobre un conjunto Set de Principals de un Subject, un Set de credenciales públicas, o un Set de credenciales privadas, los llamadores utilizan los métodos definidos en la clase
java.util.Set

. El siguiente ejemplo es una muestra de ello:

```
Subject subject;
Principal principal;
Object credential;

// Añadir un Principal y una credencial al Subject
subject.getPrincipals().add(principal);
subject.getPublicCredentials().add(credential);
```

Tenga en cuenta que se necesita un
AuthPermission("modifyPrincipals")

,
AuthPermission("modifyPublicCredentials")

o
AuthPermission("modifyPrivateCredentials")

para modificar los conjuntos respectivos. Fíjese también en que solo los conjuntos devueltos por medio de los métodos
getPrincipals

,
getPublicCredentials

y
getPrivateCredentials

están respaldados por los conjuntos internos respectivos de Subject. Por lo tanto, cualquier modificación en los conjuntos devueltos afecta también a los conjuntos internos. Los conjuntos devueltos por medio de los métodos
getPrincipals(Class c)

,
getPublicCredentials(Class c)

y
getPrivateCredentials(Class c)

no están respaldados por los conjuntos internos respectivos de Subject. Se crea y se devuelve un conjunto nuevo por cada invocación de método. Las modificaciones en estos conjuntos no afectarán a los conjuntos internos de Subject. El método siguiente devuelve el Subject asociado al
AccessControlContext

especificado, o devuelve nulo si no hay un Subject asociado al
AccessControlContext

.

```
    public static Subject getSubject(final AccessControlContext acc);
```

Se necesita un
`AuthPermission("getSubject")`

para llamar a
`Subject.getSubject`

.

La clase Subject también incluye estos métodos heredados de
`java.lang.Object`

:

```
    public boolean equals(Object o);
    public String toString();
    public int hashCode();
```

Se puede llamar a los siguientes métodos estáticos para realizar el trabajo como si fuera un Subject particular:

```
    public static Object doAs(final Subject subject,
                        final java.security.PrivilegedAction action);

    public static Object doAs(final Subject subject,
                        final java.security.PrivilegedExceptionAction action)
                        throws java.security.PrivilegedActionException;
```

Ambos métodos asocian primero el ***subject*** especificado al
`AccessControlContext`

de la hebra actual y luego procesan la ***action***. Esto consigue el efecto de hacer que la ***action*** se ejecute como el ***subject***. El primer método puede lanzar excepciones de tiempo de ejecución, pero el procedimiento normal consiste en devolver un objeto desde el método run() de su argumento de acción. El segundo método se comporta de manera parecida, salvo que puede lanzar una excepción comprobada desde su método run() de
`PrivilegedExceptionAction`

. Se necesita un
`AuthPermission("doAs")`

para llamar a los métodos
`doAs`

.

Aquí encontrará dos ejemplos de utilización del primer método
`doAs`

. Supongamos que un
`Subject`

con un Principal de clase
`com.ibm.security.Principal`

que se llame ″BOB″ haya sido autenticado por un
`LoginContext`

″lc″. Supongamos asimismo que se ha instalado un SecurityManager, y que lo siguiente existe en la política de control de acceso de JAAS (vea la sección de política para obtener más detalles sobre el archivo de política de JAAS):

```
   // Otorgar el permiso "BOB" para leer el archivo "foo.txt"
   grant Principal com.ibm.security.Principal "BOB" {
      permission java.io.FilePermission "foo.txt", "read";
};
```

Ejemplo 1 de Subject.doAs

```
    class ExampleAction implements java.security.PrivilegedAction {
        public Object run() {
            java.io.File f = new java.io.File("foo.txt");

            // exists() invoca una control de seguridad
            if (f.exists()) {
                System.out.println("El archivo foo.txt existe.");
            }
            return null;
        }
    }

    public class Example1 {
        public static void main(String[] args) {

            // Autenticar el sujeto, "BOB".
            // Este proceso se describe en la
            // sección LoginContext.

            Subject bob;
            ...

            // ejecutar "ExampleAction" como "BOB":
            Subject.doAs(bob, new ExampleAction());
        }
    }
```

Durante el proceso,
`ExampleAction`

encontrará un control de seguridad cuando haga una llamada a
`f.exists()`

. Sin embargo, puesto que
`ExampleAction`

se ejecuta como ″BOB″, y dado que la política de JAAS (más arriba) otorga el permiso
`FilePermission`

necesario a ″BOB″, la acción
`ExampleAction`

pasará el control de seguridad.

El ejemplo 2 tiene el mismo escenario que el ejemplo 1.

Ejemplo 2 de Subject.doAs

```
public class Example2 {
    // Ejemplo de utilización de una clase de acción anónima.
    public static void main(String[] args) {
        // Autenticar el sujeto, "BOB".
        // Este proceso se describe en la
        // sección LoginContext.

        Subject bob;
        ...

        // ejecutar "ExampleAction" como "BOB":
        Subject.doAs(bob, new ExampleAction() {
            public Object run() {
                java.io.File f = new java.io.File("foo.txt");
                if (f.exists()) {
                    System.out.println("El archivo foo.txt existe.");
                }
                return null;
            }
        });
    }
}
```

Ambos ejemplos lanzan una excepción
`SecurityException`

si la declaración de concesión de permiso se altera incorrectamente, como al añadir un CodeBase incorrecto o al cambiar el Principal por ″MOE″. El hecho de eliminar el campo Principal del bloque de concesión y luego moverlo a un archivo de política Java 2 no hará que se lance una
`SecurityException`

, porque el permiso es más general ahora (disponible para todos los Principals).

Puesto que ambos ejemplos realizan la misma función, tiene que haber una razón para escribir el código de una forma y no de otra. El ejemplo 1 puede ser más fácil de leer para algunos programadores no familiarizados con las clases anónimas. La clase ***action*** también se podría colocar en un archivo aparte con un CodeBase exclusivo, y luego la concesión de permiso podría utilizar esta información. El ejemplo 2 es más compacto y la ***action*** que debe realizarse es más fácil de encontrar, ya que está justo en la llamada
`doAs`

.

Los siguientes métodos también realizan trabajo como un Subject particular. Sin embargo, los métodos
`doAsPrivileged`

tendrán controles de seguridad basados en la ***action*** y en el ***subject*** suministrados. El contexto suministrado estará ligado al ***subject*** y a la ***action*** especificados. Un objeto de contexto nulo se desentenderá del
`AccessControlContext`

actual.

```
public static Object doAsPrivileged(final Subject subject,
                     final java.security.PrivilegedAction action,
                     final java.security.AccessControlContext acc);

public static Object doAsPrivileged(final Subject subject,
                     final java.security.PrivilegedExceptionAction action,
                     final java.security.AccessControlContext acc)
                     throws java.security.PrivilegedActionException;
```

Los métodos
`doAsPrivileged`

se comportan de forma parecida a los métodos
`doAs`

: el ***subject*** se asocia al contexto ***acc***, se realiza una ***action*** y pueden lanzarse excepciones de tiempo de ejecución o excepciones comprobadas. Sin embargo, los métodos
`doAsPrivileged`

vacían en primer lugar el
`AccessControlContext`

existente de la hebra antes de asociar el ***subject*** al contexto suministrado y antes de invocar la ***action***. Un argumento ***acc*** nulo tiene el efecto de hacer que las decisiones de control de acceso (invocadas mientras se procesa la ***action***) se basen solo en el ***subject*** y en la ***action***. Se necesita un
`AuthPermission("doAsPrivileged")`

cuando se llama a los métodos
`doAsPrivileged`

.

Principals

Como se ha mencionado anteriormente, los Principals pueden estar asociados a un Subject. Los Principals representan identidades de Subject y deben implementar las interfaces
`java.security.Principal`

y
`java.io.Serializable`

. La sección Subject describe maneras de actualizar los Principals asociados a un Subject.

Credenciales

Las clases de credenciales públicas y privadas no forman parte de la biblioteca de clases núcleo de JAAS. Cualquier clase java, por lo tanto, puede representar una credencial. Sin embargo, los desarrolladores pueden optar por que las correspondientes clases de credenciales implementen dos interfaces relacionadas con las credenciales: Refreshable (renovable) y Destroyable (destruible).

Renovable

Esta **interfaz** proporciona la capacidad de que una credencial se renueve. Por ejemplo, una credencial con un período de vida restringido en el tiempo puede ejecutar esta interfaz para permitir a los llamadores renovar el período de tiempo durante el que es válida. La interfaz tiene dos métodos abstractos:

```
boolean isCurrent();
```

Determina si la credencial es actual o válida.

```
void refresh() throws RefreshFailedException;
```

Actualiza o prolonga la validez de la credencial. La implementación de este método realiza un control de seguridad
`AuthPermission("refreshCredential")`

para asegurar que el llamador tiene permiso para renovar la credencial.

Destruible

Esta **interfaz** proporciona la capacidad de destruir los contenidos dentro de una credencial. La interfaz tiene dos métodos abstractos:

```
boolean isDestroyed();
```

Determina si la credencial ha sido destruida.

```
void destroy() throws DestroyFailedException;
```

Destruye y borra la información asociada a esta credencial. Las llamadas posteriores a ciertos métodos de esta credencial harán que se lance una excepción
`IllegalStateException`

. Esta implementación de método realiza un control de seguridad de
`AuthPermission("destroyCredential")`

para garantizar que el llamador tiene permiso para destruir la credencial.

Clases de autenticación

Para autenticar un
`Subject`

, hay que realizar estos pasos:

1. Una aplicación crea instancia de un
   `LoginContext`

   .
2. El
   `LoginContext`

   consulta una configuración para cargar todos los LoginModules configurados para esa aplicación.
3. La aplicación invoca el método *login* de LoginContext.
4. El método *login* invoca todos los LoginModules cargados. Cada
   `LoginModule`

   intenta autenticar el
   `Subject`

   . Si tienen éxito, los LoginModules asocian los Principals y las credenciales relevantes al
   `Subject`

   .
5. El
   `LoginContext`

   devuelve el estado de autenticación a la aplicación.
6. Si la autenticación es satisfactoria, la aplicación recupera el
   `Subject`

autenticado a partir del
`LoginContext`

.

LoginContext

La clase
`LoginContext`

proporciona los métodos básicos utilizados para autenticar Subjects, así como una forma de desarrollar una aplicación con independencia de la tecnología de autenticación subyacente. El
`LoginContext`

consulta una configuración
`Configuration`

para determinar los servicios de autenticación, o LoginModules, configurados para una aplicación particular. Por lo tanto, pueden conectarse diferentes LoginModules bajo una aplicación sin necesidad de realizar modificaciones en la propia aplicación.
`LoginContext`

ofrece cuatro constructores entre los que elegir:

```
public LoginContext(String name) throws LoginException;

public LoginContext(String name, Subject subject) throws LoginException;

public LoginContext(String name, CallbackHandler callbackHandler)
       throws LoginException

public LoginContext(String name, Subject subject,
       CallbackHandler callbackHandler) throws LoginException
```

Todos estos constructores comparten un parámetro común: ***name***. Este argumento lo utiliza el
`LoginContext`

para indexar la configuración de inicio de sesión. Los constructores que no toman un
`Subject`

como parámetro de entrada crean una instancia de un nuevo
`Subject`

. Las entradas nulas no están permitidas para ningún constructor. Los llamadores necesitan un
`AuthPermission("createLoginContext")`

para crear una instancia de un
`LoginContext`

.

La autenticación real se produce con una llamada al siguiente método:

```
public void login() throws LoginException;
```

Cuando se invoca *login*, se invocan todos los métodos *login* de los respectivos LoginModules configurados, para realizar la autenticación. Si la autenticación es satisfactoria, el

Subject

autenticado (que ahora puede contener Principals, credenciales públicas y credenciales privadas) se puede recuperar mediante este método:

```
public Subject getSubject();
```

Para cerrar la sesión de un
Subject

y eliminar sus Principals y credenciales autenticadas, se proporciona el siguiente método:

```
public void logout() throws LoginException;
```

El siguiente fragmento de código en una aplicación autenticará a un Subject llamado ″bob″ después de acceder a un archivo de configuración con una entrada de configuración llamada ″moduleFoo″:

```
Subject bob = new Subject();
LoginContext lc = new LoginContext("moduleFoo", bob);
try {
    lc.login();
    System.out.println("autenticación satisfactoria");
} catch (LoginException le) {
    System.out.println("autenticación no satisfactoria"+le.printStackTrace());
}
```

Este fragmento de código en una aplicación autenticará a un Subject ″sin nombre″ y luego utilizará el método getSubject para recuperarlo:

```
LoginContext lc = new LoginContext("moduleFoo");
try {
    lc.login();
    System.out.println("autenticación satisfactoria");
} catch (LoginException le) {
    System.out.println("autenticación no satisfactoria"+le.printStackTrace());
}
Subject subject = lc.getSubject();
```

Si la autenticación falla, el *getSubject* devuelve nulo. Tampoco se necesita un
AuthPermission("getSubject")

para hacer esto, como en el caso de
Subject.getSubject

.

LoginModule

La **interfaz** LoginModule permite a los programadores implementar varios tipos de tecnologías de autenticación que se pueden conectar bajo una aplicación. Por ejemplo, un tipo de
LoginModule

puede realizar una forma de autenticación basada en el nombre del usuario y la contraseña.

El manual LoginModule Developer's Guide es un documento detallado que da a los desarrolladores instrucciones paso a paso para implementar LoginModules.

Para crear una instancia de un
LoginModule

, el

`LoginContext`

espera que cada
`LoginModule`

proporcione un constructor público que no tome argumentos. Luego, para inicializar un
`LoginModule`

con la información relevante, el
`LoginContext`

llama al método
`initialize`

del LoginModule. Se garantiza que el *subject* proporcionado no sea no-nulo.

```
void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options);
```

El siguiente método empieza el proceso de autenticación:

```
boolean login() throws LoginException;
```

Una implementación de método de ejemplo puede solicitar al usuario un nombre de usuario y una contraseña, y luego verificar la información con respecto a los datos almacenados en un servicio de denominación como NIS o LDAP. Otras implementaciones alternativas podrían intercambiar información con tarjetas inteligentes y dispositivos biométricos, o podrían sencillamente extraer información de usuario del sistema operativo subyacente. Esto se considera como la ***fase 1*** del proceso de autenticación de JAAS.

El siguiente método completa y finaliza el proceso de autenticación:

```
boolean commit() throws LoginException;
```

Si la ***fase 1*** del proceso de autenticación ha sido satisfactoria, este método continua con la ***fase 2***: asociar Principals, credenciales públicas y credenciales privadas al Subject. Si la ***fase 1*** falla, el método *commit* elimina cualquier estado de autenticación almacenado con anterioridad, como los nombres de usuario y las contraseñas.

El siguiente método para el proceso de autenticación si la ***fase 1*** no se realizó satisfactoriamente:

```
boolean abort() throws LoginException;
```

Las implementaciones típicas de este método borran el estado de autenticación almacenado con anterioridad, como los nombres de usuario o las contraseñas. El siguiente método cierra la sesión de un Subject:

```
boolean logout() throws LoginException;
```

Este método elimina los Principals y las credenciales originalmente asociados al
`Subject`

durante la operación
`commit`

. Las credenciales se destruyen tras la eliminación.

CallbackHandler

En ciertos casos, el LoginModule debe comunicarse con el usuario para obtener información de autenticación. Los LoginModules utilizan un CallbackHandler con esta finalidad. Las aplicaciones implementan la **interfaz** CallbackHandler y la pasan al LoginContext, que la reenvía directamente a los LoginModules subyacentes. Los LoginModules utilizan el CallbackHandler para reunir datos de entrada de los usuarios (como una contraseña o un número pin de la tarjeta inteligente) o para suministrar información a los usuarios (como información de estado). Al permitir a la aplicación especificar el CallbackHandler, los LoginModules subyacentes pueden permanecer independientes de las distintas maneras en que las aplicaciones interaccionan con los usuarios. Por ejemplo, la implementación de un CallbackHandler para una aplicación GUI puede visualizar una ventana para solicitar datos de entrada de un usuario. La implementación de un CallbackHandler para una herramienta no GUI podría solicitar al usuario una entrada directamente desde la línea de mandatos.

`CallbackHandler`

es una **interfaz** con un método para implementar:

```
void handle(Callback[] callbacks)
throws java.io.IOException, UnsupportedCallbackException;
```

Callback

El paquete javax.security.auth.callback contiene la **interfaz** Callback además de varias implementaciones. Los LoginModules pueden pasar una matriz de Callbacks directamente al método *handle* de un CallbackHandler.

Consulte las diversas API de Callback para obtener más información sobre su utilización.

Clases de autorización

Al producirse una autenticación satisfactoria de un
`Subject`

, se pueden colocar controles precisos en relación con ese
`Subject`

a base de invocar los métodos Subject.doAs o Subject.doAsPrivileged. Los permisos otorgados a dicho
`Subject`

están configurados en una
`Policy`

JAAS.

Policy

Es una clase **abstract** para representar el control de acceso JAAS a escala de todo el sistema. Como valor predeterminado, JAAS proporciona una implementación de subclase basada en archivo, PolicyFile. Cada subclase
`Policy`

debe implementar los siguientes métodos:

```
public abstract java.security.PermissionCollection getPermissions
                (Subject subject,
                java.security.CodeSource cs);
public abstract void refresh();
```

El método
`getPermissions`

devuelve los permisos otorgados a los objetos
`Subject`

y
`CodeSource`

especificados. El método
`refresh`

actualiza la clase
`Policy`

de tiempo de ejecución con las modificaciones realizadas desde la última vez que se cargó desde su almacén permanente (por ejemplo, un archivo o una base de datos). El método
`refresh`

necesita un
`AuthPermission("refreshPolicy")`

.

El siguiente método recupera el

objeto
`Policy`

de tiempo de ejecución actual, y está protegido por un control de seguridad que exige que el llamador tenga un
`AuthPermission("getPolicy")`

.

```
    public static Policy getPolicy();
```

El siguiente código de ejemplo hace una demostración de cómo se puede consultar un objeto
`Policy`

para obtener el conjunto de permisos otorgados a los objetos
`Subject`

y
`CodeSource`

especificados:

```
    policy = Policy.getPolicy();
    PermissionCollection perms = policy.getPermissions(subject, codeSource);
```

Para establecer un nuevo objeto
`Policy`

en el entorno de tiempo de ejecución Java, se puede utilizar el método

`Policy.setPolicy`

. Este método exige que el llamador tenga un
`AuthPermission("setPolicy")`

.

```
    public static void setPolicy(Policy policy);
```

Entradas de ejemplo del archivo de política:

Estos ejemplos solo son relevantes para la implementación del PolicyFile predeterminado.

Cada entrada del

objeto
`Policy`

se representa como una entrada de tipo ***grant***. Cada entrada de tipo ***grant*** especifica una tripleta de base de código/firmantes de código/Principals, así como los permisos otorgados a la tripleta. Concretamente, los permisos se otorgarán a cualquier código que se descargue de la *base de código* y que esté firmado por los *firmantes de código* especificados, siempre y cuando el
`Subject`

que ejecute el código tenga todos los *Principals* especificados en su conjunto
`Principal`

. Consulte los ejemplos de Subject.doAs para ver cómo se asocia un
`Subject`

al código en ejecución.

```
    grant CodeBase ["URL"],
          Signedby ["signers"],
          Principal [Principal_Class] "Principal_Name",
          Principal ... {
        permission Permission_Class ["Target_Name"]
                                    [, "Permission_Actions"]
                                    [, signedBy "SignerName"];
    };

    // Entrada grant de ejemplo
    grant CodeBase "http://griffin.ibm.com", Signedby "davis",
          Principal com.ibm.security.auth.NTUserPrincipal "kent" {
        permission java.io.FilePermission "c:/kent/files/*", "read, write";
    };
```

Si no se especifica información de *Principal* en la entrada grant (otorgar) de la
`Policy`

de JAAS, se lanzará una excepción de análisis. Sin embargo, las entradas grant (ortorgar) que ya existan en el archivo de política basado en el origen de código Java 2 (y que por lo tanto no tienen información de *Principal*) siguen siendo válidas. En estos casos, la información de *Principal* se entiende que es '*' (las entradas grant atañen a todos los Principales).

Los componentes CodeBase y Signedby de la entrada grant son opcionales en la
`Política`

JAAS. Si no están presentes, no habrá ninguna base de código que coincida ni tampoco ningún firmante (incluido el código no firmado) que coincida.

En el ejemplo anterior, la entrada ***grant*** especifica que el código descargado desde ″http://griffin.ibm.com″, firmado por ″davis″ y que se ejecuta como el usuario ″kent″ de NT tiene un `Permission`. Este `Permission` permite que el código de proceso lea y escriba archivos en el directorio ″c:\kent\files″.

En una sola entrada ***grant*** pueden figurar múltiples Principals. El `Subject` actual que ejecuta el código debe tener todos los Principals especificados en su conjunto `Principal` para que se le otorguen los permisos de la entrada.

```
grant Principal com.ibm.security.auth.NTUserPrincipal "kent",
      Principal com.ibm.security.auth.NTSidGroupPrincipal "S-1-1-0" {
    permission java.io.FilePermission "c:/user/kent/", "read, write";
    permission java.net.SocketPermission "griffin.ibm.com", "connect";
};
```

Esta entrada otorga permiso para leer y escribir archivos en ″c:\user\kent″, así como permiso para establecer conexiones por socket con ″griffin.ibm.com″, a todo código que se ejecute como usuario ″kent″ de NT con el número de identificación de grupo ″S-1-1-0″ de NT.

AuthPermission

Esta clase encapsula los permisos básicos necesarios para This JAAS. Un permiso AuthPermission contiene un nombre (también llamado ″nombre destino″), pero ninguna lista de acciones; se tiene el permiso nombrado o no se tiene. Además de los métodos heredados (de la clase `Permission`), el `AuthPermission` tiene dos constructores públicos:

```
public AuthPermission(String name);
public AuthPermission(String name, String actions);
```

El primer constructor crea un nuevo AuthPermission con el nombre especificado. El segundo constructor también crea un nuevo objeto AuthPermission con el nombre especificado, pero además tiene un argumento *actions* adicional que actualmente no se utiliza y que es nulo. Este constructor existe solamente para el objeto `Policy` para crear instancias de nuevos objetos Permission. El primer constructor es apropiado para la mayor parte del código.

El objeto AuthPermission se utiliza para proteger el acceso a los objetos Policy, Subject, LoginContext y Configuration. Consulte el Javadoc de AuthPermission para obtener una lista de los nombres válidos soportados.

PrivateCredentialPermission

Esta clase protege el acceso a las credenciales privadas de un Subject y proporciona un constructor público:

```
public PrivateCredentialPermission(String name, String actions);
```

Consulte el Javadoc de PrivateCredentialPermission para obtener información más detallada sobre esta clase.

Implementación

Nota: el Apéndice A contiene un archivo de ejemplo **java.security** que incluye las propiedades estáticas mencionadas aquí.

Puesto que allí existen valores predeterminados para los archivos de proveedores y los archivos de política de JAAS, los usuarios no necesitan enumerar sus valores para implementar JAAS de forma estática (en el archivo java.security) ni dinámica (opción de línea de mandatos ***-D***). También la configuración por omisión y los proveedores de archivo de política pueden sustituirse por un proveedor desarrollado por el usuario. Por lo tanto, esta sección intenta explicar los proveedores por omisión de JAAS y los archivos de política, así como las propiedades que habilitan proveedores alternativos.

Lea la API de archivo de política y la API de archivo de configuración por omisión para obtener más información de la que se ha resumido aquí.

Proveedor de autenticación

El proveedor de autenticación, o clase de configuración, se establece estáticamente con
`login.configuration.provider=[class]`

en el archivo *java.security*. Este proveedor crea el objeto
`Configuration`

.

Por ejemplo:

```
login.configuration.provider=com.foo.Config
```

Si la propiedad de seguridad
`login.configuration.provider`

no está en java.security, JAAS la establecerá en el valor predeterminado:
`com.ibm.security.auth.login.ConfigFile`

.

Si se establece un gestor de seguridad antes de

crear
`Configuration`

, habrá que otorgar un
`AuthPermission("getLoginConfiguration")`

.

No existe una forma de establecer dinámicamente el proveedor de configuración en la línea de mandatos.

Archivo de configuración de autenticación

Los archivos de configuración de autenticación se pueden establecer estáticamente en *java.security* con
`login.config.url.n=[URL]`

, siendo *n* es un número entero que toma valores consecutivos, empezando por el 1. El formato es idéntico al de los archivos de política de seguridad Java (policy.url.n=[URL]).

Si la propiedad de seguridad
`policy.allowSystemProperty`

se establece en ″true″ en java.security, los usuarios pueden establecer dinámicamente archivos de política en la línea de mandatos utilizando la opción ***-D*** con esta propiedad:
`java.security.auth.login.config`

. El valor puede ser una vía de acceso o un URL. Por ejemplo (en NT):

```
... -Djava.security.auth.login.config=c:\config_policy\login.config ...
o bien
... -Djava.security.auth.login.config=file:c:/config_policy/login.config ...
```

Nota: el uso de signos igual dobles (==) en la línea de mandatos permite que un usuario altere temporalmente todos los demás archivos de de política encontrados.

Si no se pueden encontrar archivos de configuración de manera estática ni dinámica, JAAS intentará cargar el archivo de configuración desde su ubicación predeterminada:
${user.home}`\.java.login.config`

siendo *${user.home}* es una ubicación que depende del sistema.

Proveedor de autorización

El proveedor de autorización, o clase de política JAAS, se establece estáticamente con
`auth.policy.provider=[class]`

en el archivo *java.security*. Este proveedor crea el objeto
`Policy`

basado en el Subject JAAS.

Por ejemplo:

```
auth.policy.provider=com.foo.Policy
```

Si la propiedad de seguridad
`auth.policy.provider`

no se encuentra en java.security, JAAS la establecerá en el valor predeterminado:
`com.ibm.security.auth.PolicyFile`

.

Si se establece un gestor de seguridad antes de

crear

`Configuration`

, habrá que otorgar un
`AuthPermission("getPolicy")`

.

No existe una forma de establecer dinámicamente el proveedor de autorización en la línea de mandatos.

Archivo de política de autorización

Los archivos de configuración de autenticación se pueden establecer estáticamente en *java.security* con
`auth.policy.url.`*n*`=[URL]`

, siendo *n* un número entero que toma valores consecutivos, empezando por el 1. El formato es idéntico al de los archivos de política de seguridad Java (policy.url.n=[URL]).

Si la propiedad de seguridad
`policy.allowSystemProperty`

se establece en ″true″ en java.security, los usuarios pueden establecer dinámicamente archivos de política en la línea de mandatos utilizando la opción ***-D*** con esta propiedad:
`java.security.auth.policy`

. El valor puede ser una vía de acceso o un URL. Por ejemplo (en NT):

```
... -Djava.security.auth.policy=c:\auth_policy\java.auth.policy ...
o bien
... -Djava.security.auth.policy=file:c:/auth_policy/java.auth.policy ...
```

Nota: el uso de signos igual dobles (==) en la línea de mandatos permite que un usuario altere temporalmente todos los demás archivos de de política encontrados.

No existe una ubicación predeterminada desde la que cargar una política de autorización.

¡″Hello World″, estilo JAAS!

Ponte las gafas de sol oscuras y tu sombrero de fedora favorito, luego coge el saxo alto... es hora de ponerse **JAAS**-eros! Sí, otro programa ″Hello World!″. En esta sección, se habilitará un programa para comprobar la instalación de JAAS.

Instalación Se supone que ya se ha instalado JAAS. Por ejemplo, los archivos JAR de JAAS se han copiado en el directorio de extensiones del Development Kit.

Recuperar archivos Descargue theHelloWorld.tar al directorio de prueba. Expándalo utilizando ″jar xvf HelloWorld.tar″.

Verifique el contenido de su directorio de prueba.

archivos fuente:
- HWLoginModule.java
- HWPrincipal.java
- HelloWorld.java

archivos de clase
- Los archivos fuente se han sido precompilado automáticamente en el directorio classes.

archivos de política

- jaas.config
- java2.policy
- jaas.policy

Compilar archivos fuente Los tres archivos fuente, ***HWLoginModule.java***, ***HWPrincipal.java*** y ***HelloWorld.java***, ya se han compilado y, por lo tanto, no hace falta compilarlos de nuevo.

Si alguno de los archivos fuente se modifica, vaya al directorio de prueba en el que se guardaron y entre:

```
javac -d .\classes *.java
```

Hay que añadir el directorio de clases (.\classes) a la vía de acceso de clases para poder compilar las clases.

Nota:

```
HWLoginModule
```

y

```
HWPrincipal
```

están en el paquete

```
com.ibm.security
```

y se crearán en el direcorio pertinente durante la compilación (>directorio_prueba<\classes\com\ibm\ security).

Examinar archivos de política El archivo de configuración, ***jaas.config***, contiene una entrada:

```
helloWorld {
   com.ibm.security.HWLoginModule required debug=true;
};
```

Solo se suministra un

```
LoginModule
```

con el caso de prueba. Al procesar la aplicación ***HelloWorld***, experimente cambiando

```
LoginModuleControlFlag
```

(required, requisite, sufficient, optional) y suprimiendo el distintivo de depuración. Si están disponibles más módulos LoginModule para las pruebas, podrá modificar esta configuración y experimentar con múltiples módulos LoginModule.

```
HWLoginModule
```

se tratará brevemente.

El archivo de política Java 2, ***java2.policy***, contiene un bloque de permisos:

```
grant {
   permission javax.security.auth.AuthPermission "createLoginContext";
   permission javax.security.auth.AuthPermission "modifyPrincipals";
   permission javax.security.auth.AuthPermission "doAsPrivileged";
};
```

Los tres permisos se necesitan porque la aplicación ***HelloWorld*** (1) crea un objeto LoginContext, (2) modifica los Principals del

```
Subject
```

autenticado y (3) llama al método doAsPrivileged de la clase
`Subject`

.

El archivo de política de JAAS, ***jaas.policy***, también contiene un bloque de permisos:

```
grant Principal com.ibm.security.HWPrincipal "bob" {
   permission java.util.PropertyPermission "java.home", "read";
   permission java.util.PropertyPermission "user.home", "read";
   permission java.io.FilePermission "foo.txt", "read";
};
```

Los tres permisos se otorgan inicialmente a un
`HWPrincipal`

llamado ***bob***. El Principal real que se añade al
`Subject`

autenticado es el nombre de usuario usado durante el proceso de inicio de sesión (más reciente).

A continuación figura el código de acción de ***HelloWorld*** con las tres llamadas de sistema (la razón de los permisos necesarios) en **negrita**:

```
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("\nSu propiedad java.home: "
                           +System.getProperty("java.home"));

        System.out.println("\nSu propiedad user.home: "
                           +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("\nfoo.txt does ");
        if (!f.exists()) System.out.print("no ");
        System.out.println("existe en el directorio actual");

        System.out.println("\nOh, por cierto ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignorar
        }
        System.out.println("\n\nHello World!\n");
        return null;
    }
}, null);
```

Cuando ejecute el programa HelloWorld, utilice diversos nombres de usuario y modifique el archivo jaas.policy de acuerdo con ello. No hay ninguna razón que haga necesario modificar el archivo java2.policy. Asimismo, cree un archivo llamado foo.txt en el directorio de pruebas (test) para probar la última llamada del sistema.

Examinar archivos fuente El LoginModule,
`HWLoginModule`

, autentica a cualquier usuario que entre la contraseña correcta (sensible a las mayúsculas y minúsculas): **Go JAAS**.

La aplicación HelloWorld permite a los usuarios tres intentos de autenticación. Cuando **Go JAAS** se entra correctamente, un
`HWPrincipal`

con el mismo nombre de usuario se añade al
`Subject`

autenticado.

La clase Principal,
`HWPrincipal`

, representa un Principal basado en un nombre de usuario entrado. Este nombre es lo importante cuando se otorgan permisos a los sujetos autenticados.

La aplicación principal,
`HelloWorld`

, primero crea un
`LoginContext`

basado en una entrada de configuración cuyo nombre es **helloWorld**. El archivo de configuración ya ha sido tratado. Se utilizan retornos de llamada para recuperar la entrada de usuario. Consulte la clase
`MyCallbackHandler`

situada en el archivo ***HelloWorld.java*** para ver este proceso.

```
LoginContext lc = null;
try {
    lc = new LoginContext("helloWorld", new MyCallbackHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}
```

El usuario entra un nombre de usuario/contraseña (hasta tres veces) y si se entra **Go JAAS** como contraseña, el sujeto se autentica (
`HWLoginModule`

añade un
`HWPrincipal`

al sujeto).

Como se ha mencionado anteriormente, luego se efectúa trabajo en nombre del sujeto autenticado.

Ejecutar la prueba HelloWorld

Para ejecutar el programa ***HelloWorld***, primero hay que cambiar al directorio de prueba. Habrá que cargar los archivos de configuración y de política. Consulte Implementación para establecer las propiedades correctas en *java.security* o en la línea de mandatos. Aquí se tratará este último método.

El siguiente mandato se ha partido en varias líneas por cuestión de claridad. Entre el mandato en una sola línea.

```
java -Djava.security.manager=
     -Djava.security.auth.login.config=.\jaas.config
     -Djava.security.policy=.\java2.policy
     -Djava.security.auth.policy=.\jaas.policy
     HelloWorld
```

Nota: es necesaria la utilización de ".\filename" para los archivos de política porque la vía de acceso canónica del directorio de prueba variará para cada usuario. Si lo desea, puede sustituir "." por la vía de acceso al directorio de prueba. Por ejemplo, si el directorio de prueba es "c:\test\hello", el primer archivo también pasa a ser:

```
-Djava.security.auth.login.config=c:\test\hello\jaas.config
```

Si no se encuentran los archivos de política, se lanzará una

`SecurityException`

. De lo contrario, se visualizará la información relativa a las propiedades ***java.home*** y ***user.home***. Además, se comprobará la existencia de un archivo llamado ***foo.txt*** en el directorio de prueba. Por último, se muestra el omnipresente mensaje "Hello World".

Practicar con HelloWorld

Ejecute HelloWorld todas las veces que desee. Se recomienda variar las entradas de nombre de usuario/contraseña, cambiar las entradas de archivos de configuración, cambiar los permisos de archivo de política e incluso añadir (apilar) más LoginModules a la entrada de configuración ***helloWorld***. También puede añadir campos de base de código a los archivos de política.

Finalmente, intente ejecutar el programa sin un gestor de seguridad (SecurityManager) para ver cómo funciona si surgen problemas.

Apéndice A: Valores JAAS en el archivo de propiedades de seguridad java.security

Más abajo encontrará una copia del

archivo

`java.security`

que aparece en cada instalación de Java 2. Este archivo aparece en el directorio

`lib/security`

(

`lib\security`

en Windows) del entorno de ejecución Java 2. Así, si el entorno de ejecución Java 2 está instalado en un directorio llamado

`jdk1.3`

, el archivo es

- `jdk1.3/lib/security/java.security`

  (Unix)

- `jdk1.3\lib\security\java.security`

  (Windows)

JAAS añade cuatro nuevas propiedades a
`java.security`

:

- Propiedades de autenticación
  - `login.configuration.provider`
  - `login.policy.url.n`
- Propiedades de autorización
  - `auth.policy.provider`
  - `auth.policy.url.n`

Las nuevas propiedades JAAS están situadas al final de este archivo:

```
#
# Este es el "archivo de propiedades de seguridad maestro".
#
# En este archivo, las clases java.security establecen diversas propiedades
# de seguridad para su utilización. Aquí los usuarios pueden registrar estáticamente
# proveedores de paquetes de criptografía ("proveedores"). El término
# "proveedor" se refiere a un paquete o conjunto de paquetes que suministra una
# implementación concreta de un subconjunto de aspectos criptográficos de
# la API de seguridad Java. El proveedor puede, por ejemplo, implementar uno o
# más algoritmos de firma digital o algoritmos digest de mensaje.
#
# Cada proveedor debe implementar una subclase de la clase Provider.
# Para registrar un proveedor en este archivo de propiedades de seguridad maestro,
# especifique el nombre de subclase de proveedor y la prioridad con el formato
#
#    security.provider.n=nombreClase
#
# Este declara a un proveedor y especifica su orden de preferencia
# n. El orden de preferencia es aquel en que se buscan los proveedores
# para los algoritmos solicitados (cuando no se ha solicitado ningún
# proveedor específico). El orden se basa en 1; 1 es el más preferido, seguido
# del 2, y así sucesivamente.
#
# nombreClase debe especificar la subclase de la clase Provider cuyo
# constructor establece los valores de diversas propiedades necesarias
# para que la API de seguridad Java pueda consultar los algoritmos u otros
# servicios implementados por el proveedor.
#
# Tiene que haber al menos una especificación de proveedor en java.security.
# Hay un proveedor predeterminado estándar que viene con el JDK. Se
# llama proveedor "SUN", y su subclase Provider
# que se llama Sun aparece en el paquete sun.security.provider. Así, el
# proveedor "SUN" se registra por medio de:
#
#    security.provider.1=sun.security.provider.Sun
#
# (El número 1 se utiliza para el proveedor predeterminado)
#
# Nota: Se crea una instancia de las subclases Provider registradas estáticamente
# cuando el sistema se inicializa. Se pueden registrar proveedores dinámicamente
# en lugar de hacerlo con llamadas al método addProvider o al
# método insertProviderAt en la clase Security.

#
# Lista de proveedores y su orden de preferencia (ver arriba):
```

```
#
security.provider.1=sun.security.provider.Sun

#
# Clase para crear instancia como política de sistema. Es el nombre de la clase
# que se utilizará como el objeto Policy.
#
policy.provider=sun.security.provider.PolicyFile

# El valor predeterminado es tener un solo archivo de política a escala del sistema,
# y un archivo de política en el directorio inicial del usuario.
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy

# Tanto si expandimos propiedades en el archivo de política como si no
# si esto está establecido en false, las propiedades (${...}) no se expandirán
# en los archivos de política.
policy.expandProperties=true

# Tanto si se permite a una política extra pasar a la línea de mandatos como si no
# con -Djava.security.policy=somefile. Comente esta línea para inhabilitar
# esta característica.
policy.allowSystemProperty=true

# Tanto si se busca como si no en el IdentityScope identidades de confianza
# cuando se encuentra un archivo JAR firmado 1.1. Si se encuentra la identidad
# y es de confianza, le otorgamos AllPermission.
policy.ignoreIdentityScope=false

#
# Tipo almacén de claves por omisión.
#
keystore.type=jks

#
# Clase de la que crear una instancia con ámbito de sistema:
#
system.scope=sun.security.provider.IdentityDatabase

############################################################################
#
# Servicio de autenticación y autorización Java (JAAS)
# Archivos de propiedades y de política:
#

# Clase para crear instancia como configuración del sistema para la autenticación.
# Este es el nombre de la clase que se utilizará para el objeto de configuración
# de la autenticación.
#
login.configuration.provider=com.ibm.security.auth.login.ConfigFile

# El valor predeterminado es tener un archivo de configuración de inicio de sesión a escala de sistema,
# en el directorio inicial del usuario.  El formato es para muchos archivos parecido
# al de los archivos de política basada en CodeSource anteriores, esto es, policy.url.n
login.config.url.1=file:${user.home}/.java.login.config

# Clase para crear instancia como política de autorización basada en Principal de sistema.
# Este es el nombre de la clase que se utilizará para el objeto de política de
# autorización.
#
auth.policy.provider=com.ibm.security.auth.PolicyFile

# El valor predeterminado es tener un archivo de política basada en Principal a escala de sistema,
# en el directorio inicial del usuario.  El formato es para muchos archivos parecido
# al de los archivos de política basada en CodeSource anteriores, esto es, policy.url.n y
# auth.policy.url.n
auth.policy.url.1=file:${user.home}/.java.auth.policy
```

Apéndice B: Archivos de configuración de inicio de sesión

El archivo de configuración de inicio de sesión contiene uno o más nombres de aplicaciones
`LoginContext`

que tienen el siguiente formato:

```
Application {
    LoginModule Flag ModuleOptions;
    > más entradas de LoginModule <
    LoginModule Flag ModuleOptions;
};
```

Los archivos de configuración de inicio de sesión se localizan utilizando la propiedad de seguridad
`login.config.url.n`

que se encuentra en el archivo
`java.security`

. Para obtener más información sobre esta propiedad y la ubicación del archivo
`java.security`

, vea el Apéndice A.

El valor de *Flag* controla el comportamiento global a medida que la autenticación continúa en la pila. Esto representa una descripción de los valores válidos de *Flag* y sus semánticas respectivas:

1. **Necesario** Se necesita el
   `LoginModule`

   para tener éxito. Tanto si tiene éxito como si falla, la autenticación todavía sigue avanzando por la lista de
   `LoginModules`

   .
2. **Requisito** Se necesita el
   `LoginModule`

   para tener éxito. Si tiene éxito, la autenticación sigue avanzando por la lista de
   `LoginModules`

   . Si falla, el control vuelve inmediatamente a la aplicación (la autenticación no sigue avanzando por la lista de
   `LoginModules`

).
3. **Suficiente** No se necesita el
   `LoginModule`

   para tener éxito. Si no tiene éxito, el control vuelve inmediatamente a la aplicación (la autenticación no sigue avanzando por la lista de
   `LoginModules`

). Si falla, la autenticación sigue avanzando por la lista de
   `LoginModules`

.

4. **Opcional** No se necesita el
   `LoginModule`

   para tener éxito. Tanto si tiene éxito como si falla, la autenticación todavía sigue avanzando por la lista de
   `LoginModules`

   .

La autenticación global solo tiene éxito si todos los LoginModules *Necesarios* y *Requisitos* tienen éxito. Si un
`LoginModule`

Suficiente

está configurado y tiene éxito, para que la autenticación global tenga éxito solo es necesario que tengan éxito los LoginModules *Necesarios* y *Requisitos* anteriores a ese
`LoginModule`

Suficiente. Si no hay LoginModules *Necesarios* ni *Requisitos* configurados para una aplicación, debe haber por lo menos un
`LoginModule`

Suficiente u *Opcional* que tenga éxito.

Archivo de configuración de ejemplo:

```
/* Archivo de configuración de ejemplo */

Login1 {
   com.ibm.security.auth.module.SampleLoginModule required debug=true;
};

Login2 {
   com.ibm.security.auth.module.SampleLoginModule required;
   com.ibm.security.auth.module.NTLoginModule sufficient;
   ibm.loginModules.SmartCard requisite debug=true;
   ibm.loginModules.Kerberos optional debug=true;
};
```

Nota: los distintivos no son sensibles a las mayúsculas y minúsculas. *REQUISITO = requisito = Requisito.*

Login1 solo tiene un LoginModule que es una instancia de la clase
`com.ibm.security.auth.module.SampleLoginModule`

. Por lo tanto, un
`LoginContext`

asociado a **Login1** tendrá una autenticación satisfactoria solo si su módulo solitario se autentica satisfactoriamente. El distintivo *Necesario* es trivial en este ejemplo; los valores del distintivo tienen un efecto importante sobre la autenticación cuando hay presentes dos o más módulos.

Login2 es más fácil de explicar con una tabla.

Estado de autenticación de Login2									
Módulo de inicio de sesión de ejemplo	necesario	pasar	pasar	pasar	pasar	fallar	fallar	fallar	fallar
Módulo de inicio de sesión de NT	suficiente	pasar	fallar	fallar	fallar	pasar	fallar	fallar	fallar
Tarjeta inteligente	requisito	*	pasar	pasar	fallar	*	pasar	pasar	fallar
Kerberos	opcional	*	pasar	fallar	*	*	pasar	fallar	*
Autenticación global		pasar	pasar	pasar	fallar	fallar	fallar	fallar	fallar

* = valor trivial debido al control que vuelve a la aplicación porque un módulo *REQUISITO* anterior falló o un módulo *SUFICIENTE* anterior tuvo éxito.

Apéndice C: Archivo de política de autorización

En caso de que no hubieran suficientes ejemplos de bloques grant de política de JAAS basados en Principal, aquí hay algunos más,

```
// ARCHIVO DE POLÍTICA DE JAAS DE EJEMPLO:  java.auth.policy

// los siguientes permisos se otorgan al Principal 'Pooh' y a todas las fuentes de código:

grant Principal com.ibm.security.Principal "Pooh" {
   permission javax.security.auth.AuthPermission "setPolicy";
   permission java.util.PropertyPermission "java.home", "read";
   permission java.util.PropertyPermission "user.home", "read";
   permission java.io.FilePermission "c:/foo/jaas.txt", "read";
};

// Los siguientes permisos se otorgan al Principal 'Pooh' y 'Eyeore':
// y al CodeSource signedBy "DrSecure":

grant signedBy "DrSecure"
      Principal com.ibm.security.Principal "Pooh",
      Principal com.ibm.security.Principal "Eyeore" {
   permission javax.security.auth.AuthPermission "modifyPublicCredentials";
   permission javax.security.auth.AuthPermission "modifyPrivateCredentials";
   permission java.net.SocketPermission "us.ibm.com", "connect,accept,resolve";
   permission java.net.SocketPermission "griffin.ibm.com", "accept";
};

// Los siguientes permisos se otorgan al Principal 'Pooh' y 'Eyeore':
// 'Piglet' y CodeSource desde el directorio c:\jaas firmado por "kent" y "bruce":

grant codeBase "file:c:/jaas/*",
      signedBy "kent, bruce",
      Principal com.ibm.security.Principal "Pooh",
      Principal com.ibm.security.Principal "Eyeore",
      Principal com.ibm.security.Principal "Piglet"  {
   permission javax.security.auth.AuthPermission "getSubject";
   permission java.security.SecurityPermission "printIdentity";
   permission java.net.SocketPermission "guapo.ibm.com", "accept";
};
```

Ejemplos del servicio de autenticación y autorización Java

Este tema contiene ejemplos del servicio de autenticación y autorización Java (JAAS) en un System i5.

Hay dos ejemplos de JAAS, que se llaman HelloWorld y SampleThreadSubjectLogin. Si desea obtener instrucciones y el código fuente, pulse estos enlaces.

Compilar y ejecutar HelloWorld con el servicio de autenticación y autorización Java en un System i5:

En esta información se muestra cómo se compila y ejecuta **HelloWorld** para el servicio de autenticación y autorización Java (JAAS) en el System i5.

Este información se debe considerar como sustitución del apartado **HelloWorld** del tema "Servicio de autenticación y autorización Java (JAAS)" en la página 345. Los archivos de código fuente, de política y de configuración son idénticos a los del documento API Developers Guide. Sin embargo, hay algunos aspectos que son exclusivos del System i5.

1. Deberá poner los siguientes archivos fuente en su propio directorio de pruebas:
   - HWLoginModule.java
   - HWPrincipal.java
   - HelloWorld.java

   Estos archivos fuente se tienen que compilar en el directorio ./classes.

   Para ver el código fuente de estos archivos formateados para su navegador HTML, consulte "Ejemplos: HelloWorld para JAAS" en la página 507.
2. Hay que compilar los tres archivos fuente HWLoginModule.java, HWPrincipal.java y HelloWorld.java. Ejecute los siguientes mandatos (cada uno en una línea) en una línea de mandatos de i5/OS:
   a. strqsh
   b. cd *suDirPruebas*
   c.
   ```
   javac -J-Djava.version=1.3
         -classpath /qibm/proddata/os400/java400/ext/jaas13.jar:.
         -d ./classes *.java
   ```

   Siendo *suDirPruebas* el directorio que creó para almacenar los archivos de ejemplo. Para compilar las clases, habrá que añadir el directorio de clases (.\classes) a la vía de acceso de clases.

   Nota: HWLoginModule y HWPrincipal están en el paquete com.ibm.security y se crean en el directorio apropiado durante la compilación (\classes\com\ibm\security).
3. Ejecute los siguientes mandatos (cada uno en una línea) en la línea de mandatos de i5/OS:
   a. strqsh
   b. cd *suDirPruebas*

      Siendo *suDirPruebas* el directorio que creó para almacenar los archivos de ejemplo. Para compilar las clases, habrá que añadir el directorio de clases (.\classes) a la vía de acceso de clases.
   c. Deberá poner los siguientes archivos fuente en su propio directorio de pruebas:
      - jaas.config
      - java2.policy
      - jaas.policy

d.

```
java -Djava.security.manager=
     -Djava.security.auth.login.config=./jaas.config
     -Djava.security.policy=./java2.policy
     -Djava.security.auth.policy=./jaas.policy
     -Djava.version=1.3
     -classpath ./classes
     HelloWorld
```

e. Cuando se le pida el nombre de usuario, entre **bob**. Si se ejecuta con un gestor de seguridad, es necesario entrar el usuario **bob** para que sean satisfactorios todos los permisos de acceso. Cuando se le pida una contraseña, entre **Go JAAS**, que es sensible a las mayúsculas/minúsculas y contiene un espacio.

Detalles: cómo funciona HelloWorld para el servicio de autenticación y autorización Java (JAAS):

En este documento se describe con más detenimiento cómo funciona **HelloWorld** para el servicio de autenticación y autorización Java (JAAS).

Esta información sustituye a la sección **HelloWorld** de API Developers Guide. Los archivos de código fuente, de política y de configuración son idénticos a los del documento API Developers Guide. Sin embargo, hay algunos aspectos que son exclusivos de la plataforma System i5.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

Archivos de configuración y política

El archivo de configuración, **jaas.config**, contiene una sola entrada:

```
helloWorld {
    com.ibm.security.HWLoginModule required debug=true;
};
```

El caso de prueba incluye un solo LoginModule (módulo de inicio de sesión). Cuando ejecute la aplicación HelloWorld, puede experimentar haciendo cambios en el distintivo LoginModuleControlFlag (cuyos valores pueden ser required, requisite, sufficient, optional) y suprimiendo el distintivo debug. Si están disponibles más módulos LoginModule para las pruebas, podrá modificar esta configuración y experimentar con múltiples módulos LoginModule.

El archivo de política de Java 2, **java2.policy**, contiene un bloque de permisos:

```
grant {
    permission javax.security.auth.AuthPermission "createLoginContext";
    permission javax.security.auth.AuthPermission "modifyPrincipals";
    permission javax.security.auth.AuthPermission "doAsPrivileged";
};
```

Se necesitan los tres permisos porque la aplicación HelloWorld hace lo siguiente:

1. Crea un objeto LoginContext (contexto de inicio de sesión).
2. Cambia los identificadores principales del sujeto autenticado.
3. Llama al método doAsPrivileged de la clase Subject.

El archivo de política de JAAS, **jaas.policy**, también contiene un bloque de permisos:

```
grant Principal com.ibm.security.HWPrincipal "bob" {
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "foo.txt", "read";
};
```

Estos tres permisos se otorgan inicialmente a un HWPrincipal denominado ″bob″. El Principal real que se añade al sujeto autenticado es el nombre de usuario que se utiliza durante el proceso de inicio de sesión.

A continuación figura el código de acción de HelloWorld con las tres llamadas al sistema (la razón de los permisos necesarios) en negrita:

```
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("\nSu propiedad java.home: "
                           +System.getProperty("java.home"));

        System.out.println("\nSu propiedad user.home: "
                           +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("\nfoo.txt does ");
        if (!f.exists()) System.out.print("no ");
        System.out.println("existe en el directorio actual");

        System.out.println("\nOh, por cierto ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // Ignorar
        }
        System.out.println("\n\nHello World!\n");
        return null;
    }
}, null);
```

Cuando ejecute el programa HelloWorld, utilice diversos nombres de usuario y modifique el archivo jaas.policy de acuerdo con ello. No debe haber ninguna razón que haga necesario modificar el archivo java2.policy. Asimismo, cree un archivo llamado foo.txt en el directorio de pruebas (test) para probar la última llamada al sistema y confirmar que se ha otorgado el nivel correcto de acceso a ese archivo.

Examinar los archivos fuente de HelloWorld

La clase LoginModule, HWLoginModule, no hace más que autenticar a los usuarios que entran la contraseña correcta (sensible a las mayúsculas/minúsculas y con un espacio):

- **Go JAAS**

Si se ejecuta con un gestor de seguridad, se debe entrar el usuario 'bob' para que sean satisfactorios todos los permisos de acceso.

La aplicación HelloWorld permite a los usuarios tres intentos de autenticarse. Cuando se ha entrado Go JAAS correctamente, se añade al sujeto autenticado un objeto HWPrincipal cuyo nombre es igual al nombre de usuario.

La clase Principal, HWPrincipal, representa un Principal basado en el nombre de usuario entrado. Este nombre es importante cuando se otorgan permisos a los sujetos autenticados.

La aplicación principal, HelloWorld, crea en primer lugar un LoginContext basado en una entrada de configuración cuyo nombre es helloWorld. Se utilizan retornos de llamada para recuperar la entrada de usuario. Podrá ver este proceso en la clase MyCallbackHandler (mi manejador de retornos de llamada) situada en el archivo HelloWorld.java. A continuación figura un extracto del código fuente:

```
LoginContext lc = null;
try {
    lc = new LoginContext("helloWorld", new MyCallbackHandler());
```

```
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}
```

El usuario entra un nombre de usuario y una contraseña (puede hacerlo hasta tres veces) y si se entra la contraseña Go JAAS, el sujeto queda autenticado (HWLoginModule añade un HWPrincipal al sujeto). Luego se efectúa el trabajo en nombre del sujeto autenticado.

Si no se encuentran los archivos de política, se lanza una excepción **SecurityException**. En caso contrario, se visualiza información relacionada con las propiedades java.home y user.home. Asimismo, se comprueba la existencia de un archivo llamado foo.txt en su directorio de pruebas (test). Por último, se muestra el omnipresente mensaje ″Hello World″.

Practicar con HelloWorld

Ejecute HelloWorld todas las veces que desee. En la siguiente lista figuran algunos ejercicios que puede practicar:

- Varíe los nombres de usuario y las contraseñas que se entran
- Cambie las entradas del archivo de configuración
- Cambie los permisos del archivo de política
- Añada otros módulos LoginModule a la entrada de configuración helloWorld
- Añada campos de base de código (codeBase) a los archivos de política
- Ejecute el programa sin un gestor de seguridad (SecurityManager) para ver cómo funciona si surgen problemas.

Instrucciones para SampleThreadSubjectLogin del servicio de autenticación y autorización Java:

Siga estas instrucciones para utilizar el ejemplo de SampleThreadSubjectLogin.java.

Archivos fuente

Coloque el archivo fuente SampleThreadSubjectLogin.java en su propio directorio de pruebas: este archivo fuente debe compilarse en el directorio ./classes.

Para ver el código fuente de este archivo formateado para su navegador HTML, consulte el Ejemplo: SampleThreadSubjectLogin de JAAS.

Archivos de política

En API Developers Guide hallará información general relacionada con los archivos de política de JAAS. Los archivos de política específicos del ejemplo SampleThreadSubjectLogin son estos:

- threadLogin.config
- threadJava2.policy
- threadJaas.policy

Consulte los comentarios del principio de SampleThreadSubjectLogin.java para obtener información acerca de la compilación y ejecución de este ejemplo.

Servicio de seguridad genérico Java Generic (JGSS) de IBM

El servicio de seguridad genérico Java (JGSS) proporciona una interfaz genérica para la autenticación y la mensajería segura. Bajo esta interfaz puede conectar distintos mecanismos de seguridad basados en claves secretas, claves públicas u otras tecnologías de seguridad.

Mediante la abstracción de la complejidad y las peculiaridades de los mecanismos de seguridad subyacentes en una interfaz estándar, JGSS aporta las siguientes ventajas al desarrollo de aplicaciones de red seguras:

- Puede desarrollar la aplicación para una única interfaz abstracta
- Puede emplear la aplicación con distintos mecanismos de seguridad sin efectuar ningún cambio

JGSS define los enlaces Java para la interfaz de programación de aplicaciones del servicio de seguridad genérico (GSS-API), que es una API criptográfica estandarizada por el equipo negociador de ingenieros de Internet (IETF) y adoptada por el X/Open Group.

La implementación IBM de JGSS se llama IBM JGSS. IBM JGSS es una implementación de la infraestructura GSS-API que emplea Kerberos V5 como sistema de seguridad subyacente predeterminado. También presenta un módulo de inicio de sesión del servicio de autenticación y autorización Java (JAAS) para crear y utilizar credenciales de Kerberos. Además, puede hacer que JGSS efectúe comprobaciones de autorización de JAAS cuando utilice esas credenciales.

IBM JGSS incluye un proveedor de JGSS nativo de System i5, un proveedor de JGSS Java y versiones Java de las herramientas de gestión de credenciales de Kerberos (kinit, ktab y klist).

Nota: El proveedor de JGSS nativo de System i5 utiliza una biblioteca de servicios de autenticación de red (NAS) nativa de System i5. Cuando utilice el proveedor nativo, debe utilizar las utilidades Kerberos de System i5. Para obtener más información, consulte Proveedores de JGSS.

Mejora de seguridad de Sun Microsystems, Inc.

IETF (Internet Engineering Task Force) RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1

IETF RFC 2853 Generic Security Service API Version 2: Java Bindings

The X/Open Group GSS-API Extensions for DCE

Conceptos de JGSS

Las operaciones de JGSS constan de cuatro fases diferenciadas, según el estándar de GSS-API (Generic Security Service Application Programming Interface).

Las fases son las siguientes:

1. Recopilación de credenciales para principales.
2. Creación y establecimiento de un contexto de seguridad entre los principales de iguales que se comunican.
3. Intercambio de mensajes seguros entre los iguales.
4. Borrado y liberación de recursos.

Asimismo, JGSS aprovecha la arquitectura criptográfica Java (JCA) para ofrecer una capacidad de conexión sin fisuras de los distintos mecanismos de seguridad.

Consulte los enlaces siguientes para obtener descripciones de alto nivel de estos importantes conceptos de JGSS.

Principales y credenciales de JGSS:

La identidad con la que una aplicación participa en una comunicación segura JGSS con un igual se denomina principal. Un principal puede ser un usuario real o un servicio desatendido. El principal adquiere credenciales específicas del mecanismo de seguridad como documento de identidad bajo ese mecanismo.

Por ejemplo, al utilizar el mecanismo Kerberos, la credencial de un principal tiene el formato de un ticket de otorgamiento de tickets (TGT) emitido por un centro de distribución de claves (KDC) de Kerberos. En un entorno de varios mecanismos, una credencial de GSS-API puede contener varios elementos de credencial, cada uno de los cuales representa una credencial de mecanismo subyacente.

El estándar de GSS-API no establece cómo adquiere credenciales un principal y las implementaciones de GSS-API normalmente no proporcionan un método para la adquisición de credenciales. Un principal obtiene las credenciales antes de emplear GSS-API; GSS-API simplemente consulta al mecanismo de seguridad las credenciales en nombre del principal.

IBM JGSS incluye versiones Java de la clase Kinit de com.ibm.security.krb5.internal.tools, la clase Ktab de com.ibm.security.krb5.internal.tools y la clase Klist de com.ibm.security.krb5.internal.tools de las herramientas de gestión de credenciales Kerberos. Además, IBM JGSS mejora el estándar GSS-API proporcionando una interfaz de inicio de sesión Kerberos opcional que utiliza JAAS. El proveedor de JGSS Java puro admite la interfaz de inicio de sesión opcional; no así el proveedor de i5/OS.

Conceptos relacionados

"Obtener credenciales de Kerberos y crear claves secretas" en la página 391
GSS-API no define ningún método para obtener credenciales. Por lo tanto, el mecanismo Kerberos de IBM JGSS exige que el usuario obtenga credenciales de Kerberos. Este tema le enseña a obtener credenciales de Kerberos y a crear claves secretas, y le explica cómo utilizar JAAS para realizar inicios de sesión y comprobaciones de autorización de Kerberos, así como revisar una lista de los permisos de JAAS necesarios para la máquina virtual Java (JVM).

"Proveedores de JGSS" en la página 388
IBM JGSS incluye un proveedor de JGSS nativo de System i5 y un proveedor de JGSS Java puro. El proveedor que elija utilizar dependerá de las necesidades de la aplicación.

Clase Klist de com.ibm.security.krb5.internal.tools:

Esta clase se puede ejecutar como una herramienta de línea de mandatos para enumerar las entradas de la caché de credenciales y de la tabla de claves.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Klist
```

public class **Klist**
extends java.lang.Object

Esta clase se puede ejecutar como una herramienta de línea de mandatos para enumerar las entradas de la caché de credenciales y de la tabla de claves.

Resumen del constructor

`Klist()`

Resumen de los métodos

static void	`main(java.lang.String[] args)` El programa main que se puede invocar en la línea de mandatos.

Métodos heredados de la clase java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Detalles del constructor

Klist

```
public Klist()
```

Detalles del método

main

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

El programa main que se puede invocar en la línea de mandatos.

Uso: java com.ibm.security.krb5.tools.Klist [[-c] [-f] [-e] [-a]] [-k [-t] [-K]] [nombre]

Opciones disponibles para las cachés de credenciales:
- **-f** muestra distintivos de credenciales
- **-e** muestra el tipo de cifrado
- **-a** visualiza la lista de direcciones

Opciones disponibles para las tablas de claves:
- **-t** muestra las indicaciones horarias de las entradas de la tabla de claves
- **-K** muestra las claves DES de las entradas de la tabla de claves

Clase Kinit de com.ibm.security.krb5.internal.tools:

Herramienta Kinit para obtener tickets Kerberos v5.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Kinit
```

public class **Kinit**
extends java.lang.Object

Herramienta Kinit para obtener tickets Kerberos v5.

Resumen del constructor

`Kinit(java.lang.String[] args)`
Construye un objeto Kinit nuevo.

Resumen de los métodos

`static void`	`main(java.lang.String[] args)` El método main se utiliza para aceptar la entrada de línea de mandatos para petición de tickets

Métodos heredados de la clase java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Detalles del constructor

Kinit

```
public Kinit(java.lang.String[] args)
      throws java.io.IOException,
             RealmException,
             KrbException
```

Construye un objeto Kinit nuevo.

Parámetros:
`args` - matriz de opciones de peticiones de tickets. Las opciones disponibles son: -f, -F, -p, -P, -c, -k, principal, password.

Lanza: java.io.IOException - Si se produce un error de E/S.
RealmException - Si no se ha podido crear una isntancia del reino.
KrbException - Si se produce un error durante la operación de Kerberos.

Detalles del método

main

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

El método main se utiliza para aceptar la entrada de línea de mandatos del usuario para petición de tickets.

Utilización: java com.ibm.security.krb5.tools.Kinit [-f] [-F] [-p] [-P] [-k] [-c nombre de la caché] [principal] [password]

- **-f** reenviable
- **-F** no reenviable
- **-p** proxiable
- **-P** no proxiable
- **-c** nombre de la caché (ejemplo, FILE:d:\temp\mykrb5cc)
- **-k** utilizar tabla de claves
- **-t** nombre de archivo de tabla de claves
- principal el nombre de principal (ejemplo, qwedf qwedf@IBM.COM)
- password la contraseña del principal de Kerberos

Utilice java com.ibm.security.krb5.tools.Kinit -help para que se abra el menú de ayuda.

Actualmente, solo se da soporte a cachés de credenciales basadas en archivos. Por defecto, se generará un archivo de caché llamado krb5cc_{user.name} en el directorio {user.home} para almacenar el ticket obtenida de KDC. Por ejemplo, en Windows NT, podría ser c:\winnt\profiles\qwedf\krb5cc_qwedf, donde qwedf es el {user.name} y c:\winnt\profile\qwedf es el {user.home}. {user.home} lo obtiene Kerberos de la propiedad Java ″user.home″ del sistema. Si en algún caso {user.home} es null (lo que no ocurre apenas), el archivo de caché se almacenará en el directorio actual desde el que se ejecuta el programa. {user.name} es el nombre de usuario de inicio de sesión del sistema operativo. Puede ser diferente del nombre de principal del usuario. Un usuario puede tener múltiples nombres de principal, pero solo uno es el principal primario de la caché de credenciales, lo que significa que un archivo de caché solo puede almacenar tickets de un principal de usuario específico. Si el usuario cambia el nombre de principal en el próximo Kinit, el archivo de caché generado para el nuevo ticket sustituiría al archivo de caché antiguo por omisión. Al pedir un nuevo ticket, para evitar la sobrescritura, debe especificar un nombre de directorio diferente o un nombre archivo de caché diferente.

Ubicación del archivo de caché

Existen diferentes formas de definir el nombre y la ubicación del archivo de caché específico de un usuario, que figuran a continuación, en el orden de búsqueda empleado por Kerberos:

1. Opción **-c**. Utilice java com.ibm.security.krb5.tools.Kinit -c FILE:<nombre de archivo y directorio específicos del usuario>. ″FILE:″ es el prefijo para identificar el tipo de caché de credenciales. El valor predeterminado es el tipo basado en archivo.
2. Establezca la propiedad Java ″KRB5CCNAME″ del sistema utilizando -DKRB5CCNAME=FILE:<nombre de archivo y directorio específicos del usuario> en tiempo de ejecución.
3. Establezca la variable de entorno ″KRB5CCNAME″ en una solicitud de mandatos antes del momento de la ejecución. Los diferentes sistemas operativos tienen formas diferentes de establecer las variables de entorno. Por ejemplo, Windows utiliza set KRB5CCNAME=FILE:<nombre de archivo y directorio específicos del usuario>, mientras que UNIX utiliza export KRB5CCNAME=FILE:<nombre de archivo y directorio específicos del usuario>. Observe que Kerberos se basa en el mandato específico del sistema para recuperar la variable de entorno. El mandato que se emplea en UNIX es ″/usr/bin/env″.

KRB5CCNAME es sensible a las mayúsculas y minúsculas y está todo en mayúsculas.

Si KRB5CCNAME no se establece como está indicado arriba, se utiliza un archivo de caché predeterminado. La caché predeterminada se localiza por este orden:

1. /tmp/krb5cc_<uid> en las plataformas Unix, donde <uid> es el ID del usuario que ejecuta la JVM de Kinit.
2. <user.home>/krb5cc_<user.name>, donde <user.home> y <user.name> son las propiedades Java user.home y user.name, respectivamente.
3. <user.home>/krb5cc (si <user.name> no se puede obtener a partir de la JVM).

Tiempo de espera de comunicación de KDC

Kinit se comunica con el centro de distribución de claves (KDC) para obtener un ticket de otorgamiento de tickets (TGT), es decir, la credencial. Se puede establecer que esta comunicación exceda el tiempo de espera si el KDC no responde dentro de un tiempo determinado. El período de tiempo excedido puede establecerse (en milisegundos) en el archivo de configuración de Kerberos en la estrofa libdefaults (aplicable a todos los centros de distribución de claves) o en la estrofa del KDC individual. El valor predeterminado del tiempo excedido es de 30 segundos.

Clase Ktab de com.ibm.security.krb5.internal.tools:

Esta clase se puede ejecutar como herramienta de línea de mandatos para ayudar al usuario a gestionar las entradas de la tabla de claves. Las funciones disponibles son listar/añadir/actualizar/suprimir clave(s) de servicio.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Ktab
```

public class **Ktab**
extends java.lang.Object

Esta clase se puede ejecutar como herramienta de línea de mandatos para ayudar al usuario a gestionar las entradas de la tabla de claves. Las funciones disponibles son listar/añadir/actualizar/suprimir clave(s) de servicio.

Resumen del constructor

`Ktab()`

Resumen de los métodos

`static void`	`main(java.lang.String[] args)` El programa main que se puede invocar en la línea de mandatos.

Métodos heredados de la clase java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Detalles del constructor

Ktab

```
public Ktab()
```

Detalles del método

main

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

El programa main que se puede invocar en la línea de mandatos.

Utilización: java com.ibm.security.krb5.tools.Ktab <opciones>

Opciones disponibles para Ktab:
- **-l** Enumerar el nombre y las entradas de la tabla de claves
- **-a** <*nombre de principal*>(<*contraseña*>) Añadir una entrada a la tabla de claves
- **-d** <*nombre de principal*> Suprimir una entrada de la tabla de claves
- **-k** <*nombre de tabla de claves*> Especificar el nombre de la tabla de claves y la vía de acceso con el prefijo FILE:
- **-help** Visualizar instrucciones

Establecimiento de contexto JGSS:

Tras adquirir las credenciales de seguridad, los dos iguales que se comunican establecen un contexto de seguridad mediante sus credenciales. Aunque los iguales establecen un único contexto conjunto, cada igual mantiene su propia copia local del contexto. El establecimiento del contexto supone que el igual iniciador se autentica para el igual aceptante. El iniciador puede solicitar la autenticación mutua, en cuyo caso el aceptante se autentica para el iniciador.

Una vez efectuado el establecimiento del contexto, el contexto establecido incluye información de estado (como, por ejemplo, las claves criptográficas compartidas) que hace posible el intercambio posterior de mensajes seguros entre los dos iguales.

Protección e intercambio de mensajes JGSS:

Tras el establecimiento del contexto, los dos iguales están preparados para participar en intercambios de mensajes seguros. El originador del mensaje llama a su implementación de GSS-API local para codificar

el mensaje, con lo que se garantiza la integridad del mensaje y, si se desea, la confidencialidad del mismo. A continuación la aplicación transporta el símbolola obtenido al igual.

La implementación de GSS-API local del igual utiliza la información del contexto establecido del modo siguiente:

- Verifica la integridad del mensaje
- Descifra el mensaje, si estaba cifrado

Borrado y liberación de recursos:

Para liberar recursos, una aplicación JGSS suprime un contexto que ya no es necesario. Aunque una aplicación JGSS puede acceder a un contexto suprimido, todo intento de utilizarlo para el intercambio de mensajes generará una excepción.

Mecanismos de seguridad:

La especificación GSS-API consiste en una infraestructura abstracta sobre uno o varios mecanismos de seguridad subyacentes. El modo de interactuar de la infraestructura con los mecanismos de seguridad subyacentes es específico de la implementación.

Tales implementaciones se enmarcan en dos categorías generales:

- En un extremo, una implementación monolítica enlaza estrechamente la infraestructura con un único mecanismo. Este tipo de implementación impide el uso de otros mecanismos o incluso distintas implementaciones del mismo mecanismo.
- En el otro extremo, una implementación de gran modularidad ofrece facilidad de uso y flexibilidad. Este tipo de implementación ofrece la posibilidad de conectar distintos mecanismos de seguridad y sus implementaciones a la infraestructura de forma fácil y transparente.

IBM JGSS pertenece a esta última categoría. Como implementación modular, IBM JGSS aprovecha la infraestructura de proveedor definida por la arquitectura criptográfica Java (JCA) y trata los mecanismos subyacentes como proveedores (JCA). El proveedor JGSS proporciona una implementación concreta de un mecanismo de seguridad JGSS. Una aplicación puede crear instancias de varios mecanismos y utilizarlos.

Un proveedor puede dar soporte a varios mecanismos y JGSS facilita el uso de distintos mecanismos de seguridad. Sin embargo, la especificación GSS-API no proporciona un método para que dos iguales que se comunican elijan un mecanismo cuando hay disponibles varios mecanismos. Una forma de elegir un mecanismo consiste en empezar con el mecanismo de negociación GSS-API simple y protegido (SPNEGO)), un seudomecanismo que negocia un mecanismo real entre los dos iguales. IBM JGSS no incluye el mecanismo SPNEGO.

Para obtener más información sobre SPNEGO, consulte la RFC 2478 The Simple and Protected GSS-API Negotiation Mechanism del equipo negociador de ingenieros de Internet (IETF).

Configurar el servidor para que utilice IBM JGSS

El procedimiento adecuado que permite configurar el servidor para que utilice JGSS depende de qué versión de la plataforma Java 2, Standard Edition (J2SE) se ejecute en el sistema.

Configurar el System i5 para que utilice JGSS con J2SDK, versión 1.3:

Cuando utiliza Java 2 Software Development Kit (J2SDK), versión 1.3, en su servidor, debe preparar y configurar el servidor para que utilice JGSS. En la configuración predeterminada se emplea el proveedor de JGSS Java puro.

Requisitos de software

Para utilizar JGSS con J2SDK, versión 1.3, el servidor debe tener instalado el servicio de autenticación y autorización Java (JAAS) 1.3.

Configurar el servidor para que utilice JGSS

Para configurar el servidor de modo que utilice JGSS con J2SDK, versión 1.3, añada al directorio de extensiones un enlace simbólico para el archivo ibmjgssprovider.jar. En el archivo ibmjgssprovider.jar están las clases de JGSS y el proveedor de JGSS Java. La adición del enlace simbólico permite al cargador de clases de extensión cargar el archivo ibmjgssprovider.jar.

Añadir el enlace simbólico

Para añadir el enlace simbólico, vaya a una línea de mandatos de i5/OS y teclee el siguiente mandato (en una sola línea) y pulse **Intro**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssprovider.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/ibmjgssprovider.jar')
```

Nota: La política predeterminada de Java 1.3 en el servidor otorga los permisos pertinentes a JGSS. Si tiene previsto crear su propio archivo java.policy, vea el tema: Permisos de JVM, donde hallará una lista de los permisos que se pueden otorgar a ibmjgssprovider.jar.

Cambiar los proveedores de JGSS

Después de configurar el servidor para que utilice JGSS, que emplea por defecto el proveedor de Java puro, puede configurar JGSS para que utilice el proveedor de JGSS nativo de System i5. Entonces, después de haber configurado JGSS para que utilice el proveedor nativo, le resultará fácil conmutar entre los dos proveedores.

Gestores de seguridad

Si se propone ejecutar la aplicación IBM JGSS teniendo habilitado un gestor de seguridad Java, vea el tema: Utilizar un gestor de seguridad.

Conceptos relacionados

"Permisos de JVM" en la página 389
Además de las comprobaciones de control de acceso efectuadas por JGSS, la máquina virtual Java (JVM) lleva a cabo comprobaciones de autorización al acceder a una gran variedad de recursos, como los archivos, las propiedades Java, los paquetes y los sockets.

"Proveedores de JGSS" en la página 388
IBM JGSS incluye un proveedor de JGSS nativo de System i5 y un proveedor de JGSS Java puro. El proveedor que elija utilizar dependerá de las necesidades de la aplicación.

"Utilizar un gestor de seguridad" en la página 389
Si ejecuta la aplicación JGSS teniendo habilitado un gestor de seguridad Java, debe asegurarse de que la aplicación y JGSS tienen los permisos necesarios.

Configurar JGSS para que utilice el proveedor de JGSS de System i5:

IBM JGSS emplea por defecto el proveedor de Java puro. También puede elegir la opción de utilizar el proveedor de JGSS nativo de System i5.

Requisitos de software

El proveedor de JGSS nativo de System i5 debe poder acceder a las clases de IBM Toolbox para Java. En el subtema que sigue encontrará instrucciones para acceder a IBM Toolbox para Java.

Asegúrese de que ha configurado el servicio de autenticación de red.

Nota: En las siguientes instrucciones, ${java.home} indica la vía de acceso a la versión de Java que está utilizando en el servidor. Por ejemplo, si utiliza J2SDK, versión 1.4, ${java.home} es /QIBM/ProdData/Java400/jdk14. No olvide sustituir ${java.home} en los mandatos por la vía real de acceso al directorio inicial de Java.

Para configurar JGSS con el fin de que utilice el proveedor de JGSS nativo de System i5, realice estas tareas:

Añadir un enlace simbólico

Para añadir un enlace simbólico al directorio de extensiones para el archivo ibmjgssiseriesprovider.jar, vaya a una línea de mandatos de i5/OS, teclee el mandato siguiente (en una sola línea) y pulse **Intro**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssiseriesprovider.jar')
NEWLNK('${java.home}/lib/ext/ibmjgssiseriesprovider.jar')
```

Tras añadir un enlace simbólico al directorio de extensiones para el archivo ibmjgssiseriesprovider.jar, el cargador de clases de extensión cargará el archivo JAR.

Añadir el proveedor a la lista de proveedores de seguridad

Añada el proveedor nativo a la lista de proveedores de seguridad del archivo java.security.

1. Abra ${java.home}/lib/security/java.security para editarlo.
2. Busque la lista de proveedores de seguridad. Debe encontrarse cerca del principio del archivo java.security y debe tener un aspecto parecido al siguiente:

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=com.sun.rsajca.Provider
   security.provider.3=com.ibm.crypto.provider.IBMJCE
   security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
   ```

3. Añada el proveedor de JGSS de System i5 a la lista de proveedores de seguridad, antes del proveedor de Java original. Dicho de otro modo, añada com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider a la lista con un número inferior al de com.ibm.jgss.IBMJGSSProvider y, a continuación, actualice la posición de IBMJGSSProvider. Por ejemplo:

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=com.sun.rsajca.Provider
   security.provider.3=com.ibm.crypto.provider.IBMJCE
   security.provider.4=com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider
   security.provider.5=com.ibm.security.jgss.IBMJGSSProvider
   ```

   Observe que IBMJGSSiSeriesProvider ha pasado a ser la cuarta entrada de la lista y IBMJGSSProvider se ha convertido en la quinta entrada. Asimismo, compruebe que los números de entrada de la lista de proveedores de seguridad son secuenciales y que cada entrada aumenta el número de entrada en un solo dígito.
4. Guarde y cierre el archivo java.security.

Conceptos relacionados

"Proveedores de JGSS" en la página 388
IBM JGSS incluye un proveedor de JGSS nativo de System i5 y un proveedor de JGSS Java puro. El proveedor que elija utilizar dependerá de las necesidades de la aplicación.

"Configurar el System i5 para que utilice JGSS con J2SDK, versión 1.3" en la página 385
Cuando utiliza Java 2 Software Development Kit (J2SDK), versión 1.3, en su servidor, debe preparar y configurar el servidor para que utilice JGSS. En la configuración predeterminada se emplea el proveedor de JGSS Java puro.

Información relacionada

Servicio de autenticación de red

Habilitar el proveedor de JGSS nativo de System i5 para que acceda a IBM Toolbox para Java:

El proveedor de JGSS nativo de System i5 debe poder acceder a las clases de IBM Toolbox para Java.

Utilice una de las opciones siguientes para permitir que IBM JGSS acceda al archivo jt400Native.jar de Toolbox para Java:

Nota: En las siguientes instrucciones, *${java.home}* indica la vía de acceso a la ubicación de la versión de Java que esté utilizando en el servidor. Por ejemplo, si utiliza J2SE, versión 1.5, *${java.home}* es /QIBM/ProdData/Java400/jdk15. No olvide sustituir *${java.home}* en los mandatos por la vía real de acceso al directorio inicial de Java.

Opción 1: colocar un enlace simbólico en el directorio de extensiones Java

Para colocar un enlace que lleve al archivo jt400Native.jar en el directorio de extensiones Java, vaya a una línea de mandatos de i5/OS, teclee el siguiente mandato (en una sola línea) y pulse **Intro**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('${java.home}/lib/ext/jt400Native.jar')
```

Nota: Al colocar un enlace simbólico a jt400Native.jar en el directorio de extensiones Java, todos los usuarios del J2SE deben utilizar forzosamente esta versión de jt400Native.jar. Esto puede no ser conveniente si diversos usuarios necesitan distintas versiones de las clases de IBM Toolbox para Java.

Opción 2: coloque un enlace simbólico en su propio directorio de extensiones

Para enlazar el archivo jt400Native.jar con su propio directorio, vaya a una línea de mandatos de i5/OS, teclee el siguiente mandato (en una sola línea) y pulse **Intro**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('<su directorio de extensiones>/jt400Native.jar')
```

Cuando llame al programa, emplee el formato siguiente para el mandato Java:

```
java -Djava.ext.dirs=<su directorio de extensiones>:${java.home}/lib/ext:/QIBM/UserData/Java400/ext
```

Configurar el System i5 para que utilice JGSS con J2SDK, versión 1.4 o posterior:

Cuando utiliza Java 2 Software Development Kit (J2SDK), versión 1.4 o superior, en el servidor, JGSS ya está configurado. En la configuración predeterminada se emplea el proveedor de JGSS Java puro.

Cambiar los proveedores de JGSS

Puede configurar JGSS para que utilice el proveedor de JGSS nativo de System i5, en lugar del proveedor de JGSS Java puro. Entonces, después de haber configurado JGSS para que utilice el proveedor nativo, le resultará fácil conmutar entre los dos proveedores. Para obtener más información, consulte los siguientes temas:

- "Proveedores de JGSS"
- "Configurar JGSS para que utilice el proveedor de JGSS de System i5" en la página 386

Gestores de seguridad

Si se propone ejecutar la aplicación JGSS teniendo habilitado un gestor de seguridad Java, vea: Utilizar un gestor de seguridad.

Proveedores de JGSS:

IBM JGSS incluye un proveedor de JGSS nativo de System i5 y un proveedor de JGSS Java puro. El proveedor que elija utilizar dependerá de las necesidades de la aplicación.

Las características que ofrece el proveedor de JGSS Java puro son las siguientes:
- Garantiza el mayor nivel de portabilidad de la aplicación.
- Funciona con la interfaz de inicio de sesión de Kerberos JAAS opcional.
- Es compatible con las herramientas de gestión de credenciales de Kerberos Java.

Las características que ofrece el proveedor de JGSS nativo de System i5 son las siguientes:
- Utiliza las bibliotecas Kerberos nativas de System i5.
- Es compatible con las herramientas de gestión de credenciales de Kerberos Qshell.
- Las aplicaciones JGSS se ejecutan con mayor rapidez.

Nota: Ambos proveedores de JGSS cumplen la especificación GSS-API y, por consiguiente, son compatibles entre sí. En otras palabras, una aplicación que utilice el proveedor de JGSS Java puro puede interoperar con una aplicación que utilice el proveedor de JGSS nativo de System i5.

Cambiar el proveedor JGSS

Le resultará fácil cambiar el proveedor JGSS mediante una de las opciones siguientes:
- Edite la lista de proveedores de seguridad de `${java.home}/lib/security/java.security`.

  Nota: ${java.home} indica la vía de acceso a la ubicación de la versión de Java que esté utilizando en el servidor. Por ejemplo, si utiliza J2SE, versión 1.5, ${java.home} es /QIBM/ProdData/Java400/jdk15.
- Especifique el nombre del proveedor en la aplicación JGSS mediante GSSManager.addProviderAtFront() o GSSManager.addProviderAtEnd(). Para obtener más información, consulte el javadoc de GSSManager.

Utilizar un gestor de seguridad:

Si ejecuta la aplicación JGSS teniendo habilitado un gestor de seguridad Java, debe asegurarse de que la aplicación y JGSS tienen los permisos necesarios.

Permisos de JVM:

Además de las comprobaciones de control de acceso efectuadas por JGSS, la máquina virtual Java (JVM) lleva a cabo comprobaciones de autorización al acceder a una gran variedad de recursos, como los archivos, las propiedades Java, los paquetes y los sockets.

La lista siguiente indica los permisos necesarios al utilizar las funciones de JAAS de JGSS o al emplear JGSS con un gestor de seguridad:
- 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″

Información relacionada

Permisos en el kit Java 2 SDK de Sun Microsystems, Inc.

Comprobaciones de permisos de JAAS:

IBM JGSS realiza comprobaciones de permisos en tiempo de ejecución cuando el programa habilitado para JAAS utiliza credenciales y accede a servicios. Puede inhabilitar esta característica JAAS opcional estableciendo que la propiedad Java javax.security.auth.useSubjectCredsOnly sea igual a false. Además, JGSS lleva a cabo comprobaciones de permisos únicamente cuando la aplicación se ejecuta con un gestor de seguridad.

JGSS realiza comprobaciones de permisos en relación con la política Java que está en vigor para el contexto de control de acceso actual. JGSS lleva a cabo las siguientes comprobaciones de permisos específicos:

- javax.security.auth.kerberos.DelegationPermission
- javax.security.auth.kerberos.ServicePermission

Comprobación DelegationPermission

DelegationPermission permite el control por parte de la política de seguridad del uso de las funciones de reenvío de tickets y servicio proxy de Kerberos. Mediante estas funciones, un cliente puede permitir que un servicio actúe en nombre del cliente.

DelegationPermission toma dos argumentos, en el orden siguiente:

1. El principal de subordinado, que es el nombre del principal de servicio que actúa en nombre del cliente y bajo su autorización.
2. El nombre del servicio que el cliente desea permitir que el principal de subordinado utilice.

Ejemplo: utilizar la comprobación DelegationPermission

En el ejemplo siguiente, superSecureServer es el principal de subordinado y krbtgt/REALM.IBM.COM@REALM.IBM.COM es el servicio que se desea permitir que superSecureServer utilice en nombre del cliente. En este caso, el servicio es el ticket de otorgamiento de tickets del cliente, lo que significa que superSecureServer puede obtener un ticket para cualquier servicio en nombre del cliente.

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
```

En el ejemplo anterior, DelegationPermission otorga al cliente permiso para obtener un nuevo ticket de otorgamiento de tickets del centro de distribución de claves (KDC) que solo superSecureServer puede utilizar. Una vez que el cliente ha enviado el nuevo ticket de otorgamiento de tickets a superSecureServer, superSecureServer tiene la posibilidad de actuar en nombre del cliente.

El ejemplo siguiente permite al cliente obtener un nuevo ticket que permita a superSecureServer acceder únicamente al servicio ftp en nombre del cliente:

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"ftp/ftp.ibm.com@REALM.IBM.COM\"";
```

Comprobación ServicePermission

Las comprobaciones ServicePermission restringen el uso de credenciales para la iniciación y la aceptación del contexto. Un iniciador de contexto debe tener permiso para iniciar un contexto. Del mismo modo, un aceptante de contexto debe tener permiso para aceptar un contexto.

Ejemplo: utilizar la comprobación ServicePermission

El ejemplo siguiente permite que el lado del cliente inicie un contexto con el servicio ftp, otorgando permiso al cliente:

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "initiate";
```

El ejemplo siguiente permite que el lado del servidor acceda a la clave secreta del servicio ftp y la utilice, otorgando permiso al servidor:

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "accept";
```

Información relacionada

Documentación de Sun Microsystems, Inc.

Ejecutar aplicaciones IBM JGSS

Al API de servicios IBM de seguridad genéricos (GSS) Java (JGSS) 1.0 apantalla las aplicaciones seguras ante las complejidades y peculiaridades de los distintos mecanismos de seguridad subyacentes. JGSS utiliza características proporcionadas por el servicio de autenticación y autorización Java (JAAS) y la extensión de criptografía Java (JCE) de IBM.

Las características de JGSS son las siguientes:

- Autenticación de identidades
- Integridad y confidencialidad de mensajes
- Interfaz de inicio de sesión de Kerberos JAAS opcional y comprobaciones de autorización

Obtener credenciales de Kerberos y crear claves secretas:

GSS-API no define ningún método para obtener credenciales. Por lo tanto, el mecanismo Kerberos de IBM JGSS exige que el usuario obtenga credenciales de Kerberos. Este tema le enseña a obtener credenciales de Kerberos y a crear claves secretas, y le explica cómo utilizar JAAS para realizar inicios de sesión y comprobaciones de autorización de Kerberos, así como revisar una lista de los permisos de JAAS necesarios para la máquina virtual Java (JVM).

Puede obtener credenciales mediante uno de estos métodos:

- Herramientas Kinit y Ktab
- Interfaz de inicio de sesión de Kerberos JAAS opcional

Herramientas Kinit y Ktab:

El proveedor de JGSS que elija determinará qué herramientas utilizará para obtener las credenciales y claves secretas de Kerberos.

Con el proveedor de JGSS Java puro

Si se propone utilizar el proveedor de JGSS Java puro, emplee las herramientas Kinit y Ktab de IBM JGSS para obtener credenciales y claves secretas. Las herramientas Kinit y Ktab utilizan interfaces de línea de mandatos y proporcionan opciones similares a las de otras versiones.

- Puede obtener credenciales de Kerberos mediante la herramienta Kinit. Esta herramienta establece contacto con el centro de distribución de Kerberos (KDC) y obtiene un ticket de otorgamiento de tickets (TGT). El TGT permite acceder a otros servicios habilitados para Kerberos, entre ellos los que utilizan GSS-API.
- Un servidor puede obtener una clave secreta mediante la herramienta Ktab. JGSS almacena la clave secreta en el archivo de tabla de claves del servidor. Hallará más información en la documentación Java de Ktab.

La aplicación también puede emplear la interfaz de inicio de sesión de JAAS a fin de obtener tickets TGT y claves secretas.

Con el proveedor de JGSS nativo de System i5

Si se propone utilizar el proveedor de JGSS nativo de System i5, emplee las utilidades kinit y klist de Qshell.

Conceptos relacionados

"Interfaz de inicio de sesión de Kerberos JAAS"
IBM JGSS presenta una interfaz de inicio de sesión Kerberos del servicio de autenticación y autorización Java (JAAS). Puede inhabilitar esta característica estableciendo que la propiedad Java javax.security.auth.useSubjectCredsOnly sea igual a false.

Referencia relacionada

"Clase Kinit de com.ibm.security.krb5.internal.tools" en la página 381
Herramienta Kinit para obtener tickets Kerberos v5.

"Clase Ktab de com.ibm.security.krb5.internal.tools" en la página 383
Esta clase se puede ejecutar como herramienta de línea de mandatos para ayudar al usuario a gestionar las entradas de la tabla de claves. Las funciones disponibles son listar/añadir/actualizar/suprimir clave(s) de servicio.

Interfaz de inicio de sesión de Kerberos JAAS:

IBM JGSS presenta una interfaz de inicio de sesión Kerberos del servicio de autenticación y autorización Java (JAAS). Puede inhabilitar esta característica estableciendo que la propiedad Java javax.security.auth.useSubjectCredsOnly sea igual a false.

Nota: Aunque el proveedor de JGSS Java puro puede utilizar la interfaz de inicio de sesión, el proveedor de JGSS nativo de System i5 no puede hacerlo.

Para obtener más información sobre JAAS, vea: Servicio de autenticación y autorización Java (JAAS).

Permisos de JAAS y JVM

Si utiliza un gestor de seguridad, debe comprobar que la aplicación y JGSS tienen los permisos de JVM y JAAS necesarios. Para obtener más información, consulte Utilizar un gestor de seguridad.

Opciones del archivo de configuración de JAAS

La interfaz de inicio de sesión requiere un archivo de configuración de JAAS que especifique com.ibm.security.auth.module.Krb5LoginModule como el módulo de inicio de sesión que se empleará. La tabla siguiente muestra las opciones que admite Krb5LoginModule. Tenga en cuenta que las opciones no

son sensibles a las mayúsculas y minúsculas.

Nombre de opción	Valor	Valor predeterminado	Descripción
principal	<serie>	Ninguno; se solicita.	Nombre de principal de Kerberos
credsType	initiator | acceptor | both	initiator	Tipo de credencial de JGSS
forwardable	true | false	false	Si debe adquirirse un ticket de otorgamiento de tickets (TGT) reenviable
proxiable	true | false	false	Si debe adquirirse un TGT que admita proxy
useCcache	<URL>	No utilizar la caché de credenciales	Recuperar el TGT de la caché de credenciales especificada
useKeytab	<URL>	No utilizar la tabla de claves	Recuperar la clave secreta de la tabla de claves especificada
useDefaultCcache	true | false	No utilizar la caché de credenciales predeterminada	Recuperar el TGT de la caché de credenciales predeterminada
useDefaultKeytab	true | false	No utilizar la tabla de claves predeterminado	Recuperar la clave secreta de la tabla de claves especificada

Para ver un sencillo ejemplo de cómo utilizar Krb5LoginModule, consulte el archivo de configuración de inicio de sesión de JAAS de ejemplo.

Incompatibilidades de opciones

Algunas opciones de Krb5LoginModule, sin incluir el nombre de principal, son incompatibles entre ellas, lo que significa que no se pueden especificar juntas. La tabla siguiente indica las opciones de módulo de inicio de sesión compatibles e incompatibles.

Los indicadores de la tabla describen la relación entre las dos opciones asociadas:

- X = Incompatible
- N/A = Combinación no aplicable
- Blanco = Compatible

Krb5LoginModule option	credsType initiator	credsType acceptor	credsType both	forward	proxy	use Ccache	use Keytab	useDefault Ccache	useDefault Keytab
credsType=initiator		N/A	N/A				X		X
credsType=acceptor	N/A		N/A	X	X	X		X	
credsType=both	N/A	N/A							
forwardable		X				X	X	X	X
proxiable		X				X	X	X	X
useCcache		X		X	X		X	X	X
useKeytab	X			X	X	X		X	X
useDefaultCcache		X		X	X	X	X		X
useDefaultKeytab	X			X	X	X	X	X	

Opción de nombre de principal

Puede especificar un nombre de principal junto con cualquier otra opción. Si no especifica un nombre de principal, Krb5LoginModule puede solicitar al usuario un nombre de principal. Krb5LoginModule solicitará o no esta información al usuario en función de las demás opciones especificadas.

Formato del nombre de principal de servicio

Debe emplear uno de los formatos siguientes para especificar un nombre de principal de servicio:

- <nombre_servicio> (por ejemplo, superSecureServer)
- <nombre_servicio>@<host> (por ejemplo, superSecureServer@myhost)

En el segundo formato, <host> es el nombre de host de la máquina donde reside el servicio. Aunque no está obligado a ello, puede utilizar un nombre de host totalmente calificado.

Nota: JAAS reconoce determinados caracteres como delimitadores. Si emplea alguno de los caracteres siguientes en una serie de JAAS (como un nombre de principal), escriba el carácter entre comillas:

```
_  (subrayado)
: (dos puntos)
/ (barra inclinada)
\  (barra inclinada invertida)
```

Solicitar el nombre de principal y la contraseña

Las opciones que especifique en el archivo de configuración de JAAS determinarán si el inicio de sesión de Krb5LoginModule será o no interactivo.

- Un inicio de sesión no interactivo no solicita ninguna información.
- Un inicio de sesión interactivo solicita el nombre de principal, la contraseña o ambos.

Inicios de sesión no interactivos

El inicio de sesión se efectuará de modo no interactivo si especifica el tipo de credencial initiator (`credsType=initiator`) y lleva a cabo una de las acciones siguientes:

- Especificar la opción useCcache
- Especificar la opción useDefaultCcache en true

El inicio de sesión también se efectuará de modo no interactivo si especifica el tipo de credencial acceptor o both (`credsType=acceptor` o `credsType=both`) y lleva a cabo una de las acciones siguientes:

- Especificar la opción useKeytab
- Especificar la opción useDefaultKeytab en true

Inicios de sesión interactivos

Otras configuraciones hacen que el módulo de inicio de sesión solicite un nombre de principal y una contraseña a fin de obtener un TGT de un KDC de Kerberos. El módulo de inicio de sesión solo solicita una contraseña si se especifica la opción principal.

Los inicios de sesión interactivos requieren que la aplicación especifique com.ibm.security.auth.callback.Krb5CallbackHandler como manejador de retornos de llamada al crear el contexto de inicio de sesión. El manejador de retornos de llamada es el encargado de solicitar la entrada.

Opción de tipo de credencial

Si se requiere que el tipo de credencial sea tanto initiator como acceptor (`credsType=both`), Krb5LoginModule obtiene un TGT y una clave secreta. El módulo de inicio de sesión utiliza el TGT para iniciar contextos y la clave secreta para aceptar contextos. El archivo de configuración de JAAS debe contener suficiente información para permitir al módulo de inicio de sesión adquirir los dos tipos de credenciales.

Para los tipos de credencial acceptor y both, el módulo de inicio de sesión toma un principal de servicio.

Archivos de configuración y política:

JGSS y JAAS dependen de varios archivos de configuración y política. Debe editar estos archivos para que se ajusten al entorno y la aplicación que utiliza. Si no emplea JAAS con JGSS, puede omitir los archivos de configuración y política de JAAS.

Nota: En las siguientes instrucciones, ${java.home} indica la vía de acceso a la versión de Java que está utilizando en el servidor. Por ejemplo, si utiliza J2SE, versión 1.5, ${java.home} es /QIBM/ProdData/Java400/jdk15. No olvide que, en los valores de las propiedades, ${java.home} debe indicar la vía de acceso real al directorio inicial Java.

Archivo de configuración de Kerberos

Para IBM JGSS se necesita un archivo de configuración de Kerberos. El nombre por omisión y la ubicación del archivo de configuración de Kerberos dependen del sistema operativo que se utilice. JGSS utiliza el orden siguiente para buscar el archivo de configuración por omisión:

1. El archivo al que hace referencia la propiedad Java java.security.krb5.conf
2. ${java.home}/lib/security/krb5.conf
3. c:\winnt\krb5.ini en las plataformas Microsoft Windows
4. /etc/krb5/krb5.conf en las plataformas Solaris
5. /etc/krb5.conf en otras plataformas Unix

Archivo de configuración de JAAS

El uso de la función de inicio de sesión de JAAS requiere un archivo de configuración de JAAS. Puede especificar el archivo de configuración de JAAS estableciendo una de las propiedades siguientes:

- La propiedad Java java.security.auth.login.config del sistema
- La propiedad de seguridad login.config.url.<entero> en el archivo ${java.home}/lib/security/java.security

Hallará más información en el sitio Web del servicio de autenticación y autorización Java (JAAS) de Sun.

Archivo de política de JAAS

Al utilizar la implementación de política por omisión, JGSS otorga permisos de JAAS a las entidades anotando los permisos en un archivo de política. Puede especificar el archivo de política de JAAS estableciendo una de las propiedades siguientes:

- La propiedad Java java.security.policy del sistema
- La propiedad de seguridad policy.config.url.<enteror> en el archivo ${java.home}/lib/security/java.security

Si utiliza J2SDK, versión 1.4 o posterior, la especificación de un archivo de política aparte para JAAS es opcional. El proveedor de política por omisión en J2SDK, versión 1.4 o posterior, da soporte a las entradas de archivo de política que requiere JAAS.

Hallará más información en el sitio Web del servicio de autenticación y autorización Java (JAAS) de Sun.

Archivo de propiedades Java maestro de seguridad

La máquina virtual Java (JVM) emplea importantes propiedades de seguridad que se establecen editando el archivo de propiedades Java maestro de seguridad. Este archivo, cuyo nombre es java.security, suele residir en el directorio ${java.home}/lib/security del servidor.

La lista siguiente describe varias propiedades de seguridad relevantes para utilizar JGSS. Utilice las descripciones a modo de guía para editar el archivo java.security.

Nota: Cuando corresponde, las descripciones incluyen los valores adecuados que son necesarios para ejecutar los ejemplos de JGSS.

security.provider.<entero>: el proveedor JGSS que desea utilizar. Estáticamente también registra las clases de proveedor criptográfico. IBM JGSS emplea servicos criptográficos y otros servicios de seguridad proporcionados por el proveedor de JCE IBM. Especifique los paquetes sun.security.provider.Sun y com.ibm.crypto.provider.IBMJCE exactamente igual que en el ejemplo siguiente:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

policy.provider: clase de manejador de política del sistema. Por ejemplo:

```
policy.provider=sun.security.provider.PolicyFile
```

policy.url.<entero>: URL de los archivos de política. Para emplear el archivo de política de ejemplo, incluya una entrada como:

```
policy.url.1=file:/home/user/jgss/config/java.policy
```

login.configuration.provider: clase de manejador de configuración de inicio de sesión de JAAS, como por ejemplo:

```
login.configuration.provider=com.ibm.security.auth.login.ConfigFile
```

auth.policy.provider: clase de manejador de política de control de acceso basado en el principal de JAAS, como por ejemplo:

```
auth.policy.provider=com.ibm.security.auth.PolicyFile
```

login.config.url.<entero>: URL para los archivos de configuración de inicio de sesión de JAAS. Para emplear el archivo de configuración de ejemplo, incluya una entrada parecida a la siguiente:

```
login.config.url.1=file:/home/user/jgss/config/jaas.conf
```

auth.policy.url.<entero>: URL para los archivos de política de JAAS. Puede incluir construcciones basadas en el principal y el origen del código en el archivo de política de JAAS. Para emplear el archivo de política de ejemplo, incluya una entrada como:

```
auth.policy.url.1=file:/home/user/jgss/config/jaas.policy
```

Caché de credenciales y tabla de claves de servidor

El usuario principal mantiene sus credenciales de Kerberos en una caché de credenciales. Un principal de servicio mantiene su clave secreta en una tabla de claves. En tiempo de ejecución, IBM JGSS localiza estas cachés de varias maneras, que son:

Caché de credenciales de usuario

JGSS utiliza el orden siguiente para localizar la caché de credenciales de usuario:

1. El archivo al que hace referencia la propiedad Java KRB5CCNAME
2. El archivo al que hace referencia la variable de entorno KRB5CCNAME
3. /tmp/krb5cc_<uid> en sistemas Unix
4. ${user.home}/krb5cc_${user.name}
5. ${user.home}/krb5cc (si no es posible obtener ${user.name})

Tabla de claves de servidor

JGSS utiliza el orden siguiente para localizar el archivo de tabla de claves de servidor:

1. El valor de la propiedad Java KRB5_KTNAME

2. La entrada default_keytab_name de la stanza libdefaults del archivo de configuración de Kerberos
3. ${user.home}/krb5_keytab

Desarrollar aplicaciones IBM JGSS

Utilizar JGSS para desarrollar aplicaciones seguras. Obtenga información sobre cómo generar símbolos de transporte, crear objetos JGSS, establecer un contexto y mucho más.

Para desarrollar aplicaciones JGSS, debe estar familiarizado con la especificación GSS-API de alto nivel y con la especificación de enlaces Java. IBM JGSS 1.0 se basa principalmente en estas especificaciones y está en conformidad con ellas. Consulte los siguientes enlaces para obtener más información.

- RFC 2743: Interfaz de programación de aplicaciones (API) de servicios de seguridad genéricos (GSS) Versión 2, Actualización 1
- RFC 2853: API de servicios de seguridad genéricos (GSS) Versión 2: enlaces Java

Pasos de programación de aplicaciones IBM JGSS:

Son múltiples los pasos necesarios para desarrollar una aplicación JGSS, tales como utilizar símbolos de transporte, crear los objetos JGSS necesarios, establecer y suprimir un contexto y emplear los servicios por mensaje.

Las operaciones de una aplicación JGSS siguen el modelo operativo de GSS-API (interfaz de programación de aplicaciones de servicios de seguridad genéricos). Para obtener información sobre los conceptos importantes de las operaciones de JGSS, consulte Conceptos de JGSS.

Símbolos de transporte de JGSS

Algunas de las operaciones importantes de JGSS generan símbolos en formato de matrices de bytes Java. La aplicación es la encargada de reenviar las símbolos de un igual JGSS al otro. JGSS no restringe en modo alguno el protocolo que utiliza la aplicación para transportar símbolos. Las aplicaciones pueden transportar símbolos JGSS junto con otros datos de aplicación (es decir, datos que no son de JGSS). Sin embargo, las operaciones de JGSS aceptan y utilizan únicamente símbolos específicas de JGSS.

Secuencia de operaciones en una aplicación JGSS

Las operaciones de JGSS requieren determinadas construcciones de programación que se deben emplear en el orden indicado a continuación. Cada uno de los pasos se aplica tanto al iniciador como al aceptante.

Nota: La información contiene fragmentos de código de ejemplo que muestran cómo utilizar las API de JGSS de alto nivel y suponen que la aplicación importa el paquete org.ietf.jgss. Aunque muchas de las API de alto nivel se cargan a posteriori, los fragmentos de código solo muestran los formatos más empleados de esos métodos. Por supuesto, utilice los métodos de las API que mejor se adapten a sus necesidades.

Crear un GSSManager:

La clase abstracta GSSManager actúa como una fábrica para crear objetos JGSS.

La clase abstracta GSSManager crea lo siguiente:

- GSSName
- GSSCredential
- GSSContext

GSSManager también tiene métodos para determinar los mecanismos de seguridad y tipos de nombres soportados y especificar proveedores JGSS. Utilice el método estático de GSSManager getInstance para crear una instancia del GSSManager por omisión:

```
GSSManager manager = GSSManager.getInstance();
```

Crear un GSSName:

GSSName representa la identidad de un principal GSS-API. Un GSSName puede contener numerosas representaciones del principal, una para cada mecanismo subyacente soportado. Un GSSName que contiene una sola representación de nombre se denomina nombre de mecanismo (MN).

GSSManager tiene varios métodos cargados a posteriori para crear un GSSName a partir de una serie o una matriz contigua de bytes. Los métodos interpretan la serie o matriz de bytes según un tipo de nombre especificado. Por lo general, se utilizan los métodos de matriz de bytes de GSSName para volver a formar un nombre exportado. El nombre exportado normalmente es un nombre de mecanismo de tipo GSSName.NT_NOMBRE_EXPORT. Algunos de estos métodos permiten especificar un mecanismo de seguridad con el que se creará el nombre.

Ejemplos: utilizar GSSName

El fragmento de código básico siguiente muestra cómo utilizar GSSName.

Nota: Especifique series de nombre de servicio de Kerberos como servicio o servicio@sistemaprincipal donde servicio es el nombre del servicio y sistemaprincipal es el nombre de sistema principal de la máquina en el que se ejecuta el servicio. Aunque no está obligado a ello, puede utilizar un nombre de sistema principal totalmente calificado. Si omite la parte @<sistemaprincipal> de la serie, GSSName utiliza el nombre de sistema principal local.

```
// Crear GSSName para el usuario foo.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);

// Crear un nombre de mecanismo Kerberos V5 para el usuario foo.
Oid krb5Mech = new Oid("1.2.840.113554.1.2.2");
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME, krb5Mech);

// Crear un nombre de mecanismo a partir de un nombre no de mecanismo mediante
// el método canonicalize de GSSName.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);
GSSName fooKrb5Name = fooName.canonicalize(krb5Mech);
```

Crear un GSSCredential:

Un GSSCredential contiene toda la información criptográfica necesaria para crear un contexto en nombre de un principal y puede contener información de credenciales para varios mecanismos.

GSSManager tiene tres métodos de creación de credenciales. Dos de los métodos toman para los parámetros un GSSName, el tiempo de vida de la credencial, uno o varios mecanismos de los que se obtendrán credenciales y el tipo de uso de credenciales. El tercer método toma solo un tipo de uso y utiliza los valores predeterminados para los demás parámetros. Al especificar un mecanismo nulo también se utiliza el mecanismo por omisión. Al especificar una matriz nula de mecanismos, el método devuelve credenciales para el conjunto por omisión de mecanismos.

Nota: Dado que IBM JGSS solo admite el mecanismo Kerberos V5, ese es el mecanismo predeterminado.

La aplicación solo puede crear uno de estos tipos de credenciales (*initiate, accept* o *initiate and accept*) en un momento dado.

- Un iniciador de contexto crea credenciales *initiate.*
- Un aceptante crea credenciales *accept*.
- Un aceptante que también se comporta como iniciador crea credenciales *initiate and accept* .

Ejemplos: obtener credenciales

El ejemplo siguiente obtiene las credenciales por omisión para un iniciador:

```
GSSCredentials fooCreds = manager.createCredentials(GSSCredential.INITIATE)
```

El ejemplo siguiente obtiene las credenciales de Kerberos V5 para el iniciador foo con el periodo de validez por omisión:

```
GSSCredential fooCreds = manager.createCredential(fooName, GSSCredential.DEFAULT_LIFETIME,
                                                  krb5Mech,GSSCredential.INITIATE);
```

El ejemplo siguiente obtiene una credencial de aceptante con todos los valores predeterminados:

```
GSSCredential serverCreds = manager.createCredential(null, GSSCredential.DEFAULT_LIFETIME,
                                                     (Oid)null, GSSCredential.ACCEPT);
```

Crear GSSContext:

IBM JGSS admite dos métodos proporcionados por GSSManager para crear un contexto. El primero es un método empleado por el iniciador de contexto y el otro es un método empleado por el aceptante de contexto.

Nota: GSSManager proporciona un tercer método para crear un contexto que implica volver a crear contextos exportados anteriormente. Sin embargo, como el mecanismo Kerberos V5 de IBM JGSS no admite el uso de contextos exportados, IBM JGSS no permite utilizar este método.

La aplicación no puede emplear un contexto de iniciador para la aceptación del contexto, ni puede utilizar un contexto de aceptante para la iniciación del contexto. Ambos métodos soportados para crear un contexto requieren una credencial como entrada. Si el valor de la credencial es nulo, JGSS utiliza la credencial por omisión.

Ejemplos: utilizar GSSContext

El ejemplo siguiente crea un contexto con el que el principal (foo) puede iniciar un contexto con el igual (superSecureServer) en el sistema principal (securityCentral). El ejemplo especifica el igual como superSecureServer@securityCentral. El contexto creado es válido durante el periodo por omisión:

```
GSSName serverName = manager.createName("superSecureServer@securityCentral",
                                        GSSName.NT_HOSTBASED_SERVICE, krb5Mech);
GSSContext fooContext = manager.createContext(serverName, krb5Mech, fooCreds,
                                              GSSCredential.DEFAULT_LIFETIME);
```

El ejemplo siguiente crea un contexto para superSecureServer a fin de aceptar los contextos iniciados por cualquier igual:

```
GSSContext serverAcceptorContext = manager.createContext(serverCreds);
```

Tenga en cuenta que la aplicación puede crear y utilizar de forma simultánea ambos tipos de contextos.

Solicitar servicios de seguridad JGSS opcionales:

La aplicación puede solicitar cualquiera de los diversos servicios de seguridad opcionales. IBM JGSS soporta varios servicios.

Los servicios soportados son:

- Delegación
- Autenticación mutua
- Detección de reproducción
- Detección de fuera de secuencia
- Confidencialidad por mensaje disponible
- Integridad por mensaje disponible

Para solicitar un servicio opcional, la aplicación debe solicitarlo explícitamente empleando el método de petición adecuado en el contexto. Solo un iniciador puede solicitar estos servicios opcionales. El iniciador debe efectuar la petición antes de que comience el establecimiento del contexto.

Para obtener más información sobre los servicios opcionales, vea el soporte de servicios opcionales en la RFC 2743 Interfaz de programación de aplicaciones (API) de servicios de seguridad genéricos (GSS) Versión 2, Actualización 1 del equipo negociador de ingenieros de Internet (IETF).

Ejemplo: solicitar servicios opcionales

En el ejemplo siguiente, un contexto (fooContext) efectúa solicitudes para habilitar los servicios de autenticación mutua y delegación:

```
fooContext.requestMutualAuth(true);
     fooContext.requestCredDeleg(true);
```

Establecer un contexto JGSS:

Los dos iguales que se comunican deben establecer un contexto de seguridad en el que pueden utilizar servicios por mensaje.

El iniciador llama a initSecContext() en su contexto, que devuelve un símbolo a la aplicación del iniciador. La aplicación del iniciador transporta el símbolo de contexto a la aplicación del aceptante. El aceptante llama a acceptSecContext() en su contexto especificando el símbolo de contexto recibido del iniciador. Según el mecanismo subyacente y los servicios opcionales que ha seleccionado el iniciador, acceptSecContext() puede generar un símbolo que la aplicación del aceptante tiene que reenviar a la aplicación del iniciador. A continuación, la aplicación del iniciador utiliza el símbolo recibido para llamar a initSecContext() una vez más.

Una aplicación puede efectuar varias llamadas a GSSContext.initSecContext() y GSSContext.acceptSecContext(). Asimismo, una aplicación puede intercambiar múltiples símbolos con un igual durante el establecimiento del contexto. Por consiguiente, el método habitual para establecer un contexto utiliza un bucle para llamar a GSSContext.initSecContext() o GSSContext.acceptSecContext() hasta que las aplicaciones establecen el contexto.

Ejemplo: establecer el contexto

El ejemplo siguiente muestra el lado del iniciador (foo) del establecimiento del contexto:

```
byte array[] inToken = null; // El símbolo de entrada es nulo para la primera llamada
int inTokenLen = 0;

do {
    byte[] outToken = fooContext.initSecContext(inToken, 0, inTokenLen);

    if (outToken != null) {
        send(outToken); // símbolo de transporte al aceptante
    }

    if( !fooContext.isEstablished()) {
        inToken = receive(); // símbolo de recepción del aceptante
          inTokenLen = inToken.length;
    }
} while (!fooContext.isEstablished());
```

El ejemplo siguiente muestra el lado del aceptante del establecimiento del contexto:

```
// El código del aceptante para establecer el contexto puede ser:
do {
    byte[] inToken = receive(); // símbolo de recepción del iniciador
    byte[] outToken =
        serverAcceptorContext.acceptSecContext(inToken, 0, inToken.length);
```

```
        if (outToken != null) {
            send(outToken); // símbolo de transporte al iniciador
        }
    } while (!serverAcceptorContext.isEstablished());
```

Utilizar los servicios por mensaje de JGSS:

Tras establecer un contexto de seguridad, dos iguales que se comunican pueden intercambiar mensajes seguros en el contexto establecido.

Cualquiera de los dos iguales puede originar un mensaje seguro, independientemente de si ha actuado como iniciador o como aceptante al establecer el contexto. Para que el mensaje sea seguro, IBM JGSS calcula un código de integridad de mensaje (MIC) criptográfico a través del mensaje. De modo opcional, IBM JGSS puede hacer que el mecanismo de Kerberos V5 cifre el mensaje para ayudar a garantizar la privacidad.

Enviar mensajes

IBM JGSS proporciona dos conjuntos de métodos para proteger los mensajes: wrap() y getMIC().

Utilizar wrap()

El método wrap lleva a cabo las acciones siguientes:
- Calcula un MIC
- Cifra el mensaje (opcional)
- Devuelve un símbolo

La aplicación que efectúa la llamada utiliza la clase MessageProp junto con GSSContext para especificar si debe aplicarse cifrado al mensaje.

El símbolo devuelto contiene el MIC y el texto del mensaje. El texto del mensaje es texto cifrado (en el caso de un mensaje cifrado) o el texto sin formato original (en el caso de los mensajes no cifrados).

Utilizar getMIC()

El método getMIC lleva a cabo las acciones siguientes pero no puede cifrar el mensaje:
- Calcula un MIC
- Devuelve un símbolo

El símbolo devuelto solo contiene el MIC calculado y no incluye el mensaje original. Por consiguiente, además de transportar el símbolo de MIC al igual, es preciso hacer que de algún modo el igual tenga conocimiento del mensaje original para que pueda verificar su MIC.

Ejemplo: utilizar los servicios por mensaje para enviar un mensaje

El ejemplo siguiente muestra cómo un igual (foo) puede envolver un mensaje para la entrada a otro igual (superSecureServer):

```
byte[] message = "Ready to roll!".getBytes();
MessageProp mprop = new MessageProp(true); // foo quiere el mensaje cifrado
byte[] wrappedMessage =
    fooContext.wrap(message, 0, message.length, mprop);
send(wrappedMessage); // transferir el mensaje envuelto a superSecureServer

// Así puede obtener superSecureServer un MIC para la entrega a foo:
byte[] message = "You bet!".getBytes();
MessageProp mprop = null;  // superSecureServer está satisfecho con
```

```
                                // la calidad de protección predeterminada

byte[] mic =
    serverAcceptorContext.getMIC(message, 0, message.length, mprop);
send(mic);
// enviar el MIC a foo. foo también necesita el mensaje original para verificar el MIC
```

Recibir mensajes

El receptor de un mensaje envuelto utiliza unwrap() para descodificar el mensaje. El método unwrap lleva a cabo las acciones siguientes:

- Verifica el MIC criptográfico incorporado en el mensaje
- Devuelve el mensaje original a partir del cual el emisor ha calculado el MIC

Si el emisor ha cifrado el mensaje, unwrap() descifra el mensaje antes de verificar el MIC y, a continuación, devuelve el mensaje de texto sin formato original. El receptor de un símbolo MIC utiliza verifyMIC() para verificar el MIC a partir de un mensaje determinado.

Las aplicaciones de iguales utilizan su propio protocolo para entregarse símbolos de mensaje y contexto de JGSS. Las aplicaciones de iguales también deben definir un protocolo para determinar si el símbolo es un MIC o un mensaje envuelto. Por ejemplo, una parte de este protocolo puede ser tan sencilla (y rígida) como el empleado por las aplicaciones de capa de seguridad y autenticación simple (SASL). El protocolo SASL especifica que el aceptante del contexto siempre es el primer igual en enviar un símbolo por mensaje (envuelto) tras el establecimiento del contexto.

Hallará más información en Capa de seguridad y autenticación simple (SASL).

Ejemplo: utilizar los servicios por mensaje para recibir un mensaje

Los ejemplos siguientes muestran cómo un igual (superSecureServer) desenvuelve el símbolo envuelto que ha recibido de otro igual (foo):

```
MessageProp mprop = new MessageProp(false);

byte[] plaintextFromFoo =
    serverAcceptorContext.unwrap(wrappedTokenFromFoo, 0,
                                 wrappedTokenFromFoo.length, mprop);

// superSecureServer ahora puede examinar mprop para determinar las propiedades del mensaje
// (por ejemplo, si se ha cifrado el mensaje) que foo ha aplicado.

// foo verifica el MIC recibido de superSecureServer:

MessageProp mprop = new MessageProp(false);
fooContext.verifyMIC(micFromFoo, 0, micFromFoo.length, messageFromFoo, 0,
                     messageFromFoo.length, mprop);

// foo ahora puede examinar mprop para determinar las propiedades de mensaje aplicadas por

// superSecureServer. En concreto, puede verificar que el mensaje no se ha

// cifrado, ya que getMIC nunca debe cifrar un mensaje.
```

Suprimir un contexto JGSS:

Un igual suprime un contexto cuando el contexto ya no es necesario. En las operaciones de JGSS, cada igual decide de modo unilateral cuándo suprimir un contexto y no es necesario que informe de ello a su igual.

JGSS no define un símbolo de supresión de contexto. Para suprimir un contexto, el igual llama al método de eliminación (dispose) del objeto GSSContext para liberar los recursos empleados por el contexto. Es posible seguir accediendo a un objeto GSSContext eliminado, salvo que la aplicación establezca el objeto como nulo. Sin embargo, todo intento de utilizar un contexto eliminado (pero al que todavía se puede acceder) generará una excepción.

Utilizar JAAS con la aplicación JGSS:

IBM JGSS incluye un recurso de inicio de sesión de JAAS opcional que permite a la aplicación utilizar JAAS para obtener credenciales. Una vez que el recurso de inicio de sesión de JAAS guarda las credenciales de principales y claves secretas en el objeto de sujeto de un contexto de inicio de sesión de JAAS, JGSS puede recuperar las credenciales de ese sujeto.

El comportamiento por omisión de JGSS consiste en recuperar las credenciales y claves secretas del sujeto. Puede inhabilitar esta característica estableciendo que la propiedad Java javax.security.auth.useSubjectCredsOnly sea igual a false.

Nota: Aunque el proveedor de JGSS Java puro puede utilizar la interfaz de inicio de sesión, el proveedor de JGSS nativo de System i5 no puede hacerlo.

Para obtener más información sobre las características de JAAS, vea: “Obtener credenciales de Kerberos y crear claves secretas” en la página 391.

Para utilizar la función de inicio de sesión de JAAS, la aplicación debe seguir el modelo de programación de JAAS del modo siguiente:

- Crear un contexto de inicio de sesión de JAAS
- Operar dentro los límites de una construcción JAAS Subject.doAs

El fragmento de código siguiente ilustra el concepto de operar dentro de los límites de una construcción JAAS Subject.doAs:

```
static class JGSSOperations implements PrivilegedExceptionAction {
    public JGSSOperations() {}
    public Object run () throws GSSException {
        // El código de la aplicación JGSS va/se ejecuta aquí
    }
}

public static void main(String args[]) throws Exception {
    // Crear un contexto de inicio de sesión que utilizará el
    // manejador de retornos de llamada de Kerberos
    // com.ibm.security.auth.callback.Krb5CallbackHandler

    // Debe haber una configuración de JAAS para "JGSSClient"
    LoginContext loginContext =
        new LoginContext("JGSSClient", new Krb5CallabackHandler());
        loginContext.login();

    // Ejecutar toda la aplicación JGSS en modalidad privilegiada de JAAS
    Subject.doAsPrivileged(loginContext.getSubject(),
                           new JGSSOperations(), null);
}
```

Depuración de JGSS

Cuando intente identificar problemas de JGSS, utilice la posibilidad de depuración de JGSS para generar mensajes categorizados, de gran utilidad.

Puede activar una o más categorías estableciendo los valores pertinentes de la propiedad Java com.ibm.security.jgss.debug. Para activar varias categorías, utilice una coma a fin de separar los nombres de categoría.

Las categorías de depuración son las siguientes:

Categoría	Descripción
help	Listar categorías de depuración
all	Activar la depuración para todas las categorías
off	Desactivar la depuración por completo
app	Depuración de aplicaciones (valor predeterminado)
ctx	Depuración de operaciones de contexto
cred	Operaciones de credenciales (con el nombre)
marsh	Armado de símbolos
mic	Operaciones de MIC
prov	Operaciones de proveedor
qop	Operaciones de QOP
unmarsh	Desarmado de símbolos
unwrap	Operaciones de desenvoltura
wrap	Operaciones de envoltura

Clase de depuración de JGSS

Para depurar programáticamente la aplicación JGSS, utilice la clase de depuración de la infraestructura IBM JGSS. La aplicación puede emplear la clase de depuración para activar y desactivar las categorías de depuración y visualizar información de depuración para las categorías activas.

El constructor de depuración predeterminado lee la propiedad Java com.ibm.security.jgss.debug para determinar qué categorías hay que activar.

Ejemplo: depuración para la categoría de aplicaciones

El ejemplo siguiente muestra cómo solicitar información de depuración para la categoría de aplicaciones:

```
import com.ibm.security.jgss.debug;

Debug debug = new Debug(); // Obtiene categorías de la propiedad Java

// Se necesita mucho trabajo para configurar someBuffer. Probar que la
// categoría está activa antes de configurar para la depuración.

if (debug.on(Debug.OPTS_CAT_APPLICATION)) {
    // Llenar someBuffer con datos.
 debug.out(Debug.OPTS_CAT_APPLICATION, someBuffer);
    // someBuffer puede ser una matriz de bytes o una serie.
```

Ejemplos: servicio de seguridad genérico Java (JGSS) de IBM

Los archivos de ejemplo del servicio de seguridad genérico Java (JGSS) de IBM incluyen programas de cliente y servidor, archivos de configuración, archivos de política e información de consulta Javadoc. Utilice los programas de ejemplo para probar y verificar la configuración de JGSS.

Puede ver versiones HTML de los ejemplos o descargar la información Javadoc y el código fuente de los programas de ejemplo. El hecho de descargar los ejemplos le permite ver la información de consulta Javadoc, examinar el código, editar los archivos de configuración y de política y compilar y ejecutar los programas de ejemplo.

Descripción de los programas de ejemplo

Los ejemplos de JGSS incluyen cuatro programas:
- Servidor no JAAS
- Cliente no JAAS
- Servidor habilitado para JAAS
- Cliente habilitado para JAAS

Las versiones habilitadas para JAAS son plenamente interoperativas con sus versiones no JAAS correspondientes. Por consiguiente, puede ejecutar un cliente habilitado para JAAS con un servidor no JAAS, así como ejecutar un cliente no JAAS con un servidor habilitado para JAAS.

Nota: Cuando ejecuta un ejemplo, puede especificar una o varias propiedades Java opcionales, como los nombres de los archivos de configuración y de política, las opciones de depuración de JGSS y el gestor de seguridad. También puede activar y desactivar las características de JAAS.

Puede ejecutar los ejemplos en una configuración de un servidor o de dos servidores. La configuración de un servidor consta de un cliente que se comunica con un servidor primario. La configuración de dos servidores consta de un servidor primario y otro secundario, donde el servidor primario actúa como iniciador, o cliente, para el servidor secundario.

Al utilizar la configuración de dos servidores, el cliente primero inicia un contexto e intercambia mensajes seguros con el servidor primario. A continuación, el cliente delega sus credenciales al servidor primario. Posteriormente, en nombre del cliente, el servidor primario utiliza estas credenciales para iniciar un contexto e intercambiar mensajes seguros con el servidor secundario. También puede emplear una configuración de dos servidores en la que el servidor primario actúa como cliente en nombre propio. En este caso, el servidor primario utiliza sus propias credenciales para iniciar un contexto e intercambiar mensajes con el servidor secundario.

Puede ejecutar cualquier número de clientes con el servidor primario de forma simultánea. Aunque puede ejecutar un cliente directamente con el servidor secundario, el servidor secundario no puede emplear las credenciales delegadas ni ejecutarse como iniciador con sus propias credenciales.

Ver los ejemplos de IBM JGSS:

Los archivos de ejemplo del servicio de seguridad genérico Java (JGSS) de IBM incluyen programas de cliente y servidor, archivos de configuración, archivos de política e información de consulta Javadoc. Emplee los enlaces siguientes para ver las versiones HTML de los ejemplos de JGSS.

Conceptos relacionados

"Ejemplos: servicio de seguridad genérico Java (JGSS) de IBM" en la página 404
Los archivos de ejemplo del servicio de seguridad genérico Java (JGSS) de IBM incluyen programas de cliente y servidor, archivos de configuración, archivos de política e información de consulta Javadoc. Utilice los programas de ejemplo para probar y verificar la configuración de JGSS.

Tareas relacionadas

"Ejemplos: descargar y ejecutar los programas JGSS de ejemplo" en la página 410
En este tema hallará instrucciones para descargar y ejecutar la información de Javadoc de los ejemplos.

Referencia relacionada

"Ejemplo: programa cliente IBM JGSS no JAAS" en la página 526
Utilice este cliente de ejemplo JGSS junto con el servidor de ejemplo JGSS.

"Ejemplo: programa servidor IBM JGSS no JAAS" en la página 534
Este es un ejemplo de un servidor JGSS que se debe usar juntamente con un cliente JGSS de ejemplo.

"Ejemplo: programa cliente IBM JGSS habilitado para JAAS" en la página 546
Este programa de ejemplo realiza un inicio de sesión JAAS y funciona dentro del contexto de inicio de sesión JAAS. No establece la variable javax.security.auth.useSubjectCredsOnly, dejándola como valor predeterminado, que es "true", para que el servicio GSS Java adquiera credenciales del sujeto JAAS asociado al contexto de inicio de sesión creado por el cliente.

"Ejemplo: programa servidor IBM JGSS habilitado para JAAS" en la página 548
Este programa de ejemplo realiza un inicio de sesión JAAS y funciona dentro del contexto de inicio de sesión JAAS.

Ejemplo: archivo de configuración de Kerberos:

Este tema contiene el archivo de configuración de Kerberos para ejecutar las aplicaciones de ejemplo de JGSS.

Para obtener más información sobre cómo utilizar el archivo de configuración de ejemplo, vea: Descargar y ejecutar los ejemplos de IBM JGSS.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
# --------------------------------------------------------------------------------
# Archivo de configuración de Kerberos para ejecutar las aplicaciones de ejemplo de JGSS.
# Modifique las entradas de modo que se ajusten a su entorno.
#--------------------------------------------------------------------------------

[libdefaults]
 default_keytab_name   = /QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab
 default_realm         = REALM.IBM.COM
 default_tkt_enctypes  = des-cbc-crc
 default_tgs_enctypes  = des-cbc-crc
 default_checksum      = rsa-md5
 kdc_timesync          = 0
 kdc_default_options   = 0x40000010
 clockskew             = 300
 check_delegate        = 1
 ccache_type           = 3
 kdc_timeout           = 60000


[realms]
 REALM.IBM.COM = {
         kdc = kdc.ibm.com:88
 }

[domain_realm]
 .ibm.com = REALM.IBM.COM
```

Ejemplo: archivo de configuración de inicio de sesión de JAAS:

Este tema contiene la configuración de inicio de sesión de JAAS para los ejemplos de JGSS.

Para obtener más información sobre cómo utilizar el archivo de configuración de ejemplo, vea: Descargar y ejecutar los ejemplos de IBM JGSS.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/**
 * --------------------------------------------------------------------------------
 * Configuración de inicio de sesión de JAAS para los ejemplos de JGSS.
 * --------------------------------------------------------------------------------
 *
  * Declaración de limitación de responsabilidad sobre el código de ejemplo
```

```
  * IBM otorga al usuario una licencia de copyright no exclusiva para utilizar
  * todos los ejemplos de código de programación, a partir de los que puede generar
  * funciones similares adaptadas a sus necesidades específicas.
  * IBM proporciona la totalidad del código de ejemplo solo con propósito ilustrativo.
  * Estos ejemplos no se han probado exhaustivamente en todas las condiciones.
  * Por consiguiente, IBM no puede garantizar la fiabilidad, la capacidad de servicio o
  * el funcionamiento de estos programas.
  * Todos los programas que contiene este ejemplo se suministran "TAL CUAL",
  * sin garantías de ninguna clase.
  * Se renuncia explícitamente a las garantías implícitas de no infringibilidad, comercialización
  * y adecuación a un propósito determinado.
 *
 *
  * Opciones soportadas:
  *     principal=<serie>
  *     credsType=initiator|acceptor|both (valor predeterminado=initiator)
  *     forwardable=true|false (valor predeterminado=false)
  *     proxiable=true|false (valor predeterminado=false)
  *     useCcache=<serie_URL>
  *     useKeytab=<serie_URL>
  *     useDefaultCcache=true|false (valor predeterminado=false)
  *     useDefaultKeytab=true|false (valor predeterminado=false)
  *     noAddress=true|false (valor predeterminado=false)
 *
  * El reino por omisión (que se obtiene del archivo de configuración de Kerberos)
  * se emplea si el principal especificado no incluye un componente de reino.
 */

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;
};
```

Ejemplo: archivo de política de JAAS:

Este tema contiene el archivo de política de JAAS para ejecutar las aplicaciones de ejemplo de JGSS.

Para obtener más información sobre cómo utilizar el archivo de política de ejemplo, vea: Descargar y ejecutar los ejemplos de IBM JGSS.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// --------------------------------------------------------------------------
// Archivo de política de JAAS para ejecutar las aplicaciones de ejemplo de JGSS.
// Modifique estos permisos de modo que se ajusten a su entorno.
// No se recomienda su uso para un fin distinto del indicado anteriormente.
// En concreto, no utilice este archivo de política ni su
// contenido para proteger recursos en un entorno de producción.
//
// Declaración de limitación de responsabilidad sobre el código de ejemplo
// IBM otorga al usuario una licencia de copyright no exclusiva para utilizar
// todos los ejemplos de código de programación, a partir de los que puede generar
// funciones similares adaptadas a sus necesidades específicas.
// IBM proporciona la totalidad del código de ejemplo solo con propósito ilustrativo.
// Estos ejemplos no se han probado exhaustivamente en todas las condiciones.
// Por consiguiente, IBM no puede garantizar la fiabilidad, la capacidad de servicio o
// el funcionamiento de estos programas.
// Todos los programas que contiene este ejemplo se suministran "TAL CUAL",
// sin garantías de ningún tipo.
```

```
// Se renuncia explícitamente a las garantías implícitas de no infringibilidad, comercialización
// y adecuación a un propósito determinado.
//
// ----------------------------------------------------------------------------

//----------------------------------------------------------------------------
// Permisos para el cliente únicamente
//----------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "bob@REALM.IBM.COM"
{
    // foo necesita poder iniciar un contexto con el servidor
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "initiate";

    // Para que foo pueda delegar sus credenciales al servidor
    permission javax.security.auth.kerberos.DelegationPermission
        "\"gss_service/myhost.ibm.com@REALM.IBM.COM\" \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
};

//----------------------------------------------------------------------------
// Permisos para el servidor únicamente
//----------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service/myhost.ibm.com@REALM.IBM.COM"
{
    // Permiso para que el servidor acepte conexiones de red en su host
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // Permiso para que el servidor acepte contextos de JGSS
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "accept";

    // El servidor actúa como cliente en la comunicación con el servidor secundario (de reserva)
    // Este permiso permite al servidor iniciar un contexto con el servidor secundario
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "initiate";
};

//----------------------------------------------------------------------------
// Permisos para el servidor secundario
//----------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service2/myhost.ibm.com@REALM.IBM.COM"
{
    // Permiso para que el servidor secundario acepte conexiones de red en su sistema principal
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // Permiso para que el servidor acepte contextos de JGSS
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "accept";
};
```

Ejemplo: archivo de política Java:

Este tema contiene el archivo de política Java que ejecuta las aplicaciones JGSS de ejemplo en el servidor.

Para obtener más información sobre cómo utilizar el archivo de política de ejemplo, vea: Descargar y ejecutar los ejemplos de IBM JGSS.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// --------------------------------------------------------------
// Archivo de política Java para ejecutar las aplicaciones de ejemplo de JGSS
// en el servidor.
// Modifique estos permisos de modo que se ajusten a su entorno.
// No se recomienda su uso para un fin distinto del indicado anteriormente.
// En concreto, no utilice este archivo de política ni su
// contenido para proteger recursos en un entorno de producción.
//
// Declaración de limitación de responsabilidad sobre el código de ejemplo
// IBM otorga al usuario una licencia de copyright no exclusiva para utilizar
// todos los ejemplos de código de programación, a partir de los que puede generar
// funciones similares adaptadas a sus necesidades específicas.
// IBM proporciona la totalidad del código de ejemplo solo con propósito ilustrativo.
// Estos ejemplos no se han probado exhaustivamente en todas las condiciones.
// Por consiguiente, IBM no puede garantizar la fiabilidad, la capacidad de servicio o
// el funcionamiento de estos programas.
// Todos los programas que contiene este ejemplo se suministran "TAL CUAL",
// sin garantías de ningún tipo.
// Se renuncia explícitamente a las garantías implícitas de no infringibilidad, comercialización
// y adecuación a un propósito determinado.
//
//---------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar" {

    // For Java 1.4
    permission javax.security.auth.AuthPermission "createLoginContext.JAASClient";
    permission javax.security.auth.AuthPermission "createLoginContext.JAASServer";

    permission javax.security.auth.AuthPermission "doAsPrivileged";

    // Permiso para solicitar un ticket del KDC
    permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/REALM.IBM.COM@REALM.IBM.COM", "initiate";

    // Permiso para acceder a las clases de sun.security.action
    permission java.lang.RuntimePermission "accessClassInPackage.sun.security.action";

    // Se accede a todo un paquete de propiedades 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";

    // Permiso para comunicarse con el host del KDC de Kerberos
    permission java.net.SocketPermission "kdc.ibm.com", "connect,accept,resolve";

    // Ejecuto los ejemplos desde mi host local
    permission java.net.SocketPermission "myhost.ibm.com", "accept,connect,resolve";
    permission java.net.SocketPermission "localhost", "listen,accept,connect,resolve";

    // Acceder a algunas ubicaciones de configuración de Kerberos posibles
    // Modificar las vías de acceso de archivo según corresponda al entorno
    permission java.io.FilePermission "${user.home}/krb5.ini", "read";
    permission java.io.FilePermission "${java.home}/lib/security/krb5.conf", "read";

    // Acceder a la tabla de claves de Kerberos para poder obtener la clave del servidor.
```

```
    permission java.io.FilePermission
    "/QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab", "read";

    // Acceder a la caché de credenciales de Kerberos del usuario.
    permission java.io.FilePermission "${user.home}/krb5cc_${user.name}",
    "read";
};
```

Ejemplos: descargar y ver la información Javadoc de los ejemplos IBM JGSS:

Para descargar y ver la documentación de los programas de ejemplo IBM, siga estos pasos.

1. Elija un directorio existente (o cree uno nuevo) donde desee almacenar la información de Javadoc.
2. Descargue la información de Javadoc (jgsssampledoc.zip) y colóquela en el directorio.
3. Extraiga los archivos de jgsssampledoc.zip y póngalos en el directorio.
4. Utilice el navegador para acceder al archivo index.htm.

Ejemplos: descargar y ejecutar los programas JGSS de ejemplo:

En este tema hallará instrucciones para descargar y ejecutar la información de Javadoc de los ejemplos.

Antes de modificar o ejecutar los ejemplos, lea la descripción de los programas de ejemplo en los Ejemplos: artículo sobre los servicios de seguridad genéricos Java JGSS de IBM.

Para ejecutar los programas de ejemplo, lleve a cabo las siguientes tareas:

1. Descargue los archivos de ejemplo y colóquelos en el servidor
2. Prepárese para ejecutar los archivos de ejemplo
3. Ejecute los programas de ejemplo

Conceptos relacionados

"Ejemplos: servicio de seguridad genérico Java (JGSS) de IBM" en la página 404
Los archivos de ejemplo del servicio de seguridad genérico Java (JGSS) de IBM incluyen programas de cliente y servidor, archivos de configuración, archivos de política e información de consulta Javadoc. Utilice los programas de ejemplo para probar y verificar la configuración de JGSS.

Tareas relacionadas

"Ejemplos: descargar y ejecutar los programas JGSS de ejemplo"
En este tema hallará instrucciones para descargar y ejecutar la información de Javadoc de los ejemplos.

Ejemplos: descargar los ejemplos de IBM JGSS:

En este tema hallará instrucciones para descargar la información de Javadoc JGSS de ejemplo y colocarla en su sistema.

Antes de modificar o ejecutar los ejemplos, lea la descripción de los programas de ejemplo.

Para descargar los archivos de ejemplo y almacenarlos en el servidor, siga estos pasos:

1. En el servidor, elija un directorio existente (o cree uno nuevo) donde desee almacenar los programas de ejemplo, los archivos de configuración y los archivos de política.
2. Descargue los programas de ejemplo (ibmjgsssample.zip).
3. Extraiga los archivos de ibmjgsssample.zip y colóquelos en el directorio del servidor.

Al extraer el contenido de ibmjgsssample.jar se llevan a cabo las acciones siguientes:

- Se coloca ibmgjsssample.jar, que contiene los archivos .class de ejemplo, en el directorio seleccionado.
- Se crea un subdirectorio (denominado config) que contiene los archivos de configuración y política.

- Se crea un subdirectorio (denominado src) que contiene los archivos fuente .java de ejemplo.

Información relacionada

Si lo desea, puede leer información acerca de las tareas relacionadas o examinar un ejemplo:
- "Ejemplos: prepararse para ejecutar los programas de ejemplo JGSS"
- "Ejemplos: ejecutar los programas de ejemplo JGSS"
- "Ejemplo: ejecutar el ejemplo no JAAS" en la página 412

Ejemplos: prepararse para ejecutar los programas de ejemplo JGSS:

Después de descargar el código fuente, debe llevar a cabo preparativos antes de ejecutar los programas de ejemplo.

Antes de modificar o ejecutar los ejemplos, vea: "Ejemplos: servicio de seguridad genérico Java (JGSS) de IBM" en la página 404.

Tras bajar el código fuente, debe llevar a cabo las tareas siguientes para poder ejecutar los programas de ejemplo:
- Edite los archivos de configuración y política de modo que se adecuen al entorno. Para obtener más información, consulte los comentarios de cada uno de los archivos de configuración y política.
- Asegúrese de que el archivo java.security file contiene los valores correctos para System i5. Para obtener más información, consulte "Archivos de configuración y política" en la página 394.
- Coloque el archivo de configuración de Kerberos (krb5.conf) modificado en el directorio del servidor adecuado para la versión de J2SDK que esté utilizando:
  - Para la versión 1.4 de J2SDK: /QIBM/ProdData/Java400/jdk14/lib/security
  - Para la versión 1.5 de J2SE: /QIBM/ProdData/Java400/jdk15/lib/security

  Tareas relacionadas

  "Ejemplos: descargar los ejemplos de IBM JGSS" en la página 410
  En este tema hallará instrucciones para descargar la información de Javadoc JGSS de ejemplo y colocarla en su sistema.

  "Ejemplos: ejecutar los programas de ejemplo JGSS"
  Tras bajar y modificar el código fuente, puede ejecutar uno de los ejemplos.

  Referencia relacionada

  "Ejemplo: ejecutar el ejemplo no JAAS" en la página 412
  Para ejecutar un ejemplo, debe descargar y modificar el código fuente de ejemplo.

Ejemplos: ejecutar los programas de ejemplo JGSS:

Tras bajar y modificar el código fuente, puede ejecutar uno de los ejemplos.

Antes de modificar o ejecutar los ejemplos, lea la descripción de los programas de ejemplo.

Para ejecutar un ejemplo, primero debe iniciar el programa servidor. El programa servidor debe estar en ejecución y preparado para recibir conexiones antes de iniciar el programa cliente. El servidor estará preparado para recibir conexiones cuando vea el mensaje `listening on port <puerto_servidor>`. Asegúrese de recordar o anotar el <puerto_servidor>, que es el número de puerto que deberá especificar cuando inicie el cliente.

Utilice el mandato siguiente para iniciar un programa de ejemplo:

```
java [-Dpropiedad1=valor1 ... -DpropiedadN=valorN] com.ibm.security.jgss.test.<programa> [opciones]
```

donde

- [-DpropertyN=valueN] es una o más propiedades Java opcionales, como las de los nombres de los archivos de configuración y política, las opciones de depuración JGSS y el gestor de seguridad. Para obtener más información, consulte el ejemplo siguiente y Ejecutar aplicaciones JGSS.
- <programa> es un parámetro obligatorio que especifica el programa de ejemplo que desea ejecutar (`Client`, `Server`, `JAASClient` o `JAASServer`).
- [opciones] es un parámetro opcional para el programa de ejemplo que desea ejecutar. Para ver una lista de las opciones permitidas, utilice el mandato siguiente:

```
java com.ibm.security.jgss.test.<programa> -?
```

Nota: Desactive las características JAAS en un ejemplo habilitado por JGSS, estableciendo que la propiedad Java javax.security.auth.useSubjectCredsOnly sea igual a false. Naturalmente, el valor predeterminado de los ejemplos habilitados para JAAS establece la activación de JAAS, con lo que el valor de la propiedad es true. Los programas cliente y servidor no JAAS establecen la propiedad en false, salvo que se haya establecido explícitamente el valor de la propiedad.

Información relacionada

Si lo desea, puede leer información acerca de las tareas relacionadas o examinar un ejemplo:
- “Ejemplos: prepararse para ejecutar los programas de ejemplo JGSS” en la página 411
- “Ejemplos: descargar los ejemplos de IBM JGSS” en la página 410
- “Ejemplo: ejecutar el ejemplo no JAAS”

Ejemplo: ejecutar el ejemplo no JAAS:

Para ejecutar un ejemplo, debe descargar y modificar el código fuente de ejemplo.

Para obtener más información, consulte Descargar y ejecutar los programas de ejemplo.

Iniciar el servidor primario

Utilice el mandato siguiente para iniciar un servidor no JAAS que está a la escucha en el puerto 4444. El servidor se ejecuta como principal (superSecureServer) y utiliza un servidor secundario (backupServer). El servidor también visualiza información de depuración de credenciales y aplicaciones.

```
java -classpath ibmjgsssample.jar
     -Dcom.ibm.security.jgss.debug="app, cred"
     com.ibm.security.jgss.test.Server -p 4444
     -n superSecureServer -s backupServer
```

Al ejecutarse correctamente este ejemplo se muestra el mensaje siguiente:

```
listening on port 4444
```

Iniciar el servidor secundario

Utilice el mandato siguiente para iniciar un servidor secundario no JAAS que está a la escucha en el puerto 3333 y se ejecuta como el principal backupServer:

```
java -classpath ibmjgsssample.jar
     com.ibm.security.jgss.test.Server -p 3333
     -n backupServer
```

Iniciar el cliente

Utilice el mandato siguiente (escrito en una sola línea) para ejecutar el cliente habilitado para JAAS (myClient). El cliente se comunica con el servidor primario en el sistema principal (securityCentral). El cliente se ejecuta teniendo habilitado el gestor de seguridad, y utiliza los archivos de configuración y

política de JAAS y el archivo de política Java del directorio config. Para obtener más información sobre el directorio config, vea: Descargar los ejemplos de IBM JGSS.

```
java -classpath ibmjgsssample.jar
     -Djava.security.manager
     -Djava.security.auth.login.config=config/jaas.conf
     -Djava.security.policy=config/java.policy
     -Djava.security.auth.policy=config/jaas.policy
     com.ibm.security.jgss.test.JAASClient -n myClient
     -s superSecureServer -h securityCentral:4444
```

Información de consulta Javadoc de IBM JGSS

La información de consulta Javadoc de IBM JGSS incluye clases y métodos del paquete de API org.ietf.jgss y las versiones Java de algunas herramientas de gestión de credenciales Kerberos.

Aunque JGSS contiene varios paquetes de acceso público (por ejemplo, com.ibm.security.jgss y com.ibm.security.jgss.spi), se recomienda utilizar únicamente las API del paquete org.ietf.jgss estándar. El uso exclusivo de este paquete garantiza el cumplimiento de las especificaciones GSS-API por parte de la aplicación, así como una interoperatividad y una portabilidad óptimas.

- org.ietf.jgss
- "Clase Kinit de com.ibm.security.krb5.internal.tools" en la página 381
- "Clase Ktab de com.ibm.security.krb5.internal.tools" en la página 383
- "Clase Klist de com.ibm.security.krb5.internal.tools" en la página 380

Ajustar el rendimiento de los programas Java con IBM Developer Kit para Java

Conviene que tenga presentes varios aspectos del rendimiento de las aplicaciones Java cuando se disponga a construir una aplicación Java.

Para conseguir optimizar el rendimiento, puede lleva a cabo estas acciones:

- Mejore el rendimiento del código Java utilizando el compilador Just-In-Time (JIT) o utilizando la caché para los cargadores de clases de usuario.
- Establezca cuidadosamente los valores para obtener un rendimiento de recogida de basura óptimo.
- Solo debe utilizar los métodos nativos para iniciar funciones del sistema que sean de relativamente larga ejecución y que no estén disponibles directamente en Java.
- Utilice excepciones Java en los casos en que no se produzca el flujo normal por la aplicación.

Para localizar los problemas de rendimiento en los programas Java, utilice las herramientas indicadas a continuación junto con el explorador de rendimiento (PEX):

- Puede recoger información de rendimiento de rastreo de eventos Java utilizando la máquina virtual Java de i5/OS.
- Para determinar el tiempo que se invierte en cada uno de los métodos Java, utilice los rastreo de llamadas Java.
- Localice el tiempo de CPU relativo que se invierte en cada método Java y en todas las funciones del sistema que el programa Java utiliza con el perfilado Java.
- Utilice el colector de datos de rendimiento Java (JPDC) para proporcionar información de perfilado sobre los programas que se ejecutan en el servidor.

Cualquier sesión de trabajo puede iniciar y finalizar PEX. Normalmente, los datos se recogen a escala de todo el sistema y están relacionados con todos los trabajos del sistema, incluidos los programas Java. A veces, puede ser necesario iniciar y detener la recogida de rendimiento desde el interior de una aplicación Java. Con ello se reduce el tiempo de recogida y puede reducirse el gran volumen de datos producidos generalmente por un rastreo de retorno o de llamada. PEX no se puede ejecutar desde dentro de una

hebra Java. Para iniciar y detener una recogida, es necesario escribir un método nativo que se comunique con un trabajo independiente a través de una cola o memoria compartida. Luego, el segundo trabajo inicia y detiene la recogida en el momento oportuno.

Además de los datos de rendimiento a nivel de aplicación, puede utilizar las herramientas existentes de rendimiento a nivel del sistema System i. Estas herramientas proporcionan un informe de estadísticas por cada hebra Java.

Conceptos relacionados

"Consideraciones sobre el rendimiento de la compilación estática Java" en la página 418
A partir de la V6R1, la velocidad de transformación no quedará afectada por el nivel de optimización que se establezca.

"Herramientas de rendimiento del perfilado Java" en la página 420
El perfilado de unidad central de proceso (CPU) a escala de todo el sistema calcula el tiempo relativo de CPU que se invierte en cada uno de los métodos Java y en todas las funciones del sistema que el programa Java utiliza.

Información relacionada


Performance Tools for iSeries, SC41-5340


Este manual contiene ejemplos de informes de PEX.

Herramientas de rendimiento de rastreo de eventos Java

La máquina virtual Java de i5/OS permite rastrear determinados eventos Java.

Estos eventos pueden recogerse sin ninguna instrumentación en el código Java. Abarcan actividades tales como la recogida de basura, la creación de hebras, la carga de clases y los bloqueos. El mandato Ejecutar Java (RUNJVA) no especifica estos eventos. En cambio, se puede crear una definición del explorador de rendimiento (PEX) y utilizar el mandato Arrancar explorador de rendimiento (STRPEX) para recoger los eventos. Cada evento contiene información útil de rendimiento como, por ejemplo, la indicación de la hora y los ciclos de unidad central de proceso (CPU). Se pueden rastrear eventos Java y otras actividades del sistema, tales como la entrada y la salida de disco, con una misma definición de rastreo.

Para obtener una descripción completa de los eventos Java, consulte la publicación Performance Tools for iSeries, SC41-5340.

Consideraciones sobre el rendimiento de Java

La total comprensión de las siguientes consideraciones puede ayudarle a mejorar el rendimiento de sus aplicaciones Java.

Crear programas Java optimizados

El parámetro OPTIMIZE solo se emplea al crear programas Java para un release destino anterior a V6R1. En el caso de un release V6R1 o posterior, el valor se ignora, y se emplea OPTIMIZE(*INTERPRET). Esto quiere decir que el programa Java se interpreta a partir del bytecode o se ejecuta con el compilador Just-in-Time (JIT) cuando se inicia. Las variables se pueden visualizar y modificar durante la depuración.

Utilizar el compilador Just-In-Time

Utilizar el compilador Just-In-Time (JIT) con el Intérprete de modalidad mixta (MMI) da como resultado un rendimiento de arranque casi similar al del código compilado. MMI interpreta el código Java hasta alcanzar el umbral especificado por la propiedad Java os400.jit.mmi.threshold del sistema. Una vez alcanzado el umbral, i5/OS emplea el tiempo y los recursos necesarios para utilizar el compilador JIT en

compilar un método sobre los métodos utilizados con mayor frecuencia. Utilizar el compilador JIT da como resultado un código altamente optimizado que mejora el rendimiento de la ejecución en comparación con el código precompilado.

Hallará más información en el mandato de lenguaje de control Visualizar programa Java (DSPJVAPGM).

Utilizar cachés para cargadores de clases de usuario

Cuando se utiliza la caché de la máquina virtual Java (JVM) de i5/OS para los cargadores de clases de usuario, el rendimiento de arranque mejora para las clases que se carguen desde un cargador de clases de usuario. En la caché se almacenan los objetos programa Java, y así la JVM los puede reutilizar. Al reutilizar los programas Java almacenados, mejora el rendimiento, ya que no hace falta volver a crear los objetos programa Java puestos en la caché ni verificar el bytecode.

Utilice las siguientes propiedades para controlar las cachés para los cargadores de clases de usuario:

os400.define.class.cache.file
El valor de esta propiedad especifica el nombre (con la vía de acceso completa) de un archivo Java de ARchivado (JAR) válido. Como mínimo, el archivo JAR especificado debe contener un directorio JAR válido (tal como se ha construido mediante el mandato QSH `jar`) y el miembro individual necesario para el mandato `jar` para poder funcionar. No incluya el archivos JAR especificado en ninguna CLASSPATH Java. El valor predeterminado de esta propiedad es /QIBM/ProdData/Java400/QDefineClassCache.jar. Para inhabilitar la puesta en caché, especifique esta propiedad sin ningún valor.

os400.define.class.cache.hours
El valor de esta propiedad especifica cuánto tiempo (en horas) desea que un objeto programa Java permanezca en la caché. Cuando la JVM no utiliza un objeto programa Java en caché durante el tiempo especificado, el i5/OS elimina el objeto programa Java de la caché. El valor predeterminado de esta propiedad es de 768 horas (33 días). El valor máximo es de 9999 (unas 59 semanas). Cuando especifique un valor de 0 o un valor que no reconozca como un número decimal válido, utilizará el valor predeterminado.

os400.define.class.cache.maxpgms
El valor de esta propiedad especifica el número máximo de objetos programa Java que pueden caber en la caché. Cuando se sobrepasa este límite de la caché, i5/OS elimina de ella el objeto programa Java más antiguo. i5/OS determina qué programa de la caché es el más antiguo comparando las horas en que la JVM hizo la última referencia a los objetos programa Java. El valor predeterminado es 5000 y el valor máximo es 40000. Cuando especifique un valor de 0 o un valor que no reconozca como un número decimal válido, utilizará el valor predeterminado.

Utilice DSPJVAPGM en el archivo JAR, que se especifica en la propiedad 400.define.class.cache.file, para determinar el número de objetos programa Java situados en la caché.

- El campo **Programas Java** de la pantalla DSPJVAPGM indica el número de objetos programa Java situados en la caché.
- El campo **Tamaño de programa Java** indica la cantidad de almacenamiento empleado por los objetos programa Java de la caché.
- Otros campos de la pantalla DSPJVAPGM no tienen sentido cuando se emplea el mandato en un archivo JAR que esté utilizando para la puesta en caché.

Rendimiento de la caché

Al ejecutar algunas aplicaciones Java, se puede poner en caché un gran número de objetos programa Java. Utilice DSPJVAPGM para determinar si el número de programas Java de la caché se acerca al valor máximo antes de que la aplicación termine de ejecutarse. El rendimiento de la aplicación podría deteriorarse cuando se llena la caché, ya que i5/OS podría eliminar de la caché algunos programas que la aplicación puede necesitar.

Puede evitar la degradación del rendimiento que se produce cuando se llena la caché. Por ejemplo, puede configurar aplicaciones para utilizar cachés separadas para las aplicaciones que se ejecutan con frecuencia pero cargar programas distintos en la caché. Si se utilizan cachés separadas, se puede impedir que la caché se llene y así evitar que i5/OS elimine programas Java de la caché. Otra solución es aumentar el número que especifique para la propiedad os400.define.class.cache.maxpgms.

Puede utilizar el mandato de lenguaje de control Cambiar programa Java (CHGJVAPGM) en el archivo JAR para cambiar la optimización de las clases en la caché. CHGJVAPGM afecta solamente a los programas que están actualmente en la caché. Tras realizar cambios en los niveles de optimización, la propiedad os400.defineClass.optLevel especifica cómo optimizar las clases que se añaden a la caché.

Por ejemplo, para utilizar el JAR de conjunto suministrado, con un máximo de 20000 objetos programa Java, donde cada programa Java tiene una vida máxima de 1 año, establezca los siguientes valores para las propiedades de la caché:

```
os400.define.class.cache.file    /QIBM/ProdData/Java400/QDefineClassCache.jar
     os400.define.class.cache.hours   8760
os400.define.class.cache.maxpgms 20000
```

Seleccionar qué modalidad hay que utilizar al ejecutar un programa Java

Cuando ejecuta un programa Java, puede seleccionar qué modalidad se gustaría utilizar. Todas las modalidades verifican el código y crean un objeto programa Java en el que conservar el formato de programa anterior a la verificación.

Las modalidades que se pueden usar son:

- Interpretado
- Compilación Just-In-Time (JIT)

Modalidad de selección	Detalles
Interpretado	Cada bytecode se interpreta en el momento de la ejecución. Si desea información sobre cómo ejecutar el programa Java en la modalidad interpretada, vea: Mandato Ejecutar Java (RUNJVA).

Modalidad de selección	Detalles

Compilación Just-In-Time (JIT)	El i5/OS interpreta los métodos Java hasta alcanzar el umbral especificado por la propiedad Java os400.jit.mmi.threshold del sistema. Una vez alcanzado el umbral, el i5/OS utiliza el compilador JIT para compilar métodos como instrucciones de máquina nativa.
	
	Para utilizar el compilador Just-In-Time, tiene que establecer el valor del compilador en `jitc`. Puede establecer el valor añadiendo una variable de entorno o estableciendo la propiedad java.compiler del sistema. Seleccione un método de la lista siguiente para establecer el valor del compilador:
	• Desde una línea de mandatos de i5/OS, añada la variable de entorno con el mandato Añadir variable de entorno (ADDENVVAR). Luego ejecute el programa Java con el mandato Ejecutar Java (RUNJVA) o con el mandato JAVA. Por ejemplo, utilice:
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc)`
	`JAVA CLASS(Test)`
	• Establezca la propiedad java.compiler del sistema en la línea de mandatos de i5/OS. Por ejemplo, entre `JAVA CLASS(Test) PROP((java.compiler jitc))`
	• Establezca la propiedad java.compiler del sistema en la línea de mandatos del intérprete Qshell. Por ejemplo, entre `java -Djava.compiler=jitc Test`

El programa Java se puede ejecutar de tres maneras (CL, QSH y JNI). Cada una de ellas tiene una forma exclusiva de especificar la modalidad. En esta tabla podrá ver lo que se hace:

Modalidad	Mandato CL	Mandato de QShell	API de invocación de JNI
Intérprete	INTERPRET(*YES)	-Djava.compiler=NONE -interpret	os400.run.mode=interpret
JIT	INTERPRET(*JIT)	-Djava.compiler=jitc	os400.run.mode=jitc

Intérprete de Java

El intérprete de Java es el componente de la máquina virtual Java que interpreta los archivos de clase Java para una plataforma de hardware determinada. El intérprete de Java decodifica cada bytecode y ejecuta una serie de instrucciones de máquina para ese bytecode.

Máquina virtual Java (JVM)

Compilación Java estática

En V6R1, el proceso directo ha dejado de estar soportado. Esto quiere decir que el nivel de optimización de los programas Java se ignora y que se emplea OPTIMIZE(*INTERPRET) cuando se crea un programa Java en V6R1 o releases posteriores.

Al igual que en los releases anteriores, el programa Java contiene una versión preverificada de uno o más archivos de clase Java. Sin embargo, en V6R1, el programa Java no contiene instrucciones de máquina. En tiempo de ejecución, el programa Java se interpreta a partir del bytecode o bien se ejecuta con el compilador Just-In-Time (JIT).

En el caso de los releases destinos anteriores a V6R1, el parámetro OPTIMIZE especifica el nivel de optimización del programa Java. Al crear un programa Java para un release destino anterior a V6R1, el

valor del parámetro OPTIMIZE se encapsula dentro del programa Java, pero no se generan instrucciones de máquina. El valor encapsulado del parámetro OPTIMIZE se emplea durante la conversión del programa Java en el release destino del sistema operativo.

Consideraciones sobre el rendimiento de la compilación estática Java:

A partir de la V6R1, la velocidad de transformación no quedará afectada por el nivel de optimización que se establezca.

El parámetro OPTIMIZE solo se emplea al crear programas Java para un release destino anterior a V6R1. En el caso de un release V6R1 o posterior, el valor se ignora, y se emplea OPTIMIZE(*INTERPRET). El programa Java se interpreta a partir del bytecode o se ejecuta con el compilador Just-in-Time (JIT) cuando se inicia. Si los bytecodes se interpretan, las variables se pueden visualizar y modificar durante la depuración. Si el método se ha compilado mediante el compilador JIT, no hay información de depuración disponible para el depurador System i5.

En el caso de los releases destinos anteriores a V6R1, el parámetro OPTIMIZE especifica el nivel de optimización del programa Java. Al crear un programa Java para un release destino anterior a V6R1, el valor del parámetro OPTIMIZE se encapsula dentro del programa Java, pero no se generan instrucciones de máquina. El valor encapsulado del parámetro OPTIMIZE se emplea durante la conversión del programa Java en el release destino del sistema operativo.

Compilador Just-In-Time

Un compilador Just-In-Time (JIT) es un compilador específico de plataforma que genera instrucciones de máquina para cada método a medida que son necesarias.

Nota: El valor predeterminado de i5/OS consiste en interpretar (no compilar) métodos Java utilizando el intérprete de modalidad mixta (MMI). MMI perfila cada método Java a medida que lo interpreta. Tras alcanzar el umbral especificado por la propiedad os400.jit.mmi.threshold, MMI especifica entonces que i5/OS utilice el compilador JIT para compilar el método.

Para obtener más información, vea las entradas correspondientes a la propiedad java.compiler y a la propiedad os400.jit.mmi.threshold en el tema Lista de propiedades Java del sistema.

Conceptos relacionados

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Recogida de basura en Java

La recogida de basura es el proceso por el cual se libera el almacenamiento que utilizan los objetos a los que ya no hace referencia un programa. Gracias a la recogida de basura, ya no es necesario que los programadores escriban código, susceptible de sufrir errores, para "liberar" o "suprimir" de manera explícita los objetos. En muchas ocasiones, el resultado que da este código son errores de programa que provocan "fugas de memoria". El recogedor de basura detecta automáticamente el objeto o grupo de objetos a los que ya no puede llegar el programa de usuario. Lo detecta porque ya no hay referencias al objeto en ninguna de las estructuras del programa. Una vez recogido un objeto, se puede asignar el espacio para otros usos.

El entorno Java de tiempo de ejecución (JRE) incluye un recogedor de basura que libera la memoria que ya no se utiliza. El recogedor de basura se ejecuta automáticamente según convenga.

El recogedor de basura puede iniciarse también de forma explícita bajo el control del programa Java; para ello, debe utilizarse el método java.lang.Runtime.gc().

Conceptos relacionados

"Utilizar el mandato Trabajar con trabajos de JVM" en la página 422
Si emplea la máquina virtual de tecnología IBM para Java, puede utilizar el mandato CL Trabajar con trabajos de JVM (WRKJVMJOB) para recoger datos de rendimiento.

Recogida de basura avanzada de IBM Developer Kit para Java

IBM Developer Kit para Java implementa un algoritmo de recogedor de basura avanzado. Este algoritmo permite descubrir y recoger objetos no alcanzables sin que se produzcan pausas significativas en el funcionamiento del programa Java. Un recogedor concurrente descubre de manera cooperativa las referencias hechas a objetos en las hebras en ejecución, en lugar de en una sola hebra.

Muchos recogedores de basura son de "detención total". Esto significa que, en el punto en que se produce un ciclo de recogida, se detienen todas las hebras, excepto la que efectúa la recogida de basura, mientras el recogedor de basura realiza su trabajo. Cuando esto sucede, los programas Java sufren una pausa y la prestación multiprocesador que pueda tener la plataforma se malgasta por lo que a Java se refiere mientras el recogedor realiza su trabajo. El algoritmo de System i no detiene simultáneamente todas las hebras del programa. Por el contrario, deja que las hebras sigan funcionando mientras el recogedor de basura efectúa su tarea. Esto impide que se produzcan pausas y permite utilizar todos los procesadores mientras tiene lugar la recogida de basura.

La recogida de basura se efectúa de forma automática tomando como base los parámetros especificados al iniciar la máquina virtual Java. La recogida de basura también se puede iniciar explícitamente bajo el control del programa Java utilizando el método java.lang.Runtime.gc().

Para obtener una definición básica, vea: Recogida de basura Java.

Para obtener las opciones de recogida de basura disponibles con la tecnología IBM para Java, vea: Diagnósticos del recogedor de basura en el manual Java Diagnostics Guide 6.

Información relacionada

Tecnología Java de estilo IBM: políticas de recogida de basura

Consideraciones sobre el rendimiento de la recogida de basura Java

La recogida de basura en la máquina virtual Java de i5/OS funciona en modalidad asíncrona continua. El parámetro tamaño inicial de recogida de basura (GCHINL) del mandato Ejecutar Java (RUNJVA) puede afectar al rendimiento de las aplicaciones.

El parámetro GCHINL significar especifica el espacio de objetos nuevos que está permitido entre recogidas de basura. Si el valor es bajo, puede provocar una actividad general excesiva de recogida de basura. Si es alto, la recogida de basura puede sufrir limitaciones y provocar errores por falta de memoria. Sin embargo, para la mayoría de las aplicaciones, los valores predeterminados deberían ser correctos.

La recogida de basura determina si un objeto ya no es necesario evaluando si hay o no referencias válidas a dicho objeto.

Consideraciones sobre el rendimiento de la invocación de métodos nativos Java

La invocación de métodos nativos en la plataforma System i podría no realizarse tan bien como la invocación de métodos nativos en otras plataformas.

En el caso de la JVM clásica, se ha optimizado Java en el System i5 a base de trasladar la máquina virtual Java por debajo de la interfaz de máquina (MI). La invocación de métodos nativos requiere una llamada por encima del código MI y puede exigir costosas llamadas a la interfaz Java nativa (JNI) de vuelta a la máquina virtual Java. La JVM de tecnología IBM para Java se ejecuta por encima de la MI. En ambos casos, clásica y tecnología IBM para Java, los métodos nativos no deben llevar a cabo rutinas pequeñas,

que resultan fáciles de escribir en Java. Solo debe utilizar los métodos nativos para iniciar funciones del sistema que sean de relativamente larga ejecución y que no estén disponibles directamente en Java.

Consideraciones sobre el rendimiento de la excepción de Java

La arquitectura de excepciones de System i permite prestaciones versátiles de interrupción y reintento. También permite la interacción de lenguajes mixtos. El hecho de lanzar excepciones Java en la plataforma System i puede resultar más costoso que en otras plataformas. Esto no debe afectar al rendimiento global de la aplicación a menos que se utilicen rutinariamente excepciones Java en la vía habitual de la aplicación.

Herramientas de rendimiento de rastreo de llamadas Java

Los rastreos de llamadas a métodos Java facilitan información significativa de rendimiento acerca del tiempo que se invierte en cada uno de los métodos Java.

La salida de rastreo de llamada/retorno generada con el mandato Imprimir informe del explorador de rendimiento (PRTPEXRPT) muestra el tiempo de CPU para cada una de las llamadas de todos los métodos Java de los que se ha realizado un rastreo. En algunos casos, puede que no resulte posible habilitar todos los archivos de clase para el rastreo de llamada/retorno. O tal vez esté llamando a métodos nativos y a funciones del sistema que no estén habilitados para el rastreo. En esta situación, el tiempo de CPU invertido en dichos métodos o funciones del sistema se acumula. Después, se envía notificación al último método Java que se haya llamado y que esté habilitado.

Herramientas de rendimiento del perfilado Java

El perfilado de unidad central de proceso (CPU) a escala de todo el sistema calcula el tiempo relativo de CPU que se invierte en cada uno de los métodos Java y en todas las funciones del sistema que el programa Java utiliza.

Emplee una definición del explorador de rendimiento (PEX) que rastree los eventos de ciclo de ejecución de desbordamiento del contador del supervisor de rendimiento (*PMCO). Las muestras suelen especificarse a intervalos de un milisegundo. Para recoger un perfil de rastreo válido, debe ejecutar la aplicación Java hasta que acumule de dos a tres minutos de tiempo de CPU. Esto debería generar más de 100.000 muestras. El mandato Imprimir informe del explorador de rendimiento (PRTPEXRPT) genera un histograma del tiempo de CPU invertido en toda la aplicación. Esto incluye todos los métodos Java y toda actividad a nivel de sistema.

Nota: El perfilado de CPU no muestra el uso relativo de CPU de los programas Java interpretados.

Interfaz de herramientas de la máquina virtual Java (JVMTI)

La interfaz de herramientas de la máquina virtual Java sirve para analizar la máquina virtual Java (JVM).

JVMTI sustituye a la interfaz de perfilador de la máquina virtual Java (JVMPI) y a la interfaz de depuración de la máquina virtual Java (JVMDI). JVMTI tiene la misma funcionalidad que ambas JVMDI y JVMPI, además de otras funciones. JVMTI se añade como parte de J2SE 5.0. En JDK 6, las interfaces JVMDI y JVMPI han dejado de ofrecerse, y JVMTI es la única opción disponible.

El soporte de JVMTI coloca ganchos en la JVM y en el compilador Just-In-Time (JIT) que, cuando se activan, proporcionan información de eventos a un agente de perfilado. El agente de perfilado se implementa como un programa de servicio del entorno de lenguajes integrados (ILE). El perfilador envía información de control a la JVM para habilitar e inhabilitar los eventos de JVMTI. Por ejemplo, el perfilador puede no estar interesado en ganchos de entrada o salida de método, y puede indicar a la JVM que no desea recibir estas notificaciones de eventos. La JVM y JIT tienen ganchos de eventos JVMTI incorporados que envían notificaciones de eventos al agente de perfilado si el evento está habilitado. El perfilador indica a la JVM qué eventos son de interés, y la JVM envía notificaciones de los eventos al perfilador cuando se producen.

Hay un programa de servicio denominado QJVAJVMTI, que reside en la biblioteca QSYS y da soporte a las funciones de JVMTI.

Información relacionada

Interfaz de herramientas de la máquina virtual Java (JVMTI) de Sun Microsystems, Inc.

Recoger datos de rendimiento Java

Para recoger datos de rendimiento Java en el servidor, siga estos pasos.

1. Cree una definición del explorador de rendimiento (PEX) que especifique:
   - Un nombre definido por el usuario
   - El tipo de recogida de datos
   - El nombre de trabajo
   - Los eventos del sistema sobre los que desea recoger información del sistema

   Nota: Si la salida que desea es de tipo `java_g -prof` y sabe cuál es el nombre concreto del trabajo del programa Java, es preferible que la definición de PEX sea *STATS en lugar de *TRACE.

   A continuación se ofrece un ejemplo de una definición *STATS:

   ```
   ADDPEXDFN DFN(YOURDFN) JOB(*ALL/YOURID/QJVACMDSRV) DTAORG(*HIER)
   TEXT('su definición stats')
   ```

   Esta definición *STATS no obtiene todos los eventos Java en ejecución. Solo se hace el perfilado de los eventos Java que estén en su sesión Java. Esta modalidad de operación puede incrementar el tiempo que se tarda en ejecutar el programa Java.

   A continuación se ofrece un ejemplo de una definición *TRACE:

   ```
   ADDPEXDFN DFN(YOURDFN) TYPE(*TRACE) JOB(*ALL) TRCTYPE(*SLTEVT)
   SLTEVT(*YES) PGMEVT(*JVAENTRY *JVAEXIT)
   ```

   Esta definición de *TRACE recoge datos procedentes de eventos de entrada y de salida de los programas Java del sistema. El compilador justo a tiempo (JIT) genera código con ganchos de entrada y salida cuando se habilita *JVAENTRY y *JVAEXIT en la definición de PEX utilizando la propiedad os400.enbpfrcol del sistema. El análisis de este tipo de recogida puede ser más lento que en el caso de un rastreo *STATS, en función de cuántos eventos de programa Java se tengan y de cuál sea la duración de la recogida de datos de PEX.
2. Habilite *JVAENTRY y *JVAEXIT en la definición de PEX utilizando la propiedad os400.enbpfrcol del sistema. Establezca que el valor de os400.enbpfrcol sea igual a **1**.
3. Inicie la recogida de datos de PEX con el mandato Arrancar explorador de rendimiento (STRPEX).
4. Ejecute el programa que desea analizar.

   Este programa no debe estar en un entorno de producción. Generará un volumen elevado de datos en poco tiempo. Debe limitar el tiempo de recogida a cinco minutos. Un programa Java que se ejecute durante ese tiempo genera muchos datos PEX del sistema. Si se recogen demasiados datos, se necesitará demasiado tiempo para procesarlos.
5. Finalice la recogida de datos de PEX con el mandato Finalizar explorador de rendimiento (ENDPEX).

   Nota: Si no es la primera vez que ha finalizado la recogida de datos de PEX, debe especificar *YES en sustituir archivo o, de lo contrario, no se guardarán los datos.
6. Conecte el directorio del sistema de archivos integrado con el visor que prefiera: java_g -prof o Jinsight.

   Puede copiar este archivo desde el servidor y utilizarlo como entrada de cualquier herramienta de perfilado que considere oportuna.

Conceptos relacionados

| “Propiedades Java del sistema” en la página 15
| Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son
| parecidas a los valores del sistema o a las variables de entorno de i5/OS.

| **Información relacionada**

| Mandato CL Crear programa Java (CRTJVAPGM)

| Utilizar el mandato Trabajar con trabajos de JVM

| Si emplea la máquina virtual de tecnología IBM para Java, puede utilizar el mandato CL Trabajar con
| trabajos de JVM (WRKJVMJOB) para recoger datos de rendimiento.

| Puede acceder a la información disponible del mandato WRKJVMJOB desde la pantalla Trabajar con
| trabajo (WRKJOB) y también emitiendo el mandato WRKJVMJOB. Solo se visualizará información
| relacionada con los trabajos de la máquina virtual de tecnología IBM para Java.

| La información o funcionalidad que está disponible al utilizar WRKJVMJOB es la siguiente:

| • Los argumentos y opciones con los que se ha iniciado la JVM.
| • Las variables de entorno de ILE y de PASE.
| • Peticiones de bloqueo Java lock pendientes para el trabajo de JVM.
| • Información sobre recogida de basura.
| • Propiedades Java del sistema.
| • La lista de hebras asociadas a la JVM.
| • Las anotaciones parcialmente completadas del trabajo de JVM.
| • Capacidad para trabajar con archivos de entrada y salida en spool del trabajo de JVM.
| • Capacidad para generar vuelcos de JVM (del sistema, de memoria dinámica, Java) desde una opción de
| panel. Estas prestaciones también están disponibles desde el mandato Generar vuelco de JVM
| (GENJVMDMP).
| • Capacidad para habilitar e inhabilitar la recogida de basura verbosa desde una opción de panel.

| **Información relacionada**

| Descripción del mandato CL WRKJVMJOB

Mandatos y herramientas de IBM Developer Kit para Java

Al utilizar IBM Developer Kit para Java, puede emplear herramientas Java con el intérprete Qshell o bien emplear mandatos CL.

Si ya tiene experiencia en la programación Java, tal vez le resulte más cómodo utilizar las herramientas Java del intérprete Qshell, ya que son parecidas a las herramientas que utilizaría con Java Development Kit. En el tema Qshell encontrará información sobre cómo utilizar el entorno Qshell.

Si es usted programador de i5/OS, tal vez le interese utilizar los mandatos CL para Java más habituales del entorno System i. Si desea obtener más información sobre cómo se utilizan los mandatos CL y los mandatos de System i Navigator, siga leyendo.

Información relacionada

Intérprete Qshell

Herramientas Java soportadas por IBM Developer Kit para Java

El entorno Qshell incluye las herramientas de desarrollo Java (JDT) que se suelen necesitar para el desarrollo de programas.

Con algunas excepciones, las herramientas Java soportan la sintaxis y las opciones documentadas por Sun Microsystems, Inc. Todas deben ejecutarse mediante el intérprete Qshell.

Para iniciar el intérprete Qshell, puede utilizar el mandato Arrancar Qshell (STRQSH o QSH). Cuando el intérprete Qshell está en ejecución, aparece la pantalla Entrada de mandato QSH. En esta pantalla aparecen los datos de salida y los mensajes de los programas y herramientas Java que se ejecutan en Qshell. En esta pantalla también se pueden leer los datos que se hayan entrado en un programa Java.

Nota: Las funciones de entrada de mandatos de i5/OS command no están disponibles directamente desde Qshell. Para obtener una línea de mandatos, pulse F21 (Entrada de mandatos CL).

Conceptos relacionados

"NAWT (Native Abstract Windowing Toolkit)" en la página 248
Native Abstract Windowing Toolkit (NAWT) no es en realidad un kit de herramientas, sino más bien un término que ha evolucionado para referirse al soporte de i5/OS nativo que proporciona a las aplicaciones y servlets Java capacidad para utilizar las funciones gráficas de use the Abstract Windowing Toolkit (AWT) que ofrece la plataforma Java 2, Standard Edition (J2SE).

Herramientas Java

Consulte estos temas para obtener una descripción de las herramientas Java.

Herramienta Java appletviewer:

La herramienta Java `appletviewer` le permite ejecutar applets sin un navegador Web. Es compatible con la herramienta `appletviewer` proporcionada por Sun Microsystems, Inc.

Para ejecutar la herramienta `appletviewer`, debe utilizar Native Abstract Window Toolkit (NAWT) y la clase `sun.applet.AppletViewer`, pero también puede ejecutar la herramienta appletviewer en el intérprete Qshell.

El siguiente es un ejemplo de cómo usar la clase `sun.applet.AppletViewer` y ejecutar el ejemplo demo TicTacToe. Para obtener información sobre cómo cargar los ejemplos demo, vea las instrucciones de: Cómo extraer archivos de ejemplo.

En la línea de mandatos, entre:

```
cd '/home/MyUserID/demo/applets/TicTacToe'
```

Para JDK 1.4, emita el mandato:

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.4))
```

Para JDK 1.5, emita el mandato:

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.5))
```

| El siguiente es un ejemplo del uso de la herramienta `appletviewer` en el intérprete Qshell y ejecutar el ejemplo demo TicTacToe. Para obtener información sobre cómo cargar los ejemplos demo, vea las instrucciones de: Cómo extraer archivos de ejemplo.

Los mandatos correspondientes serían:

```
cd /home/MyUserID/demo/applets/TicTacToe
```

Para JDK 1.4, emita el mandato:

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.4 example1.html
```

Para JDK 1.5, emita el mandato:

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.5 example1.html
```

Nota: -J son distintivos de tiempo de ejecución para Appletviewer. -D son propiedades.

Conceptos relacionados

"NAWT (Native Abstract Windowing Toolkit)" en la página 248
Native Abstract Windowing Toolkit (NAWT) no es en realidad un kit de herramientas, sino más bien un término que ha evolucionado para referirse al soporte de i5/OS nativo que proporciona a las aplicaciones y servlets Java capacidad para utilizar las funciones gráficas de use the Abstract Windowing Toolkit (AWT) que ofrece la plataforma Java 2, Standard Edition (J2SE).

Información relacionada

Herramienta appletviewer de Sun Microsystems, Inc.

Cómo extraer archivos de ejemplo:

El siguiente procedimiento muestra una manera de extraer los archivos de ejemplo antes de ejecutar la herramienta Java appletviewer. El procedimiento presupone que desea extraer los archivos de ejemplo en el directorio inicial.

1. Entre el mandato Iniciar Qshell (QSH) en la línea de mandatos.
2. Si aún no existe, cree un directorio del sistema de archivos integrado (IFS) a nivel inicial para su ID de usuario:
   ```
   mkdir /home/MyUserID
   ```
3. Cree un directorio demo dentro del directorio IFS:
   ```
   mkdir /home/MyUserID/demo
   ```
4. Cambie de directorio al directorio demo:
   ```
   cd /home/myUserId/demo
   ```
5. Para JDK 1.4, utilice este mandato:
   ```
   jar xf /QIBM/ProdData/Java400/jdk14/demo.jar
   ```
   Para JDK 1.5, utilice este mandato:
   ```
   jar xf /QIBM/ProdData/Java400/jdk15/demo.jar
   ```

Herramienta Java apt:

La herramienta `apt` Java procesa anotaciones de programa.

La herramienta `apt` solo está disponible con JDK 1.5 y versiones posteriores. La herramienta `apt` está disponible en el intérprete Qshell.

Hallará más información sobre la herramienta `apt` en Herramienta apt de Sun Microsystems, Inc.

Enlace fuera de Information Center

Herramienta Java extcheck:

En la plataforma Java 2, Standard Edition (J2SE), la herramienta `extcheck` detecta los conflictos de versión entre un archivo JAR destino y los archivos JAR de extensión instalados actualmente. Es compatible con la herramienta `keytool` suministrada por Sun Microsystems, Inc.

La herramienta `extcheck` está disponible cuando se utiliza el intérprete Qshell.

Información relacionada

Herramienta extcheck de Sun Microsystems, Inc.

Aplicación Java hwkeytool:

La aplicación `hwkeytool` le permite utilizar las prestaciones de criptografía del coprocesador criptográfico modelo 4764 con la extensión de criptografía Java (JCE) y la arquitectura de criptografía Java (JCA).

| La aplicación `hwkeytool` para hardware utiliza la misma sintaxis y los mismos mandatos que la aplicación `keytool`, con la salvedad de dos mandatos y el almacén de claves predeterminado. La aplicación keytool para hardware proporciona parámetros adicionales en los mandatos `-genkey` y `delete`.

| En el mandato `-genkey`, los parámetros adicionales que están disponible son:

| **-KeyLabel**
| Le permite utilizar una etiqueta específica para la clave por hardware.

| **-hardwaretype**
| Determinar el tipo del par de claves: conjunto de datos de clave pública (PKDS) o RETAINED.

| **-hardwareusage**
| Establecer el uso del par de claves que se generan, ya sea una clave solo de firma o una clave de firma y gestión de claves.

| En el mandato `delete`, hay un parámetro adicional llamado **-hardwarekey** que suprime el par de claves del almacén de claves y del hardware.

| El nombre del almacén de claves predeterminado es `.HWkeystore`. Se puede cambiar utilizando el parámetro **-keystore**.

| Coprocesador criptográfico 4764

| "Herramienta Java keytool" en la página 427
| En la plataforma Java 2, Standard Edition (J2SE), la herramienta `keytool` crea pares de claves pública y privada, certificados autofirmados, y además gestiona almacenes de claves. En J2SDK, las herramientas `jarsigner` y `keytool` sustituyen a la herramienta `javakey`. Es compatible con la herramienta `keytool` suministrada por Sun Microsystems, Inc.

Herramienta Java idlj:

La herramienta `idlj` genera enlaces Java a partir de un archivo de lenguaje de definición de interfaces (IDL) dado.

La herramienta `idlj` también se conoce como compilador de IDL a Java. Es compatible con la herramienta `idlj` proporcionada por Sun Microsystems, Inc.

Información relacionada

Herramienta idlj de Sun Microsystems, Inc.

Herramienta Java jar:

La herramienta `jar` combina varios archivos en un único archivo JAR Java. Es compatible con la herramienta `jar` proporcionada por Sun Microsystems, Inc.

La herramienta `jar` está disponible en el intérprete Qshell.

Sistema de archivos integrado

jar - La herramienta Java de archivado (JAR) de Sun Microsystems, Inc.

Herramienta Java jarsigner:

En la plataforma Java 2, Standard Edition (J2SE), la herramienta `jarsigner` firma archivos JAR y verifica las signaturas de los archivos JAR firmados.

La herramienta `jarsigner` accede al almacén de claves, creado y gestionado por la herramienta `keytool`, cuando tiene que localizar la clave privada para firmar un archivo JAR. En J2SDK, las herramientas `jarsigner` y `keytool` sustituyen a la herramienta `javakey`. Es compatible con la herramienta `jarsigner` proporcionada por Sun Microsystems, Inc.

La herramienta `jarsigner` está disponible en el intérprete Qshell.

Información relacionada

jarsigner - Herramienta de firmado y verificación de JAR de Sun Microsystems, Inc.

Herramienta Java javac:

La herramienta `javac` compila programas Java. Es compatible con la herramienta `javac` proporcionada por Sun Microsystems, Inc., con una excepción.

-classpath
No altera temporalmente la vía de acceso de clases por omisión. En lugar de ello, la opción se añade al final de la vía de acceso de clases por omisión del sistema. La opción `-classpath` altera temporalmente la variable de entorno CLASSPATH.

La herramienta `javac` está disponible en el intérprete Qshell.

Información relacionada

javac - Compilador del lenguaje de programación Java de Sun Microsystems, Inc.

Herramienta Java javadoc:

La herramienta `javadoc` genera documentación de API. Es compatible con la herramienta `javadoc` proporcionada por Sun Microsystems, Inc.

La herramienta `javadoc` está disponible en el intérprete Qshell.

Información relacionada

Herramienta javadoc de Sun Microsystems, Inc.

Herramienta Java javah:

La herramienta `javah` facilita la implementación de los métodos Java nativos. Es compatible con la herramienta `javah` proporcionada por Sun Microsystems, Inc., con algunas excepciones.

Nota: Si se escriben métodos nativos, significa que la aplicación no es 100% puro Java. También significa que la aplicación no es portable directamente de una plataforma a otra. Los métodos nativos son, por naturaleza, específicos de una plataforma o un sistema. La utilización de métodos nativos puede incrementar el coste de desarrollo y mantenimiento de las aplicaciones.

La herramienta `javah` está disponible en el intérprete Qshell. Lee un archivo de clase Java y crea un archivo de cabecera escrito en C dentro del directorio de trabajo actual. El archivo de cabecera que se escribe es un archivo continuo de System i5 (STMF). Para poder incluirlo en un programa C de System i5, primero hay que copiarlo en un miembro de archivo.

La herramienta `javah` es compatible con la herramienta proporcionada por Sun Microsystems, Inc. Sin embargo, si se especifican estas opciones, el servidor las ignora.

-td La herramienta `javah` de System i5 no necesita un directorio temporal.

-stubs

Java en el System i5 solo permite usar métodos nativos en formato de interfaz Java nativa (JNI). Los apéndices solo se necesitaron para el formato de métodos nativos anterior a JNI.

-trace Guarda relación con la salida de archivo de apéndice .c, que no está soportada por Java en el System i5.

-v No está soportada.

Nota: Se debe especificar siempre la opción -jni. El System i5 no permite usar implementaciones de métodos nativos anteriores a JNI.

Información relacionada

Herramienta javah de Sun Microsystems, Inc.

Herramienta Java javap:

La herramienta `javap` desensambla archivos Java compilados e imprime una representación del programa Java. Esto puede resultar útil cuando el código fuente original ha dejado de estar disponible en un sistema.

-b Se hace caso omiso de esta opción. La compatibilidad hacia atrás no es necesaria, porque, en el System i5, Java solo permite utilizar las versiones recientes del Java Development Kit (JDK).

-p En el System i5, la opción -p no es válida. Hay que escribir -private.

-verify
Se hace caso omiso de esta opción. La herramienta `javap` no realiza verificación alguna en el System i5.

La herramienta `javap` está disponible en el intérprete Qshell.

Nota: La utilización de la herramienta javap para desensamblar clases puede constituir una violación del acuerdo de licencia de dichas clases. Antes de utilizar la herramienta `javap`, consulte el acuerdo de licencia de las clases.

Información relacionada

Herramienta javap de Sun Microsystems, Inc.

Herramienta Java keytool:

En la plataforma Java 2, Standard Edition (J2SE), la herramienta `keytool` crea pares de claves pública y privada, certificados autofirmados, y además gestiona almacenes de claves. En J2SDK, las herramientas `jarsigner` y `keytool` sustituyen a la herramienta `javakey`. Es compatible con la herramienta `keytool` suministrada por Sun Microsystems, Inc.

La herramienta `keytool` está disponible en el intérprete Qshell.

Información relacionada

Herramienta keytool de Sun Microsystems, Inc.

Herramienta Java native2ascii:

La herramienta `native2ascii` convierte un archivo que contenga caracteres nativos (caracteres que no son Latin-1 ni Unicode) en un archivo con caracteres Unicode. Es compatible con la herramienta `native2ascii` proporcionada por Sun Microsystems, Inc.

La herramienta `native2ascii` está disponible en el intérprete Qshell.

Información relacionada

Herramienta native2ascii de Sun Microsystems, Inc.

Herramienta Java orbd:

La herramienta `orbd` proporciona soporte de clientes para localizar e invocar objetos persistentes de forma transparente en servidores del entorno CORBA.

ORBD se utiliza en lugar del Servicio de denominación transitorio (tnameserv), que incluye un Servicio de denominación transitorio y un Servicio de denominación persistente. La herramienta `orbd` incorpora la funcionalidad de un Gestor de servidor, un Servicio de denominación interoperativo y un Servidor de nombres de rutina de carga. Cuando se utiliza conjuntamente con `servertool`, el Gestor de servidor localiza, registra y activa un servidor cuando un cliente desea acceder al servidor.

Información relacionada

Herramienta orbd de Sun Microsystems, Inc.

Herramienta Java pack200:

La herramienta `pack200` es una aplicación Java que comprime un archivo JAR en un archivo `pack200`.

La herramienta `pack200` solo está disponible con JDK 1.5 y versiones posteriores. La herramienta `pack200` está disponible al utilizar el intérprete Qshell.

Hallará más información en Herramienta pack200 de Sun Microsystems, Inc. Enlace fuera de Information Center

Conceptos relacionados

"Herramienta Java unpack200" en la página 429
La herramienta Java `unpack200` descomprime un archivo `pack200` para convertirlo en un archivo JAR.

Herramienta Java policytool:

En Java 2 SDK, Standard Edition, la herramienta `policytool` crea y cambia los archivos de configuración de política externa que define la política de seguridad Java de su instalación. Es compatible con la herramienta `policytool` proporcionada por Sun Microsystems, Inc.

`policytool` es una herramienta de interfaz gráfica de usuario (GUI) que está disponible cuando se utiliza el intérprete Qshell y NAWT (Native Abstract Window Toolkit). Hallará más información en IBM Developer Kit para Java Native Abstract Window Toolkit.

Información relacionada

Herramienta policytool de Sun Microsystems, Inc.

Herramienta Java rmic:

La herramienta `rmic` genera archivos de apéndice y archivos de clase para los objetos Java. Es compatible con la herramienta `rmic` proporcionada por Sun Microsystems, Inc.

La herramienta `rmic` está disponible en el intérprete Qshell.

Para obtener más información sobre la herramienta `rmic`, consulte la herramienta rmic de Sun Microsystems, Inc.

Herramienta Java rmid:

En la plataforma Java 2, Standard Edition (J2SE), la herramienta `rmid` inicia el daemon del sistema de activación, para que los objetos se puedan registrar y activar en una máquina virtual Java. Es compatible con la herramienta `rmid` suministrada por Sun Microsystems, Inc.

La herramienta `rmid` está disponible en el intérprete Qshell.

Información relacionada

Herramienta rmid de Sun Microsystems, Inc.

Herramienta Java rmiregistry:

La herramienta `rmiregistry` inicia un registro de objetos remotos en un puerto especificado. Es compatible con la herramienta `rmiregistry` proporcionada por Sun Microsystems, Inc.

La herramienta `rmiregistry` está disponible en el intérprete Qshell.

Información relacionada

Herramienta rmiregistry de Sun Microsystems, Inc.

Herramienta Java serialver:

La herramienta `serialver` devuelve el número de versión o el identificador exclusivo de serialización correspondiente a una o varias clases. Es compatible con la herramienta `serialver` proporcionada por Sun Microsystems, Inc.

La herramienta `serialver` está disponible en el intérprete Qshell.

Información relacionada

Herramienta serialver de Sun Microsystems, Inc.

Herramienta Java servertool:

La herramienta `servertool` proporciona una interfaz de línea de mandatos para que los programadores de aplicaciones registren, desregistren, inicien y concluyan un servidor persistente.

Información relacionada

Herramienta servertool de Sun Microsystems, Inc.

Herramienta Java tnameserv:

En la plataforma Java 2, Standard Edition (J2SE), la herramienta `tnameserv` (servicio de denominación transitorio) proporciona acceso al servicio de denominación. Es compatible con la herramienta `tnameserv` proporcionada por Sun Microsystems, Inc.

La herramienta `tnameserv` está disponible en el intérprete Qshell.

Herramienta Java unpack200:

La herramienta Java `unpack200` descomprime un archivo `pack200` para convertirlo en un archivo JAR.

La herramienta `unpack200` solo está disponible con JDK 1.5 y versiones posteriores. La herramienta `unpack200` está disponible en el intérprete Qshell.

Hallará más información en Herramienta unpack200 de Sun Microsystems, Inc. Enlace fuera de Information Center

Conceptos relacionados

"Herramienta Java pack200" en la página 428
La herramienta `pack200` es una aplicación Java que comprime un archivo JAR en un archivo `pack200`.

Mandato java de Qshell

El mandato `java` de Qshell ejecuta programas Java. Es compatible con la herramienta `java` proporcionada por Sun Microsystems, Inc., con algunas excepciones.

IBM Developer Kit para Java ignora estas opciones del mandato `java` de Qshell.

Opción	Descripción
-cs	Esta opción no está soportada.
-checksource	Esta opción no está soportada.
-debug	Esta opción está soportada por el depurador interno de System i5.
-noasyncgc	La recogida de basura siempre se está ejecutando con IBM Developer Kit para Java.
-prof	El System i5 tiene herramientas de rendimiento propias.
-ss	Esta opción no es aplicable a la plataforma System i5.
-oss	Esta opción no es aplicable a la plataforma System i5.
-t	El System i5 utiliza una función de rastreo propia.
-verify	Verificar siempre en la plataforma System i5.
-verifyremote	Verificar siempre en la plataforma System i5.
-noverify	Verificar siempre en la plataforma System i5.

En la plataforma System i5, la opción `-classpath` no altera temporalmente la vía de acceso de clases predeterminada. En lugar de ello, la opción se añade al final de la vía de acceso de clases predeterminada del sistema. La opción `-classpath` altera temporalmente la variable de entorno CLASSPATH.

El mandato `java` de Qshell admite opciones de la plataforma System i5. Las opciones soportadas son:

Opción	Descripción
-chkpath	Esta opción comprueba si existe acceso público de escritura a los directorios de la variable CLASSPATH.
-Xrun[:]	Se visualiza un mensaje que indica un programa de servicio y una serie de parámetro opcional para la función JVM_OnLoad durante el inicio de la JVM. Esta opción ha caído en desuso. En su lugar hay que utilizar -agentlib: o -agentpath:.
-agentlib:	Indica un programa de servicio i5/OS que contiene un agente VM. La VM intenta cargar el programa de servicio desde una biblioteca i5/OS incluida en la lista de bibliotecas durante el inicio.
-agentpath:	Carga la biblioteca desde la vía de acceso absoluta que sigue a esta opción. No se produce la expansión del nombre de biblioteca y las opciones pasan al agente durante el inicio.

Opción	Descripción
-javaagent:<jarpath>[=<opciones>]	Carga los agentes del lenguaje de programación Java para utilizarlos con el paquete java.lang.instrument. *jarpath* es la vía de acceso al archivo JAR de agente. *opciones* son las opciones del agente. Puede utilizar -javaagent:<jarpath>[=<opciones>] más de una vez en la misma línea de mandatos para crear múltiples agentes. Más de un agente puede utilizar la misma *jarpath*.

El mandato Ejecutar Java (RUNJVA), en la información de consulta de Mandatos CL, describe detalladamente estas nuevas opciones. En la información de consulta de los mandatos CL Crear programa Java (CRTJVAPGM), Suprimir programa Java (DLTJVAPGM) y Visualizar programa Java (DSPJVAPGM) encontrará detalles sobre cómo gestionar los programas Java.

El mandato `java` de Qshell está disponible en el intérprete Qshell.

Para obtener más información sobre el mandato `java` de Qshell, le remitimos a la herramienta java de Sun Microsystems, Inc.

Mandatos CL soportados por Java

El entorno CL contiene mandatos CL para optimizar y gestionar programas Java.

- El mandato Analizar programa Java (ANZJVAPGM) analiza un programa Java, lista sus clases y muestra el estado actual de cada clase.
- El mandato Analizar máquina virtual Java (ANZJVM) recupera y establece información en una máquina virtual Java (JVM). Este mandato ayuda a depurar programas Java devolviendo información acerca de las clases activas.
- El mandato Cambiar programa Java (CHGJVAPGM) cambia los atributos de un programa Java.
- El mandato Crear programa Java (CRTJVAPGM) crea un programa Java en un System i a partir de un archivo JAR, archivo ZIP o archivo de clase Java.
- El mandato Suprimir programa Java (DLTJVAPGM) suprime un programa Java de System i asociado a un archivo JAR, archivo ZIP o archivo de clase Java.
- El mandato Visualizar programa Java (DSPJVAPGM) visualiza información sobre un programa Java.
- El mandato Visualizar trabajos de máquina virtual Java (DSPJVMJOB) visualiza información sobre los trabajos de JVM activos para ayudarle a gestionar la aplicación de arreglos temporales de programa (PTF). También puede encontrar información más detallada sobre DSPJVMJOB en “Aplicar arreglos temporales del programa” en la página 579.
- El mandato Volcar máquina virtual Java (DMPJVM) vuelca información acerca de la máquina virtual Java para un trabajo especificado en un archivo de impresora en spool.
- El mandato Generar vuelco de JVM (GENJVMDMP) genera vuelcos de la máquina virtual Java (JVM) a petición.
- El mandato Imprimir trabajo de JVM (PRTJVMJOB) le permite imprimir máquinas virtuales Java (JVM) que se estén ejecutando en trabajos activos.
- El mandato JAVA y el mandato Ejecutar Java (RUNJVA) ejecutan programas Java de System i.
- Trabajar con trabajos de JVM (WRKJVMJOB) visualiza información sobre trabajos que se ejecutan en tecnología IBM para máquina virtual Java.

Series de parámetro de opción de código interno bajo licencia (LIC)

Interfaces API de programas y mandatos CL

Consideraciones sobre el uso del mandato ANZJVM

Debido a la posible duración de la ejecución de ANZJVM, es muy posible que una JVM finalice antes de que ANZJVM pueda finalizar. Si la JVM finaliza, ANZJVM devuelve el mensaje JVAB606 (es decir, la JVM ha finalizado mientras se procesaba ANZJVM) junto con los datos que ha podido obtener.

Tampoco existe límite superior en cuanto al número de clases que una JVM puede manejar. Si existen más clases que las que han podido manejarse, ANZJVM debe devolver los datos que pueden manejarse junto con un mensaje indicando que existe información adicional de la que no se ha informado. Si los datos requieren truncamiento, ANZJVM devuelve tanta información como es posible.

El parámetro interno está restringido a 3600 segundos (una hora) de duración. El número de clases acerca de las que ANZJVM puede devolver información está limitado por la cantidad de almacenamiento del sistema.

Mandatos de System i Navigator soportados por Java

System i Navigator es una interfaz gráfica del escritorio Windows. Forma parte de System i Access para Windows y cubre muchas de las funciones de i5/OS que los necesitan los administradores o los usuarios para llevar a cabo su trabajo diario. Los mandatos de System i Navigator le permiten crear y ejecutar programas Java.

System i Navigator admite Java en forma de plug-in contenido en la opción de sistemas de archivos de System i Access para Windows. Para utilizar el plug-in Java de System i Navigator, debe instalar IBM Developer Kit para Java en el servidor. Luego, para instalar el plug-in Java en el PC, seleccione Sistemas de archivos mediante la instalación selectiva de la carpeta Client Access.

Los archivos de clase, JAR, ZIP y Java residen en el sistema de archivos integrado. System i Navigator le permite ver esos archivos en el panel de la derecha. Pulse con el botón derecho del ratón el archivo de clase, JAR, ZIP o Java que desea utilizar. Así aparecerá un menú de contexto.

Si selecciona **Programa Java asociado --> Nuevo...** en el menú contextual, se inicia el transformador de Java, el cual crea programas Java de System i asociados al archivo de clase, JAR o ZIP. Un recuadro de diálogo le permite especificar detalles sobre cómo crear el programa. Los programas se pueden crear ya sea para transformación Java o para interpretación Java.

Nota: si selecciona la transformación, los bytecodes del archivo de clase se transformarán en instrucciones RISC, con lo que se obtiene mejor rendimiento que si se utiliza la interpretación.

Si selecciona **Programa Java asociado --> Editar...** en el menú contextual, podrá cambiar los atributos de los programas Java conectados a los archivos de clase, archivos ZIP o archivos JAR Java.

Si selecciona **Programa Java asociado --> Ejecutar...** en el menú contextual, se ejecutará el archivo de clase en el servidor. También puede seleccionar un archivo JAR o ZIP y ejecutar un archivo de clase situado en ese archivo JAR o ZIP. Aparecerá un diálogo que permite especificar información detallada sobre la manera de ejecutar el programa. Si ya ha seleccionado **Programa Java asociado --> Nuevo...**, se utiliza el programa Java de System i asociado al archivo de clase al ejecutar el programa. Si aún no existe ningún programa Java de System i que esté asociado al archivo de clase, el programa Java de System i se crea antes de ejecutar el programa.

Si selecciona **Programa Java asociado --> Suprimir...** en el menú contextual, se suprimirán los programas Java de System i asociados al archivo de clase, JAR o ZIP.

Si selecciona **Propiedades** en el menú contextual, aparece un diálogo de propiedades que contiene las pestañas **Programas Java** y **Opciones Java**. Estas pestañas le permiten ver los detalles de cómo se crearon los programas Java asociados de System i para el archivo de clase, JAR o ZIP.

Nota: estos paneles corresponden a la información del mandato Visualizar programa Java.

Si selecciona **Compilar archivo Java** en el menú contextual, los archivos Java que haya seleccionado se convertirán en los bytecodes de archivo de clase.

Vea la información de ayuda, incluida junto con System i Navigator, para conocer los parámetros y opciones de los diálogos **Programa Java nuevo**, **Editar programa Java**, **Ejecutar programa Java**, **Programas Java**, **Opciones de Java**, **Compilar archivo Java** y **Suprimir programa Java** de System i Navigator.

Depurar programas Java en i5/OS

Tiene varias opciones para depurar y resolver problemas de los programas Java que se ejecuten en el sistema, incluido el depurador de IBM System i5, la pantalla interactiva del sistema, los depuradores habilitados para el protocolo Java Debug Wire Protocol, y las herramientas de análisis de memoria dinámica (HAT) para Java.

La siguiente información no es una valoración completa de las posibilidades, pero enumera varias opciones.

Una de las maneras más fáciles de depurar programas Java que se ejecuten en su sistema consiste en utilizar el depurador de IBM System i5. El depurador de IBM System i5 proporciona una interfaz gráfica de usuario (GUI) que le permite utilizar con mayor facilidad las prestaciones de depuración del servidor. Puede utilizar la pantalla interactiva del servidor para depurar programas Java, aunque el depurador de System i5 proporciona una GUI más fácil de usar que le permite realizar las mismas funciones.

Además, la máquina virtual Java (JVM) de i5/OS soporta el protocolo Java Debug Wire Protocol (JDWP), que forma parte de la arquitectura de depuradores de la plataforma Java. Los depuradores habilitados para JDWP le permiten realizar la depuración remota desde clientes que se ejecutan en sistemas operativos distintos. (El depurador de IBM System i5 también le permite realizar la depuración remota de una manera parecida, aunque no utiliza el protocolo JDWP). Un programa de ese tipo, habilitado para JDWP, es el depurador de Java en la plataforma de herramientas universales del proyecto Eclipse.

Si el rendimiento del programa se degrada al ejecutarse durante más tiempo, es posible que haya codificado una fuga de memoria inadvertidamente. Puede utilizar las herramientas de análisis de memoria dinámica (HAT) para Java para ayudarle a depurar el programa y localizar las fugas de memoria, realizando análisis de memoria dinámica de aplicaciones y perfilado de creación de objetos Java a lo largo del tiempo.

Depurador de IBM System i5

"Arquitectura de depuradores de la plataforma Java" en la página 444
La arquitectura de depuradores de la plataforma Java (JPDA) consta de la interfaz de depuración de JVM/interfaz de herramientas de JVM, del protocolo Java Debug Wire Protocol y de la interfaz de depuración Java. Todos estos componentes de JPDA permiten a cualquier componente frontal de un depurador que utilice JDWP realizar operaciones de depuración. El componente frontal del depurador se puede ejecutar remotamente o como aplicación de System i5.

Depurar herramientas de desarrollo Java (JDT)

Sitio Web del proyecto Eclipse

Herramientas de análisis de memoria dinámica (HAT) para Java

Depurar programas Java utilizando el depurador de System i5

La manera más fácil de depurar programas Java que se ejecuten en su sistema consiste en utilizar el depurador de IBM System i5. El depurador de IBM System i5 proporciona una interfaz gráfica de usuario que le permite utilizar con mayor facilidad las prestaciones de depuración del sistema.

Para obtener más información sobre cómo utilizar el depurador de System i5 para depurar y probar programas Java que se ejecuten en su servidor, vea: IBM System i5 Debugger.

Depuración del sistema en el caso de la tecnología IBM para Java

Estas instrucciones presentan varias opciones para depurar las JMV de tecnología IBM para Java.

Depuración interactiva desde la línea de mandatos CL

La manera más sencilla de iniciar el depurador del sistema es con el parámetro OPTION(*DEBUG) del mandato CL JAVA. Por ejemplo:

```
> JAVA CLASS(Hello) OPTION(*DEBUG)
```

Habilitar la depuración de la JVM de tecnología IBM para Java

Para depurar un trabajo de la JVM de tecnología IBM para Java desde otro trabajo, hay que iniciar la JVM teniendo habilitada la depuración. La depuración Java se gestiona mediante un agente de depuración. El factor clave que permite depurar satisfactoriamente el código Java consiste en iniciar este agente durante el arranque de la JVM. Una vez que la JVM se haya iniciado satisfactoriamente con el agente de depuración, la JVM se puede depurar utilizando los mandatos Arrancar trabajo de servicio (STRSRVJOB) y Arrancar depuración (STRDBG) o bien desde la interfaz gráfica de usuario del depurador de System i5. En los apartados que siguen se presentan varias formas de iniciar el agente de depuración. En cada caso, la finalidad de un parámetro o de una variable de entorno es indicar que se debe iniciar el agente de depuración Java. En estas descripciones se empieza por presentar las situaciones más sencillas y se deja para el final las más complicadas.

Nota:

- El agente de depuración solo se necesita para la JVM de tecnología IBM para Java. No hace falta iniciarlo para la JVM clásica.
- El agente de depuración no suspende la JVM antes de entrar en el método main. En el caso de los programas de corta ejecución o para depurar el método main, puede ser necesario añadir código Java para detener la JVM. Una manera de hacerlo es con un bucle de espera (wait-loop) temporizado. Otra manera consiste en leer en la entrada estándar.
- Si se intenta depurar en una JVM que no tenga habilitado el agente de depuración, se envía un mensaje de diagnóstico JVAB307 a las anotaciones del trabajo de la JVM y del trabajo de servicio técnico. El texto del mensaje identifica el trabajo de la JVM que no tiene habilitada la depuración. Este mensaje indica que hay que reiniciar la JVM para poder depurarla satisfactoriamente. No se puede habilitar la depuración después de haber iniciado la JVM.

Habilitar la depuración Java desde CL

Para habilitar la depuración Java desde CL, añada el parámetro `AGTPGM(D9TI)` al mandato CL JAVA. Por ejemplo:

```
> JAVA CLASS(Hello) AGTPGM(D9TI)
```

Habilitar la depuración Java desde Qshell o desde el terminal PASE

Para habilitar la depuración Java desde Qshell (QSH) o desde el terminal PASE (QP2TERM), añada el parámetro `-debug` en la invocación `java`. Por ejemplo:

```
> java -debug Hello
```

La forma más sencilla de iniciar el agente de depuración es con el parámetro `-debug`. Es equivalente a añadir el parámetro `-agentlib:d9ti`. Para iniciar el agente de depuración, también se puede especificar:

```
> java -agentlib:d9ti Hello
```

Habilitar la depuración Java para una JVM de trabajo por lotes

Si la JVM de trabajo por lotes se inicia con el mandato CL Someter trabajo (SBMJOB), se puede añadir el parámetro `AGTPGM(D9TI)` al mandato CL JAVA. Por ejemplo, el siguiente mandato iniciará la JVM de trabajo por lotes con un agente de depuración:

```
> SBMJOB CMD(JAVA CLASS(HELLO) AGTPGM(D9TI))
        CPYENVVAR(*YES) ALWMLTTHD(*YES)
```

Si el trabajo por lotes se inicia con algún otro procedimiento, se puede utilizar la variable de entorno JAVA_TOOL_OPTIONS para iniciar el agente de depuración. La JVM consulta automáticamente la variable de entorno JAVA_TOOL_OPTIONS durante el inicio. Si se establece que sea igual a *-debug* o *-agentlib:d9ti*, se iniciará el agente de depuración para la JVM. Por ejemplo, para establecer la variable de entorno se puede utilizar uno de los siguientes mandatos:

```
> ADDENVVAR ENVVAR(JAVA_TOOL_OPTIONS) VALUE('-debug')
> ADDENVVAR ENVVAR(JAVA_TOOL_OPTIONS) VALUE('-agentlib:d9ti')
```

Si el trabajo por lotes no hereda automáticamente todas las variables de entorno, habrá que establecer la variable de entorno JAVA_TOOL_OPTIONS a escala del sistema. Por ejemplo:

```
> ADDENVVAR ENVVAR(JAVA_TOOL_OPTIONS) VALUE('-debug') LEVEL(*SYS)
```

Nota: Cuando establece la variable de entorno JAVA_TOOL_OPTIONS a escala del sistema, todas las JVM de tecnología IBM para Java que se inicien en el sistema lo hacen teniendo habilitada la depuración. **Esto puede provocar una disminución notable del rendimiento**.

Habilitar la depuración Java para una JVM creada con la API de invocación Java

Cuando se utiliza la API C/C++ JNI_CreateJavaVM para crear una JVM, puede habilitar la depuración mediante uno de estos métodos:

- Establezca que la variable de entorno JAVA_TOOL_OPTIONS sea igual a *-debug*.
- Establezca que la variable de entorno JAVA_TOOL_OPTIONS sea igual a *-agentlib:d9ti*.
- Añada el parámetro `-debug` a la lista de parámetros de opciones que se pasa a la API C/C++ JNI_CreateJavaVM.
- Añada el parámetro `-agentlib:d9ti` a la lista de parámetros de opciones que se pasa a la API C/C++ JNI_CreateJavaVM.

Además, para ver el código fuente Java de las clases que se depuran, habrá que establecer que la variable de entorno DEBUGSOURCEPATH señale hacia la ubicación del directorio base del código fuente Java.

Iniciar la JVM de tecnología IBM para Java desde la interfaz gráfica de usuario del depurador de System i5

Para iniciar una JVM de tecnología IBM para Java desde la interfaz gráfica de usuario del depurador de System i5, hay que establecer la variable de entorno JAVA_HOME al iniciar el trabajo de la JVM. Esta variable de entorno se puede establecer mediante la pantalla **Mandato de inicialización** al iniciar la JVM. Esta pantalla se encuentra en la ventana Iniciar depuración, en la interfaz del depurador de System i5.

Por ejemplo, para iniciar una JVM del JDK 5.0 32 de bites, añada el mandato siguiente a la pantalla **Mandato de inicialización**:

```
ADDENVVAR ENVVAR(JAVA_HOME) VALUE('/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit')
```

Nota: No se pueden usar puntos de observación para las variables locales en la JVM de tecnología IBM para Java. En la implementación de depuración del sistema de la JVM de tecnología IBM para Java se utiliza JVMTI, que no proporciona prestaciones de punto de observación para variables locales.

Información relacionada

Depurador de System i5

Depuración del sistema para las JVM clásicas

Estas instrucciones presentan varias opciones para depurar las máquinas virtuales Java (JVM) clásicas.

Depuración interactiva desde la línea de mandatos CL

La manera más sencilla de iniciar el depurador del sistema es con el parámetro OPTION(*DEBUG) del mandato CL JAVA. Por ejemplo:

```
> JAVA CLASS(Hello) OPTION(*DEBUG)
```

Habilitar la depuración para la JVM clásica

Para depurar un trabajo de la JVM clásica desde otro trabajo, hay que iniciar la JVM teniendo inhabilitado el compilador Just-in-Time (JIT). Una vez que la JVM se haya iniciado satisfactoriamente teniendo inhabilitado el compilador JIT, la JVM se puede depurar utilizando los mandatos Arrancar trabajo de servicio (STRSRVJOB) y Arrancar depuración (STRDBG) o bien desde la interfaz gráfica de usuario del depurador de System i5. En los apartados que siguen se presentan varias formas de inhabilitar el compilador JIT.

Nota:

- Solo es necesario inhabilitar el compilador JIT en el caso de la JVM clásica.
- Estas instrucciones no suspenden la JVM antes de entrar en el método main. En el caso de los programas de corta ejecución o para depurar el método main, puede ser necesario añadir código Java para detener la JVM. Una manera de hacerlo es con un bucle de espera (wait-loop) temporizado. Otra manera consiste en leer en la entrada estándar.
- Si se intenta depurar en una JVM que esté utilizando el compilador JIT, las funciones de depuración no funcionarán como cabría esperar. No se puede inhabilitar el compilador JIT después de haber iniciado la JVM.

Habilitar la depuración Java desde CL

Para habilitar la depuración Java desde CL, añada el parámetro `PROP((java.compiler NONE))` al mandato CL JAVA. Por ejemplo:

```
>  JAVA CLASS(Hello) PROP((java.compiler NONE))
```

Habilitar la depuración Java desde Qshell o desde el terminal PASE

Para habilitar la depuración Java desde Qshell (QSH) o desde el terminal PASE (QP2TERM), añada el parámetro `-Djava.compiler=NONE` en la invocación `java`. Por ejemplo:

```
> java -Djava.compiler=NONE Hello
```

Habilitar la depuración Java para una JVM de trabajo por lotes

Si la JVM de trabajo por lotes se inicia con el mandato CL Someter trabajo (SBMJOB), se puede añadir el parámetro `PROP((java.compiler NONE))` al mandato CL JAVA. Por ejemplo, el siguiente mandato iniciará la JVM de trabajo por lotes con un agente de depuración:

```
> SBMJOB CMD(JAVA CLASS(HELLO) PROP((java.compiler NONE)))
         CPYENVVAR(*YES) ALWMLTTHD(*YES)
```

Habilitar la depuración Java para una JVM creada con la API de invocación Java

Cuando se utiliza la API C/C++ JNI_CreateJavaVM para crear una JVM, puede inhabilitar el JIT mediante un archivo SystemDefault.properties para establecer `java.compiler=NONE`. Además, para ver el código fuente Java de las clases que se depuran, habrá que establecer que la variable de entorno DEBUGSOURCEPATH señale hacia la ubicación del directorio base del código fuente Java.

Iniciar la JVM clásica desde la interfaz gráfica de usuario del depurador de System i5

La JVM clásica se emplea por defecto cuando se inicia Java desde el depurador de System i5. No establezca la variable de entorno JAVA_HOME cuando inicie una JVM clásica.

Conceptos relacionados

"Archivo SystemDefault.properties" en la página 16
El archivo SystemDefault.properties es un archivo de propiedades Java estándar que le permite especificar propiedades predeterminadas del entorno Java.

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Información relacionada

Depurador de System i5

Operaciones de depuración

Puede utilizar la pantalla interactiva del servidor para emplear la opción `*DEBUG` y ver el código fuente antes de ejecutar el programa. Entonces puede establecer puntos de interrupción o bien emitir mandatos de recorrer principal o recorrer todo en un programa con el fin de analizar los errores mientras se ejecuta el programa.

Depurar programas Java utilizando la opción *DEBUG

Para depurar programas Java con la opción *DEBUG, siga estos pasos:

1. Compile el programa Java con la opción DEBUG, que es la opción -g de la herramienta `javac`.
2. Inserte el archivo de clase (.class) y el archivo fuente (.java) en el mismo in directorio del servidor.
3. Ejecute el programa Java emitiendo el mandato Ejecutar Java (RUNJVA) en la línea de mandatos de i5/OS. Especifique OPTION(*DEBUG) en el mandato Ejecutar Java (RUNJVA). Por ejemplo: `RUNJVA CLASS(nombre_clase) OPTION(*DEBUG)`

   Solo puede depurarse una clase. Si se especifica un nombre de archivo JAR para la palabra clave CLASS, OPTION(*DEBUG) no está soportado.
4. Se visualiza el fuente del programa Java.
5. Pulse F6 (Añadir/Borrar punto de interrupción), para establecer puntos de interrupción, o F10 (Recorrer), para recorrer paso a paso el programa.

Nota:

- Mientras utiliza los puntos de interrupción y los mandatos de recorrer, compruebe el flujo lógico del programa Java y, a continuación, vea las variables y cámbielas según convenga.
- La utilización de OPTION(*DEBUG) en el mandato RUNJVA inhabilita el compilador Just-In-Time (JIT).
- Si no tiene autorización para utilizar el mandato Arrancar trabajo de servicio (STRSRVJOB), se hará caso omiso de OPTION(*DEBUG).

Depurar programas Java desde otra pantalla

Al depurar un programa Java utilizando la pantalla interactiva del servidor, se visualiza el fuente del programa cada vez que este se encuentra con un punto de interrupción. Esto puede interferir con la

salida de pantalla del programa Java. Para evitarlo, depure el programa Java desde otra pantalla. La salida del programa Java se visualiza donde se ejecuta el mandato Java, y el fuente del programa aparece en la otra pantalla.

También es posible depurar de esta forma un programa Java, siempre y cuando se haya iniciado teniendo habilitada la depuración.

Nota: La depuración Java se puede habilitar de varias maneras:

- Si se propone utilizar la JVM clásica:
  - Especifique la propiedad `java.compiler=NONE` al iniciar la máquina virtual Java.
  - Especifique INTERPRET(*YES) en el mandato Ejecutar Java (RUNJVA).
- Si se propone utilizar Tecnología IBM para Java, hay que añadir la opción AGTPGM(D9TI) al mandato JAVA/RUNJVA para poder utilizar el depurador de System i5 con la JVM. No se necesita AGTPGM(D9TI) cuando se emplea OPTION(*DEBUG).

  No puede especificar la opción AGTPGM(D9TI) para la JVM clásica.

Para depurar Java desde otra pantalla, haga lo siguiente:

1. El programa Java debe estar retenido mientras empieza la puesta a punto de la depuración.

   Para retener el programa Java, puede hacer que el programa:
   - Espere a que se produzca una entrada desde el teclado.
   - Espere durante un intervalo de tiempo.
   - Entre en un bucle para comprobar una variable, lo que requiere que usted haya establecido un valor para sacar el programa Java del bucle.
2. Una vez retenido el programa Java, vaya a otra pantalla y siga estos pasos:
   a. Entre el mandato Trabajar con trabajos activos (WRKACTJOB) en la línea de mandatos.
   b. Busque el trabajo inmediato por lotes (BCI) en el que se está ejecutando el programa Java. Busque QJVACMDSRV en el listado Subsistema/trabajo. Busque su ID de usuario en el listado Usuario. Busque BCI bajo Tipo.
   c. Entre la opción 5 para trabajar con el trabajo.
   d. En la parte superior de la pantalla Trabajar con trabajo, figura el número, el usuario y el trabajo. Entre `STRSRVJOB Número/Usuario/Trabajo`
   e. Entre `STRDBG CLASS(nombreclase)`. Nombreclase es el nombre de la clase Java que desea depurar. Puede ser el nombre de clase que ha especificado en el mandato Java o puede ser el de otra clase.
   f. El fuente de dicha clase aparece en la pantalla Visualizar fuente de módulo.
   g. Establezca puntos de interrupción, pulsando F6 (Añadir/Borrar punto de interrupción), allí donde desee detenerse dentro de la clase Java. Pulse F14 para añadir más clases, programas o programas de servicio que depurar.
   h. Pulse F12 (Reanudar) para seguir ejecutando el programa.
3. Deje de retener el programa Java original. Cuando se llegue a un punto de interrupción, aparecerá la pantalla Visualizar fuente de módulo en la pantalla en la que se hayan entrado los mandatos Arrancar programa de servicio (STRSRVJOB) y Arrancar depuración (STRDBG). Cuando finalice el programa Java, aparecerá el mensaje `El trabajo al que se ha dado servicio ha finalizado`.
4. Entre el mandato Finalizar depuración (ENDDBG).
5. Entre el mandato Finalizar trabajo de servicio (ENDSRVJOB).

Cuando depura un programa Java, en realidad el programa Java se ejecuta en la máquina virtual Java en un trabajo inmediato por lotes (BCI). El código fuente aparece en la pantalla interactiva, pero el programa Java no se ejecuta en ella. Se ejecuta en el otro trabajo, que es un trabajo al que se da servicio. Vea el tema de la variable de entorno QIBM_CHILD_JOB_SNDINQMSG para obtener más información sobre esta variable, que controla si el trabajo BCI queda en espera antes de llamar a la máquina virtual Java.

Conceptos relacionados

"Compilador Just-In-Time" en la página 418
Un compilador Just-In-Time (JIT) es un compilador específico de plataforma que genera instrucciones de máquina para cada método a medida que son necesarias.

Información relacionada

Depurador de System i5

Pantallas iniciales de depuración de programas Java:

Al depurar los programas Java, siga estas pantallas de ejemplo para sus programas. En ellas aparece un programa de ejemplo llamado Hellod.

- Entre `ADDENVVAR ENVVAR(CLASSPATH) VALUE ('/MIDIR')`.
- Entre este mandato: `RUNJVA CLASS(HELLOD) OPTION(*DEBUG)`. Inserte el nombre del programa Java en lugar de HELLOD.
- Espere a que se visualice la pantalla Visualizar fuente de módulo. Es el fuente del programa Java HELLOD.

```
+------------------------------------------------------------------------------+
|                    Visualizar fuente de módulo                               |
|                                                                              |
|  Nombre archivo clase:   HELLOD                                              |
|       1  import java.lang.*;                                                 |
|       2                                                                      |
|       3  public class Hellod extends Object                                  |
|       4  {                                                                   |
|       5  int k;                                                              |
|       6  int l;                                                              |
|       7  int m;                                                              |
|       8  int n;                                                              |
|       9  int o;                                                              |
|      10  int p;                                                              |
|      11  String myString;                                                    |
|      12  Hellod myHellod;                                                    |
|      13  int myArray[];                                                      |
|      14                                                                      |
|      15  public Hellod()                                                     |
|                                                                    Más...    |
|  Depurar . . .                                                               |
|                                                                              |
|  F3=Fin programa  F6=Añadir/Borrar pto interrup  F10=Recorrer F11=Ver variable |
|  F12=Reanudar  F17=Observar var  F18=Trabajar con pto observ  F24=Más teclas |
|                                                                              |
+------------------------------------------------------------------------------+
```

- Pulse F14 (Trabajar con lista de módulos).
- Aparece la pantalla Trabajar con lista de módulos. Puede añadir otras clases y otros programas para depurar entrando la opción 1 (Añadir programa). Para visualizar el fuente de los módulos, utilice la opción 5 (Visualizar fuente de módulo).

```
+------------------------------------------------------------------------------+
|                          Trabajar con lista de módulos                       |
|                                                  Sistema:   AS400            |
|  Teclee opciones, pulse Intro.                                               |
|    1=Añadir programa   4=Eliminar programa   5=Visualizar fuente de módulo   |
|    8=Trabajar con puntos de interrupción de módulo                           |
|                                                                              |
|  Opc     Progr/módulo       Biblioteca       Tipo                            |
|                             *LIBL            *SRVPGM                         |
|          HELLOD                              *CLASS      Seleccionado        |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
```

```
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                 Final        |
|  Mandato                                                                     |
|  ===>                                                                        |
|  F3=Salir   F4=Solicitud   F5=Renovar   F9=Recuperar   F12=Cancelar          |
|  F22=Visualizar nombre de archivo de clase                                   |
|                                                                              |
+------------------------------------------------------------------------------+
```

- Cuando añada una clase para depurar, puede que necesite entrar un nombre de clase calificado por paquete cuya longitud supere la del campo de entrada Programa/módulo. Para entrar un nombre de mayor longitud, siga estos pasos:
  1. Entre la opción 1 (Añadir programa).
  2. Deje en blanco el campo Programa/módulo.
  3. Deje el campo Biblioteca como *LIBL.
  4. Entre *CLASS en Tipo.
  5. Pulse Intro.
  6. Se visualiza una pantalla emergente, en la que tiene más espacio para especificar el nombre de archivo de clase calificado por paquete. Por ejemplo: `nombrepaquete1.nombrepaquete2.nombreclase`

Establecer puntos de interrupción:

Puede controlar la ejecución de un programa con los puntos de interrupción. Los puntos de interrupción detienen la ejecución de un programa en una sentencia determinada.

Para establecer puntos de interrupción, lleve a cabo los siguientes pasos:

1. Sitúe el cursor en la línea de código en la que desee establecer un punto de interrupción.
2. Pulse F6 (Añadir/Borrar punto de interrupción) para establecer el punto de interrupción.
3. Pulse F12 (Reanudar) para ejecutar el programa.

Nota: Justo antes de que se ejecute la línea de código en la que está establecido el punto de interrupción, se visualiza el fuente del programa para indicar que se ha llegado al punto de interrupción.

```
+------------------------------------------------------------------------------+
|                    Visualizar fuente de módulo                               |
|                                                                              |
|Hebra actual:   00000019     Hebra detenida:   00000019                       |
|Nombre archivo clase:   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();                                                          |
|Depurar . .                                                                   |
|                                                                              |
|F3=Fin programa  F6=Añadir/Borrar pto interrup  F10=Recorrer F11=Ver variable |
```

```
|F12=Reanudar  F17=Observar var  F18=Trabajar con pto observ  F24=Más teclas      |
|Se ha añadido el punto de interrupción a la línea 41.                           |
+--------------------------------------------------------------------------------+
```

Una vez que se haya encontrado un punto de interrupción o que se haya completado un recorrido, puede utilizar el mandato TBREAK para establecer un punto de interrupción que solo sea válido para la hebra actual.

Recorrer paso a paso los programas Java:

Puede recorrer paso a paso el programa mientras lo depura. Puede recorrer la función principal o bien otras funciones del mismo. Los programas Java y los métodos nativos pueden utilizar la función de recorrer.

Cuando aparezca el fuente del programa por primera vez, podrá empezar a recorrer. El programa se detendrá antes de ejecutar la primera sentencia. Pulse F10 (Recorrer). Siga pulsando F10 (Recorrer) para recorrer paso a paso el programa. Pulse F22 (Recorrer todo) para recorrer cualquier función a la que llame el programa. También puede empezar a recorrer siempre que se llegue a un punto de interrupción. Para obtener información sobre cómo establecer puntos de interrupción, vea el tema: Establecer puntos de interrupción.

```
+--------------------------------------------------------------------------------+
|                    Visualizar fuente de módulo                                 |
|                                                                                |
|Hebra actual:   00000019     Hebra detenida:   00000019                         |
|Nombre archivo clase:   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();                                                            |
|Depurar . .                                                                     |
|                                                                                |
|F3=Fin programa  F6=Añadir/Borrar pto interrup  F10=Recorrer F11=Ver variable   |
|F12=Reanudar  F17=Observar var  F18=Trabajar con pto observ  F24=Más teclas     |
|Recorrer completado en la línea 42 de la hebra 00000019                         |
+--------------------------------------------------------------------------------+
```

Para seguir ejecutando el programa, pulse F12 (Reanudar).

Evaluar variables en programas Java:

Existen dos formas de evaluar una variable cuando un programa detiene su ejecución en un punto de interrupción o en una parte del recorrido.

- Opción 1: Entre `EVAL NombreVariable` en la línea de mandatos de depuración.
- Opción 1:Sitúe el cursor en el nombre de la variable dentro del código fuente visualizado y pulse F11 (Visualizar variable).

Nota: Con el mandato EVAL, también se puede cambiar el contenido de una variable. Para obtener más información acerca de las variaciones del mandato EVAL, vea el manual WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712 y la información de ayuda en línea.

Cuando examine las variables de un programa Java, tenga presente lo siguiente:

- Si evalúa una variable que es una instancia de una clase Java, la primera línea de la pantalla muestra el tipo de objeto de que se trata. También muestra el identificador del objeto. A continuación de la primera línea de la pantalla, se visualiza el contenido de cada uno de los campos del objeto. Si la variable es nula, la primera línea de la pantalla indica que lo es. El contenido de cada uno de los campos (de un objeto nulo) se muestra por medio de asteriscos.
- Si evalúa una variable que sea un objeto de tipo serie Java, se visualiza el contenido de la serie. Si la serie es nula, se visualiza null.
- No puede cambiar una variable que sea una serie o un objeto.
- Si evalúa una variable que es una matriz, se visualiza 'ARR' seguido del identificador de la matriz. Para evaluar los elementos de la matriz, puede utilizar un subíndice del nombre de variable. Si la matriz es nula, se visualiza null.
- No se pueden cambiar las variables que son de tipo matriz. Se puede cambiar un elemento de una matriz si no se trata de una matriz de series o de objetos.
- En el caso de las variables de tipo matriz, se puede especificar arrayname.length para ver cuántos elementos hay en la matriz.
- Si desea ver el contenido de una variable que es un campo de una clase, puede especificar classvariable.fieldname.
- Si intenta evaluar una variable antes de que se haya inicializado, pueden ocurrir dos cosas. O bien aparece un mensaje según el cual `la variable no está disponible para visualizarse` o bien se muestra el contenido de la variable no inicializada, que podría ser un valor extraño.

Depurar programas Java y programas de métodos nativos:

Pueden depurarse programas Java y programas de métodos nativos a la vez. Mientras se depura el fuente en la pantalla interactiva, se puede depurar un método nativo programado en C que esté dentro de un programa de servicio (*SRVPGM). El *SRVPGM debe compilarse y crearse con datos de depuración.

Para utilizar la pantalla interactiva del servidor para depurar programas Java y programas de método nativo a la vez, complete los pasos siguientes:

1. Pulse F14 (Trabajar con lista de módulos) cuando aparezca el fuente del programa Java para visualizar la pantalla Trabajar con lista de módulos (WRKMODLST).
2. Seleccione la opción 1 (Añadir programa) para añadir el programa de servicio.
3. Seleccione la opción 5 (Visualizar fuente del módulo) para visualizar el objeto *MODULE que desea depurar y el fuente.
4. Pulse F6 (Añadir/Borrar punto de interrupción), para establecer puntos de interrupción en el programa de servicio. Hallará más información sobre cómo establecer puntos de interrupción en: "Establecer puntos de interrupción" en la página 440.
5. Pulse F12 (Reanudar) para ejecutar el programa.

Nota: Cuando se llega al punto de interrupción en el programa de servicio, se detiene la ejecución del programa y se visualiza el fuente del programa de servicio.

Utilizar la variable de entorno QIBM_CHILD_JOB_SNDINQMSG para la depuración

La variable de entorno QIBM_CHILD_JOB_SNDINQMSG es la que controla si el trabajo inmediato por lotes (BCI), en el que se ejecuta la máquina virtual Java, queda a la espera antes de iniciar la máquina virtual Java.

Si establece que la variable de entorno sea igual a 1 al ejecutar el mandato Ejecutar Java (RUNJVA), se envía un mensaje a la cola de mensajes del usuario. El mensaje se envía antes de que se inicie la máquina virtual Java en el trabajo BCI. Es similar al siguiente:

```
El proceso (hijo) engendrado 023173/JOB/QJVACMDSRV está detenido (G C)
```

Para ver este mensaje, entre SYSREQ y seleccione la opción 4.

El trabajo BCI espera hasta usted entre una respuesta al mensaje. Si la respuesta es (G), se inicia la máquina virtual Java.

Antes de responder al mensaje, puede establecer puntos de interrupción en el programa *SRVPGM o *PGM al que llamará el trabajo BCI.

Nota: Nota: no puede establecer puntos de interrupción en una clase Java porque en este momento todavía no se ha iniciado la máquina virtual Java.

Depurar clases Java cargadas mediante un cargador de clases personalizado

Para utilizar la pantalla interactiva del servidor para depurar una clase cargada mediante un cargador de clases personalizado, lleve a cabo los siguientes pasos.

1. Establezca la variable de entorno DEBUGSOURCEPATH en el directorio que contiene el código fuente o, en el caso de una clase calificada por paquete, en el directorio inicial de los nombres de paquete.

   Por ejemplo, si el cargador de clases personalizadas carga clases ubicadas bajo el directorio /MYDIR, haga lo siguiente:

   ```
   ADDENVVAR ENVVAR(DEBUGSOURCEPATH) VALUE('/MYDIR')
   ```

2. Añada la clase a la vista de depuración desde la pantalla Visualizar fuente de módulo.

   Si la clase ya se ha cargado en la máquina virtual Java (JVM), añada simplemente la *CLASS como es habitual y visualice el código fuente para depurarlo.

   Por ejemplo, para ver el código fuente de pkg1/test14.class, especifique lo siguiente:

   ```
   Opc    Programa/módulo     Biblioteca     Tipo
   1       pkg1.test14_       *LIBL          *CLASS
   ```

   Si la clase no se ha cargado en la JVM, realice los mismo pasos para añadir la *CLASS como se ha indicado anteriormente. Entonces se visualizará el mensaje **Archivo de clase Java no disponible**. En este punto, puede reanudar el proceso del programa. La JVM se detiene automáticamente cuando se especifica cualquier método de la clase que coincide con el nombre dado. El código fuente de la clase se visualiza y puede depurarse.

Depurar servlets

La depuración de servlets es un caso especial de las clases de depuración cargadas mediante un cargador de clases personalizadas. Los servlets se ejecutan en la ejecución Java de IBM HTTP Server. Tiene varias opciones para depurar servlets.

Puede depurar servlets siguiendo las instrucciones de clases cargadas mediante un cargador de clases personalizado.

También puede utilizar la pantalla interactiva del servidor para depurar un servlet completando los pasos siguientes:

1. Utilice el mandato `javac -g` del intérprete Qshell para compilar el servlet.
2. Copie el código fuente (archivo .java) y el código compilado (archivo .class) en /QIBM/ProdData/Java400.
3. Ejecute el mandato Crear programa Java (CRTJVAPGM) en el archivo .class utilizando el nivel de optimización 10, OPTIMIZE(10).
4. Inicie el servidor.
5. Ejecute el mandato Arrancar trabajo de servicio (STRSRVJOB) en el trabajo donde se ejecuta el servlet.
6. Entre STRDBG CLASS(myServlet), siendo myServlet el nombre del servlet. Debe visualizarse el fuente.
7. Establezca un punto de interrupción en el servlet y pulse F12.
8. Ejecute el servlet. Cuando el servlet alcance el punto de interrupción, puede continuar depurando.

Otra manera de depurar programas y servlets Java que se ejecuten en su sistema consiste en utilizar el depurador de IBM System i5. El depurador de System i5 proporciona una interfaz gráfica de usuario que le permite utilizar con mayor facilidad las prestaciones de depuración del sistema.

Información relacionada

Depurador de System i5

Arquitectura de depuradores de la plataforma Java

La arquitectura de depuradores de la plataforma Java (JPDA) consta de la interfaz de depuración de JVM/interfaz de herramientas de JVM, del protocolo Java Debug Wire Protocol y de la interfaz de depuración Java. Todos estos componentes de JPDA permiten a cualquier componente frontal de un depurador que utilice JDWP realizar operaciones de depuración. El componente frontal del depurador se puede ejecutar remotamente o como aplicación de System i5.

Interfaz de herramientas de la máquina virtual Java (JVMTI)

JVMTI sustituye a la interfaz de depuración de la máquina virtual Java (JVMDI) y a la interfaz de perfilador de la máquina virtual Java (JVMPI). JVMTI tiene la misma funcionalidad que ambas interfaces JVMDI y JVMPI, además de otras funciones. JVMTI se añadió como parte de J2SE 5.0. En JDK 6, las interfaces JVMDI y JVMPI han dejado de ofrecerse, y JVMTI es la única opción disponible.

Para obtener más información sobre cómo utilizar JVMTI, vea la página de consulta de JVMTI, en el sitio Web de Sun Microsystems, Inc.

Interfaz de depuración de la máquina virtual Java (solo en JDK 1.4)

En Java 2 SDK (J2SDK), Standard Edition, JVMDI forma parte de las interfaces de programación de aplicaciones (API) de la plataforma Sun Microsystems, Inc. JVMDI permite escribir un depurador Java para un System i5 en código C. No hace falta que el depurador conozca la estructura interna de la máquina virtual Java, porque utiliza interfaces JVMDI. JVMDI es la interfaz de nivel más bajo de JPDA, que se encuentra más cerca de la máquina virtual Java.

Protocolo JDWP (Java Debug Wire Protocol)

El protocolo JDWP (Java Debug Wire Protocol) es un protocolo de comunicaciones definido entre un proceso del depurador y la interfaz JVMDI/JVMTI. JDWP puede utilizarse desde un sistema remoto o a través de un socket local. Está a a una capa de distancia de la JVMDI/JVMTI.

Iniciar JDWP en QShell

Para iniciar JDWP y ejecutar la clase Java SomeClass, entre el siguiente mandato en QShell:

```
java -interpret -agentlib:jdwp=transport=dt_socket,
  address=8000,server=y,suspend=n SomeClass
```

En este ejemplo, JDWP está a la escucha de las conexiones de depuradores remotos en el puerto TCP/IP 8000, pero puede utilizar cualquier número de puerto que desee; dt_socket es el nombre del SRVPGM que maneja el transporte de JDWP, y no cambia.

Si desea saber qué opciones adicionales puede utilizar con -agentlib, vea Sun VM Invocation Options, de Sun Microsystems, Inc. Estas opciones están disponibles para ambos JDK, 1.4 y 1.5, en i5/OS.

Iniciar JDWP desde una línea de mandatos CL

Para iniciar JDWP con el mandato CL, se han proporcionado dos opciones nuevas: AGTPGM y AGTOPTIONS.

El valor de AGTPM es JDWP, y el valor de AGTOPTIONS se puede definir para que sea la misma serie que se utilizaría en la línea de mandatos de QShell.

Para iniciar JDWP y ejecutar la clase Java SomeClass, entre el mandato:

```
JAVA CLASS(SomeClass) INTERPRET(*YES) AGTPGM(JDWP)
AGTOPTIONS('transport=dt_socket,address=8000,server=y,suspend=n')
```

Interfaz de depuración Java

La interfaz de depuración Java (JDI) es una interfaz del lenguaje Java de alto nivel proporcionada para el desarrollo de heramientas. JDI oculta la complejidad de JVMDI/JVMTI y JDWP detrás de algunas definiciones de clases Java. JDI está incluida en el archivo rt.jar y, por lo tanto, el componente frontal del depurador existe en cualquier plataforma que tenga instalado Java.

Si desea escribir depuradores para Java, debe utilizar JDI, ya que es la interfaz más sencilla y el código es independiente de la plataforma.

Para obtener más información sobre JDPA, vea Java Platform Debugger Architecture Overview, de Sun Microsystems, Inc.

Depuración a máxima velocidad en la JVM:

La máquina virtual i5/OS Java (JVM) admite la depuración a máxima velocidad.

La depuración a máxima velocidad le permite ejecutar la aplicación con todas las ventajas de rendimiento del código compilado de JIT, sin perder la capacidad de realizar algunas de las actividades de depuración más comunes como, por ejemplo, establecer puntos de interrupción, recorrer el código y visualizar variables locales.

Dado que la depuración a máxima velocidad permite que los métodos se compilen con JIT, existen algunas limitaciones para la depuración:

- Las operaciones de recorrer en las sentencias de retorno no funcionan si el llamante es código compilado.
- Los puntos de observación solamente se activan en los métodos no compilados que modifican el campo observado.
- Las variables locales no se pueden visualizar para los métodos compilados por JIT.
- No está permitido redifinir una clase.

Nota: Esta característica solamente está soportada para los depuradores que utilizan el protocolo JDWP (Java Debug Wire Protocol) para realizar operaciones de depuración. Actualmente, el depurador del sistema no da soporte a la depuración a máxima velocidad.

Localizar fugas de memoria

Si el rendimiento del programa se degrada al ejecutarse durante mucho más tiempo, es posible que haya codificado una fuga de memoria erróneamente. Puede utilizar las herramientas de análisis de memoria dinámica (HAT) para Java para ayudarle a depurar el programa y localizar las fugas de memoria, realizando análisis de memoria dinámica de aplicaciones y perfilado de creación de objetos Java a lo largo del tiempo.

Encontrará más detalles en: Herramientas de análisis de memoria dinámica (HAT) para Java.

También puede utilizar el mandato de lenguaje de control Analizar máquina virtual Java (ANZJVM) para buscar fugas de objetos. ANZJVM busca fugas de objeto tomando dos copias del almacenamiento dinámico de recogida de basura separadas por un intervalo de tiempo especificado. Para buscar fugas de

objeto, observará el número de instancias de cada clase que figuran en el almacenamiento dinámico. Las clases que tienen un número de instancias inusitadamente alto deben considerarse como posibles fugas.

También debe observar el cambio en el número de instancias de cada clase entre las dos copias del almacenamiento dinámico de recogida de basura. Si el número de instancias de una clase aumenta continuamente, dicha clase debe considerarse como posible fuga. Cuanto mayor sea el intervalo de tiempo entre las dos copias, más posibilidades hay de que realmente existan fugas en los objetos. Ejecutando ANZJVM una serie de veces con un intervalo de tiempo mayor, debe ser capaz de diagnosticar las fugas con un mayor grado de certeza.

Utilizar el mandato Generar vuelco de JVM

Si utiliza la máquina virtual de tecnología IBM para Java, puede emplear el mandato CL Generar vuelco de JVM (GENJVMDMP) para generar vuelcos Java, del sistema y de la memoria dinámica.

El mandato GENJVMDMP genera vuelcos de la máquina virtual Java (JVM) a petición. Solo se visualizará información relacionada con los trabajos de la máquina virtual de tecnología IBM para Java. Los tipos de vuelcos que puede generar son los siguientes:

***JAVA** Genera múltiples archivos que contienen información de diagnóstico para la JVM y las aplicaciones Java que se ejecutan en la JVM.

***SYSTEM**
Genera la imagen de memoria en bruto con formato binario del trabajo que se ejecutaba en el momento de iniciarse el vuelco.

***HEAP**
Genera un vuelco de toda las asignaciones de espacio de la memoria dinámica que todavía no se hayan liberado.

Información relacionada

Mandato CL Generar vuelco de JVM (GENJVMDMP)

Ejemplos de código para IBM Developer Kit para Java

A continuación se ofrece una lista de ejemplos de código para IBM Developer Kit para Java.

Internacionalización

- "Ejemplo: internacionalización de las fechas con la clase java.util.DateFormat" en la página 449
- "Ejemplo: internacionalización de las presentaciones numéricas con la clase java.util.NumberFormat" en la página 449
- "Ejemplo: internacionalización de los datos específicos de entorno nacional con la clase java.util.ResourceBundle" en la página 450

JDBC

- "Ejemplo: propiedad Access" en la página 451
- "Ejemplo: BLOB" en la página 144
- "Ejemplo: interfaz CallableStatement de IBM Developer Kit para Java" en la página 455
- "Ejemplo: eliminar valores de una tabla mediante el cursor de otra sentencia" en la página 125
- "Ejemplo: CLOB" en la página 148
- "Ejemplo: crear un UDBDataSource y enlazarlo con JNDI" en la página 59
- "Ejemplo: crear un UDBDataSource y obtener un ID de usuario y una contraseña" en la página 61
- "Ejemplo: crear un UDBDataSourceBind y establecer las propiedades de DataSource" en la página 60
- "Ejemplo: devolver una lista de tablas utilizando la interfaz DatabaseMetaData de using the IBM Developer Kit para Java" en la página 70
- "Ejemplo: crear un UDBDataSource y enlazarlo con JNDI" en la página 59

- "Ejemplo: programa servidor IBM JGSS habilitado para JAAS" en la página 548

Extensión de sockets seguros Java (JSSE)
- "Ejemplos: IBM Java Secure Sockets Extension 1.4" en la página 298

Java con otros lenguajes de programación
- "Ejemplo: llamar a un programa CL con java.lang.Runtime.exec()" en la página 232
- "Ejemplo: llamar a un mandato CL con java.lang.Runtime.exec()" en la página 233
- "Ejemplo: llamar a otros programa Java con java.lang.Runtime.exec()" en la página 232
- "Ejemplo: llamar a Java desde C" en la página 239
- "Ejemplo: llamar a Java desde RPG" en la página 239
- "Ejemplo: utilizar corrientes de entrada y de salida para la comunicación entre procesos" en la página 238
- "Ejemplo: API de invocación Java" en la página 215
- "Ejemplo: método nativo IBM i5/OS PASE para Java" en la página 227
- Sockets
- "Ejemplos: utilizar la interfaz Java nativa (JNI) para métodos nativos" en la página 561

SQLJ
- "Ejemplo: intercalar sentencias SQL en la aplicación Java" en la página 187

SSL
- "Ejemplos: cambiar el código Java para que utilice fábricas de sockets de cliente" en la página 278
- "Ejemplos: cambiar el código Java para que utilice fábricas de sockets de servidor" en la página 276
- "Ejemplos: cambiar el cliente Java para que utilice la capa de sockets segura (SSL)" en la página 282
- "Ejemplos: cambiar el servidor Java para que utilice la capa de sockets segura (SSL)" en la página 280

Ejemplo: internacionalización de las fechas con la clase java.util.DateFormat

Este ejemplo muestra cómo pueden utilizarse los entornos nacionales para formatear las fechas.

Ejemplo 1: enseña a utilizar la clase java.util.DateFormat para internacionalizar las fechas

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//*************************
// Archivo: DateExample.java
//*************************

import java.text.*;
import java.util.*;
import java.util.Date;

public class DateExample {

   public static void main(String args[]) {

     // Obtener la fecha
     Date now = new Date();

     // Obtener los formateadores de fecha de los entornos nacionales
     // predeterminados, alemán y francés
     DateFormat theDate = DateFormat.getDateInstance(DateFormat.LONG);
     DateFormat germanDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.GERMANY);
     DateFormat frenchDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);

     // Formatear e imprimir las fechas
     System.out.println("Fecha en el entorno nacional predeterminado: " + theDate.format(now));
     System.out.println("Fecha en el entorno nacional alemán: " + germanDate.format(now));
     System.out.println("Fecha en el entorno nacional francés: " + frenchDate.format(now));
   }
}
```

"Ejemplos: crear un programa Java internacionalizado" en la página 33

Si necesita personalizar un programa Java(TM) para una región concreta del mundo, puede crear un programa Java internacionalizado con los Entornos nacionales Java.

Ejemplo: internacionalización de las presentaciones numéricas con la clase java.util.NumberFormat

Este ejemplo muestra cómo pueden utilizarse los entornos nacionales para formatear los números.

Ejemplo 1: enseña a utilizar la clase java.util.NumberFormat para internacionalizar la salida numérica.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//*************************
// Archivo: NumberExample.java
//*************************

 import java.lang.*;
 import java.text.*;
 import java.util.*;

 public class NumberExample {

     public static void main(String args[]) throws NumberFormatException {

       // El número que debe formatearse
```

```
       double number = 12345.678;

       // Obtener formateadores de los entornos nacionales
       // predeterminado, español y japonés
       NumberFormat defaultFormat = NumberFormat.getInstance();
       NumberFormat spanishFormat = NumberFormat.getInstance(new
 Locale("es", "ES"));
       NumberFormat japaneseFormat = NumberFormat.getInstance(Locale.JAPAN);

       // Imprimir número con los formatos predeterminado, español y japonés
       // (Nota: NumberFormat no es necesario para el formato predeterminado)
       System.out.println("El número formateado para el entorno nacional predeterminado; " +
                          defaultFormat.format(number));
       System.out.println("El número formateado para el entorno nacional español; " +
                          spanishFormat.format(number));
       System.out.println("El número formateado para el entorno nacional japonés; " +
                          japaneseFormat.format(number));
     }
 }
```

"Ejemplos: crear un programa Java internacionalizado" en la página 33
Si necesita personalizar un programa Java(TM) para una región concreta del mundo, puede crear un programa Java internacionalizado con los Entornos nacionales Java.

Ejemplo: internacionalización de los datos específicos de entorno nacional con la clase java.util.ResourceBundle

Este ejemplo muestra cómo pueden utilizarse los entornos nacionales junto con paquetes de recursos para internacionalizar las series del programa.

Para que el programa ResourceBundleExample funcione como se pretende, se necesitan los archivos de propiedades siguientes:

Contenido de RBExample.properties
Hello.text=Hello

Contenido de RBExample_de.properties
Hello.text=Guten Tag

Contenido de RBExample_fr_FR.properties
Hello.text=Bonjour

Ejemplo 1: enseña a utilizar la clase java.util.ResourceBundle para internacionalizar los datos específicos de entorno nacional

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//*********************************
// Archivo: ResourceBundleExample.java
//*********************************

import java.util.*;

public class ResourceBundleExample {
   public static void main(String args[]) throws MissingResourceException {

     String resourceName = "RBExample";
     ResourceBundle rb;

     // Entorno nacional predeterminado
     rb = ResourceBundle.getBundle(resourceName);
     System.out.println("Predeterminado: " + rb.getString("Hello" + ".text"));

     // Solicitar un paquete de recursos con un entorno nacional
```

```
     // especificado de manera explícita
     rb = ResourceBundle.getBundle(resourceName, Locale.GERMANY);
     System.out.println("Alemán: " + rb.getString("Hello" + ".text"));

     // No existe ningún archivo de propiedades para China en este
     // ejemplo... se utiliza el valor predeterminado
     rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);
     System.out.println("Chino: " + rb.getString("Hello" + ".text"));

     // He aquí otra manera de hacerlo...
     Locale.setDefault(Locale.FRANCE);
     rb = ResourceBundle.getBundle(resourceName);
     System.out.println("Francés: " + rb.getString("Hello" + ".text"));

     // No existe ningún archivo de propiedades para China en este
     // ejemplo... utilizar el predeterminado, que ahora es fr_FR.
     rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);
     System.out.println("Chino: " + rb.getString("Hello" + ".text"));
   }
}
```

"Ejemplos: crear un programa Java internacionalizado" en la página 33
Si necesita personalizar un programa Java(TM) para una región concreta del mundo, puede crear un programa Java internacionalizado con los Entornos nacionales Java.

Ejemplo: propiedad Access

Este es un ejemplo de cómo utilizar la propiedad Java Access.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Este programa presupone que el directorio cujosql existe.
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();
    }


/**
Configurar el DataSource utilizado en la prueba.
**/
    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) { // Ignorarla - no existe
        }
```

```
    try {
        String sql = "CREATE PROCEDURE CUJOSQL.TEMP "
              + " LANGUAGE SQL SPECIFIC CUJOSQL.TEMP "
              + " MYPROC: BEGIN"
              + "   RETURN 11;"
              + " END MYPROC";
        s.executeUpdate(sql);
    } catch (SQLException e) {
        // Ignorarla - existe.
    }
    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("Excepción: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignorarla.
            }
        }
    }
}


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("Excepción: SQLState(" +
        //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignorarla.
            }
        }
    }
}


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("Excepción: SQLState(" +
        //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignorarla.
            }
        }
    }
}


public void run()
throws SQLException
{
    System.out.println("Establecer que la propiedad de acceso de conexión sea solo de lectura");
    resetConnection("read only");

    System.out.println("Puede ejecutar consultas  -->" + canQuery());
    System.out.println("Puede ejecutar actualizaciones  -->" + canUpdate());
    System.out.println("Puede ejecutar llamadas sp -->" + canCall());

    System.out.println("Establecer que la propiedad de acceso de conexión sea llamada de lectura");
    resetConnection("read call");

    System.out.println("Puede ejecutar consultas  -->" + canQuery());
    System.out.println("Puede ejecutar actualizaciones  -->" + canUpdate());
    System.out.println("Puede ejecutar llamadas sp -->" + canCall());

    System.out.println("Establecer que la propiedad de acceso de conexión sea todo");
    resetConnection("all");

    System.out.println("Puede ejecutar consultas  -->" + canQuery());
    System.out.println("Puede ejecutar actualizaciones  -->" + canUpdate());
    System.out.println("Puede ejecutar llamadas sp -->" + canCall());
```

```
    }


    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            // Ignorarla.
        }
    }
}
```

Ejemplo: BLOB

Este es un ejemplo de cómo puede colocarse un BLOB en la base de datos o recuperarse de la misma.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// PutGetBlobs es una aplicación de ejemplo
// que muestra cómo trabajar con la API de JDBC
// para colocar objetos BLOB en columnas de base
// de datos y obtenerlos desde ellas.
//
// Los resultados de la ejecución de este programa
// provocan la inserción de dos valores de BLOB
// en una tabla nueva. Ambos son idénticos
// y contienen 500k de datos de byte
// aleatorios.
////////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      // Establecer una conexión y una sentencia con las que trabajar.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Borrar ejecuciones anteriores de esta aplicación.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
      } catch (SQLException e) {
          // Ignorarlo y suponer que la tabla no existía.
      }

      // Crear una tabla con una columna BLOB. El tamaño de columna BLOB
      // predeterminado es 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

      // Crear un objeto PreparedStatement que permita colocar
      // un nuevo objeto Blob en la base de datos.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

      // Crear un valor BLOB grande...
      Random random = new Random ();
```

```
      byte [] inByteArray = new byte[500000];
      random.nextBytes (inByteArray);

      // Establecer el parámetro PreparedStatement. Nota: esto no es
      // portable a todos los controladores JDBC. Estos no tienen
      // soporte al utilizar setBytes para columnas BLOB. Esto se usa para
      // permitirle generar nuevos BLOB. También permite a
      // los controladores JDBC 1.0 trabajar con columnas que contengan datos BLOB.
      ps.setBytes(1, inByteArray);

      // Procesar la sentencia, insertando el BLOB en la base de datos.
      ps.executeUpdate();

      // Procesar una consulta y obtener el BLOB que se acaba de insertar
      // a partir de la base de datos como objeto Blob.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
      rs.next();
      Blob blob = rs.getBlob(1);


      // Colocar de nuevo ese Blob en la base de datos mediante
      // la PreparedStatement.
      ps.setBlob(1, blob);
      ps.execute();

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: interfaz CallableStatement de IBM Developer Kit para Java

Este es un ejemplo de cómo utilizar la interfaz CallableStatement.

Ejemplo: interfaz CallableStatement

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

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

// Crear el objeto CallableStatement.
// Este precompila la llamada especificada a un procedimiento almacenado.
// Los signos de interrogación señalan el lugar en que deben establecerse
// los parámetros de entrada y  el lugar en que deben recuperarse los
// parámetros de salida.
// Los dos primeros parámetros son de entrada y el tercero es de salida.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

// Establecer los parámetros de entrada.
cs.setInt (1, 123);
cs.setInt (2, 234);

// Registrar el tipo del parámetro de salida.
cs.registerOutParameter (3, Types.INTEGER);

// Ejecutar el procedimiento almacenado.
cs.execute ();

// Obtener el valor del parámetro de salida.
int sum = cs.getInt (3);

// Cerrar CallableStatement y la conexión.
cs.close();
c.close();
```

"CallableStatements" en la página 106
La interfaz CallableStatement de JDBC amplía PreparedStatement y proporciona soporte para parámetros de salida y de entrada/salida. La interfaz CallableStatement tiene también soporte para parámetros de entrada, que proporciona la interfaz PreparedStatement.

Ejemplo: eliminar valores de una tabla mediante el cursor de otra sentencia

Este ejemplo Java enseña a eliminar valores de una tabla mediante el cursor de otra sentencia.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }



/**
Manejar todo el trabajo de configuración necesario.
**/
    public void setup() {
        try {
            // Registrar el controlador 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) {
                // Aquí, pasar por alto los problemas.
            }

            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("Excepción capturada: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
En esta sección, debe añadirse todo el código destinado a realizar
```

```
la prueba. Si solo se necesita una conexión con la base de datos,
se puede utilizar la variable global 'connection'.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Actualizar cada valor utilizando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("El nombre de cursor es " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement
                                      ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
                                      rs.getCursorName ());


            // Repetir en bucle el ResultSet y actualizar las demás entradas.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Borrar los recursos una vez utilizados.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
En esta sección, colocar todo el trabajo de borrado para la prueba.
**/
    public void cleanup() {
        try {
            // Cerrar la conexión global abierta en setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
Visualizar el contenido de la tabla.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Índice " + rs.getInt(1) + " valor " + rs.getString(2));
            }
```

```
            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Ejemplo: CLOB

Este es un ejemplo de cómo puede colocarse un CLOB en la base de datos o recuperarse de la misma.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////
// PutGetClobs es una aplicación de ejemplo
// que muestra cómo trabajar con la API de JDBC
// para colocar objetos CLOB en columnas de base
// de datos y obtenerlos desde ellas.
//
// Los resultados de la ejecución de este programa
// son que existan dos valores de CLOB
// en una tabla nueva. Ambos son idénticos
// y contienen alrededor de 500k de datos
// de texto de repetición.
//////////////////////////////////////
import java.sql.*;

public class PutGetClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      // Establecer una conexión y una sentencia con las que trabajar.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Borrar ejecuciones anteriores de esta aplicación.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
      } catch (SQLException e) {
          // Ignorarlo y suponer que la tabla no existía.
      }

      // Crear una tabla con una columna CLOB. El tamaño de columna CLOB
      // predeterminado es 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

      // Crear un objeto PreparedStatement que permita colocar
      // un nuevo objeto Clob en la base de datos.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

      // Crear un valor 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();

      // Establecer el parámetro PreparedStatement. Esto no es
      // portable a todos los controladores JDBC. Estos no tienen
      // que soportar setBytes para columnas CLOB. Esto se hace para
      // permitirle que genere nuevos CLOB. También
      // permite a los controladores JDBC 1.0 trabajar con columnas que contengan
      // datos Clob.
      ps.setString(1, clobValue);

      // Procesar la sentencia, insertando el clob en la base de datos.
      ps.executeUpdate();

      // Procesar una consulta y obtener el CLOB que se acaba de insertar
      // a partir de la base de datos como objeto Clob.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
      rs.next();
      Clob clob = rs.getClob (1);


      // Colocar de nuevo ese Clob en la base de datos mediante
      // la PreparedStatement.
      ps.setClob(1, clob);
      ps.execute();

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: crear un UDBDataSource y enlazarlo con JNDI

Este es un ejemplo de cómo crear un UDBDataSource y enlazarlo con JNDI.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Importar los paquetes necesarios. En el momento del despliegue,
// la clase específica del controlador JDBC que implementa
// DataSource se tiene que importar.
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
    {
        // Crear un nuevo objeto UDBDataSource y proporcionarle
        // una descripción.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("Un UDBDataSource simple");

        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Enlazar el objeto UDBDataSource recién creado
        // con el servicio de directorios JNDI, dándole un nombre
        // que pueda utilizarse para buscar de nuevo este objeto
        // más adelante.
        ctx.rebind("SimpleDS", ds);
    }
}
```

Ejemplo: crear un UDBDataSource y obtener un ID de usuario y una contraseña

Este es un ejemplo de cómo crear un UDBDataSource y utilizar el método getConnection para obtener un ID de usuario y una contraseña durante la ejecución.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/// Importar los paquetes necesarios.
// No hay código específico del controlador que se necesite en las
// aplicaciones de tiempo de ejecución.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Recuperar el objeto UDBDataSource enlazado mediante el
        // nombre con el que estaba enlazado anteriormente. En tiempo de ejecución,
        // solo se utiliza la interfaz DataSource y, por lo tanto,
        // no hace falta convertir el objeto a la clase de
        // implementación de UDBDataSource. (No hace falta saber
        // cuál es la clase de implementación. El nombre JNDI lógico
        // es lo único necesario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una vez obtenido el DataSource, puede utilizarse para establecer
        // una conexión. El perfil de usuario cujo y la contraseña newtiger
        // se utilizan para crear la conexión en lugar del ID de usuario
        // y la contraseña por omisión para el DataSource.
        Connection connection = ds.getConnection("cujo", "newtiger");

        // La conexión puede utilizarse para crear objetos Statement y
        // actualizar la base de datos o procesar consultas de la forma siguiente.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // La conexión se cierra antes de que finalice la aplicación.
        connection.close();
    }
}
```

Ejemplo: crear un UDBDataSourceBind y establecer las propiedades de DataSource

Este es un ejemplo de cómo crear un UDBDataSource y establecer el ID de usuario y la contraseña como propiedades de DataSource.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Importar los paquetes necesarios. En el momento del despliegue,
// la clase específica del controlador JDBC que implementa
// DataSource se tiene que importar.
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
    {
        // Crear un nuevo objeto UDBDataSource y proporcionarle
        // una descripción.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("Un UDBDataSource simple"  +
                          "con cujo como perfil por" +
                          "defecto al que conectarse.");

        // Proporcionar un ID de usuario y una contraseña que se deban usar para
        // las propiedades de conexión.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Enlazar el objeto UDBDataSource recién creado
        // con el servicio de directorios JNDI, dándole un nombre
        // que pueda utilizarse para buscar de nuevo este objeto
        // más adelante.
        ctx.rebind("SimpleDS2", ds);
    }
}
```

Ejemplo: devolver una lista de tablas utilizando la interfaz DatabaseMetaData de using the IBM Developer Kit para Java

Este ejemplo muestra cómo devolver una lista de tablas.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

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

// Obtener los metadatos de base de datos de la conexión.
DatabaseMetaData dbMeta = c.getMetaData();

// Obtener una lista de tablas que cumplen estos criterios.
String catalog = "myCatalog";
String schema = "mySchema";
String table  = "myTable%"; // % indica el patrón de búsqueda
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"}:
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterar por ResultSet para obtener los valores.

// Cerrar la conexión.
c.close():
```

Ejemplo: Datalink

En esta aplicación de ejemplo se enseña a utilizar la API de JDBC para manejar columnas de base de datos de tipo datalink.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// PutGetDatalinks es una aplicación de ejemplo
// que muestra cómo utilizar la API de JDBC
// para manejar columnas de base de datos de tipo datalink.
////////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrar el controlador JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Error de configuración.
        }

        // Establecer una conexión y una sentencia con las que trabajar.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Borrar ejecuciones anteriores de esta aplicación.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
        } catch (SQLException e) {
            // Ignorarlo y suponer que la tabla no existía.
        }

        // Crear una tabla con una columna datalink.
        s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

        // Crear un objeto PreparedStatement que permita añadir
        // un nuevo objeto datalink en la base de datos. Dado que la conversión
        // a un datalink no puede realizarse directamente en la base de datos,
        // puede codificar la sentencia SQL para realizar la conversión explícita.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
                                                   VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

        // Establecer el datalink. Este URL señala hacia un tema sobre
        // las nuevas características de JDBC 3.0.
        ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

        // Procesar la sentencia, insertando el CLOB en la base de datos.
        ps.executeUpdate();

        // Procesar una consulta y obtener el CLOB que se acaba de insertar
        // a partir de la base de datos como objeto Clob.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        String datalink = rs.getString(1);


        // Colocar ese valor de datalink en la base de datos mediante
        // la PreparedStatement. Nota: para esta función se necesita
        // soporte de JDBC 3.0.
        /*
```

```
        try {
            URL url = new URL(datalink);
            ps.setURL(1, url);
            ps.execute();
        } catch (MalformedURLException mue) {
            // Manejar aquí este problema.
        }

        rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        URL url = rs.getURL(1);
        System.out.println("el valor de URL es " + url);
        */

        c.close(); // El cierre de la conexión también cierra stmt y rs.
    }
}
```

Ejemplo: tipos distinct

Este es un ejemplo de utilización de tipos distinct.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// Este programa de ejemplo muestra ejemplos de
// diversas tareas comunes que pueden realizarse
// con tipos distinct.
////////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
    throws SQLException
    {
        // Registrar el controlador JDBC nativo.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Error de configuración.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Borrar ejecuciones antiguas.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignorarlo y suponer que la tabla no existía.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
        } catch (SQLException e) {
            // Ignorarlo y suponer que la tabla no existía.
        }

        // Crear el tipo, crear la tabla e insertar un valor.
        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();
```

```
        // Puede obtener detalles sobre los tipos disponibles con nuevos metadatos en
        // 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("Nombre de tipo " + rs.getString(3) +
                           " tiene el tipo " + rs.getString(4));


        // Acceder a los datos que ha insertado.
        rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
        rs.next();
        System.out.println("SSN es " + rs.getString(1));

        c.close(); // El cierre de la conexión también cierra stmt y rs.
    }
}
```

Ejemplo: intercalar sentencias SQL en la aplicación Java

La siguiente aplicación SQLJ de ejemplo, App.sqlj, utiliza SQL estático para recuperar y actualizar datos de la tabla EMPLOYEE de la base de datos de ejemplo DB2.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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
{

  /**********************
   **  Controlador de registro **
   **********************/

 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;

// El URL es jdbc:db2:nombredb
String url = "jdbc:db2:sample";

DefaultContext ctx = DefaultContext.getDefaultContext();
if (ctx == null)
{
  try
  {
    // Conectar con id/contraseña predeterminados.
    Connection con = DriverManager.getConnection(url);
    con.setAutoCommit(false);
    ctx = new DefaultContext(con);
  }
  catch (SQLException e)
  {
    System.out.println("Error: no se ha podido obtener un contexto predeterminado");
    System.err.println(e) ;
    System.exit(1);
  }
  DefaultContext.setDefaultContext(ctx);
}

// Recuperar datos de la base de datos.
System.out.println("Recuperar algunos datos de la base de datos.");
#sql cursor1 = {SELECT empno, firstnme FROM employee}; // [2]

// Visualizar el conjunto de resultados.
// cursor1.next() devuelve false cuando no hay más filas.
System.out.println("Resultados recibidos:");
while (cursor1.next()) // [3]
{
  str1 = cursor1.empno(); // [4]
  str2 = cursor1.firstnme();

  System.out.print (" empno= " + str1);
  System.out.print (" firstname= " + str2);
  System.out.println("");
}
cursor1.close(); // [9]

// Recuperar el número de empleado de la base de datos.
#sql { SELECT count(*) into :count1 FROM employee }; // [5]
if (1 == count1)
  System.out.println ("Hay una fila en la tabla de empleados");
else
  System.out.println ("Hay " + count1
                      + " filas en la tabla de empleados");

// Actualizar la base de datos.
System.out.println("Actualizar la base de datos.");
#sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

// Recuperar los datos actualizados de la base de datos.
System.out.println("Recuperar los datos actualizados de la base de datos.");
str1 = "000010";
#sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // [6]

// Visualizar el conjunto de resultados.
// cursor2.next() devuelve false cuando no hay más filas.
System.out.println("Resultados recibidos:");
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

    // Retrotraer la actualización.
    System.out.println("Retrotraer la actualización.");
    #sql { ROLLBACK work };
    System.out.println("Retrotracción terminada.");
  }
  catch( Exception e )
  {
    e.printStackTrace();
  }
 }
}
```

Declarar iteradores. Esta sección declara dos tipos de iteradores:

- App_Cursor1: Declara nombres y tipos de datos de columna, y devuelve los valores de las columnas de acuerdo con el nombre de columna (enlace por nombre con columnas).
- App_Cursor2: Declara tipos de datos de columna, y devuelve los valores de las columnas por posición de columna (enlace posicional con columnas).

Inicializar el iterador. El objeto iterador cursor1 se inicializa utilizando el resultado de una consulta. La consulta almacena el resultado en cursor1.

Adelantar el iterador a la próxima fila. El método cursor1.next() devuelve un Boolean `false` si no existen más filas para recuperar.

[4]Mover los datos. El método de acceso por nombre empno() devuelve el valor de la columna denominada empno en la fila actual. El método de acceso por nombre firstnme() devuelve el valor de la columna denominada firstnme en la fila actual.

[5]Aplicar SELECT a los datos de una variable del lenguaje principal. La sentencia SELECT pasa el número de filas de la tabla a la variable de lenguaje principal count1.

[6] Inicializar el iterador. El objeto iterador cursor2 se inicializa utilizando el resultado de una consulta. La consulta almacena el resultado en cursor2.

[7]Recuperar los datos. La sentencia FETCH devuelve el valor actual de la primera columna declarada en el cursor ByPos desde la tabla de resultados a la variable de lenguaje principal str2.

[8]Comprobar el éxitode una sentencia FETCH.INTO. El método endFetch() devuelve Boolean true si el iterador no está situado en una fila, es decir, si el último intento de captar una fila ha fallado. El método endFetch() devuelve false si el último intento de captar una fila ha sido satisfactorio. DB2 intenta captar una fila cuando se llama al método next(). Una sentencia FETCH...INTO llama implícitamente al método next().

[9]Cerrar los iteradores. El método close() libera los recursos retenidos por los iteradores. Los iteradores se deben cerrar explícitamente para asegurar que se liberan los recursos del sistema de forma oportuna.

Ejemplo: finalizar una transacción

Este es un ejemplo de cómo finalizar una transacción en la aplicación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
            }

            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();
            }
        }
    }


    /**
     * Esta prueba utiliza el soporte JTA para manejar transacciones.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Presuponer que el origen de datos se apoya en un UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // Desde el DataSource, obtener un objeto XAConnection que
            // contiene un XAResource y un objeto Connection.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
```

```
            Connection    c      = xaConn.getConnection();

            // Para transacciones XA, es necesario un identificador de transacción.
            // No se incluye una implementación de la interfaz XID con
            // el controlador JDBC. Vea: Transacciones con JTA
            // para obtener una descripción de esta interfaz para crear una clase para ella.
            Xid xid = new XidImpl();

            // La conexión de XAResource puede usarse como cualquier otra
            // conexión JDBC.
            Statement stmt = c.createStatement();

            // Hay que notificar al recurso XA antes de iniciar cualquier
            // trabajo transaccional.
            xaRes.start(xid, XAResource.TMNOFLAGS);

            // Crear un ResultSet durante el proceso de JDBC y captar una fila.
            ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
            rs.next();

            // Cuando se llama al método end, se cierran todos los cursores de ResultSet.
            // El intento de acceder a ResultSet después de este punto provoca
            // el lanzamiento de una excepción.
            xaRes.end(xid, XAResource.TMNOFLAGS);

            try {
                String value = rs.getString(1);
                System.out.println("Algo ha fallado si recibe ese mensaje.");
            } catch (SQLException e) {
                System.out.println("Se ha lanzado la excepción esperada.");
            }

            // Comprometer la transacción para asegurarse que todos los bloqueos
            // se liberan.
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

"Transacciones JDBC distribuidas" en la página 82
Generalmente, en Java Database Connectivity (JDBC) las transacciones son locales. Esto significa que una sola conexión realiza todo el trabajo de la transacción y que la conexión solo puede trabajar en una transacción a la vez.

Ejemplo: ID de usuario y contraseña no válidos

Este es un ejemplo de utilización de la propiedad Connection en la modalidad de denominación SQL.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de InvalidConnect.
//
// Este programa utiliza la propiedad Connection en modalidad de denominación SQL.
//
//////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo del controlador JDBC de IBM Developer para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo con fines ilustrativos
// solamente. Los ejemplos no se han probado minuciosamente bajo todas
// las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas contenidos aquí se proporcionan "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se declinan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// 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)
    {
        // Registrar el controlador.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: el controlador JDBC no se ha cargado.");
            System.exit(0);
        }

        // Intento de obtener una conexión sin especificar ningún usuario o
        // la contraseña. El intento funciona y la conexión utiliza el
        // mismo perfil de usuario bajo el que se ejecuta el trabajo.
        try {
            Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
            c1.close();
        } catch (SQLException e) {
            System.out.println("Esta prueba no debe incluirse en esta vía de excepciones.");
            e.printStackTrace();
            System.exit(1);
        }

        try {
            Connection c2 = DriverManager.getConnection("jdbc:db2:*local",
                                                        "notvalid", "notvalid");
        } catch (SQLException e) {
            System.out.println("Este es un error esperado.");
            System.out.println("El mensaje es " + e.getMessage());
            System.out.println("SQLSTATE es " + e.getSQLState());
```

```
        }

    }
}
```

Ejemplo: JDBC

Este es un ejemplo de cómo utilizar el programa BasicJDBC. Este programa emplea un controlador JDBC nativo para IBM Developer Kit para Java para construir una tabla simple y procesar una consulta que visualiza los datos de esa tabla.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de BasicJDBC. Este programa utiliza el controlador JDBC nativo de
// Developer Kit para Java para crear una tabla simple y procesar una consulta
// que visualice los datos de dicha tabla.
//
// Sintaxis de mandato:
//    BasicJDBC
//
////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo del controlador JDBC de IBM Developer Kit para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo solo con fines ilustrativos.
// Estos ejemplos no se han probado exhaustivamente bajo todas las
// condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas que contiene este ejemplo se suministran "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se rechazan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////////////////////////////////////////////

// Incluir las clases Java que deban utilizarse. En esta aplicación
// se utilizan muchas clases del paquete java.sql y también se utiliza
// la clase java.util.Properties como parte del proceso de obtención
// de una conexión con la base de datos.
import java.sql.*;
import java.util.Properties;

// Crear una clase pública para encapsular el programa.
public class BasicJDBC {

    // La conexión es una variable privada del objeto.
    private Connection connection = null;

    // Cualquier clase que deba ser un "punto de entrada" para ejecutar
    // un programa debe tener un método main. El método main
    // es donde empieza el proceso cuando se llama al programa.
    public static void main(java.lang.String[] args) {
```

```
    // Crear un objeto de tipo BasicJDBC. Esto
    // es fundamental en la programación orientada a objetos. Una vez
    // creado un objeto, llamar a diversos métodos de
    // ese objeto para realizar el trabajo.
    // En este caso, al llamar al constructor del objeto
    // se crea una conexión de base de datos que los otros
    // métodos utilizan para realizar el trabajo en la base de datos.
    BasicJDBC test = new BasicJDBC();

    // Llamar al método rebuildTable. Este método asegura que
    // la tabla utilizada en este programa existe y tiene el aspecto
    // correcto. El valor de retorno es un booleano para indicar
    // si la reconstrucción de la tabla se ha completado
    // satisfactoriamente. Si no es así, visualizar un mensaje
    // y salir del programa.
    if (!test.rebuildTable()) {
        System.out.println("Se produjo una anomalía al configurar " +
                           " para ejecutar la prueba.");
        System.out.println("La prueba no continuará.");
        System.exit(0);
    }

    // A continuación, se llama al método para ejecutar la consulta. Este método
    // procesa una sentencia SQL select con respecto a la tabla que
    // se creó en el método rebuildTable. La salida de
    // esa consulta va a la salida estándar de visualización.
    test.runQuery();

    // Por último, se llama al método cleanup. Este método
    // garantiza que la conexión de base de datos en la que el objeto
    // ha estado a la espera se ha cerrado.
    test.cleanup();
}


/**
Este es el constructor de la prueba básica JDBC. Crea una conexión
de base de datos que se almacena en una variable de instancia que se usará en
posteriores llamadas de método.
**/
public BasicJDBC() {

    // Una forma de crear una conexión de base de datos es pasar un URL
    // y un objeto java Properties al DriverManager. El siguiente
    // código construye un objeto Properties que contiene el ID de usuario y
    // la contraseña. Estos datos se emplean para establecer conexión
    // con la base de datos.
    Properties properties = new Properties ();
    properties.put("user", "cujo");
    properties.put("user", "newtiger");

    // Utilizar un bloque try/catch para capturar todas las excepciones que
    // puedan surgir del código siguiente.
    try {
        // DriverManager debe saber que existe un controlador JDBC disponible
        // para manejar una petición de conexión de usuario. La siguiente línea hace
        // que el controlador JDBC nativo se cargue y registre con DriverManager.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        // Crear el objeto Connection de base de datos que este programa utiliza
        // en las demás llamadas de método que se realicen. El código que sigue
        // especifica que debe establecerse una conexión con la base de datos local
        // y que dicha conexión debe ajustarse a las propiedades configuradas
        // anteriormente (es decir, debe utilizar el ID de usuario
        // y la contraseña especificados).
        connection = DriverManager.getConnection("jdbc:db2:*local", properties);
```

```
    } catch (Exception e) {
        // Si falla alguna de las líneas del bloque try/catch, el control pasa
        // a la siguiente línea de código. Una aplicación robusta intenta manejar
        // el problema o proporcionar más detalles. En este programa, se visualiza
        // el mensaje de error de la excepción, y la aplicación permite
        // el retorno del programa.
        System.out.println("Excepción capturada: " + e.getMessage());
    }
}


/**
Garantiza que la tabla qgpl.basicjdbc tiene el aspecto deseado al principio de
a prueba.

@returns boolean    Devuelve true si la tabla se ha reconstruido satisfactoriamente;
                    Devuelve false si se ha producido alguna anomalía.
**/
public boolean rebuildTable() {
    // Reiniciar todas las funciones de un bloque try/catch para que se realice
    // un intento de manejar los errores que puedan producirse dentro de este
    // método.
    try {

        // Se utilizan objetos Statement para procesar sentencias SQL en la
        // base de datos. El objeto Connection se utiliza para crear un
        // objeto Statement.
        Statement s = connection.createStatement();

        try {
            // Construir la tabla de prueba desde cero. Procesar una sentencia update
            // que intenta suprimir la tabla si existe actualmente.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // No realizar nada si se produjo una excepción. Presuponer
            // que el problema es que la tabla que se ha eliminado no
            // existe y que se puede crear a continuación.
        }

        // Utilizar el objeto sentencia para crear la tabla.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Utilizar el objeto sentencia para llenar la tabla con algunos
        // datos.
        s.executeUpdate("insert into qgpl.basicjdbc values(1, 'Frank Johnson')");
        s.executeUpdate("insert into qgpl.basicjdbc values(2, 'Neil Schwartz')");
        s.executeUpdate("insert into qgpl.basicjdbc values(3, 'Ben Rodman')");
        s.executeUpdate("insert into qgpl.basicjdbc values(4, 'Dan Gloore')");

        // Cerrar la sentencia SQL para indicar a la base de datos que ya no es
        // necesaria.
        s.close();

        // Si todo el método se ha procesado satisfactoriamente, devolver true. En este punto,
        // la tabla se ha creado o renovado correctamente.
        return true;

    } catch (SQLException sqle) {
        // Si ha fallado alguna de las sentencias SQL (que no sea la eliminación de la tabla
        // manejada en el bloque try/catch interno), el mensaje de error
        // se visualiza y se devuelve false al llamador, para indicar que la tabla
        // no puede completarse.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}
```

```
/**
Ejecuta una consulta a la tabla de muestra y los resultados se visualizan en
la salida estándar.
**/
public void runQuery() {
    // Reiniciar todas las funciones de un bloque try/catch para que se realice
    // un intento de manejar los errores que puedan producirse dentro de este
    // método.
    try {
        // Crear un objeto Statement.
        Statement s = connection.createStatement();

        // Usar el objeto Statement para ejecutar una consulta SQL. Las consultas devuelven
        // objetos ResultSet que se utilizan para observar los datos que la consulta
        // proporciona.
        ResultSet rs = s.executeQuery("select * from qgpl.basicjdbc");

        // Visualizar el principio de la 'tabla' e inicializar el contador del
        // número de filas devueltas.
        System.out.println("--------------------");
        int i = 0;

        // El método next de ResultSet se utiliza para procesar las filas de un
        // ResultSet. Hay que llamar al método next una vez antes de que
        // los primeros datos estén disponibles para verlos. Siempre que next devuelva
        // true, existe otra fila de datos que puede utilizarse.
        while (rs.next()) {

            // Obtener ambas columnas de la tabla para cada fila y escribir una fila en
            // nuestra tabla en pantalla con los datos. Luego, incrementar la cuenta
            // de filas que se han procesado.
            System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
            i++;
        }

        // Colocar un borde al final de la tabla y visualizar el número de filas
        // como salida.
        System.out.println("--------------------");
        System.out.println("Se han devuelto " + i + " filas.");
        System.out.println("Salida completada.");

    } catch (SQLException e) {
        // Visualizar más información acerca de las excepciones SQL
        // generadas como salida.
        System.out.println("Excepción SQLException: ");
        System.out.println("Mensaje:....." + e.getMessage());
        System.out.println("SQLState:...." + e.getSQLState());
        System.out.println("Código proveedor:." + e.getErrorCode());
        e.printStackTrace();
    }
}


/**
El siguiente método garantiza que se liberen los recursos JDBC que aún
están asignados.
**/
public void cleanup() {
    try {
        if (connection != null)
            connection.close();
    } catch (Exception e) {
        System.out.println("Excepción capturada: ");
```

```
            e.printStackTrace();
        }
    }
}
```

Ejemplo: varias conexiones que funcionan en una transacción

Este es un ejemplo de utilización de varias conexiones que trabajan en una sola transacción.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
   }
/**
* Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
*/
   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) {
         // Ignorar... no existe
         }
         s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
                         (50))");
               s.close();
      }
      finally {
         if (c != null) {
            c.close();
         }
      }
   }
/**
* Esta prueba utiliza el soporte JTA para manejar transacciones.
*/
   public void run() {
      Connection c1 = null;
      Connection c2 = null;
      Connection c3 = null;
      try {
         Context ctx = new InitialContext();
         // Presuponer que el origen de datos se apoya en un UDBXADataSource.
         UDBXADataSource ds = (UDBXADataSource)
                              ctx.lookup("XADataSource");
         // Desde el DataSource, obtener algunos objetos XAConnection que
         // contienen un XAResource y un objeto 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();
   // Para transacciones XA, es necesario un identificador de transacción.
   // El soporte para crear XID se deja de nuevo al programa
   // de aplicación.
   Xid xid = JDXATest.xidFactory();
   // Realizar algún trabajo transaccional bajo cada una de las tres
   // conexiones que se han creado.
   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);
   // Al terminar, comprometer la transacción como una sola unidad.
   // Es necesario prepare() y commit() o commit() de 1 fase para
   // cada base de datos independiente (XAResource) que ha participado en la
   // transacción. Dado que los recursos accedidos (xaRes1, xaRes2 y xaRes3)
   // se refieren todos a la misma base de datos, solo se necesita una
   // prepare o una commit.
   int rc = xaRes.prepare(xid);
   xaRes.commit(xid, false);
}
catch (Exception e) {
   System.out.println("Algo ha fallado.");
   e.printStackTrace();
}
finally {
   try {
      if (c1 != null) {
         c1.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Nota: Excepción de limpieza " +
                         e.getMessage());
   }
   try {
      if (c2 != null) {
         c2.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Nota: Excepción de limpieza " +
                         e.getMessage());
   }
   try {
      if (c3 != null) {
         c3.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Nota: Excepción de limpieza " +
                         e.getMessage());
```

```
        }
      }
   }
}
```

Ejemplo: obtener un contexto inicial antes de enlazar UDBDataSource

En el ejemplo siguiente se obtiene un contexto inicial antes de enlazar el UDBDataSource. A continuación, se utiliza el método lookup en ese contexto para devolver un objeto de tipo DataSource para que lo utilice la aplicación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Importar los paquetes necesarios. No hay ningún
// código específico del controlador que se necesite en las
// aplicaciones de tiempo de ejecución.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Recuperar un contexto JNDI. El contexto funciona
        // como raíz donde los objetos se enlazan o
        // se encuentran en JNDI.
        Context ctx = new InitialContext();

        // Recuperar el objeto UDBDataSource enlazado mediante el
        // nombre con el que estaba enlazado anteriormente. En tiempo de ejecución,
        // solo se utiliza la interfaz DataSource y, por lo tanto,
        // no hace falta convertir el objeto a la clase de
        // implementación de UDBDataSource. (No hace falta saber cuál
        // es la clase de implementación. El nombre JNDI lógico es
        // lo único necesario).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Una vez obtenido, el DataSource se puede usar para establecer
        // una conexión. Este objeto Connection es el mismo tipo
        // de objeto que el que se devuelve si se emplea el enfoque DriverManager
        // para establecer conexión. Por lo tanto, todo lo que hay desde
        // este punto en adelante es exactamente igual que en cualquier otra
        // aplicación JDBC.
        Connection connection = ds.getConnection();

        // La conexión puede utilizarse para crear objetos Statement y
        // actualizar la base de datos o procesar consultas de la forma siguiente.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // La conexión se cierra antes de que finalice la aplicación.
        connection.close();
    }
}
```

Ejemplo: ParameterMetaData

Este es un ejemplo de utilización de la interfaz ParameterMetaData para recuperar información acerca de los parámetros.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de ParameterMetaData. Este programa muestra
// el nuevo soporte de JDBC 3.0 para obtener información
// acerca de los parámetros de una PreparedStatement.
//
// Sintaxis de mandato:
//    java PMD
//
//////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo del controlador JDBC de IBM Developer para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo con fines ilustrativos
// solamente. Los ejemplos no se han probado minuciosamente bajo todas
// las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas contenidos aquí se proporcionan "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se declinan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// 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 de entrada del programa.
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Obtener configuración.
        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("Número parámetro " + i);
            System.out.println("  El nombre de clase es " + pmd.getParameterClassName(i));
            // Nota: la modalidad hace referencia a entrada, salida y entrada/salida
            System.out.println("  La modalidad es " + pmd.getParameterClassName(i));
            System.out.println("  El tipo es " + pmd.getParameterType(i));
            System.out.println("  El nombre de tipo es " + pmd.getParameterTypeName(i));
            System.out.println("  La precisión es " + pmd.getPrecision(i));
            System.out.println("  La escala es " + pmd.getScale(i));
            System.out.println("  ¿Posibilidad de nulos? es " + pmd.isNullable(i));
            System.out.println("  ¿Firmado? es " + pmd.isSigned(i));
        }
    }
}
```

Ejemplo: cambiar valores con una sentencia mediante el cursor de otra sentencia

Este ejemplo Java enseña a cambiar valores con una sentencia mediante el cursor de otra sentencia.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }



/**
Manejar todo el trabajo de configuración necesario.
**/
    public void setup() {
        try {
            // Registrar el controlador 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) {
                // Aquí, pasar por alto los problemas.
            }

            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("Excepción capturada: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
En esta sección, debe añadirse todo el código destinado a realizar
la prueba. Si solo se necesita una conexión con la base de datos,
se puede utilizar la variable global 'connection'.
**/
    public void run() {
        try {
```

```
            Statement stmt1 = connection.createStatement();

            // Actualizar cada valor utilizando next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("El nombre de cursor es " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
                                                                   + " CUJOSQL.WHERECUREX
                                                                       SET COL_VALUE = 'CHANGED'
                                                                       WHERE CURRENT OF "
                                                                   + rs.getCursorName ());


            // Repetir en bucle el ResultSet y actualizar las demás entradas.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Borrar los recursos una vez utilizados.
            rs.close ();
            stmt2.close ();


        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
En esta sección, colocar todo el trabajo de borrado para la prueba.
**/
    public void cleanup() {
        try {
            // Cerrar la conexión global abierta en setup().
            connection.close();

        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }



/**
Visualizar el contenido de la tabla.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Índice " + rs.getInt(1) + " valor " + rs.getString(2));
            }

            rs.close ();
            s.close();
```

```
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Ejemplo: interfaz ResultSet de IBM Developer Kit para Java

Este es un ejemplo de cómo utilizar la interfaz ResultSet.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.sql.*;

/**
ResultSetExample.java

Este programa muestra la utilización de ResultSetMetaData y
ResultSet para visualizar todos los datos de una tabla aunque
el programa que obtiene los datos no sabe cuál es el aspecto
que tendrá la tabla (el usuario pasa los valores correspondientes
a la tabla y a la biblioteca).
**/
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Uso:  java ResultSetExample <biblioteca> <tabla>");
            System.out.println(" siendo <biblioteca> la biblioteca que contiene la <tabla>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Obtener una conexión a base de datos y preparar una sentencia.
            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("Datos para la fila " + rowCount);
                for (int i = 1; i <= colCount; i++)
                    System.out.println("   Fila " + i + ": " + rs.getString(i));
            }

        } catch (Exception e) {
            // Manejar los errores.
            System.out.println("Tenemos un error... ");
            e.printStackTrace();
        } finally {
            // Hay que asegurarse de que siempre se haga
            // el borrado. Si la conexión se cierra, la
```

```
            // sentencia que hay debajo de ella también se cerrará.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Error grave: no se puede cerrar el objeto conexión");
                }
            }
        }
    }
}
```

Ejemplo: sensibilidad de ResultSet

El ejemplo siguiente muestra cómo un cambio puede afectar a una cláusula where de una sentencia SQL en función de la sensibilidad del ResultSet.

Puede que el formato de este ejemplo no sea del todo correcto como consecuencia de haber adaptado este ejemplo a una página impresa.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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("Se utiliza controlador JDBC nativo");
            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) {
                // Ignorado.
            }

            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) {
            // Ignorado.
        }

        s.executeUpdate("create table cujosql.sensitive2(col2 int)");
        s.executeUpdate("insert into cujosql.sensitive2 values(1)");
        s.executeUpdate("insert into cujosql.sensitive2 values(2)");
        s.executeUpdate("insert into cujosql.sensitive2 values(3)");
        s.executeUpdate("insert into cujosql.sensitive2 values(4)");
        s.executeUpdate("insert into cujosql.sensitive2 values(5)");

        s.close();

    } catch (Exception e) {
        System.out.println("Excepción capturada: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Otra:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creando cursor TYPE_SCROLL_INSENSITIVE");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creando cursor TYPE_SCROLL_SENSITIVE");
            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("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));

        System.out.println("captadas las cuatro filas...");

        // Otra sentencia crea un valor que no se ajusta a la cláusula 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("Todavía hay una fila: " + rs.getInt(1));
            } else {
                System.out.println("No hay más filas.");
            }

        } catch (SQLException e) {
            System.out.println("Excepción SQLException: ");
            System.out.println("Mensaje:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Código proveedor:." + e.getErrorCode());
            System.out.println("----------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("Se ha lanzado una excepción que no es una SQLException: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Ejemplo: ResultSets sensibles e insensibles

El ejemplo siguiente muestra la diferencia entre los ResultSets sensibles y los insensibles cuando se insertan filas en una tabla.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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) {
            // Ignorado.
        }

        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("Excepción capturada: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Otra:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {
        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creando un cursor TYPE_SCROLL_INSENSITIVE");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creando un cursor TYPE_SCROLL_SENSITIVE");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

        // Captar los cinco valores que se encuentran allí.
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        rs.next();
        System.out.println("el valor es " + rs.getInt(1));
        System.out.println("captadas las cinco filas...");

        // Nota:  Si capta la última fila, el ResultSet interpreta
        //        que las filas cerradas y las nuevas que se añadan
        //        no se reconocen.

        // Permitir que otra sentencia inserte un valor nuevo.
        Statement s2 = connection.createStatement();
```

```
            s2.executeUpdate("insert into cujosql.sensitive values(6)");
            s2.close();

            // El hecho de que una fila se reconozca se basa en el valor
            // de sensibilidad.
            if (rs.next()) {
                System.out.println("Ahora existe una fila: " + rs.getInt(1));
            } else {
                System.out.println("No hay más filas.");
            }


        } catch (SQLException e) {
            System.out.println("Excepción SQLException: ");
            System.out.println("Mensaje:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Código proveedor:." + e.getErrorCode());
            System.out.println("-------------------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("Se ha lanzado una excepción que no es una SQLException: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Excepción capturada: ");
            e.printStackTrace();
        }
    }
}
```

Ejemplo: configurar una agrupación de conexiones con UDBDataSource y UDBConnectionPoolDataSource

Este es un ejemplo de cómo utilizar una agrupación de conexiones con UDBDataSource y UDBConnectionPoolDataSource.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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
    {
        // Crear una implementación de ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Objeto DataSource de agrupación de conexiones");

        // Establecer un contexto JNDI y enlazar el origen de datos de agrupación
        // de conexiones
        Context ctx = new InitialContext();
```

```
        ctx.rebind("ConnectionSupport", cpds);

        // Crear un origen de datos estándar que haga referencia al mismo.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource que soporta la agrupación");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
  }
}
```

Ejemplo: SQLException

Este es un ejemplo de cómo capturar una SQLException y volcar toda la información que proporciona.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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("No se esperaba que la tabla existiera.");


        } catch (SQLException e) {
            System.out.println("Excepción SQLException: ");
            System.out.println("Mensaje:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Código proveedor:." + e.getErrorCode());
            System.out.println("-----------------------------------------------------");
            e.printStackTrace();
        } catch (Exception ex) {
            System.out.println("Se ha lanzado una excepción que no es una SQLException: ");
            ex.printStackTrace();
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                System.out.println("Excepción capturada al intentar concluir...");
            }
        }
    }
}
```

Ejemplo: suspender y reanudar una transacción

Este es un ejemplo de una transacción que se suspende y luego se reanuda.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
            }

            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();
            }
        }
    }


    /**
     * Esta prueba utiliza el soporte JTA para manejar transacciones.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Presuponer que el origen de datos se apoya en un UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // Desde el DataSource, obtener un objeto XAConnection que
            // contiene un XAResource y un objeto Connection.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();

            // Para transacciones XA, es necesario un identificador de transacción.
            // No se incluye una implementación de la interfaz XID con
            // el controlador JDBC. Vea Transacciones con JTA
            // para obtener una descripción de esta interfaz para crear una clase para ella.
```

```
    Xid xid = new XidImpl();

    // La conexión de XAResource puede usarse como cualquier otra
    // conexión JDBC.
    Statement stmt = c.createStatement();

    // Hay que notificar al recurso XA antes de iniciar cualquier
    // trabajo transaccional.
    xaRes.start(xid, XAResource.TMNOFLAGS);

    // Crear un ResultSet durante el proceso de JDBC y captar una fila.
    ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
    rs.next();

    // Se llama al método end con la opción suspend. Los
    // ResultSets asociados a la transacción actual quedan 'suspendidos'.
    // En este estado, no se eliminan ni son accesibles.
    xaRes.end(xid, XAResource.TMSUSPEND);


    // Pueden realizarse otras tareas con la transacción.
    // Como ejemplo, puede crear una sentencia y procesar una consulta.
    // Este trabajo, y cualquier otro transaccional que la transacción pueda
    // realizar, está separado del trabajo realizado anteriormente bajo XID.
    Statement nonXAStmt = conn.createStatement();
    ResultSet nonXARS = nonXAStmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
    while (nonXARS.next()) {
        // Procesar aquí...
    }
    nonXARS.close();
    nonXAStmt.close();


    // Si se intenta utilizar recursos de transacciones
    // suspendidas, se produce una excepción.
    try {
        rs.getString(1);
        System.out.println("El valor de la primera fila es " + rs.getString(1));
    } catch (SQLException e) {
        System.out.println("Esta es una excepción esperada - " +
                           "se ha utilizado ResultSet suspendido.");
    }


    // Reanudar la transacción suspendida y terminar el trabajo en ella.
    // El ResultSet es exactamente igual que antes de la suspensión.
    xaRes.start(newXid, XAResource.TMRESUME);
    rs.next();
    System.out.println("El valor de la segunda fila es " + rs.getString(1));


    // Cuando la transacción termine, finalizarla
    // y comprometer el trabajo que contenga.
    xaRes.end(xid, XAResource.TMNOFLAGS);
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);


} catch (Exception e) {
    System.out.println("Algo ha fallado.");
    e.printStackTrace();
} finally {
    try {
        if (c != null)
            c.close();
    } catch (SQLException e) {
        System.out.println("Nota: excepción de limpieza.");
```

```
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: ResultSets suspendidos

Este es un ejemplo de cómo se reprocesa un objeto Statement bajo otra transacción para realizar el trabajo.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
            }

            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();
            }
        }
    }


    /**
     * Esta prueba utiliza el soporte JTA para manejar transacciones.
     */
    public void run() {
        Connection c = null;
```

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

    // Presuponer que el origen de datos se apoya en un UDBXADataSource.
    UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

    // Desde el DataSource, obtener un objeto XAConnection que
    // contiene un XAResource y un objeto Connection.
    XAConnection  xaConn = ds.getXAConnection();
    XAResource    xaRes  = xaConn.getXAResource();
    Connection    c      = xaConn.getConnection();

    // Para transacciones XA, es necesario un identificador de transacción.
    // No se incluye una implementación de la interfaz XID con
    // el controlador JDBC. Vea: Transacciones con JTA
    // si desea obtener una descripción de esta interfaz para construir una
    // clase para ella.
    Xid xid = new XidImpl();

    // La conexión de XAResource puede usarse como cualquier otra
    // conexión JDBC.
    Statement stmt = c.createStatement();

    // Hay que notificar al recurso XA antes de iniciar cualquier
    // trabajo transaccional.
    xaRes.start(xid, XAResource.TMNOFLAGS);

    // Crear un ResultSet durante el proceso de JDBC y captar una fila.
    ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
    rs.next();

    // Se llama al método end con la opción suspend. Los
    // ResultSets asociados a la transacción actual quedan 'suspendidos'.
    // En este estado, no se eliminan ni son accesibles.
    xaRes.end(xid, XAResource.TMSUSPEND);


    // Mientras tanto, pueden realizarse otras tareas fuera de la
    // transacción.
    // Los ResultSets bajo la transacción pueden cerrarse si
    // se reutiliza el objeto Statement utilizado para crearlos.
    ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
    while (nonXARS.next()) {
        // Procesar aquí...
    }


    // Intento de volver a la transacción suspendida. El Resultset
    // de la transacción suspendida ha desaparecido porque la sentencia
    // se ha procesado de nuevo.
    xaRes.start(newXid, XAResource.TMRESUME);
    try {
        rs.next();
    } catch (SQLException ex) {
        System.out.println("Esta es una excepción esperada. " +
                           "El ResultSet se ha cerrado debido a otro proceso.");
    }


    // Cuando la transacción termine, finalizarla
    // y comprometer el trabajo que contenga.
    xaRes.end(xid, XAResource.TMNOFLAGS);
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);
```

```
        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: probar el rendimiento de una agrupación de conexiones

Este es un ejemplo de cómo probar el rendimiento del ejemplo de agrupación en comparación con el rendimiento del ejemplo sin agrupación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
        // Realizar el trabajo sin una agrupación:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("\nIniciar temporización de versión de DataSource sin agrupación...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Tiempo transcurrido: " + (endTime - startTime));

        // Realizar el trabajo con agrupación:
        ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nIniciar temporización de la versión con agrupación...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Tiempo transcurrido: " + (endTime - startTime));
    }
}
```

Ejemplo: probar el rendimiento de dos orígenes de datos (objeto DataSource)

En este ejemplo se ve cómo probar un DataSource que solo utiliza la agrupación de conexiones y otro DataSource que utiliza la agrupación de sentencias y conexiones.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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("desplegando origen de datos de agrupación de sentencias");
        deployStatementPoolDataSource();

        // Realizar el trabajo solo con la agrupación de conexiones.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nIniciar temporización de la versión solo de agrupación de conexiones...");

        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("Tiempo transcurrido: " + (endTime - startTime));


        // Realizar el trabajo añadiendo la agrupación de sentencias.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("\nIniciar temporización de la versión de agrupación de sentencias...");

        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("Tiempo transcurrido: " + (endTime - startTime));
    }



    private static void deployStatementPoolDataSource()
    throws Exception
    {
        // Crear una implementación de ConnectionPoolDataSource
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Objeto DataSource de agrupación de conexiones con agrupación de sentencias");
        cpds.setMaxStatements(10);

        // Establecer un contexto JNDI y enlazar el origen de datos de agrupación
        // de conexiones
        Context ctx = new InitialContext();
        ctx.rebind("StatementSupport", cpds);
```

```
        // Crear un datasource estándar que haga referencia a él.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource que soporta la agrupación de sentencias");
        ds.setDataSourceName("StatementSupport");
        ctx.rebind("StatementPoolingDataSource", ds);

    }
}
```

Ejemplo: actualizar objetos BLOB

Este es un ejemplo de cómo actualizar objetos BLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// UpdateBlobs es una aplicación de ejemplo
// que muestra algunas de las API que proporcionan
// soporte para cambiar objetos Blob
// y reflejar esos cambios en la
// base de datos.
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetBlobs.
////////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
       } catch (Exception e) {
          System.exit(1);  // Error de configuración.
       }

       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);


       // Truncar un BLOB.
       blob1.truncate((long) 150000);
       System.out.println("La nueva longitud de Blob1 es " + blob1.length());

       // Actualizar parte del BLOB con una matriz de bytes nueva.
       // El código siguiente obtiene los bytes que están en
       // las posiciones 4000-4500 y los establece en las posiciones 500-1000.

       // Obtener parte del BLOB como matriz de bytes.
       byte[] bytes = blob1.getBytes(4000L, 4500);

       int bytesWritten = blob2.setBytes(500L, bytes);

       System.out.println("Los bytes escritos son " + bytesWritten);

       // Los bytes se encuentran ahora en la posición 500 de blob2
       long startInBlob2 = blob2.position(bytes, 1);
```

```
      System.out.println("encontrado patrón que empieza en la posición " + startInBlob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: actualizar objetos CLOB

Este es un ejemplo de cómo actualizar objetos CLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////
// UpdateClobs es una aplicación de ejemplo
// que muestra algunas de las API que proporcionan
// soporte para cambiar objetos Clob
// y reflejar esos cambios en la
// base de datos.
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetClobs.
////////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      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);


      // Truncar un CLOB.
      clob1.truncate((long) 150000);
      System.out.println("La nueva longitud de Clob1 es " + clob1.length());

      // Actualizar una parte del CLOB con un nuevo valor de tipo String.
      String value = "Some new data for once";
      int charsWritten = clob2.setString(500L, value);

      System.out.println("Los caracteres escritos son " + charsWritten);

      // Los bytes se encuentran en la posición 500 de clob2
      long startInClob2 = clob2.position(value, 1);

      System.out.println("encontrado patrón que empieza en la posición " + startInClob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: utlizar una conexión con múltiples transacciones

Este es un ejemplo de utilización de una sola conexión con varias transacciones.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();
    }


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
            }

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

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


    /**
     * Esta prueba utiliza el soporte JTA para manejar transacciones.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Presuponer que el origen de datos se apoya en un UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // Desde el DataSource, obtener un objeto XAConnection que
            // contiene un XAResource y un objeto Connection.
```

```
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();
            Statement     stmt   = c.createStatement();

            // Para transacciones XA, es necesario un identificador de transacción.
            // Esto no implica que todos los XID sean iguales.
            // Cada XID debe ser exclusivo para distinguir las diversas transacciones
            // que se producen.
            // El soporte para crear XID se deja de nuevo al programa
            // de aplicación.
            Xid xid1 = JDXATest.xidFactory();
            Xid xid2 = JDXATest.xidFactory();
            Xid xid3 = JDXATest.xidFactory();

            // Realizar el trabajo bajo tres transacciones para esta conexión.
            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);

            // Preparar todas las transacciones
            int rc1 = xaRes.prepare(xid1);
            int rc2 = xaRes.prepare(xid2);
            int rc3 = xaRes.prepare(xid3);

            // Dos de las transacciones se comprometen y la tercera se retrotrae.
            // El intento de insertar el segundo valor en la tabla
            // no se compromete.
            xaRes.commit(xid1, false);
            xaRes.rollback(xid2);
            xaRes.commit(xid3, false);


        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: utilizar objetos BLOB

Este es un ejemplo de cómo utilizar objetos BLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/////////////////////////////////////////
// UseBlobs es una aplicación de ejemplo
// que muestra algunas de las API asociadas
// con objetos Blob.
```

```
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetBlobs.
/////////////////////////////////////////
import java.sql.*;

public class UseBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      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);


      // Determinar la longitud de un LOB.
      long end = blob1.length();
      System.out.println("La longitud de Blob1 es " + blob1.length());

      // Al trabajar con objetos LOB, toda la indexación relacionada con ellos
      // se basa en 1, y no en 0 como en las series y matrices.
      long startingPoint = 450;
      long endingPoint = 500;

      // Obtener parte del BLOB como matriz de bytes.
      byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

      // Buscar donde se encuentra primero un sub-BLOB o matriz de bytes en un
      // BLOB. La configuración de este programa ha colocado dos copias idénticas
      // de un BLOB aleatorio en la base de datos. Por tanto, la posición inicial
      // de la matriz de bytes extraída de blob1 se puede encontrar en la
      // posición inicial de blob2. La excepción sería si hubiera 50
      // bytes aleatorios idénticos en los LOB anteriormente.
      long startInBlob2 = blob2.position(outByteArray, 1);

      System.out.println("encontrado patrón que empieza en la posición " + startInBlob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: utilizar objetos CLOB

Este es un ejemplo de cómo utilizar objetos CLOB en las aplicaciones Java.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/////////////////////////////////////////
// UpdateClobs es una aplicación de ejemplo
// que muestra algunas de las API que proporcionan
// soporte para cambiar objetos Clob
// y reflejar esos cambios en la
// base de datos.
```

```
//
// Este programa debe ejecutarse después de
// finalizar el programa PutGetClobs.
//////////////////////////////////////
import java.sql.*;

public class UseClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Registrar el controlador JDBC nativo.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Error de configuración.
      }

      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);


      // Determinar la longitud de un LOB.
      long end = clob1.length();
      System.out.println("La longitud de Clob1 es " + clob1.length());

      // Al trabajar con objetos LOB, toda la indexación relacionada con ellos
      // se basa en 1, y no en 0 como en las series y matrices.
      long startingPoint = 450;
      long endingPoint = 50;

      // Obtener parte del CLOB como matriz de bytes.
      String outString = clob1.getSubString(startingPoint, (int)endingPoint);
      System.out.println("La subserie de Clob es " + outString);

      // Buscar donde se encuentra primero un sub-CLOB o serie en un
      // CLOB. La configuración de este programa ha colocado dos copias idénticas
      // de un CLOB repetido en la base de datos. Por tanto, la posición inicial
      // de la serie extraída de clob1 se puede encontrar en la
      // posición inicial de clob2 si la búsqueda empieza cerca de la posición
      // en la que empieza la serie.
      long startInClob2 = clob2.position(outString, 440);

      System.out.println("encontrado patrón que empieza en la posición " + startInClob2);

      c.close(); // El cierre de la conexión también cierra stmt y rs.
   }
}
```

Ejemplo: utilizar JTA para manejar una transacción

Este es un ejemplo de cómo utilizar la API de transacciones Java para manejar una transacción en una aplicación.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
```

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

public class JTACommit {

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

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


    /**
     * Manejar la ejecución de limpieza anterior para que esta prueba pueda volver a empezar.
     */
    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) {
                // Ignorar... no existe
            }

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


    /**
     * Esta prueba utiliza el soporte JTA para manejar transacciones.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Presuponer que el origen de datos se apoya en un UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // Desde el DataSource, obtener un objeto XAConnection que
            // contenga un XAResource y un objeto Connection.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();

            // Para transacciones XA, es necesario un identificador de transacción.
            // No se incluye una implementación de la interfaz XID con el
            // controlador JDBC. En Transacciones con JTA hay una descripción de
            // esta interfaz para construir una clase para ella.
            Xid xid = new XidImpl();

            // La conexión de XAResource puede usarse como cualquier otra
            // conexión JDBC.
```

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

            // Hay que notificar al recurso XA antes de iniciar cualquier
            // trabajo transaccional.
            xaRes.start(xid, XAResource.TMNOFLAGS);

            // Se realiza trabajo JDBC estándar.
            int count =
              stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA es bastante divertido.')");

            // Cuando el trabajo de transacción ha terminado, el recurso XA debe
            // recibir notificación de nuevo.
            xaRes.end(xid, XAResource.TMSUCCESS);

            // La transacción representada por el ID de transacción se prepara
            // para el compromiso.
            int rc = xaRes.prepare(xid);

            // La transacción se compromete mediante el XAResource.
            // No se utiliza el objeto JDBC Connection para comprometer
            // la transacción al utilizar JTA.
            xaRes.commit(xid, false);


        } catch (Exception e) {
            System.out.println("Algo ha fallado.");
            e.printStackTrace();
        } finally {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Nota: excepción de limpieza.");
                e.printStackTrace();
            }
        }
    }
}
```

Ejemplo: utilizar ResultSets de metadatos que tienen más de una columna

Este es un ejemplo de utilización de ResultSets de metadatos que tienen más una columna.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
//////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de SafeGetUDTs. Este programa muestra una forma de tratar con
// ResultSets de metadatos que tienen más columnas en JDK 1.4 que en
// releases anteriores.
//
// Sintaxis de mandato:
//    java SafeGetUDTs
//
//////////////////////////////////////////////////////////////////////////////
//
// Este código fuente es un ejemplo del controlador JDBC de IBM Developer para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo con fines ilustrativos
// solamente. Los ejemplos no se han probado minuciosamente bajo todas
// las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
```

```
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas que contiene este ejemplo se suministran "TAL CUAL",
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se declinan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// 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: El bloque estático se ejecuta antes de que se inicie main.
    // Por tanto, existe acceso a jdbcLevel en
    // main.
    {
        try {
            Class.forName("java.sql.Blob");

            try {
                Class.forName("java.sql.ParameterMetaData");
                // Encontrada una interfaz JDBC 3.0. Debe soportar JDBC 3.0.
                jdbcLevel = 3;
            } catch (ClassNotFoundException ez) {
                // No se ha encontrado la clase ParameterMetaData de JDBC 3.0.
                // Debe estar en ejecución bajo una JVM solo con soporte
                // JDBC 2.0.
                jdbcLevel = 2;
            }

        } catch (ClassNotFoundException ex) {
            // No se ha encontrado la clase Blob de JDBC 2.0. Debe estar
            // en ejecución bajo una JVM solo con soporte para JDBC 1.0.
            jdbcLevel = 1;
        }
    }

    // Punto de entrada del programa.
    public static void main(java.lang.String[] args)
    {
        Connection c = null;

        try {
            // Obtener el controlador registrado.
            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 hay soporte para getUDTs. Solo retornar.");
                System.exit(1);
            }

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

```
                // Captar todas las columnas que han estado disponibles desde
                // el release JDBC 2.0.
                System.out.println("TYPE_CAT es " + rs.getString("TYPE_CAT"));
                System.out.println("TYPE_SCHEM es " + rs.getString("TYPE_SCHEM"));
                System.out.println("TYPE_NAME es " + rs.getString("TYPE_NAME"));
                System.out.println("CLASS_NAME es " + rs.getString("CLASS_NAME"));
                System.out.println("DATA_TYPE es " + rs.getString("DATA_TYPE"));
                System.out.println("REMARKS es " + rs.getString("REMARKS"));

                // Captar todas las columnas que se han añadido en JDBC 3.0.
                if (jdbcLevel > 2) {
                    System.out.println("BASE_TYPE es " + rs.getString("BASE_TYPE"));
                }
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            if (c != null) {
                try {
                    c.close();
                } catch (SQLException e) {
                    // Ignorar excepción de cierre.
                }
            }
        }
    }
}
```

Ejemplo: utilizar JDBC nativo y JDBC de IBM Toolbox para Java de forma concurrente

Este es un ejemplo de cómo utilizar la conexión JDBC nativa y la conexión JDBC de IBM Toolbox para Java en un programa.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de GetConnections.
//
// Este programa muestra la posibilidad de utilizar ambos controladores JDBC a
// la vez en un programa. Se crean dos objetos Connection en este
// programa. Uno es una conexión JDBC nativa y el otro, una conexión JDBC de
// IBM Toolbox para Java.
//
// Esta es una técnica cómoda porque le permite utilizar distintos
// controladores JDBC para distintas tareas de manera concurrente. Por ejemplo,
// El controlador JDBC de IBM Toolbox para Java es ideal para conectar con un System i5 remoto
// y el controlador JDBC nativo es más rápido para las conexiones locales.
// Puede utilizar el potencial de cada controlador concurrentemente en la
// aplicación escribiendo código similar al de este ejemplo.
//
////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo del controlador JDBC de IBM Developer para Java.
// IBM le otorga una licencia no exclusiva para utilizarlo como un ejemplo
// a partir del cual puede generar funciones similares adaptadas
// a sus necesidades específicas.
//
// IBM suministra este código de ejemplo con fines ilustrativos
// solamente. Los ejemplos no se han probado minuciosamente bajo todas
// las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la
// fiabilidad, capacidad de servicio o funcionamiento de estos programas.
//
// Todos los programas contenidos aquí se proporcionan "TAL CUAL",
```

```
// sin garantías de ningún tipo. Las garantías implícitas de
// comercialización y adecuación a un propósito determinado
// se declinan explícitamente.
//
// IBM Developer Kit para Java
// (C) Copyright IBM Corp. 2001
// Reservados todos los derechos.
// 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)
   {
       // Verificar entrada.
       if (args.length != 2) {
          System.out.println("Utilización (línea de mandatos CL):  java GetConnections PARM(<usuario>  <contraseña>)");
          System.out.println(" donde <usuario> es un ID de usuario válido de System i5");
          System.out.println("y <contraseña> es la contraseña de ese ID de usuario");
          System.exit(0);
       }

       // Registrar ambos controladores.
      try {
         Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
         Class.forName("com.ibm.as400.access.AS400JDBCDriver");
      } catch (ClassNotFoundException cnf) {
         System.out.println("ERROR: uno de los controladores JDBC no se ha cargado.");
         System.exit(0);
      }

      try {
         // Obtener una conexión con cada controlador.
         Connection conn1 = DriverManager.getConnection("jdbc:db2://localhost", args[0], args[1]);
         Connection conn2 = DriverManager.getConnection("jdbc:as400://localhost", args[0], args[1]);

         // Verificar que son diferentes.
         if (conn1 instanceof com.ibm.db2.jdbc.app.DB2Connection)
            System.out.println("conn1 se ejecuta bajo el controlador JDBC nativo.");
         else
            System.out.println("Algo ha salido mal con respecto a conn1.");

         if (conn2 instanceof com.ibm.as400.access.AS400JDBCConnection)
            System.out.println("conn2 se ejecuta bajo el controlador JDBC de IBM Toolbox para Java.");
         else
            System.out.println("Algo ha salido mal con respecto a conn2.");

         conn1.close();
         conn2.close();
      } catch (SQLException e) {
         System.out.println("ERROR: " + e.getMessage());
      }
   }
}
```

Ejemplo: utilizar PreparedStatement para obtener un ResultSet

Este es un ejemplo de utilización del método executeQuery de un objeto PreparedStatement para obtener un ResultSet.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

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

public class PreparedStatementExample {

    public static void main(java.lang.String[] args)
    {
        // Cargar lo siguiente desde un objeto de propiedades.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2:*local";

        // Registrar el controlador JDBC nativo. Si el controlador no puede
        // registrarse, la prueba no puede continuar.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("No se ha podido registrar el controlador.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        //    Este programa crea una tabla que
        //    las sentencias preparadas utilizan más tarde.
        try {
            // Crear las propiedades de conexión.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Conectar con la base de datos local.
            c = DriverManager.getConnection(URL, properties);

            // Crear un objeto Statement.
            s = c.createStatement();
            // Suprimir la tabla de prueba, si existe. Observar que
            // en todo este ejemplo se presupone que la colección
            // MYLIBRARY existe en el sistema.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Continuar simplemente... es probable que la tabla no exista.
            }

            // Ejecutar una sentencia SQL que cree una tabla en la base de datos.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

        } catch (SQLException sqle) {
            System.out.println("El proceso de base de datos ha fallado.");
            System.out.println("Razón: " + sqle.getMessage());
        } finally {
            // Cerrar los recursos de base de datos.
            try {
                if (s != null) {
                    s.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Statement.");
            }
        }
```

```
// Luego, este programa usa una sentencia preparada para insertar muchas
// filas en la base de datos.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Crear un objeto PreparedStatement utilizado para insertar datos en la
    // tabla.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

    for (int i = 0; i < nameArray.length; i++) {
        ps.setString(1, nameArray[i]);  // Establecer el nombre a partir de nuestra matriz.
        ps.setInt(2, i+1);              // Establecer el ID.
        ps.executeUpdate();
    }

} catch (SQLException sqle) {
    System.out.println("El proceso de base de datos ha fallado.");
    System.out.println("Razón: " + sqle.getMessage());
} finally {
    // Cerrar los recursos de base de datos.
    try {
        if (ps != null) {
            ps.close();
        }
    } catch (SQLException e) {
        System.out.println("El borrado no ha podido cerrar Statement.");
    }
}


//    Utilizar una sentencia preparada para consultar la tabla de
//    base de datos creada y devolver datos desde ella. En
//    este ejemplo, el parámetro usado se ha establecido arbitrariamente
//    en 5, es decir, devolver todas las filas en que el campo ID sea menor
//    o igual a 5.
try {
    ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
                            "WHERE ID <= ?");

    ps.setInt(1, 5);

    // Ejecutar una consulta SQL en la tabla.
    ResultSet rs = ps.executeQuery();
    // Visualizar todos los datos de la tabla.
    while (rs.next()) {
        System.out.println("El empleado " + rs.getString(1) + " tiene el ID " + rs.getInt(2));
    }

} catch (SQLException sqle) {
    System.out.println("El proceso de base de datos ha fallado.");
    System.out.println("Razón: " + sqle.getMessage());
} finally {
    // Cerrar los recursos de base de datos.
    try {
        if (ps != null) {
            ps.close();
        }
    } catch (SQLException e) {
        System.out.println("El borrado no ha podido cerrar Statement.");
    }

    try {
        if (c != null) {
            c.close();
        }
    } catch (SQLException e) {
```

```
                System.out.println("El borrado no ha podido cerrar Connection.");
;
            }

        }
    }
}
```

Ejemplo: utilizar el método executeUpdate del objeto Statement

Este es un ejemplo de utilización del método executeUpdate del objeto Statement.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

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

public class StatementExample {

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

        // Sugerencia: cargarlos desde un objeto de propiedades.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Registrar el controlador JDBC nativo. Si el controlador no puede
        // registrarse, la prueba no puede continuar.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("No se ha podido registrar el controlador.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        try {
            // Crear las propiedades de conexión.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Conectar con la base de datos local de System i5.
            c = DriverManager.getConnection(URL, properties);

            // Crear un objeto Statement.
            s = c.createStatement();
            // Suprimir la tabla de prueba, si existe. Nota: en este
            // ejemplo se presupone que la colección MYLIBRARY
            // existe en el sistema.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Continuar simplemente... es probable que la tabla no exista.
            }

            // Ejecutar una sentencia SQL que crea una tabla en la base de datos.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

            // Ejecutar algunas sentencias SQL que insertan registros en la tabla.
            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)");
```

```
            // Ejecutar una consulta SQL en la tabla.
            ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

            // Visualizar todos los datos de la tabla.
            while (rs.next()) {
                System.out.println("El empleado " + rs.getString(1) + " tiene el ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("El proceso de base de datos ha fallado.");
            System.out.println("Razón: " + sqle.getMessage());
        } finally {
            // Cerrar los recursos de base de datos.
            try {
                if (s != null) {
                    s.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Statement.");
            }
             }

            try {
                if (c != null) {
                    c.close();
                }
            } catch (SQLException e) {
                System.out.println("El borrado no ha podido cerrar Connection.");
            }
        }
    }
}
```

Ejemplos: HelloWorld para JAAS

Estos ejemplos muestran los tres archivos que se necesitan para compilar y ejecutar HelloWorld para JAAS.

HelloWorld.java

Este es el fuente del archivo HelloWorld.java:

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/*
 * ============================================================================
 * Materiales con licencia - Propiedad de IBM
 *
 * (C) Copyright IBM Corp. 2000 Reservados todos los derechos.
 *
 *  Derechos restringidos de los usuarios del Gobierno de EE. UU.
 *  El uso, la reproducción o la divulgación están sujetos a las
 *  restricciones establecidas por GSA ADP Schedule Contract con IBM Corp.
 * ============================================================================
 *
 * Archivo: HelloWorld.java
 */

import java.io.*;
import java.util.*;
import java.security.Principal;
import java.security.PrivilegedAction;
import javax.security.auth.*;
import javax.security.auth.callback.*;
```

```
import javax.security.auth.login.*;
import javax.security.auth.spi.*;

/**
 * Esta aplicación SampleLogin intenta autenticar a un usuario.
 *
 * Si el usuario se autentica satisfactoriamente,
 * se visualiza el nombre de usuario y el número de credenciales.
 *
 * @version 1.1, 09/14/99
 */
public class HelloWorld {

    /**
     * Intentar autenticar al usuario.
     */
    public static void main(String[] args) {
        // usar los módulos LoginModule configurados para la entrada "helloWorld"
        LoginContext lc = null;
        try {
            lc = new LoginContext("helloWorld", new MyCallbackHandler());
        } catch (LoginException le) {
            le.printStackTrace();
            System.exit(-1);
        }

        // El usuario tiene 3 intentos para autenticarse satisfactoriamente.
        int i;
        for (i = 0; i < 3; i++) {
            try {

                // intentar autenticación
                lc.login();

                // Si no se devuelve ninguna excepción,
                // la autenticación ha sido satisfactoria.
                break;

            } catch (AccountExpiredException aee) {

                System.out.println("Su cuenta ha caducado");
                System.exit(-1);

            } catch (CredentialExpiredException cee) {

                System.out.println("Sus credenciales han caducado.");
                System.exit(-1);

            } catch (FailedLoginException fle) {

                System.out.println("Autenticación fallida");
                    try {
                        Thread.currentThread().sleep(3000);
                    } catch (Exception e) {
                        // Ignorar
                    }

            } catch (Exception e) {

                System.out.println("Excepción no prevista - imposible continuar");
                e.printStackTrace();
                System.exit(-1);
            }
        }

        // ¿Se ha fallado tres veces?
        if (i == 3) {
```

```
            System.out.println("Lo lamentamos");
            System.exit(-1);
        }

        // Veamos qué Principales tenemos:
        Iterator principalIterator = lc.getSubject().getPrincipals().iterator();
        System.out.println("\n\nEl usuario autenticado tiene los siguientes Principales:");
        while (principalIterator.hasNext()) {
            Principal p = (Principal)principalIterator.next();
            System.out.println("\t" + p.toString());
        }

        // Examinemos parte del trabajo basado en Principal:
        Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
            public Object run() {
                System.out.println("\nSu propiedad java.home: "
                                +System.getProperty("java.home"));

                System.out.println("\nSu propiedad user.home: "
                                +System.getProperty("user.home"));

                File f = new File("foo.txt");
                System.out.print("\nfoo.txt does ");
                if (!f.exists()) System.out.print("not ");
                System.out.println("existe en el directorio actual");

                System.out.println("\nOh, por cierto ...");

                try {
                    Thread.currentThread().sleep(2000);
                } catch (Exception e) {
                    // Ignorar
                }
                System.out.println("\n\nHello World!\n");
                return null;
            }
        }, null);
        System.exit(0);
    }
}


/**
 * La aplicación debe implementar CallbackHandler.
 *
 * Esta aplicación está basada en texto. Por lo tanto, visualiza información
 * ante el usuario mediante las corrientes de salida System.out y System.err,
 * y reúne la entrada procedente del usuario que utiliza la corriente de entrada System.in.
 */
class MyCallbackHandler implements CallbackHandler {

    /**
     * Invocar una matriz de objetos Callback.
     *
     *
     * @param callbacks una matriz de objetos Callback que contiene
     *          la información solicitada por un servicio de seguridad
     *          subyacente y que se debe recuperar o visualizar.
     *
     * @exception java.io.IOException si se produce un error de entrada o de salida.
     *
     * @exception UnsupportedCallbackException si la implementación de este
     *          método no da soporte a uno o más de los objetos Callback
     *          especificados en el parámetro callbacks.
     */
    public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException {
```

```
for (int i = 0; i < callbacks.length; i++) {
    if (callbacks[i] instanceof TextOutputCallback) {

    // Visualiza el mensaje de acuerdo con el tipo especificado.
    TextOutputCallback toc = (TextOutputCallback)callbacks[i];
    switch (toc.getMessageType()) {
    case TextOutputCallback.INFORMATION:
        System.out.println(toc.getMessage());
        break;
    case TextOutputCallback.ERROR:
        System.out.println("ERROR: " + toc.getMessage());
        break;
    case TextOutputCallback.WARNING:
        System.out.println("AVISO: " + toc.getMessage());
        break;
    default:
        throw new IOException("Tipo de mensaje no soportado: " +
                toc.getMessageType());
    }

    } else if (callbacks[i] instanceof NameCallback) {

    // Solicitar al usuario un nombre de usuario.
    NameCallback nc = (NameCallback)callbacks[i];

    // Ignoraracer el defaultName proporcionado.
    System.err.print(nc.getPrompt());
    System.err.flush();
    nc.setName((new BufferedReader
        (new InputStreamReader(System.in))).readLine());

    } else if (callbacks[i] instanceof PasswordCallback) {

    // Solicitar al usuario información confidencial.
    PasswordCallback pc = (PasswordCallback)callbacks[i];
    System.err.print(pc.getPrompt());
    System.err.flush();
    pc.setPassword(readPassword(System.in));

    } else {
    throw new UnsupportedCallbackException
        (callbacks[i], "Unrecognized Callback");
    }
}
}

// Lee la contraseña de usuario en la corriente de entrada proporcionada.
private char[] readPassword(InputStream in) throws IOException {

char[] lineBuffer;
char[] buf;
int i;

buf = lineBuffer = new char[128];

int room = buf.length;
int offset = 0;
int c;

loop:   while (true) {
    switch (c = in.read()) {
    case -1:
    case '\n':
    break loop;

    case '\r':
```

```
        int c2 = in.read();
        if ((c2 != '\n') && (c2 != -1)) {
            if (!(in instanceof PushbackInputStream)) {
            in = new PushbackInputStream(in);
            }
            ((PushbackInputStream)in).unread(c2);
        } else
            break loop;

        default:
        if (--room < 0) {
            buf = new char[offset + 128];
            room = buf.length - offset - 1;
            System.arraycopy(lineBuffer, 0, buf, 0, offset);
            Arrays.fill(lineBuffer, ' ');
            lineBuffer = buf;
        }
        buf[offset++] = (char) c;
        break;
        }
    }

    if (offset == 0) {
        return null;
    }

    char[] ret = new char[offset];
    System.arraycopy(buf, 0, ret, 0, offset);
    Arrays.fill(buf, ' ');

    return ret;
    }
}
```

HWLoginModule.java

Este es el fuente del archivo HWLoginModule.java.

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
/*
 * ============================================================================
 * Materiales con licencia - Propiedad de IBM
 *
 * (C) Copyright IBM Corp. 2000 Reservados todos los derechos.
 *
 *  Derechos restringidos de los usuarios del Gobierno de EE. UU.
 *  El uso, la reproducción o la divulgación están sujetos a las
 *  restricciones establecidas por GSA ADP Schedule Contract con IBM Corp.
 * ============================================================================
 *
 * Archivo: HWLoginModule.java
 */

package com.ibm.security;

import java.util.*;
import java.io.IOException;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;
import com.ibm.security.HWPrincipal;

/**
```

```
 * Este LoginModule autentica a los usuarios con una contraseña.
 *
 * Este LoginModule solo reconoce a los usuarios que entran
 *     la contraseña obligatoria:   Go JAAS
 *
 * Si el usuario se autentica satisfactoriamente,
 * un HWPrincipal con el nombre de usuario
 * se añade al Sujeto.
 *
 * Este LoginModule reconoce la opción de depuración (debug).
 * Si está establecida en true en la configuración de inicio de sesión,
 * los mensajes de depuración se envían a la corriente de salida, System.out.
 *
 * @version 1.1, 09/10/99
 */
public class HWLoginModule implements LoginModule {

        // Estado inicial.
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;

    // Opción configurable.
    private boolean debug = false;

    // El estado de autenticación.
    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    // Nombre de usuario y contraseña.
    private String user name;
    private char[] password;

    private HWPrincipal userPrincipal;

    /**
     * Inicializar este LoginModule.
     *
     * @param subject el sujeto que hay que autenticar.
     *
     * @param callbackHandler un CallbackHandler para comunicarse
     *          con el usuario final (solicitando nombres de usuario y
     *          contraseñas, por ejemplo).
     *
     * @param sharedState estado de LoginModule compartido.
     *
     * @param options opciones especificadas en la configuración de
     *          inicio de sesión para este determinado
     *          LoginModule.
     */
    public void initialize(Subject subject, CallbackHandler callbackHandler,
            Map sharedState, Map options) {

    this.subject = subject;
    this.callbackHandler = callbackHandler;
    this.sharedState = sharedState;
    this.options = options;

    // Inicializar las opciones que se hayan configurado.
    debug = "true".equalsIgnoreCase((String)options.get("debug"));
    }

    /**
     * Autenticar al usuario solicitando un nombre de usuario y una contraseña.
     *
     *
```

```
 * @return true en todos los casos, ya que este LoginModule
 *      no se debe pasar por alto.
 *
 @exception FailedLoginException si falla la autenticación.
 *
 * @exception LoginException si este LoginModule
 *      no puede efectuar la autenticación.
 */
public boolean login() throws LoginException {

// Solicitar un nombre de usuario y una contraseña.
if (callbackHandler == null)
    throw new LoginException("Error: no hay ningún CallbackHandler disponible " +
        "para recoger información de autenticación del usuario");

Callback[] callbacks = new Callback[2];
callbacks[0] = new NameCallback("\n\nHWModule user name: ");
callbacks[1] = new PasswordCallback("HWModule password: ", false);

try {
    callbackHandler.handle(callbacks);
    user name = ((NameCallback)callbacks[0]).getName();
    char[] tmpPassword = ((PasswordCallback)callbacks[1]).getPassword();
    if (tmpPassword == null) {
    // Manejar las contraseñas NULL como vacías.
    tmpPassword = new char[0];
    }
    password = new char[tmpPassword.length];
    System.arraycopy(tmpPassword, 0,
        password, 0, tmpPassword.length);
    ((PasswordCallback)callbacks[1]).clearPassword();

} catch (java.io.IOException ioe) {
    throw new LoginException(ioe.toString());
} catch (UnsupportedCallbackException uce) {
    throw new LoginException("Error: " + uce.getCallback().toString() +
    " no disponible para recoger información de autenticación " +
    "del usuario");
}

// Imprimir información de depuración.
if (debug) {
    System.out.println("\n\n\t[HWLoginModule] " +
            "usuario ha entrado nombre de usuario: " +
            user name);
    System.out.print("\t[HWLoginModule] " +
            "usuario ha entrado contraseña: ");
    for (int i = 0; i > password.length; i++)
    System.out.print(password[i]);
    System.out.println();
}

// Verificar la contraseña.
if (password.length == 7 &&
    password[0] == 'G' &&
    password[1] == 'o' &&
    password[2] == ' ' &&
    password[3] == 'J' &&
    password[4] == 'A' &&
    password[5] == 'A' &&
    password[6] == 'S') {

    // ¡La autenticación ha sido satisfactoria!
    if (debug)
    System.out.println("\n\t[HWLoginModule] " +
            "autenticación satisfactoria");
    succeeded = true;
```

```
    return true;
} else {

    // La autenticación ha fallado -- borrar estado.
    if (debug)
    System.out.println("\n\t[HWLoginModule] " +
            "autenticación fallida");
    succeeded = false;
    user name = null;
    for (int i = 0; i < password.length; i++)
    password[i] = ' ';
    password = null;
    throw new FailedLoginException("Contraseña incorrecta");
}
}

/**
 * Se llama a este método si la autenticación global de LoginContext
 * ha sido satisfactoria
 * (los módulos de inicio de sesión relevantes REQUIRED, REQUISITE,
 * SUFFICIENT y OPTIONAL
 * han sido satisfactorios).
 *
 * Si este intento de autenticación de LoginModule ha sido
 * satisfactorio (se comprueba al recuperar el estado privado guardado por
 * el método login), este método asociará un
 * SolarisPrincipal
 * al sujeto ubicado en el
 *          LoginModule.  Si este LoginModule
 * ha fallado, este método elimina
 * los estados que se hayan guardado originariamente.
 *
 * @exception LoginException si falla el compromiso.
 *
 * @return true si los intentos de LoginModule de inicio de sesión
 *      y compromiso han sido satisfactorios, o false en caso contrario.
 */
public boolean commit() throws LoginException {
if (succeeded == false) {
    return false;
} else {
    // Añadir un Principal (identidad autenticada)
    // al sujeto.

    // Se presupone que el usuario que hemos autenticado es HWPrincipal.
    userPrincipal = new HWPrincipal(user name);
    final Subject s = subject;
    final HWPrincipal sp = userPrincipal;
    java.security.AccessController.doPrivileged
    (new java.security.PrivilegedAction() {
    public Object run() {
        if (!s.getPrincipals().contains(sp))
        s.getPrincipals().add(sp);
        return null;
    }
    });

    if (debug) {
    System.out.println("\t[HWLoginModule] " +
            "añadió HWPrincipal al sujeto");
    }

    // En cualquier caso, borrar el estado.
    user name = null;
    for (int i = 0; i > password.length; i++)
    password[i] = ' ';
    password = null;
```

```
    commitSucceeded = true;
    return true;
}
}

/**
 * Se llama a este método si la autenticación global de LoginContext
 * ha fallado
 * (los módulos de inicio de sesión relevantes REQUIRED, REQUISITE,
 * SUFFICIENT y OPTIONAL
 * no han sido satisfactorios).
 *
 * Si este intento de autenticación de LoginModule ha sido
 * satisfactorio (se comprueba al recuperar el estado privado guardado por
 * los métodos login y commit),
 * este método borra los estados que se hayan guardado originariamente.
 *
 * @exception LoginException si falla la cancelación anómala.
 *
 * @return false si este intento de inicio de sesión o de compromiso de LoginModule
 *      ha fallado, y true en caso contrario.
 */
public boolean abort() throws LoginException {
if (succeeded == false) {
    return false;
} else if (succeeded == true && commitSucceeded == false) {
    // El inicio de sesión ha sido satisfactorio, pero
    // la autenticación global ha fallado.
    succeeded = false;
    user name = null;
    if (password != null) {
    for (int i = 0; i > password.length; i++)
        password[i] = ' ';
    password = null;
    }
    userPrincipal = null;
} else {
    // Han sido satisfactorios la autenticación global y el compromiso,
    // pero ha fallado otro compromiso.
    logout();
}
return true;
}

/**
 * Finalizar la sesión (método logout) del usuario.
 *
 * Este método elimina el HWPrincipal
 * que se añadió con el método commit.
 *
 * @exception LoginException si falla el método logout.
 *
 * @return true en todos los casos, ya que este LoginModule
 *      no se debe pasar por alto.
 */
public boolean logout() throws LoginException {

final Subject s = subject;
final HWPrincipal sp = userPrincipal;
java.security.AccessController.doPrivileged
    (new java.security.PrivilegedAction() {
    public Object run() {
    s.getPrincipals().remove(sp);
    return null;
    }
});
```

```
    succeeded = false;
    succeeded = commitSucceeded;
    user name = null;
    if (password != null) {
        for (int i = 0; i > password.length; i++)
        password[i] = ' ';
        password = null;
    }
    userPrincipal = null;
    return true;
    }
}
```

HWPrincipal.java

Este es el fuente del archivo HWPrincipal.java.

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
/*
 * ============================================================================
 * Materiales con licencia - Propiedad de IBM
 *
 * (C) Copyright IBM Corp. 2000 Reservados todos los derechos.
 *
 *  Derechos restringidos de los usuarios del Gobierno de EE. UU.
 *  El uso, la reproducción o la divulgación están sujetos a las
 *  restricciones establecidas por GSA ADP Schedule Contract con IBM Corp.
 * ============================================================================
 *
 * Archivo: HWPrincipal.java
 */

package com.ibm.security;

import java.security.Principal;

/**
 * Esta clase implementa la interfaz Principal
 * y representa un comprobador de HelloWorld.
 *
 * @version 1.1, 09/10/99
 * @author D. Kent Soper
 */
public class HWPrincipal implements Principal, java.io.Serializable {

    private String name;

    /*
     * Crear un HWPrincipal con el nombre suministrado.
     */
    public HWPrincipal(String name) {
        if (name == null)
            throw new NullPointerException("entrada null no permitida");

        this.name = name;
    }

    /*
     * Devolver el nombre del HWPrincipal.
     */
    public String getName() {
        return name;
    }
```

```
    /*
     * Devolver una representación de tipo serie del HWPrincipal.
     */
    public String toString() {
        return("HWPrincipal:  " + name);
    }

    /*
     * Compara el objeto (Object) especificado con el HWPrincipal para ver si son iguales.
     * Devuelve true si el objeto dado también es un HWPrincipal y los
     * dos HWPrincipals tienen el mismo nombre de usuario.
     */
    public boolean equals(Object o) {
        if (o == null)
            return false;

        if (this == o)
            return true;

        if (!(o instanceof HWPrincipal))
            return false;
        HWPrincipal that = (HWPrincipal)o;

        if (this.getName().equals(that.getName()))
            return true;
        return false;
    }

    /*
     * Devolver un código hash para el HWPrincipal.
     */
    public int hashCode() {
        return name.hashCode();
    }
}
```

Ejemplo: SampleThreadSubjectLogin de JAAS

Este ejemplo muestra la implementación de la clase SampleThreadSubjectLogin.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
////////////////////////////////////////////////////////////////////////
//
// 5761-JV1
//
////////////////////////////////////////////////////////////////////////
//
// Nombre de archivo:   SampleThreadSubjectLogin.java
//
// Clase:               SampleThreadSubjectLogin
//
////////////////////////////////////////////////////////////////////////
//
// ACTIVIDAD DE CAMBIO:
//
//
// FIN DE ACTIVIDAD DE CAMBIO
//
////////////////////////////////////////////////////////////////////////

import com.ibm.security.auth.ThreadSubject;

import com.ibm.as400.access.*;
```

```
import java.io.*;

import java.util.*;

import java.security.Principal;

import javax.security.auth.*;

import javax.security.auth.callback.*;

import javax.security.auth.login.*;


/**
  * Esta aplicación SampleThreadSubjectLogin autentica a un solo
  * usuario, intercambia la identidad de hebra de OS por el usuario autenticado
  * y después escribe "Hello World" en un archivo con autorización privada,
  * llamado thread.txt, situado en el directorio de pruebas (test) del usuario.
 *
  * Se solicita al usuario que entre el ID de usuario y la contraseña
  * que hay que autenticar.
 *
  * Si ello es satisfactorio, el nombre del usuario y el número de credenciales
  * se visualizan.
 *
 *

  Instrucciones de configuración y ejecución:

  1) Cree un usuario nuevo, JAAS14, invocando
 "CRTUSRPRF USRPRF(JAAS14) PASSWORD() TEXT('ID de usuario de ejemplo para JAAS')"
  con la autorización de clase *USER.

 2) Asigne un archivo de prueba ficticio, "suDirPruebas/thread.txt", y
 otorgue de manera privada la autorización *RWX de JAAS13 sobre él para acceso de escritura.

  3) Copie SampleThreadSubjectLogin.java en su directorio de pruebas.

  4) Cambie el directorio actual por el directorio de pruebas y compile el
  código fuente Java.

   Entre:

   strqsh

   cd 'suDirPruebas'

   javac -J-Djava.version=1.4
     -classpath /qibm/proddata/os400/java400/ext/jaas14.jar:
                /QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar:.
     -d ./classes
     *.java

  5) Copie los archivos threadLogin.config, threadJaas.policy y threadJava2.policy
  en su directorio de pruebas.

  6) Si aún no se ha hecho, añada el enlace simbólico al directorio de
  extensiones del archivo jaas14.jar.
  El cargador de clases de extensión debe cargar normalmente el archivo JAR.

 ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas14.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jaas14.jar')


  7) Si aún no se ha hecho para ejecutar este ejemplo, añada el enlace simbólico al directorio
  de extensiones correspondiente a los archivos jt400.jar y jt400ntv.jar. Ello hará que
  el cargador de clases de extensión cargue esos archivos. El cargador de clases de aplicación
```

```
  también puede cargar dichos archivos si se les incluye en la variable CLASSPATH.
  Si esos archivos se cargan a partir del directorio de vía de acceso de clases,
  no añada el enlace simbólico al directorio de extensiones.
  El archivo jaas14.jar necesita estos archivos JAR para las clases de
  implementación de credenciales que forman parte del programa producto
  bajo licencia Java (5761-JC1).
  (En el tema acerca de IBM Toolbox para Java hallará documentación
  relacionada con las clases de credenciales situadas en el marco izquierdo
  bajo Clases de seguridad => Autenticación. Seleccione el enlace de acceso a la
  clase ProfileTokenCredential. En la parte superior, seleccione 'Este paquete' para
  obtener todo el paquete Java com/ibm/as400/security/auth. Para obtener el Javadoc
  de las clases de autenticación, puede seleccionar 'Javadoc' =>
  'Clases de acceso' en el marco izquierdo. Seleccione 'Todos los paquetes' en la parte
  superior y localice los paquetes com.ibm.as400.security.*).


 ADDLNK OBJ('/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400.jar')

 ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')

////////////////////////////////////
  NOTAS IMPORTANTES -
////////////////////////////////////

  Si va a actualizar los archivos de política de Java2
  para una aplicación real, no olvide otorgar los debidos
  permisos sobre las ubicaciones reales de los archivos jar de
  IBM Toolbox para Java. Aunque estos archivos estén simbólicamente enlazados
  con los directorios de extensiones anteriores a los que se ha otorgado
  java.security.AllPermission en el archivo
 ${java.home}/lib/security/java.policy file,la autorización se basa en
  la ubicación real de los archivos JAR.

  Por ejemplo, para utilizar satisfactoriamente las clases de credenciales
  de IBM Toolbox para Java, se añadiría el siguiente fragmento de código al
  archivo de política Java2 de la aplicación:

 grant  codeBase "file:/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar"
 {
   permission javax.security.auth.AuthPermission "modifyThreadIdentity";
   permission java.lang.RuntimePermission "loadLibrary.*";
   permission java.lang.RuntimePermission "writeFileDescriptor";
   permission java.lang.RuntimePermission "readFileDescriptor";
 }

  También tendrá que añadir estos permisos para el parámetro
  codeBase de la aplicación, ya que las operaciones efectuadas por los
  archivos JAR de IBM Toolbox para Java no se ejecutan en modalidad privilegiada.

  Este ejemplo ya otorga estos permisos a todas las clases Java
  al omitir el parámetro codeBase del archivo threadJava2.policy.

  8) Asegúrese de que se han iniciado los servidores de sistema principal
 y que están funcionando.
 Las clases ProfileTokenCredential que
residen en IBM Toolbox para Java,
 es decir jt400.jar, se emplean como
credenciales conectadas
 al sujeto autenticado mediante el programa SampleThreadSubjectLogin.java.
 Las clases de credenciales de IBM Toolbox para Java necesitan acceso
 a los servidores de host.

  9) Invoque SampleThreadSubjectLogin mientras esté conectado como usuario que
 no tiene acceso a 'suDirPruebas/thread.txt'.
```

```
 10) Inicie el ejemplo entrando los siguientes mandatos CL  =>

 CHGCURDIR DIR('suDirPruebas')

 JAVA CLASS(SampleThreadSubjectLogin)
  CLASSPATH('suDirPruebas/classes')
  PROP((java.version '1.3')
       (java.security.manager)
       (java.security.auth.login.config
        'suDirPruebas/threadLogin.config')
       (java.security.policy
        'suDirPruebas/threadJava2.policy')
       (java.security.auth.policy
        'suDirPruebas/threadJaas.policy'))

 Cuando se le solicite, entre el ID de usuario y la contraseña del paso 1.

 11) Busque en suDirPruebas/thread.txt la entrada "Hello World".

 *
 **/


public class SampleThreadSubjectLogin {
/**
 * Se intenta autenticar al usuario.
 *
 * @param args
 *      Argumentos de entrada para esta aplicación (se pasan por alto).
 *
 */
    public static void main(String[] args) {

    // Usar los LoginModules configurados para la entrada "AS400ToolboxApp".
    LoginContext lc = null;
    try {
        // Si se proporciona, se usará el mismo sujeto para varios intentos de inicio de sesión.
        lc = new LoginContext("AS400ToolboxApp",
                  new Subject(),
                  new SampleCBHandler());
    } catch (LoginException le) {
        le.printStackTrace();
        System.exit(-1);
    }

    // El usuario tiene 3 intentos para autenticarse satisfactoriamente.
    int i;
    for (i = 0; i < 3; i++) {
        try {

        // intentar autenticación
        lc.login();

        // Si no se devuelve ninguna excepción,
        // la autenticación ha sido satisfactoria.
        break;

        } catch (AccountExpiredException aee) {

        System.out.println("Su cuenta ha caducado");
        System.exit(-1);

        } catch (CredentialExpiredException cee) {

        System.out.println("Sus credenciales han caducado.");
        System.exit(-1);
```

```
    } catch (FailedLoginException fle) {

    System.out.println("Autenticación fallida");
    try {
        Thread.currentThread().sleep(3000);
    } catch (Exception e) {
        // Ignorar
    }

    } catch (Exception e) {

    System.out.println("Excepción no prevista - imposible continuar");
    e.printStackTrace();
    System.exit(-1);
    }
}

// ¿Han fallado tres veces?
if (i == 3) {
    System.out.println("Lo lamentamos, la autenticación ha fallado");
    System.exit(-1);
}


// Visualizar los Principales autenticados y las credenciales.
System.out.println("Autenticación satisfactoria");

System.out.println("Principales:");

Iterator itr = lc.getSubject().getPrincipals().iterator();

while (itr.hasNext())
    System.out.println(itr.next());

itr = lc.getSubject().getPrivateCredentials().iterator();

while (itr.hasNext())
    System.out.println(itr.next());

itr = lc.getSubject().getPublicCredentials().iterator();

while (itr.hasNext())
    System.out.println(itr.next());


     // Hacer algún trabajo basado en Principales:
ThreadSubject.doAsPrivileged(lc.getSubject(), new java.security.PrivilegedAction() {
       public Object run() {
       System.out.println("\nSu propiedad java.home: "
              +System.getProperty("java.home"));
       System.out.println("\nSu propiedad user.home: "
                  +System.getProperty("user.home"));
       File f = new File("thread.txt");
       System.out.print("\nthread.txt ");
       if (!f.exists()) System.out.print("no ");
       System.out.println("existe en el directorio actual");

       try {
           // Escribir "Hello World número   x"  en thread.txt.
           PrintStream ps = new PrintStream(new FileOutputStream("thread.txt", true), true);

           long flen = f.length();
           ps.println("Hello World número    " +
              Long.toString(flen/22) +
              "\n");
           ps.close();
       } catch (Exception e) {
```

```
            e.printStackTrace();
        }

        System.out.println("\nPor cierto, " + SampleThreadSubjectLogin.getCurrentUser());
        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
         // Ignorar
            }
        System.out.println("\n\nHello World!\n");
        return null;
        }
      }, null);

      System.exit(0);

   }// Finalizar main().


// Devuelve la identidad de OS actual de la hebra principal de la aplicación.
// (Esta rutina utiliza clases de IBM Toolbox para Java)
// Nota: las aplicaciones que se ejecutan en una hebra secundaria no
// pueden emplear esta API para determinar el usuario actual.
    static public String getCurrentUser() {

    try {
        AS400 localSys = new AS400("localhost", "*CURRENT", "*CURRENT");

        int ccsid = localSys.getCcsid();
        ProgramCall qusrjobi = new ProgramCall(localSys);
        ProgramParameter[] parms = new ProgramParameter[6];

        int rLength = 100;
        parms[0] = new ProgramParameter(rLength);
        parms[1] = new ProgramParameter(new AS400Bin4().toBytes(rLength));
        parms[2] = new ProgramParameter(new AS400Text(8, ccsid, localSys).toBytes("JOBI0600"));
        parms[3] = new ProgramParameter(new AS400Text(26,ccsid, localSys).toBytes("*"));
        parms[4] = new ProgramParameter(new AS400Text(16,ccsid, localSys).toBytes(""));
        parms[5] = new ProgramParameter(new AS400Bin4().toBytes(0));

        qusrjobi.setProgram(QSYSObjectPathName.toPath("QSYS", "QUSRJOBI", "PGM"), parms);
        AS400Text uidText = new AS400Text(10, ccsid, localSys);

    // Invocar la API QUSRJOBI.
        qusrjobi.run();

        byte[] uidBytes = new byte[10];
        System.arraycopy((qusrjobi.getParameterList())[0].getOutputData(), 90, uidBytes, 0, 10);

        return ((String)(uidText.toObject(uidBytes))).trim();
    }


    catch (Exception e) {
        e.printStackTrace();
    }

    return "";
    }

} //Finalizar la clase SampleThreadSubjectLogin.



/**
 * Se pasa un CallbackHandler a los servicios de seguridad
 * subyacentes para que puedan interaccionar con la aplicación
```

```
 * con el fin de recuperar datos de autenticación específicos,
 * como los nombres de usuario y las contraseñas, o con el fin
 * de visualizar información concreta, como los mensajes de error
 * y de aviso.
 *
 * CallbackHandlers se implementan de forma dependiente
 * de aplicación y de plataforma. La implementación decide
 * cómo hay que recuperar y visualizar la información en función
 * de los retornos de llamada (Callback) que se le pasan.
 *
 * Esta clase proporciona un CallbackHandler de ejemplo para las aplicaciones
 * en funcionamiento en un entornoi5/OS. Sin embargo, no se pretende
 * satisfacer los requisitos de las aplicaciones en la fase de producción.
 * Según se ha indicado, el CallbackHandler se considera en último término
 * como dependiente de la aplicación, pues las aplicaciones individuales
 * tienen sus propios requisitos de comprobación de errores, manejo de datos
 * e interfaz de usuario.
 *
 * Se manejan los siguientes retornos de llamada (callback):
 *
```

- *
- NameCallback *
- PasswordCallback *
- TextOutputCallback *

```
 *
 * A efectos de simplicidad, las solicitudes se manejan de forma interactiva
 * por medio de la entrada y la salida estándar. Sin embargo, cabe mencionar
 * que, cuando la consola proporciona entrada estándar, este
 * enfoque permitirá que se vean las contraseñas según se van
 * tecleando. Este aspecto se debe evitar en el caso de las
 * aplicaciones en fase de producción.
 *
 * Este CallbackHandler también permite adquirir un nombre
 * y una contraseña mediante un mecanismo alternativo
 * y establecerlos directamente en el manejador para eludir
 * la necesidad de la interacción de usuario en los
 * correspondientes retornos de llamada.
 *
 */
class SampleCBHandler implements CallbackHandler {
    private String name_ = null;
    private String password_ = null;
/**
 * Construye un nuevo SampleCBHandler.
 *
 */
public SampleCBHandler() {
    this(null, null);
}
/**
 * Construye un nuevo SampleCBHandler.
 *
 * Se puede especificar opcionalmente un nombre y una contraseña
 * con el fin de eludir la necesidad de solicitar información
 * en los correspondientes retornos de llamada.
 *
 * @param name
 *      Valor predeterminado de los retornos de llamada del nombre.
 *      El valor nulo indica que hay que solicitar esta
 *      información al usuario. Los valores no nulos deben
 *      tener una longitud mayor que cero, sin superar los 10 caracteres.
 *
 * @param password
 *      Valor predeterminado de los retornos de llamada de la contraseña.
 *      El valor nulo indica que hay que solicitar esta
```

```
 *      información al usuario. Los valores no nulos deben
 *      tener una longitud mayor que cero, sin superar los 10 caracteres.
 */
public SampleCBHandler(String name, String password) {
    if (name != null)
        if ((name.length()==0) || (name.length()>10))
            throw new IllegalArgumentException("name");
    name_ = name;

    if (password != null)
        if ((password.length()==0) || (password.length()>10))
            throw new IllegalArgumentException("password");
    password_ = password;
}
/**
 * Manejar el retorno de llamada del nombre dado.
 *
 * En primer lugar, se comprueba si se ha pasado un nombre
 * en el constructor. Si es así, se asigna el nombre al
 * retorno de llamada y se elude la solicitud.
 *
 * Si no se ha preestablecido ningún valor, se intenta solicitar
 * el nombre utilizando la entrada y la salida estándar.
 *
 * @param c
 *      El retorno de llamada del nombre (NameCallback).
 *
 * @exception java.io.IOException
 *      Si se produce un error de entrada o de salida.
 *
 */
private void handleNameCallback(NameCallback c) throws IOException {
    // Comprobar si existe un valor en caché.
    if (name_ != null) {
        c.setName(name_);
        return;
    }
    // No hay ningún valor preestablecido; se intenta stdin/out.
    c.setName(
        stdIOReadName(c.getPrompt(), 10));
}
/**
 * Manejar el retorno de llamada del nombre dado.
 *
 * En primer lugar, se comprueba si se ha pasado una contraseña
 * en el constructor. Si es así, se asigna la contraseña al
 * retorno de llamada y se elude la solicitud.
 *
 * Si no se ha preestablecido ningún valor, se intenta solicitar
 * la contraseña utilizando la entrada y la salida estándar.
 *
 * @param c
 *      El retorno de llamada de la contraseña (PasswordCallback).
 *
 * @exception java.io.IOException
 *      Si se produce un error de entrada o de salida.
 *
 */
private void handlePasswordCallback(PasswordCallback c) throws IOException {
    // Comprobar si existe un valor en caché.
    if (password_ != null) {
        c.setPassword(password_.toCharArray());
        return;
    }

    // No hay ningún valor preestablecido; se intenta stdin/out.
    // Nota:  Uso no adecuado para producción.
```

```
    //        La consola estándar de E/S no esconde la contraseña.
    if (c.isEchoOn())
        c.setPassword(
            stdIOReadName(c.getPrompt(), 10).toCharArray());
    else
    {

        //  Nota: la consola estándar de E/S no esconde la contraseña.
        c.setPassword(stdIOReadName(c.getPrompt(), 10).toCharArray());

    }
}
/**
 * Manejar el retorno de llamada de la salida de texto dada.
 *
 * Si el texto es informativo o representa un aviso,
 * se escribe en la salida estándar. Si el
 * retorno de llamada define un mensaje de error, el texto
 * se escribe en la salida de error estándar.
 *
 * @param c
 *      El retorno de llamada de la salida de texto (TextOutputCallback).
 *
 * @exception java.io.IOException
 *      Si se produce un error de entrada o de salida.
 *
 */
private void handleTextOutputCallback(TextOutputCallback c) throws IOException {
    if (c.getMessageType() == TextOutputCallback.ERROR)
        System.err.println(c.getMessage());
    else
        System.out.println(c.getMessage());
}
/**
 * Recuperar o visualizar la información solicitada en
 * los retornos de llamada proporcionados.
 *
 * La implementación del método handle
 * comprueba la(s) instancia(s) del objeto Callback
 * (uno o varios) pasado para recuperar o visualizar la
 * información solicitada.
 *
 * @param callbacks
 *      Una matriz de objetos Callback proporcionada
 *      por un servicio de seguridad subyacente y que contiene
 *      la información solicitada que hay que recuperar o visualizar.
 *
 * @exception java.io.IOException
 *      Si se produce un error de entrada o de salida.
 *
 * @exception UnsupportedCallbackException
 *      Si la  implementación de este método no da soporte a
 *      uno o varios de los objetos Callback especificados en
 *      el parámetro callbacks.
 *
 */
public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException
{
    for (int i=0; i<callbacks.length; i++) {
        Callback c = callbacks[i];

        if (c instanceof NameCallback)
            handleNameCallback((NameCallback)c);
        else if (c instanceof PasswordCallback)
            handlePasswordCallback((PasswordCallback)c);
        else if (c instanceof TextOutputCallback)
```

```
            handleTextOutputCallback((TextOutputCallback)c);
        else
            throw new UnsupportedCallbackException
                (callbacks[i]);
    }
}
/**
 * Visualiza la serie dada utilizando la salida estándar,
 * seguida de un espacio para separar de la próxima
 * entrada.
 *
 * @param prompt
 *      El texto que hay que visualizar.
 *
 * @exception IOException
 *      Si se produce un error de entrada o de salida.
 *
 */
private void stdIOPrompt(String prompt) throws IOException {
    System.out.print(prompt + ' ');
    System.out.flush();
}
/**
 * Lee una serie de la entrada estándar, terminada según el valor
 * de maxLength o mediante un carácter de nueva línea.
 *
 * @param prompt
 *      El texto que hay que visualizar inmediatamente en la salida estándar
 *      antes de leer el valor solicitado.
 *
 * @param maxLength
 *      Longitud máxima de la serie que hay que devolver.
 *
 * @return
 *      La serie entrada. El valor devuelto no
 *      contendrá espacio en blanco al principio ni al final
 *      y se convertirá a mayúsculas.
 *
 * @exception IOException
 *      Si se produce un error de entrada o de salida.
 *
 */
private String stdIOReadName(String prompt, int maxLength) throws IOException {
    stdIOPrompt(prompt);
    String s =
        (new BufferedReader
            (new InputStreamReader(System.in))).readLine().trim();
    if (s.length() < maxLength)
        s = s.substring(0,maxLength);
    return s.toUpperCase();
}

}// Finalizar la clase SampleCBHandler.
```

Ejemplo: programa cliente IBM JGSS no JAAS

Utilice este cliente de ejemplo JGSS junto con el servidor de ejemplo JGSS.

Para obtener más información sobre cómo utilizar el programa cliente de ejemplo, vea: “Ejemplos: descargar y ejecutar los programas JGSS de ejemplo” en la página 410.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
// Programa cliente de ejemplo de 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 cliente de ejemplo de JGSS
  * para utilizarse junto con el servidor de ejemplo de JGSS.
  * El cliente primero establece un contexto con el servidor
  * y después envía un mensaje envuelto seguido de un MIC al servidor.
  * El MIC se calcula a partir del texto sin formato envuelto.
  * El cliente requiere que el servidor se autentique
  * (autenticación mutua) durante el establecimiento del contexto.
  * También delega sus credenciales al servidor.
 *
  * Establece la variable JAVA
  * javax.security.auth.useSubjectCredsOnly en false
  * por lo que JGSS no adquirirá credenciales por medio de JAAS.
 *
  * El cliente toma parámetros de entrada y los complementa
  * con información del archivo jgss.ini; la entrada necesaria no
  * proporcionada en la línea de mandatos se toma del archivo jgss.ini.
 *
 * Uso: Client [opciones]
 *
 * La opción -? genera un mensaje de ayuda que contiene las opciones soportadas.
 *
 * Este cliente de ejemplo no utiliza JAAS.
 * El cliente se puede ejecutar para el cliente y el servidor de ejemplo de JAAS.
 * Vea en {@link JAASClient JAASClient} un cliente de ejemplo que emplea JAAS.
 */

class Client
{
    private Util testUtil      = null;
    private String myName      = null;
    private GSSName gssName    = null;
    private String serverName = null;
    private int servicePort    = 0;
    private GSSManager mgr     = GSSManager.getInstance();
    private GSSName service    = null;
    private GSSContext context        = null;
    private String program     = "Client";
    private String debugPrefix = "Client: ";
    private TCPComms tcp       = null;
    private String data               = null;
    private byte[] dataBytes   = null;
    private String serviceHostname= null;
    private GSSCredential gssCred = null;

    private static Debug debug        = new Debug();

    private static final String usageString =
          "\t[-?] [-d | -n nombre] [-s nombreServidor]"
        + "\n\t[-h hostServidor [:puerto]] [-p puerto] [-m msg]"
        + "\n"
        + "\n  -?\t\t\thelp; produce este mensaje"
        + "\n  -n name\t\tnombre del principal del cliente (sin reino)"
        + "\n  -s serverName\t\tnombre del principal del servidor (sin reino)"
        + "\n  -h serverHost[:port]\tnombre de host del servidor"
        +         " (y número de puerto opcional)"
        + "\n  -p port\t\tpuerto en el que estará a la escucha el servidor"
```

```
    + "\n  -m msg\t\tmensaje que hay que enviar al servidor";


// El llamador debe llamar a initialize (puede que primero tenga que llamar a processArgs).
public Client (String programName) throws Exception
{
    testUtil = new Util();
    if (programName != null)
    {
        program = programName;
        debugPrefix = programName + ": ";
    }
}

// El llamador debe llamar a initialize (puede que primero tenga que llamar a processArgs).
Client (String programName, boolean useSubjectCredsOnly) throws Exception
{
    this(programName);
    setUseSubjectCredsOnly(useSubjectCredsOnly);
}

public Client(GSSCredential myCred,
              String serverNameWithoutRealm,
              String serverHostname,
              int serverPort,
              String message)
    throws Exception
{
    testUtil = new Util();

    if (myCred != null)
    {
        gssCred = myCred;
    }
    else
    {
        throw new GSSException(GSSException.NO_CRED, 0,
                               "Null input credential");
    }

    init(serverNameWithoutRealm, serverHostname, serverPort, message);
}

void setUseSubjectCredsOnly(boolean useSubjectCredsOnly)
{
    final String subjectOnly = useSubjectCredsOnly ? "true" : "false";
    final String property = "javax.security.auth.useSubjectCredsOnly";

    String temp = (String)java.security.AccessController.doPrivileged(
                  new sun.security.action.GetPropertyAction(property));

    if (temp == null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
          + "setting useSubjectCredsOnly property to "
          + useSubjectCredsOnly);

        // Propiedad no establecida. Establecerla en el valor especificado.

        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
{
    // nombre del igual
    if (serverNameWithoutRealm != null)
    {
        this.serverName = serverNameWithoutRealm;
    }
    else
    {
        this.serverName = testUtil.getDefaultServicePrincipalWithoutRealm();
    }

    // host del igual
    if (serverHostname != null)
    {
        this.serviceHostname = serverHostname;
    }
    else
    {
        this.serviceHostname = testUtil.getDefaultServiceHostname();
    }

    // puerto del igual
    if (serverPort > 0)
    {
        this.servicePort = serverPort;
    }
    else
    {
        this.servicePort = testUtil.getDefaultServicePort();
    }

    // mensaje para el igual
    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; // para credenciales por omisión


        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");
    }
    // Añadir más posteriormente
}

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( "Entrada de puerto incorrecta: "+port);
        }

        if (p != -1)
```

```
                servicePort = p;
        }

        if ((sHost != null) && (sHost.length() > 0)) {
                serviceHostname = sHost;
        }

        init(myName, serverName, serviceHostname, servicePort, msg);
    }

    void interactWithAcceptor(BitSet flags) throws Exception
    {
        establishContext(flags);
        doWrap();
        doMIC();
    }

    void interactWithAcceptor() throws Exception
    {
        BitSet flags = new BitSet();
        flags.set(Util.CONTEXT_OPTS_MUTUAL);
        flags.set(Util.CONTEXT_OPTS_CONF);
        flags.set(Util.CONTEXT_OPTS_INTEG);
        flags.set(Util.CONTEXT_OPTS_DELEG);
        interactWithAcceptor(flags);
    }

    void dispose() throws Exception
    {
        if (tcp != null)
        {
            tcp.close();
        }
    }

    public static void main(String args[]) throws Exception
    {
        System.out.println(debug.toString()); // XXXXXXX
        String programName = "Client";
        Client client = null;
        try {
            client = new Client(programName,
                                false); // no utilizar credenciales del sujeto
            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");
    }
}
```

Ejemplo: programa servidor IBM JGSS no JAAS

Este es un ejemplo de un servidor JGSS que se debe usar juntamente con un cliente JGSS de ejemplo.

Para obtener más información sobre cómo utilizar el programa servidor de ejemplo, vea: “Ejemplos: descargar y ejecutar los programas JGSS de ejemplo” en la página 410.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
// Programa servidor de ejemplo de 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 servidor JGSS de ejemplo que hay que usar con un cliente JGSS de ejemplo.
 *
 * Está continuamente a la escucha de conexiones de cliente
 * y engendra una hebra para dar servicio a una conexión entrante.
 * Puede ejecutar varias hebras de forma concurrente.
 * Dicho de otro modo, puede servir a varios clientes a la vez.
 *
 * Cada hebra establece primero un contexto con el cliente
 * y después espera un mensaje con envoltura seguido de un MIC.
 * Se supone que el cliente ha calculado el MIC a partir del texto
 * plano envuelto por el cliente.
 *
 * Si el cliente delega su credencial al servidor, la credencial
 * delegada se utiliza para la comunicación con un servidor secundario.
 *
 * Asimismo, el servidor puede iniciarse para actuar como cliente y
 * servidor (con la opción -b). En este caso, la primera
 * hebra engendrada por el servidor usa la credencial propia del principal
 * del servidor para comunicarse con el servidor secundario.
 *
 * El servidor secundario debe haberse iniciado antes que el servidor (primario)
 * inicie el contacto con él (el servidor secundario).
 * En la comunicación con el servidor secundario, el servidor primario actúa
 * como iniciador de JGSS (es decir, cliente) que establece un contexto y
 * participa en intercambios de envoltura y MIC por mensaje con el servidor secundario.
 *
 * El servidor toma parámetros de entrada y los complementa
 * con información del archivo jgss.ini; la entrada necesaria no
 * proporcionada en la línea de mandatos se toma del archivo jgss.ini.
 * Se emplean valores predeterminados incorporados si no hay un archivo jgss.ini
 * o si una variable determinada no está especificada en el archivo ini.
 *
 * Uso: Server [opciones]
 *
 * La opción -? genera un mensaje de ayuda que contiene las opciones soportadas.
 *
 * Este servidor de ejemplo no utiliza JAAS.
 * Establece la variable JAVA
 * javax.security.auth.useSubjectCredsOnly en false
 * por lo que JGSS no adquirirá credenciales por medio de JAAS.
 * El servidor se puede ejecutar para los clientes y servidores JAAS de ejemplo.
 * Vea en {@link JAASServer JAASServer} un servidor de ejemplo que emplea JAAS.
 */

class Server implements Runnable
{
    /*
     * NOTAS:
     * Está previsto que esta clase, Server, se ejecute en varias
     * hebras concurrentes. Las variables estáticas constan de variables
```

```
 * establecidas desde argumentos de línea de mandatos y variables (como
 * las propias credenciales del servidor, gssCred) establecidas una vez
 * durante la inicialización. Estas variables no cambian
 * una vez establecidas y se comparten entre todas las hebras en ejecución.
 *
 * La única variable estática que cambia tras establecerse inicialmente
 * es la variable beenInitiator, que la establece en true
 * la primera hebra que ejecuta el servidor como iniciador utilizando
 * las credenciales propias del servidor. Ello garantiza que el servidor
 * solo se ejecuta como iniciador una vez. La consulta y la modificación
 * de beenInitiator están sincronizadas entre las hebras.
 *
 * La variable tcp no es estática y se establece para cada hebra
 * a fin de representar el socket en el se sirve al cliente
 * mediante la hebra conectada.
 */

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[-?] [-# número] [-d | -n nombre] [-p puerto]"
   + "\n\t[-s nombreServidor] [-h hostServidor [:port]] [-P puertoServidor] [- msg]"
   + "\n"
   + "\n  -?\t\t\thelp; produce este mensaje"
   + "\n  -# number\t\tSi es el servidor primario o secundario"
   +         " \n\t\t\t(1 = primario, 2 = secundario; default = primero)"
   + "\n  -n name\t\tnombre del principal del servidor (sin reino)"
   + "\n  -p port\t\tpuerto en el que estará a la escucha el servidor"
   + "\n  -s serverName\t\tnombre del principal del servidor secundario"
   +         " (sin reino)"
   + "\n  -h serverHost[:port]\tnombre de host del servidor secundario"
   +         " (y número de puerto opcional)"
   + "\n  -P port\t\tnúmero de puerto del servidor secundario"
   + "\n  -m msg\t\tmensaje que hay que enviar al servidor secundario"
   + "\n  -b \t\tejecutar como cliente y servidor"
   +         " con las credenciales propias del servidor";

// Las variables no estáticas son específicas de las hebras
// ya que cada hebra ejecuta una instancia aparte de esta clase.

private String debugPrefix = null;
private TCPComms tcp       = null;

static {
    try {
        testUtil = new Util();
    } catch (Exception exc) {
        exc.printStackTrace();
        System.exit(1);
    }
}
```

```
Server (Socket socket) throws Exception
{
    debugPrefix = program + ": ";
    tcp = new TCPComms(socket);
}

Server (String program) throws Exception
{
    debugPrefix = program + ": ";
    this.program = program;
}

Server (String program, boolean useSubjectCredsOnly) throws Exception
{
    this(program);
    setUseSubjectCredsOnly(useSubjectCredsOnly);
}

void setUseSubjectCredsOnly(boolean useSubjectCredsOnly)
{
    final String subjectOnly = useSubjectCredsOnly ? "true" : "false";
    final String property = "javax.security.auth.useSubjectCredsOnly";

    String temp = (String)java.security.AccessController.doPrivileged(
                    new sun.security.action.GetPropertyAction(property));

    if (temp == null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
          + "setting useSubjectCredsOnly property to "
          + (useSubjectCredsOnly ? "true" : "false"));

        // Propiedad no establecida. Establecerla en el valor especificado.

        java.security.AccessController.doPrivileged(
             new java.security.PrivilegedAction() {
               public Object run() {
                  System.setProperty(property, subjectOnly);
                  return null;
               }
             });
    }
    else
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
          + "useSubjectCredsOnly property already set "
          + "in JVM  to " + temp);
    }
}

private void init(boolean  primary,
          String   myNameWithoutRealm,
          int      port,
          String   serverNameWithoutRealm,
          String   serverHostname,
          int      serverPort,
          String   message,
          boolean  clientServer)
    throws Exception
{
    primaryServer = primary;
    this.clientServer = clientServer;

    myName = myNameWithoutRealm;

    // mi puerto
    if (port > 0)
```

```
{
    myPort = port;
}
else if (primary)
{
    myPort = testUtil.getDefaultServicePort();
}
else
{
    myPort = testUtil.getDefaultService2Port();
}

if (primary)
{
    ///// nombre del igual
    if (serverNameWithoutRealm != null)
    {
        serviceNameNoRealm = serverNameWithoutRealm;
    }
    else
    {
        serviceNameNoRealm =
              testUtil.getDefaultService2PrincipalWithoutRealm();
    }

    // host del igual
    if (serverHostname != null)
    {
        if (serverHostname.equalsIgnoreCase("localHost"))
        {
            serverHostname = InetAddress.getLocalHost().getHostName();
        }

        serviceHost = serverHostname;
    }
    else
    {
        serviceHost = testUtil.getDefaultService2Hostname();
    }

    // puerto del igual
    if (serverPort > 0)
    {
        servicePort = serverPort;
    }
    else
    {
        servicePort = testUtil.getDefaultService2Port();
    }

    // mensaje para el igual
    if (message != null)
    {
        serviceMsg = message;
    }
    else
    {
        serviceMsg = "Hi there! I am a server."
                      + "But I can be a client, too";
    }
}

guitaString temp = debugPrefix + "details"
              + "\n\tPrimary:\t" + primary
              + "\n\tName:\t\t" + myName
              + "\n\tPort:\t\t" + myPort
              + "\n\tClient+server:\t" + clientServer;
```

```
    if (primary)
    {
        temp += "\n\tOther Server:"
                  + "\n\t\tName:\t" + serviceNameNoRealm
                  + "\n\t\tHost:\t" + serviceHost
                  + "\n\t\tPort:\t" + servicePort
                  + "\n\t\tMsg:\t" + serviceMsg;
    }

    debug.out(Debug.OPTS_CAT_APPLICATION, temp);
}


void initialize() throws GSSException
{
   debug.out(Debug.OPTS_CAT_APPLICATION,
                     debugPrefix + "creating GSSManager");

    mgr = GSSManager.getInstance();

    int usage = clientServer ? GSSCredential.INITIATE_AND_ACCEPT
                             : GSSCredential.ACCEPT_ONLY;

    if (myName != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                           + "creating GSSName for " + myName);

        gssName = mgr.createName(myName,
                                 GSSName.NT_HOSTBASED_SERVICE);

        Oid krb5MechanismOid = new Oid("1.2.840.113554.1.2.2");
        gssName.canonicalize(krb5MechanismOid);

        debug.out(Debug.OPTS_CAT_APPLICATION,
              debugPrefix + "Canonicalized GSSName=" + gssName);
    }
    else
        gssName = null;

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "creating"
           + ((gssName == null)? " default " : " ")
           + "credential");

    gssCred = mgr.createCredential(
                           gssName, GSSCredential.DEFAULT_LIFETIME,
                           (Oid)null, usage);
    if (gssName == null)
    {
        gssName = gssCred.getName();
        myName = gssName.toString();

        debug.out(Debug.OPTS_CAT_APPLICATION,
           debugPrefix + "default credential principal=" + myName);
    }
}



void processArgs(String[] args) throws Exception
{
    String port    = null;
    String name    = null;
    int iport      = 0;

    String sport   = null;
    int isport     = 0;
```

```
String sname   = null;
String shost   = null;
String smessage        = null;

boolean primary = true;
String status  = null;

boolean defaultPrinc = false;
boolean clientServer = false;

GetOptions options = new GetOptions(args, "?#:p:n:P:s:h:m:b");
int ch = -1;
while ((ch = options.getopt()) != options.optEOF)
{
    switch(ch)
    {
        case '?':
            printUsage();
            System.exit(1);

        case '#':
            if (status == null)
                status = options.optArgGet();
            continue;

        case 'p':
            if (port == null)
                port = options.optArgGet();
            continue;

        case 'n':
            if (name == null)
                name = options.optArgGet();
            continue;

        case 'b':
            clientServer = true;
            continue;

        ////// El otro servidor

        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: Las opciones '-d' y '-n ' se excluyen mutuamente");
        printUsage();
        System.exit(1);
    }

    if (status != null)
    {
        int p = -1;
        try {
            p = Integer.parseInt(status);
        } catch (Exception exc) {
            System.out.println( "Entrada de estado incorrectat: "+status);
        }

        if (p != -1)
        {
            primary = (p == 1);
        }
    }

    if (port != null)
    {
        int p = -1;
        try {
            p = Integer.parseInt(port);
        } catch (Exception exc) {
            System.out.println( "Entrada de puerto incorrecta: "+port);
        }
        if (p != -1)
            iport = p;
    }

    if (sport != null)
    {
        int p = -1;
        try {
            p = Integer.parseInt(sport);
        } catch (Exception exc) {
            System.out.println( "Entrada de puerto de servidor incorrecta: "+port);
        }
        if (p != -1)
            isport = p;
    }

    init(primary,  // el servidor primero o segundo
         name,     // mi nombre
         iport,    // mi puerto
         sname,    // el nombre del otro servidor
         shost,    // el nombre de host del otro servidor
         isport,   // el puerto del otro servidor
         smessage, // mensaje para el otro servidor
         clientServer); // si se ejecutará como iniciador con credenciales propias
}

void processRequests() throws Exception
{
    ServerSocket ssocket = null;
```

```
    Server server = null;
    try {
        ssocket = new ServerSocket(myPort);
        do {
           debug.out(Debug.OPTS_CAT_APPLICATION,
                 debugPrefix + "listening on port " + myPort + " ...");
            Socket csocket = ssocket.accept();

           debug.out(Debug.OPTS_CAT_APPLICATION,
               debugPrefix + "incoming connection on " + csocket);

            server = new Server(csocket); // establecer socket de cliente por hebra
            Thread thread = new Thread(server);
           thread.start();
            if (!thread.isAlive())
                server.dispose(); // cerrar el socket de cliente
       } while(true);
    } catch (Exception exc) {
       debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "*** ERROR al procesar peticiones ***");
        exc.printStackTrace();
    } finally {
        try {
            if (ssocket != null)
               ssocket.close(); // cerrar el socket de servidor
            if (server != null)
                server.dispose(); // cerrar el socket de cliente
        } 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
                + "No se han recibido datos; el cliente podría estar desconectado");

            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 +
                    "Servidor primario ha recibido cred delegadas; usarlas");
            runAsInitiator(delCred); // utilizar credenciales delegadas
        }
        else
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                   "Servidor no primario ha recibido cred delegadas; "
                        + "ignorarlas");

        }
    }
    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);

  // Establecer primero el contexto con el iniciador.
  if (!establishContext(context))
      return;

  // A continuación procesar los mensajes del iniciador.
  // Se espera recibir un mensaje envuelto seguido de un MIC.
  // El MIC debe haberse calculado a partir del texto sin formato
  // que se ha recibido envuelto.
  // Utilizar las credenciales delegadas si existen.
  // A continuación, ejecutar como iniciador utilizando las credenciales propias
  // si es necesario; solo hace esto la primera hebra.

  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
             + "No se han recibido datos; el cliente podría estar desconectado");

         return;
      }

      // Esperar primero mensaje envuelto.
      if (!unwrapped)
      {
          response = unwrap(context, request);
          unwrapped = true;
          continue; // obtener siguiente petición
      }

      // Seguido de un MIC.
      verifyMIC(context, request, response);

      // Suplantar al iniciador si ha delegado sus credenciales.
      if (context.getCredDelegState())
          useDelegatedCred(context);

      debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
          + "clientServer=" + clientServer
          + ", beenInitiator=" + beenInitiator);

      // Si es necesario, ejecutar como iniciador utilizando las credenciales propias.
      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();

        // Silenciar las excepciones para cada hebra para no
        // desactivar el servidor debido a excepciones en
        // hebras individuales.
        return;
    } finally {
        if (context != null)
        {
            try {
                context.dispose();
            } catch (Exception exc) {}
        }
    }
}

synchronized void runAsInitiatorOnce(Thread thread)
    throws InterruptedException
{
    if (!beenInitiator)
    {
        // establecer pronto distintivo en true para evitar que hebras
        // posteriores intenten ejecutar runAsInitiator.
        beenInitiator = true;

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                "A punto de ejecutar como iniciador con cred propias ...");

        //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
             + "Excepción al ejecutar como iniciador");

        exc.printStackTrace();
    } finally {
        try {
            client.dispose();
        } catch (Exception exc) {}
    }
}
```

```
    void runAsInitiator()
    {
        if (clientServer)
        {
           debug.out(Debug.OPTS_CAT_APPLICATION,
               debugPrefix + "ejecutar como iniciador con cred propias");

            runAsInitiator(gssCred); // utilizar credenciales propias;
        }
        else
        {
           debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
               + "No se puede ejecutar como iniciador con cred propias "
               + "\nporque no se ejecuta como iniciador y aceptante.");
        }
    }

    void printUsage()
    {
        System.out.println(program + usageString);
    }

    public static void main(String[] args) throws Exception
    {
        System.out.println(debug.toString()); // XXXXXXX
        String programName = "Server";
        try {
            Server server = new Server(programName,
                                       false); // no utilizar credenciales del sujeto
            server.processArgs(args);
            server.initialize();
            server.processRequests();
        } catch (Exception exc) {
            debug.out(Debug.OPTS_CAT_APPLICATION, programName + ": EXCEPTION");
            exc.printStackTrace();
            throw exc;
        }
    }
}
```

Ejemplo: programa cliente IBM JGSS habilitado para JAAS

Este programa de ejemplo realiza un inicio de sesión JAAS y funciona dentro del contexto de inicio de sesión JAAS. No establece la variable javax.security.auth.useSubjectCredsOnly, dejándola como valor predeterminado, que es "true", para que el servicio GSS Java adquiera credenciales del sujeto JAAS asociado al contexto de inicio de sesión creado por el cliente.

Para obtener más información sobre cómo utilizar el programa cliente de ejemplo, vea: "Ejemplos: descargar y ejecutar los programas JGSS de ejemplo" en la página 410.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Programa cliente IBM Java GSS 1.0 habilitado para JAAS de ejemplo

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 cliente de ejemplo Java GSS que utiliza JAAS.
 *
 * Inicia una sesión en JAAS y opera en el contexto de inicio de sesión de JAAS creado con ese fin.
```

```
 *
 * No establece la variable JAVA
 * javax.security.auth.useSubjectCredsOnly, dejando
 * que la variable utilice el valor predeterminado true
 * para que Java GSS adquiera credenciales del sujeto JAAS
 * asociado al contexto de inicio de sesión (creado por el cliente).
 *
 * JAASClient equivale a su superclase {@link Client Client}
 * en todos los demás aspectos y puede
 * ejecutarse para los servidores y clientes de ejemplo no JAAS.
 */

class JAASClient extends Client
{
    JAASClient(String programName) throws Exception
    {
        // No establecer useSubjectCredsOnly. Establecer solo el nombre de programa.
        // useSubjectCredsOnly toma el valor predeterminado "true" si no se establece.
        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);//utilizar credenciales del sujeto
            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 ...");
    }
}
```

Ejemplo: programa servidor IBM JGSS habilitado para JAAS

Este programa de ejemplo realiza un inicio de sesión JAAS y funciona dentro del contexto de inicio de sesión JAAS.

Para obtener más información sobre cómo utilizar el programa servidor de ejemplo, vea: "Ejemplos: descargar y ejecutar los programas JGSS de ejemplo" en la página 410.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
// Programa servidor IBM Java GSS 1.0 habilitado para JAAS de ejemplo

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 servidor de ejemplo Java GSS que utiliza JAAS.
 *
 * Inicia una sesión en JAAS y opera en el contexto de inicio de sesión de JAAS creado con ese fin.
 *
 * No establece la variable JAVA
 * javax.security.auth.useSubjectCredsOnly, dejando
 * que la variable utilice el valor predeterminado true
 * para que Java GSS adquiera credenciales del sujeto JAAS
 * asociado al contexto de inicio de sesión (creado por el servidor).
 *
 * JAASServer equivale a su superclase {@link Server Server}
 * en todos los demás aspectos y puede
 * ejecutarse para los servidores y clientes de ejemplo no JAAS.
 */

class JAASServer extends Server
{
    JAASServer(String programName) throws Exception
    {
        super(programName);
    }

    static class JAASServerAction implements PrivilegedExceptionAction
    {
        private JAASServer server = null;

        JAASServerAction(JAASServer server)
        {
            this.server = server;
        }
```

```
        public Object run() throws Exception
        {
            server.initialize();
            server.processRequests();

            return null;
        }
    }

    public static void main(String[] args) throws Exception
    {
        String programName     = "JAASServer";
        Debug dbg              = new Debug();

        System.out.println(dbg.toString()); // XXXXXXX

        try {
            // No establecer useSubjectCredsOnly.
            // useSubjectCredsOnly toma el valor predeterminado "true" si no se establece.

            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;
        }
    }
}
```

Ejemplos: IBM Java Secure Sockets Extension 1.4

Los ejemplos de JSSE muestran cómo pueden un cliente y un servidor emplear el proveedor de JSSE nativo de System i5 para crear un contexto que habilite las comunicaciones seguras.

Nota: Ambos ejemplos utilizan el proveedor de JSSE nativo de System i5, sean cuales sean las propiedades especificadas por el archivo java.security.

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

Ejemplo: llamar a un programa CL con java.lang.Runtime.exec()

En este ejemplo se muestra cómo ejecutar programas CL desde dentro de un programa Java. En este ejemplo, la clase Java CallCLPgm ejecuta un programa CL.

El programa CL utiliza el mandato Visualizar programa Java (DSPJVAPGM) para visualizar el programa asociado al archivo de clase Hello. En este ejemplo, se supone que el programa CL se ha compilado y está en una biblioteca que se llama JAVSAMPLIB. La salida del programa CL está en el archivo en spool QSYSPRT.

Para obtener un ejemplo de cómo llamar a un programa CL desde dentro de un programa Java, vea: "Ejemplo: llamar a un mandato CL con java.lang.Runtime.exec()" en la página 233.

Nota: La biblioteca JAVSAMPLIB no se crea como parte del proceso de instalación del programa bajo licencia (LP) IBM Developer número 5761-JV1. Tendrá que crear la biblioteca de manera explícita.

Ejemplo 1: clase CallCLPgm

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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 en el método exec()");
         e.printStackTrace();
      }
   } // Finalizar el método main()
} // Finalizar la clase
```

Ejemplo 2: visualizar un programa CL Java

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
               OUTPUT(*PRINT)
ENDPGM
```

Ejemplo: llamar a un mandato CL con java.lang.Runtime.exec()

Este ejemplo muestra cómo ejecutar un mandato de lenguaje de control (CL) desde un programa Java.

En este ejemplo, la clase Java ejecuta un mandato CL. El mandato CL utiliza el mandato CL Visualizar programa Java (DSPJVAPGM) para visualizar el programa asociado al archivo de clase Hello. La salida del mandato CL está en el archivo en spool QSYSPRT.

Al establecer la propiedad del sistema os400.runtime.exec como EXEC (que es el valor predeterminado), los mandatos que pase a la función `Runtime.getRuntime().exec()` utilizarán el siguiente formato:

```
Runtime.getRuntime()Exec("system CLCOMMAND");
```

donde *CLCOMMAND* es el mandato CL que desea ejecutar.

Nota: Al establecer os400.runtime.exec como QSHELL, debe añadir una barra inclinada y comillas (\"). Por ejemplo, el mandato anterior tendrá este aspecto:

```
Runtime.getRuntime()Exec("system \"CLCOMMAND\"");
```

Ejemplo: Clase para llamar a un mandato CL

El código siguiente presupone que utilizará el valor predeterminado de EXEC para la propiedad del sistema os400.runtime.exec.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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 en el método exec()");
         e.printStackTrace();
      }
   } // Finalizar el método main()
} // Finalizar la clase
```

Conceptos relacionados

"Utilizar java.lang.Runtime.exec()" en la página 230
Utilice el método java.lang.Runtime.exec para llamar a programas o mandatos desde dentro de un programa Java. Al utilizar el método java.lang.Runtime.exec() se crean uno o varios trabajos adicionales habilitados para hebras. Los trabajos adicionales procesan la serie de mandatos que se pasa en el método.

"Lista de propiedades Java del sistema" en la página 16
Las propiedades Java del sistema determinan el entorno en el que se ejecutan los programas Java. Son parecidas a los valores del sistema o a las variables de entorno de i5/OS.

Ejemplo: llamar a otros programa Java con java.lang.Runtime.exec()

En este ejemplo se enseña a llamar a otro programa Java con java.lang.Runtime.exec(). Esta clase llama al programa Hello que viene como parte de IBM Developer Kit para Java. Cuando la clase Hello escribe en System.out, este programa obtiene un handle para acceder a la corriente y puede leer en ella.

Nota: Para llamar al programa, utilice el intérprete Qshell.

Ejemplo 1: clase CallHelloPgm

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.io.*;

public class CallHelloPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallHelloPgm.main() invocado");

      // llamar a la clase Hello.
```

```
      try
      {
          theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
      }
      catch(IOException e)
      {
         System.err.println("Error en el método exec()");
         e.printStackTrace();
      }

      // leer en la corriente de salida estándar del programa llamado.
      try
      {
         inStream = new BufferedReader(
                               new InputStreamReader( theProcess.getInputStream() ));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error en inStream.readLine()");
         e.printStackTrace();
      }

   } // Finalizar el método.

} // Finalizar la clase
```

Ejemplo: llamar a Java desde C

Este es un ejemplo de programa C que utiliza la función system() para llamar al programa Java Hello.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
#include <stdlib.h>

 int main(void)
 {
   int result;

  /* La función system pasa la serie dada al procesador de mandatos CL
    para su proceso. */

   result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
 }
```

Ejemplo: llamar a Java desde RPG

Este es un ejemplo de un programa RPG que utiliza la API QCMDEXC para llamar al programa Java Hello.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
     D*             DEFINIR LOS PARÁMETROS DE LA API QCMDEXC
     D*
     DCMDSTRING        S             25    INZ('JAVA CLASS(''com.ibm.as400.system.Hello'')')
     DCMDLENGTH        S             15P 5 INZ(25)
     D*             AHORA SE LLAMA A QCMDEXC CON EL MANDATO CL 'JAVA'
     C                   CALL      'QCMDEXC'
     C                   PARM                    CMDSTRING
     C                   PARM                    CMDLENGTH
     C*       La siguiente línea visualiza 'DID IT' después de salir de la
     C*       shell Java por medio de F3 o F12.
```

```
     C     'DID IT'      DSPLY
     C*       Activar LR para salir del programa RPG
     C                   SETON                                        LR
     C
```

Ejemplo: utilizar corrientes de entrada y de salida para la comunicación entre procesos

Este ejemplo muestra cómo llamar a un programa C desde Java y utilizar corrientes de entrada y salida para la comunicación entre procesos.

En este ejemplo, el programa C escribe una serie en su corriente de salida estándar y el programa Java la lee y la visualiza. En este ejemplo, se supone que se ha creado una biblioteca llamada JAVSAMPLIB y que en ella se ha creado el programa CSAMP1.

Nota: La biblioteca JAVSAMPLIB no se crea como parte del proceso de instalación del programa bajo licencia (LP) IBM Developer número 5761-JV1. Debe crearse de manera explícita.

Ejemplo 1: clase CallPgm

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
import java.io.*;

public class CallPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallPgm.main() invocado");

      // llamar al programa CSAMP1
      try
      {
          theProcess = Runtime.getRuntime().exec(
                       "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
      }
      catch(IOException e)
      {
         System.err.println("Error en el método exec()");
         e.printStackTrace();
      }

      // leer en la corriente de salida estándar del programa llamado.
      try
      {
         inStream = new BufferedReader(new InputStreamReader
                           (theProcess.getInputStream()));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error en inStream.readLine()");
         e.printStackTrace();
      }

   } // Finalizar el método.

 } // Finalizar la clase
```

Ejemplo 2: programa C CSAMP1

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
     /* Convertir la serie a ASCII en tiempo de compilación */
#pragma convert(819)
     printf("Se ha invocado el programa JAVSAMPLIB/CSAMP1\n");
#pragma convert(0)
     /* es posible que Stdout esté en memoria intermedia, */
     /* así que hay que vaciar la memoria intermedia */

     fflush(stdout);

}
```

Ejemplo: API de invocación Java

Este ejemplo sigue el paradigma de la API de invocación estándar.

Hace lo siguiente:

- Se crea una máquina virtual Java mediante JNI_CreateJavaVM.
- Se utiliza la máquina virtual Java para buscar el archivo de clase que se desea ejecutar.
- Se busca el ID del método main de la clase.
- Se llama al método main de la clase.
- Se notifican los errores si se produce una excepción.

Al crear el programa, el programa de servicio QJVAJNI o QJVAJNI64 proporciona la función de API de invocación JNI_CreateJavaVM. JNI_CreateJavaVM crea la máquina virtual Java.

Nota: QJVAJNI64 es un nuevo programa de servicio para teraespacio/método nativo LLP64 y soporte de API de invocación.

Estos programas de servicio residen en el directorio de enlace del sistema y no es necesario identificarlos explícitamente en un mandato crear de lenguaje de control (CL). Por ejemplo, no identificaría explícitamente los programas de servicio mencionados anteriormente al utilizar el mandato Crear programa (CRTPGM) o el mandato Crear programa de servicio (CRTSRVPGM).

Una manera de ejecutar este programa es utilizar el siguiente mandato de lenguaje de control:

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```

Todo trabajo que cree una máquina virtual Java debe tener capacidad multihebra. La salida del programa principal, así como cualquier salida del programa, va a parar a los archivos en spool QPRINT. Estos archivos en spool resultan visibles si se utiliza el mandato de lenguaje de control (CL) Trabajar con trabajos sometidos (WRKSBMJOB) y se visualiza el trabajo iniciado utilizando el mandato CL Someter trabajo (SBMJOB).

Ejemplo: Utilizar la API de invocación Java

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
#define OS400_JVM_15
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
```

```
#include <jni.h>

/* Especificar el pragma que provoca que todas las series literales
 * del código fuente se almacenen en ASCII (que, para las series
 * utilizadas, es equivalente a UTF-8)
 */

#pragma convert(819)

/* Procedimiento: Oops
 *
 * Descripción: Rutina de ayuda a la que se llama cuando una función JNI
 *              devuelve un valor cero, indicando un error grave.
 *              Esta rutina informa de la excepción a la salida de error estándar y
 *              finaliza la JVM abruptamente con una llamada a FatalError.
 *
 * Parámetros:  env -- JNIEnv* que debe utilizarse para llamadas JNI
 *              msg -- char* que señala hacia la descripción del error en UTF-8
 *
 * Nota:        El control no se devuelve después de la llamada a FatalError
 *              y no se devuelve desde este procedimiento.
 */

void Oops(JNIEnv* env, char *msg) {
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*env)->FatalError(env, msg);
}

/* Esta es la rutina "main" del programa. */
int main (int argc, char *argv[])
{

    JavaVMInitArgs initArgs; /* Estructura de inicialización de máquina virtual, pasada por
                              * referencia a JNI_CreateJavaVM(). Los detalles están en jni.h
                              */
    JavaVM* myJVM;           /* Puntero de JavaVM establecido por llamada a
                              * JNI_CreateJavaVM */
    JNIEnv* myEnv;           /* Puntero de JNIEnv establecido por llamada a
                              * JNI_CreateJavaVM */
     char*   myClasspath;     /* 'Serie' de vía de acceso de clases modificable */
    jclass  myClass;         /* Clase a la que hay que llamar, 'NativeHello'. */
    jmethodID mainID;        /* ID de método de esta rutina 'main' routine. */
    jclass  stringClass;     /* Necesaria para crear la String[] arg para main */
    jobjectArray args;       /* La propia String[] */
    JavaVMOption options[1]; /* Matriz de opciones -- utilizar opciones para
                              * establecer vía de acceso de clases */
    int     fd0, fd1, fd2;   /* Descriptores de archivo para IO */

    /* Abrir descriptores de archivo para que IO funcione. */
    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);

    /*  Establecer campo versión de argumentos de inicialización para J2SE v1.5. */
    initArgs.version = 0x00010005;
 /*  Para utilizar J2SDK v1.4, establezca initArgs.version = 0x00010004; */


    /* Ahora, interesa especificar el directorio para que la clase
     * se ejecute en la vía de acceso de clases.
     * Con Java2, la vía de acceso de clases se pasa como opción.
     * Nota: debe especificar el nombre de directorio en formato UTF-8. Envuelva
     " los bloques de código con sentencias #pragma convert.
     */
    options[0].optionString="-Djava.class.path=/CrtJvmExample";
```

```
|   /*para utilizar J2SDK v1.4, sustituya '1.5' por '1.4'.
|   options[1].optionString="-Djava.version=1.5" */
|
|      initArgs.options=options;  /* Pasar la vía de acceso de clases configurada. */
|      initArgs.nOptions = 1;     /* Pasar vía de acceso de clases y opciones de versión */
|
|      /*  Crear la JVM -- un código de retorno no cero indica que se produjo
|       *  un error. Vuelva a EBCDIC y escriba un mensaje en stderr
|       *  antes de salir del programa.
|       */
|      if (JNI_CreateJavaVM(myJVM, (void **)myEnv, (void *)initArgs)) {
|     #pragma convert(0)
|           fprintf(stderr, "No se ha podido crear la JVM\n");
|     #pragma convert(819)
|            exit(1);
|      }
|
|      /*  Utilizar la JVM recién creada para localizar la clase de ejemplo,
|       *  denominada 'NativeHello'.
|       */
|      myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
|      if (! myClass) {
|          Oops(myEnv, "No se ha encontrado la clase 'NativeHello'");
|      }
|
|      /* Ahora, obtener el identificador de método del punto de entrada 'main'
|       * de la clase.
|       * Nota: la firma de 'main' siempre es la misma para cualquier
|       *       clase llamada mediante el siguiente mandato java:
|       *           "main" , "([Ljava/lang/String;)V"
|       */
|      mainID = (*myEnv)->GetStaticMethodID(myEnv,myClass,"main",
|                                          "([Ljava/lang/String;)V");
|      if (! mainID) {
|          Oops(myEnv, "No se ha encontrado jmethodID de 'main'");
|      }
|
|      /* Obtener la jclass para String para crear la matriz
|       * de String que debe pasarse a 'main'.
|       */
|      stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
|      if (! stringClass) {
|          Oops(myEnv, "No se ha encontrado java/lang/String");
|      }
|
|      /*  Ahora, es necesario crear una matriz de series vacía,
|       *  dado que main requiere una matriz de este tipo como parámetro.
|       */
|      args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
|      if (! args) {
|          Oops(myEnv, "No se ha podido crear la matriz de args");
|      }
|
|      /* Ahora, ya tiene el methodID de main y la clase, así que puede
|       * llamar al método main.
|       */
|      (*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);
|
|      /* Comprobar si hay errores. */
|      if ((*myEnv)->ExceptionOccurred(myEnv)) {
|          (*myEnv)->ExceptionDescribe(myEnv);
|      }
|
|      /* Finalmente, destruir la JavaVM que creó. */
|      (*myJVM)->DestroyJavaVM(myJVM);
```

```
|
|     /* Eso es todo. */
|     return 0;
| }
```

| Para obtener más información, consulte “API de invocación Java” en la página 213.

Ejemplo: método nativo IBM i5/OS PASE para Java

El ejemplo de método nativo IBM i5/OS PASE para Java llama a una instancia de un método nativo C que luego utiliza la interfaz Java nativa (JNI) para llamar de nuevo al código Java. En lugar de acceder a la serie directamente desde el código Java, el ejemplo llama a un método nativo que luego llama de nuevo a Java mediante JNI para obtener el valor de la serie.

Para ver las versiones HTML de los archivos fuente de ejemplo, utilice los enlaces siguientes:

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

- “Ejemplo: PaseExample1.java”
- “Ejemplo: PaseExample1.c” en la página 558

Para poder ejecutar el ejemplo de método nativo i5/OS PASE, primero debe llevar a cabo las tareas de los siguientes temas:

1. “Ejemplo: descargar el código fuente de ejemplo a la estación de trabajo AIX” en la página 559
2. “Ejemplo: preparar el código fuente de ejemplo” en la página 559
3. “Ejemplo: preparar el System i5 para que ejecute el ejemplo de método nativo PASE para Java” en la página 560

Ejecutar el ejemplo de método nativo i5/OS PASE para Java

Tras completar las tareas anteriores, puede ejecutar el ejemplo. Utilice cualquiera de los mandatos siguientes para ejecutar el programa de ejemplo:

- Desde un indicador de mandatos de i5/OS:

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- Desde un indicador de mandatos de Qshell o desde una sesión de terminal i5/OS PASE:

```
cd /home/example
java PaseExample1
```

Ejemplo: PaseExample1.java

Este programa de ejemplo carga la biblioteca de métodos nativos PaseExample1. El código fuente del método nativo está en PaseExample1.c. El método printString de este programa Java utiliza un método nativo, getStringNative, para recuperar el valor de la serie (String). El método nativo tan solo llama de nuevo al método getStringCallback de esta clase.

Nota: lea el apartado Declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Este programa de ejemplo carga la biblioteca de métodos nativos PaseExample1.
// El código fuente del método nativo se incluye en PaseExample1.c
// El método printString de este programa Java utiliza un método nativo,
// getStringNative, para recuperar el valor de la serie (String). El método nativo
// simplemente vuelve a llamar al método getStringCallback de esta clase.
//
////////////////////////////////////////////////////////////////////////////
```

```
public class PaseExample1 {
    public static void main(String args[]) {
 PaseExample1 pe1 = new PaseExample1("String for PaseExample1");
 pe1.printString();
    }

    String str;

    PaseExample1(String s) {
 str = s;
    }


    //--------------------------------------------------------------
    public void printString() {
 String result = getStringNative();
 System.out.println("El valor de str es '" + result + "'");
    }

    // Esto llama a getStringCallback mediante JNI.
    public native String getStringNative();

    // Llamado por getStringNative por medio de JNI.
    public String getStringCallback() {
 return str;
    }


    //--------------------------------------------------------------
    static {
 System.loadLibrary("PaseExample1");
    }

}
```

Ejemplo: PaseExample1.c

Este método nativo implementa el método getStringNative de la clase PaseExample1. Utiliza la función JNI CallObjectMethod para llamar de nuevo al método getStringCallback de la clase PaseExample1.

Nota: lea el apartado Declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
/*
 *

 * Este método nativo implementa el método getStringNative de la clase
 * PaseExample1. Utiliza la función JNI CallObjectMethod para llamar
 * de nuevo al método getStringCallback de la clase PaseExample1.
 *
  * Compilar este código en AIX para crear el módulo libPaseExample1.so.
 *
 */

#include "PaseExample1.h"
#include <stdlib.h>

/*
 * Clase:     PaseExample1
 * Método:    getStringNative
 * Signatura: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_PaseExample1_getStringNative(JNIEnv* env, jobject obj) {
   char* methodName = "getStringCallback";
   char* methodSig = "()Ljava/lang/String;";
```

```
    jclass clazz = (*env)->GetObjectClass(env, obj);
    jmethodID methodID = (*env)->GetMethodID(env, clazz, methodName, methodSig);
    return (*env)->CallObjectMethod(env, obj, methodID);
}
```

Ejemplo: descargar el código fuente de ejemplo a la estación de trabajo AIX

Para poder ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java, primero debe descargar un archivo comprimido que contiene el código fuente. Para descargar el archivo comprimido a la estación de trabajo AIX, siga estos pasos.

1. Cree un directorio temporal en la estación de trabajo AIX en la que desea colocar los archivos fuente.
2. Descargue el código fuente de ejemplo de i5/OSPASE al directorio temporal.
3. Descomprima los archivos de ejemplo en el directorio temporal.

Para obtener más información sobre el ejemplo de método nativo IBM i5/OS PASE para Java, vea los siguientes temas:

"Ejemplo: método nativo IBM i5/OS PASE para Java" en la página 227
El ejemplo de método nativo IBM i5/OS PASE para Java llama a una instancia de un método nativo C que luego utiliza la interfaz Java nativa (JNI) para llamar de nuevo al código Java. En lugar de acceder a la serie directamente desde el código Java, el ejemplo llama a un método nativo que luego llama de nuevo a Java mediante JNI para obtener el valor de la serie.

"Ejemplo: preparar el código fuente de ejemplo"
Antes de mover el ejemplo de método nativo IBM i5/OS PASE para Java al servidor, debe compilar el código fuente, crear un archivo de inclusión C y crear un objeto de tipo biblioteca compartida.

"Ejemplo: preparar el System i5 para que ejecute el ejemplo de método nativo PASE para Java" en la página 560
Antes de ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java, debe preparar el servidor para que ejecute el ejemplo. Para preparar el servidor es necesario copiar los archivos en el servidor y añadir las variables de entorno necesarias para ejecutar el ejemplo.

Ejemplo: preparar el código fuente de ejemplo

Antes de mover el ejemplo de método nativo IBM i5/OS PASE para Java al servidor, debe compilar el código fuente, crear un archivo de inclusión C y crear un objeto de tipo biblioteca compartida.

En este ejemplo se incluyen los siguientes archivos fuente C y Java:

- **PaseExample1.c:** archivo de código fuente C que contiene una implementación de getStringNative().
- **PaseExample1.java:** archivo de código fuente Java que llama al método nativo getStringNative en el programa C.

Se utiliza el archivo Java .class compilado para crear un archivo de inclusión C, PaseExample1.h, que contiene un prototipo de función para el método getStringNative incluido en el código fuente C.

Para preparar el código fuente de ejemplo en la estación de trabajo AIX, siga estos pasos:

1. Utilice el siguiente mandato para compilar el código fuente Java:

   ```
   javac PaseExample1.java
   ```

2. Utilice el mandato siguiente para crear un archivo de inclusión C que contenga los prototipos de métodos nativos:

   ```
   javah -jni PaseExample1
   ```

   El nuevo archivo de inclusión C (PaseExample1.h) contiene un prototipo de función para el método getStringNative. El código fuente C de ejemplo (PaseExample1.c) ya incluye la información que copiaría y modificaría del archivo de inclusión C para emplear el método getStringNative. Para obtener más información sobre cómo utilizar JNI, vea la Interfaz Java nativa en el sitio Web de Sun.

3. Utilice el mandato siguiente para compilar el código fuente C y crear un objeto de biblioteca compartida.

```
xlc -G -I/usr/local/java/J1.5.0/include PaseExample1.c -o libPaseExample1.so
```

El nuevo archivo de objeto de biblioteca compartida (libPaseExample1.so) contiene la biblioteca de métodos nativos ″PaseExample1″ que utiliza el ejemplo.

Nota: es posible que deba cambiar la opción -I para que señale hacia el directorio que contiene los archivos de inclusión de métodos Java nativos correctos (por ejemplo, jni.h) del sistema AIX.

Para obtener más información sobre el ejemplo de método nativo IBM i5/OS PASE para Java, vea los siguientes temas:

"Ejemplo: método nativo IBM i5/OS PASE para Java" en la página 227
El ejemplo de método nativo IBM i5/OS PASE para Java llama a una instancia de un método nativo C que luego utiliza la interfaz Java nativa (JNI) para llamar de nuevo al código Java. En lugar de acceder a la serie directamente desde el código Java, el ejemplo llama a un método nativo que luego llama de nuevo a Java mediante JNI para obtener el valor de la serie.

"Ejemplo: descargar el código fuente de ejemplo a la estación de trabajo AIX" en la página 559
Para poder ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java, primero debe descargar un archivo comprimido que contiene el código fuente. Para descargar el archivo comprimido a la estación de trabajo AIX, siga estos pasos.

"Ejemplo: preparar el System i5 para que ejecute el ejemplo de método nativo PASE para Java"
Antes de ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java, debe preparar el servidor para que ejecute el ejemplo. Para preparar el servidor es necesario copiar los archivos en el servidor y añadir las variables de entorno necesarias para ejecutar el ejemplo.

Ejemplo: preparar el System i5 para que ejecute el ejemplo de método nativo PASE para Java

Antes de ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java, debe preparar el servidor para que ejecute el ejemplo. Para preparar el servidor es necesario copiar los archivos en el servidor y añadir las variables de entorno necesarias para ejecutar el ejemplo.

Para preparar el servidor, siga estos pasos:

1. Cree el siguiente directorio del sistema de archivos integrado en el servidor que desea que contenga los archivos de ejemplo. Por ejemplo, utilice el siguiente mandato de lenguaje de control (CL) para crear el directorio denominado /home/example:
   ```
   mkdir /home/example
   ```
2. Copie los siguientes archivos en el nuevo directorio:
   - PaseExample1.class
   - libPaseExample1.so
3. Desde un indicador de mandatos de i5/OS, utilice los siguientes mandatos de lenguaje de control (CL) para añadir las variables de entorno necesarias:
   ```
   addenvvar PASE_THREAD_ATTACH 'Y'
   addenvvar PASE_LIBPATH '/home/example'
   addenvvar QIBM_JAVA_PASE_STARTUP '/usr/lib/start32'
   ```

Nota: Cuando se utilizan métodos nativos PASE desde una sesión de terminal i5/OS PASE, ya se ha iniciado un entorno PASE de 32 bits. En este caso, establezca únicamente PASE_THREAD_ATTACH en Y y PASE_LIBPATH en la vía de acceso de las bibliotecas de métodos nativos PASE. En esta situación, al definir QIBM_JAVA_PASE_STARTUP, la JVM no se inicia correctamente.

Para obtener más información sobre las variables de entorno añadidas, vea: "Ejemplos: variables de entorno para el ejemplo de IBM i5/OS PASE" en la página 222.

Para obtener más información sobre el ejemplo de método nativo IBM i5/OS PASE para Java, vea los siguientes temas:

"Ejemplo: método nativo IBM i5/OS PASE para Java" en la página 227
El ejemplo de método nativo IBM i5/OS PASE para Java llama a una instancia de un método nativo C que luego utiliza la interfaz Java nativa (JNI) para llamar de nuevo al código Java. En lugar de acceder a la serie directamente desde el código Java, el ejemplo llama a un método nativo que luego llama de nuevo a Java mediante JNI para obtener el valor de la serie.

"Ejemplo: descargar el código fuente de ejemplo a la estación de trabajo AIX" en la página 559
Para poder ejecutar el ejemplo de método nativo IBM i5/OS PASE para Java, primero debe descargar un archivo comprimido que contiene el código fuente. Para descargar el archivo comprimido a la estación de trabajo AIX, siga estos pasos.

"Ejemplo: preparar el código fuente de ejemplo" en la página 559
Antes de mover el ejemplo de método nativo IBM i5/OS PASE para Java al servidor, debe compilar el código fuente, crear un archivo de inclusión C y crear un objeto de tipo biblioteca compartida.

Ejemplos: utilizar la interfaz Java nativa (JNI) para métodos nativos

Este programa es un ejemplo sencillo de la interfaz Java nativa (JNI) en el que se utiliza un método nativo C para visualizar "Hello, World." Para generar el archivo NativeHello.h, se utiliza la herramienta `javah` con el archivo de clase NativeHello. En este ejemplo, se supone que la implementación C de NativeHello forma parte de un programa de servicio llamado NATHELLO.

Nota: Para poder ejecutar este ejemplo, la biblioteca en la que se encuentra el programa de servicio NATHELLO debe estar en la lista de bibliotecas.

Ejemplo 1: clase NativeHello

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
public class NativeHello {

    //  Declarar un campo de tipo 'String' (serie) en el objeto NativeHello.
    //  Este es un campo de 'instancia', así que cada objeto NativeHello
    //  contiene uno.
    public String theString;           // variable de instancia

    //  Declarar el propio método nativo. Este método nativo
    //  crea un nuevo objeto de tipo serie y coloca una referencia a dicho objeto
    //  en 'theString'
    public native void setTheString(); // método nativo para establecer la serie

    //  Se llama a este código de 'inicializador estático' antes de que la clase
    //  se utilice por primera vez.
    static {

        //  Intentar cargar la biblioteca de métodos nativos. Si no se
        //  encuentra, escribir un mensaje en 'out' e intentar una vía
        //  codificada de manera fija. Si esto falla, salir.
        try {

            //  System.loadLibrary usa la lista de bibliotecas de JDK 1.1,
            //  y la propiedad java.library.path o la variable de entorno LIBPATH
            //  de JDK1.2
              System.loadLibrary("NATHELLO");
            }

        catch (UnsatisfiedLinkError e1) {

            //  No se ha encontrado el programa de servicio.
            System.out.println
              ("No he encontrado NATHELLO *SRVPGM.");
```

```
            System.out.println ("(Lo intentaré con una vía codificada de manera fija)");

            try {

                //  System.load toma el formato completo de la vía de acceso del
                //  sistema de archivos integrado.
                System.load ("/qsys.lib/jniexample.lib/nathello.srvpgm");
                }

        catch (UnsatisfiedLinkError e2) {

            //  Si llega a este punto, ya ha acabado. Escribir el mensaje
            //  y salir.
            System.out.println
              ("<suspiro> NATHELLO *SRVPGM no está en ningún sitio. Adiós");
            System.exit(1);
            }
        }
    }

    //  Aquí está el código 'main' de esta clase. Es lo que se ejecuta al
    //  entrar 'java NativeHello' en la línea de mandatos.
    public static void main(String argv[]){

        //  Asignar un objeto NativeHello nuevo.
        NativeHello nh = new NativeHello();

        //  Hacer eco de la ubicación.
        System.out.println("(Java) Objeto NativeHello instanciado");
        System.out.println("(Java) campo serie es '" + nh.theString + "'");
        System.out.println("(Java) Llamar a método nativo para establecer la serie");

        //  Aquí está la llamada al método nativo.
        nh.setTheString();

        //  Ahora, imprimir el valor tras la llamada para mayor seguridad.
        System.out.println("(Java) Devuelto por el método nativo");
        System.out.println("(Java) campo serie es '" + nh.theString + "'");
        System.out.println("(Java) Eso es todo...");
    }
}
```

Ejemplo 2: archivo de cabecera NativeHello.h generado

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
/* NO EDITE ESTE ARCHIVO - está generado por máquina */
#include <jni.h>
/* Cabecera de la clase NativeHello */

#ifndef _Included_NativeHello
#define _Included_NativeHello
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Clase:     NativeHello
 * Método:    setTheString
 * Firma:     ()V
 */
JNIEXPORT void JNICALL Java_NativeHello_setTheString
  (JNIEnv *, jobject);
```

```
#ifdef __cplusplus
}
#endif
#endif
```

Este ejemplo de NativeHello.c muestra la implementación del método nativo en C, así como la manera de enlazar Java con los métodos nativos. No obstante, también señala las complicaciones que surgen por el hecho de que el System i5 es internamente una máquina EBCDIC (código de intercambio decimal ampliado codificado en binario). Asimismo, indica las complicaciones derivadas de la falta de verdaderos elementos de internacionalización en JNI.

Estas razones, aunque no son una novedad en JNI, originan algunas diferencias exclusivas de System i5 en el código C que usted escriba. Conviene que recuerde que si escribe en stdout o en stderr o bien si lee en stdin, es probable que los datos estén codificados en el formato EBCDIC.

En código C, resulta fácil convertir la mayoría de las series literales, aquellas que contienen únicamente caracteres de 7 bits, al formato UTF-8 que requiere JNI. Para ello, hay que incluir las series literales entre sentencias pragma de conversión de página de códigos. No obstante, dado que es posible que se escriba información directamente en stdout o en stderr desde el código C, podría permitirse que algunos literales conservasen el formato EBCDIC.

Nota: Las sentencias #pragma convert(0) convierten los datos de tipo carácter a EBCDIC. Las sentencias #pragma convert(819) convierten los datos de tipo carácter a ASCII (American Standard Code for Information Interchange). Estas sentencias realizan la conversión de los datos de tipo carácter del programa C en tiempo de compilación.

Ejemplo 3: Implementación del método nativo NativeHello.c de la clase Java NativeHello

```
#include <stdlib.h>       /* malloc, free, etcétera */
#include <stdio.h>        /* fprintf(), etcétera */
#include <qtqiconv.H>     /* interfaz iconv() */
#include <string.h>       /* memset(), etcétera */
#include "NativeHello.h"  /* generado por 'javah-jni' */

/*  La página de códigos de todas las series literales es la ISO-8859-1 Latin 1
/* (y en el caso de los caracteres de 7 bits, también son automáticamente UTF-8). */
#pragma convert(819)  /* manejar todas las series literales como ASCII */

/*  Notificar y borrar una excepción JNI.  */
static void HandleError(JNIEnv*);

/*  Imprimir una serie UTF-8 en stderr con el ID de juego de caracteres */
/*  codificado (CCSID) del trabajo actual.  */
static void JobPrint(JNIEnv*, char*);

/*  Constantes que indican en qué dirección hay que convertir:  */
#define CONV_UTF2JOB 1
#define CONV_JOB2UTF 2

/*  Convertir una serie del CCSID del trabajo a UTF-8, o viceversa.  */
int StringConvert(int direction, char *sourceStr, char *targetStr);

/*  Implementación de método nativo de 'setTheString()'.  */

JNIEXPORT void JNICALL Java_NativeHello_setTheString
(JNIEnv *env, jobject javaThis)
{
    jclass thisClass;     /* clase del objeto 'this' */
    jstring stringObject; /* serie nueva; se colocará en un campo de 'this' */
    jfieldID fid;         /* ID de campo necesario para actualizar campo de 'this' */
    jthrowable exception; /* excepción; se recupera con ExceptionOccurred */

    /*  Escribir wl estado en la consola.  */
```

```
    JobPrint(env, "( C ) En el método nativo\n");

    /* Construir el nuevo objeto string.  */
    if (! (stringObject = (*env)->NewStringUTF(env, "Hello, native world!")))
    {
        /*  Para casi toda función de JNI, un valor de retorno nulo indica
        que ha habido un error y que se ha colocado una excepción en un punto en
        que se podía recuperar por 'ExceptionOccurred()'. En este caso, el error
        sería normalmente muy grave, pero a efectos de este ejemplo, seguir adelante
        y capturar el error, y continuar.  */
        HandleError(env);
        return;
    }

    /* obtener la clase del objeto 'this', requisito para obtener fieldID  */
    if (! (thisClass = (*env)->GetObjectClass(env,javaThis)))
    {
        /*  Si GetObjectClass devuelve una clase nula, significa que ha
        habido un problema. En lugar de manejar este problema, utilice return y
        sepa que el retorno a Java 'lanza' automáticamente la excepción
        Java almacenada.  */
        return;
    }

    /*  Obtener el fieldID que hay que actualizar.  */
    if (! (fid = (*env)->GetFieldID(env,
                                    thisClass,
                                    "theString",
                                    "Ljava/lang/String;")))
    {
        /*  Si GetFieldID devuelve un fieldID nulo, significa que ha
        habido un problema. Hay que notificarlo desde aquí y borrarlo.
        Dejar la serie tal como está.  */
        HandleError(env);
        return;
    }

    JobPrint(env, "( C ) Establecer el campo\n");

    /*  Hacer la actualización propiamente dicha.
    Nota: SetObjectField es un ejemplo de interfaz que
    no devuelve un valor de retorno que se puede probar. En este caso,
    hay que llamar a ExceptionOccurred() para ver si ha
    habido un problema relacionado con el almacenamiento del valor  */
    (*env)->SetObjectField(env, javaThis, fid, stringObject);

    /*  Mirar a ver si la actualización se ha realizado. Si no, informar del error.  */
    if ((*env)->ExceptionOccurred(env)) {

        /*  Se ha devuelto un objeto excepción no nulo desde ExceptionOccurred;
        así pues, hay un problem y hay que informar del error. */
        HandleError(env);
    }

    JobPrint(env, "( C ) Volver del método nativo\n");
    return;

}

static void HandleError(JNIEnv *env)
{
    /*  Rutina simple para notificar y manejar una excepción.  */
    JobPrint(env, "( C  ) Error producido en llamada JNI: ");
    (*env)->ExceptionDescribe(env); /* escribir datos de excepción en la consola */
    (*env)->ExceptionClear(env);    /* borrar la excepción pendiente */
}
```

```
static void JobPrint(JNIEnv *env, char *str)
{
    char   *jobStr;
    char   buf[512];
    size_t len;

    len = strlen(str);

    /*  Imprimir solo la serie no vacía. */
    if (len) {
        jobStr = (len >= 512) ? malloc(len+1) : &buf;
        if (! StringConvert(CONV_UTF2JOB, str, jobStr))
            (*env)->FatalError
              (env,"ERROR en JobPrint: imposible convertir UTF2JOB");
        fprintf(stderr, jobStr);
        if (len >= 512) free(jobStr);
    }
}

int StringConvert(int direction, char *sourceStr, char *targetStr)
{
    QtqCode_T source, target;     /* parámetros para instanciar iconv           */
    size_t    sStrLen, tStrLen;   /* copias locales de las longitudes de serie  */
    iconv_t   ourConverter;       /* descriptor de conversión propiamente dicho */
    int       iconvRC;            /* código de retorno de la conversión         */
    size_t    originalLen;        /* longitud original de sourceStr             */

    /*  Hacer copias locales de los tamaños de entrada y salida inicializados
    al tamaño de de la serie de entrada. iconv() exige que
    los parámetros de longitud se pasen por dirección (como int*).          */
    originalLen = sStrLen = tStrLen = strlen(sourceStr);

    /*  Inicializar los parámetros de QtqIconvOpen() a cero.  */
    memset(&source,0x00,sizeof(source));
    memset(&target,0x00,sizeof(target));

    /*  En función del parámetro direction, establecer CCSID SOURCE
    o el CCSID TARGET igual a ISO 8859-1 Latin.  */
    if (CONV_UTF2JOB == direction ) {
        source.CCSID = 819;
    }
    else {
        target.CCSID = 819;
    }

    /*  Crear el objeto convertidor iconv_t.  */
    ourConverter = QtqIconvOpen(&target,&source);

    /*  Asegurarse de que se tiene un convertidor válido; si no, devolver 0.  */
    if (-1 == ourConverter.return_value) return 0;

    /*  Realizar la conversión.  */
    iconvRC = iconv(ourConverter,
                    (char**) &sourceStr,
                    &sStrLen,
                    &targetStr,
                    &tStrLen);

    /*  Si la conversión ha fallado, devolver un cero.  */
    if (0 != iconvRC ) return 0;

    /*  Cerrar el descriptor de conversión.  */
    iconv_close(ourConverter);

    /*  targetStr retorna señalando hacia el carácter que está justo
    después del último carácter convertido; así pues, establecer el nulo
    en ese punto ahora.  */
```

```
    *targetStr = '\0';

    /*  Devolver el número de caracteres procesados. */
    return originalLen-tStrLen;

}

#pragma convert(0)
```

"Utilizar la interfaz Java nativa (JNI) para métodos nativos" en la página 211
Los métodos nativos solo se deben usar en los casos en que Java puro no pueda responder a sus necesidades de programación.

Ejemplo: utilizar sockets para la comunicación entre procesos

En este ejemplo se utilizan sockets para la comunicación entre un programa Java y un programa C.

El programa C, que está a la escucha en un socket, debe iniciarse primero. Una vez que el programa Java se haya conectado al socket, el programa C le envía una serie utilizando la conexión por socket. La serie que se envía desde el programa C es una serie ASCII (American Standard Code for Information Interchange) de la página de códigos 819.

El programa Java se debe iniciar con el mandato `java TalkToC xxxxx nnnn` en la línea de mandatos del intérprete Qshell o en otra plataforma Java. También puede teclear `JAVA TALKTOC PARM(xxxxx nnnn)` en la línea de mandatos de i5/OS para iniciar el programa Java. `xxxxx` es el nombre de dominio o la dirección de protocolo Internet (IP) del sistema en el que se ejecuta el programa C. `nnnn` es el número de puerto del socket utilizado por el programa C. Este número de puerto también se tiene que utilizar como primer parámetro de la llamada al programa C.

Ejemplo 1: clase de cliente TalkToC

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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();

   } // Finalizar el método main()

   public void setUp()
   {
      System.out.println("TalkToC.setUp() invocado");

      try
      {
         socket = new Socket(host, port);
         inStream = new BufferedReader(new InputStreamReader(
```

```
                                    socket.getInputStream()));
      }
      catch(UnknownHostException e)
      {
         System.err.println("No se puede encontrar el host llamado: " + host);
         e.printStackTrace();
         System.exit(-1);
      }
      catch(IOException e)
      {
         System.err.println("No se ha podido establecer conexión para " + host);
         e.printStackTrace();
         System.exit(-1);
      }
   } // Finalizar el método setUp().

   public void converse()
   {
      System.out.println("TalkToC.converse() invocado");

      if (socket != null && inStream != null)
      {
         try
         {
            System.out.println(inStream.readLine());
         }
         catch(IOException e)
         {
            System.err.println("Error de conversación con host " + host);
            e.printStackTrace();
         }

      } // Finalizar if.

   } // Finalizar el método converse()

   public void cleanUp()
   {
      try
      {

         if(inStream != null)
         {
            inStream.close();
         }
         if(socket != null)
         {
            socket.close();
         }
      } // Finalizar try.
      catch(IOException e)
      {
         System.err.println("Error de borrado");
         e.printStackTrace();
         System.exit(-1);
      }
   } // Finalizar el método cleanUp().

} // Finalizar la clase TalkToC.
```

SockServ.C se inicia pasando un parámetro correspondiente al número de puerto. Por ejemplo, `CALL SockServ '2001'`.

Ejemplo 2: programa servidor SockServ.C

Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

```
#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 = "Este es un mensaje del servidor de sockets C.";
   #pragma convert (0)

   /*  Asignar el socket.  */
   if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
   {
      printf("anomalía en la asignación de socket\n");
      perror(NULL);
      exit(-1);
   }

   /* Realizar el enlace (bind). */
   if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
   {
     printf("Enlace fallido\n");
     perror(NULL);
     exit(-1);
   }

   /* Invocar el método listen. */
   listen(server, 1);

   /* Esperar la petición del cliente. */
   if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
   {
     printf("aceptar ha fallado\n");
     perror(NULL);
     exit(-1);
   }


   /* Envía un saludo (greeting) al cliente. */
   if((sendrc = send(client, greeting, strlen(greeting),0))<0)
   {
      printf("Envío fallido\n");
      perror(NULL);
```

```
      exit(-1);
   }

   close(client);
   close(server);

}
```

Ejemplo: intercalar sentencias SQL en la aplicación Java

La siguiente aplicación SQLJ de ejemplo, App.sqlj, utiliza SQL estático para recuperar y actualizar datos de la tabla EMPLOYEE de la base de datos de ejemplo DB2.

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
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
{

  /**********************
   **  Controlador de registro **
   **********************/

 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;

     // El URL es jdbc:db2:nombredb
     String url = "jdbc:db2:sample";

     DefaultContext ctx = DefaultContext.getDefaultContext();
     if (ctx == null)
     {
       try
       {
         // Conectar con id/contraseña predeterminados.
         Connection con = DriverManager.getConnection(url);
```

```
      con.setAutoCommit(false);
      ctx = new DefaultContext(con);
    }
    catch (SQLException e)
    {
      System.out.println("Error: no se ha podido obtener un contexto predeterminado");
      System.err.println(e) ;
      System.exit(1);
    }
    DefaultContext.setDefaultContext(ctx);
  }

  // Recuperar datos de la base de datos.
  System.out.println("Recuperar algunos datos de la base de datos.");
  #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

  // Visualizar el conjunto de resultados.
  // cursor1.next() devuelve false cuando no hay más filas.
  System.out.println("Resultados recibidos:");
  while (cursor1.next()) // 3
  {
    str1 = cursor1.empno(); // 4
    str2 = cursor1.firstnme();

    System.out.print (" empno= " + str1);
    System.out.print (" firstname= " + str2);
    System.out.println("");
  }
  cursor1.close(); // 9

  // Recuperar el número de empleado de la base de datos.
  #sql { SELECT count(*) into :count1 FROM employee }; // 5
  if (1 == count1)
    System.out.println ("Hay una fila en la tabla de empleados");
  else
    System.out.println ("Hay " + count1
                        + " filas en la tabla de empleados");

  // Actualizar la base de datos.
  System.out.println("Actualizar la base de datos.");
  #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

  // Recuperar los datos actualizados de la base de datos.
  System.out.println("Recuperar los datos actualizados de la base de datos.");
  str1 = "000010";
  #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

  // Visualizar el conjunto de resultados.
  // cursor2.next() devuelve false cuando no hay más filas.
  System.out.println("Resultados recibidos:");
  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

  // Retrotraer la actualización.
  System.out.println("Retrotraer la actualización.");
  #sql { ROLLBACK work };
  System.out.println("Retrotracción terminada.");
}
catch( Exception e )
```

```
    {
      e.printStackTrace();
    }
  }
}
```

Declarar iteradores. Esta sección declara dos tipos de iteradores:

- App_Cursor1: Declara nombres y tipos de datos de columna, y devuelve los valores de las columnas de acuerdo con el nombre de columna (enlace por nombre con columnas).
- App_Cursor2: Declara tipos de datos de columna, y devuelve los valores de las columnas por posición de columna (enlace posicional con columnas).

Inicializar el iterador. El objeto iterador cursor1 se inicializa utilizando el resultado de una consulta. La consulta almacena el resultado en cursor1.

Adelantar el iterador a la próxima fila. El método cursor1.next() devuelve un Boolean `false` si no existen más filas para recuperar.

[4]Mover los datos. El método de acceso por nombre empno() devuelve el valor de la columna denominada empno en la fila actual. El método de acceso por nombre firstnme() devuelve el valor de la columna denominada firstnme en la fila actual.

[5]Aplicar SELECT a los datos de una variable del lenguaje principal. La sentencia SELECT pasa el número de filas de la tabla a la variable de lenguaje principal count1.

[6] Inicializar el iterador. El objeto iterador cursor2 se inicializa utilizando el resultado de una consulta. La consulta almacena el resultado en cursor2.

[7]Recuperar los datos. La sentencia FETCH devuelve el valor actual de la primera columna declarada en el cursor ByPos desde la tabla de resultados a la variable de lenguaje principal str2.

[8]Comprobar el éxitode una sentencia FETCH.INTO. El método endFetch() devuelve Boolean true si el iterador no está situado en una fila, es decir, si el último intento de captar una fila ha fallado. El método endFetch() devuelve false si el último intento de captar una fila ha sido satisfactorio. DB2 intenta captar una fila cuando se llama al método next(). Una sentencia FETCH...INTO llama implícitamente al método next().

[9]Cerrar los iteradores. El método close() libera los recursos retenidos por los iteradores. Los iteradores se deben cerrar explícitamente para asegurar que se liberan los recursos del sistema de forma oportuna.

Ejemplos: cambiar el código Java para que utilice fábricas de sockets de cliente

Estos ejemplos muestran cómo cambiar una clase de socket simple, denominada simpleSocketClient, de forma que utilice fábricas de sockets para crear todos los sockets. El primer ejemplo muestra la clase simpleSocketClient sin fábricas de sockets. El segundo muestra la clase simpleSocketClient con fábricas de sockets. En el segundo ejemplo, simpleSocketClient cambia de nombre y pasa a llamarse factorySocketClient.

Ejemplo 1: programa Socket Client sin fábricas de sockets

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
/* Programa Simple Socket Client */

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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Conectando a host " + args[0] + " en el puerto " +
                       serverPort);

    // Crear el socket y conectar con el servidor.
    Socket  s = new Socket(args[0], serverPort);
    .
    .
    .

    // El resto del programa sigue a partir de aquí.
```

Ejemplo 2: programa Simple Socket Client con fábricas de sockets

```
/* Programa Simple Socket Factory Client */

// Observe que se importa javax.net.* para tomar la clase 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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Conectando a host " + args[0] + " en el puerto " +
                       serverPort);

    // Cambiar el programa simpleSocketClient original para crear una
    // fábrica de sockets y luego utilizarla para crear sockets.

    SocketFactory socketFactory = SocketFactory.getDefault();

    // Ahora, la fábrica crea el socket. Es el último cambio
    // que se realiza en el programa simpleSocketClient original.

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // El resto del programa sigue a partir de aquí.
```

Ejemplos: cambiar el código Java para que utilice fábricas de sockets de servidor

Estos ejemplos muestran cómo cambiar una clase de socket simple, denominada simpleSocketServer, de forma que utilice fábricas de sockets para crear todos los sockets. El primer ejemplo muestra la clase simpleSocketServer sin fábricas de sockets. El segundo muestra la clase simpleSocketServer con fábricas de sockets. En el segundo ejemplo, simpleSocketServer cambia de nombre y pasa a llamarse factorySocketServer.

Ejemplo 1: programa Socket Server sin fábricas de sockets

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/* Archivo 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    ServerSocket    serverSocket =
      new ServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Este servidor tan solo se hace eco de lo que se le envía...

    byte buffer[] = new byte[4096];

    int bytesRead;

    // Leer hasta que se devuelva "eof".
    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead); // Escribir lo recibido.
      os.flush();    // Vaciar la memoria intermedia de salida.
    }


    s.close();
    serverSocket.close();
  }// Finalizar main().

}       // Finalizar la definición de clase.
```

Ejemplo 2: programa Simple Socket Server con fábricas de sockets

```
/* Archivo factorySocketServer.java */

// Hay que importar javax.net para tomar la clase 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    // Cambiar la clase simpleSocketServer original para que utilice una
    // ServerSocketFactory con el fin de crear sockets de servidor.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ahora, la fábrica ha de crear el socket de servidor. Es el último
    // cambio que se realiza en el programa original.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Este servidor tan solo se hace eco de lo que se le envía...

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }


    s.close();
    serverSocket.close();
  }

}
```

Ejemplos: cambiar el cliente Java para que utilice la capa de sockets segura (SSL)

Estos ejemplos muestran cómo cambiar una clase, denominada factorySocketClient, de forma que utilice SSL (capa de sockets segura). El primer ejemplo muestra la clase factorySocketClient sin SSL. El segundo muestra la misma clase, que cambia de nombre y pasa a llamarse factorySSLSocketClient, con SSL.

Ejemplo 1: clase factorySocketClient simple sin soporte SSL

Nota: Al utilizar los ejemplos de código, acepta los términos de: “Información sobre licencia de código y exención de responsabilidad” en la página 583.

```
/* Programa Simple Socket Factory Client */

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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Conectando a host " + args[0] + " en el puerto " +
                       serverPort);

    SocketFactory socketFactory = SocketFactory.getDefault();

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // El resto del programa sigue a partir de aquí.
```

Ejemplo 2: clase factorySocketClient simple con soporte SSL

```
// Observe que importamos javax.net.ssl.* para tomar el soporte 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 toma por defecto 3000 si no se especifica.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Conectando a host " + args[0] + " en el puerto " +
                       serverPort);

    // Cambiar esto para crear una SSLSocketFactory en lugar de una SocketFactory.
    SocketFactory socketFactory = SSLSocketFactory.getDefault();

    // No es necesario cambiar nada más.
    // ¡Ahí está la gracia de utilizar fábricas!
    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // El resto del programa sigue a partir de aquí.
```

Ejemplos: cambiar el servidor Java para que utilice la capa de sockets segura (SSL)

Estos ejemplos muestran cómo cambiar una clase, denominada factorySocketServer, de forma que utilice la capa de sockets segura (SSL).

El primer ejemplo muestra la clase factorySocketServer sin SSL. El segundo muestra la misma clase, que cambia de nombre y pasa a llamarse factorySSLSocketServer, con SSL.

Ejemplo 1: clase factorySocketServer simple sin soporte SSL

Nota: Al utilizar los ejemplos de código, acepta los términos de: "Información sobre licencia de código y exención de responsabilidad" en la página 583.

```
/* Archivo factorySocketServer.java */
// Hay que importar javax.net para tomar la clase 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    // Cambiar la clase simpleSocketServer original para que utilice una
    // ServerSocketFactory con el fin de crear sockets de servidor.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ahora, la fábrica ha de crear el socket de servidor. Es el último
    // cambio que se realiza en el programa original.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Este servidor tan solo se hace eco de lo que se le envía.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

Ejemplo 2: clase factorySocketServer simple con soporte SSL

```
/* Archivo factorySocketServer.java */

// Hay que importar javax.net para tomar la clase 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("Se toma por defecto el puerto 3000 porque no se ha especificado serverPort.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Estableciendo socket de servidor en el puerto " + serverPort);

    // Cambiar la clase simpleSocketServer original para que utilice una
    // ServerSocketFactory con el fin de crear sockets de servidor.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Ahora, la fábrica ha de crear el socket de servidor. Es el último
    // cambio que se realiza en el programa original.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // Un servidor real manejaría más de un único cliente así...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // Este servidor tan solo se hace eco de lo que se le envía.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

Resolución de problemas relacionados con IBM Developer Kit para Java

En este tema se le enseña a localizar los archivos de anotaciones de trabajo y a recoger datos para el análisis de programas Java. En este tema también se facilita información sobre los arreglos temporales del programa (PTF) y la manera de obtener soporte técnico para IBM Developer Kit para Java.

Si el rendimiento del programa se degrada al ejecutarse durante mucho más tiempo, es posible que haya codificado una fuga de memoria erróneamente. Puede utilizar JavaWatcher, un componente de iDoctor del System i como ayuda para depurar el programa y localizar las posibles fugas de memoria. Hallará más información en: Herramientas de análisis de memoria dinámica (HAT) para Java.

Limitaciones

En esta lista se señalan las limitaciones, las restricciones o los comportamientos propios que se conocen en el IBM Developer Kit para Java.

- Cuando se carga una clase y no se encuentran las superclases de la misma, el error indica que no se ha encontrado la clase original. Por ejemplo, si la clase B amplía la clase A y no se encuentra la segunda al cargar la primera, el error indica que no se ha encontrado la clase B, aunque en realidad sea la clase A la que no se ha encontrado. Si aparece un error que indica que no se ha encontrado una clase, compruebe que la clase y todas sus superclases están en la vía de acceso de clases. Esto también es válido para las interfaces implementadas por la clase que se esté cargando.
- El almacenamiento dinámico de la recogida de basura está limitado a 240 GB.
- No hay un límite explícito para el número de objetos construidos.
- El parámetro backlog de java.net puede tener en el System i5 un comportamiento distinto del que tiene en otras plataformas. Por ejemplo:
  - Si el parámetro backlog es 0 ó 1 en Listen:
    - Listen(0) significa que se permite una conexión pendiente; no inhabilita un socket.
    - Listen(1) significa que se permite una conexión pendiente y equivale a Listen(0).
  - Si el parámetro backlog es mayor que 1 en Listen:
    - Esto permite que haya muchas peticiones pendientes en la cola de Listen. Si llega una nueva petición de conexión y la cola está al límite, se suprimirá una de las peticiones pendientes.
- Sea cual sea la versión de JDK que esté utilizando, solo puede utilizar la máquina virtual Java en entornos con capacidad para multihebra (es decir, seguros en ejecución multihebra). La plataforma del System i5 es segura en ejecución multihebra, pero no así algunos sistemas de archivos. Si desea obtener una lista de los sistemas de archivos que no tienen seguridad multihebra, vea el tema Sistema de archivos integrado.

Localizar anotaciones de trabajo para el análisis de problemas Java

Para analizar las causas de una anomalía de Java, utilice las anotaciones de trabajo del trabajo que ha ejecutado el mandato Java y las del trabajo inmediato por lotes (BCI) en el que se ha ejecutado el programa Java. Ambos pueden contener información de error importante.

Existen dos formas de buscar el archivo de anotaciones del trabajo BCI. Puede localizar el nombre del trabajo BCI que figura en las anotaciones del trabajo que ejecutó el mandato Java. A continuación, utilice el nombre de trabajo para buscar el archivo de anotaciones del trabajo BCI.

También puede buscar el archivo de anotaciones del trabajo BCI realizando los pasos siguientes:

1. Entre el mandato Trabajar con trabajos sometidos (WRKSBMJOB) en la línea de mandatos de i5/OS.
2. Vaya al final de la lista.
3. Busque el último trabajo de la lista; se llama QJVACMDSRV.
4. Entre la opción 8 (Trabajar con archivos en spool) para el trabajo.
5. Se visualizará un archivo llamado QPJOBLOG.
6. Pulse F11 para ver la vista 2 de los archivos en spool.
7. Verifique que la fecha y la hora coinciden con la fecha y la hora en que se ha producido la anomalía.

   Si la fecha y la hora no coinciden con la fecha y la hora en que usted ha finalizado la sesión, siga buscando en la lista de trabajos sometidos. Intente localizar un archivo de anotaciones llamado QJVACMDSRV cuya fecha y hora coincida con el momento en que ha finalizado la sesión.

Si no puede hallar ningún archivo de anotaciones para el trabajo BCI, es posible que no se haya generado ninguno. Esto ocurre si ha establecido un valor ENDSEP demasiado alto para la descripción de trabajo QDFTJOBD o si el valor LOG de la descripción de trabajo QDFTJOBD especifica *NOLIST. Compruebe estos valores y cámbielos para que se genere un archivo de anotaciones para el trabajo BCI.

Para generar un archivo de anotaciones para el trabajo que ha ejecutado el mandato Ejecutar Java (RUNJVA), lleve a cabo los siguientes pasos:

1. Entre SIGNOFF *LIST.
2. A continuación, vuelva a iniciar la sesión.
3. Entre el mandato Trabajar con archivos en spool (WRKSPLF) en la línea de mandatos de i5/OS.
4. Vaya al final de la lista.
5. Busque un archivo llamado QPJOBLOG.
6. Pulse F11.
7. Verifique que la fecha y la hora coincidan con la fecha y la hora en que ha entrado el mandato de fin de sesión.

   Si la fecha y la hora no coinciden con la fecha y la hora en que usted ha finalizado la sesión, siga buscando en la lista de trabajos sometidos. Intente localizar un archivo de anotaciones llamado QJVACMDSRV cuya fecha y hora coincida con el momento en que ha finalizado la sesión.

Recoger datos para el análisis de problemas de Java

Para reunir datos con el fin de elaborar un informe autorizado de análisis de programa (APAR), siga estos pasos.

1. Incluya una descripción completa del problema.
2. Guarde el archivo de clase Java que ha originado el problema al ejecutarse.
3. Puede utilizar el mandato SAV para guardar objetos del sistema de archivos integrado. Es posible que necesite guardar otros archivos de clase que este programa debe ejecutar. Asimismo, tal vez le interese guardar y enviar un directorio completo con el fin de que IBM lo utilice, si fuese necesario, cuando intente reproducir el problema. He aquí un ejemplo de cómo guardar un directorio entero.

   Ejemplo: guardar un directorio

   Nota: Lea la declaración de limitación de responsabilidad sobre el código de ejemplo para obtener información legal importante.

   ```
   SAV DEV('/QSYS.LIB/TAP01.DEVD') OBJ(('/midir'))
   ```

   Si es posible, guarde los archivos fuente de las clases Java implicadas en el problema. Le servirán de ayuda a IBM a la hora de reproducir y analizar el problema.
4. Guarde los programas de servicio que contienen los métodos nativos que se necesitan para ejecutar el programa.
5. Guarde los archivos de datos que se necesitan para ejecutar el programa Java.
6. Añada una explicación completa de cómo reproducir el problema.

   Incluirá lo siguiente:

   - El valor de la variable de entorno CLASSPATH.
   - La descripción del mandato Java ejecutado.
   - Una indicación de cómo debe responderse a la entrada que necesite el programa.
7. Incluya las anotaciones del código interno vertical bajo licencia (VLIC) que se hayan realizado alrededor del momento en que produjo la anomalía.
8. Añada los archivos de anotaciones del trabajo interactivo y del trabajo BCI en el que se ejecutaba la máquina virtual Java.

Aplicar arreglos temporales del programa

A partir de i5/OS V5R4, puede utilizar el mandato CL Visualizar trabajos de máquina virtual Java (DSPJVMJOB) para gestionar los trabajos de la JVM y aplicar arreglos PTF mientras el sistema está activo.

Muchos arreglos temporales de programa (PTF) Java afectan a la máquina virtual Java (JVM) tanto que, si se aplica el código mientras la máquina virtual Java se está ejecutando, el hecho de aplicar el PTF podría

provocar resultados imprevisibles. Antes, la aplicación de algunos PTF Java se aplazaba hasta que se podría realizar una carga del programa inicial (IPL) en el sistema para asegurar que no se estaban ejecutando trabajos de la JVM en el sistema. Muchos usuarios, sin embargo, pensaron que era poco práctico. Así, se añadió una condición previa a la máquina virtual Java para que muchos de esos arreglos previos de programa aplazados pudieran aplicarse de inmediato cuando no hubieran trabajos de JVM activos en el sistema.. El mandato DSPJVMJOB permite ver qué trabajos ejecutan las máquinas virtuales de Java. Con esta información, se pueden finalizar los trabajos activos de las máquinas virtuales de Java de forma apropiada antes de aplicar los arreglos temporales de programa, en vez de tener que esperar una carga de programa inicial para poder aplicar los arreglos temporales de programa.

Para obtener más información acerca del mandato DSPJVMJOB, vea Visualizar trabajos de máquina virtual Java en el tema CL.

Información relacionada

Mantener y gestionar i5/OS y el software relacionado

Utilizar los arreglos de software

Obtener soporte para IBM Developer Kit para Java

Los servicios de soporte de IBM Developer Kit para Java se prestan según los términos y condiciones habituales de los productos de software de System i. Dichos servicios incluyen servicio técnico de programas, atención telefónica y servicios de consultoría.

Utilice la información en línea proporcionada en la página inicial de IBM System i bajo el tema "Soporte", si desea obtener más información. Utilice los servicios de soporte de IBM para 5761-JV1 (IBM Developer Kit para Java). O bien, póngase en contacto con el representante local de IBM.

Es posible que, por indicación de IBM, deba obtener un nivel más reciente de IBM Developer Kit para Java para recibir servicios de programa de forma continuada. Para obtener más información, consulte el apartado Soporte para varios Java Development Kits (JDK).

Los defectos que pueda presentar el programa IBM Developer Kit para Java se pueden resolver bajo los servicios de programa o bajo atención telefónica. Las cuestiones relacionadas con la depuración o la programación de la aplicación deben resolverse a través de los servicios de consultoría.

Las cuestiones relacionadas con las llamadas a la interfaz de programación de aplicaciones (API) de IBM Developer Kit para Java deben remitirse a los servicios de consultoría, a menos que:

1. Se trate claramente de un defecto de la API Java, que pueda demostrarse por medio de su reproducción en un programa de relativa simplicidad.
2. Sea una cuestión que requiere una aclaración de lo explicado en la documentación.
3. Sea una cuestión referente a la ubicación de los ejemplos o la documentación.

Los servicios de consultoría comprenden también el prestar ayuda de programación. Esto incluye los ejemplos de programa que se facilitan en el producto programa bajo licencia (LP) IBM Developer Kit para Java. Puede haber más ejemplos disponibles en Internet, en la página inicial de IBM System i, no basados en soporte.

El programa bajo licencia IBM Developer Kit para Java facilita información sobre cómo resolver problemas. Si usted cree que existe un defecto potencial en la API de IBM Developer Kit para Java, será necesario que suministre un programa simple que demuestre la existencia del error.

Información relacionada con IBM Developer Kit para Java

Aquí encontrará una lista de fuentes de información que guardan relación con el tema IBM Developer Kit para Java.

Sitios Web

- Java.sun.com: The Source for Java Developers (www.java.sun.com)
  Visite Sun Microsystems, Inc., si desea obtener información sobre los diversos usos de Java, incluidas las nuevas tecnologías.
- IBM developerWorks Java technology zone
  Ofrece información, formación y herramientas para ayduarle a utilizar Java, productos IBM y otras tecnologías destinadas a crear soluciones de negocio.
- IBM alphaWorks Java
  Incluye información sobre las nuevas tecnologías Java, y se incluyen desacargas y enlaces que llevan a recursos de desarrollo.

Javadocs

La siguiente información de consulta Javadoc está relacionada con IBM Developer Kit para Java:

- Javadoc JAAS específico del System i5
- Especificación de la API de JAAS
- Especificación de la API de la plataforma Java 2, Standard Edition de Sun Microsystems, Inc.

Vea la siguiente información de consulta relacionada con IBM Developer Kit para Java.

Java Naming and Directory Interface

Java Naming and Directory Interface (JNDI) forma parte de la interfaz de programa de aplicación(API) de la plataforma JavaSoft. Gracias a JNDI, se pueden establecer conexiones sin fisuras con varios servicios de directorio y denominación. Con esta interfaz, se pueden construir aplicaciones Java habilitadas para directorio que sean potentes y portables.

JavaSoft ha desarrollado la especificación JNDI con la colaboración de destacadas empresas del sector, entre ellas IBM, SunSoft, Novell, Netscape y Hewlett-Packard Co.

Nota: El entorno Java de tiempo de ejecución (JRE) de i5/OS y las versiones de la plataforma Java 2, Standard Edition (J2SE) que ofrece IBM Developer Kit para Java incluyen el proceedor LDAP de Sun. Dado que el soporte i5/OS Java incluye el suministrador LDAP Sun, ese soporte ya no incluye el archivo ibmjndi.jar. El archivo ibmjndi.jar ofrecía un suministrador de servicio de LDAP desarrollado por IBM para las versiones anteriores de J2SDK.

Java Naming and Directory interface by Sun Microsystems, Inc.

JavaMail

La API JavaMail proporciona un conjunto de clases abstractas que modela un sistema de correo electrónico (e-mail). La API proporciona funciones generales de correo para leer y enviar correo, y requiere proveedores de servicio para implementar los protocolos.

Los proveedores de servicio implementan protocolos específicos. Por ejemplo, el protocolo simple de transferencia de correo (SMTP) es un protocolo de transporte para enviar correo electrónico. El protocolo de oficina postal 3 (POP3) es el protocolo estándar para recibir correo electrónico. El protocolo de acceso a mensajes de Internet (IMAP) es un protocolo alternativo a POP3.

Además de proveedores de servicio, JavaMail necesita que la infraestructura de activación de JavaBeans (JAF) maneje el contenido de correo que no sea de texto sin formato. Esto incluye MIME (Multipurpose Internet Mail Extensions), páginas URL (Localizador Universal de Recursos) y anexos de archivo.

Todos los componentes JavaMail vienen como parte del IBM Developer Kit para Java. Estos componentes incluyen los siguientes:

- **mail.jar** Este archivo JAR contiene las API de JavaMail, el proveedor de servicio SMTP, el proveedor de servicio POP3 y el proveedor de servicio IMAP.
- **activation.jar** Este archivo JAR contiene la infraestructura de activación de the JavaBeans (JAF).

 JavaMail

Servicio de impresión Java

La API de servicio de impresión Java (JPS) permite imprimir en todas las plataformas Java. Java 1.4 y versiones ulteriores proporcionan una infraestructura en la que los entornos Java de tiempo de ejecución (JRE) y terceras partes pueden ofrecer plug-ins de generador de archivos continuos para producir diversos formatos de impresión, como PDF, Postscript y Advanced Function Presentation (AFP). Estos plug-ins generan los formatos de salida a partir de llamadas de gráficos bidimensionales (2D).

Cada servicio de impresión de System i5 representa un dispositivo de impresora configurado en el System i5 con el mandato de i5/OS Crear descripción de dispositivo (impresora) (CRTDEVPRT). Al crear un dispositivo de impresora, especifique los parámetros de información de publicación. Esto aumenta el número de atributos de servicio soportados por los servicios de impresión de System i5.

Si una impresora soporta el protocolo simple de gestión de red (SNMP), configúrela en el servidor. Especifique *IBMSNMPDRV para el valor del parámetro de programa controlador del sistema en el mandato CRTDEVPRT. Los servicios de impresión utilizan SNMP para recuperar información específica (atributos de servicio de impresora) sobre una impresora configurada.

Las variantes de documentos soportadas por System i5 son *AFPDS, *SCS, *USERASCII - (PCL), *USERASCII - (Postscript) y *USERASCII - (PDF). Especifique las Variantes de documentos a las que la impresora da soporte en el parámetro Corrientes de datos soportadas, dentro de la Información de publicación del mandato CRTDEVPRT.

Cuando una aplicación utiliza un servicio de impresión para imprimir un trabajo (documento) en el System i5, el servicio de impresión coloca el documento en un archivo en spool en una cola de salida con el mismo nombre que el dispositivo de impresora (también el mismo nombre que el especificado en el atributo PrinterName). Inicie un transcriptor de impresora con el mandato STRPRTWTR para que los documentos puedan imprimirse en el dispositivo de impresora.

Además de los atributos definidos por la especificación Servicios de impresión Java, los servicios de impresión de System i5 soportan los siguientes atributos para todas las variantes de documentos:

- PrinterFile (especifica un archivo de impresora, un nombre y una biblioteca a utilizar al crear el archivo en spool)
- SaveSpooledFile (indica si debe guardarse el archivo en spool)
- UserData (una serie de 10 caracteres de datos definidos por usuario)
- JobHold (indica si debe retenerse el archivo en spool)
- SourceDrawer (indica la bandeja alimentadora a utilizar para el soporte de salida)

Cómo habilitar JPS al utilizar JDK 1.5

A continuación se muestran los enlaces simbólicos que hay que configurar para habilitar el servicio de impresión Java:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjps.jar')
     NEWLNK('/QIBM/ProdData/Java400/jdk15/lib/ext/ibmjps.jar')
     LNKTYPE(*SYMBOLIC)
```

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
     NEWLNK('/QIBM/ProdData/Java400/jdk15/lib/ext/jt400Native.jar')
     LNKTYPE(*SYMBOLIC)
```

Información relacionada

Servicio de impresión Java (JPS) de Sun Microsystems, Inc.

Información sobre licencia de código y exención de responsabilidad

IBM le otorga una licencia de copyright no exclusiva para utilizar todos los ejemplos de código de programación, a partir de los que puede generar funciones similares adaptadas a sus necesidades específicas.

SUJETO A LAS GARANTÍAS ESTATUTARIAS QUE NO PUEDAN EXCLUIRSE, IBM, LOS DESARROLLADORES Y LOS SUMINISTRADORES DE PROGRAMAS NO OFRECEN NINGUNA GARANTÍA NI CONDICIÓN, YA SEA IMPLÍCITA O EXPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS O CONDICIONES IMPLÍCITAS DE COMERCIALIZACIÓN, ADECUACIÓN A UN PROPÓSITO DETERMINADO Y NO VULNERACIÓN CON RESPECTO AL PROGRAMA O AL SOPORTE TÉCNICO, SI EXISTE.

BAJO NINGUNA CIRCUNSTANCIA, IBM, LOS DESARROLLADORES O SUMINISTRADORES DE PROGRAMAS DE IBM SE HACEN RESPONSABLES DE NINGUNA DE LAS SIGUIENTES SITUACIONES, NI SIQUIERA EN CASO DE HABER SIDO INFORMADOS DE TAL POSIBILIDAD:

1. PÉRDIDA DE DATOS O DAÑOS CAUSADOS EN ELLOS;
2. DAÑOS ESPECIALES, ACCIDENTALES, DIRECTOS O INDIRECTOS, O DAÑOS ECONÓMICOS DERIVADOS;
3. PÉRDIDAS DE BENEFICIOS, COMERCIALES, DE INGRESOS, CLIENTELA O AHORROS ANTICIPADOS.

ALGUNAS JURISDICCIONES NO PERMITEN LA EXCLUSIÓN O LA LIMITACIÓN DE LOS DAÑOS DIRECTOS, ACCIDENTALES O DERIVADOS, POR LO QUE PARTE DE LAS LIMITACIONES O EXCLUSIONES ANTERIORES, O TODAS ELLAS, PUEDE NO SER PROCEDENTE EN SU CASO.

Apéndice. Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en los EE.UU.

IBM puede no ofrecer los productos, servicios o características tratados en este documento en otros países. Póngase en contacto con el representante local de IBM que le informará sobre los productos y servicios disponibles actualmente en su área. Las referencias hechas a productos, programas o servicios de IBM no pretenden afirmar ni dar a entender que únicamente puedan utilizarse dichos productos, programas o servicios de IBM. Puede utilizarse en su lugar cualquier otro producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. No obstante, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patente pendientes de aprobación que cubran temas descritos en este documento. La posesión de este documento no le otorga ninguna licencia sobre dichas patentes. Puede enviar consultas sobre las licencias, por escrito, a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
Estados Unidos

Para consultas sobre licencias relativas a la información de doble byte (DBCS), póngase en contacto con el departamento de propiedad intelectual de IBM en su país o envíe las consultas, por escrito, a:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japón

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país en que dichas disposiciones entren en contradicción con las leyes locales: INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN GARANTÍA DE NINGÚN TIPO, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO, PERO NO LIMITÁNDOSE, A LAS GARANTÍAS IMPLÍCITAS DE NO VULNERABILIDAD, COMERCIALIZACIÓN O ADECUACIÓN A UN PROPÓSITO DETERMINADO. Algunas legislaciones no contemplan la declaración de limitación de responsabilidad, ni implícita ni explícita, en determinadas transacciones, por lo que cabe la posibilidad de que esta declaración no se aplique en su caso.

Esta información puede contener imprecisiones técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información incluida en este documento; estos cambios se incorporarán en nuevas ediciones de la publicación. IBM puede efectuar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento y sin previo aviso.

Cualquier referencia hecha en esta información a sitios Web no de IBM se proporciona únicamente para su comodidad y no debe considerarse en modo alguno como promoción de esos sitios Web. Los materiales de estos sitios Web no forman parte de los materiales de IBM para este producto y el uso que se haga de estos sitios Web es de la entera responsabilidad del usuario.

IBM puede utilizar o distribuir la información que usted le suministre del modo que IBM considere conveniente sin incurrir por ello en ninguna obligación para con usted.

Los licenciatarios de este programa que deseen obtener información acerca de él con el fin de: (i) intercambiar la información entre los programas creados independientemente y otros programas (incluido este) y (ii) utilizar mutuamente la información que se ha intercambiado, deben ponerse en contacto con:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
Estados Unidos

Esta información puede estar disponible, sujeta a los términos y condiciones pertinentes, e incluir en algunos casos el pago de una cantidad.

El programa bajo licencia descrito en este documento, así como todo el material bajo licencia disponible para él, lo proporciona IBM según los términos del Acuerdo de Cliente de IBM, el Acuerdo Internacional de Programas bajo Licencia de IBM, el Acuerdo de Licencia para Código de Máquina de IBM o cualquier otro acuerdo equivalente entre ambas partes.

Los datos de rendimiento aquí contenidos se han determinado en un entorno controlado. Por consiguiente, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Pueden haberse realizado mediciones en sistemas en nivel de desarrollo, por lo que no hay garantía de que dichas mediciones sean iguales en los sistemas disponibles en general. Incluso algunas de esas mediciones pueden haberse estimado mediante extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deberán verificar los datos aplicables a su entorno específico.

La información concerniente a productos no IBM se ha obtenido de los suministradores de esos productos, de sus anuncios publicados o de otras fuentes de información pública disponibles. IBM no ha comprobado los productos y no puede afirmar la exactitud en cuanto a rendimiento, compatibilidad u otras características relativas a productos no IBM. Las consultas acerca de las posibilidades de productos no IBM deben dirigirse a los suministradores de los mismos.

Todas las declaraciones respecto a las intenciones futuras de IBM están sujetas a cambios o a su retirada sin aviso, y solamente representan metas y objetivos.

Todos los precios de IBM que se muestran son precios al detalle aconsejados, actuales pero susceptibles de ser modificados sin previo aviso. Los precios de distribuidor pueden variar.

Esta información se utiliza solo para proyectos de planificación. La información aquí indicada puede estar sujeta a cambios antes de que el producto que describe esté disponible.

Esta información contiene ejemplos de datos e informes utilizados en operaciones comerciales diarias. Para ilustrarlos tan completamente como sea posible, los ejemplos pueden incluir nombres de individuos, compañías, marcas y productos. Todos estos nombres son ficticios y cualquier parecido con los nombres y direcciones utilizados por una empresa real es pura coincidencia.

LICENCIA DE COPYRIGHT:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente, que muestran técnicas de programación en varias plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo de cualquier forma sin pagar nada a IBM, bajo el propósito de desarrollo, uso, márketing o distribución de programas de aplicación de acuerdo con la interfaz de programación de la aplicación para la plataforma operativa para la cual se han escrito los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por tanto, IBM no puede garantizar la fiabilidad, capacidad de servicio o funcionamiento de estos programas.

Cada copia o parte de estos programas de ejemplo o de los trabajos derivados de ellos, debe incluir un aviso de copyright de este tipo:

(C) (el nombre de la empresa) (año). Partes de este código derivan de IBM Corp. Programas de ejemplo. (C) Copyright IBM Corp. _entre el año o años_. Reservados todos los derechos.

Si está viendo esta información en copia software, las fotografías y las ilustraciones a color podrían no aparecer.

Información de la interfaz de programación

Esta documentos de publicación del IBM Developer Kit para Javaestán destinados a programar Interfaces que permitan al cliente escribir programas para obtener los servicios de IBM Developer Kit para Java.

Marcas registradas

Los términos siguientes son marcas registradas de International Business Machines Corporation en Estados Unidos y/o en otros países:

Advanced Function Presentation
AFP
AIX
alphaWorks
C/400
DB2
DB2 Universal Database
developerWorks
Distributed Relational Database Architecture
DRDA
i5/OS
IBM
Integrated Language Environment
iSeries
OS/400
PowerPC
System i
System i5
VisualAge
WebSphere

Adobe, el logotipo de Adobe, PostScript y el logotipo de PostScript son marcas registradas de Adobe Systems Incorporated en Estados Unidos y/o en otros países.

Microsoft, Windows, Windows NT, y el logotipo de Windows son marcas registradas de Microsoft Corporation en los Estados Unidos y/o en otros países.

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en los Estados UNidos y/o en otros países.

UNIX es una marca registrada de The Open Group en los Estados Unidos y en otros países.

Los demás nombres de compañías, productos o servicios pueden ser marcas registradas o de servicio de terceros.

Términos y condiciones

Los permisos para utilizar estas publicaciones están sujetos a los siguientes términos y condiciones.

Uso personal: puede reproducir estas publicaciones para uso personal (no comercial) siempre y cuando incluya una copia de todos los avisos de derechos de autor. No puede distribuir ni visualizar estas publicaciones ni ninguna de sus partes, como tampoco elaborar trabajos que se deriven de ellas, sin el consentimiento explícito de IBM.

Uso comercial: puede reproducir, distribuir y visualizar estas publicaciones únicamente dentro de su empresa, siempre y cuando incluya una copia de todos los avisos de derechos de autor. No puede elaborar trabajos que se deriven de estas publicaciones, ni tampoco reproducir, distribuir ni visualizar estas publicaciones ni ninguna de sus partes fuera de su empresa, sin el consentimiento explícito de IBM.

Aparte de la autorización que se concede explícitamente en este permiso, no se otorga ningún otro permiso, licencia ni derecho, ya sea explícito o implícito, sobre las publicaciones, la información, los datos, el software o cualquier otra propiedad intelectual contenida en ellas.

IBM se reserva el derecho de retirar los permisos aquí concedidos siempre que, según el parecer del fabricante, se utilicen las publicaciones en detrimento de sus intereses o cuando, también según el parecer del fabricante, no se sigan debidamente las instrucciones anteriores.

No puede bajar, exportar ni reexportar esta información si no lo hace en plena conformidad con la legislación y normativa vigente, incluidas todas las leyes y normas de exportación de Estados Unidos.

IBM NO PROPORCIONA NINGUNA GARANTÍA SOBRE EL CONTENIDO DE ESTAS PUBLICACIONES. LAS PUBLICACIONES SE PROPORCIONAN ″TAL CUAL″, SIN GARANTÍA DE NINGUNA CLASE, YA SEA EXPLÍCITA O IMPLÍCITA, INCLUIDAS, PERO SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN, NO VULNERACIÓN E IDONEIDAD PARA UN FIN DETERMINADO.

Impreso en España