IBM

IBM 系統 - iSeries
程式設計
IBM Developer Kit for Java

版本 5 版次 4

IBM

IBM 系統 - iSeries
程式設計
IBM Developer Kit for Java

版本 5 版次 4

請注意

使用此資訊及其支援的產品之前，請先閱讀第 499 頁的『注意事項』中的資訊。

第十版 (2006 年 2 月)

此版本適用於 IBM Developer Kit for Java (產品編號 5722-JV1) 版本 5 版次 4 修正層次 0，以及所有後續的版次和修訂版 (除非新版中另有指示)。此版本並非適用於所有的精簡指令集電腦 (RISC) 機型和 CISC 機型。

目錄

IBM Developer Kit for Java

IBM Developer Kit for Java™ 是針對 iSeries™ 伺服器環境而最佳化的產品。由於採納 Java 程式設計與使用者介面的相容性，您可以針對 iSeries 伺服器來開發自己的應用程式。

IBM® Developer Kit for Java 可讓您在 iSeries 伺服器上建立及執行 Java 程式。IBM Developer Kit for Java 是 Sun Microsystems 公司 Java Technology 的相容實作方式，因此，我們假設您已熟悉相關的 Java Development Kit (JDK) 文件。爲了讓您更易於使用兩者的資訊，本文會提供 Sun Microsystems, Inc. 資訊的鏈結。

萬一 Sun Microsystems 公司 Java Development Kit 文件的鏈結失效，請參考他們的 HTML 參照文件以獲得所需的資訊。在全球資訊網上，您可以造訪 The Source for Java Technology java.sun.com，以取得這項資訊。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

新增功能

本主題的重點在於 IBM Developer Kit for Java 在 V5R4 中的變更。

新除錯介面

第 361 頁的『Java Platform Debugger Architecture』 及 第 341 頁的『Java 虛擬機器 Profiler 介面』主題說明「Java 虛擬機器工具介面 (JVMTI)」。

新 CL 指令

如需使用「顯示 Java 虛擬機器工作 (DSPJVMJOB)」指令的相關資訊，請參閱第 495 頁的『套用暫時修訂程式』及 第 351 頁的『Java 支援的 CL 指令』主題。

新 Java Cryptography Extension 提供者

如需 IBMJCEFIPS JCE 提供者的相關資訊，請參閱 第 236 頁的『Java Cryptography Extension』主題。

新內容

如需更新的 J2SE 5.0 版內容，請參閱 第 12 頁的『Java 系統內容的清單』。

新版本選項

若要安裝 J2SE 5.0 版及其他 JDK 版本，請參閱第 3 頁的『多重 Java 2 Software Development Kit 的支援』主題。

新 Java 工具：

- 第 346 頁的『Java apt 工具』

- 第 348 頁的『Java pack200 工具』
- 第 350 頁的『Java unpack200 工具』

如何查看新增內容或變更內容

爲了協助您易於查閱技術變更之處，本資訊內容使用：

- 圖示，標示新增或變更資訊開始的位置。
- 圖示，標示新增或變更資訊結束的位置。

如需本版次新增或變更功能的其他相關資訊，請參閱使用者備忘錄。

可列印的 PDF

遵循下列步驟來檢視及下載此主題的 PDF。

若要檢視或下載 PDF 版本，請選取 IBM Developer Kit for Java (約 4585 KB)。

儲存 PDF 檔

若要儲存 PDF 至您的工作站，以方便您檢視或列印，請：

1. 以滑鼠右鍵按一下瀏覽器內的 PDF (以滑鼠右鍵按一下上述的鏈結)。
2. 按一下選項以本端儲存 PDF。
3. 瀏覽至您要儲存此 PDF 的目錄。
4. 按一下**儲存**。

下載 Adobe Reader

您需要在系統上安裝 Adobe Reader 才能檢視或列印 PDF。 您可以從 Adobe 網站 (www.adobe.com/products/acrobat/readstep.html) 免費下載。

安裝及配置 IBM Developer Kit for Java

若您未曾用過 IBM Developer Kit for Java，請遵循下列步驟來安裝、配置它，然後練習執行簡單的 Hello World Java 程式。

第 1 頁的『新增功能』
本主題的重點在於 IBM Developer Kit for Java 在 V5R4 中的變更。

第 10 頁的『針對 IBM Developer Kit for Java 自訂 iSeries 伺服器』
在 iSeries 伺服器上安裝 IBM Developer Kit for Java 之後，即可開始自訂您的伺服器。

第 5 頁的『下載及安裝 Java 套件』
若要在 iSeries 伺服器上更有效率地下載、安裝及使用 Java 套件，請參閱下列主題。

第 28 頁的『版次相容性』
Java 類別檔案只要不採用 Sun 已捨棄或已變更支援的功能，即向上相容 (JDK 1.1.x -> 1.2.x -> 1.3.x -> 1.4.x -> 1.5.x)。

安裝 IBM Developer Kit for Java

安裝 IBM Developer Kit for Java 可讓您在 iSeries 伺服器上建立及執行 Java 程式。

系統 CD 附有授權程式 5722-JV1，因此 JV1 已預先安裝。請輸入「跳至授權程式 (GO LICPGM)」指令，然後選取「選項 10」(顯示)。若未看到此授權程式列示出來，請執行下列步驟：

1. 在指令行輸入 GO LICPGM 指令。
2. 選取選項 11 (安裝授權程式)。
3. 針對授權程式 (LP) 5722-JV1 *BASE 選擇選項 1 (安裝)，再針對您要安裝的 Java Development Kit (JDK) 選取符合的選項。若清單未列出您要安裝的選項，請在「選項」欄位中輸入選項 1 (安裝)，將它加入清單中。在「授權程式」欄位中輸入 5722JV1，再於「產品選項」欄位中輸入您的選項號碼。

   附註：您一次可以安裝多個選項。

在 iSeries 伺服器上安裝了 IBM Developer Kit for Java 之後，即可選擇自訂您的系統。

如需 IBM Developer Kit for Java 的入門資訊，請參閱執行第一個 Hello World Java 程式。

使用「復置授權程式」指令來安裝授權程式

*安裝授權程式*畫面中列出的程式，就是您的伺服器在最初安裝好時，LICPGM 安裝環境所支援的程式。偶而會有新的程式出現，但並不在您的伺服器授權程式之列。萬一您要安裝的程式剛好是這種情形，則必須使用「復置授權程式 (RSTLICPGM)」指令來安裝。

若要使用「復置授權程式 (RSTLICPGM)」指令來安裝授權程式，請遵循下列步驟：

1. 在適當的光碟機內放入含有授權程式的磁帶或 CD-ROM。
2. 在 iSeries 指令行上鍵入：

   RSTLICPGM

   然後按 Enter 鍵。

   此時會出現*復置授權程式 (RSTLICPGM)* 畫面。
3. 在*產品*欄位中，鍵入您要安裝的授權程式 ID 號碼。
4. 在*裝置*欄位中，指定安裝裝置。

   附註：若是從磁帶機安裝，則裝置 ID 通常是 **TAP*xx*** 這種格式，其中 ***xx*** 代表數字，例如 **01**。
5. 至於*復置授權程式*畫面的其他參數，請維持預設值。接著按 Enter 鍵。
6. 此時會出現更多參數。同樣也請保留這些預設值。接著按 Enter 鍵。程式就會開始安裝。

當授權程式安裝完成時，*復置授權程式*畫面會再次出現。

多重 Java 2 Software Development Kit 的支援

iSeries 伺服器支援多個版本的 Java Development Kit (JDK) 及 Java 2 Software Development Kit (J2SDK) 標準版。

註: 在本文件中，JDK 術語指任何支援的 JDK 及 J2SDK 版本，視上下文而定。在出現 JDK 的地方，通常也會指出特定的版本及版次號碼。

iSeries 伺服器支援同時使用多套 JDK，但必須透過多個 Java 虛擬機器。單一 Java 虛擬機器僅可執行一套指定的 JDK。

請找出您正在使用或打算使用的 JDK，然後選取相對的選項來安裝。 若要同時安裝多套 JDK，請參閱第 2 頁的『安裝 IBM Developer Kit for Java』。java.version 系統內容會決定應該執行哪個 JDK 版本。 Java 虛擬機器一旦開始執行，再變更 java.version 系統內容就沒有效果。

附註：在 V5R3 及更高版本中，不再提供下列選項：選項 1 (JDK 1.1.6)、選項 2 (JDK 1.1.7)、選項 3 (JDK 1.2.2) 及選項 4 (JDK 1.1.8)。下表列出這個版次支援的 J2SDK。

選項	JDK	java.home	java.version
5	1.3	/QIBM/ProdData/Java400/jdk13/	1.3
6	1.4	/QIBM/ProdData/Java400/jdk14/	1.4
7	1.5 (也稱爲 J2SE 5.0)	/QIBM/ProdData/Java400/jdk15/	1.5

在此多重 JDK 環境中選擇的預設 JDK，取決於已安裝的 5722-JV1 選項。下表將提供一些範例。

安裝	輸入	結果
選項 5 (1.3)	`java Hello`	執行 J2SDK 標準版 1.3 版。
選項 6 (1.4)	`java Hello`	執行 J2SDK 標準版 1.4 版。
選項 5 (1.3) 及選項 6 (1.4)	`java Hello`	執行 J2SDK 標準版 1.4 版。

註: 若僅安裝一套 JDK，則預設的 JDK 即爲您所安裝的版本。若安裝了多套 JDK，則依下列優先順序來決定預設的 JDK：

1. 選項 6 (1.4)
2. 選項 5 (1.3)
3. 選項 7 (1.5)

安裝 IBM Developer Kit for Java 的延伸套件

延伸套件是指可用來延伸核心平台功能的 Java 類別套件。延伸套件會封裝成一或多個 ZIP 檔或 JAR 檔，由延伸類別載入器載入 Java 虛擬機器內。

此延伸機制可讓 Java 虛擬機器使用延伸類別，就像虛擬機器使用系統類別一樣。若 J2SDK 1.2 版或更高版本，或 Java 2 Runtime Environment 標準版 1.2 版及更高版本尚未安裝延伸套件，這項延伸機制亦可讓您從指定的「全球資源定位器 (URL)」擷取延伸套件。

iSeries 伺服器隨附了一些延伸套件的 JAR 檔。如需安裝這些延伸套件，請輸入下列指令：

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

其中，

`extensionToInstall.jar`

是指 ZIP 或 JAR 檔的名稱，內含您要安裝的延伸套件。

註: IBM 未提供的延伸套件 JAR 檔，可能放在 /QIBM/UserData/Java400/ext 目錄中。

對於 /QIBM/UserData/Java400/ext 目錄中的延伸套件，當您建立鏈結或加入檔案時，延伸類別載入器所搜尋的檔案清單會隨著*執行於 iSeries 伺服器的每一部 Java 虛擬機器*而變更。若不希望影響 iSeries 伺服器上其他 Java 虛擬機器的延伸類別載入器，但仍然想要建立延伸套件的鏈結，或想要安裝 IBM 未附在 iSeries 伺服器中的延伸套件，請遵循下列步驟：

1. 建立目錄來安裝延伸套件。請在 iSeries 指令行上使用「建立目錄 (MKDIR)」指令，或在「Qshell 直譯器」中使用 `mkdir` 指令。

2. 在已建立的目錄中放入延伸套件 JAR 檔。
3. 在 java.ext.dirs 內容中新增目錄。您可以從 iSeries 指令行中，使用 JAVA 指令的 PROP 欄位，藉以在 java.ext.dirs 內容中新增目錄。

若新目錄的名稱是 /home/username/ext、延伸套件的檔名是 extensionToInstall.jar、Java 程式的名稱是 Hello，則您應輸入的指令如下：

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

CPY OBJ('/productA/extensionToInstall.jar') TODIR('/home/username/ext') 或
使用 FTP (檔案傳送通訊協定) 將檔案複製到 /home/username/ext。
JAVA Hello PROP((java.ext.dirs '/home/username/ext'))
```

下載及安裝 Java 套件

若要在 iSeries 伺服器上更有效率地下載、安裝及使用 Java 套件，請參閱下列主題。

具有圖形式使用者介面的套件

以圖形式使用者介面 (GUI) 操作的 Java 程式，需要使用具備圖形顯示能力的呈現裝置。例如，個人電腦、技術工作站或網路電腦。您可以使用 Native Abstract Windowing Toolkit (NAWT)，讓 Java 應用程式及 Servlet 具備 Java 2 Software Development Kit (J2SDK) 標準版 Abstract Windowing Toolkit (AWT) 的完整圖形功能。如需相關資訊，請參閱 Native Abstract Windowing Toolkit (NAWT)。

區分大小寫及整合檔案系統

整合檔案系統提供的檔案系統，不僅區分大小寫，且與檔名無關。例如，QOpenSys 即爲整合檔案系統內一種區分大小寫的檔案系統範例。相反地，Root (/) 是一種不區分大小寫的檔案系統。如需詳細資訊，請參閱整合檔案系統主題。

雖然 JAR 或類別可能位於不區分大小寫的檔案系統中，但 Java 仍然是一種區分大小寫的語言。例如，`wrklnk '/home/Hello.class'` 與 `wrklnk '/home/hello.class'` 的結果相同，但 `JAVA CLASS(Hello)` 與 `JAVA CLASS(hello)` 則會呼叫不同的類別。

ZIP 檔處理及 JAR 檔處理

ZIP 檔及 JAR 檔包含一組 Java 類別。對這些檔案執行建立 Java 程式 (CRTJVAPGM) 指令時，類別會經過驗證，然後轉換成內部機器碼格式，最後再轉換成 iSeries 機器碼 (若有指定)。您可以將 ZIP 檔及 JAR 檔視爲其他任何個別的類別檔案一樣。若其中一個檔案有一個相關聯的內部機器碼格式，則會繼續維持這種檔案關聯性。以後執行時，內部機器碼格式就會代替類別檔案來提高效能。若不確定目前的 Java 程式與您的類別檔案或 JAR 檔是否有關聯，請使用顯示 Java 程式 (DSPJVAPGM) 指令，顯示 iSeries 伺服器上 Java 程式的相關資訊。

在前版次的 IBM Developer Kit for Java 中，一旦 JAR 檔或 ZIP 檔有任何變動，就必須重建 Java 程式，因爲相關的 Java 程式會變成無法使用。但現在不用這麼麻煩。在許多情況下，即使變更 JAR 檔或 ZIP 檔，Java 程式仍然有效，不必再重建。若只是局部變更，例如僅更新 JAR 檔內的單一類別檔案，只要重建 JAR 檔內受影響的類別檔案即可。

對 JAR 檔進行最常見的變更之後，仍然會繼續保持 Java 程式與 JAR 檔的關聯。例如，在下列情況中，仍然會保持這些 Java 程式與 JAR 檔的關聯：

- 使用 ajar 工具來變更或重建 JAR 檔。
- 使用 jar 工具來變更或重建 JAR 檔。
- 使用 0S/400 COPY 指令或 Qshell cp 公用程式來置換 JAR 檔。

如果您透過 iSeries Access for Windows® 或個人電腦 (PC) 上對映的磁碟機來存取整合檔案系統中的 JAR 檔，則即使在下列情況中，這些 Java 程式與 JAR 檔的關聯仍會繼續保持：

- 將另一個 JAR 檔拖放到現有的整合檔案系統 JAR 檔之上。
- 使用 jar 工具來變更或重建整合檔案系統 JAR 檔。
- 使用 PC copy 指令來置換整合檔案系統 JAR 檔。

當變更或置換 JAR 檔時，相關聯的 Java 程式就不再是最新的。

有一種例外情況，Java 程式與 JAR 檔會失去關聯性。如果使用檔案傳送通訊協定 (FTP) 來置換 JAR 檔，就會毀損相關的 Java 程式。例如，使用 FTP put 指令來置換 JAR 檔，便會發生此情況。

如需 JAR 檔的效能性質的詳細資訊，請參閱 Java Runtime 效能。

Java 延伸組織架構

在 J2SDK 中，延伸套件是指 Java 類別套件，可用來延伸核心平台的功能。延伸套件或應用程式會封裝成一或多個 JAR 檔。此延伸機制可讓 Java 虛擬機器使用延伸類別，就像虛擬機器使用系統類別一樣。若 J2SDK 或 Java 2 Runtime Environment 標準版尚未安裝延伸套件，這項延伸機制亦可讓您從指定的 URL 擷取延伸套件。

如需安裝延伸套件的相關資訊，請參閱安裝 IBM Developer Kit for Java 的延伸套件。

執行第一個 Hello World Java 程式

本主題將協助您執行第一個程式。

下列方法可讓您的 Hello World Java 程式啓動及開始執行：

1. 直接執行 IBM Developer Kit for Java 隨附的 Hello World Java 程式。

   若要執行提供的程式，請執行下列步驟：

   a. 輸入「跳至授權程式 (GO LICPGM)」指令，檢查是否已安裝 IBM Developer Kit for Java。然後選取選項 10 (顯示安裝的授權程式)。驗證授權程式 5722-JV1 *BASE 及其中至少一個選項顯示爲已安裝。
   b. 在「iSeries 主功能表」指令行輸入 `java Hello`。按 Enter 鍵來執行 Hello World Java 程式。
   c. 若已正確安裝 IBM Developer Kit for Java，則 `Hello World` 會出現在「Java Shell 顯示畫面」中。請按 F3 (結束) 或 F12 (結束)，回到輸入指令畫面。
   d. 若 Hello World 類別未執行，請檢查是否已順利完成安裝，或參閱取得 IBM Developer Kit for Java 的支援來取得服務資訊。

2. 亦可執行您自己的 Hello Java 程式。如需如何建立自己的 Hello Java 程式的相關資訊，請參閱建立、編譯及執行 Hello World Java 程式。

將網路磁碟機對映至 iSeries 伺服器

若要對映網路磁碟機，請完成下列步驟。

1. 請確定 iSeries Access for Windows 已安裝在您的伺服器及工作站上。如需如何安裝及配置 iSeries Access for Windows 的相關資訊，請參閱安裝 iSeries Access for Windows。您必須先配置 iSeries 伺服器的連線，才能夠對映網路磁碟機。
2. 開啓「Windows 檔案總管」：

   a. 以滑鼠右鍵按一下 Windows 工作列的**開始**按鈕。
   b. 在功能表中按一下**檔案總管**。

3. 從**工具**功能表中，選取**連線網路磁碟機**。

4. 選取您要用來連接 iSeries 伺服器的磁碟機。
5. 鍵入伺服器的路徑名稱。例如，*MYSERVER*，其中 *MYSERVER* 是 iSeries 伺服器的名稱。
6. 勾選**登入時重新連線**框 (若未勾選)。
7. 按一下**完成**結束。

現在，您對映的磁碟機會出現在「Windows 檔案總管」的**所有資料夾**區段中。

在 iSeries 伺服器上建立目錄

您必須在 iSeries 伺服器上建立可儲存 Java 應用程式的目錄。

有兩種作法：

- 『使用 iSeries 領航員建立目錄』

  若已安裝 iSeries Access for Windows，請選擇這個選項。若您打算使用「iSeries 領航員」編譯、最佳化及執行 Java 程式，則必須選取此選項，以確使程式儲存於正確的位置，方能執行這些作業。
- 『使用指令輸入行來建立目錄』

  若尚未安裝 iSeries Access for Windows，請選擇此選項。

如需「iSeries 領航員」的相關資訊，包括安裝資訊，請參閱 iSeries 領航員入門。

使用 iSeries 領航員建立目錄

若要在 iSeries 伺服器上建立目錄，請遵循下列步驟：

1. 開啓「iSeries 領航員」。
2. 在**我的連線**視窗中，連按兩下您的伺服器名稱來登入。若**我的連線**視窗中未列出您的伺服器，請遵循下列步驟來新增：
   a. 按一下**檔案 --> 新增連線....**
   b. 在**系統**欄位中，鍵入您的伺服器名稱。
   c. 按**下一步**。
   d. 在**使用預設使用者 ID，必要時提示**欄位中，輸入您的使用者 ID (若尚未輸入)。
   e. 按**下一步**。
   f. 按一下**驗證連線**。這麼做可確認您可以連接伺服器。
   g. 按一下**完成**。
3. 在您要使用的連線下方展開資料夾。尋找名爲**檔案系統**的資料夾。若沒看到此資料夾，則表示「iSeries 領航員」安裝期間未選取安裝「檔案系統」這個選項。您必須選取**開始 --> 程式集 --> IBM iSeries Access for Windows --> 選擇性安裝**，來安裝「iSeries 領航員」的「檔案系統」選項。
4. 展開**檔案系統**資料夾，找出**整合檔案系統**資料夾。
5. 展開**整合檔案系統**資料夾，再展開 **Root** 資料夾。展開 **Root** 資料夾之後，您看到的結構就如同在 iSeries 指令行執行 WRKLNK ('/') 指令一樣。
6. 以滑鼠右鍵按一下您要新增子目錄的資料夾。選取**新建資料夾**，然後輸入您要建立的子目錄名稱。

使用指令輸入行來建立目錄

若要在 iSeries 伺服器上建立目錄，請遵循下列步驟：

1. 登入 iSeries 伺服器。

2. 在指令行鍵入：

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

   其中 *mydir* 即爲您建立的目錄名稱。

   按 **Enter** 鍵。

   螢幕底端會出現訊息，顯示**已建立目錄**。

建立、編譯及執行 HelloWorld Java 程式

建立簡單的 Hello World Java 程式，是熟悉 IBM Developer Kit for Java 的一個不錯的途徑。

若要建立、編譯及執行您自己的 Hello World Java 程式，請執行下列步驟：

1. 將網路磁碟機對映至 iSeries 伺服器。
2. 針對您的 Java 應用程式在 iSeries 伺服器上建立目錄。
3. 在整合檔案系統上建立一個「美國國家標準交換碼 ASCII」文字檔格式的來源檔。編寫 Java 應用程式時，您可以採用整合開發環境 (IDE) 產品，或 Windows 記事本之類的文字型編輯器。
   a. 將文字檔命名爲 `HelloWorld.java`。如需如何建立及編輯檔案的相關資訊，請參閱建立及編輯 Java 來源檔。
   b. 確定檔案包含下列原始程式碼：

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

4. 編譯來源檔。
   a. 輸入「處理環境變數 (WRKENVVAR)」指令來檢查 CLASSPATH 環境變數。若 CLASSPATH 變數不存在，請加以新增並設定爲 '.' (現行目錄)。若 CLASSPATH 變數存在，請確定 '.' 位於路徑名稱清單的開頭。如需 CLASSPATH 環境變數的詳細資料，請參閱 Java 類別路徑。
   b. 輸入「啓動 Qshell (STRQSH)」指令來啓動 Qshell 直譯器。
   c. 使用切換目錄 (cd) 指令，將現行目錄切換至 `HelloWorld.java` 檔案所在的整合檔案系統目錄。
   d. 輸入 `javac`，後面接著您要儲存於磁碟上的檔案名稱。例如輸入 `javac HelloWorld.java`。
5. 設定類別檔案在整合檔案系統中的檔案權限。
6. 最佳化 Java 應用程式。
   a. 在 *QSH 輸入指令*行上，鍵入：

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

      其中，*mydir* 是儲存 Java 應用程式之目錄的路徑名稱，而 *myclass* 則是編譯後的 Java 應用程式名稱。

      附註：最高可以指定最佳化等級 40。最佳化等級 40 雖可提高 Java 應用程式的效率，但卻也限制除錯的能力。在 Java 應用程式的早期開發階段，可選擇將最佳化等級設爲 20，對應用程式較易於除錯。如需相關資訊，請參閱 CRTJVAPGM 指令及 OPTIMIZE 參數。
   b. 按 **Enter** 鍵。

      此時會出現訊息，表示已爲您的類別建立 Java 程式。
7. 執行類別檔案。
   a. 確認 Java 類別路徑設定正確無誤。

b. 在 Qshell 指令行上鍵入 `java` 及 `HelloWorld`，開始在 Java 虛擬機器上執行 `HelloWorld.class`。例如輸入 `java HelloWorld`。亦可在 iSeries 伺服器上使用「執行 Java (RUNJVA)」指令來執行 `HelloWorld.class`。

c. 若一切輸入正確，`"Hello World"` 會列印在螢幕上。畫面上會出現 shell 提示 (預設是 $)，表示 Qshell 處於可接受另一個指令的狀態。

d. 請按 F3 (跳出) 或 F12 (切斷)，回到輸入指令畫面。

另外，亦可透過「iSeries 領航員」這個可供您在 iSeries 伺服器上執行作業的圖形式使用者介面，輕鬆地編譯、最佳化及執行 Java 應用程式。如需相關指示，請參閱第 352 頁的『Java 支援的 iSeries 領航員指令』。如需「iSeries 領航員」的相關資訊，包括安裝資訊，請參閱瞭解 iSeries 領航員。

建立及編輯 Java 來源檔

您有許多方法可以建立及編輯 Java 來源檔。

使用 iSeries Access for Windows

在 iSeries 伺服器上，Java 來源檔是整合檔案系統中的「美國國家標準交換碼 (ASCII)」文字檔。

您可以使用 iSeries Access for Windows 及工作站型的編輯器來建立及編輯 Java 來源檔。

在工作站上

您可以在工作站上建立 Java 來源檔。然後，透過「檔案傳送通訊協定 (FTP)」，將檔案傳送至整合檔案系統。

若要在工作站上建立及編輯 Java 來源檔，請：

1. 在工作站上選擇一種編輯器來建立 ASCII 檔案。
2. 透過 FTP 連接 iSeries 伺服器。
3. 以二進位檔案的格式，將來源檔傳送至整合檔案系統中的目錄，讓檔案保持 ASCII 格式。

使用 EDTF

您可以使用 EDTF CL 指令，編輯來自任何檔案系統的檔案。這是一個類似「原始檔登錄公用程式 (SEU)」的編輯器，可用於編輯串流檔或資料庫檔案。如需相關資訊，請參閱 EDTF CL 指令。

使用原始檔登錄公用程式

您可以利用原始檔登錄公用程式 (SEU)，建立文字檔格式的 Java 來源檔。

若要使用 SEU 建立文字檔格式的 Java 來源檔，請執行下列步驟：

1. 使用 SEU 建立來源檔成員。
2. 使用「複製到串流檔 (CPYTOSTMF)」指令，將來源檔成員複製成爲一個整合檔案系統串流檔，同時將資料轉換成 ASCII。

如需變更原始程式碼，請使用 SEU 來變更資料庫成員，然後再重新複製檔案。

如需儲存檔案的相關資訊，請參閱整合檔案系統中的檔案。

針對 IBM Developer Kit for Java 自訂 iSeries 伺服器

在 iSeries 伺服器上安裝 IBM Developer Kit for Java 之後，即可開始自訂您的伺服器。

關於可能的自訂事項，請參閱下列資訊：

Java 類別路徑

Java(TM) 虛擬機器在執行時間使用 Java 類別路徑來尋找類別。Java 指令及工具亦使用類別路徑來尋找類別。預設系統類別路徑、CLASSPATH 環境變數及類別路徑指令參數，皆會決定尋找特定類別時所搜尋的目錄。

在 Java 2 Software Development Kit (J2SDK) 標準版中，java.ext.dirs 內容將決定載入之延伸套件的類別路徑。如需相關資訊，請參閱安裝 IBM Developer Kit for Java 的延伸套件。

預設啓動類別路徑是由系統定義，不應該變更。在伺服器上，預設啓動類別路徑將指定到何處尋找屬於 IBM Developer Kit、Native Abstract Window Toolkit (NAWT) 的類別，以及其他系統類別。

若要尋找系統上的其他類別，您必須使用 CLASSPATH 環境變數或類別路徑參數來指定要搜尋的類別路徑。工具或指令上使用的類別路徑參數，可以置換 CLASSPATH 環境變數中指定的值。

您可以使用「處理環境變數 (WRKENVVAR)」指令來處理 CLASSPATH 環境變數。您可以在 WRKENVVAR 顯示畫面上新增或變更 CLASSPATH 環境變數。使用「新增環境變數 (ADDENVVAR)」指令及「變更環境變數 (CHGENVVAR)」指令，可新增或變更 CLASSPATH 環境變數。

CLASSPATH 環境變數的值是一連串以冒號 (:) 隔開的路徑名稱，可用來搜尋特定的類別。路徑名稱是一連串的零或目錄名稱。這些目錄名稱的後面，緊接著要在整合檔案系統中搜尋的目錄、ZIP 檔或 JAR 檔的名稱。路徑名稱的元件以斜線 (/) 字元隔開。可以使用句點 (.) 來表示現行工作目錄。

在 Qshell 環境中，您可以使用 Qshell 直譯器所提供的匯出公用程式來設定 CLASSPATH 變數。

這些指令可在 Qshell 環境中加入 CLASSPATH 變數，並將值設定爲 `.:/myclasses.zip:/Product/classes`。

- 下列指令可在 Qshell 環境中設定 CLASSPATH 變數：

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

- 下列指令可從指令行設定 CLASSPATH 變數：

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

J2SDK 會先搜尋啓動類別路徑，接著搜尋延伸目錄，最後才是類別路徑。以上述範例而言，J2SDK 的搜尋次序如下：

1. sun.boot.class.path 內容中的啓動類別路徑、
2. java.ext.dirs 內容中的延伸目錄、
3. 現行工作目錄、
4. root (/) 檔案系統中的 myclasses.zip 檔案、
5. root (/) 檔案系統中 Product 目錄的 classes 目錄。

在進入 Qshell 環境時，CLASSPATH 變數會設爲此環境變數。類別路徑參數可指定一連串的路徑名稱。它所採用的語法同於 CLASSPATH 環境變數。下列工具及指令皆提供類別路徑參數：

- Qshell 的 `java` 指令
- `javac` 工具
- `javah` 工具

- `javap` 工具
- `javadoc` 工具
- `rmic` 工具
- 「執行 Java (RUNJVA)」指令

如需這些指令的相關資訊，請參閱 IBM Developer Kit for Java 的指令及工具。若在這些指令或工具中使用類別路徑參數，則會忽略 CLASSPATH 環境變數。

您可以使用 java.class.path 內容來置換 CLASSPATH 環境變數。亦可使用 SystemDefault.properties 檔案來變更 java.class.path 內容及其他內容。SystemDefault.properties 檔案中的值會置換 CLASSPATH 環境變數。如需 SystemDefault.properties 檔案的相關資訊，請參閱 SystemDefault.properties 檔案。

在 J2SDK 中，-Xbootclasspath 選項亦會影響系統在尋找類別時所搜尋的目錄。使用 -Xbootclasspath/a:*path* 可將 *path* 附加至預設啓動類別路徑的後面，使用 /p:*path* 可將 *path* 置於啓動類別路徑的前面，而使用 :*path* 則會將啓動類別路徑置換成 *path*。

註: 指定 -Xbootclasspath 時請務必小心，因爲若找不到系統類別，或誤由使用者定義的類別所取代，將發生無法預期的結果。因此，應該在搜尋任何使用者指定的類別路徑之前，先搜尋系統預設類別路徑。

如需如何決定 Java 程式執行環境的相關資訊，請參閱 Java 系統內容。

如需詳細資訊，請參閱程式及 CL 指令 API 或整合檔案系統。

Java 系統內容

Java 系統內容可以決定 Java 程式的執行環境。它們類似於 i5/OS™ 的系統值或環境變數。

啓動 Java 虛擬機器 (JVM) 實例時，即會設定影響 JVM 的系統內容值。

您可以選擇採用 Java 系統內容的預設值，亦可透過下列方法指定其內容值：

- 啓動 Java 程式時，在指令行 (或「Java 原生介面 (JNI)」呼叫 API) 中新增參數
- 使用 QIBM_JAVA_PROPERTIES_FILE 工作層次的環境變數，以指向特定的內容檔。例如：

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

- 在 user.home 目錄中建立 SystemDefault.properties 檔案
- 使用 /qibm/userdata/java400/SystemDefault.properties 檔案

i5/OS 及 JVM 會依照下列優先順序，決定 Java 系統內容的值：

1. 指令行或 JNI 呼叫 API
2. QIBM_JAVA_PROPERTIES_FILE 環境變數
3. user.home SystemDefault.properties 檔案
4. /QIBM/UserData/Java400/SystemDefault.properties
5. 預設系統內容值

SystemDefault.properties 檔案

SystemDefault.properties 檔案是標準的 Java 內容檔，可用來指定 Java 環境的預設內容。

位於起始目錄中的 SystemDefault.properties 檔案，優先順序高於 /QIBM/UserData/Java400 目錄中的 SystemDefault.properties 檔案。

在 /YourUserHome/SystemDefault.properties 檔案中設定的內容，僅影響下列特定的 Java 虛擬機器：

- 在未指定不同 user.home 內容的情況下所啓動的 JVM
- 其他使用者藉由指定 user.home = /YourUserHome/ 內容來啓動的 JVM

範例：SystemDefault.properties 檔案

下列範例設定數個 Java 內容：

```
#Comments start with pound sign
     #Use J2SDK 1.4
     java.version=1.4
     #This sets a special property
     myown.propname=6
```

有關系統內容的資訊，請參閱下列各頁：

Java 系統內容

Java 系統內容的清單

Java 系統內容的清單

Java 系統內容可以決定 Java 程式的執行環境。它們類似於 i5/OS 的系統值或環境變數。

啓動 Java 虛擬機器 (JVM) 時，即會設定此 JVM 實例的系統內容。如需如何指定 Java 系統內容值的相關資訊，請參閱下列各頁：

Java 系統內容

SystemDefault.properties 檔案

如需 Java 系統內容的相關資訊，請參閱 Java Secure Socket Extension (JSSE) 系統內容。

下表列出 Java 2 Software Development Kit (J2SDK) 標準版支援版本的 Java 系統內容：

- J2SDK 1.3 版
- J2SDK 1.4 版
- J2SE 5.0 版

表格中列出每一個內容的名稱，以及可用的預設值或簡短說明。表格亦指出哪些系統內容在不同的 J2SDK 版本中會有不同的值。若列出預設值的直欄並未指出不同的 J2SDK 版本，就表示所有支援的 J2SDK 版本全部採用此預設值。

awt.toolkit	sun.awt.motif.MToolkit 只有 os400.awt.native=true 或 java.awt.headless=true 時，才會設定 awt.toolkit
file.encoding	ISO8859_1 (預設值) 將編碼字集 ID 對映至相對應的 ISO ASCII CCSID。同時將 file.encoding 值設爲代表 ISO ASCII CCSID 的 Java 值。請參閱 file.encoding 值及 iSeries CCSID，其中的表格將顯示可能的 file.encoding 值與最相近 CCSID 之間的關係。
file.encoding.pkg	sun.io
file.separator	/ (斜線)

java.awt.headless	• J2SDK v1.3：執行 J2SDK v.1.3 時，此內容不可用。 • J2SDK v1.4：false (預設值) • J2SE 5.0：false 這個內容指定 Abstract Windowing Toolkit (AWT) API 是否在遠端控制模式之下運作。 預設值 false 的意義是，唯有將 os400.awt.native 設爲 true 來啓用 AWT 時，才能夠使用完整的 AWT 功能。 此內容設爲 true 時，即可支援遠端控制 AWT 模式，亦會明確地強制 os400.awt.native 設爲 true。
java.class.path	. (句點) (預設值) 指定供 i5/OS 用來尋找類別的路徑。預設值爲使用者指定的 CLASSPATH。
java.class.version	• J2SDK v1.3：47.0 • J2SDK v1.4：48.0 • J2SE 5.0：49.0
java.compiler	jitc_de (預設值) 指定使用「即時 (JIT)」編譯器 (jitc) 或同時使 JIT 編譯器與直接處理 (jitc_de) 來編譯程式碼。
java.ext.dirs	J2SDK v1.3： • /QIBM/ProdData/Java400/jdk13/lib/ext: • /QIBM/UserData/Java400/ext J2SDK v1.4： • /QIBM/ProdData/OS400/Java400/jdk/lib/ext: • /QIBM/ProdData/Java400/jdk14/lib/ext: • /QIBM/UserData/Java400/ext (預設值) J2SE 5.0： • /QIBM/ProdData/Java400/jdk15/lib/ext: • /QIBM/UserData/Java400/ext
java.home	J2SDK v1.3：/QIBM/Prodata/Java400/jdk13 J2SDK v1.4：/QIBM/ProdData/Java400/jdk14 (預設值) J2SDK v1.5：/QIBM/ProdData/Java400/jdk15 如需詳細資料，請參閱多重 Java Development Kit (JDK) 的支援。
java.library.path	• /QSYS.LIB/QSHELL.LIB:/QSYS.LIB/QGPL.LIB: • /QSYS.LIB/QTEMP.LIB:/QSYS.LIB/QDEVELOP.LIB: • /QSYS.LIB/QBLDSYS.LIB:/QSYS.LIB/QBLDSYSR.LIB (預設值) • i5/OS 檔案庫清單

java.net.preferIPv4Stack	• true (預設值) • false (否) 在雙堆疊機器上，有系統內容可用來設定偏好的通訊協定堆疊 (IPv4 或 IPv6)，以及偏好的位址系列類型 (inet4 或 inet6)。預設的偏好堆疊是 IPv6，因爲在雙堆疊機器上，IPv6 Socket 與 IPv4 及 IPv6 兩個對等節點都可溝通。這個內容可以變更這項設定。java.net.preferIPv4Stack 爲 J2SDK v1.4 所專用。 如需相關資訊，請參閱 IPv6 通訊協定。
java.net.preferIPv6Addresses	• true • false (否) (預設值) 即使作業系統上有 IPv6 可用，預設仍將偏好 IPv4 對映位址，而非 IPv6 位址。這個內容可以控制是要使用 IPv6 (true) 還是 IPv4 (false) 位址。java.net.preferIPv4Stack 爲 J2SDK v1.4 所專用。 如需相關資訊，請參閱 IPv6 通訊協定。
java.policy	J2SDK v1.3：/QIBM/ProdData/ Java400/jdk13/lib/security/java.policy J2SDK v1.4：/QIBM/ProdData/OS400/ Java400/jdk/lib/security/java.policy (預設值) J2SE v5.0：/QIBM/ProdData/ Java400/jdk15/lib/security/java.policy
java.specification.name	• Java Platform API Specification (預設值) • Java Language Specification
java.specification.vendor	Sun Microsystems, Inc.
java.specification.version	• J2SDK v1.3：1.3 • J2SDK v1.4：1.4 (預設值) • J2SE v5.0：1.5
java.use.policy	true
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
java.version	• 1.3.1 • 1.4.2 (預設值) • 1.5.0 決定您要執行哪個版本的 J2SDK。 若安裝單一 J2SDK 版本，該版本即爲預設值。若指定未安裝的版本，會發生錯誤訊息。無法指定版本時，就會以最新的 J2SDK 版本爲預設值。 **註:** 若在 SystemDefault.properties 檔案中放置了 java.version，但卻嘗試使用「Java 原生介面 (JNI)」，就會忽略 java.version。如需詳細資訊，請參閱多重 J2SDK 的支援。
java.vm.name	Classic VM
java.vm.specification.name	Java Virtual Machine Specification
java.vm.specification.vendor	Sun Microsystems, Inc.
java.vm.specification.version	1.0
java.vm.vendor	IBM Corporation

java.vm.version	• J2SDK v1.3：1.3 • J2SDK v1.4：1.4 (預設值) • J2SE v5.0：1.5
line.separator	\n
os.arch	PowerPC®
os.name	i5/OS
os.version	V5R4M0 (預設值) 從「擷取產品資訊」應用程式設計介面 (API) 取得 i5/OS 版次。
os400.awt.native	控制是否支援 Abstract Windowing Toolkit (AWT) API。有效值爲 true 及 false。預設值爲 false，除非設定 java.awt.headless=true，這暗指 os400.awt.native 爲 true。
os400.certificateContainer	針對已啓動的 Java 程式及已指定的內容，指示 Java Secure Sockets Layer (SSL) 支援使用指定的憑證儲存區。若指定 os400.secureApplication 系統內容，則此系統內容會被忽略。例如，在整合檔案系統中輸入 `-Dos400.certificateContainer=/home/username/mykeyfile.kdb` 或任何其他金鑰檔。
os400.certificateLabel	此系統內容可與 os400.certificateContainer 系統內容一起搭配指定。這個內容可讓您在指定的儲存區內選取您希望 Secure Sockets Layer (SSL) 使用的憑證。例如，輸入 `-Dos400.certificateLabel=myCert`，其中 myCert 代表您建立或匯入憑證時，透過「數位憑證管理程式 (DCM)」指定給憑證的標籤名稱。
os400.child.stdio.convert	控制 Java 中 stdin、stdout 及 stderr 的資料轉換。在 Java 虛擬機器中，依預設會執行 ASCII 資料與「擴充二進位編碼十進位交換碼 (EBCDIC)」資料之間的資料轉換。若使用這個內容來開啓及關閉這些轉換，亦會影響此程序透過 runtime.exec() 方法所啓動的任何子項程序。請參閱預設值。
os400.class.path.security.check	20 (預設值) 有效值： • 0 無安全性檢查 • 10 相當於 RUNJVA CHKPATH(*IGNORE) • 20 相當於 RUNJVA CHKPATH(*WARN) • 30 相當於 RUNJVA CHKPATH(*SECURE)

os400.class.path.tools	0 (預設值) 有效值： • 0 java.class.path 內容中沒有 Sun 工具 • 1 將 J2SDK 特定的工具檔加在 java.class.path 內容的開頭 若爲 J2SDK v1.3，tools.jar 的路徑是 /QIBM/ProdData/Java400/jdk13/lib/ 若爲 J2SDK v1.4，tools.jar 的路徑是 /QIBM/ProdData/OS400/Java400/jdk/lib/ 若爲 J2SE v5.0，tools.jar 的路徑是 /QIBM/ProdData/Java400/jdk15/lib/
os400.create.type	• interpret (預設值) 相當於 RUNJVA OPTIMIZE(*INTERPRET)、INTERPRET(*OPTIMIZE) 或 INTERPRET(*YES) • direct 否則
os400.define.class.cache.file	預設值是 /QIBM/ProdData/Java400/QDefineClassCache.jar 指定 JAR 檔或 ZIP 檔的名稱。請參閱 Java 效能考量的「使用快取記憶體來快取使用者類別載入器」。
os400.define.class.cache.hour	• 預設值 = 768 • 最大十進位值 = 9999 指定十進位值。請參閱 Java 效能考量的「使用快取記憶體來快取使用者類別載入器」。
os400.define.class.cache.maxpgms	• 預設值 = 5000 • 最大十進位值 = 40000 指定十進位值。請參閱 Java 效能考量的「使用快取記憶體來快取使用者類別載入器」。
os400.defineClass.optLevel	0
os400.display.properties	若此值設爲 'true'，則所有「Java 虛擬機器」內容會列印至標準輸出。沒有其他可辨識的值。
os400.enbpfrcol	• 0 (預設值) 相當於 CRTJVAPGM ENBPFRCOL(*NONE) • 1 相當於 CRTJVAPGM ENBPFRCOL(*ENTRYEXIT) • 7 相當於 CRTJVAPGM ENBPFRCOL(*FULL) 若不爲零值，JIT 將產生 *JVAENTRY、*JVAEXIT、*JVAPRECALL 及 *JVAPOSTCALL 事件。

os400.exception.trace	此內容用於除錯。若指定這個內容，當 JVM 結束時，就會將最近發生的異常傳送至標準輸出。
os400.file.create.auth、os400.dir.create.auth	這些內容可指定要指派給檔案及目錄的權限。若指定時未提供內容值，或指定不受支援的值，將導致公用權限變成 *NONE。 您可以指定 os400.file.create.auth=RWX 或 os400.dir.create.auth=RWX， 其中 R=read (讀取)、W=write (寫入)、X=execute (執行)。這些權限的所有組合皆有效。
os400.file.io.mode	當您指定 TEXT 而非預設的 BINARY 時，若檔案的 CCSID 不是 file.encoding 值，則會予以轉換。
os400.gc.heap.size.init	-Xms (設定起始 GC 大小) 的替代用法。強烈建議您繼續使用 -Xms，除非由於此內容爲 i5/OS 專用而無其他選擇。此內容的主要用途，是讓您在 SystemDefault.properties 檔案中指定起始 GC 大小。 **註:** 請慎用此內容，因爲一旦指定後，它就會置換 -Xms。其值必須爲不含逗點的整數 (以 KB 爲單位)。
os400.gc.heap.size.max	-Xmx (設定 GC 大小上限) 的替代用法。強烈建議您繼續使用 -Xmx，除非由於此內容爲 i5/OS 專用而無其他選擇。此內容可讓您在 SystemDefault.properties 檔案中指定 GC 大小上限。 **註:** 請慎用此內容，因爲一旦指定後，它就會置換 -Xmx。其值必須爲不含逗點的整數 (以 KB 爲單位)。
os400.interpret	• 0 (預設值) 相當於 CRTJVAPGM INTERPRET(*NO) • 1 相當於 CRTJVAPGM INTERPRET(*YES)
os400.jit.mmi.threshold	在 i5/OS 使用 JIT 編譯器將某個方法編譯成原有的機器指令之前，使用「混合模式直譯器 (MMI)」設定此方法的執行次數。通常，您不應變更預設值 2000。 • 零值會停用 MMI，方法將在第一次呼叫時才進行編譯。 • 值若低於預設值，則不但容易拖長啓動時間，也會造成最終效能退化。 • 值若高於預設值，則起初效能會退化，直到抵達臨界值之後，才開始提升最終執行時間效能。
os400.job.file.encoding	此內容僅用於輸出。它列出 JVM 所在之 i5/OS 工作的檔案編碼方式。
os400.optimization	• 0 (預設值) 相當於 CRTJVAPGM OPTIMIZE(*INTERPRET) • 10 相當於 CRTJVAPGM OPTIMIZE(10) • 20 相當於 CRTJVAPGM OPTIMIZE(20) • 30 相當於 CRTJVAPGM OPTIMIZE(30) • 40 相當於 CRTJVAPGM OPTIMIZE(40)
os400.pool.size	定義緒本端資料堆內的每一個資料堆儲存區有多少空間可用 (以 KB 爲單位)。

os400.run.mode	• interpret 相當於 RUNJVA OPTIMIZE(*INTERPRET)、INTERPRET(*OPTIMIZE) 或 INTERPRET(*YES) • program_create_type • jitc_de (預設值) 否則
os400.run.verbose	若此值設爲 true，則詳細的類別載入情形會列印至標準輸出。沒有其他可辨識的值。在 QSHELL 中指定 -verbose，或在 CL 指令上指定 OPTION(*VERBOSE)，可以達成相同的效果，不同之處在於這個內容必須在 SystemDefault.properties 檔案中使用。
os400.runtime.exec	• EXEC (預設值) 使用 EXEC 介面，透過 runtime.exec() 來呼叫函數。 • QSHELL 使用 Qshell 直譯器，透過 runtime.exec() 來呼叫函數。 如需詳細資訊，請參閱使用 java.lang.Runtime.exec()
os400.secureApplication	將使用此系統內容 (os400.secureApplication) 時所啓動的 Java 程式，與已登記的安全應用程式名稱相關聯。可以使用「數位憑證管理程式 (DCM)」來檢視已登記的安全應用程式名稱。
os400.security.properties	可以完全控制您使用哪個 java.security 檔案。指定這個內容時，J2SDK 就不會使用任何其他的 java.security 檔案，包括 J2SDK 特定的 java.security 預設值。
os400.stderr	容許將 stderr 對映至檔案或 Socket。請參閱預設值。
os400.stdin	容許將 stdin 對映至檔案或 Socket。請參閱預設值。
os400.stdin.allowed	1 (預設值) 指定是 (1) 否 (0) 容許 stdin。若呼叫程式正在執行批次工作，則不應該容許 stdin。
os400.stdio.convert	可以控制 Java 中 stdin、stdout 及 stderr 的資料轉換。依預設，Java 虛擬機器會在 ASCII 資料與 EBCDIC 之間轉換資料。您可以使用此內容來開啓或關閉這些轉換，但會影響現行 Java 程式。請參閱預設值。
os400.stdout	容許將 stdout 對映至檔案或 Socket。請參閱預設值。
os400.xrun.option	此系統內容可讓您指定某個內容來使用 Qshell -Xrun 選項。 它可用來指定在 JVM 啓動期間執行某個代理程式。
os400.verify.checks.disable	65535 (預設值) 此系統內容值是一個字串，代表一或多個數值的總和。關於這些值的清單，請參閱 os400.verify.checks.disable 數值。
os400.vm.inputargs	此內容僅用於輸出。它會顯示 JVM 以輸入形式接收的引數。此內容對除錯 JVM 啓動時指定的內容很有用。
path.separator	: (冒號)
sun.boot.class.path	列出預設啓動類別載入器所需的全部檔案。請勿變更此值。
user.dir	使用 getcwd API 來取得現行工作目錄。

user.home	使用 Get API (getpwnam) 來擷取起始工作目錄。 您可以在 user.home 路徑中加入 SystemDefault.properties 檔案，藉此置換 /QIBM/UserData/Java400/SystemDefault.properties 中的預設內容。您可以自訂 iSeries 伺服器來指定自己的預設內容值集合。
user.language	Java 虛擬機器會使用此系統內容來讀取工作 LANGID 值，再根據此值找出相對應的語言。
user.name	Java 虛擬機器會使用此系統內容，從「授信的計算庫 (TCB)」的 Security 區段 (Security.UserName) 中擷取有效的使用者設定檔名稱。
user.region	Java 虛擬機器會使用此系統內容來讀取工作 CNTRYID 值，再根據此值決定使用者區域。
user.timezone	Universal Time Coordinate (UTC) (預設值) Java 虛擬機器會透過 QlgRetrieveLocalInformation API，使用此系統內容來取得時區名稱。JVM 會先尋找系統 QLOCALE 物件。若找不到，JVM 會查閱 QTIMZON 系統值。 若發現 QTIMZON 系統值包含無法辨識的 QTIMZON 物件，JVM 就會將 user.timezone 預設爲 UTC。 如需相關資訊，請參閱 WebSphere Software「資訊中心」中的支援 Development Kit for Java 的 user.timezone 內容值。

os400.stdio.convert 及 os400.child.stdio.convert 系統內容的值:

下表顯示 os400.stdio.convert 及 os400.child.stdio.convert 系統內容的系統值。

值	說明
N (預設值)	讀寫期間不執行 stdio 轉換。
Y	讀寫期間，所有 stdio 都會轉換成 file.encoding 值，或從 file.encoding 轉換成工作 CCSID。
1	僅 stdin 資料在讀取期間從工作 CCSID 轉換成 file.encoding。
2	僅 stdout 資料在寫入期間從 file.encoding 轉換成工作 CCSID。
3	stdin 及 stdout 轉換都執行。
4	僅 stderr 資料在寫入期間從 file.encoding 轉換成工作 CCSID。
5	stdin 及 stderr 轉換都執行。
6	stdout 及 stderr 轉換都執行。
7	執行所有 stdio 轉換。

os400.stdin、os400.stdout 及 os400.stderr 系統內容值:

下表顯示 os400.stdin、os400.stdout 及 os400.stderr 系統內容的系統值。

值	範例名稱	說明	範例
File	SomeFileName	SomeFileName 爲現行目錄的絕對路徑或相對路徑。	file:/QIBM/UserData/Java400/Output.file
Port	HostName	埠位址	port:myhost:2000
Port	TCPAddress	埠位址	port:1.1.11.111:2000

os400.verify.checks.disable 系統內容的值: os400.verify.checks.disable 系統內容值是一個字串，代表下列清單中一或多個數值的總和：

值	說明
1	**略過本端類別的存取檢查**： 亦即，對於從本端檔案系統載入的類別，您希望 Java(TM) 虛擬機器在 private 與 protected 的欄位和方法上，能夠略過存取檢查。若應用程式包含內部類別，且這些類別參照其含括類別的 private 與 protected 方法及欄位，則傳送這種應用程式時，略過存取檢查將適時發揮作用。
2	**早期載入期間忽略 NoClassDefFoundError**： 亦即，在類型強制轉型及欄位或方法存取方面，您希望 Java 虛擬機器忽略早期驗證檢查時所發生的 NoClassDefFoundError。
4	**容許略過 LocalVariableTable 檢查**：亦即，若類別的 LocalVariableTable 中發生錯誤， 會將 LocalVariableTable 視爲不存在，讓類別繼續運作下去。否則，LocaleVariableTable 中的錯誤會導致 ClassFormatError。
7	執行時間使用的值。

此值可用十進位、十六進位或八進位格式來表示。小於零的值會被忽略。例如，若要選取清單中的前兩個值，請使用此 iSeries 指令語法：

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

國際化

您可以藉由建立國際化 Java 程式，針對特定地區來自訂 Java 程式。利用時區、語言環境及字元編碼，可確保 Java 程式能夠反映正確的時間、地區及語言。

如需詳細資訊，請參閱：

時區 瞭解如何在伺服器上配置時區，讓注重時區的 Java 程式能夠使用正確的時間。

Java 語言環境 利用 Java 語言環境清單，來確保 Java 程式能夠支援某個地理區域的語言、文化資料或特定字元。

字元編碼 瞭解 Java 程式如何轉換不同格式的資料，讓您的應用程式能夠傳送及使用許多種國際字集的資訊。

範例 閱讀這些範例，以協助您使用時區、語言環境及字元編碼來建立國際化 Java 程式。

有關國際化的資訊，請參閱：

- i5/OS 全球化
- Sun Microsystems, Inc. 的國際化

時區配置

若時區會影響您的 Java 程式，則應在伺服器上配置時區，讓 Java 程式使用正確的時間。

爲了正確判斷本端時間，Java 虛擬機器 (JVM) 要求您必須設定 QUTCOFFSET i5/OS 系統值，並在現行使用者或工作的 LOCALE 使用者參數中設定當日時間資訊：

- JVM 會比較 QUTCOFFSET 值與系統的本端時間，藉以判斷正確的「世界標準時間 (UTC)」

- JVM 會透過 Java 系統內容 user.timezone，將正確的本端時間傳回系統。

註: 另外一種設定 QUTCOFFSET 及 LOCALE 的方法是使用 QTIMZON 系統值。JVM 會先尋找系統 QLOCALE 物件。若找不到，JVM 接著會查閱 QTIMZON 系統值。若 QTIMZON 系統值包含無法辨識的 QTIMZON 物件，JVM 就會將 user.timezone 預設爲 UTC。

QUTCOFFSET 及 user.timezone

QUTCOFFSET i5/OS 系統值代表您系統上的「世界標準時間 (UTC) 誤差」。QUTCOFFSET 可指定 UTC (或「格林威治標準時間」) 與目前系統時間的誤差。QUTCOFFSET 的預設值爲 0 (+00:00)。

QUTCOFFSET 值可讓 JVM 判斷正確的本端時間值。例如，若要指定美國中部標準時間 (CST)，QUTCOFFSET 值爲 -6:00。若要指定美國中部夏令時間 (CDT)，QUTCOFFSET 的值爲 -5:00。

user.timezone Java 系統內容以 UTC 時間爲預設值。除非指定不同的值，否則 JVM 一律以 UTC 時間爲目前的時間。

如需 QUTCOFFSET 及 Java 系統內容的相關資訊，請參閱下列主題：

i5/OS 系統值：QUTCOFFSET

Java 系統內容

LOCALE

使用者設定檔的 LOCALE 參數可以指定 LANG 環境變數所用的 *LOCALE 物件。請勿將 *LOCALE 物件與 Java 語言環境相混淆。

正確地設定語言環境資訊，可讓 JVM 將 user.timezone 內容設爲正確的時區。您可以設定 user.timezone 內容以置換 *LOCALE 物件所提供的預設值。

如需使用語言環境及設定 Java 系統內容的相關資訊，請參閱下列各頁：

語言環境

Java 系統內容

LC_TOD 種類可定義語言環境的日光節約時間規則及時區資訊。

註: 若要使用日光節約時間，您必須將 QUTCOFFSET 系統值的時差調整正確。

下列範例將顯示，若要爲 Java 配置正確的時區，您必須在語言環境物件中包括哪些 LC_TOD 種類資訊：

```
LC_TOD

% TZDIFF is number of minutes difference from UTC (or GMT)
tzdiff   360
% Timezone name (this is the value that you would have
% passed to the JVM as the user.timezone property.)
tname    "<C><S><T>"
% Remember to adjust the value of QUTCOFFSET when using
% daylight savings time (DST)
% Name used for DST.
dstname   "<C><D><T>"
% DST start in this part of the US is the first Sunday in
% April at 2am
dststart 4,1,1,7200
% DST End in this area of US is Last Sunday in October.
```

```
dstend   10,-1,1,7200
% shift in seconds
dstshift 3600

END LC_TOD
```

LC_TOD 種類的語言環境包含 tname 欄位，其設定值必須同於您時區的設定值。關於有效的時區字串，請參閱 java.util.TimeZone 類別的 Javadoc 參考資訊。有關如何使用語言環境的資訊，請參閱下列各頁：

使用語言環境

TimeZone Javadoc 參考資訊

Java 字元編碼

Java 程式可轉換不同格式的資料，讓您的應用程式能夠傳送及使用許多種國際字集的資訊。

Java 虛擬機器 (JVM) 的內部一律以 Unicode 處理資料。然而，進出 JVM 的所有資料，皆以符合 file.encoding 內容的格式來傳送。讀入 JVM 的資料會從 file.encoding 轉換成 Unicode，由 JVM 送出的資料則從 Unicode 轉換成 file.encoding。

Java 程式的資料檔儲存於整合檔案系統中。整合檔案系統中的檔案都會標示編碼字集 ID (CCSID)，用以識別檔案內含資料的字元編碼。如需 iSeries 伺服器上 file.encoding 與 CCSID 相互關係的說明，請參閱 File.encoding 值及 iSeries CCSID 表格。

當 Java 程式讀取資料時，資料的字元編碼應該符合 file.encoding。當 Java 程式將資料寫入檔案時，則會以符合 file.encoding 的字元編碼來寫入資料。此規則亦適用於 javac 指令所處理的 Java 原始程式碼檔案 (.java 檔)，以及利用 .net 套件並透過「傳輸控制通訊協定/網際網路通訊協定 (TCP/IP)」Socket 來收送的資料。

對 System.in、System.out 及 System.err 所讀取或寫入的資料，以及對已指定至 stdin、stdout 及 stderr 的其他來源所讀取或寫入的資料，兩者的處理方式不同。stdin、stdout 及 stderr 通常連接至 iSeries 伺服器的 EBCDIC 裝置，因此 JVM 會轉換資料，將正常的 file.encoding 字元編碼轉換成符合 iSeries 工作 CCSID 的 CCSID。當 System.in、System.out 或 System.err 重新導向至檔案或 Socket，而非導向 stdin、stdout 或 stderr 時，就不會另外執行這項資料轉換，資料仍然維持符合 file.encoding 的字元編碼。

當 Java 程式必須以 file.encoding 以外的字元編碼來讀入或寫出資料時，此程式可以使用 Java IO 類別 java.io.InputStreamReader、java.io.FileReader、java.io.OutputStreamReader 及 java.io.FileWriter。您可以在這些 Java 類別中指定優先順序優於 JVM 目前使用的預設 file.encoding 內容之 file.encoding 值。

透過 JDBC API 往返 DB2/400 資料庫的資料，將轉換成 iSeries 資料庫的 CCSID，或從這個 CCSID 進行轉換。

透過「Java 原生介面」往返其他程式的資料，則不會經過轉換。

如需國際化的相關資訊，請參閱 i5/OS 全球化。

如需詳細資訊，亦請參閱 Sun Microsystems, Inc. 的國際化。

File.encoding 值及 iSeries CCSID:

此表格顯示 file.encoding 可能值與最相符 iSeries 編碼字集 ID (CCSID) 之間的關係。

如需 file.encoding 支援的相關資訊，請參閱 Sun Microsystems 公司支援的編碼

file.encoding	CCSID	說明
ASCII	367	美國國家標準交換碼
Big5	950	8 位元 ASCII 繁體中文 BIG-5
Big5_HKSCS	950	Big5_HKSCS
Big5_Solaris	950	Big5，附加 Solaris zh_TW.BIG5 語言環境的 7 個漢字表意字元對映
CNS11643	964	繁體中文的中文國家字集
Cp037	037	IBM EBCDIC 美國、加拿大、荷蘭
Cp273	273	IBM EBCDIC 德國、奧地利
Cp277	277	IBM EBCDIC 丹麥、挪威
Cp278	278	IBM EBCDIC 芬蘭、瑞典
Cp280	280	IBM EBCDIC 義大利
Cp284	284	IBM EBCDIC 西班牙文、拉丁美洲
Cp285	285	IBM EBCDIC 英國
Cp297	297	IBM EBCDIC 法國
Cp420	420	IBM EBCDIC 阿拉伯文
Cp424	424	IBM EBCDIC 希伯來文
Cp437	437	8 位元 ASCII 美國 PC
Cp500	500	IBM EBCDIC 國際
Cp737	737	8 位元 ASCII 希臘文 MS-DOS
Cp775	775	8 位元 ASCII 波羅的海文 MS-DOS
Cp838	838	IBM EBCDIC 泰國
Cp850	850	8 位元 ASCII Latin-1 諸國
Cp852	852	8 位元 ASCII Latin-2
Cp855	855	8 位元 ASCII 斯拉夫文
Cp856	0	8 位元 ASCII 希伯來文
Cp857	857	8 位元 ASCII Latin-5
Cp860	860	8 位元 ASCII 葡萄牙
Cp861	861	8 位元 ASCII 冰島
Cp862	862	8 位元 ASCII 希伯來文
Cp863	863	8 位元 ASCII 加拿大
Cp864	864	8 位元 ASCII 阿拉伯文
Cp865	865	8 位元 ASCII 丹麥、挪威
Cp866	866	8 位元 ASCII 斯拉夫文
Cp868	868	8 位元 ASCII 烏都文
Cp869	869	8 位元 ASCII 希臘文
Cp870	870	IBM EBCDIC Latin-2
Cp871	871	IBM EBCDIC 冰島
Cp874	874	8 位元 ASCII 泰國
Cp875	875	IBM EBCDIC 希臘文
Cp918	918	IBM EBCDIC 烏都文
Cp921	921	8 位元 ASCII 波羅的海文
Cp922	922	8 位元 ASCII 愛沙尼亞

file.encoding	CCSID	說明
Cp930	930	IBM EBCDIC 日文延伸片假名
Cp933	933	IBM EBCDIC 韓文
Cp935	935	IBM EBCDIC 簡體中文
Cp937	937	IBM EBCDIC 繁體中文
Cp939	939	IBM EBCDIC 日文延伸拉丁文
Cp942	942	8 位元 ASCII 日文
Cp942C	942	Cp942 變體
Cp943	943	用於開放環境的混合日文 PC 資料
Cp943C	943	用於開放環境的混合日文 PC 資料
Cp948	948	8 位元 ASCII IBM 繁體中文
Cp949	944	8 位元 ASCII 韓文 KSC5601
Cp949C	949	Cp949 變體
Cp950	950	8 位元 ASCII 繁體中文 BIG-5
Cp964	964	EUC 繁體中文
Cp970	970	EUC 韓文
Cp1006	1006	ISO 8 位元烏都文
Cp1025	1025	IBM EBCDIC 斯拉夫語
Cp1026	1026	IBM EBCDIC 土耳其
Cp1046	1046	8 位元 ASCII 阿拉伯文
Cp1097	1097	IBM EBCDIC 波斯文
Cp1098	1098	8 位元 ASCII 波斯文
Cp1112	1112	IBM EBCDIC 波羅的海文
Cp1122	1122	IBM EBCDIC 愛沙尼亞
Cp1123	1123	IBM EBCDIC 烏克蘭
Cp1124	0	ISO 8 位元烏克蘭
Cp1140	1140	Cp037 變體，含歐元字元
Cp1141	1141	Cp273 變體，含歐元字元
Cp1142	1142	Cp277 變體，含歐元字元
Cp1143	1143	Cp278 變體，含歐元字元
Cp1144	1144	Cp280 變體，含歐元字元
Cp1145	1145	Cp284 變體，含歐元字元
Cp1146	1146	Cp285 變體，含歐元字元
Cp1147	1147	Cp297 變體，含歐元字元
Cp1148	1148	Cp500 變體，含歐元字元
Cp1149	1149	Cp871 變體，含歐元字元
Cp1250	1250	MS-Win Latin-2
Cp1251	1251	MS-Win 斯拉夫文
Cp1252	1252	MS-Win Latin-1
Cp1253	1253	MS-Win 希臘文
Cp1254	1254	MS-Win 土耳其文
Cp1255	1255	MS-Win 希伯來文

file.encoding	CCSID	說明
Cp1256	1256	MS-Win 阿拉伯文
Cp1257	1257	MS-Win 波羅的海文
Cp1258	1251	MS-Win 俄文
Cp1381	1381	8 位元 ASCII 簡體中文 GB
Cp1383	1383	EUC 簡體中文
Cp33722	33722	EUC 日文
EUC_CN	1383	簡體中文 EUC
EUC_JP	5050	日文 EUC
EUC_JP_LINUX	0	JISX 0201、0208、EUC 編碼日文
EUC_KR	970	韓文 EUC
EUC_TW	964	繁體中文 EUC
GB2312	1381	8 位元 ASCII 簡體中文 GB
GB18030	1392	簡體中文、PRC 標準
GBK	1386	新簡體中文 8 位元 ASCII 9
ISCII91	806	印度語系 Script 的 ISCII91 編碼
ISO2022CN	965	ISO 2022 CN、中文 (僅限轉換爲 Unicode)
ISO2022_CN_CNS	965	ISO 2022 CN 形式的 CNS11643、繁體中文 (僅限從 Unicode 轉換)
ISO2022_CN_GB	1383	ISO 2022 CN 形式的 GB2312、簡體中文 (僅限從 Unicode 轉換)
ISO2022CN_CNS	965	繁體中文的 7 位元 ASCII
ISO2022CN_GB	1383	簡體中文的 7 位元 ASCII
ISO2022JP	5054	日文的 7 位元 ASCII
ISO2022KR	25546	韓文的 7 位元 ASCII
ISO8859_1	819	ISO 8859-1 拉丁字母編號 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 (阿拉伯文)
ISO8859_7	813	ISO 8859-7 ISO Latin-7 (希臘文/拉丁文)
ISO8859_8	916	ISO 8859-8 ISO Latin-8 (希伯來文)
ISO8859_9	920	ISO 8859-9 ISO Latin-9 (ECMA-128、土耳其)
ISO8859_13	0	拉丁字母編號 7
ISO8859_15	923	ISO8859_15
ISO8859_15_FDIS	923	ISO 8859-15、拉丁字母編號 9
ISO-8859-15	923	ISO 8859-15、拉丁字母編號 9
JIS0201	897	日文工業標準 X0201
JIS0208	5052	日文工業標準 X0208
JIS0212	0	日文工業標準 X0212
JISAutoDetect	0	從 Shift-JIS、EUC-JP、ISO 2022 JP 偵測及轉換 (僅限轉換爲 Unicode)
Johab	0	韓文組合諺文編碼 (完整)
K018_R	878	斯拉夫文

file.encoding	CCSID	說明
KSC5601	949	8 位元 ASCII 韓文
MacArabic	1256	Macintosh 阿拉伯文
MacCentralEurope	1282	Macintosh Latin-2
MacCroatian	1284	Macintosh 克羅埃西亞文
MacCyrillic	1283	Macintosh 斯拉夫語
MacDingbat	0	Macintosh 圖形符號
MacGreek	1280	Macintosh 希臘文
MacHebrew	1255	Macintosh 希伯來文
MacIceland	1286	Macintosh 冰島
MacRoman	0	Macintosh 羅馬文
MacRomania	1285	Macintosh 羅馬尼亞
MacSymbol	0	Macintosh 符號
MacThai	0	Macintosh 泰文
MacTurkish	1281	Macintosh 土耳其文
MacUkraine	1283	Macintosh 烏克蘭
MS874	874	MS-Win 泰國
MS932	943	Windows 日文
MS936	936	Windows 簡體中文
MS949	949	Windows 韓文
MS950	950	Windows 繁體中文
MS950_HKSCS	NA	Windows 繁體中文，含中國香港特別行政區延伸字集
SJIS	932	8 位元 ASCII 日文
TIS620	874	泰文工業標準 620
US-ASCII	367	美國國家標準交換碼
UTF8	1208	UTF-8 (IBM CCSID 1208，iSeries 伺服器尚不支援)
UTF-16	1200	16 位元 UCS 轉換格式，依選用的位元組次序標記來識別位元組次序
UTF-16BE	1200	16 位元 Unicode 轉換格式，大位元組排序法
UTF-16LE	1200	16 位元 Unicode 轉換格式，小位元組排序法
UTF-8	1208	8 位元 UCS 轉換格式
Unicode	13488	UNICODE、UCS-2
UnicodeBig	13488	同於 Unicode
UnicodeBigUnmarked		無位元組次序標記的 Unicode
UnicodeLittle		小位元組排序法的 Unicode
UnicodeLittleUnmarked		無位元組次序標記的 UnicodeLittle

關於預設值，請參閱預設 file.encoding 值。

預設 file.encoding 值:

此表格顯示當 Java 虛擬機器啓動時，如何根據 iSeries 編碼字集 ID (CCSID) 來設定 file.encoding 值。

iSeries CCSID	預設 file.encoding 值	說明
37	ISO8859_1	美國、加拿大、紐西蘭及澳洲適用英文；葡萄牙及巴西適用葡萄牙文；荷蘭適用荷蘭文
256	ISO8859_1	國際 #1
273	ISO8859_1	德文/德國、德文/澳洲
277	ISO8859_1	丹麥文/丹麥、挪威文/挪威、挪威文/挪威，NY
278	ISO8859_1	芬蘭文/芬蘭
280	ISO8859_1	義大利文/義大利
284	ISO8859_1	卡達隆尼亞文/西班牙、西班牙文/西班牙
285	ISO8859_1	英文/大不列巔、英文/愛爾蘭
290	Cp943C	日文 EBCDIC 混合 (CCSID 5026) 的 SBCS 部份
297	ISO8859_1	法文/法國
420	Cp1046	阿拉伯文/埃及
423	ISO8859_7	希臘
424	ISO8859_8	希伯來文/以色列
500	ISO8859_1	德文/瑞士、法文/比利時、法文/加拿大、法文/瑞士
833	Cp970	韓文 EBCDIC 混合 (CCSID 933) 的 SBCS 部份
836	Cp1383	簡體中文 EBCDIC 混合 (CCSID 935) 的 SBCS 部份
838	TIS620	泰文
870	ISO8859_2	捷克文/捷克共和國、克羅埃西亞文/克羅埃西亞、匈牙利文/匈牙利、波蘭文/波蘭
871	ISO8859_1	冰島文/冰島
875	ISO8859_7	希臘文/希臘
880	ISO8859_5	保加利亞 (ISO 8859_5)
905	ISO8859_9	土耳其延伸字集
918	Cp868	烏都文
930	Cp943C	日文 EBCDIC 混合 (類似 CCSID 5026)
933	Cp970	韓文/韓國
935	Cp1383	簡體中文
937	Cp950	繁體中文
939	Cp943C	日文 EBCDIC 混合 (類似 CCSID 5035)
1025	ISO8859_5	白俄羅斯文/白俄羅斯、保加利亞文/保加利亞、馬其頓文/馬其頓、俄文/俄羅斯
1026	ISO8859_9	土耳其文/土耳其

iSeries CCSID	預設 file.encoding 值	說明
1027	Cp943C	日文 EBCDIC 混合 (CCSID 5035) 的 SBCS 部份
1097	Cp1098	波斯文
1112	Cp921	立陶宛文/立陶宛、拉脫維亞文/拉脫維亞、波羅的海文
1388	GBK	簡體中文 EBCDIC 混合 (含 GBK)
5026	Cp943C	日文 EBCDIC 混合 CCSID (延伸片假名)
5035	Cp943C	日文 EBCDIC 混合 CCSID (延伸拉丁文)
8612	Cp1046	阿拉伯文 (僅基本象形) (或 ASCII 420 及 8859_6)
9030	Cp874	泰文 (主電腦延伸 SBCS)
13124	GBK	簡體中文 EBCDIC 混合 (含 GBK) 的 SBCS 部份
28709	Cp948	繁體中文 EBCDIC 混合 (CCSID 937) 的 SBCS 部份

範例：建立國際化 Java 程式

如需自訂 Java 程式以提供給特定地區使用，請利用 Java 語言環境來建立國際化 Java 程式。

Java 語言環境。

建立國際化 Java 程式涉及到數個作業：

1. 隔離與語言環境有關的程式碼及資料。例如，程式中的字串、日期及數字。
2. 使用 Locale 類別來設定或取得語言環境。
3. 若不想使用預設的語言環境，則需格式化日期和數字來指定語言環境。
4. 建立資源軟體組，負責處理字串及其他與語言環境有關的資料。

請檢視下列範例，其中的方法可協助您完成建立國際化 Java 程式所需的作業：

- 範例：使用 java.util.DateFormat 類別來進行日期的國際化
- 範例：使用 java.util.NumberFormat 類別來進行數字呈現方式的國際化
- 範例：使用 java.util.ResourceBundle 類別來進行語言環境特有資料的國際化

有關國際化的資訊，請參閱：

- i5/OS 全球化
- Sun Microsystems, Inc. 的國際化

版次相容性

Java 類別檔案只要不採用 Sun 已捨棄或已變更支援的功能，即向上相容 (JDK 1.1.x -> 1.2.x -> 1.3.x -> 1.4.x -> 1.5.x)。

如需版次間可用性的相關資訊，請參閱 The Source for Java Technology java.sun.com。

使用建立 Java 程式 (CRTJVAPGM) 指令來最佳化 iSeries 伺服器上的 Java 程式時，會有「Java 程式 (JVAPGM)」附加到類別檔案。V4R4 已變更這些 JVAPGM 的內部結構。 這表示 V4R4 以前建立的 JVAPGM，無法在 V4R4 及更高版次上使用。必須以與原先相同的最佳化等級來重建 JVAPGM，或讓系統自動建立 JVAPGM。 然而，建議手動執行 CRTJVAPGM，尤其是對於 JAR 或 ZIP 檔。這樣可以達成最完美的最佳化及產生最小的程式。

爲了獲得最佳化等級 40 的最大效能，建議在每一次 i5/OS 版次或 JDK 版本變更時，都執行 CRTJVAPGM。尤其當 CRTJVAPGM 採用 JDKVER 機能時，更應該這樣做，因爲可以將 Sun JDK 的方法列入 JVAPGM 之內。這在效能方面的表現非常卓越。然而，若後續 JDK 版次出現變動而造成這些行內程式碼失效，程式實際執行起來可能很慢，還不如採用較低的最佳化等級。這是因爲必須執行特殊情況的程式碼，才能正常運作。

如需效能的詳細資訊，請參閱 Java Runtime 效能。

資料庫存取與 IBM Developer Kit for Java

IBM Developer Kit for Java 提供三種方式讓 Java 程式存取資料庫檔案。

- JDBC 驅動程式說明 IBM Developer Kit for Java JDBC 驅動程式如何讓 Java 程式存取資料庫檔案。
- SQLJ 支援說明 IBM Developer Kit for Java 如何讓您使用 Java 應用程式中內含的 SQL 陳述式。
- Java SQL 常式說明如何使用 Java 儲存程序及 Java 使用者定義的函數來存取 Java 程式。

使用 IBM Developer Kit for Java JDBC 驅動程式來存取 iSeries 資料庫

IBM Developer Kit for Java JDBC 驅動程式 (亦稱爲「原有的」驅動程式) 可透過程式設計方法來存取 iSeries 資料庫檔案。運用 Java Database Connectivity (JDBC) API，以 Java 語言編寫的應用程式可透過內含的「結構化查詢語言 (SQL)」來存取 JDBC 資料庫功能、執行 SQL 陳述式、擷取結果，以及將變更送回資料庫。在分散式異質環境中，亦可利用 JDBC API 與多重資料來源進行互動。

JDBC API 所根據的「SQL99 指令語言介面 (CLI)」是 ODBC 的基礎。JDBC 提供自然而簡單的方法，可以將 Java 程式設計語言對映至 SQL 標準所定義的抽象規格及概念。

若要使用 JDBC 驅動程式，請參閱：

JDBC 入門 您可以遵循指導教學來編寫 JDBC 程式，再於 iSeries 伺服器上執行。

連線 應用程式一次可以有多個連線。您可以使用 Connection 物件，代表在 JDBC 中對於某個資料來源的連線。必須透過 Connection 物件，才能建立 Statement 物件，以便對資料庫處理 SQL 陳述式。

JVM 內容 連線內容無法設定原有 JDBC 驅動程式所用的一些設定值。在執行原有的 JDBC 驅動程式的 JVM 中，必須設定這些設定值。

DatabaseMetaData 應用程式伺服器及工具會利用 DatabaseMetaData 介面，決定如何與給定的資料來源互動。應用程式亦可能使用 DatabaseMetaData 方法來取得特定資料來源的相關資訊。

異常 Java 語言使用異常來提供程式的錯誤處理能力。異常就是程式執行時中斷正常指令流程的事件。

異動 異動就是工作的邏輯單元。異動可提供資料整合性、正確的應用語意，以及在並行存取資料時的一致性。所有符合 JDBC 標準的驅動程式都必須支援異動。

陳述式類型 Statement 介面及其 PreparedStatement 與 CallableStatement 子類別，可用來處理向資料庫提出的 SQL 指令。SQL 陳述式會導致 ResultSet 物件產生。

ResultSet ResultSet 介面可存取執行查詢所產生的結果。ResultSet 的資料可視爲一個表格，內含一定的欄位數及列數。依預設，表格列是按順序來擷取。在一列中，可依任何次序來存取直欄值。

JDBC 物件儲存區作業 JDBC 中有許多物件的建立代價很高，例如 Connection、Statement 及 ResultSet 物件，但可利用 JDBC 物件儲存區作業來明顯提高效能。利用物件儲存區作業，就可重覆使用這些物件，不必每次需要時就重新建立一次。

批次更新 批次更新支援可讓資料庫的許多更新，在使用者程式與資料庫之間視爲單一異動來傳遞。當必須一次完成許多更新時，批次更新可明顯改善效能。

進階資料類型 在 iSeries 資料庫中，有一些新的資料類型，稱爲 SQL3 資料類型。SQL3 資料類型提供極大的彈性。很適合用來儲存序列化 Java 物件、「可延伸標記語言 (XML)」文件，以及多媒體資料，例如歌曲、產品圖片、員工照片及電影片段等。SQL3 資料類型包括：

- 特殊類型
- 大型物件，例如「二進位大型物件」、「字元大型物件」及「雙位元組字元大型物件」
- Datalink

RowSet RowSet 規格的設計與其說是實際的實作方式，其實更接近一種組織架構。Rowset 介面定義一組核心功能，所有 Rowset 都能夠使用。

分散式異動 Java Transaction API (JTA) 支援複雜的異動。亦支援將異動與 Connection 物件分離。JTA 與 JDBC 互相合作，將異動與 Connection 物件分離後，即可讓單一連線同時處理多個異動。相反地，它也可讓您使用多個連線來處理單一異動。

效能要訣 您可以利用這些效能要訣讓 JDBC 應用程式獲得最高效能。

如需 JDBC 的相關資訊，請參閱 Sun Microsystems 公司的 JDBC 文件。

如需 iSeries 原有 JDBC 驅動程式的相關資訊，請參閱 iSeries native JDBC Driver FAQS。

JDBC 入門

Developer Kit for Java 隨附的 Java Database Connectivity (JDBC) 驅動程式稱爲 Developer Kit for Java JDBC 驅動程式。此驅動程式亦通稱爲「原有的 JDBC 驅動程式」。

若要選擇最符合需求的 JDBC 驅動程式，請考慮下列建議：

- 直接在資料庫所在的伺服器上執行的程式，應該使用原有的 JDBC 驅動程式，以獲取最大效能。其中包括大部分的 Servlet 及 JavaServer Pages (JSP) 解決方案，以及編寫爲在 iSeries 伺服器本端執行的應用程式。
- 必須連接遠端 iSeries 伺服器的程式，應該使用 IBM Toolbox for Java JDBC 驅動程式。IBM Toolbox for Java JDBC 驅動程式是 JDBC 的健全實作方式，由 IBM Toolbox for Java 直接提供。由於是 Pure Java，用戶端是否設定 IBM Toolbox for Java JDBC 驅動程式並不重要，但伺服器需要少許的設定。
- 在 iSeries 伺服器上執行且需要連接遠端非 iSeries 資料庫的程式，應該使用原有的 JDBC 驅動程式，並且設定要連接至該遠端伺服器的「分散式關聯資料庫架構™ (DRDA®)」連線。

若要開始使用 JDBC，請參閱：

JDBC 驅動程式的類型 本主題定義 JDBC 驅動程式的類型。其中定義驅動程式類型，將資料庫連接技術予以分類。

基本要求 本主題指出存取下列項目的基本要求：

- 核心 JDBC

- JDBC 2.0 選用套件
- Java Transaction API (JTA)

JDBC 指導教學 此爲利用原有 JDBC 驅動程式編寫 JDBC 程式並於 iSeries 伺服器上執行的一個重要起步。

JDBC 驅動程式的類型:

本主題定義 Java Database Connectivity (JDBC) 驅動程式類型。其中使用驅動程式類型，將資料庫連接技術予以分類。JDBC 驅動程式廠商就使用這些類型來說明產品的運作方式。某些 JDBC 驅動程式類型比其他類型更適合於某些應用程式。

類型 1

「類型 1」驅動程式屬於「橋接」驅動程式。它們採用另一項技術與資料庫進行通訊，例如「開放式資料庫連線功能 (ODBC)」。好處在於許多「關聯式資料庫管理系統 (RDBMS)」平台都已經有 ODBC 驅動程式。可從 JDBC 驅動程式透過「Java 原生介面 (JNI)」來呼叫 ODBC 函數。

但需要安裝及配置橋接驅動程式，JDBC 才能使用「類型 1」驅動程式。對正式運作的應用程式而言，這是一項很嚴重的缺點。Applet 中無法使用「類型 1」驅動程式，因爲 Applet 無法載入原有的程式碼。

類型 2

「類型 2」驅動程式採用原有的 API 與資料庫系統進行通訊。使用 Java 原生方法來呼叫執行資料庫作業的 API 功能。「類型 2」驅動程式通常比「類型 1」驅動程式更快。

需安裝及配置原有的二位元碼，「類型 2」驅動程式才能運作。「類型 2」驅動程式亦使用 JNI。Applet 中無法使用「類型 2」驅動程式，因爲 Applet 無法載入原有的程式碼。「類型 2」JDBC 驅動程式可能需要安裝一些「資料庫管理系統 (DBMS)」網路作業軟體。

Developer Kit for Java JDBC 驅動程式即爲一種「類型 2」JDBC 驅動程式。

類型 3

這種驅動程式使用網路通訊協定及中介軟體與伺服器進行通訊。然後，伺服器再將通訊協定轉換成 DBMS 專用的 DBMS 函數呼叫。

「類型 3」JDBC 驅動程式是最有彈性的 JDBC 解決方案，因爲用戶端不需要任何原有的二位元碼。「類型 3」驅動程式不需任何用戶端安裝動作。

類型 4

「類型 4」驅動程式採用 Java 來實作 DBMS 供應商網路通訊協定。因爲通訊協定通常爲專利產品，所以 DBMS 供應商通常就是提供「類型 4」JDBC 驅動程式的唯一公司。

「類型 4」驅動程式全部都是 Java 驅動程式。這表示不需要用戶端安裝或配置。然而，若底層通訊協定無法妥善處理安全性及網路連線功能，則「類型 4」驅動程式可能不適用於某些應用程式。

IBM Toolbox for Java JDBC 驅動程式是屬於「類型 4」JDBC 驅動程式，這表示 API 是 Pure Java 的網路通訊協定驅動程式。

JDBC 基本要求:

在編寫及部署 JDBC 應用程式之前，可能需要在類別路徑中納入特定的 Jar 檔。

核心 JDBC

在核心 Java Database Connectivity (JDBC) 存取本端資料庫方面，沒有任何基本要求。所有支援皆已內建、預先安裝及完成配置。

JDBC 2.0 選用套件

如需使用 JDBC 2.0 選用套件的類別，則必須在類別路徑中包含 jdbc2_0-stdext.jar 檔案。此 Java ARchive (JAR) 檔包含的介面，即爲編寫應用程式來使用 JDBC 2.0 選用套件所需的所有標準介面。若要在延伸類別路徑中加入此 JAR 檔，請建立一個從 UserData 延伸套件目錄至 JAR 檔位置的符號鏈結。這只需要執行一次；之後，您的應用程式在執行時，JDBC 2.0 選項套件中的 JAR 檔一定可供使用。請使用下列指令，將選用套件新增至延伸類別路徑中：

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

附註：這項基本要求僅適用於 J2SDK 1.3。由於 J2SDK 1.4 是第一個加入 JDBC 3.0 支援的版次，所以整個 JDBC (亦即，核心 JDBC 及選用套件) 都會移至您的程式一定找得到的基礎 J2SDK Runtime JAR 檔。

Java Transaction API

如需在應用程式中使用 Java Transaction API (JTA)，則必須在類別路徑中包含 jta-spec1_0_1.jar 檔案。此 JAR 檔包含編寫應用程式來使用 JTA 時所需的所有標準介面。若要在延伸類別路徑中加入此 JAR 檔，請建立一個從 UserData 延伸套件目錄至 JAR 檔位置的符號鏈結。此動作只需執行一次，完成之後，您的應用程式在執行時，JTA JAR 檔就一定可供使用。請使用下列指令，將 JTA 新增至延伸類別路徑中：

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

符合 JDBC 標準

原有的 JDBC 驅動程式符合所有相關的 JDBC 規格的標準。JDBC 驅動程式的符合程度，並非根據 i5/OS 版次，而是依您使用的 JDK 版次而定。以下列出原有的 JDBC 驅動程式與各種 JDK 的符合程度：

J2SDK 版次	JDBC 驅動程式的符合程度
JDK 1.1	此 JDK 符合於 JDBC 1.0 標準。
JDK 1.2	此 JDK 符合於 JDBC 2.0 標準，且支援 JDBC 2.1 選用套件。
JDK 1.3	此 JDK 符合於 JDBC 2.0 標準，且支援 JDBC 2.1 選用套件 (JDK 1.3 沒有 JDBC 相關的變動)。
JDK 1.4 及後續版本	這些 JDK 版本符合於 JDBC 3.0 標準，但 JDBC 選用套件已不存在 (核心 JDK 現在已內建其支援)。

JDBC 指導教學:

以下爲利用原有 JDBC 驅動程式編寫 Java Database Connectivity (JDBC) 程式並於 iSeries 伺服器上執行的指導教學。用意在於說明程式執行 JDBC 所需的基本步驟。

第 34 頁的『範例：JDBC』會先建立表格，並且輸入一些資料。程式會處理查詢，從資料庫取出資料，然後顯示在螢幕上。

執行範例程式

若要執行範例程式，請執行下列步驟：

1. 將程式複製到您的工作站。
   a. 複製第 34 頁的『範例：JDBC』，然後貼在工作站的檔案內。
   b. 以 public class 提供的相同名稱來儲存檔案，副檔名爲 .java。以此例而言，您必須在本端工作站將檔案命名爲 BasicJDBC.java。
2. 將檔案從工作站傳送至 iSeries 伺服器。在指令提示下，請輸入下列指令：

```
ftp <iSeries 伺服器名稱>
     <輸入使用者 ID>
     <輸入密碼>
cd /home/cujo
     put BasicJDBC.java
     quit
```

   爲了讓這些指令發揮作用，您必須有一個目錄來存放檔案。在範例中，/home/cujo 就是位置，但可改成您喜歡的位置。

   註: 根據您的伺服器設定方式，上述 FTP 指令可能有所不同，但應該類似。不論檔案傳送至 iSeries 伺服器的方式爲何，只要是傳送至整合檔案系統內就可以。VisualAge® for Java 等工具可爲您自動完成這項程序。
3. 請務必在類別路徑中指向您存放檔案的目錄，讓您執行的 Java 指令能夠找到此檔案。您可以從 CL 指令行使用 WRKENVVAR，即可明瞭您的使用者設定檔所設定的環境變數。
   - 若看到 CLASSPATH 這個環境變數，請確定其中的目錄字串含有您放置 .java 檔的位置，不然就得自行加入。
   - 若看不到 CLASSPATH 環境變數，則必須新增。請執行下列指令來新增此環境變數：

```
ADDENVVAR ENVVAR(CLASSPATH)
     VALUE('/home/cujo:/QIBM/ProdData/Java400/jdk13/lib/tools.jar')
```

   註: 若要透過 CL 指令來編譯 Java 程式碼，您必須加上 tools.jar 檔案。此 JAR 檔包含 javac 指令。
4. 將 Java 檔案編譯成類別檔案。從 CL 指令行輸入下列指令：

```
java class(com.sun.tools.javac.Main) prop(BasicJDBC)
java BasicJDBC
```

   亦可從 QSH 編譯 Java 檔案：

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

   QSH 會自動確實找到 tools.jar 檔案。因此，不必將它加入類別路徑中。類別路徑也包含現行目錄。就算發出切換目錄 (cd) 指令，一樣可以找到 BasicJDBC.java 檔案。

   註: 您也可以在工作站上編譯檔案，然後使用 FTP 以二進位模式將類別檔案傳送至 iSeries 伺服器。以下範例顯示 Java 在所有平台上順利執行的能力。

   從 CL 指令行或 QSH 中，使用下列指令來執行程式：

```
java BasicJDBC
```

   輸出如下：

```
----------------------
| 1 | Frank Johnson  |
|                    |
| 2 | Neil Schwartz  |
|                    |
```

```
| 3 | Ben Rodman     |
|                    |
| 4 | Dan Gloore     |
----------------------
There were 4 rows returned.
Output is complete.
Java program completed.
```

如需 Java 及 JDBC 的相關資訊，請參考下列資源：

- IBM Toolbox for Java JDBC 驅動程式外部網站
- Sun 的 JDBC 網頁
- iSeries 及 iSeries 使用者的 Java/JDBC 討論區
- IBM JDBC 電子郵件位址

範例：JDBC:

以下爲如何使用 BasicJDBC 程式的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////////////////////////////////////////
//
// BasicJDBC example.  This program uses the native JDBC driver for the
// Developer Kit for Java to build a simple table and process a query
// that displays the data in that table.
//
// Command syntax:
//    BasicJDBC
//
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////////////////////////////////////////////

// Include any Java classes that are to be used. In this application,
// many classes from the java.sql package are used and the
// java.util.Properties class is also used as part of obtaining
// a connection to the database.
import java.sql.*;
import java.util.Properties;

// Create a public class to encapsulate the program.
```

```
public class BasicJDBC {

    // The connection is a private variable of the object.
    private Connection connection = null;

    // Any class that is to be an 'entry point' for running
    // a program must have a main method. The main method
    // is where processing begins when the program is called.
    public static void main(java.lang.String[] args) {

        // Create an object of type BasicJDBC. This
        // is fundamental to object-oriented programming. Once
        // an object is created, call various methods on
        // that object to accomplish work.
        // In this case, calling the constructor for the object
        // creates a database connection that the other
        // methods use to do work against the database.
        BasicJDBC test = new BasicJDBC();

        // Call the rebuildTable method. This method ensures that
        // the table used in this program exists and looks
        // correct. The return value is a boolean for
        // whether or not rebuilding the table completed
        // successfully. If it did no, display a message
        // and exit the program.
        if (!test.rebuildTable()) {
            System.out.println("Failure occurred while setting up " +
                               " for running the test.");
            System.out.println("Test will not continue.");
            System.exit(0);
        }

        // The run query method is called next. This method
        // processes an SQL select statement against the table that
        // was created in the rebuildTable method. The output of
        // that query is output to standard out for you to view.
        test.runQuery();

        // Finally, the cleanup method is called. This method
        // ensures that the database connection that the object has
        // been hanging on to is closed.
        test.cleanup();
    }


    /**
    This is the constructor for the basic JDBC test. It creates a database
    connection that is stored in an instance variable to be used in later
    method calls.
    **/
    public BasicJDBC() {

        // One way to create a database connection is to pass a URL
        // and a java Properties object to the DriverManager. The following
        // code constructs a Properties object that has your user ID and
        // password. These pieces of information are used for connecting
        // to the database.
        Properties properties = new Properties ();
        properties.put("user", "cujo");
        properties.put("user", "newtiger");

        // Use a try/catch block to catch all exceptions that can come out of the
        // following code.
        try {
            // The DriverManager must be aware that there is a JDBC driver available
            // to handle a user connection request. The following line causes the
            // native JDBC driver to be loaded and registered with the DriverManager.
```

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

        // Create the database Connection object that this program uses in all
        // the other method calls that are made. The following code specifies
        // that a connection is to be established to the local database and that
        // that connection should conform to the properties that were set up
        // previously (that is, it should use the user ID and password specified).
        connection = DriverManager.getConnection("jdbc:db2:*local", properties);

    } catch (Exception e) {
        // If any of the lines in the try/catch block fail, control transfers to
        // the following line of code. A robust application tries to handle the
        // problem or provide more details to you. In this program, the error
        // message from the exception is displayed and the application allows
        // the program to return.
        System.out.println("Caught exception: " + e.getMessage());
    }
}


/**
Ensures that the qgpl.basicjdbc table looks you want it to at the start of
the test.

@returns boolean    Returns true if the table was rebuild successfully;
                    returns false if any failure occurred.
**/
public boolean rebuildTable() {
    // Wrap all the functionality in a try/catch block so an attempt is
    // made to handle any errors that may happen within this method.
    try {

        // Statement objects are used to process SQL statements against the
        // database. The Connection object is used to create a Statement
        // object.
        Statement s = connection.createStatement();

        try {
            // Build the test table from scratch. Process an update statement
            // that attempts to delete the table if it currently exists.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // Do not perform anything if an exception occurred. Assume
            // that the problem is that the table that was dropped does not
            // exist and that it can be created next.
        }

        // Use the statement object to create our table.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Use the statement object to populate our table with some data.
        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')");

        // Close the SQL statement to tell the database that it is no longer
        // needed.
        s.close();

        // If the entire method processed successfully, return true. At this point,
        // the table has been created or refreshed correctly.
        return true;

    } catch (SQLException sqle) {
        // If any of our SQL statements failed (other than the drop of the table
        // that was handled in the inner try/catch block), the error message is
```

```
        // displayed and false is returned to the caller, indicating that the table
        // may not be complete.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}



/**
Runs a query against the demonstration table and the results are displayed to
standard out.
**/
public void runQuery() {
    // Wrap all the functionality in a try/catch block so an attempts is
    // made to handle any errors that might happen within this
    // method.
    try {
        // Create a Statement object.
        Statement s = connection.createStatement();

        // Use the statement object to run an SQL query. Queries return
        // ResultSet objects that are used to look at the data the query
        // provides.
        ResultSet rs = s.executeQuery("select * from qgpl.basicjdbc");

        // Display the top of our 'table' and initialize the counter for the
        // number of rows returned.
        System.out.println("--------------------");
        int i = 0;

        // The ResultSet next method is used to process the rows of a
        // ResultSet. The next method must be called once before the
        // first data is available for viewing. As long as next returns
        // true, there is another row of data that can be used.
        while (rs.next()) {

            // Obtain both columns in the table for each row and write a row to
            // our on-screen table with the data. Then, increment the count
            // of rows that have been processed.
            System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
            i++;
        }

        // Place a border at the bottom on the table and display the number of rows
        // as output.
        System.out.println("--------------------");
        System.out.println("There were " + i + " rows returned.");
        System.out.println("Output is complete.");

    } catch (SQLException e) {
        // Display more information about any SQL exceptions that are
        // generated as output.
        System.out.println("SQLException exception: ");
        System.out.println("Message:....." + e.getMessage());
        System.out.println("SQLState:...." + e.getSQLState());
        System.out.println("Vendor Code:." + e.getErrorCode());
        e.printStackTrace();
    }
}


/**
The following method ensures that any JDBC resources that are still
allocated are freed.
**/
public void cleanup() {
```

```
        try {
            if (connection != null)
                connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

使用 JNDI 的範例:

DataSource 與「Java 命名和目錄介面 (JNDI)」可以互相搭配使用。JNDI 是目錄服務的 Java 摘要層，就好比 Java Database Connectivity (JDBC) 是資料庫的摘要層一樣。

JNDI 最常搭配 Lightweight Directory Access Protocol (LDAP) 一起使用，但亦可用於 CORBA Object Services (COS)、「Java 遠端方法呼叫 (RMI)」登錄或基礎的檔案系統。透過各種目錄服務提供者將一般的 JNDI 要求轉換成特定的目錄服務要求，才得以達成各種用法。Java 2 SDK 1.3 版包含三種服務提供者：LDAP 服務提供者、COS 命名服務提供者及 RMI 登錄服務提供者。

註: 請注意，使用 RMI 可能讓工作更複雜。 在選擇 RMI 作爲解決方案之前，請確定您已瞭解後果如何。請利用 Java 遠端方法呼叫 (RMI) 開始評定 RMI。

DataSource 範例設計爲使用 JNDI 檔案系統服務提供者。若要執行提供的範例，必須先備妥 JNDI 服務提供者。

請遵循下列指示來設定檔案系統服務提供者的環境：

1. 從 Sun Microsystems JNDI 網站下載檔案系統 JNDI 支援。
2. 將 fscontext.jar 與 providerutil.jar 傳送至您的系統 (使用 FTP 或其他機制)，並且放入 /QIBM/UserData/Java400/ext 中。此爲延伸套件目錄，在執行您的應用程式時，就會自動搜尋您放在此處的 JAR 檔 (亦即，不需在類別路徑中加入)。

當您爲 JNDI 的服務提供者備妥支援時，就必須開始設定應用程式的環境定義資訊。在 SystemDefault.properties 檔案中放入必要的資訊，即可完成這項動作。系統上有幾個位置可供您指定預設內容，但最好的方法就是在您的起始目錄中 (亦即 /home/)，建立名爲 SystemDefault.properties 的文字檔。

若要建立檔案，請採用以下幾行，或直接加入現有的檔案中：

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

這幾行指定檔案系統服務提供者將處理 JNDI 要求，且 /DataSources/jdbc 代表使用 JNDI 的作業的根目錄。您可以變更此位置，條件是您所指定的目錄必須存在。您指定的位置，就是連結及部署 DataSource 範例的位置。

連線

在 Java Database Connectivity (JDBC) 中，Connection 物件代表某個資料來源的連線。必須透過 Connection 物件，才能建立 Statement 物件，以便對資料庫處理 SQL 陳述式。應用程式一次可以有多個連線。這些 Connection 物件可以全部連接至相同的資料庫，也可以連接至不同的資料庫。

JDBC 中有兩種方法可以取得連線：

- 透過 DriverManager 類別。
- 使用 DataSource。

使用 DataSource 來取得連線比較理想，因爲可以提高應用程式的可攜性及維護性。亦可讓應用程式更直接地使用連線和陳述式儲存區作業，以及分散式異動。

關於如何取得連線的詳細資料，請參閱下列各節：

DriverManager DriverManager 是靜態類別，負責管理一組可用的 JDBC 驅動程式供應用程式使用。

連線內容 此表格列出有效的 JDBC 驅動程式連線內容、內容值及其說明。

搭配使用 DataSource 與 UDBDataSource 您可以對 DataSource 部署 UDBDataSource 類別，方法是先設定 DataSource 的特定內容，再透過「Java 命名和目錄介面 (JNDI)」連結至一些目錄服務。

DataSource 內容 此表格列出有效的 DataSource 內容、內容值及其說明。

其他 DataSource 實作方式 原有的 JDBC 驅動程式所提供的 DataSource 介面，還有其他實作方式。這些實作方式，只是正式採用 UDBDataSource 及相關的函數之前的過渡解決方案。

一旦取得連線之後，即可用來完成下列 JDBC 作業：

- 建立各種 Statement 物件，與資料庫相互作用。
- 控制資料庫的異動。
- 針對資料庫來擷取 meta 資料。

DriverManager:

DriverManager 是 Java 2 Software Development Kit (J2SDK) 的靜態類別。DriverManager 管理可供應用程式使用的 Java Database Connectivity (JDBC) 驅動程式集。

必要的話，應用程式可同時使用多個 JDBC 驅動程式。每一個應用程式都可利用「全球資源定位器 (URL)」來指定 JDBC 驅動程式。只要將特定 JDBC 驅動程式的 URL 傳給 DriverManager，應用程式即可通知 DriverManager，應該將何種 JDBC 連線傳回給應用程式。

但在此之前，DriverManager 必須先得知可用的 JDBC 驅動程式，才能夠交出連線。透過呼叫 Class.forName 方法，即可根據傳入此方法的字串名稱，於執行中的 Java 虛擬機器 (JVM) 內載入類別。以下範例說明如何使用 class.forName 方法來載入原有 JDBC 驅動程式：

範例：載入原有的 JDBC 驅動程式

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Load the native JDBC driver into the DriverManager to make it
// available for getConnection requests.

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

根據設計，JDBC 驅動程式會在載入驅動程式實作類別時，自動將本身的資訊告知 DriverManager。處理完前述的程式碼行之後，原有的 JDBC 驅動程式即可供 DriverManager 使用。下列程式碼行會利用原有的 JDBC URL 來要求 Connection 物件：

範例：要求 Connection 物件

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Get a connection that uses the native JDBC driver.

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

JDBC URL 最簡單的形式，就是三個分別以冒號隔開的值。第一個值代表通訊協定，就 JDBC URL 而言，一律爲 jdbc。第二個值是子通訊協定，以 db2 或 db2iSeries 來指定原有的 JDBC 驅動程式。第三個值是系統名稱，用來建立特定系統的連線。還有兩個特殊值可用來連接本端資料庫。分別是 *LOCAL 與 localhost (皆不區分大小寫)。當然，亦可提供特定的系統名稱，如下所示：

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

這樣會建立 rchasmop 系統的連線。若您嘗試連接的系統是遠端系統 (例如，透過「分散式關聯資料庫架構 (DRDA)」)，則必須使用關聯式資料庫目錄的系統名稱。

註: 若未指定使用者 ID 及密碼，則直接使用目前登入的使用者 ID 及密碼來建立資料庫連線。

註: IBM DB2® JDBC Universal 驅動程式亦使用 db2 子通訊協定。爲了確保原有的 JDBC 驅動程式能夠處理 URL，應用程式需使用 jdbc:db2iSeries:xxxx URL，而非 jdbc:db2:xxxx URL。若應用程式不希望原有的驅動程式接受含有 db2 子通訊協定的 URL，則應該載入類別 com.ibm.db2.jdbc.app.DB2iSeriesDriver，而非 com.ibm.db2.jdbc.app.DB2Driver。載入此類別之後，原有的驅動程式就不再處理含有 db2 子通訊協定的 URL。

內容

DriverManager.getConnection 方法接受前述的單一字串 URL，它是 DriverManager 取得 Connection 物件的唯一方法。DriverManager.getConnection 方法還有另一個版本，可以接受使用者 ID 及密碼。以下爲此版本的範例：

範例：採用使用者 ID 及密碼的 DriverManager.getConnection 方法

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Get a connection that uses the native JDBC driver.

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

就此行程式碼而言，不論是誰正在執行應用程式，皆會試圖以使用者 cujo 與密碼 newtiger 來連接本端資料庫。DriverManager.getConnection 方法也有另一個版本，可以接受 java.util.Properties 物件，允許進一步自訂。範例如下：

範例：使用 java.util.Properties 物件的 DriverManager.getConnection 方法

```
// Get a connection that uses the native JDBC driver.

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

這段程式碼的功能，相當於前一段傳入使用者 ID 及密碼作爲參數的程式碼。

如需原有的 JDBC 驅動程式連線內容的完整清單，請參閱 Connection 內容。

URL 內容

另外一種指定內容的方式，就是在 URL 物件本身放入內容清單。清單中的每一個內容須以分號隔開，且清單的格式必須爲內容名稱=內容值。這只是簡便的方法，對於處理方式不會有太大的改變，如下列範例所示：

範例：指定 URL 內容

```
// Get a connection that uses the native JDBC driver.

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

這行程式碼的功能，同樣也相當於前述的範例。

若同時在 properties 物件與 URL 物件中指定某個內容值，URL 的值將優先於 properties 物件的值。範例如下：

範例：URL 內容

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Get a connection that uses the native JDBC driver.
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);
```

此範例會採用 URL 字串中的使用者 ID 及密碼，而非 Properties 物件的值。其結果在功能上也相當於前述程式碼。

如需詳細資訊，請參閱下列範例：

- 同時使用原有的 JDBC 與 IBM Toolbox for Java JDBC
- Access 內容
- 無效的使用者 ID 和密碼

範例：同時使用原有的 JDBC 與 IBM Toolbox for Java JDBC:

以下範例說明如何在程式中使用原有的 JDBC 連線與 IBM Toolbox for Java JDBC 連線。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////////////////////////////////////////
//
// GetConnections example.
//
// This program demonstrates being able to use both JDBC drivers at
// once in a program. Two Connection objects are created in this
// program.
One is a native JDBC connection and one is a IBM Toolbox for Java
// JDBC connection.
//
// This technique is convenient because it allows you to use different
// JDBC drivers for different tasks concurrently. For example, the
// IBM Toolbox for Java JDBC driver is ideal for connecting to remote iSeries servers
// and the native JDBC driver is faster for local connections.
// You can use the strengths of each driver concurrently in your
// application by writing code similar to this example.
//
//////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
```

```
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// 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)
   {
       // Verify input.
       if (args.length != 2) {
          System.out.println("Usage (CL command line):  java GetConnections PARM(<user> <password>)");
          System.out.println(" where <user> is a valid iSeries user ID");
          System.out.println("   and <password> is the password for that user ID");
          System.exit(0);
       }

       // Register both drivers.
      try {
         Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
         Class.forName("com.ibm.as400.access.AS400JDBCDriver");
      } catch (ClassNotFoundException cnf) {
         System.out.println("ERROR: One of the JDBC drivers did not load.");
         System.exit(0);
      }

      try {
         // Obtain a connection with each driver.
         Connection conn1 = DriverManager.getConnection("jdbc:db2://localhost", args[0], args[1]);
         Connection conn2 = DriverManager.getConnection("jdbc:as400://localhost", args[0], args[1]);

         // Verify that they are different.
         if (conn1 instanceof com.ibm.db2.jdbc.app.DB2Connection)
            System.out.println("conn1 is running under the native JDBC driver.");
         else
            System.out.println("There is something wrong with conn1.");

         if (conn2 instanceof com.ibm.as400.access.AS400JDBCConnection)
            System.out.println("conn2 is running under the IBM Toolbox for Java JDBC driver.");
         else
            System.out.println("There is something wrong with conn2.");

         conn1.close();
         conn2.close();
      } catch (SQLException e) {
         System.out.println("ERROR: " + e.getMessage());
      }
   }
}
```

鏈結集合

程式碼範例免責聲明

範例：Access 內容:

以下爲如何使用 Access 內容的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Note: This program assumes directory cujosql exists.
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();
    }


/**
Set up the DataSource used in the testing.
**/
    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) { // Ignore it - it doesn't exist
        }

        try {
            String sql = "CREATE PROCEDURE CUJOSQL.TEMP "
                  + " LANGUAGE SQL SPECIFIC CUJOSQL.TEMP "
                  + " MYPROC: BEGIN"
                  + "   RETURN 11;"
                  + " END MYPROC";
            s.executeUpdate(sql);
        } catch (SQLException e) {
            // Ignore it - it exists.
        }
        s.executeUpdate("create table cujosql.temp (col1 char(10))");
        s.executeUpdate("insert into cujosql.temp values ('compare')");
        s.close();
    }


    public void resetConnection(String property)
    throws SQLException
    {
        if (connection != null)
            connection.close();

        connection = DriverManager.getConnection(url + ";access=" + property);
    }


    public boolean canQuery() {
```

```
    Statement s = null;
    try {
        s = connection.createStatement();
        ResultSet rs = s.executeQuery("SELECT * FROM cujosql.temp");
        if (rs == null)
            return false;

        rs.next();

        if (rs.getString(1).equals("compare   "))
            return true;

        return false;

    } catch (SQLException e) {
        // System.out.println("Exception: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}


public boolean canUpdate() {
    Statement s = null;
    try {
        s = connection.createStatement();
        int count = s.executeUpdate("INSERT INTO CUJOSQL.TEMP VALUES('x')");
        if (count != 1)
            return false;

        return true;

    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}


public boolean canCall() {
    CallableStatement s = null;
    try {
        s = connection.prepareCall("? = CALL CUJOSQL.TEMP()");
        s.registerOutParameter(1, Types.INTEGER);
        s.execute();
        if (s.getInt(1) != 11)
            return false;

        return true;
```

```
        } catch (SQLException e) {
            //System.out.println("Exception: SQLState(" +
            //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
            return false;
        } finally {
            if (s != null) {
                try {
                    s.close();
                } catch (Exception e) {
                    // Ignore it.
                }
            }
        }
    }


    public void run()
    throws SQLException
    {
        System.out.println("Set the connection access property to read only");
        resetConnection("read only");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

        System.out.println("Set the connection access property to read call");
        resetConnection("read call");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

        System.out.println("Set the connection access property to all");
        resetConnection("all");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

    }


    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            // Ignore it.
        }
    }
}
```

範例：無效的使用者 ID 和密碼:

下列範例顯示如何以 SQL 命名模式來使用 Connection 內容。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////////////////////////////////////
//
// InvalidConnect example.
//
// This program uses the Connection property in SQL naming mode.
//
//////////////////////////////////////////////////////////////////////////
//
```

```
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// 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)
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: JDBC driver did not load.");
            System.exit(0);
        }

        // Attempt to obtain a connection without specifying any user or
        // password. The attempt works and the connection uses the
        // same user profile under which the job is running.
        try {
            Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
            c1.close();
        } catch (SQLException e) {
            System.out.println("This test should not get into this exception path.");
            e.printStackTrace();
            System.exit(1);
        }

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

    }
}
```

Connection 內容:

下表包含有效的 JDBC 驅動程式連線內容、內容值及其說明。

內容	值	意義
access	all, read call, read only	此值可限制特定連線所能執行的作業類型。預設值爲 all，基本上表示連線對 JDBC API 具有完整的存取權限。read call 值僅允許連線執行查詢及呼叫儲存程序。禁止透過 SQL 陳述式來更新資料庫。read only 值可將連線限制爲只能執行查詢。禁止呼叫儲存程序及使用更新陳述式。
auto commit	true, false	此值用來設定連線的自動確定設定值。預設值爲 true，除非 transaction isolation 內容已設爲 none 以外的值。在此情況下，預設值爲 false。
batch style	2.0, 2.1	JDBC 2.1 規格定義了第二種方法來決定如何處理批次更新的異常。此驅動程式與其中一者相容。預設值使用 JDBC 2.0 規格的定義。
block size	0, 8, 16, 32, 64, 128, 256, 512	此爲結果集一次提取的列數。在結果集常見的 forward-only 處理方式中，即先取得此大小的區塊。然後就不再存取資料庫，因爲應用程式已經處理每一列。僅於到達區塊的尾端時，資料庫才會再要求另一塊資料。 只有在 blocking enabled 內容設爲 true 時，才會用到此值。 將 block size 內容設爲 0 的效果，相當於將 blocking enabled 內容設爲 false。 預設值使用區塊大小 32 來傳送區塊。目前大多視情況任意決定，未來有可能改變預設值。 可捲動的結果集並不採用區塊傳輸。
blocking enabled	true, false	此值決定連線在擷取結果集的列時，是否使用區塊傳輸。區塊傳輸可大幅提升結果集的處理效能。 依預設，這個內容設定爲 true。

內容	值	意義
cursor hold	true, false	此值指定當異動確定之後，結果集是否保持開啓狀態。true 值表示應用程式在呼叫確定之後，可存取已開啓的結果集。false 值表示確定之後即關閉連線上任何開啓的游標。 依預設，這個內容設定爲 true。 這個內容值即爲連線上所有完成的結果集的預設值。由於 JDBC 3.0 新增的游標保留支援，若應用程式後來指定不同的保留方式，將會直接取代此預設值。 如果您要從較早的版本移轉至 JDBC 3.0，請注意在 JDBC 3.0 之前的版本中沒有新增游標保留支援。在較早的版本中，連接時會傳送預設值 ″true″，但是 JVM 無法辨識它。因此，在 JDBC 3.0 之前，游標保留內容不會影響資料庫功能。
data truncation	true, false	此值指定當字元資料被截斷時，是否應該產生警告及異常 (true)，或是在資料被截斷時不發出任何警告聲響 (false)。若預設值爲 true，則在字元欄位發生資料截斷時，將會做出回應。
date format	julian, mdy, dmy, ymd, usa, iso, eur, jis	這個內容可讓您變更日期的格式化方法。
date separator	″/″(斜線), ″-″(破折號), ″.″(句點), ″,″(逗點), ″b″	這個內容可讓您變更日期分隔字元。此內容必須搭配 dateFormat 值 (根據系統規則) 才有效。
decimal separator	″.″(句點), ″,″(逗點)	這個內容可讓您變更小數分隔字元。

內容	值	意義
do escape processing	true, false	這個內容可設定旗標，指定 Connection 下的陳述式是否必須處理跳出字元。採用跳出字元處理這種程式編寫方式，可讓 SQL 陳述式更通用且更相容於所有平台，但資料庫需要讀取跳出子句，爲使用者替換適當的系統專用字元。 這種做法很好，只是會強迫系統執行額外的工作。若已知 SQL 陳述式內只採用有效的iSeries SQL 語法，建議您將此值設爲 false 以提高效能。 因爲必須符合 JDBC 規格 (亦即，依預設會啓用跳出字元處理)，這個內容的預設值爲 true。 新增此值是爲了彌補 JDBC 規格的一項缺失。您只能在 Statement 子句中關閉跳出字元處理。在處理簡單的陳述式時，這沒有問題。您只要建立陳述式、關閉跳出字元處理，然後就開始處理陳述式。然而，在備妥陳述式及呼叫陳述式的情況下，就不適用這種方式。您需在建構備妥陳述式或呼叫陳述式時提供 SQL 陳述式，此後就不再變動。所以，事前已備妥陳述式，此後再變更跳出字元處理就沒有意義。請善用這個 Connection 內容，以避免額外的負荷。
errors	basic, full	這個內容可讓 SQLException 物件訊息傳回完整系統的第二層錯誤文字。預設值爲 basic，表示僅傳回標準的訊息文字。

內容	值	意義
檔案庫	以空格隔開的檔案庫清單。(亦可使用冒號或逗點來隔開檔案庫清單。)	使用此內容，即可在伺服器工作的檔案庫清單中加入一連串的檔案庫，或設定特定的預設檔案庫。 naming 內容會影響這個內容的運作方式。在預設情況下，naming 會設爲 sql，JDBC 將如同 ODBC 一般運作。檔案庫清單不影響連線的處理方式。所有未限定的表格都有一個預設檔案庫。依預設，此檔案庫與連接的使用者設定檔同名。若指定了 libraries 內容，則清單中的第一個檔案庫就是預設檔案庫。若在連線 URL 上指定了預設檔案庫 (例如在 jdbc:db2:*local/mylibrary 中)，則該預設值會取代此內容的值。 在 naming 設爲 system 的情況下，針對此內容所指定的每一個檔案庫，都會新增至檔案庫清單的使用者部份，所以會搜尋此檔案庫清單來解析未限定的表格參照。
lob threshold	小於 500000 的任何值	若 lob 直欄小於臨界值大小，這個內容會指示驅動程式在結果集儲存體中放入實際值，而非 lob 直欄的定位器。這個內容的處理對象是直欄大小，而非 lob 資料本身的大小。比方說，如果 lob 直欄定義爲每一個 lob 最多可存放 1 MB，但所有直欄值皆小於 500 KB，則仍會使用定位器。 請注意，大小限制的設定的目的，是爲了在提取資料區塊時，不會發生資料區塊超過最大配置大小 16 MB 的危險。至於較大的結果集，仍然很容易超出這項限制，導致提取失敗。請務必留意 block size 內容、這個內容及資料區塊的大小三者之間的交互影響。 預設值爲 0。一律在 lob 資料中使用定位器。
最大精準度	31, 63	此值指定結果資料類型所傳回的最大精準度 (長度)。預設值是 31。
最多小數位數	0-63	此值指定結果資料類型所傳回的最多小數位數 (小數點右邊的小數位數)。此值的範圍介於 0 至最大精準度之間。預設值是 31。

內容	值	意義
最少小數位數	0-9	此值指定中間資料類型與結果資料類型所傳回的最少小數位數 (小數點右邊的小數位數)。此值的範圍介於 0 至 9 之間，不超過最多小數位數。若指定 0，則不使用最少小數位數。預設值爲 0。
naming	sql, system	這個內容可讓您使用傳統的 iSeries 命名語法，或標準的 SQL 命名語法。系統命名表示您將使用 /(斜線) 字元來隔開集合值和表格值，SQL 命名則表示使用 . (句點) 字元來隔開值。 此值的設定亦決定預設檔案庫的最終結果。如需此內容的進一步的資訊，請參閱上面的檔案庫內容。 預設爲使用 SQL 命名。
password	任何值	這個內容可以指定連線的密碼。必須同時指定 user 內容，這個內容才能正確地發揮用途。這些內容可讓執行 iSeries 工作以外的使用者建立資料庫的連線。 指定 user 及 password 內容的效果，就如同使用簽名 getConnection(String url, String userId, String password) 的連線方法。
prefetch	true, false	這個內容指定是要在驅動程式處理之後，隨即提取結果集的第一筆資料，還是等到要求資料時再擷取。若預設值爲 true，則會預先提取資料。 對於使用「原有的」JDBC 驅動程式的應用程式，通常不必考慮這項內容。此內容主要供 Java 儲存程序及使用者定義的函數在內部使用，因爲在您要求資料之前，資料庫引擎不會代替您從結果集提取任何資料，這一點很重要。
reuse objects	true, false	這個內容指定某些類型的物件被您關閉之後，是否可供驅動程式再重覆使用。此爲效能加強功能。預設值爲 true。
server trace	整數的字串表示法	這個內容可以追蹤 JDBC 伺服器工作。若啓用伺服器追蹤，則會在用戶端連上伺服器時開始追蹤，而於斷線時結束追蹤。 收集的追蹤資料存放在伺服器的排存檔中。只要新增常數，並且在 set 方法上傳入總和，就可以同時開啓多層次的伺服器追蹤。 **註:** 此內容通常由支援人員使用，在此不再進一步討論其值。

內容	值	意義
time format	hms, usa, iso, eur, jis	這個內容可讓您變更時間值的格式化方法。
time separator	″:″(冒號), ″.″(句點), ″,″(逗點), ″b″	這個內容可讓您變更時間分隔字元。此內容必須搭配 timeFormat 值 (根據系統規則) 才有效。
trace	true, false	這個內容可以開啓連線的追蹤功能。這個內容可充當簡單的除錯輔助工具。 預設值爲 false，亦即不使用追蹤。
transaction isolation	none, read committed, read uncommitted, repeatable read, serializable	這個內容可讓您設定連線的異動隔離層次。將這個內容設爲特定層次，或在 Connection 介面的 setTransactionIsolation 方法上指定層次，效果都一樣。 這個內容的預設值爲 none，因爲 JDBC 預設爲自動確定模式。
translate binary	true, false	這個內容可強制 JDBC 驅動程式，將 binary 和 varbinary 資料值視爲 char 和 varchar 資料值。 這個內容的預設值爲 false，表示將二進位資料與字元資料視爲不同的資料。
轉換十六進位	binary, character	此值可在 SQL 表示式裡選取十六進位常數所用的資料類型。binary 設定表示十六進位常數將使用 BINARY 資料類型。character 設定表示十六進位常數將使用 CHARACTER FOR BIT DATA 資料類型。預設值爲 character。
use block insert	true, false	這個內容可讓原有的 JDBC 驅動程式進入區塊插入模式，以將資料區塊插入資料庫中。這是最佳化的批次更新模式。唯有確定應用程式不會破壞某些系統限制時，才可採用此最佳化模式，否則，資料插入會失敗，甚至可能造成資料毀損。 開啓此內容的應用程式在試圖執行批次更新時，僅連接本端系統。使用 DRDA 建立遠端連線，因爲無法在 DRDA 上管理區塊化插入。 應用程式亦必須確定是完全由含有 SQL insert 陳述式與 values 子句的 PreparedStatements，提供所有插入值參數的值。values 清單中不允許出現常數。此爲系統的區塊化插入引擎的基本要求。 預設值爲 false。

內容	值	意義
user	任何值	這個內容可以指定連線的使用者 ID。必須同時指定 password 內容，這個內容才能正確地發揮用途。這些內容可讓執行 iSeries 工作以外的使用者建立資料庫的連線。 指定 user 及 password 內容的效果，就如同使用簽名 getConnection(String url, String userId, String password) 的連線方法。

範例：建立 UDBDataSource 並連結 JNDI:

以下爲如何建立 UDBDataSource 並將其連結 JNDI 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
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
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS", ds);
    }
}
```

範例：建立 UDBDataSourceBind 並設定 DataSource 內容:

以下爲如何建立 UDBDataSource 並設定使用者 ID 及密碼作爲 DataSource 內容的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
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
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource "  +
                          "with cujo as the default " +
                          "profile to connect with.");

        // Provide a user ID and password to be used for
        // connection requests.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS2", ds);
    }
}
```

範例：連結 UDBDataSource 之前取得起始環境定義:

下列範例會在連結 UDBDataSource 之前先取得起始環境定義。 接著會在此環境定義上使用 lookup 方法，以傳回 DataSource 類型的物件，供應用程式使用。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Import the required packages. There is no
// driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Retrieve the bound UDBDataSource object using the
        // name with which it was previously bound. At runtime,
        // only the DataSource interface is used, so there
        // is no need to convert the object to the UDBDataSource
        // implementation class. (There is no need to know what
        // the implementation class is. The logical JNDI name is
        // only required).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Once the DataSource is obtained, it can be used to establish
        // a connection. This Connection object is the same type
```

```
        // of object that is returned if the DriverManager approach
        // to establishing connection is used. Thus, so everything from
        // this point forward is exactly like any other JDBC
        // application.
        Connection connection = ds.getConnection();

        // The connection can be used to create Statement objects and
        // update the database or process queries as follows.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // The connection is closed before the application ends.
        connection.close();
    }
}
```

範例：建立 UDBDataSource 並取得使用者 ID 與密碼:

以下範例將示範如何建立 UDBDataSource，並於執行時透過 getConnection 方法來取得使用者 ID 及密碼。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/// Import the required packages. There is
// no driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Retrieve the bound UDBDataSource object using the
        // name with which it was previously bound. At runtime,
        // only the DataSource interface is used, so there
        // is no need to convert the object to the UDBDataSource
        // implementation class. (There is no need to know
        // what the implementation class is. The logical JNDI name
        // is only required).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Once the DataSource is obtained, it can be used to establish
        // a connection. The user profile cujo and password newtiger
        // used to create the connection instead of any default user
        // ID and password for the DataSource.
        Connection connection = ds.getConnection("cujo", "newtiger");

        // The connection can be used to create Statement objects and
        // update the database or process queries as follows.
         Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }
```

```
        // The connection is closed before the application ends.
        connection.close();
    }
}
```

使用 DataSource 與 UDBDataSource:

DataSource 介面的設計，主要是爲了讓 Java Database Connectivity (JDBC) 驅動程式的使用更具彈性。

使用 DataSource 的過程可以分爲兩個階段：

- **部署**

  部署爲實際執行 JDBC 應用程式之前發生的設定階段。部署通常涉及設定 DataSource 的特定內容，然後利用「Java 命名和目錄介面 (JNDI)」連結至目錄服務。最常用的目錄服務爲 Lightweight Directory Access Protocol (LDAP)，但也可能是其他眾多的目錄服務，例如「共同物件要求分配管理系統架構 (CORBA) 物件服務」、「Java 遠端方法呼叫 (RMI)」或基礎的檔案系統。

- **使用**

  由於隔開 DataSource 的部署與執行時間兩者的用法，所以許多應用程式可以重覆使用 DataSource 設定。只要稍微變更部署方面的設定，所有使用此 DataSource 的應用程式就會自動套用這些變更。

註: 請注意，使用 RMI 可能讓工作更複雜。 在選擇 RMI 作爲解決方案之前，請確定您已瞭解後果如何。

DataSource 的一項優點是讓 JDBC 驅動程式代替應用程式來工作，不會直接影響到應用程式開發過程。如需詳細資訊，請參閱：

- 連線儲存區
- 陳述式儲存區作業
- 分散式異動

UDBDataSourceBind

UDBDataSourceBind 程式爲如何建立 UDBDataSource 並連結 JNDI 的範例。此程式會達成所有必要的基本作業。換言之，範例中會建立 UDBDataSource 物件的實例、設定此物件的內容、擷取 JNDI 環境定義，然後將物件連結至 JNDI 環境定義內的名稱。

部署時期的程式碼依開發廠商而定。應用程式必須匯入希望採用的特定 DataSource 實作方式。 在匯入清單中，請匯入完整套件的 UDBDataSource 類別。此應用程式中，最不熟悉的部份莫過於 JNDI 所處理的工作 (例如，擷取 Context 物件及連結的呼叫)。 如需相關資訊，請參閱 Sun Microsystems 公司的 JNDI。

此程式順利執行完成之後，JNDI 目錄服務中將出現 SimpleDS 這個新的項目。此項目出現在 JNDI 環境定義所指定的位置。到此已部署 DataSource 實作方式。 應用程式可以利用此 DataSource 來擷取資料庫連線及 JDBC 相關的工作。

UDBDataSourceUse

UDBDataSourceUse 程式爲 JDBC 應用程式的範例，將會用到先前已部署的應用程式。

就如同先前的範例一樣，在連結 UDBDataSource 之前，此 JDBC 應用程式也同樣先取得起始環境定義。接著會在此環境定義上使用 lookup 方法，以傳回 DataSource 類型的物件，供應用程式使用。

註: 此執行時間應用程式僅專注於處理 DataSource 介面的方法，所以不必顧及類別的實作方式。應用程式也因此更具可攜性。

假設 UDBDataSourceUse 是一個在組織內執行大量作業的綜合應用程式。而且，組織內尚有一大堆類似的大型應用程式。現在您必須在網路中變更一個系統的名稱。只要執行部署工具，再變更一個 UDBDataSource 內容，即可讓這項新的動作在所有應用程式中生效，完全不必改變程式碼。 DataSource 的一項好處是可以合併系統設定資訊。另一項主要的好處是讓驅動程式實作的功能不會干擾應用程式，例如，連線儲存區作業、陳述式儲存區作業及分散式異動的支援。

仔細分析 UDBDataSourceBind 與 UDBDataSourceUse 之後，將令您感到驚訝 DataSource 物件是如何辦到的。這些程式完全沒有程式碼來指定系統、使用者 ID 或密碼。因爲 UDBDataSource 類別已提供所有內容的預設值；依預設，一律以執行中應用程式的使用者設定檔及密碼來連接本端 iSeries 伺服器。若希望改用使用者設定檔 cujo 來完成連線，您有兩種方法可以執行：

- 設定使用者 ID 及密碼作爲 DataSource 內容。有關如何採取這項技術的資訊，請參閱範例：建立 UDBDataSourceBind 並設定 DataSource 內容。
- 使用 DataSource getConnection 方法，在執行時取得使用者 ID 及密碼。有關如何採取這項技術的資訊，請參閱範例：建立 UDBDataSource 並取得使用者 ID 與密碼。

UDBDataSource 有許多內容可以指定，就像 DriverManager 所建立的連線也有許多內容可以指定一樣。如需原有的 JDBC 驅動程式支援的內容清單，請參閱 DataSource 內容。

雖然這些清單很相似，但不保證未來版次中也仍然相似。建議您著手編寫 DataSource 介面的程式碼。

註: 原有的 JDBC 驅動程式也有兩個其他 DataSource 實作方式，但不建議直接採用。

- DB2DataSource
- DB2StdDataSource

DataSource 內容:

下表包含有效的資料來源內容、內容值及說明。

設定方法 (資料類型)	值	說明
setAccess(String)	all, read call, read only	這個內容可限制特定連線所能執行的作業類型。預設值爲 ″all″，基本上表示連線對 Java Database Connectivity (JDBC) API 具有完整的存取權限。 read call 值僅允許連線執行查詢及呼叫儲存程序。試圖透過 SQL 陳述式來更新資料庫會導致 SQLException。 read only 值限制連線只能執行查詢。試圖處理儲存程序呼叫或 UPDATE 陳述式，都會導致 SQLException。
setBatchStyle(String)	2.0, 2.1	JDBC 2.1 規格定義了第二種方法來決定如何處理批次更新的異常。此驅動程式與其中一者相容。預設值使用 JDBC 2.0 規格的定義。
setUseBlocking(boolean)	true, false	這個內容決定連線在擷取結果集的列時，是否使用區塊傳輸。區塊傳輸可大幅提升結果集的處理效能。 依預設，這個內容設定爲 true。

設定方法 (資料類型)	值	說明
setBlockSize(int)	0, 8, 16, 32, 64, 128, 256, 512	這個內容指出結果集一次提取的列數。在結果集常見的 forward only 處理方式中，即先取得此大小的區塊，前提是資料庫內有這麼多列滿足查詢準則。唯有當 JDBC 驅動程式的內部儲存體中到達區塊尾端時，才會再向資料庫送出另一個資料區塊要求。 僅於 useBlocking 內容設爲 true 時，才會用到此值。如需相關資訊，請參閱上述 setUseBlocking。 將 block size 內容設爲 0 的效果，相當於呼叫 setUseBlocking(false)。 預設值使用區塊大小 32 來傳送區塊。目前大多視情況任意決定，未來版次中有可能改變預設值。 可捲動的結果集並不採用區塊傳輸。 使用區塊傳輸會影響使用者應用程式的游標靈敏度。靈敏的游標可察覺其他 SQL 陳述式所做的變更。然而，由於資料快取的緣故，唯有需要從資料庫提取資料時，才會偵測到變更。
setCursorHold(boolean)	true, false	這個內容指定當異動確定之後，結果集是否保持開啓狀態。true 值表示應用程式在呼叫確定之後，可存取已開啓的結果集。false 值表示確定之後即關閉連線上任何開啓的游標。 依預設，這個內容設定爲 true。 這個內容值即爲連線上所有完成的結果集的預設值。由於 JDBC 3.0 新增的游標支援 (如需詳細資訊，請參閱 ResultSet 性質這一節)，若應用程式後來指定不同的游標支援，則會直接取代此預設值。
setDataTruncation(boolean)	true, false	這個內容指定： • 當字元資料被截斷時，是否應該產生警告及異常 (true) • 是否在資料被截斷時不發出任何警告聲響 (false)。 如需其他詳細資料，請參閱 DataTruncation。

設定方法 (資料類型)	值	說明
setDatabaseName(String)	任何名稱	這個內容指定 DataSource 試圖連接的資料庫。預設值爲 *LOCAL。資料庫名稱必須存在於關聯式資料庫目錄中 (在執行應用程式的系統上)，或指定爲特殊值 *LOCAL 或 localhost 來指定本端系統。
setDataSourceName(String)	任何名稱	這個內容允許傳遞 ConnectionPoolDataSource「Java 命名和目錄介面 (JNDI)」名稱來支援連線儲存區作業。
setDateFormat(String)	julian, mdy, dmy, ymd, usa, iso, eur, jis	這個內容可讓您變更日期的格式化方法。
setDateSeparator(String)	″/″, ″-″, ″.″, ″,″, ″b″	這個內容可讓您變更日期分隔字元。此內容必須搭配 dateFormat 值 (根據系統規則) 才有效。
setDecimalSeparator(String)	″.″, ″,″	這個內容可讓您變更小數分隔字元。
setDescription(String)	任何名稱	這個內容允許設定此 DataSource 物件的文字說明。
setDoEscapeProcessing(boolean)	true, false	這個內容指定 SQL 陳述式是否處理跳出字元。 這個內容的預設值爲 true。
setFullErrors(boolean)	true, false	這個內容可讓 SQLException 物件訊息傳回完整系統的第二層錯誤文字。預設值爲 false。
setLibraries(String)	以空格隔開的檔案庫清單	使用此內容，即可在伺服器工作的檔案庫清單中加入一連串的檔案庫。僅於使用 setSystemNaming(true) 時，才會用到這個內容。
setLobThreshold(int)	小於 500000 的任何值	若 LOB 直欄小於臨界值大小，這個內容會指示驅動程式放入實際值，而非「定位器物件 (LOB)」定位器。
setLoginTimeout(int)	任何值	目前忽略這個內容，計劃將在未來使用。
setNetworkProtocol(int)	任何值	目前忽略這個內容，計劃將在未來使用。
setPassword(String)	任何字串	這個內容可以指定連線的密碼。若未設定使用者 ID，則會忽略這個內容。
setPortNumber(int)	任何值	目前忽略這個內容，計劃將在未來使用。
setPrefetch(boolean)	true, false	這個內容指定是要在驅動程式處理之後，隨即提取結果集的第一筆資料，還是等到要求資料時再擷取。預設值爲 true。
setReuseObjects(boolean)	true, false	這個內容指定某些類型的物件被您關閉之後，是否可供驅動程式再重覆使用。此爲效能加強功能。預設值爲 true。

設定方法 (資料類型)	值	說明
setServerName(String)	任何名稱	目前忽略這個內容，計劃將在未來使用。
setServerTraceCategories(int)	整數的字串表示法	這個內容可以追蹤 JDBC 伺服器工作。若啓用伺服器追蹤，則會在用戶端連上伺服器時開始追蹤，而於斷線時結束追蹤。 收集的追蹤資料存放在伺服器的排存檔中。只要新增常數，並且在 set 方法上傳入總和，就可以同時開啓多層次的伺服器追蹤。 **註:** 此內容通常由支援人員使用，在此不再進一步討論其值。
setSystemNaming(boolean)	true, false	這個內容可以指定用句點 (SQL 命名) 或斜線 (系統命名) 來隔開集合與表格。這個內容也決定是要使用預設的檔案庫 (SQL 命名)，還是使用檔案庫清單 (系統命名)。預設值爲 SQL 命名。
setTimeFormat(String)	hms, usa, iso, eur, jis	這個內容可讓您變更時間值的格式化方法。
setTimeSeparator(String)	″:″, ″.″, ″,″, ″b″	這個內容可讓您變更時間分隔字元。此內容必須搭配 timeFormat 值 (根據系統規則) 才有效。
setTrace(boolean)	true, false	這個內容可以啓用簡單的追蹤。預設值爲 false。
setTransactionIsolationLevel(String)	none, read committed, read uncommitted, repeatable read, serializable	這個內容可以指定異動隔離層次。這個內容的預設值爲 none，因爲 JDBC 預設爲自動確定模式。
setTranslateBinary(Boolean)	true, false	這個內容可強制 JDBC 驅動程式，將 binary 和 varbinary 資料值視爲 char 和 varchar 資料值。 這個內容的預設值爲 false。

設定方法 (資料類型)	值	說明
setUseBlockInsert(boolean)	true, false	這個內容可讓原有的 JDBC 驅動程式進入區塊插入模式，以將資料區塊插入資料庫中。這是最佳化的批次更新模式。唯有確定應用程式不會破壞某些系統限制時，才可採用此最佳化模式，否則，資料插入會失敗，甚至可能造成資料毀損。 開啓此內容的應用程式在試圖執行批次更新時，僅連接本端系統。不會使用 DRDA 來建立遠端連線，因爲無法在 DRDA 上管理區塊化插入。 應用程式亦必須確定是完全由含有 SQL insert 陳述式與 values 子句的 PreparedStatements，提供所有插入值參數的值。values 清單中不允許出現常數。此爲系統的區塊化插入引擎的基本要求。 預設值爲 false。
setUser(String)	任何值	這個內容允許設定使用者 ID 來取得連線。您必須同時設定 password 內容，才能使用這個內容。

其他 DataSource 實作方式:

DataSource 介面在原有的 JDBC 驅動程式內有兩種實作方式。 但這些 DataSource 實作方式應該視爲已作廢。這些實作方式雖然可繼續使用，但未來都不再進行功能上的加強與改進； 舉例來說，它們不會再新增健全的連線及陳述式儲存區作業。 在您採用 UDBDataSource 介面及其相關的函數之前，這些實作方式會繼續存在。

DB2DataSource

DB2DataSource 是 DataSource 介面早期的實作方式， 不符合完整的規格 (亦即，規格出現之前就已存在)。DB2DataSource 至今仍然存在的原因，只爲了讓 WebSphere® 使用者移轉至現行版次，此外無其他用途。

DB2StdDataSource

DB2StdDataSource 是 DB2DataSource 實作方式的改良版， 當 JDBC 選用性套件規格全部完成時，即成爲符合規格的實作方式。新版本不會中斷支援 DB2DataSource 版本上所編寫的程式碼。

若您的應用程式已採用這些 DataSource 實作方式來編寫， 則移轉至 UDBDataSource 將毫不費力，因爲所有舊的內容都受支援。建議您移轉至 UDBDataSource 來享受 UDBDataSource 類別的新功能。

JDBC 的 JVM 內容

連線內容無法設定原有 JDBC 驅動程式所使用的一些設定值。 在執行原有的 JDBC 驅動程式的 JVM 中，必須設定這些設定值。 原有的 JDBC 驅動程式所建立的所有連線都會用到這些設定值。

原有的驅動程式可以辨識下列 JVM 內容：

內容	值	意義
jdbc.db2.job.sort.sequence	預設值 = *HEX	將這個內容設爲 true，將使原有的 JDBC 驅動程式改採使用者 (原先啓動工作的使用者) 的「工作排序順序」，而非使用預設值 *HEX。若設爲其他值或維持不設定，JDBC 則會繼續使用預設值 *HEX。請仔細思考其中的涵義。 當 JDBC 連線在連線要求上傳入不同的使用者設定檔時，所有連線都會使用啓動伺服器的使用者設定檔的排序順序。此爲啓動時設定的環境屬性，並非動態連線屬性。
jdbc.db2.trace	1 或 error = 追蹤錯誤資訊，2 或 info = 追蹤資訊及錯誤資訊，3 或 verbose = 追蹤詳述、資訊及錯誤資訊，4 或 all or true = 追蹤所有可能的資訊	這個內容會開啓 JDBC 驅動程式的追蹤功能。應該在報告問題時使用。
jdbc.db2.trace.config	stdout = 將追蹤資訊傳送至 stdout (預設值) usrtrc = 將追蹤資訊傳送至使用者追蹤裝置。可以使用 CL 指令「傾出使用者追蹤緩衝區 (DMPUSRTRC)」來取得追蹤資訊。file://<pathtofile> = 將追蹤資訊傳送至檔案。若檔名含有 ″%j″，則 ″%j″ 將置換成工作名稱。 例如，<pathtofile> 可能是 /tmp/jdbc.%j.trace.txt。	這個內容用來指定追蹤的輸出位置。

IBM Developer Kit for Java 的 DatabaseMetaData 介面

DatabaseMetaData 介面由 IBM Developer Kit for Java JDBC 驅動程式實作，可提供其基礎資料來源的相關資訊。此介面主要供應用程式伺服器及工具使用，以決定如何與給定的資料來源相互作用。應用程式也可以使用 DatabaseMetaData 方法來取得資料來源的相關資訊，但這種情況較少見。

DatabaseMetaData 介面提供 150 種以上的方法，可按其提供的資訊類型來分類。如下所述。DatabaseMetaData 介面亦包含 40 個以上的欄位，皆爲常數，作爲各種 DatabaseMetaData 方法的回覆值。

如需 DatabaseMetaData 介面中各種方法的變更相關資訊，請參閱「JDBC 3.0 的變更」。

建立 DatabaseMetaData 物件

DatabaseMetaData 物件是透過 Connection 方法 getMetaData 而建立的。此物件建立之後，即可用來動態地尋找基礎資料來源的相關資訊。在下列範例中，將建立一個 DatabaseMetaData 物件，用來決定表格名稱容許的最大字元數：

範例：建立 DatabaseMetaData 物件

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// con is a Connection object
DatabaseMetaData dbmd = con.getMetadata();
int maxLen = dbmd.getMaxTableNameLength();
```

擷取一般資訊

有些 DatabaseMetaData 方法可用來動態地尋找資料來源的一般資訊，亦可用來取得實作方式的詳細資料。這些方法包括：

- getURL
- getUserName
- getDatabaseProductVersion、getDriverMajorVersion 及 getDriverMinorVersion
- getSchemaTerm、getCatalogTerm 及 getProcedureTerm
- nullsAreSortedHigh 及 nullsAreSortedLow
- usesLocalFiles 及 usesLocalFilePerTable
- getSQLKeywords

判斷功能支援

有許多 DatabaseMetaData 方法，可用來判斷驅動程式或基礎資料來源是否支援某特定的功能或功能集。此外，還有方法可以說明提供的支援層次。以下方法可說明個別功能所提供的支援：

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

可說明功能支援層次的方法包括：

- supportsANSI92EntryLevelSQL
- supportsCoreSQLGrammar

資料來源限制

還有另一組方法可說明特定資料來源所賦予的限制。屬於此種類的方法包括：

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

此群組的方法以整數傳回限制值。回覆值 0 表示沒有限制或限制不明。

SQL 物件及其屬性

針對在特定資料來源中輸入資料的 SQL 物件，有許多 DatabaseMetaData 方法可提供其相關資訊。這些方法可判斷 SQL 物件的屬性。這些方法也會傳回 ResultSet 物件，其中每一列各說明一個特定的物件。例如，getUDTs 方法傳回的 ResultSet 物件中，就有一列用來說明資料來源中已定義的各個使用者定義的表格 (UDT)。屬於此種類的例子有：

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

異動支援

對於資料來源所支援的異動語意，有少許方法可提供相關資訊。屬於此種類的例子有：

- supportsMultipleTransactions
- getDefaultTransactionIsolation

如需如何使用 DatabaseMetaData 介面的範例，請參閱範例：IBM Developer Kit for Java 的 DatabaseMetaData 介面。

JDBC 3.0 的變更

在 JDBC 3.0 中，部份方法的回覆值有變動。在 JDBC 3.0 已更新下列方法，在傳回的 ResultSet 中加入一些欄位。

- getTables
- getColumns
- getUDTs
- getSchemas

註: 若是以 Java Development Kit (JDK) 1.4 來開發應用程式，您可能會發覺在測試時會傳回許多直欄。您可以編寫應用程式來存取這些直欄。然而，若應用程式亦設計爲可在前版次的 JDK 上執行，則當應用程式嘗試存取這些在先前 JDK 版次中不存在的欄位時，將會收到 SQLException。請參閱 SafeGetUDTs 範例，瞭解如何編寫同時適用於數個 JDK 版次的應用程式。

範例：IBM Developer Kit for Java 的 DatabaseMetaData 介面 - 傳回表格清單:

本範例顯示如何傳回表格清單。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

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

// Get the database meta data from the connection.
     DatabaseMetaData dbMeta = c.getMetaData();

// Get a list of tables matching this criteria.

String catalog = "myCatalog";
String schema = "mySchema";
String table  = "myTable%"; // % indicates search pattern
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"}:
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterate through the ResultSet to get the values.

// Close the connection.
c.close():
```

如需相關資訊，請參閱 IBM Developer Kit for Java 的 DatabaseMetaData 介面。

範例：使用含有多個直欄的 meta 資料 ResultSet:

以下範例說明如何使用含有多個直欄的 meta 資料 ResultSet。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////////////////////////////////////////
//
// SafeGetUDTs example. This program demonstrates one way to deal with
// metadata ResultSets that have more columns in JDK 1.4 than they
// had in previous releases.
//
// Command syntax:
//    java SafeGetUDTs
//
//////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// 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;

    // Note: Static block runs before main begins.
    // Therefore, there is access to jdbcLevel in
    // main.
    {
        try {
            Class.forName("java.sql.Blob");

            try {
                Class.forName("java.sql.ParameterMetaData");
                // Found a JDBC 3.0 interface.  Must support JDBC 3.0.
                jdbcLevel = 3;
            } catch (ClassNotFoundException ez) {
                // Could not find the JDBC 3.0 ParameterMetaData class.
                // Must be running under a JVM with only JDBC 2.0
                // support.
                jdbcLevel = 2;
            }

        } catch (ClassNotFoundException ex) {
            // Could not find the JDBC 2.0 Blob class.  Must be
```

```
            // running under a JVM with only JDBC 1.0 support.
            jdbcLevel = 1;
        }
    }

    // Program entry point.
    public static void main(java.lang.String[] args)
    {
        Connection c = null;

        try {
            // Get the driver registered.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

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

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

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

                // Fetch all the columns that have been available since the
                // JDBC 2.0 release.
                System.out.println("TYPE_CAT is " + rs.getString("TYPE_CAT"));
                System.out.println("TYPE_SCHEM is " + rs.getString("TYPE_SCHEM"));
                System.out.println("TYPE_NAME is " + rs.getString("TYPE_NAME"));
                System.out.println("CLASS_NAME is " + rs.getString("CLASS_NAME"));
                System.out.println("DATA_TYPE is " + rs.getString("DATA_TYPE"));
                System.out.println("REMARKS is " + rs.getString("REMARKS"));

                // Fetch all the columns that were added in JDBC 3.0.
                if (jdbcLevel > 2) {
                    System.out.println("BASE_TYPE is " + rs.getString("BASE_TYPE"));
                }
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            if (c != null) {
                try {
                    c.close();
                } catch (SQLException e) {
                    // Ignoring shutdown exception.
                }
            }
        }
    }
}
```

異常情況

Java 語言使用異常來提供程式的錯誤處理能力。異常就是程式執行時中斷正常指令流程的事件。

Java Runtime 系統及 Java 套件的許多類別，都會在某些情況下利用 throw 陳述式擲出異常。在您的 Java 程式中，亦可採用相同的機制來擲出異常。

SQLException:

SQLException 類別及其次類型所提供的資訊，可說明存取資料來源時發生的錯誤及警告。

不同於在介面中定義的大部份 JDBC，異常支援是由類別所提供。執行 JDBC 時所發生的異常，其基礎類別爲 SQLException。JDBC API 的每一個方法，皆宣告爲可以丟出 SQLException。SQLException 是 java.lang.Exception 的延伸類別，對於資料庫環境中發生的失敗事件，可提供相關的資訊。明確地說，SQLException 可提供下列資訊：

- 文字說明
- SQLState
- 錯誤碼
- 對其他同時發生之異常的參照

ExceptionExample 程式會適當地攔截 (此案例預期會發生的) SQLException，然後傾出其中所有的資訊。

註: JDBC 亦提供機制來串連異常。這樣可讓驅動程式或資料庫利用單一要求來報告多個錯誤。至於原有的 JDBC 驅動程式是否實際運用到這項機制，目前尚無實例。當然，這項資訊僅供參考，並不代表驅動程式就永遠不會這樣做。

根據前文，當發生錯誤時，就會丟出 SQLException 物件。的確如此，但這樣的說法並不完整。實際上，原有的 JDBC 驅動程式很少丟出眞正的 SQLException。而只是丟出 SQLException 子類別的實例。這可讓您判斷實際失敗狀況的更多資訊，如下所示。

DB2Exception.java

DB2Exception 物件也不是直接就丟出。此基礎類別的用途是提供所有 JDBC 異常通用的功能。此類別有兩個子類別，用來作爲 JDBC 丟出的標準異常。這兩個子類別就是 DB2DBException.java 與 DB2JDBCException.java。DB2DBException 是直接從資料庫向您報告的異常。當 JDBC 驅動程式發現本身有問題時，就會丟出 DB2JDBCException。依此法分割異常類別階層，可讓您用不同方式來處理這兩種異常。

DB2DBException.java

根據上一段內容，可知 DB2DBException 是直接來自資料庫的異常。當 JDBC 驅動程式呼叫 CLI 之後，若收到 SQLERROR 回覆碼，則會遭遇這些異常。在這種情況下，請呼叫 CLI 函數 SQLError 來取得訊息文字、SQLState 及供應商代碼。這樣也會擷取 SQLMessage 的取代文字並傳回給您。DatabaseException 類別會造成資料庫可辨識的錯誤，並且向 JDBC 驅動程式報告以建置異常物件。

DB2JDBCException.java

DB2JDBCException 因 JDBC 驅動程式本身的錯誤狀況而產生。此異常類別的功能基本上就不相同； JDBC 驅動程式本身會處理異常的訊息語言轉換，而對於資料庫所引起的異常，雖有作業系統及資料庫負責處理，但也會一併處理這些問題。JDBC 驅動程式會儘可能配合資料庫的 SQLState。JDBC 驅動程式所丟出的異常，其供應商代碼一律爲 -99999。CLI 層可辨識及傳回的 DB2DBException，通常是含有 -99999 錯誤碼。JDBCException 類別會造成 JDBC 驅動程式可辨識的錯誤，並且爲自已建置異常。在該版次開發期間執行此範例時，會建立以下輸出。請注意，堆疊頂端含有 DB2JDBCException。這表示在向資料庫提出要求之前，JDBC 驅動程式會先報告錯誤。

範例：SQLException:

此爲攔截 SQLException 並傾出其中所有資訊的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;

public class ExceptionExample {
```

```
    public static Connection connection = null;

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

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

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

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


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

SQLWarning:

在某些介面中，若方法導致資料庫存取警告，則會產生 SQLWarning 物件。

下列介面中的方法可以產生 SQLWarning：

- Connection
- Statement 及其次類型、PreparedStatement 及 CallableStatement
- ResultSet 介面

當某個方法產生 SQLWarning 物件時，呼叫程式不會得知已發生資料存取警告。必須在適當的物件上呼叫 getWarnings 方法，才能擷取 SQLWarning 物件。 然而，在某些情況下，可能會丟出 SQLWarning 的 DataTruncation 子類別。值得注意的是，原有的 JDBC 驅動程式會選擇忽略資料庫產生的部份警告，藉此提升效率。舉例來說，假設您試圖透過 ResultSet.next 方法來擷取資料時，超過了 ResultSet 的尾端，因而產生警告。在此情況下，next 方法會根據定義傳回 false，而非 true，通知您已發生錯誤。此時沒有必要特地建立物件來重述此情況，因此直接忽略警告即可。

多個資料存取警告發生時，會全部鏈接至第一個警告，可呼叫 SQLWarning.getNextWarning 方法來加以擷取。若鏈結中已無警告，getNextWarning 會傳回 null。

後續的 SQLWarning 物件會持續新增至鏈結中，直到處理下一個陳述式爲止；若是在 ResultSet 物件的情況下，則會到游標重新定位時才停止。最後，鏈結中的所有 SQLWarning 物件都會移除。

使用 Connection、Statement 及 ResultSet 物件，即可能產生 SQLWarning。 SQLWarning 爲參考訊息，指出特定作業雖已順利完成，但可能有其他值得注意的資訊。雖然 SQLWarning 延伸自 SQLException 類別，但並不會丟出。它們會附加在產生它們的物件上。SQLWarning 產生時，並沒有任何機制會向應用程式通知警告的產生。應用程式必須主動要求警告資訊。

就像 SQLException 一樣，SQLWarning 亦可鏈結起來。 您可以在 Connection、Statement 或 ResultSet 物件上呼叫 clearWarnings 方法，來清除此物件的警告。

註: 呼叫 clearWarnings 方法不會清除所有警告。僅清除特定物件的相關警告。

若您未手動清除 SQLWarning 物件，JDBC 驅動程式會在特定的時間替您清除。發生下列動作時，就是清除 SQLWarning 物件的時機：

- 以 Connection 介面而言，建立新的 Statement、PreparedStatement 或 CallableStatement 物件時，就會清除警告。
- 以 Statement 介面而言，處理下一個陳述式時就會清除警告 (或針對 PreparedStatement 及 CallableStatement 再次處理該陳述式時)。
- 以 ResultSet 介面而言，重新定位游標時會清除警告。

DataTruncation 及自動截斷:

DataTruncation 是 SQLWarning 的子類別。在未丟出 SQLWarning 的情況下，DataTruncation 物件有時可以像其他 SQLWarning 物件一樣丟出或附加。當直欄的大小超出 setMaxFieldSize 陳述式方法指定的大小時，便會發生自動截斷，但不會報告任何警告或異常。

DataTruncation 物件提供的資訊，比 SQLWarning 傳回的資訊更詳細。可用的資訊包括：

- 已傳送的資料位元組數。
- 遭截斷的直欄或參數索引。
- 索引屬於參數或 ResultSet 直欄。
- 讀取或寫入資料庫時，是否發生截斷。
- 實際已傳送的資料量。

在某些情況下，可以解除資訊加密，但有些狀況無法完全掌控。比方說，如果使用 PreparedStatement 的 setFloat 方法，在存放整數值的欄位中插入一個值，則可能導致 DataTruncation，因爲浮點數可能大於此直欄可存放的最大值。在這些狀況下，截斷作業的位元組計數毫無意義，但在協助驅動程式提供截斷資訊方面，卻很重要。

報告 set() 與 update() 方法

各 JDBC 驅動程式之間有些細微的差異。有些驅動程式，例如原有的 JDBC 驅動程式及 IBM Toolbox for Java JDBC 驅動程式，會在設定參數時捕捉及報告資料截斷問題。這會在 PreparedStatement set 方法或 ResultSet update 方法上完成。其他驅動程式則是在處理陳述式時報告問題，且透過 execute、executeQuery 或 updateRow 方法來完成。

在您提供錯誤的資料時，問題報告就失敗，而不是等到處理程序完全無法進行時，才發生問題報告失敗，這樣的作法有兩種好處：

- 發生問題時，就直接在應用程式中解決失敗狀況，不必等到處理時才解決問題。
- 因爲在設定參數時就檢查，JDBC 驅動程式在處理陳述式時，可確定傳給資料庫的值一定有效。如此一來，資料庫可將工作最佳化，而且更快完成處理程序。

ResultSet.update() 方法丟出 DataTruncation 異常

以往一些版次中，當截斷狀況發生時，ResultSet.update() 方法會發出警告。將於資料庫內插入資料值時，即可能發生此情況。規格中已載明，JDBC 驅動程式會在這些情況下丟出異常。因此，JDBC 驅動程式以這種方式運作。

在處理收到資料截斷錯誤的 ResultSet update 函數時，以及針對收到錯誤的 update 或 insert 陳述式來處理備妥陳述式參數集時，兩者之間並無明顯的差別。兩者的問題都一樣，就是您提供的資料不符規定。

NUMERIC 及 DECIMAL 會直接截斷小數點右邊的位數，不發出警告。這就是 JDBC for UDB NT 的運作方法，也是互動式 SQL 在 iSeries 伺服器上的運作方式。

附註：資料遭到截斷時，值不會進行四捨五入。參數中無法填入 NUMERIC 或 DECIMAL 直欄的任何小數部份，將直接流失而無任何警告。

範例如下，其中假設 values 子句裡的值，實際上就是備妥陳述式上設定的參數：

```
create table cujosql.test (col1 numeric(4,2))
a) insert into cujosql.test values(22.22)  // works - inserts 22.22
b) insert into cujosql.test values(22.223) // works - inserts 22.22
c) insert into cujosql.test values(22.227) // works - inserts 22.22
d) insert into cujosql.test values(322.22) // fails - Conversion error on assignment to column COL1.
```

資料截斷警告與資料截斷異常的差別

規格中指出，當寫入資料庫的值發生資料截斷狀況時，就會丟出異常。若寫入資料庫的值並未進行資料截斷，仍產生警告。這就表示在某處偵測到資料截斷狀況，您亦必須注意資料截斷所處理的陳述式類型。依此要求爲準，以下列出幾種 SQL 陳述式類型的行爲：

- 在 SELECT 陳述式中，查詢參數絕不會損壞資料庫內容。因此，一律用警告來表示資料截斷狀況。
- 在 VALUES INTO 及 SET™ 陳述式中，輸入值僅用於產生輸出值。因此會發出警告。
- 在 CALL 陳述式中，JDBC 驅動程式無法判斷儲存程序如何處理參數。當儲存程序參數遭到截斷時，一律丟出異常。
- 其他所有陳述式類型都會丟出異常，而非發出警告。

Connection 的 data truncation 內容與 DataSource

許多版次中都有 data truncation 內容。此內容的預設值爲 true，表示會檢查資料截斷問題，然後發出警告或丟出異常。如有某個值無法填入資料庫直欄，但對您而言並不重要，在這種情況下，基於方便性及效能考量，即可使用此內容。您可讓驅動程式儘量將整個值全部放入直欄中。

data truncation 內容僅對字元及二進位種類的資料類型有效

在從前幾個版次中，data truncation 內容可決定是否丟出資料截斷異常。data truncation 內容的用途，就是讓 JDBC 應用程式不必擔心某個值遭到截斷，因爲當時對它們而言，是否截斷並不重要。當應用程式試圖在 DECIMAL(2.0) 中插入 100 時，想必您不希望資料庫中僅儲存 00 或 10。因此，JDBC 驅動程式的 data truncation 內容已有所變更，現在僅適用於字元類型的參數，例如 CHAR、VARCHAR、CHAR FOR BIT DATA 及 VARCHAR FOR BIT DATA。

data truncation 內容僅適用於參數

data truncation 內容是 JDBC 驅動程式的設定，不是資料庫的設定。因此，它並不影響陳述式文字。例如，以下處理的陳述式嘗試在資料庫的 CHAR(8) 直欄中插入一個值，但由於 data truncation 旗標已設爲 false，所以仍然會失敗 (假設 connection 爲他處已配置的 java.sql.Connection 物件)。

```
Statement stmt = connection.createStatement();
Stmt.executeUpdate("create table cujosql.test (col1 char(8))");
Stmt.executeUpdate("insert into cujosql.test values('dettinger')");
// Fails as the value does not fit into database column.
```

即使不重要的資料截斷，原有的 JDBC 驅動程式也會丟出異常

原有的 JDBC 驅動程式不會查看您提供給參數的資料。因爲這樣只會讓處理程序更慢。然而有些時候，某個值是否遭到截斷，對您而言完全不重要，但 data truncation connection 內容並未設定爲 false。

例如，傳遞 'dettinger ' 這個 char(10) 時，即使此值已符合所有重要的規則，仍然會丟出異常。這就是 JDBC for UDB NT 的運作方式； 然而，若是在 SQL 陳述式中以文字傳遞此值，就不會得到這種結果。在相同的情況下，資料庫引擎會直接捨棄額外的空格，不發出警告。

JDBC 驅動程式未丟出異常的問題如下：

- 每一個 set 方法不論是否需要，所造成的效能負荷都太大。一般而言，之所以會有負面的結果，通常導因於某個函數所造成的龐大效能負荷，例如最常見的 setString()。
- 您的解決之道通常無濟於事，例如，您可能會針對傳入的字串值呼叫 trim 函數。
- 您必須考慮一些關於資料庫直欄的問題。CCSID 37 的空格，在 CCSID 65535 或 13488 中並非空格。

自動截斷

setMaxFieldSize 陳述式方法可以指定任何直欄的最大欄位大小。若是因爲大小超過最大欄位大小值而截斷資料，並不會發出警告或異常。如同先前提及的 data truncation 內容，此方法也只會影響字元類型，例如 CHAR、VARCHAR、CHAR FOR BIT DATA 及 VARCHAR FOR BIT DATA。

異動

異動就是工作的邏輯單元。若要完成工作邏輯單元，可能必須對資料庫採取幾項動作。

採用異動支援，應用程式可提供下列保證：

- 確實遵循一項工作邏輯單元完成時所需的步驟。
- 若工作單元檔案的其中一個步驟失敗，仍可還原工作邏輯單元內已完成的任何工作，讓資料庫回到異動開始之前的狀態。

異動可提供資料整合性、正確的應用語意，以及在並行存取資料時的一致性。所有 Java Database Connectivity (JDBC) 相容的驅動程式皆必須支援異動。

註: 本節僅討論本端異動及異動的標準 JDBC 概念。Java 及原有的 JDBC 驅動程式支援 Java Transaction API (JTA)、分散式異動及兩階段確定通訊協定 (2PC)。

所有異動工作皆以 Connection 物件層次來處理。當一項異動的工作完成時，即可呼叫 commit 方法來終結。 若應用程式捨棄異動，則應該呼叫 rollback 方法。

一個連線之下的所有 Statement 物件，皆屬於異動的一部份。這表示若應用程式建立三個 Statement 物件，且利用每一個物件來變更資料庫， 則一旦呼叫 commit 或 rollback，這三個陳述式的所有工作就永久完成或捨棄。

純粹搭配 SQL 使用時，commit 及 rollback SQL 陳述式可用來終結異動。這些 SQL 陳述式並無法動態地進行準備，因此不應該在 JDBC 應用程式中用來完成異動。

自動確定模式:

依預設，JDBC 採用「自動確定」這種作業模式。這表示資料庫的每一項更新動作皆立即永久完成。

萬一工作邏輯單元需要更新資料庫一次以上，則無法於自動確定模式之下安全地完成。若在執行某項更新之後與執行其他更新前的這段時間，應用程式或系統發生問題，自動確定模式將無法還原第一個變更。

因爲在自動確定模式下會立即執行永久變更，所以就沒有必要由應用程式來呼叫 commit 方法或 rollback 方法。也因此編寫應用程式將更加容易。

連線存在期間可以動態地啓用及停用自動確定模式。假設資料來源已存在，自動確定的啓用方式如下：

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

Connection.setAutoCommit(false);        // Disables auto-commit.
```

若於異動期間變更自動確定的設定，則任何擱置中的工作都會自動確定。對於分散式異動中的連線啓用自動確定，將產生 SQLException。

異動隔離層次:

異動隔離層次可指定異動內的陳述式可以使用哪些資料。在相同目標資料來源的異動之間，只要定義可能的互動行爲，這些層次就會直接影響到並行存取的程度。

資料庫反常

資料庫反常表示從單一異動的角度看產生的結果似乎不正確，但從所有異動的整體角度來看卻又正確。以下說明不同類型的資料庫反常：

- 下列狀況會發生**錯誤**讀取：
  1. 異動 A 在表格內插入一列。
  2. 異動 B 讀取此列。
  3. 異動 A 回轉。

  異動 B 可能已根據異動 A 插入的列而對系統執行工作，但此列卻永遠都不會變成資料庫的永久部份。
- 下列狀況會發生**非連續**讀取：
  1. 異動 A 讀取一列。
  2. 異動 B 變更此列。
  3. 異動 A 再一次讀取同一列，而得到新的結果。
- 下列狀況會發生**幻象**讀取：
  1. 異動 A 讀取 SQL 查詢上滿足 WHERE 子句的所有列。
  2. 異動 B 另外插入一列，此列滿足 WHERE 子句。
  3. 異動 A 重新評估 WHERE 條件，並取得此額外新列。

註: 在前述層次上，DB2 for iSeries 基於鎖定策略，並不一定會讓應用程式發生容許的資料庫反常。

JDBC 異動隔離層次

IBM Developer Kit for Java JDBC API 有 5 種異動隔離層次。以下就依最低限制到最高限制列出：

JDBC_TRANSACTION_NONE
此爲特殊常數，指出 JDBC 驅動程式不支援異動。

JDBC_TRANSACTION_READ_UNCOMMITTED
此層次容許異動得知資料上未確定的變更。所有資料庫反常都可能出現於此層次上。

JDBC_TRANSACTION_READ_COMMITTED
此層次表示確定異動之前，異動內的任何變更皆不爲外界所知。所以完全不會發生錯誤讀取的情形。

JDBC_TRANSACTION_REPEATABLE_READ
此層次表示讀取的列會保持鎖定，因此在完成異動之前，另一個異動無法變更這幾列。如此就不容許錯誤讀取及非連續讀取。但仍有可能發生幻象讀取。

JDBC_TRANSACTION_SERIALIZABLE
已針對異動來鎖定表格，因此，目標是在表格中新增值或移除值的其他異動，無法變更 WHERE 條件。如此可以防止所有類型的資料庫反常狀況。

可以使用 setTransactionIsolation 方法來變更連線的異動隔離層次。

使用 ANZJVM 指令的注意事項

通常會誤解是 JDBC 規格定義上述的 5 個異動層次。一般都以爲 TRANSACTION_NONE 值代表在沒有確定控制的情況下執行的概念。實事上，JDBC 規格並非如此定義 TRANSACTION_NONE。TRANSACTION_NONE 是 JDBC 規格所定義的一種層次，該層次中，驅動程式不支援異動，且不是 JDBC 相容的驅動程式。 呼叫 getTransactionIsolation 方法時，絕不會報告 NONE 層次。

由於 JDBC 驅動程式的預設隔離層次是在實作方式上決定，所以這個問題不難解決。原有的 JDBC 驅動程式預設的異動隔離層次爲 NONE。這可讓驅動程式處理無異動日誌的檔案，不必您做任何指定，例如 QGPL 檔案庫裡的檔案。

原有的 JDBC 驅動程式可讓您在 setTransactionIsolation 方法中傳入 JDBC_TRANSACTION_NONE，或指定 none 作爲連線內容。但若值爲 none，則 getTransactionIsolation 方法一律會報告 JDBC_TRANSACTION_READ_UNCOMMITTED。如有必要，應用程式應該負責追蹤您正在執行的層次。

在以往版次中，即使您指定自動確定爲 true，JDBC 驅動程式仍可將異動隔離層次改爲 none，因爲系統並無眞正的自動確定模式的概念。雖然提供近似的功能，但並非在所有情況下都能產生正確的結果。問題還沒結束；資料庫會將自動確定的概念與異動隔離層次的概念分開。因此，在啓用自動確定的情況下，以 JDBC_TRANSACTION_SERIALIZABLE 層次來執行完全有效。唯一無效的情況是以 JDBC_TRANSACTION_NONE 層次來執行，但卻不在自動確定模式中。當系統未配合某個異動隔離層次來執行時，應用程式將無法掌控確定界限。

JDBC 規格與 iSeries 平台之間的異動隔離層次

iSeries 平台的異動隔離層次有其通用的名稱，但不符合 JDBC 規格所提供的名稱。下表比對 iSeries 平台所用的名稱，但並不等同於 JDBC 規格所用的名稱：

JDBC 層次*	iSeries 層次
JDBC_TRANSACTION_NONE	*NONE 或 *NC
JDBC_TRANSACTION_READ_UNCOMMITTED	*CHG 或 *UR
JDBC_TRANSACTION_READ_COMMITTED	*CS
JDBC_TRANSACTION_REPEATABLE_READ	*ALL 或 *RS
JDBC_TRANSACTION_SERIALIZABLE	*RR

* 此表格中，爲了清楚說明，JDBC_TRANSACTION_NONE 值與 iSeries 層次 *NONE 及 *NC 排在同一列。這並不表示同一列的 JDBC 規格與 iSeries 層次相符。

儲存點:

儲存點可以在異動中設定「暫置點」。儲存點就是讓應用程式可以回轉至某個狀態的檢查點，如此一來，就不須放棄整個異動。 儲存點首見於 JDBC 3.0，表示應用程式必須在 Java Development Kit (JDK) 1.4 或後續版次上執行，才可以使用儲存點。再者，儲存點在 Developer Kit for Java 中也是首次出現，表示若舊版 Developer Kit for Java 未使用 JDK 1.4 或後續版次，則不支援儲存點。

註: 系統提供 SQL 陳述式以處理儲存點。建議 JDBC 應用程式不要直接在應用程式中使用這些陳述式。若這麼做，即使運作沒問題，JDBC 驅動程式卻將無法追蹤儲存點。至少應該避免兩種模型混合使用 (亦即，避免同時使用自己的儲存點 SQL 陳述式及 JDBC API)。

設定及回轉至儲存點

您可以透過異動的運作來設定儲存點。某個地方出錯時，應用程式即可回轉至這些儲存點，再從儲存點開始繼續處理下去。下列範例中，應用程式會在資料庫表格內插入 FIRST 這個值。接著會設定儲存點，並且在資料庫內插入另一個值 SECOND。 然後發出回轉至儲存點的動作，將插入 SECOND 的工作還原，但使 FIRST 成爲擱置異動的一部份。最後，插入 THIRD 這個值，確定整個異動。此時，資料庫表格包含 FIRST 及 THIRD 這兩個值。

範例：設定及回轉至儲存點

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
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);        // Undoes most recent insert.
s.executeUpdate("insert into table1 values ('THIRD')");
     connection.commit();
```

雖然在自動確定模式中設定儲存點應該不會有問題，但因爲異動結束時，儲存點就會失效，所以無法回轉。

釋放儲存點

應用程式可以使用 Connection 物件的 releaseSavepoint 方法來釋放儲存點。一旦釋放儲存點之後，再試圖回轉就會導致異常。當異動確定或回轉時，就會自動釋放所有儲存點。回轉某個儲存點時，也會一併釋放後續的其他儲存點。

分散式異動

在 Java Database Connectivity (JDBC) 中，異動通常發生在本端。這表示單一連線即可完成整個異動的所有工作，且連線一次只能處理一個異動。當異動的所有工作皆完成或失敗時，就會呼叫確定或回復，讓工作持續進行，好讓新的交易得以開始。然而，Java 中的異動還有更進階的支援，功能超過本端異動。此支援由 Java Transaction API 完全指定。

Java Transaction API (JTA) 支援複雜的異動。亦支援將異動與 Connection 物件分離。JDBC 是仿效 Object Database Connectivity (ODBC) 與「X/Open 呼叫層次介面 (CLI)」規格，JTA 是仿效 X/Open Extended Architecture (XA) 規格。JTA 與 JDBC 可互相合作將異動與 Connection 物件分離。將異動與 Connection 物件分離後，即可讓單一連線同時處理多個異動。相反地，亦可使用多個連線來處理單一異動。

Java Transaction API (JTA) 1.0.1 規格

註: 如果您打算使用 JTA，請參閱 JDBC 入門，以取得延伸套件類別路徑中必要的 Java Archive (JAR) 的相關資訊。您需要 JDBC 2.0 選用性套件及 JTA JAR 檔 (若您執行 JDK 1.4 或後續版本，JDK 會自動尋找這些檔案)。依預設不會去找這些檔案。

使用 JTA 的異動

當 JTA 與 JDBC 一起使用時，兩者之間會進行一連串步驟來達成異動工作。對 XA 的支援須透過 XADataSource 類別來達成。此類別可支援設定連線儲存區作業，方式與其 ConnectionPoolDataSource 最高類別完全相同。

您可以利用 XADataSource 實例來擷取 XAConnection 物件。XAConnection 物件可當作 JDBC Connection 物件與 XAResource 物件的儲存區。設計 XAResource 物件的目的，即爲處理 XA 異動支援。XAResource 可透過異動 ID (XID) 這種物件來處理異動。

XID 是您必須實作的介面。它代表 Java 對 X/Open 異動 ID 的 XID 結構對映。這個物件包含三個元素：

- 廣域異動的格式 ID
- 廣域異動 ID
- 分支限定元

如需此介面的完整資訊，請參閱 JTA 規格。

範例：使用 JTA 來處理異動顯示如何在應用程式中使用 JTA 來處理異動。

使用 UDBXADataSource 對儲存區作業及分散式異動的支援

Java Transaction API 對連線儲存區作業提供直接的支援。UDBXADataSource 是 ConnectionPoolDataSource 的延伸類別，可讓應用程式存取儲存的 XAConnection 物件。因爲 UDBXADataSource 就是一個 ConnectionPoolDataSource，所以 UDBXADataSource 的配置及用法，與使用 DataSource 對物件儲存區作業的支援所說明的相同。

XADataSource 內容

除了 ConnectionPoolDataSource 提供的內容之外，XADataSource 介面還提供下列內容：

設定方法 (資料類型)	值	說明
setLockTimeout (int)	0 或任何正數值	在異動層次上，任何正數值皆爲有效的鎖定逾時 (以秒爲單位)。 鎖定逾時 0 表示異動層次上不強制鎖定逾時值，但其他層次上 (工作或表格) 有可能強制。 預設值爲 0。
setTransactionTimeout (int)	0 或任何正數值	任何正數值皆爲有效的異動逾時 (以秒爲單位)。 異動逾時 0 表示不強制異動逾時值。若異動的作用時間超過逾時值，則標示爲只能回復，此後若再於此異動下嘗試執行工作，就會導致發生異常。 預設值爲 0。

ResultSet 與異動

先前範例顯示區分異動的啓動與結束，除此之外，異動亦可暫停一段時間，然後再回復運作。就異動期間建立的 ResultSet 資源而言，這麼做就會產生許多種情況。

簡單的異動結束

當您結束異動時，此異動之下建立的所有已開啓的 ResultSet，亦隨之自動關閉。當您不再使用 ResultSet 時，建議您明確地予以關閉，以維持最大的平行處理能力。然而，對於異動期間已開啓的任何 ResultSet，若於呼叫 XAResource.end 之後又再次存取，則會導致異常。

有關此行爲的資訊，請參閱範例：結束異動。

暫停及回復

當異動暫停時，不允許存取此異動作用期間所建立的 ResultSet，並會發生異常。然而，當異動回復時，即可再次使用 ResultSet，且維持在異動暫停之前的狀態。

有關此行爲的資訊，請參閱範例：暫停及回復異動。

影響已暫停的 ResultSet

當異動暫停時，無法存取 ResultSet。然而，還是可在另一個異動之下重新處理 Statement 物件來執行工作。因爲 JDBC Statement 物件一次只能有一個 ResultSet (不過，JDBC 3.0 支援一個儲存程序呼叫可以有多個並行 ResultSet，屬於例外)，所以必須關閉已暫停的異動所建立的 ResultSet，才能滿足新異動的要求。實際情形正是如此。

有關此行爲的資訊，請參閱範例：已暫停的 ResultSet。

附註：雖然 JDBC 3.0 允許一個 Statement 可以在一個儲存程序呼叫上同時開啓多個 ResultSet，但它們會被視爲一體，而且，若在新的異動之下重新處理 Statement，ResultSet 將會全部關閉。以單一陳述式而言，不可能讓兩個異動的 ResultSet 同時都在作用中。

多工作業

JTA API 的設計可以將異動與 JDBC 連線分離。此 API 可讓您使用多個連線來處理單一異動，或讓單一連線同時處理多個異動。這稱爲**多工作業**，可以達成 JDBC 無法獨立完成的許多複雜的作業。

這個範例顯示多個連線處理單一異動。

這個範例顯示與多個交易同時發生的單一連線。

有關使用 JTA 的進一步資訊，請參閱 JTA 規格。JDBC 3.0 規格亦說明這兩項技術如何相互搭配來支援分散式異動的資訊。

兩階段確定及異動記載

JTA API 將分散式兩階段確定通訊協定的責任，全部委派給應用程式。如先前範例所示，在 JTA 異動下使用 JTA 及 JDBC 來存取資料庫時，應用程式需使用 XAResource.prepare() 及 XAResource.commit() 方法來確定變更，或僅使用 XAResource.commit() 方法亦可。

此外，使用單一異動來存取多個不同的資料庫時，應用程式必須確實完成兩階段確定通訊協定及這些資料庫之間的異動不可分割性相關記載。通常，多個資料庫 (亦即，XAResource) 之間的兩階段確定處理及記載，會在應用程式伺服器或異動監視器的掌控之下執行，實際上，應用程式本身並不關心這些問題。

例如，應用程式會呼叫一些 commit() 方法，或是處理無誤之後返回。然後，底層的應用程式伺服器或異動監視器，就會開始處理參與單一分散式異動的每一個資料庫 (XAResource)。

在兩階段確定處理期間，應用程式伺服器會大量運用記載功能。也會依序對每一個參與的資料庫 (XAResource) 呼叫 XAResource.prepare() 方法，最後再對每一個參與的資料庫 (XAResource) 呼叫 XAResource.commit() 方法。

若此處理程序期間發生失敗，則應用程式伺服器的異動監視器日誌，可讓應用程式伺服器本身接著使用 JTA API 來回復分散式異動。在應用程式伺服器或異動監視器掌控之下完成的這項回復作業，對於每一個參與的資料庫 (XAResource)，可讓應用程式伺服器將異動恢復到已知狀態。如此一來，在所有參與的資料庫之間，可確保整個分散式異動維持常見的狀態。

範例：使用 JTA 來處理異動:

下列範例說明如何在應用程式中使用 Java Transaction API (JTA) 來處理異動。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... does not exist
            }

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

```
        }
    }
}


/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // For XA transactions, a transaction identifier is required.
        // An implementation of the XID interface is not included with the
        // JDBC driver. See Transactions with JTA for a description of
        // this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Standard JDBC work is performed.
        int count =
          stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is pretty fun.')");

        // When the transaction work has completed, the XA resource must
        // again be notified.
        xaRes.end(xid, XAResource.TMSUCCESS);

        // The transaction represented by the transaction ID is prepared
        // to be committed.
        int rc = xaRes.prepare(xid);

        // The transaction is committed through the XAResource.
        // The JDBC Connection object is not used to commit
        // the transaction when using JTA.
        xaRes.commit(xid, false);


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

```
            }
        }
    }
}
```

範例：多個連線處理一個異動:

以下範例說明如何在單一異動上使用多個連線。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
   }
/**
* Handle the previous cleanup run so that this test can recommence.
*/
   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) {
         // Ignore... does not exist
         }
         s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
                         (50))");
               s.close();
      }
      finally {
         if (c != null) {
            c.close();
         }
      }
   }
/**
* This test uses JTA support to handle transactions.
*/
   public void run() {
      Connection c1 = null;
      Connection c2 = null;
      Connection c3 = null;
      try {
         Context ctx = new InitialContext();
         // Assume the data source is backed by a UDBXADataSource.
         UDBXADataSource ds = (UDBXADataSource)
                              ctx.lookup("XADataSource");
         // From the DataSource, obtain some XAConnection objects that
         // contain an XAResource and a Connection object.
         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();
   // For XA transactions, a transaction identifier is required.
   // Support for creating XIDs is again left to the application
   // program.
   Xid xid = JDXATest.xidFactory();
   // Perform some transactional work under each of the three
   // connections that have been created.
   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);
   // When completed, commit the transaction as a single unit.
   // A prepare() and commit() or 1 phase commit() is required for
   // each separate database (XAResource) that participated in the
   // transaction. Since the resources accessed (xaRes1, xaRes2, and xaRes3)
   // all refer to the same database, only one prepare or commit is required.
   int rc = xaRes.prepare(xid);
   xaRes.commit(xid, false);
}
catch (Exception e) {
   System.out.println("Something has gone wrong.");
   e.printStackTrace();
}
finally {
   try {
      if (c1 != null) {
         c1.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Note: Cleaup exception " +
                         e.getMessage());
   }
   try {
      if (c2 != null) {
         c2.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Note: Cleaup exception " +
                         e.getMessage());
   }
   try {
      if (c3 != null) {
         c3.close();
      }
   }
   catch (SQLException e) {
      System.out.println("Note: Cleaup exception " +
                         e.getMessage());
```

```
        }
      }
   }
}
```

範例：使用具有多個異動的連線:

以下範例說明如何使用單一連線來處理多個異動。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... does not exist
            }

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

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


    /**
     * This test uses JTA support to handle transactions.
     */
    public void run() {
        Connection c = null;

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

```
            // Assume the data source is backed by a UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // From the DataSource, obtain an XAConnection object that
            // contains an XAResource and a Connection object.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();
            Statement stmt = c.createStatement();

            // For XA transactions, a transaction identifier is required.
            // This is not meant to imply that all the XIDs are the same.
            // Each XID must be unique to distinguish the various transactions
            // that occur.
            // Support for creating XIDs is again left to the application
            // program.
            Xid xid1 = JDXATest.xidFactory();
            Xid xid2 = JDXATest.xidFactory();
            Xid xid3 = JDXATest.xidFactory();

            // Do work under three transactions for this connection.
            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);

            // Prepare all the transactions
            int rc1 = xaRes.prepare(xid1);
            int rc2 = xaRes.prepare(xid2);
            int rc3 = xaRes.prepare(xid3);

            // Two of the transactions commit and one rolls back.
            // The attempt to insert the second value into the table is
            // not committed.
            xaRes.commit(xid1, false);
            xaRes.rollback(xid2);
            xaRes.commit(xid3, false);


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

範例：已暫停的 ResultSet:

以下範例說明如何在另一個異動之下重新處理 Statement 物件來執行工作。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... does not exist
            }

            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();
            }
        }
    }


    /**
     * This test uses JTA support to handle transactions.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Assume the data source is backed by a UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // From the DataSource, obtain an XAConnection object that
            // contains an XAResource and a Connection object.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();

            // For XA transactions, a transaction identifier is required.
```

```
            // An implementation of the XID interface is not included with
            // the JDBC driver. See Transactions with JTA
// for a description of this interface to build a
            // class for it.
            Xid xid = new XidImpl();

            // The connection from the XAResource can be used as any other
            // JDBC connection.
            Statement stmt = c.createStatement();

            // The XA resource must be notified before starting any
            // transactional work.
            xaRes.start(xid, XAResource.TMNOFLAGS);

            // Create a ResultSet during JDBC processing and fetch a row.
            ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
            rs.next();

            // The end method is called with the suspend option. The
            // ResultSets associated with the current transaction are 'on hold'.
            // They are neither gone nor accessible in this state.
            xaRes.end(xid, XAResource.TMSUSPEND);


            // In the meantime, other work can be done outside the transaction.
            // The ResultSets under the transaction can be closed if the
            // Statement object used to create them is reused.
            ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
            while (nonXARS.next()) {
                // Process here...
            }


            // Attempt to go back to the suspended transaction. The suspended
            // transaction's ResultSet has disappeared because the statement
            // has been processed again.
            xaRes.start(newXid, XAResource.TMRESUME);
            try {
                rs.next();
            } catch (SQLException ex) {
                System.out.println("This exception is expected. " +
                                   "The ResultSet closed due to another process.");
            }


            // When the transaction had completed, end it
            // and commit any work under it.
            xaRes.end(xid, XAResource.TMNOFLAGS);
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


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

範例：結束異動:

以下爲在應用程式中結束異動的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... does not exist
            }

            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();
            }
        }
    }


    /**
     * This test use JTA support to handle transactions.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Assume the data source is backed by a UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // From the DataSource, obtain an XAConnection object that
```

```
            // contains an XAResource and a Connection object.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();

            // For XA transactions, transaction identifier is required.
            // An implementation of the XID interface is not included
            // with the JDBC driver. See Transactions with JTA for a
            // description of this interface to build a class for it.
            Xid xid = new XidImpl();

            // The connection from the XAResource can be used as any other
            // JDBC connection.
            Statement stmt = c.createStatement();

            // The XA resource must be notified before starting any
            // transactional work.
            xaRes.start(xid, XAResource.TMNOFLAGS);

            // Create a ResultSet during JDBC processing and fetch a row.
            ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
            rs.next();

            // When the end method is called, all ResultSet cursors close.
            // Accessing the ResultSet after this point results in an
            // exception being thrown.
            xaRes.end(xid, XAResource.TMNOFLAGS);

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

            // Commit the transaction to ensure that all locks are
            // released.
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);


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

鏈結集合

程式碼範例免責聲明

使用 JTA 的異動

在 Java Database Connectivity (JDBC) 中，異動通常發生在本端。這表示單一連線即可完成整個異動的所有工作，且連線一次只能處理一個異動。當異動的所有工作皆完成或失敗時，就會呼叫確定或回復，讓工作持續進行，好讓新的交易得以開始。然而，Java 中的異動還有更進階的支援，功能超過本端異動。此支援由 Java Transaction API 完全指定。

範例：暫停及回復異動:

以下爲異動暫停之後又回復的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... doesn't exist
            }

            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();
            }
        }
    }


    /**
     * This test uses JTA support to handle transactions.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Assume the data source is backed by a UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // From the DataSource, obtain an XAConnection object that
```

```
// contains an XAResource and a Connection object.
XAConnection  xaConn = ds.getXAConnection();
XAResource    xaRes  = xaConn.getXAResource();
Connection    c      = xaConn.getConnection();

// For XA transactions, a transaction identifier is required.
// An implementation of the XID interface is not included with
// the JDBC driver. Transactions with JTA for a
// description of this interface to build a class for it.
Xid xid = new XidImpl();

// The connection from the XAResource can be used as any other
// JDBC connection.
Statement stmt = c.createStatement();

// The XA resource must be notified before starting any
// transactional work.
xaRes.start(xid, XAResource.TMNOFLAGS);

// Create a ResultSet during JDBC processing and fetch a row.
ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
rs.next();

// The end method is called with the suspend option. The
// ResultSets associated with the current transaction are 'on hold'.
// They are neither gone nor accessible in this state.
xaRes.end(xid, XAResource.TMSUSPEND);


// Other work can be performed with the transaction.
// As an example, you can create a statement and process a query.
// This work and any other transactional work that the transaction may
// perform is separate from the work done previously under the XID.
Statement nonXAStmt = conn.createStatement();
ResultSet nonXARS = nonXAStmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
while (nonXARS.next()) {
    // Process here...
}
nonXARS.close();
nonXAStmt.close();


// If an attempt is made to use any suspended transactions
// resources, an exception results.
try {
    rs.getString(1);
    System.out.println("Value of the first row is " + rs.getString(1));
} catch (SQLException e) {
    System.out.println("This was an expected exception - " +
                       "suspended ResultSet was used.");
}


// Resume the suspended transaction and complete the work on it.
// The ResultSet is exactly as it was before the suspension.
xaRes.start(newXid, XAResource.TMRESUME);
rs.next();
System.out.println("Value of the second row is " + rs.getString(1));


// When the transaction has completed, end it
// and commit any work under it.
xaRes.end(xid, XAResource.TMNOFLAGS);
int rc = xaRes.prepare(xid);
xaRes.commit(xid, false);
```

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

Statement 類型

Statement 介面及其 PreparedStatement 與 CallableStatement 子類別，可用來處理向資料庫提出的結構化查詢語言 (SQL) 指令。 SQL 陳述式會導致 ResultSet 物件產生。

Connection 介面有許多方法可以建立 Statement 介面的子類別。 單一 Connection 物件底下可以同時建立許多 Statement 物件。在以往的版次中，能夠確切指出可建立的 Statement 物件數目。但在這個版次中就不可能這樣做，因爲不同類型的 Statement 物件在資料庫引擎內各接受不同的控點數目。因此，您使用的 Statement 物件的類型，將影響單一時間在一個連線下可作用的陳述式數目。

應用程式可以呼叫 Statement.close 方法，指出應用程式已處理完畢某個陳述式。當連線關閉時，它所建立的所有 Statement 物件亦隨之關閉。然而，您不應完全仰賴此行爲來關閉 Statement 物件。比方說，如果應用程式改爲使用連線儲存區，不必再明確地關閉連線，應用程式將因爲連線永不關閉而「洩漏」陳述式控點。儘速關閉不再需要的 Statement 物件，可以立即釋放陳述式佔用的外部資料庫資源。

原有的 JDBC 驅動程式會試圖偵測陳述式洩漏，替您妥善處理。然而，依賴這項支援會導致效能退化。

根據 Statement、PreparedStatement 及 CallableStatement 依序延伸而形成的繼承階層，每一個介面的繼承類別，都可以使用該介面的功能。例如，在 PreparedStatement 及 CallableStatement 類別中，亦支援 Statement 類別的功能。 但 Statement 類別的 executeQuery、executeUpdate 及 execute 方法爲例外。 這些方法會動態處理 SQL 陳述式，若試圖同時搭配 PreparedStatement 或 CallableStatement 物件來使用，則會導致異常。

Statement:

Statement 物件可以處理靜態 SQL 陳述式並取得陳述式產生的結果。每一個 Statement 物件每次只能開啓一個 ResultSet。 在所有處理 SQL 陳述式的陳述式方法中，若已有開啓的陳述式 ResultSet，就會隱含地關閉陳述式的現行 ResultSet。

建立陳述式

Connection 物件的 createStatement 方法可以建立 Statement 物件。 例如，假設有一個名爲 conn 的 Connection 物件已存在，以下程式碼行將建立一個 Statement 物件，將 SQL 陳述式傳給資料庫：

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

指定 ResultSet 性質

ResultSet 的性質與最後建立這些 ResultSet 的陳述式有關聯。Connection.createStatement 方法可讓您指定這些 ResultSet 性質。 以下爲有效呼叫 createStatement 方法的一些範例：

範例：createStatement 方法

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// The following is new in JDBC 2.0

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

// The following is new in JDBC 3.0

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

有關這些性質的資訊，請參閱 ResultSet。

處理陳述式

若要使用 Statement 物件來處理 SQL 陳述式，須透過 executeQuery()、executeUpdate() 及 execute() 方法來達成。

從 SQL 查詢傳回結果

若要處理會傳回 ResultSet 物件的 SQL 查詢陳述式，請使用 executeQuery() 方法。請參閱範例程式，此程式使用 Statement 物件的 executeQuery 方法來取得 ResultSet。

附註：若使用 executeQuery 來處理的 SQL 陳述式未傳回 ResultSet，則會丟出 SQLException。

傳回 SQL 陳述式的更新計數

若已知 SQL 是會傳回更新計數的「資料定義語言 (DDL)」陳述式或「資料操作語言 (DML)」陳述式，請使用 executeUpdate() 方法。 StatementExample 程式使用 Statement 物件的 executeUpdate 方法。

處理預期傳回結果不明的 SQL 陳述式

若不知道 SQL 陳述式類型，請使用 execute 方法。處理此方法之後，JDBC 驅動程式就可透過 API 呼叫，向應用程式表示 SQL 陳述式產生何種結果。若結果至少是一個 ResultSet，則 execute 方法會傳回 true，若回覆值爲更新計數，則會傳回 false。取得這項資訊之後，應用程式即可使用陳述式方法的 getUpdateCount 或 getResultSet， 來擷取 SQL 陳述式處理後傳回的值。StatementExecute 程式會在 Statement 物件上使用 execute 方法。 此程式預期傳遞的參數爲 SQL 陳述式。程式並不理會您提供的 SQL 文字，而是直接處理陳述式，然後決定已處理項目的相關資訊。

附註：當結果爲 ResultSet 時，呼叫 getUpdateCount 方法會傳回 -1。當結果爲更新計數時，呼叫 getResultSet 方法會傳回空值。

cancel 方法

原有 JDBC 驅動程式的方法會保持同步，以防止在相同物件上執行的兩個緒造成物件的毀損。但 cancel 方法則屬例外。一個緒可以使用 cancel 方法，藉以停止相同物件的另一個緒上長時間執行的 SQL 陳述式。 原有的 JDBC 驅動程式無法強制某個緒停止運作；只能要求某個緒停止正在進行的作業。因此，遭到取消的陳述式，仍然需要一段時間才能停止。cancel 方法可用來中止系統上失控的 SQL 查詢。

範例：使用 Statement 物件的 executeUpdate 方法:

以下範例將示範如何使用 Statement 物件的 executeUpdate 方法。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

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

public class StatementExample {

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

        // Suggestion:  Load these from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Register the native JDBC driver. If the driver cannot be
        // registered, the test cannot continue.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        try {
            // Create the connection properties.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Connect to the local iSeries database.
            c = DriverManager.getConnection(URL, properties);

            // Create a Statement object.
            s = c.createStatement();
            // Delete the test table if it exists. Note: This
            // example assumes that the collection MYLIBRARY
            // exists on the system.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Just continue... the table probably does not exist.
            }

            // Run an SQL statement that creates a table in the database.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

            // Run some SQL statements that insert records into the table.
            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)");

            // Run an SQL query on the table.
            ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

            // Display all the data in the table.
            while (rs.next()) {
                System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
            }

        } catch (SQLException sqle) {
            System.out.println("Database processing has failed.");
            System.out.println("Reason: " + sqle.getMessage());
        } finally {
            // Close database resources
            try {
```

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

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

PreparedStatements:

PreparedStatement 可延伸 Statement 介面，且支援在 SQL 陳述式內加入參數。

傳遞給資料庫的 SQL 陳述式，會透過兩步驟的程序將結果傳回給您。做好準備之後，隨即進行處理。但透過 Statement 物件，這兩個階段在應用程式中似乎成爲一個階段。 PreparedStatement 可以將這兩個步驟分離。即亦，在建立物件時進行準備步驟，而於 PreparedStatement 物件上呼叫 executeQuery、executeUpdate 或 execute 方法時進行處理步驟。

若不加入參數記號，就算可以將 SQL 處理程序分割成兩個階段，也毫無意義。在應用程式內放入參數記號，就是爲了向資料庫表示準備期間並無特定的值，但在處理之前會提供一個值。 SQL 陳述式以問號代表參數記號。

參數記號可將特定要求使用的 SQL 陳述式一般化。 例如，以下列 SQL 查詢陳述式爲例：

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

此爲特定的 SQL 陳述式，僅傳回一個值；也就是傳回 Dettinger 這位員工的相關資訊。只要加入參數記號，陳述式就會更有彈性：

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

只要設定參數記號的值，即可從表格中取得任何員工的資訊。

PreparedStatement 的效能提升程度高於 Statement， 因爲前述 Statement 範例只會經過一次準備階段，然後就按照不同的參數值來反覆地處理。

註: 爲了支援原有 JDBC 驅動程式的陳述式儲存區作業，必須使用 PreparedStatement。

有關使用備妥陳述式的資訊，包括建立備妥陳述式、指定結果集性質、處理自動產生的鍵值及設定參數記號，請參閱：

建立及使用 PreparedStatement:

prepareStatement 方法可用來建立新的 PreparedStatement 物件。 不同於 createStatement 方法，建立 PreparedStatement 物件時，也必須同時提供 SQL 陳述式。 此時，SQL 陳述式會先完成前置編譯以供使用。

舉例來說，假設有一個名爲 conn 的 Connection 物件已存在， 在下列範例中將建立一個 PreparedStatement 物件，並且備妥要在資料庫內處理的 SQL 陳述式。

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

指定 ResultSet 性質及自動產生鍵值支援

如同 createStatement 方法，prepareStatement 方法在支援指定 ResultSet 性質時，也有超載的情形。prepareStatement 方法亦有各種變體來處理自動產生的鍵值。以下範例說明如何有效呼叫 prepareStatement 方法：

範例：prepareStatement 方法

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// New in JDBC 2.0

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

ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATEABLE);

// New in JDBC 3.0

PreparedStatement ps3 = conn.prepareStatement("SELECT * FROM
    EMPLOYEE_TABLE WHERE LASTNAME = ?",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,
    ResultSet.HOLD_CURSOR_OVER_COMMIT);

PreparedStatement ps4 = conn.prepareStatement("SELECT * FROM
    EMPLOYEE_TABLE WHERE LASTNAME = ?", Statement.RETURN_GENERATED_KEYS);
```

處理參數

必須先設定每一個參數記號的值，才能處理 PreparedStatement 物件。 PreparedStatement 物件提供許多方法來設定參數。 所有方法皆爲 set<Type> 格式，其中 <Type> 爲 Java 資料類型。例如，setInt、setLong、setString、setTimestamp、setNull 及 setBlob，即屬於這些方法。這些方法幾乎都接受兩個參數：

- 第一個參數爲陳述式內的參數索引。參數記號從 1 開始編號。
- 第二個參數爲參數設定的值。還有一些 set<Type> 方法另有額外的參數， 例如 setBinaryStream 的 length 參數。

如需詳細資訊，請參閱 javadoc 的 java.sql 套件。上述範例中已指定備妥 SQL 陳述式給 ps 物件，以下程式碼將說明如何在處理之前指定參數值：

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

若試圖處理尚未設定參數記號的 PreparedStatement，將會丟出 SQLException。

註: 參數記號一旦設定後，會在各程序之間保持相同的值，直到發生下列狀況爲止：

- 又另外呼叫 set 方法來改變參數值。
- 呼叫 clearParameters 方法來移除參數值。

clearParameters 方法會將參數全部標示成未設定。 呼叫 clearParameters 之後，所有參數皆必須再次呼叫 set 方法，才能繼續下一個處理。

ParameterMetaData 支援

有一個新的 ParameterMetaData 介面可讓您擷取參數的相關資訊。 這項支援是 ResultSetMetaData 的補充，兩者很相似。例如，精準度、小數位數、資料類型、資料類型名稱及參數是否允許空值等資訊，全部都有提供。

有關如何在應用程式中採用這項新支援的資訊，請參閱範例：ParameterMetaData。

範例：ParameterMetaData:

以下爲使用 ParameterMetaData 介面來擷取參數相關資訊的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////////////////////////////////////////
//
// ParameterMetaData example. This program demonstrates
// the new support of JDBC 3.0 for learning information
// about parameters to a PreparedStatement.
//
// Command syntax:
//    java PMD
//
//////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class PMD {

    // Program entry point.
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Obtain setup.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.MYTABLE VALUES(?, ?, ?)");
        ParameterMetaData pmd = ps.getParameterMetaData();

        for (int i = 1; i < pmd.getParameterCount(); i++) {
            System.out.println("Parameter number " + i);
            System.out.println("  Class name is " + pmd.getParameterClassName(i));
            // Note: Mode relates to input, output or inout
            System.out.println("  Mode is " + pmd.getParameterClassName(i));
            System.out.println("  Type is " + pmd.getParameterType(i));
            System.out.println("  Type name is " + pmd.getParameterTypeName(i));
            System.out.println("  Precision is " + pmd.getPrecision(i));
            System.out.println("  Scale is " + pmd.getScale(i));
            System.out.println("  Nullable? is " + pmd.isNullable(i));
```

```
            System.out.println("  Signed? is " + pmd.isSigned(i));
        }
    }
}
```

處理 PreparedStatement:

利用 PreparedStatement 物件來處理 SQL 陳述式須透過 executeQuery、executeUpdate 及 execute 方法來達成，就像處理 Statement 物件一樣。 但仍不同於 Statement 版本，這些方法不需傳入參數，因爲建立物件時已經提供 SQL 陳述式。由於 PreparedStatement 可延伸 Statement，所以應用程式仍可呼叫接受 SQL 陳述式的 executeQuery、executeUpdate 及 execute 方法。 只是這樣做會丟出 SQLException。

從 SQL 查詢傳回結果

若要處理的 SQL 查詢陳述式會傳回 ResultSet 物件，請使用 executeQuery 方法。 PreparedStatementExample 程式利用 PreparedStatement 物件的 executeQuery 方法來取得 ResultSet。

註: 若使用 executeQuery 方法來處理的 SQL 陳述式未傳回 ResultSet，則會丟出 SQLException。

傳回 SQL 陳述式的更新計數

若已知 SQL 是會傳回更新計數的「資料定義語言 (DDL)」陳述式或「資料操作語言 (DML)」陳述式，請使用 executeUpdate 方法。 PreparedStatementExample 範例程式使用 PreparedStatement 物件的 executeUpdate 方法。

處理預期傳回結果不明的 SQL 陳述式

若不知道 SQL 陳述式類型，請使用 execute 方法。處理此方法之後，JDBC 應用程式就可透過 API 呼叫，向應用程式表示 SQL 陳述式產生的結果類型。若結果至少是一個 ResultSet，則 execute 方法會傳回 true，若回覆值爲更新計數，則會傳回 false。有了這項資訊，應用程式即可使用 getUpdateCount 或 getResultSet 陳述式方法， 擷取 SQL 陳述式處理後傳回的值。

附註：當結果爲 ResultSet 時，呼叫 getUpdateCount 方法會傳回 -1。當結果爲更新計數時，呼叫 getResultSet 方法會傳回空值。

範例：使用 PreparedStatement 來取得 ResultSet:

以下範例說明如何使用 PreparedStatement 物件的 executeQuery 方法來取得 ResultSet。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

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

public class PreparedStatementExample {

    public static void main(java.lang.String[] args)
    {
        // Load the following from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Register the native JDBC driver. If the driver cannot
        // be registered, the test cannot continue.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
```

```
}

Connection c = null;
Statement s = null;

//    This program creates a table that is
//    used by prepared statements later.
try {
    // Create the connection properties.
    Properties properties = new Properties ();
    properties.put ("user", "userid");
    properties.put ("password", "password");

    // Connect to the local iSeries database.
    c = DriverManager.getConnection(URL, properties);

    // Create a Statement object.
    s = c.createStatement();
    // Delete the test table if it exists.  Note that
    // this example assumes throughout that the collection
    // MYLIBRARY exists on the system.
    try {
        s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
    } catch (SQLException e) {
        // Just continue... the table probably did not exist.
    }

    // Run an SQL statement that creates a table in the database.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

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


//    This program then uses a prepared statement to insert many
//    rows into the database.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Create a PreparedStatement object that is used to insert data into the
    // table.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

    for (int i = 0; i < nameArray.length; i++) {
        ps.setString(1, nameArray[i]);      // Set the Name from our array.
        ps.setInt(2, i+1);                  // Set the ID.
        ps.executeUpdate();
    }

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
```

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


        //    Use a prepared statement to query the database
        //    table that has been created and return data from it. In
        //    this example, the parameter used is arbitrarily set to
        //    5, meaning return all rows where the ID field is less than
        //    or equal to 5.
        try {
            ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
                                    "WHERE ID <= ?");

            ps.setInt(1, 5);

            // Run an SQL query on the table.
            ResultSet rs = ps.executeQuery();
            // Display all the data in the table.
            while (rs.next ()) {
                System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
            }

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

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

        }
    }
}
```

CallableStatements:

CallableStatement 介面是 PreparedStatement 的延伸，可支援輸出及輸入/輸出參數。CallableStatement 介面亦支援 PreparedStatement 介面所提供的輸入參數。

CallableStatement 介面可讓 SQL 陳述式呼叫儲存程序。儲存程序是具有資料庫介面的程式。這些程式有下列特性：

- 具有輸入及輸出參數，或同時爲輸入/輸出的參數。
- 具有回覆值。
- 可傳回多重 ResultSet。

在觀念上，JDBC 的儲存程序呼叫是對資料庫的單一呼叫，但與儲存程序相關的程式也許可以處理上百個資料庫要求。儲存程序程式也能夠執行 SQL 陳述式通常無法完成的其他許多程式化作業。

因爲 CallableStatement 遵循 PreparedStatement 的模式來隔離準備階段與處理階段，所以也有能力達到重覆使用的最佳化效果 (如需詳細資訊，請參閱 PreparedStatement)。由於儲存程序的 SQL 陳述式結合於程式中，因此會以靜態 SQL 來處理，而得以進一步發揮效能。將大量資料庫工作封裝成可重覆使用的單一資料庫呼叫，就是儲存程序最佳用法的例子。雖然只有這個呼叫透過網路流向其他系統，但此要求足以在遠端系統上完成許多工作。

建立 CallableStatement

prepareCall 方法可用來建立新的 CallableStatement 物件。如同 prepareStatement 方法，建立 CallableStatement 物件時，也必須提供 SQL 陳述式。屆時 SQL 陳述式會進行前置編譯。例如，假設有一個名爲 conn 的 Connection 物件已存在，以下程式碼將建立一個 CallableStatement 物件，並且完成準備階段，讓 SQL 陳述式準備好開始於資料庫內進行處理：

```
PreparedStatement ps = conn.prepareStatement("? = CALL ADDEMPLOYEE(?, ?, ?");
```

ADDEMPLOYEE 儲存程序可接受新進員工的輸入參數，包括姓名、社會保險號碼及主管的使用者 ID。可從這項資訊更新多個公司資料庫表格內的員工資訊，例如到職日、科別、部門等。另外，儲存程序是一種可針對員工產生標準使用者 ID 及電子郵件地址的程式。儲存程序亦可利用起始使用者名稱與密碼傳送電子郵件給人事經理；人事經理就可以提供資訊給員工。

ADDEMPLOYEE 儲存程序設定爲有回覆值。回覆碼可能是成功碼或失敗碼，供呼叫程式於失敗發生時使用。回覆值亦可定義爲新進員工的公司 ID 號碼。也就是說，儲存程序程式在內部處理查詢，並開啓這些查詢的 ResultSet，供呼叫程式使用。先查詢所有新員工的資訊，再透過傳回的 ResultSet 提供給呼叫程式使用，是比較合理的做法。

下列各節說明如何完成各種類型的作業。

指定 ResultSet 性質及自動產生鍵值支援

如同 createStatement 與 prepareStatement，prepareCall 有多種版本可支援指定 ResultSet 性質。但不同於 prepareStatement，prepareCall 方法不提供變體來使用 CallableStatement 自動產生的鍵值 (JDBC 3.0 不支援此概念)。以下爲有效呼叫 prepareCall 方法的一些範例：

範例：prepareCall 方法

```
// The following is new in JDBC 2.0

CallableStatement cs2 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE);

// New in JDBC 3.0

CallableStatement cs3 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,
    ResultSet.HOLD_CURSOR_OVER_COMMIT);
```

處理參數

如本文開頭所述，CallableStatement 物件可接受三種參數：

- **IN**

  IN 參數的處理方式同於 PreparedStatement。利用繼承的 PreparedStatement 類別中各種 set 方法來設定參數。

- **OUT**

  OUT 參數以 registerOutParameter 方法來處理。常見的 registerOutParameter 形式以索引參數爲第一個參數，以 SQL 類型爲第二個參數。在處理陳述式時，JDBC 驅動程式就依此來預期參數的資料。registerOutParameter 方法還有其他兩種變體，請參閱 java.sql 套件 Javadoc。

- **INOUT**

  INOUT 參數需要執行 IN 參數與 OUT 參數的工作。對於每一個 INOUT 參數，您必須先呼叫 set 方法與 registerOutParameter 方法，才能夠處理陳述式。若無法設定或登記任何參數，則在處理陳述式時，將會丟出 SQLException。

如需詳細資訊，請參閱範例：建立含有輸入及輸出參數的程序。

如同 PreparedStatement，處理程序之間會維持相同的 CallableStatement 參數值，除非您再次呼叫 set 方法。clearParameters 方法不影響已登記爲輸出的參數。呼叫 clearParameters 之後，必須重新設定所有 IN 參數的值，但不必重新登記所有 OUT 參數。

註: 請勿混淆參數與參數記號索引兩者的概念。儲存程序呼叫包含特定數量的傳入參數。特定的 SQL 陳述式內有 ? 字元 (參數記號)，代表執行時間會提供的值。請參考下列範例來釐清這兩種概念的差別：

```
CallableStatement cs = con.prepareCall("CALL PROC(?, "SECOND", ?)");

cs.setString(1, "First");     //Parameter marker 1, Stored procedure parm 1

cs.setString(2, "Third");     //Parameter marker 2, Stored procedure parm 3
```

依名稱存取儲存程序參數

儲存程序的參數有其相關的名稱，如下列儲存程序宣告所示：

範例：儲存程序參數

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
CREATE
PROCEDURE MYLIBRARY.APROC
    (IN PARM1 INTEGER)
LANGUAGE SQL SPECIFIC MYLIBRARY.APROC
BODY: BEGIN
    <Perform a task here...>
END BODY
```

其中有一個整數參數，名稱是 PARM1。JDBC 3.0 支援依名稱及索引來指定儲存程序參數。爲此程序設定 CallableStatement 的程式碼如下：

```
CallableStatement cs = con.prepareCall("CALL APROC(?)");

cs.setString("PARM1", 6);     //Sets input parameter at index 1 (PARM1) to 6.
```

處理 CallableStatement:

使用 CallableStatement 物件來處理 SQL 儲存程序呼叫的方法，與使用 PreparedStatement 物件來處理時相同。

傳回儲存程序的結果

若於儲存程序內處理 SQL 查詢陳述式，則查詢結果可供呼叫儲存程序的程式使用。亦可於儲存程序內呼叫多重查詢，而呼叫程式可以處理所有可用的 ResultSet。

如需詳細資訊，請參閱範例：建立含有多重 ResultSet 的程序。

註: 若使用 executeQuery 處理儲存程序，但未傳回 ResultSet，則會丟出 SQLException。

同時存取 ResultSet

傳回儲存程序的結果說明有關 ResultSet 與儲存程序等議題，內含適用於所有 Java Development Kit (JDK) 版次的範例。此範例從儲存程序開啓的第一個 ResultSet 至最後一個開啓的 ResultSet，依序處理 ResultSet。一個 ResultSet 關閉後，才會使用下一個 ResultSet。

JDK 1.4 及後續版本支援從儲存程序同時使用 ResultSet。

註: 在 V5R2 中，此功能已透過「指令行介面 (CLI)」加入基礎的系統支援內。因此，在 V5R2 以前的系統上執行的 JDK 1.4 或後續版本沒有這項支援。

傳回儲存程序的更新計數

傳回儲存程序的更新計數是 JDBC 規格中討論的功能，但目前 iSeries 平台並不支援。無法從儲存程序呼叫傳回多個更新計數。如需從儲存程序中已處理的 SQL 陳述式傳回更新計數，有兩種方法可以傳回此值：

- 將值當作輸出參數傳回。
- 將值當作參數的回覆值傳回。此爲輸出參數的特殊情況。如需相關資訊，請參閱「處理有回覆值的儲存程序」。

處理預期傳回結果不明的儲存程序

若儲存程序呼叫所產生的結果不明，則應該使用 execute 方法。處理此方法之後，JDBC 應用程式就可透過 API 呼叫，向應用程式表示儲存程序產生何種結果。若結果是一或多個 ResultSet，則 execute 方法會傳回 true。儲存程序呼叫不會傳回更新計數。

處理有回覆值的儲存程序

iSeries 平台支援儲存程序可具有類似函數回覆值的回覆值。儲存程序回覆值的標示方式，就像其他參數標示一樣，因此會由儲存程序呼叫指派。範例如下：

```
? = CALL MYPROC(?, ?, ?)
```

儲存程序呼叫的回覆值一律爲整數類型，且必須像其他任何輸出參數一樣登記。

如需詳細資訊，請參閱範例：建立含有回覆值的程序。

範例：建立含有多重 ResultSet 的程序:

本範例顯示如何存取資料庫，然後建立含有多重 ResultSet 的程序。

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

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

public class CallableStatementExample1 {

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

        // Register the Native JDBC driver.  If we cannot
        // register the driver, the test cannot continue.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

```
// Create the connection properties
Properties properties = new Properties ();
properties.put ("user", "userid");
properties.put ("password", "password");

// Connect to the local iSeries database
Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

Statement s = c.createStatement();

// Create a procedure with multiple 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) {
    // NOTE:  We are ignoring the error here.  We are making
    //        the assumption that the only reason this fails
    //        is because the procedure already exists. Other
    //        reasons that it could fail are because the C compiler
    //        is not found to compile the procedure or because
    //        collection MYLIBRARY does not exist on the system.
}
s.close();

// Now use JDBC to run the procedure and get the results back. In
// this case we are going to get information about 'MYLIBRARY's stored
// procedures (which is also where we created this procedure, thereby
// ensuring that there is something to get.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX1");

ResultSet rs = cs.executeQuery();

// We now have the first ResultSet object that the stored procedure
// left open.  Use it.
int i = 1;
while (rs.next()) {
    System.out.println("MYLIBRARY stored procedure
                       " + i + " is " + rs.getString(1) + "." +
                       rs.getString(2));
    i++;

}
System.out.println("");


// Now get the next ResultSet object from the system - the previous
// one is automatically closed.
if (!cs.getMoreResults()) {
    System.out.println("Something went wrong.  There should have
                        been another ResultSet, exiting.");
    System.exit(0);
}
rs = cs.getResultSet();

 // We now have the second ResultSet object that the stored procedure
```

```
            // left open.  Use that one.
            i = 1;
            while (rs.next()) {
                System.out.println("MYLIBRARY procedure " + rs.getString(1)
                                   + "." + rs.getString(2) +
                                  " parameter:  " + rs.getInt(3) + " direction:
                                  " + rs.getString(4) +
                                  " data type: " + rs.getString(5));
                i++;

            }

            if (i == 1) {
                System.out.println("None of the stored procedures have any parameters.");
            }


            if (cs.getMoreResults()) {
                System.out.println("Something went wrong,
                                    there should not be another ResultSet.");
                System.exit(0);
            }

            cs.close();  // close the CallableStatement object
            c.close();   // close the Connection object.

        } catch (Exception e) {
            System.out.println("Something failed..");
            System.out.println("Reason: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

範例：建立含有輸入及輸出參數的程序:

本範例顯示如何存取資料庫，然後建立含有輸入及輸出參數的程序。

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

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

public class CallableStatementExample2 {

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

        // Register the Native JDBC driver.  If we cannot
        // register the driver, the test cannot continue.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Create the connection properties
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Connect to the local iSeries database
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Create a procedure with in, out, and in/out parameters.
            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) {
                // NOTE:  We are ignoring the error here.  We are making
                //        the assumption that the only reason this fails
                //        is because the procedure already exists.  Other
                //        reasons that it could fail are because the C compiler
                //        is not found to compile the procedure or because
                //        collection MYLIBRARY does not exist on the system.
            }
            s.close();

            // Prepare a callable statement used to run the procedure.
            CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX2(?, ?, ?)");

            // All input parameters must be set and all output parameters must
            // be registered.  Notice that this means we have two calls to make
            // for an input output parameter.
            cs.setInt(1, 5);
            cs.setInt(3, 10);
            cs.registerOutParameter(2, Types.INTEGER);
            cs.registerOutParameter(3, Types.INTEGER);

            // Run the procedure
            cs.executeUpdate();

            // Verify the output parameters have the desired values.
            System.out.println("The value of P2 should be P1 (5) + 1 = 6.  -->  " + cs.getInt(2));
            System.out.println("The value of P3 should be P3 (10) + 1 = 11.  -->  " + cs.getInt(3));

            cs.close();  // close the CallableStatement object
            c.close();   // close the Connection object.

        } catch (Exception e) {
            System.out.println("Something failed..");
            System.out.println("Reason: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

範例：建立含有回覆值的程序：

本範例顯示如何存取資料庫，然後建立含有回覆值的程序。

註： 請閱讀程式碼範例免責聲明中的重要法律資訊。

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

public class CallableStatementExample3 {

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

        // Register the native JDBC driver. If the driver cannot
        // be registered, the test cannot continue.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Create the connection properties
            Properties properties = new Properties ();
```

```
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Connect to the local iSeries database
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Create a procedure with a return value.
            String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX3 " +
                         " LANGUAGE SQL SPECIFIC MYLIBRARY.SQLSPEX3 " +
                         " EX3: BEGIN " +
                         "      RETURN 1976; " +
                         " END EX3 ";

            try {
                s.executeUpdate(sql);
            } catch (SQLException e) {
                // NOTE:  The error is ignored here. The assumptions is
                //        made that the only reason this fails is
                //        because the procedure already exists. Other
                //        reasons that it could fail are because the C compiler
                //        is not found to compile the procedure or because
                //        collection MYLIBRARY does not exist on the system.
            }
            s.close();

            // Prepare a callable statement used to run the procedure.
            CallableStatement cs = c.prepareCall("? = CALL MYLIBRARY.SQLSPEX3");

            // You still need to register the output parameter.
            cs.registerOutParameter(1, Types.INTEGER);

            // Run the procedure.
            cs.executeUpdate();

            // Show that the correct value is returned.
            System.out.println("The return value
                                should always be 1976 for this example:
                                -->  " + cs.getInt(1));

            cs.close();  // close the CallableStatement object
            c.close();   // close the Connection object.

        } catch (Exception e) {
            System.out.println("Something failed..");
            System.out.println("Reason: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

ResultSet

ResultSet 介面可存取查詢所產生的結果。 在概念上，ResultSet 的資料可視爲一個表格，內含一定的欄位數及列數。 依預設，表格列是按順序來擷取。在一列中，可依任何次序來存取直欄值。

ResultSet 性質:

本主題討論 ResultSet 性質，例如，ResultSet 類型，並行處理能力，以及藉由確定連線物件及 ResultSet 性質的規格來關閉 ResultSet 的功能。

依預設，所有建立的 ResultSet 皆爲 forward only 類型，具有唯讀並行處理能力，且可以跨越確定界限保留游標。但有例外，WebSphere 目前已改變預設的游標保留能力，所以確定時會隱含地關閉游標。這些性質皆可透過 Statement、PreparedStatement 及 CallableStatement 物件的方法來配置。

ResultSet 類型

ResultSet 類型可指定 ResultSet 的相關資訊：

- ResultSet 是否可捲動。
- ResultSet 介面上的常數所定義的 Java Database Connectivity (JDBC) ResultSet 類型。

這些 ResultSet 類型的定義如下：

TYPE_FORWARD_ONLY
只能從 ResultSet 開頭一直處理到尾端的游標。此爲預設類型。

TYPE_SCROLL_INSENSITIVE
能夠以各種方式捲動 ResultSet 的游標。這種游標在開啓時並無法感應到資料庫發生的變更。在處理查詢或提取資料之後，此游標就含有滿足查詢條件的列。

TYPE_SCROLL_SENSITIVE
能夠以各種方式捲動 ResultSet 的游標。這種游標可以感應到資料庫同時間發生的變更。資料庫的變更會直接影響 ResultSet 資料。

JDBC 1.0 ResultSet 一律爲 forward only。JDBC 2.0 已新增可捲動的游標。

附註：blocking enabled 及 block size 連線內容會影響 TYPE_SCROLL_SENSITIVE 游標的靈敏度。區塊傳輸會在 JDBC 驅動程式層本身快取資料，因此可以加強效能。

請參閱範例：靈敏與不靈敏的 ResultSet，其中顯示當表格內插入橫列時，靈敏與不靈敏的 ResultSet 兩者有何差異。

請參閱範例：ResultSet 靈敏度， 其中依據 ResultSet 的靈敏度，顯示某項變更如何影響 SQL 陳述式的 where 子句。

並行處理能力

並行處理能力會決定 ResultSet 是否接受更新。 同樣以 ResultSet 介面的常數來定義類型。可用的並行處理設定值如下：

CONCUR_READ_ONLY
只能用來從資料庫讀出資料的 ResultSet。此爲預設值。

CONCUR_UPDATEABLE
可加以變更的 ResultSet。這些變更可以放入基礎資料庫內。如需詳細資訊，請參閱變更 ResultSet。

JDBC 1.0 ResultSet 一律爲 forward only。JDBC 2.0 已新增可更新的 ResultSet。

註: 根據 JDBC 規格，若這些值無法一起使用，JDBC 驅動程式可以變更 ResultSet 類型的 ResultSet 並行處理設定值。在這些情況下，JDBC 驅動程式將對 Connection 物件發出警告。

在一種情況下，應用程式會指定 TYPE_SCROLL_INSENSITIVE、CONCUR_UPDATEABLE ResultSet。資料庫引擎會複製一份資料來達成無靈敏度的操作方式。您將無法透過此複本來更新基礎資料庫。若指定這樣的組合，驅動程式會將靈敏度變更爲 TYPE_SCROLL_SENSITIVE，並且發出警告表示已變更您的要求。

保留能力

保留性質可決定在 Connection 物件上呼叫 commit 是否會關閉 ResultSet。處理保留性質的 JDBC API 首見於 3.0 版。 然而，原有的 JDBC 驅動程式已在數個版次中提供連線內容，可讓您對連線上建立的所有 ResultSet 指定預設值 (請參閱Connection 內容及 DataSource 內容)。 API 支援會置換連線內容的任何設定。 保留性質的值由 ResultSet 常數定義如下：

HOLD_CURSOR_OVER_COMMIT
呼叫 commit 子句時，所有已開啓的游標仍然維持開啓狀態。此爲原有的 JDBC 預設值。

CLOSE_CURSORS_ON_COMMIT
呼叫 commit 子句時，即關閉所有已開啓的游標。

註: 在連線上呼叫 rollback 時，一律會關閉所有開啓的游標。知道的人不多，但這卻是資料庫處理游標的常用方法。

根據 JDBC 規格，游標保留能力的預設值由實作方式決定。有些平台選擇採用 CLOSE_CURSORS_ON_COMMIT 當作預設值。對大多數應用程式而言，這應該不會造成問題，但若您使用的游標會跨越確定界限，則必須注意驅動程式的運作方式。 IBM Toolbox for Java JDBC 驅動程式亦採用 HOLD_CURSORS_ON_COMMIT 預設值，但 UDB for Windows NT® 的 JDBC 驅動程式則採用 CLOSE_CURSORS_ON_COMMIT 作爲預設值。

指定 ResultSet 性質

一旦建立 ResultSet 物件之後，ResultSet 的性質就不再改變。因此，在建立物件之前就已指定性質。 您可以透過 createStatement、prepareStatement 及 prepareCall 方法的超載變量來指定這些性質。

請參閱下列主題來指定 ResultSet 性質：

- Statement：指定 ResultSet 性質
- PreparedStatement：指定 ResultSet 性質及自動產生鍵值支援
- CallableStatement：指定 ResultSet 性質及自動產生鍵值支援

註: 有 ResultSet 方法可以取得 ResultSet 類型及 ResultSet 的並行處理能力，但沒有方法可以取得 ResultSet 的保留能力。

範例：靈敏與不靈敏的 ResultSet:

以下範例顯示當表格內插入橫列時，靈敏與不靈敏的 ResultSet 兩者有何差異。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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) {
            // Ignored.
        }

        s.executeUpdate("create table cujosql.sensitive(col1 int)");
        s.executeUpdate("insert into cujosql.sensitive values(1)");
        s.executeUpdate("insert into cujosql.sensitive values(2)");
        s.executeUpdate("insert into cujosql.sensitive values(3)");
        s.executeUpdate("insert into cujosql.sensitive values(4)");
        s.executeUpdate("insert into cujosql.sensitive values(5)");
        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {
        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

        // Fetch the five values that are there.
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        System.out.println("fetched the five rows...");

        // Note:  If you fetch the last row, the ResultSet looks
```

```
            //        closed and subsequent new rows that are added
            //        are not be recognized.

            // Allow another statement to insert a new value.
            Statement s2 = connection.createStatement();
            s2.executeUpdate("insert into cujosql.sensitive values(6)");
            s2.close();

            // Whether a row is recognized is based on the sensitivity setting.
            if (rs.next()) {
                System.out.println("There is a row now: " + rs.getInt(1));
            } else {
                System.out.println("No more rows.");
            }


        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("-----------------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

範例：ResultSet 靈敏度:

以下範例說明隨著 ResultSet 靈敏度的不同，變更將如何影響 SQL 陳述式的 where 子句。

爲了使此範例符合列印頁，此範例中的某些格式可能不正確。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;

public class Sensitive2 {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive2 test = new Sensitive2();

        test.setup();
        test.run("sensitive");
        test.cleanup();
```

```
    test.setup();
    test.run("insensitive");
    test.cleanup();
}


public void setup() {

    try {
        System.out.println("Native JDBC used");
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        connection = DriverManager.getConnection("jdbc:db2:*local");

        Statement s = connection.createStatement();
        try {
            s.executeUpdate("drop table cujosql.sensitive");
        } catch (SQLException e) {
            // Ignored.
        }

        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) {
            // Ignored.
        }

        s.executeUpdate("create table cujosql.sensitive2(col2 int)");
        s.executeUpdate("insert into cujosql.sensitive2 values(1)");
        s.executeUpdate("insert into cujosql.sensitive2 values(2)");
        s.executeUpdate("insert into cujosql.sensitive2 values(3)");
        s.executeUpdate("insert into cujosql.sensitive2 values(4)");
        s.executeUpdate("insert into cujosql.sensitive2 values(5)");

        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
```

```
                System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
                s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
            }

            ResultSet rs = s.executeQuery("select col1, col2 From cujosql.sensitive,
                                          cujosql.sensitive2 where col1 = col2");

            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));
            rs.next();
            System.out.println("value is " + rs.getInt(1));

            System.out.println("fetched the four rows...");

            // Another statement creates a value that does not fit the where clause.
            Statement s2 =
              connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
              ResultSet.CONCUR_UPDATEABLE);
            ResultSet rs2 = s2.executeQuery("select *
            from cujosql.sensitive where col1 = 5 FOR UPDATE");
            rs2.next();
            rs2.updateInt(1, -1);
            rs2.updateRow();
            s2.close();

            if (rs.next()) {
                System.out.println("There is still a row: " + rs.getInt(1));
            } else {
                System.out.println("No more rows.");
            }

        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("----------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

游標移動:

iSeries Java Database Connectivity (JDBC) 驅動程式支援可捲動的 ResultSet。在可捲動的 ResultSet 中，可以利用許多游標定位方法，以任何順序來處理資料列。

ResultSet.next 方法可以一次一列地捲動 ResultSet。 在 Java Database Connectivity (JDBC) 2.0 中，iSeries JDBC 驅動程式支援可捲動的 ResultSet。可捲動的 ResultSet 可過透 previous、absolute、relative、first 及 last 方法，依任何順序來處理資料列。

依預設，JDBC ResultSet 一律爲 forward only，表示 next() 是唯一可呼叫的有效游標定位方法。我們必須明確地要求可捲動的 ResultSet。如需詳細資訊，請參閱 ResultSet 類型。

在可捲動的 ResultSet 中，可以使用下列游標定位方法：

方法	說明
Next	在 ResultSet 中，此方法將游標向前移動一列。 若游標定位在有效的列，此方法傳回會 true，否則傳回 false。
Previous	在 ResultSet 中，此方法將游標往回移動一列。 若游標定位在有效的列，此方法傳回會 true，否則傳回 false。
First	在 ResultSet 中，此方法將游標移至第一列。 若游標定位在第一列，此方法傳回會 true，若 ResultSet 是空的，則傳回 false。
Last	在 ResultSet 中，此方法將游標移至最後一列。 若游標定位在最後一列，此方法傳回會 true，若 ResultSet 是空的，則傳回 false。
BeforeFirst	在 ResultSet 中，此方法將游標移至第一列之前。 在空的 ResultSet 中，此方法沒有效果。此方法沒有回覆值。
AfterLast	在 ResultSet 中，此方法將游標移至最後一列之後。 在空的 ResultSet 中，此方法沒有效果。此方法沒有回覆值。
Relative (int rows)	此方法可依現行位置將游標移至相對位置。 • 若 rows 爲 0，此方法沒有效果。 • 若 rows 爲正數，游標會向前移動這麼多列。若現行位置與 ResultSet 尾端之間的列數小於輸入參數所指定的列數，此方法的效果就如同 afterLast。 • 若 rows 爲負數，游標會往回移動這麼多列。若現行位置與 ResultSet 尾端之間的列數小於輸入參數所指定的列數，此方法的效果就如同 beforeFirst。 若游標定位在有效的列，此方法傳回會 true，否則傳回 false。
Absolute (int row)	此方法將游標移至 row 值所指定的列。 若 row 值爲正數，游標會定位在 ResultSet 開頭之後的這麼多列。 第一列編號 1、第二列編號 2，依此類推。若 ResultSet 的列數小於 row 值所指定的列數，此方法的效果就如同 afterLast。 若 row 值爲負數，游標會定位在 ResultSet 尾端之前的這麼多列。 最後一列編號 -1、倒數第二列編號 -2，依此類推。若 ResultSet 的列數小於 row 值所指定的列數，此方法的效果就如同 beforeFirst。 若 row 值爲 0，此方法的效果就同於 beforeFirst。 若游標定位在有效的列，此方法傳回會 true，否則傳回 false。

擷取 ResultSet 資料:

ResultSet 物件提供數個方法可以取得一列的直欄資料。 所有方法皆爲 get<*Type*> 格式，其中 <*Type*> 爲 Java 資料類型。例如，這些方法包括 getInt、getLong、getString、getTimestamp 及 getBlob。 幾乎所有方法都接受單一參數，此參數可能是 ResultSet 內的直欄索引或直欄名稱。

ResultSet 直欄從 1 開始編號。若使用直欄名稱，但 ResultSet 內有多個同名的直欄，則傳回其中第一個。 部份 get<Type> 方法還有其他參數，例如選用的 Calendar 物件，可以傳遞至 getTime、getDate 及 getTimestamp。如需完整資料，請參閱 Javadoc 的 java.sql 套件。

以傳回物件的 get 方法而言，若 ResultSet 中的直欄爲空值，則回覆值爲空值。以初始類型而言，則無法傳回空值。在這種情況下，值爲 0 或 false。 若應用程式必須區別 null、0 或 false，可在呼叫之後立刻使用 wasNull 方法。此方法可以判斷值爲實際的 0 或 false 值， 或是由於 ResultSet 值確實爲空值，所以才傳回此值。

如需如何使用 ResultSet 介面的範例，請參閱範例：IBM Developer Kit for Java 的 ResultSet 介面。

ResultSetMetaData 支援

在 ResultSet 物件上呼叫 getMetaData 方法時， 此方法會傳回 ResultSetMetaData 物件來說明 ResultSet 物件的直欄。若要等到執行時間才會得知所處理的 SQL 陳述式，可以使用 ResultSetMetaData 來決定應該使用什麼 get 方法來擷取資料。 下列程式碼範例使用 ResultSetMetaData 來判斷結果集的每一個直欄類型：

範例：使用 ResultSetMetaData 來判斷結果集的每一個直欄類型。

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
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);
```

如需如何使用 ResultSetMetaData 介面的範例，請參閱『範例：IBM Developer Kit for Java 的 ResultSetMetaData 介面』。

範例：IBM Developer Kit for Java 的 ResultSetMetaData 介面:

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
import java.sql.*;

/**
ResultSetMetaDataExample.java

This program demonstrates using a ResultSetMetaData and
a ResultSet to display all the metadata about a ResultSet
created querying a table.  The user passes the value for the
table and library in.
**/
public class ResultSetMetaDataExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage:  java ResultSetMetaDataExample <library> <table>");
            System.out.println("where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
```

```
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Get a database connection and prepare a statement.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
            int rowCount = 0;
        for (int i = 1; i <= colCount; i++) {
                System.out.println("Information about column " + i);
                System.out.println("   Name.........: " + rsmd.getColumnName(i));
                System.out.println("   Data Type.....: " + rsmd.getColumnType(i) +
                                   " ( " + rsmd.getColumnTypeName(i) + " )");
                System.out.println("   Precision.....: " + rsmd.getPrecision(i));
                System.out.println("   Scale.........: " + rsmd.getScale(i));
                System.out.print  ("   Allows Nulls..: ");
                if (rsmd.isNullable(i)==0)
                    System.out.println("false");
                else
                    System.out.println("true");
            }

        } catch (Exception e) {
            // Handle any errors.
            System.out.println("Oops... we have an error... ");
            e.printStackTrace();
        } finally {
            // Ensure we always clean up.  If the connection gets closed, the
            // statement under it closes as well.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Critical error - cannot close connection object");
                }
            }
        }
    }
}
```

鏈結集合

程式碼範例免責聲明

變更 ResultSet:

透過 iSeries JDBC 驅動程式，可以執行數個作業來變更 ResultSet。

ResultSet 的預設值是唯讀。然而，在 Java Database Connectivity (JDBC) 2.0 中，iSeries JDBC 驅動程式對可更新的 ResultSet 提供了完整的支援。

關於如何更新 ResultSet，請參閱 ResultSet 並行處理能力。

更新橫列

透過 ResultSet 介面，可更新資料庫表格的列。此程序包含下列步驟：

1. 利用各種 update<*Type*> 方法來變更特定橫列的值，其中 <*Type*> 爲 Java 資料類型。這些 update<*Type*> 方法對應於可用來擷取值的 get<*Type*> 方法。
2. 將列套用至基礎資料庫。

在完成第二個步驟之前，資料庫本身不會更新。在 ResultSet 中更新直欄之後，若未呼叫 updateRow 方法，資料庫一樣不會有任何變更。

使用 cancelUpdates 方法，可取消原本打算對某一列的更新。 一旦呼叫 updateRow 方法，資料庫的變更就正式底定，無法還原。

附註：rowUpdated 方法將一律傳回 false，因爲資料庫無從得知哪幾列已經更新。同樣地，updatesAreDetected 方法也會傳回 false。

刪除橫列

透過 ResultSet 介面，可刪除資料庫表格的列。介面中的 deleteRow 方法可以刪除現行列。

插入橫列

透過 ResultSet 介面，可在資料庫表格內插入列。此程序採用「插入列」，應用程式會特地將游標移至此處，然後建置需要在資料庫內插入的值。此程序包含下列步驟：

1. 將游標移至插入列。
2. 爲新列的各直欄設定每一個值。
3. 將此列插入資料庫內，另外可選擇將游標移回 ResultSet 的現行列。

註: 在表格內，新的橫列不會插入游標所在處。通常新增至表格資料空間的尾端。依預設，關聯式資料庫沒有位置依存傾向。例如，若您將游標移至第三列，然後插入資料，當後續使用者提取資料時，這些資料並非一定會出現在第四列之前。

定位更新的支援

除了透過 ResultSet 來更新資料庫的方法以外，還可以使用 SQL 陳述式來發出定位更新。這項支援仰賴於使用具名游標。JDBC 在 Statement 中提供 setCursorName 方法，在 ResultSet 中提供 getCursorName 方法，兩者可以用來存取這些值。

supportsPositionedUpdated 及 supportsPositionedDelete 這兩個 DatabaseMetaData 方法都會傳回 true， 因爲原有的 JDBC 驅動程式支援這項特性。

如需詳細資訊，請參照範例：在一個陳述式中透過另一個陳述式的游標來變更值。

如需詳細資訊，請參照範例：透過另一個陳述式的游標來移除表格的值。

範例：透過另一個陳述式的游標來移除表格的值:

以下範例說明如何透過另一個陳述式的游標來移除表格的值。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }



/**
Handle all the required setup work.
**/
    public void setup() {
        try {
            // Register the JDBC driver.
            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) {
                // Ignore problems here.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
In this section, all the code to perform the testing should
be added. If only one connection to the database is needed,
the global variable 'connection' can be used.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Update each value using next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement
                                      ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
                                      rs.getCursorName ());


            // Loop through the ResultSet and update every other entry.
```

```
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Clean up the resources after they have been used.
            rs.close ();
            stmt2.close ();


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



/**
In this section, put all clean-up work for testing.
**/
    public void cleanup() {
        try {
            // Close the global connection opened in setup().
            connection.close();

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



/**
Display the contents of the table.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

鏈結集合

程式碼範例免責聲明

範例：在一個陳述式中透過另一個陳述式的游標來變更值:

以下範例說明如何在一個陳述式中透過另一個陳述式的游標來變更值。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }



/**
Handle all the required setup work.
**/
    public void setup() {
        try {
            // Register the JDBC driver.
            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) {
                // Ignore problems here.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
In this section, all the code to perform the testing should
be added. If only one connection to the database is required,
the global variable 'connection' can be used.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Update each value using next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());
```

```
            PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
                                                                   + " CUJOSQL.WHERECUREX
                                                                       SET COL_VALUE = 'CHANGED'
                                                                       WHERE CURRENT OF "
                                                                   + rs.getCursorName ());


            // Loop through the ResultSet and update every other entry.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Clean up the resources after they have been used.
            rs.close ();
            stmt2.close ();


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



/**
In this section, put all clean-up work for testing.
**/
    public void cleanup() {
        try {
            // Close the global connection opened in setup().
            connection.close();

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



/**
Display the contents of the table.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

程式碼範例免責聲明

建立 ResultSet:

若要建立 ResultSet 物件，可以使用 executeQuery 方法或其他方法。本文說明建立 ResultSet 的選項。

這些方法來自 Statement、PreparedStatement 或 CallableStatement 介面。但還有其他可用的方法。例如 DatabaseMetaData 方法， 包括 getColumns、getTables、getUDTs、getPrimaryKeys 等，也一樣可以傳回 ResultSet。 單一 SQL 陳述式亦可傳回多個 ResultSet 來處理。 在呼叫 Statement、PreparedStatement 或 CallableStatement 介面所提供的 execute 方法之後， 亦可再使用 getResultSet 方法來擷取 ResultSet 物件。

如需詳細資訊，請參閱範例：建立含有多重 ResultSet 的程序。

關閉 ResultSet

縱使 ResultSet 物件在相關的 Statement 物件關閉時亦隨之自動關閉，但還是建議您主動關閉使用完畢的 ResultSet 物件。如此一來，可以立即釋放內部資料庫資源，進而提高應用程式的產能。

關閉 DatabaseMetaData 呼叫所產生的 ResultSet 也很重要。因爲無法直接存取最初建立這些 ResultSet 的 Statement 物件，所以並非直接在 Statement 物件上呼叫 close 方法。這些物件的鏈結方式可在您關閉外部 ResultSet 物件時，讓 JDBC 驅動程式也隨之關閉內部 Statement 物件。若未手動關閉這些物件，系統仍然可以繼續運作；但會佔用寶貴的資源。

註: ResultSet 的保留性質亦可代替您自動關閉 ResultSet。一個 ResultSet 物件容許多次呼叫 close 方法。

範例：IBM Developer Kit for Java 的 ResultSet 介面:

以下爲如何使用 ResultSet 介面的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;

/**
ResultSetExample.java

This program demonstrates using a ResultSetMetaData and
a ResultSet to display all the data in a table even though
the program that gets the data does not know what the table
is going to look like (the user passes in the values for the
table and library).
**/
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage:  java ResultSetExample <library> <table>");
            System.out.println(" where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Get a database connection and prepare a statement.
```

```
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
            int rowCount = 0;
            while (rs.next ()) {
                rowCount++;
                System.out.println("Data for row " + rowCount);
                for (int i = 1; i <= colCount; i++)
                    System.out.println("   Row " + i + ": " + rs.getString(i));
            }

        } catch (Exception e) {
            // Handle any errors.
            System.out.println("Oops... we have an error... ");
            e.printStackTrace();
        } finally {
            // Ensure we always clean up.  If the connection gets closed, the
            // statement under it closes as well.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Critical error - cannot close connection object");
                }
            }
        }
    }
}
```

JDBC 物件儲存區作業

在論及 Java Database Connectivity (JDBC) 及效能時，物件儲存區作業是最常討論的主題。JDBC 中有許多物件的建立代價很高，例如 Connection、Statement 及 ResultSet 物件，但只要重覆使用這些物件，而非每次需要時就建立，即可明顯提高效能。

許多應用程式已代替您處理好物件儲存區作業。例如，WebSphere 就對 JDBC 物件儲存區作業提供非常豐富的支援，可讓您控制如何管理儲存區作業。因此，直接取用您所需的功能，不須再考慮建立您自己的儲存區作業機制。然而，若缺乏這項支援，則必須尋找解決方案來管理所有瑣碎的應用程式。

使用 DataSource 支援來運用物件儲存區作業:

您可以使用 DataSources，讓多個應用程式共用一個通用的資料庫存取配置。只要讓每一個應用程式都參照相同的 DataSource 名稱，即可達成此目的。

利用 DataSource，可以從一個中心位置變更許多應用程式。比方說，如果變更所有應用程式所使用的預設檔案庫名稱，且使用單一 DataSource 來取得所有應用程式的連線，則您可以在 DataSource 中更新此集合的名稱。所有應用程式會開始改用新的預設檔案庫。

使用 DataSource 來取得應用程式的連線時，可以使用原有的 JDBC 驅動程式內建的連線儲存區作業支援。 這項支援供作 ConnectionPoolDataSource 介面的實作方式。

只要交出「邏輯」Connection 物件 (而非實體 Connection 物件)，即可達成儲存區作業。**邏輯 Connection 物件**係指排存的 Connection 物件所傳回的連線物件。每一個邏輯 Connection 物件都是當作排存的 Connection 物件所代表實體連線的暫時控點。對應用程式而言，當 Connection 物件傳回時，兩者並無明顯的差別。 只有在

呼叫 Connection 物件的 close 方法時，才有細微的差異。此呼叫會使邏輯連線失效，然後將實體連線送回儲存區作業，供另一個應用程式繼續使用實體連線。這項技術讓許多邏輯連線物件重覆使用單一實體連線。

設定連線儲存區作業

建立 DataSource 物件來參照 ConnectionPoolDataSource 物件，即可完成連線儲存區作業。可以設定 ConnectionPoolDataSource 物件的內容來處理各種儲存區作業維護工作。

有關如何使用 UDBDataSource 及 UDBConnectionPoolDataSource 來設定連線儲存區作業的詳細資訊，請參閱範例。關於 JNDI 於此範例中扮演的角色，亦請參閱「Java 命名和目錄介面 (JNDI)」。

範例中，負責將兩個 DataSource 物件鏈結起來的是 dataSourceName。 此鏈結指示 DataSource 物件將連線建立工作，留給會自動管理儲存區作業的 ConnectionPoolDataSource 物件來處理。

儲存區作業及非儲存區作業應用程式

應用程式使用連線儲存區作業與否並無差異。因此，可於應用程式碼完成之後再加入儲存區作業支援，應用程式碼不需任何修改。

如需詳細資訊，請參閱範例：測試連線儲存區作業的效能。

以下爲上述程式於開發期間在本端執行的輸出。

Start timing the non-pooling DataSource version... Time spent: 6410

Start timing the pooling version... Time spent: 282

Java program completed.

依預設，UDBConnectionPoolDataSource 僅排存單一連線。若應用程式須使用連線數次，但一次只需一個連線，UDBConnectionPoolDataSource 是最完美的解決方案。若同時間需要許多連線，則必須配置 ConnectionPoolDataSource 來符合您的需求及資源。

範例：使用 UDBDataSource 與 UDBConnectionPoolDataSource 設定連線儲存區作業:

以下範例將說明，如何利用 UDBDataSource 與 UDBConnectionPoolDataSource 來使用連線儲存區作業。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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
    {
        // Create a ConnectionPoolDataSource implementation
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object");

        // Establish a JNDI context and bind the connection pool data source
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Create a standard data source that references it.
        UDBDataSource ds = new UDBDataSource();
```

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

範例：測試連線儲存區作業的效能:

以下範例將說明，如何測試儲存區作業範例的效能與非儲存區作業範例的效能。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
        // Do the work without a pool:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("\nStart timing the non-pooling DataSource version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));

        // Do the work with pooling:
        ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
}
```

ConnectionPoolDataSource 內容:

若要配置 ConnectionPoolDataSource 介面，必須要使用其內容設定。

下表將說明這些內容。

內容	說明
initialPoolSize	當儲存區作業最初實例化時，這個內容可決定儲存區作業內要放入多少連線。若這個指定值超出 minPoolSize 至 maxPoolSize 之間的範圍，則以 minPoolSize 或 maxPoolSize 當作要建立的起始連線數。

內容	說明
maxPoolSize	使用儲存區作業時，所要求的連線有可能超出儲存區作業內具有的連線數。這個內容指定儲存區作業中容許建立的連線數目上限。 當儲存區作業達到大小上限且所有連線皆在使用中時，應用程式不會「暫停執行」及等待連線送回儲存區作業。而是由 JDBC 驅動程式根據 DataSource 內容建構新的連線，然後傳回此連線。 若指定 maxPoolSize 爲 0，只要系統有足夠的資源，即容許儲存區作業大小無限制地增加。
minPoolSize	無法使用儲存區作業會造成其中的連線數量攀升。若活動量遞減到某個程度，導致儲存區作業內從未取出一些「連線」，就等於平白浪費資源。 在這種情況下，JDBC 驅動程式有辦法釋放一些累積的連線。這個內容可讓您指示 JDBC 釋放連線，確保隨時都有一定的連線數量可供使用。 若指定 minPoolSize 爲 0，則可能導致儲存區作業釋放所有連線，而由於每一個連線要求所引起的連線時間，應用程式將付出很大的代價。
maxIdleTime	連線會追蹤本身閒置時間的長短。這個內容可指定應用程式容許連線閒置多久之後才加以釋放 (亦即，連線數量大於需求)。 這個內容以秒爲單位，並不指定關閉連線的確切時間。它只是指定一旦經過足夠的時間，就應該釋放連線。
propertyCycle	這個內容代表上述規則施行之間容許間隔的秒數。

註: 將 maxIdleTime 或 propertyCycle 時間設定爲 0， 即表示 JDBC 驅動程式本身不檢查是否從儲存區作業移除連線。針對起始大小、大小下限及大小上限所指定的規則仍然生效。

當 maxIdleTime 及 propertyCycle 不爲 0 時，則使用管理緒來監視儲存區作業。 緒每隔 propertyCycle 秒就會起動，然後檢查儲存區作業的所有連線，找出哪些連線已超過 maxIdleTime 秒未使用。符合此準則的連線就從儲存區作業內移除，直到減少至 minPoolSize 爲止。

DataSource 形式的陳述式儲存區作業:

maxStatements 內容 (在 UDBConnectionPoolDataSource 介面上可用) 允許連線儲存區內的陳述式儲存區作業。陳述式儲存區作業僅影響 PreparedStatement 及 CallableStatement。Statement 物件不會進行排存。

陳述式儲存區作業與連線儲存區作業的實作方式相類似。當應用程式呼叫 Connection.prepareStatement(″select * from tablex″) 時， 儲存區作業模組會檢查連線上是否已備妥 Statement 物件。 如果是，則會將一個邏輯的 PreparedStatement 物件交付給您，而非交付實體物件。呼叫 close 方法時，Connection 物件會送回儲存區作業，同時丟棄邏輯 Connection 物件，而 Statement 物件可以重覆使用。

maxStatements 內容可讓 DataSource 指定一個連線可以排存多少陳述式。 0 表示不使用陳述式儲存區作業。當陳述式儲存區作業排滿時，會套用最近最不常使用的演算法來決定應該丟棄的陳述式。

範例：測試兩個 DataSource 的效能測試一個僅採用連線儲存區作業的 DataSource，以及另一個同時採用陳述式儲存區作業和連線儲存區作業的 DataSource。

下列範例係本程式於開發期間在本端執行的輸出。

Deploying statement pooling data source Start timing the connection pooling only version... Time spent: 26312

Starting timing the statement pooling version... Time spent: 2292 Java program completed

範例：測試兩個 DataSource 的效能:

以下範例將測試兩個 DataSource，一個僅採用連線儲存區作業，另一個同時採用陳述式與連線儲存區作業。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class StatementPoolingTest
{

    public static void main(java.lang.String[] args)
    throws Exception
    {
        Context ctx = new InitialContext();

        System.out.println("deploying statement pooling data source");
        deployStatementPoolDataSource();

        // Do the work with connection pooling only.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the connection pooling only version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));


        // Do the work with statement pooling added.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("\nStart timing the statement pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
```

```
    private static void deployStatementPoolDataSource()
    throws Exception
    {
        // Create a ConnectionPoolDataSource implementation
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object with Statement pooling");
        cpds.setMaxStatements(10);

        // Establish a JNDI context and bind the connection pool data source
        Context ctx = new InitialContext();
        ctx.rebind("StatementSupport", cpds);


        // Create a standard datasource that references it.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting statement pooling");
        ds.setDataSourceName("StatementSupport");
        ctx.rebind("StatementPoolingDataSource", ds);

    }
}
```

建置自己的連線儲存區作業:

您可以開發自己的連線及陳述式儲存區作業，不必仰賴 DataSource 的支援或其他產品。

雖然是在小型的 Java 應用程式上示範儲存區作業技術，但同樣可將它應用於 Servlet 或大型的 N 層應用程式。本範例示範效能方面的問題。

示範應用程式有兩項功能：

- 在資料庫表格內插入新的索引及名稱。
- 從表格中讀取特定的索引的名稱。

連線儲存區應用程式的完成碼可從

JDBC 要訣及秘訣下載。

應用程式範例執行起來並不順利。在標準的工作站上，透過此程式碼各呼叫 getValue 方法與 putValue 方法 100 次， 平均費時 31.86 秒。

問題即在於每一項要求所引起的資料庫工作太沉重。也就是說，您必須取得連線、取得陳述式、處理陳述式、關閉陳述式，最後還要關閉連線。一定有方法可以重覆使用此程序的資源，而不是每一次要求之後就必須捨棄所有資源。**連線儲存區作業**可將建立連線的程式碼，取代爲從儲存區作業取得連線的程式碼，還可將關閉連線的程式碼，取代爲將連線送回儲存區作業的程式碼，以供使用。

連線儲存區作業的建構子會建立連線，並且放入儲存區作業內。儲存區作業類別具有取得及放回方法，分別用來尋找可用的連線，以及在連線使用完畢後，將連線送回儲存區作業。這些方法可同步進行，因爲儲存區作業物件是共用資源，但是讓多個緒同時操作排存的資源並不是理想的方式。

getValue 方法的呼叫程式碼有所變更。 putValue 方法並未顯示，但確實有所變更，可從 IBM 的 Developer Kit for Java JDBC Web page 取得。範例中也沒有顯示連線儲存區作業物件的實例化。您可以呼叫建構子，然後傳入想要從儲存區作業取出的連線物件數量。這項步驟應於啓動應用程式時完成。

執行前述應用程式 (含有 100 個 getValue 方法要求及 100 個 putValue 方法要求) 再加上這些變更，由於已加入連線儲存區作業程式碼，所以平均費時 13.43 秒。此時工作量的處理時間，比原先沒有連線儲存區作業的處理時間縮減了一半以上。

建置自己的陳述式儲存區作業

使用連線儲存區作業時，在每一個陳述式的處理中都會浪費時間來建立及關閉陳述式。這又是另一個可重覆使用的物件遭到浪費的例子。

您可以利用備妥陳述式類別來重覆使用物件。在大部份的應用程式中，只要稍做修改，即可重覆使用相同的 SQL 陳述式。例如，在一個應用程式中疊代執行時可能會產生下列查詢：

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

第二次疊代可能產生下列查詢：

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

查詢相同，只是參數不同。以上兩個查詢可透過下列這個查詢即可完成：

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

在處理第一個查詢時，可以將參數記號 (以問號表示) 設定爲 100000，處理第二個查詢時，再改設爲 50000。如此即可加強效能，除了連線儲存區作業的優點之外，還有三個原因：

- 建立的物件較少。PreparedStatement 物件建立後可以重覆使用，不需在每一個要求都建立一個 Statement 物件。 因此，執行的建構子較少。
- 資料庫設定 SQL 陳述式的工作 (稱爲**準備**) 可以重覆使用。 準備 SQL 陳述式的代價相當高，因爲涉及判斷 SQL 陳述式文字的意義，以及系統完成要求作業的方式。
- 除去不必要的物件建立工作之後，您將獲得意想不到的好處。您也不必摧毀從未建立的物件。此模型透過 Java 垃圾收集器即可輕易達成，在眾多使用者存取期間，也有助於提升效能。

示範程式可以改爲排存 PreparedStatement 物件，而非 Connection 物件。變更程式即可重覆使用更多物件並改善效能。首先您可以編寫類別，內含要排存的物件。此類別必須封裝各種會用到的資源。以連線儲存區作業爲例，Connection 是唯一排存的資源，所以不需使用封裝類別。每一個排存的物件必須包含一個 Connection 及兩個 PreparedStatement。然後，您就可以建立儲存區作業類別，內含資料庫存取物件，而非連線物件。

最後，必須變更應用程式來取得資料庫存取物件，再從物件中指定所要使用的資源。除了指定特定的資源以外，應用程式仍然維持原狀。

經過這項變更之後，執行相同的測試現在平均僅費時 0.83 秒。 這比最原始的程式版本還快了 38 倍之多。

使用 ANZJVM 指令的注意事項

重覆使用性是效能提升的關鍵。若有某個項目未重覆使用，就必須予以排存而造成資源的浪費。

大部份應用程式都包含主要的程式碼區段。一般來說，應用程式將 80% 至 90% 的處理時間，都集中花在 10% 至 20% 的少數程式碼上。若應用程式中可能用到 10,000 個 SQL 陳述式，並非全部都會排存。重點是要找出應用程式的主要程式碼區段用到的 SQL 陳述式，並加以排存。

在 Java 實作方式中建立物件的成本很高。此時可採用儲存區作業解決方案提供的優點。一開始在其他使用者試圖使用系統之前，就會建立程序中使用的物件。這些物件會儘可能地重覆使用。其效能非常卓越，且可以逐漸細部調整應用程式，妥善應付大量的使用者。因此，可排存的物件會更多。何況，它採用更有效率的多緒作業來進行應用程式的資料庫存取，獲得更大的產能。

Java (使用 JDBC) 以動態 SQL 爲基礎，速度較慢。儲存區作業可減低此問題的嚴重性。一開始就備妥陳述式，即可靜態呈現資料庫存取。備妥陳述式之後，動態或靜態 SQL 之間的效能並無太大差異。

Java 中的資料庫存取效能可以更有效率，且不必犧牲物件導向設計或程式碼維護性。編寫程式碼來建置陳述式儲存區作業或連線儲存區作業並不困難。再者，還可以變更及加強程式碼來支援多重應用程式和應用程式類型 (Web 型、主從架構) 等等。

批次更新

批次更新支援可讓資料庫的任何更新，在使用者程式與資料庫之間視爲單一異動來傳遞。必須一次執行許多更新時，這項程序可明顯改善效能。

比方說，如果有一家大型公司要求新進員工統一從星期一開始上班，爲達成這項需求，就必須在員工資料庫中一次處理許多更新 (在此爲插入)。建立更新批次並將其視爲一個單位提交至資料庫，即可節省處理時間。

批次更新有兩種：

- 使用 Statement 物件的批次更新。
- 使用 PreparedStatement 物件的批次更新。

Statement 批次更新:

若要執行 Statement 批次更新，您必須關閉自動確定這項特性。在 Java Database Connectivity (JDBC) 中，依預設會開啓自動確定。自動確定表示處理某一個 SQL 陳述式之後，就立即確定資料庫的更新。若要將提交到資料庫的一組陳述式視爲一個功能群，讓資料庫個別地確定每一個陳述式並不是好方法。在未關閉自動確定的情形下，若批次處理過程中有某個陳述式失敗，您將無法回轉整個批次來重試，因爲有一半的陳述式已經確定結果。而且，要確定批次中的每一個陳述式，必須執行較多工作，將造成許多額外執行時間。

如需詳細資訊，請參閱異動。

在關閉自動確定之後，您即可建立標準的 Statement 物件。請用 addBatch 方法將陳述式加入批次中，別用 executeUpdate 之類的方法來處理。在批次內新增您所需的所有陳述式之後，即可使用 executeBatch 方法開始處理這些陳述式。您隨時都可利用 clearBatch 方法來清空批次。

下列範例顯示如何使用這些方法：

範例：Statement 批次更新

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
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();
```

在此範例中，executeBatch 方法會傳回一個整數陣列。在此陣列中，針對批次中已處理的每一個陳述式，都會有一個整數值。若要在資料庫內插入值，則每一個陳述式在此陣列中的值爲 1 (亦即，假設順利完成處理)。然而，有些陳述式可能是更新陳述式，會造成多列受到影響。若在批次中放入 INSERT、UPDATE 或 DELETE 以外的任何陳述式，則會發生異常。

PreparedStatement 批次更新:

preparedStatement 批次類似 Statement 批次；然而，preparedStatement 批次會完成相同的備妥陳述式，您只需變更陳述式的參數即可。

以下爲使用 preparedStatement 批次的範例。

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
connection.setAutoCommit(false);
PreparedStatement statement =
  connection.prepareStatement("INSERT INTO TABLEX VALUES(?, ?)");
statement.setInt(1, 1);
statement.setString(2, "Cujo");
statement.addBatch();
statement.setInt(1, 2);
statement.setString(2, "Fred");
statement.addBatch();
statement.setInt(1, 3);
statement.setString(2, "Mark");
statement.addBatch();
int [] counts = statement.executeBatch();
     connection.commit();
```

BatchUpdateException:

批次更新有一項重要的考量，就是當呼叫 executeBatch 方法失敗時，應該採取什麼動作。在此情況下，就會丟出一個新的異常類型，稱爲 BatchUpdateException。BatchUpdateException 是 SQLException 的子類別，有了它，您以往接收訊息、SQLState 及供應商代碼時所採用的呼叫方法，都可以繼續使用。

BatchUpdateException 亦提供 getUpdateCounts 方法來傳回整數陣列。此整數陣列包含批次中所有陳述式到失敗發生爲止，已完成的更新計數。陣列長度會指出批次中失敗的陳述式。比方說，如果異常裡傳回的陣列長度爲 3，則表示批次的第 4 個陳述式失敗。因此，從傳回的單一 BatchUpdateException 物件，即可判斷所有順利完成的陳述式的更新計數、哪一個陳述式失敗，以及關於失敗的所有資訊。

處理批次更新的標準效能，可媲美獨立處理每一個陳述式的效能。如需批次更新最佳化支援的相關資訊，請參閱區塊化插入支援。您應該採用新的模型來編寫程式碼，享受未來效能最佳化的成果。

註: 在 JDBC 2.1 規格中提供了不同選項可處理批次更新的異常狀況。JDBC 2.1 引進一種模型，可在某個批次項目失敗之後，繼續處理批次。針對每一個失敗項目所傳回的更新計數整數，此整數陣列中會加入一個特殊的更新計數。如此一來，即使大型批次中有某個項目失敗，批次仍然可以繼續處理。關於這兩種作業模式的詳細資料，請參閱 JDBC 2.1 或 JDBC 3.0 規格。依預設，原有的 JDBC 驅動程式會使用 JDBC 2.0 定義。此驅動程式提供一個 Connection 內容，可在使用 DriverManager 建立連線時派上用場。驅動程式亦提供 DataSource 內容，在使用 DataSource 建立連線時就會用到。這些內容可讓應用程式選擇批次作業處理失敗狀況的方式。

區塊化插入支援:

您可以在 iSeries 作業中使用區塊化插入，以在資料庫表格中一次插入多列。

區塊化插入是 iSeries 伺服器上一項特殊的作業類型，可提供在資料庫表格中一次插入多列的最佳方式。區塊化插入可視爲批次更新的一部份。批次更新可以是任何形式的更新要求，而區塊化插入的更新要求，其形式是特定的。然而，區塊化插入類型的批次更新頗爲常見；原有的 JDBC 驅動程式已改爲使用這項功能。

由於使用區塊化插入支援時的系統限制，原有的 JDBC 驅動程式的預設會停用區塊插入。但可透過 Connection 內容或 DataSource 內容來啓用。使用區塊化插入時，系統會替您代爲檢查和處理大部分的限制，但有少數限制例外；這就是預設會關閉區塊化插入支援的原因。這些限制如下：

- 使用的 SQL 陳述式必須是含有 VALUES 子句的 INSERT 陳述式，亦即不是含有 SUBSELECT 的 INSERT 陳述式。JDBC 驅動程式會辨識這項限制，並採取適當的動作。
- 必須使用 PreparedStatement，表示 Statement 物件並無最佳化支援。JDBC 驅動程式會辨識這項限制，並採取適當的動作。

- SQL 陳述式必須爲表格中所有的直欄指定參數記號。這表示不可在直欄中使用常數值，或讓資料庫插入任何直欄的預設值。JDBC 驅動程式並無機制可測試 SQL 陳述式內的特定參數記號。若您設定內容來執行最佳化的區塊化插入，但未在 SQL 陳述式中避開預設值或常數，則最後資料庫表格中將會是不正確的值。
- 連線必須連至本端系統。這表示不可使用以 DRDA 存取遠端系統的連線，因爲 DRDA 不支援區塊化插入作業。JDBC 驅動程式並無機制可測試本端系統的連線。若您設定內容來執行最佳化的區塊化插入，且嘗試連接遠端系統，則批次更新的處理將會失敗。

以下程式碼範例顯示如何啓用區塊化插入處理程序的支援。比較本程式碼與不使用區塊化插入支援的程式碼，唯一差別就在於 Connection URL 中加入的 `use block insert=true`。

範例：區塊化插入處理

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Create a database connection
Connection c = DriverManager.getConnection("jdbc:db2:*local;use block insert=true");
BigDecimal bd = new BigDecimal("123456");

// Create a PreparedStatement to insert into a table with 4 columns
PreparedStatement ps =
  c.prepareStatement("insert into cujosql.xxx values(?, ?, ?, ?)");

// Start timing...
for (int i = 1; i <= 10000; i++) {
  ps.setInt(1, i);                        // Set all the parameters for a row
  ps.setBigDecimal(2, bd);
  ps.setBigDecimal(3, bd);
  ps.setBigDecimal(4, bd);
  ps.addBatch();                          //Add the parameters to the batch
}

// Process the batch
int[] counts = ps.executeBatch();

// End timing...
```

在類似的測試案例中，以相同作業而言，使用區塊化插入的速度就比不使用區塊化插入快了好幾倍。例如，上述程式碼測試的結果，使用區塊化插入的速度就快了 9 倍。在僅使用初始類型而不使用物件的情況下，速度甚至更可快上 16 倍。若應用程式需處理大量的工作，您就得適當地調整預期的速度。

進階資料類型

進階 SQL3 資料類型提供極大的彈性。很適合用來儲存序列化 Java 物件、「可延伸標記語言 (XML)」文件，以及多媒體資料，例如歌曲、產品圖片、員工照片及電影片段等。Java Database Connectivity (JDBC) 2.0 及更高版本，支援使用這些屬於 SQL99 標準的資料類型。

特殊類型

特殊類型是指基於標準資料庫類型的使用者定義類型。例如，您可以定義「社會保險號碼」類型 (SSN)，本質上是 CHAR(9)。下列 SQL 陳述式可建立這種 DISTINCT 類型。

```
CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)
```

特殊類型一律對映至某個內建資料類型。有關在 SQL 環境中使用特殊類型的方法與時機，請參閱 SQL 參考手冊。

在 JDBC 中使用特殊類型時，您可以使用存取基礎類型的方式來存取它們。getUDTs 方法是新方法，可讓您查詢系統上可用的特殊類型。第 139 頁的『範例：特殊類型』程式將顯示下列項目：

- 建立特殊類型。
- 建立會用到此特殊類型的表格。
- 使用 PreparedStatement 來設定特殊類型參數。
- 使用 ResultSet 來傳回特殊類型。
- 使用 meta 資料「應用程式設計介面 (API)」來呼叫 getUDTs，以取得特殊類型的相關資訊。

如需詳細資訊，請參閱下列範例，內含可透過特殊類型執行的各種常見作業：

第 139 頁的『範例：特殊類型』

大型物件

「大型物件 (LOB)」共有三種：
- 二進位大型物件 (BLOB)
- 字元大型物件 (CLOB)
- 雙位元組字元大型物件 (DBCLOB)

DBCLOB 類似 CLOB，但字元資料的內部儲存表示法不同。因爲 Java 與 JDBC 皆會外部化所有的字元資料而以 Unicode 呈現它們，所以只有 JDBC 才支援 CLOB。就 JDBC 的觀點而言，DBCLOB 與 CLOB 可以互相支援。

二進位大型物件

在許多方面，「二進位大型物件 (BLOB)」直欄類似可擴大的 CHAR FOR BIT DATA 直欄。您可以在這些欄位中儲存任何資料，這些資料可表示爲不經轉換的位元組串流。通常，BLOB 直欄可用來儲存序列化 Java 物件、圖片、歌曲及其他二進位資料。

BLOB 的用法與其他標準資料庫類型的使用方式相同。BLOB 可以傳遞至儲存程序、用於備妥陳述式，以及在結果集內更新。PreparedStatement 類別的 setBlob 方法可將 BLOB 傳遞至資料庫，而 ResultSet 類別可新增 getBlob 類別，從資料庫擷取 BLOB。BLOB 在 Java 程式中以 BLOB 物件表示，它是一個 JDBC 介面。

如需如何使用 BLOB 的相關資訊，請參閱編寫使用 BLOB 的程式碼。

字元大型物件

「字元大型物件 (CLOB)」是 BLOB 的增補字元資料。資料並非不經轉換就直接儲存於資料庫，而會在資料庫中儲存爲文字，並且以處理 CHAR 直欄的相同方式加以處理。如同 BLOB，JDBC 2.0 亦提供函數來直接處理 CLOB。PreparedStatement 介面提供 setClob 方法，ResultSet 介面也提供 getClob 方法。

如需如何使用 CLOB 的相關資訊，請參閱編寫使用 CLOB 的程式碼。

雖然 BLOB 與 CLOB 直欄的操作就像 CHAR FOR BIT DATA 及 CHAR 直欄一樣，但這只是概念上從外部使用者的觀點來看。就內部而言，兩者即有所差異；因爲無法預期「大型物件 (LOB)」究竟多大，您通常會採取間接方式來處理資料。例如，從資料庫提取一整塊的資料列時，您並不是直接將整塊 LOB 移到 ResultSet。而是把稱爲 LOB 定址器的指標 (4 位元組整數) 移入 ResultSet。但在 JDBC 中使用 LOB 時，就不必考慮定址器。

Datalink

Datalink 是封裝的值，包含從資料庫至檔案 (儲存於資料庫之外) 的邏輯參照。就 JDBC 的觀點而言，Datalink 有兩種不同的表示與用法，取決於您使用的是 JDBC 2.0 或更早版本，還是 JDBC 3.0 或更高版本。

如需如何使用 Datalink 的相關資訊，請參閱編寫使用 Datalink 的程式碼。

未支援的 SQL3 資料類型

還有其他已定義且受 JDBC API 支援的 SQL3 資料類型。包括 ARRAY、REF 及 STRUCT。目前，iSeries 伺服器不支援這些類型。因此，JDBC 驅動程式對它們不提供任何形式的支援。

編寫使用 BLOB 的程式碼:

利用 Java Database Connectivity (JDBC)「應用程式設計介面 (API)」，可以對資料庫「二進位大型物件 (BLOB)」直欄完成許多作業。下列主題將簡短地討論這些作業，並提供範例來說明如何完成。

從資料庫讀取 BLOB，以及將 BLOB 插入資料庫內

透過 JDBC API，即可利用某些方法從資料庫取出 BLOB，以及將 BLOB 放入資料庫。然而，Blob 物件並無標準的建立方法。若資料庫已填滿 BLOB，則沒有問題，但若是透過 JDBC 從頭開始處理 BLOB，就會產生問題。不過，目前已支援直接以其他類型在資料庫裡放入及取出 BLOB，不必再爲 JDBC API 的 Blob 與 Clob 介面定義建構子。例如，setBinaryStream 方法就可處理 Blob 類型的資料庫直欄。『範例：BLOB』顯示在資料庫裡放入 BLOB 或從資料庫擷取 BLOB 的一些常用方式。

使用 Blob 物件 API

BLOB 定義爲 JDBC 的介面，由各種驅動程式提供實作方式。此介面有一系列的方法可與 Blob 物件互動。第 133 頁的『範例：使用 BLOB』顯示可以使用此 API 執行的一些常見作業。關於可在 Blob 物件上使用的所有方法，請參閱 JDBC Javadoc。

使用 JDBC 3.0 支援來更新 BLOB

JDBC 3.0 支援變更 LOB 物件。這些變更可儲存在資料庫的 BLOB 直欄中。第 132 頁的『範例：更新 BLOB』顯示 JDBC 3.0 的 BLOB 支援可執行的一些作業。

範例：BLOB:

以下範例說明如何在資料庫內放入或擷取 BLOB。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// PutGetBlobs is an example application
// that shows how to work with the JDBC
// API to obtain and put BLOBs to and from
// database columns.
//
// The results of running this program
// are that there are two BLOB values
// in a new table. Both are identical
// and contain 500k of random byte
// data.
//////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

```
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      // Establish a Connection and Statement with which to work.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Clean up any previous run of this application.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
      } catch (SQLException e) {
          // Ignore it - assume the table did not exist.
      }

      // Create a table with a BLOB column. The default BLOB column
      // size is 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

      // Create a PreparedStatement object that allows you to put
      // a new Blob object into the database.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

      // Create a big BLOB value...
      Random random = new Random ();
      byte [] inByteArray = new byte[500000];
      random.nextBytes (inByteArray);

      // Set the PreparedStatement parameter. Note: This is not
      // portable to all JDBC drivers. JDBC drivers do not have
      // support when using setBytes for BLOB columns. This is used to
      // allow you to generate new BLOBs. It also allows JDBC 1.0
      // drivers to work with columns containing BLOB data.
      ps.setBytes(1, inByteArray);

      // Process the statement, inserting the BLOB into the database.
      ps.executeUpdate();

      // Process a query and obtain the BLOB that was just inserted out
      // of the database as a Blob object.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
      rs.next();
      Blob blob = rs.getBlob(1);


      // Put that Blob back into the database through
      // the PreparedStatement.
      ps.setBlob(1, blob);
      ps.execute();

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：更新 BLOB:

以下爲如何在應用程式中更新 BLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////
// UpdateBlobs is an example application
// that shows some of the APIs providing
// support for changing Blob objects
// and reflecting those changes to the
// database.
```

```
//
// This program must be run after
// the PutGetBlobs program has completed.
//////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Truncate a BLOB.
      blob1.truncate((long) 150000);
      System.out.println("Blob1's new length is " + blob1.length());

      // Update part of the BLOB with a new byte array.
      // The following code obtains the bytes that are at
      // positions 4000-4500 and set them to positions 500-1000.

      // Obtain part of the BLOB as a byte array.
      byte[] bytes = blob1.getBytes(4000L, 4500);

      int bytesWritten = blob2.setBytes(500L, bytes);

      System.out.println("Bytes written is " + bytesWritten);

      // The bytes are now found at position 500 in blob2
      long startInBlob2 = blob2.position(bytes, 1);

      System.out.println("pattern found starting at position " + startInBlob2);

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：使用 BLOB:

以下爲如何在應用程式中使用 BLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// UseBlobs is an example application
// that shows some of the APIs associated
// with Blob objects.
//
// This program must be run after
// the PutGetBlobs program has completed.
//////////////////////////////////////
```

```
import java.sql.*;

public class UseBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Determine the length of a LOB.
      long end = blob1.length();
      System.out.println("Blob1 length is " + blob1.length());

      // When working with LOBs, all indexing that is related to them
      // is 1-based, and is not 0-based like strings and arrays.
      long startingPoint = 450;
      long endingPoint = 500;

      // Obtain part of the BLOB as a byte array.
      byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

      // Find where a sub-BLOB or byte array is first found within a
      // BLOB. The setup for this program placed two identical copies of
      // a random BLOB into the database. Thus, the start position of the
      // byte array extracted from blob1 can be found in the starting
      // position in blob2. The exception would be if there were 50
      // identical random bytes in the LOBs previously.
      long startInBlob2 = blob2.position(outByteArray, 1);

      System.out.println("pattern found starting at position " + startInBlob2);

      c.close(); // Connection close closes stmt and rs too.
   }
}
```

編寫使用 CLOB 的程式碼:

利用 Java Database Connectivity (JDBC)「應用程式設計介面 (API)」，可以對資料庫 CLOB 及 DBCLOB 直欄執行許多作業。下列主題將簡短地討論這些作業，並提供範例來說明如何完成。

從資料庫讀取 CLOB，以及將 CLOB 插入資料庫內

透過 JDBC API，即可利用某些方法從資料庫取出 CLOB，以及將 CLOB 放入資料庫。然而，Clob 物件並無標準的建立方法。若資料庫已填滿 CLOB，則沒有問題，但若是透過 JDBC 從頭開始處理 CLOB，就會產生問題。不過，目前已支援直接以其他類型在資料庫裡放入及取出 CLOB，不必再爲 JDBC API 的 Blob 與 Clob 介面定義建構子。例如，setCharacterStream 方法可處理 Clob 類型的資料庫直欄。第 135 頁的『範例：CLOB』顯示在資料庫裡放入 CLOB 或從資料庫擷取 CLOB 的一些常用方式。

使用 Clob 物件 API

CLOB 定義爲 JDBC 的介面，由各種驅動程式提供實作方式。此介面有一系列的方法可與 Clob 物件互動。這個範例顯示此 API 可執行的一些常見作業。關於可在 Clob 物件上使用的所有方法，請參閱 JDBC Javadoc。

使用 JDBC 3.0 支援來更新 CLOB

JDBC 3.0 支援變更 LOB 物件。這些變更可儲存在資料庫的 CLOB 直欄中。這個範例顯示 JDBC 3.0 的 CLOB 支援可執行的一些常見作業。

範例：CLOB:

以下範例說明如何在資料庫內放入或擷取 CLOB。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// PutGetClobs is an example application
// that shows how to work with the JDBC
// API to obtain and put CLOBs to and from
// database columns.
//
// The results of running this program
// are that there are two CLOB values
// in a new table. Both are identical
// and contain about 500k of repeating
// text data.
//////////////////////////////////////
import java.sql.*;

public class PutGetClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      // Establish a Connection and Statement with which to work.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Clean up any previous run of this application.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
      } catch (SQLException e) {
          // Ignore it - assume the table did not exist.
      }

      // Create a table with a CLOB column. The default CLOB column
      // size is 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

      // Create a PreparedStatement object that allow you to put
      // a new Clob object into the database.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

      // Create a big CLOB value...
      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();

      // Set the PreparedStatement parameter. This is not
      // portable to all JDBC drivers. JDBC drivers do not have
      // to support setBytes for CLOB columns. This is done to
      // allow you to generate new CLOBs. It also
      // allows JDBC 1.0 drivers a way to work with columns containing
      // Clob data.
      ps.setString(1, clobValue);

      // Process the statement, inserting the clob into the database.
      ps.executeUpdate();

      // Process a query and get the CLOB that was just inserted out of the
      // database as a Clob object.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
      rs.next();
      Clob clob = rs.getClob(1);


      // Put that Clob back into the database through
      // the PreparedStatement.
      ps.setClob(1, clob);
      ps.execute();

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：更新 CLOB:

以下爲如何在應用程式中更新 CLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database. //
// This program must be run after
// the PutGetClobs program has completed.
////////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Truncate a CLOB.
      clob1.truncate((long) 150000);
      System.out.println("Clob1's new length is " + clob1.length());

      // Update a portion of the CLOB with a new String value.
      String value = "Some new data for once";
      int charsWritten = clob2.setString(500L, value);

      System.out.println("Characters written is " + charsWritten);

      // The bytes can be found at position 500 in clob2
      long startInClob2 = clob2.position(value, 1);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：使用 CLOB:

以下爲如何在應用程式中使用 CLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database. //
// This program must be run after
// the PutGetClobs program has completed.
//////////////////////////////////////
import java.sql.*;

public class UseClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Determine the length of a LOB.
      long end = clob1.length();
      System.out.println("Clob1 length is " + clob1.length());

      // When working with LOBs, all indexing that is related to them
```

```
      // is 1-based, and not 0-based like strings and arrays.
      long startingPoint = 450;
      long endingPoint = 50;

      // Obtain part of the CLOB as a byte array.
      String outString = clob1.getSubString(startingPoint, (int)endingPoint);
      System.out.println("Clob substring is " + outString);

      // Find where a sub-CLOB or string is first found within a
      // CLOB. The setup for this program placed two identical copies of
      // a repeating CLOB into the database. Thus, the start position of the
      // string extracted from clob1 can be found in the starting
      // position in clob2 if the search begins close to the position where
   // the string starts.
      long startInClob2 = clob2.position(outString, 440);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

編寫使用 Datalink 的程式碼:

使用 Datalink 的方式視您採用的版次而定。在 JDBC 3.0 中，支援使用 getURL 與 putURL 方法以直接使用 Datalink 直欄。

而在先前的 JDBC 版本中，則必須將 Datalink 直欄當作 String 直欄來使用。目前，資料庫不支援 Datalink 與字元資料類型之間的自動轉換。因此，您需要在 SQL 陳述式中執行一些類型強制轉型。

範例：Datalink:

以下爲如何在應用程式中使用 Datalink 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////
// PutGetDatalinks is an example application
// that shows how to use the JDBC
// API to handle datalink database columns.
////////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
    throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Setup error.
        }

        // Establish a Connection and Statement with which to work.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any previous run of this application.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
        } catch (SQLException e) {
            // Ignore it - assume the table did not exist.
```

```
        }

        // Create a table with a datalink column.
        s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

        // Create a PreparedStatement object that allows you to add
        // a new datalink into the database. Since conversing
        // to a datalink cannot be accomplished directly in the database, you
        // can code the SQL statement to perform the explicit conversion.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
                                                   VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

        // Set the datalink. This URL points you to an article about
        // the new features of JDBC 3.0.
        ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

        // Process the statement, inserting the CLOB into the database.
        ps.executeUpdate();

        // Process a query and obtain the CLOB that was just inserted out of the
        // database as a Clob object.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        String datalink = rs.getString(1);


        // Put that datalink value into the database through
        // the PreparedStatement. Note: This function requires JDBC 3.0
        // support.
        /*
        try {
            URL url = new URL(datalink);
            ps.setURL(1, url);
            ps.execute();
        } catch (MalformedURLException mue) {
            // Handle this issue here.
        }

        rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        URL url = rs.getURL(1);
        System.out.println("URL value is " + url);
        */

        c.close(); // Connection close also closes stmt and rs.
    }
}
```

範例：特殊類型:

以下爲如何使用特殊類型的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// This example program shows examples of
// various common tasks that can be done
// with distinct types.
//////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
    throws SQLException
    {
        // Register the native JDBC driver.
```

```
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any old runs.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        // Create the type, create the table, and insert a value.
        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();

        // You can obtain details about the types available with new metadata in
        // JDBC 2.0
        DatabaseMetaData dmd = c.getMetaData();

        int types[] = new int[1];
        types[0] = java.sql.Types.DISTINCT;

        ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN", types);
        rs.next();
        System.out.println("Type name " + rs.getString(3) +
                           " has type " + rs.getString(4));


        // Access the data you have inserted.
        rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
        rs.next();
        System.out.println("The SSN is " + rs.getString(1));

        c.close(); // Connection close also closes stmt and rs.
    }
}
```

RowSet

RowSet 首見於 Java Database Connectivity (JDBC) 2.0 選用性套件。不同於 JDBC 規格中一些常見的介面，RowSet 規格的設計與其說是實際的實作方式，其實更接近於一種組織架構。 Rowset 介面定義了一組核心功能，所有 Rowset 都能夠使用。實作 RowSet 的供應商有極大的彈性可以定義所需的功能， 以期達成特定問題領域中的需求。

RowSet 性質:

可以要求 RowSet 必須滿足某些內容。一般內容包括結果 Rowset 所支援的介面集。

RowSet 就是 ResultSet

RowSet 介面可延伸 ResultSet 介面，表示 RowSet 能夠執行 ResultSet 具備的所有功能。例如，RowSet 可捲動，亦可更新。

RowSet 與資料庫的連線可以中斷

RowSet 有以下兩種：

- **連線** 當在已連線的 RowSet 內輸入資料時，會一直將基礎資料庫的內部連線保持爲開啓，並成爲 ResultSet 實作方法的封套。
- **離線** 離線 RowSet 不必一直維持資料來源的連線。離線 RowSet 可以從資料庫分離，經過各種用途的操作之後，再重新連接資料庫來反映任何變更。

RowSet 是 JavaBeans™ 元件

RowSet 可支援 JavaBeans 事件處理模型的事件處理。其中亦提供可設定的內容。RowSet 可利用這些內容來執行下列工作：

- 建立資料庫連線。
- 處理 SQL 陳述式。
- 判斷 RowSet 所代表的資料特性，以及處理 RowSet 物件的其他內部特性。

RowSet 可進行序列化

RowSet 可以進行序列化及解除序列化，以便能夠流通於網路連線上、寫入純文字檔 (亦即，無任何文書處理或其他結構字元的文字文件) 等等。

DB2CachedRowSet:

DB2CachedRowSet 物件是一種斷線狀態的 RowSet，表示可在不連接資料庫的情況下使用。其實作方式嚴格遵循 CachedRowSet 的說明。DB2CachedRowSet 是存放 ResultSet 資料列的儲存區。DB2CachedRowSet 自己會保留全部的資料，因此不必保持資料庫連線，除非對資料庫進行讀寫時，才需明確的連線。

使用 DB2CachedRowSet:

因爲 DB2CachedRowSet 物件可以處於斷線狀態及進行序列化，在無法隨時執行完整 JDBC 驅動程式的環境中，就顯得十分有用 (例如，個人數位助理 (PDA) 及 Java 行動電話)。

由於 DB2CachedRowSet 物件會存在記憶體中，且其資料皆爲已知資料，因此可供應用程式當作高度最佳化的捲動式 ResultSet。然而，由於其隨機移動的介面，再加上 JDBC 驅動程式會快取資料列，可捲動的 DB2ResultSet 在效能方面通常得付出代價，RowSet 就沒有這種問題。

DB2CachedRowSet 提供兩個方法來建立新的 RowSet：

- createCopy 方法可建立新的 RowSet，與原來被複製的 RowSet 一模一樣。
- createShared 方法可建立新的 RowSet，與原始 RowSet 共用相同的基礎資料。

您可以使用 createCopy 方法，將相同的 ResultSet 傳送給用戶端。若表格資料不會變更，則建立一個 RowSet 複本並傳給每一個用戶端，會比每一次都查詢資料庫還來得更有效率。

您可以使用 createShared 方法，讓許多人同時使用相同的資料，進而提升資料庫的效能。舉例來說，假設您有一個網站，每當客戶造訪時，首頁會顯示排名前 20 的最暢銷商品。您需要經常更新首頁的資訊，但每次客戶造訪首頁時就查詢最暢銷商品，實在沒有效率。只要利用 createShared 方法，即可有效率地爲每一位客戶建立

「游標」，不必再處理查詢，或浪費記憶體來儲存大量的資訊。當然，必要時仍可重新執行查詢，找出最暢銷的商品。RowSet 中可以大量輸入新的資料，再用 RowSet 建立共用的游標，供 Servlet 使用。

DB2CachedRowSet 提供一項延遲處理功能。此功能可將多個查詢要求組合起來，視爲單一要求來提交資料庫處理。在這個範例中，說明了如何使用 DB2CachedRowSets 來避免資料庫原本可能面臨的運算壓力。

因爲 RowSet 必須隨時掌握本身發生的變更，以便立刻反映給資料庫，因此您可以使用一些函數來還原變更，或讓您檢視所有曾執行的變更。例如，有一個 showDeleted 方法，可用來指示 RowSet 讓您提取已刪除的列。另外還有 cancelRowInsert 及 cancelRowDelete 方法，分別可還原已完成的列插入及列刪除動作。

DB2CachedRowSet 物件與其他 Java API 的互用性甚佳，因爲它所具備的事件處理支援及 toCollection 方法，可將整個或部分的 RowSet 轉換成 Java 集合。

DB2CachedRowSet 的事件處理支援，可應用於圖形式使用者介面 (GUI) 應用程式，用來控制顯示畫面、記載 RowSet 變更時的相關資訊，或尋找 RowSet 以外的資源的變更資訊。如需詳細資料，請參閱範例：DB2JdbcRowSet 事件。

關於使用 DB2CachedRowSet 的特定詳細資料，請參閱下列主題：

- 建立 DB2CachedRowSet 並大量輸入資料
- 存取 DB2CachedRowSet 資料及游標操作
- 變更 DB2CachedRowSet 資料及向資料來源反映變更
- 其他 DB2CachedRowSet 功能

如需事件模型及事件處理的相關資訊，請參閱 DB2JdbcRowSet，因爲此支援在這兩種 RowSet 中的運作方式完全一樣。

建立 DB2CachedRowSet 並大量輸入資料:

有幾種方法可以將資料放入 DB2CachedRowSet。

使用 populate 方法

DB2CachedRowSets 的 populate 方法，可用來將 DB2ResultSet 物件的資料放入 RowSet。以下爲這種方式的範例。

範例：使用 populate 方法

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Establish a connection to the database.
    Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a statement and use it to perform a query.
            Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");

// Create and populate a DB2CachedRowSet from it.
DB2CachedRowSet crs = new DB2CachedRowSet();
crs.populate(rs);

// Note: Disconnect the ResultSet, Statement,
// and Connection used to create the RowSet.
rs.close();
            stmt.close();
conn.close();
```

```
// Loop through the data in the RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

crs.close();
```

使用 DB2CachedRowSet 內容及 DataSource

DB2CachedRowSet 的內容可讓 DB2CachedRowSets 接受 SQL 查詢及 DataSource 名稱。接著，即可使用 SQL 查詢及 DataSource 名稱來爲自己建立資料。以下爲這種方式的範例。將名爲 BaseDataSource 的 DataSource 的參照，假設爲事先已設定的有效 DataSource。

範例：使用 DB2CachedRowSet 內容及 DataSource

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for
// the RowSet to use a DataSource to populate itself.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the DataSource and SQL query
// specified to populate itself with data. Once
// the RowSet is populated, it disconnects from the database.
crs.execute();

// Loop through the data in the RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

使用 DB2CachedRowSet 內容及 JDBC URL

DB2CachedRowSet 的內容可讓 DB2CachedRowSets 接受 SQL 查詢及 JDBC URL。接著，即可使用查詢及 JDBC URL 來爲自己建立資料。以下爲這種方式的範例。

範例：使用 DB2CachedRowSet 內容及 JDBC URL

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for
// the RowSet to use a JDBC URL to populate itself.
crs.setUrl("jdbc:db2:*local");
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the DataSource and SQL query
// specified to populate itself with data. Once
// the RowSet is populated, it disconnects from the database.
crs.execute();

// Loop through the data in the RowSet.
while (crs.next()) {
```

```
  System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

使用 setConnection(Connection) 方法來利用現有的資料庫連線

爲了提高 JDBC Connection 物件的重覆使用性，DB2CachedRowSet 提供了一項機制，可將已建立的 Connection 物件傳遞至 DB2CachedRowSet (用於將資料輸入 RowSet 中)。若傳入使用者提供的 Connection 物件，則 DB2CachedRowSet 本身在輸入資料之後，不會中斷此連線物件。

範例：使用 setConnection(Connection) 方法來利用現有的資料庫連線

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Establish a JDBC connection to the database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for the
// RowSet to use an already connected connection
// to populate itself.
crs.setConnection(conn);
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the connection that it was provided
// with previously. Once the RowSet is populated, it does not
// close the user-supplied connection.
crs.execute();

// Loop through the data in the RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

使用 execute(Connection) 方法來利用現有的資料庫連線

爲了提高 JDBC Connection 物件的重覆使用性，DB2CachedRowSet 提供了一項機制，可在呼叫 execute 方法時，將已建立的 Connection 物件傳遞至 DB2CachedRowSet。若傳入使用者提供的 Connection 物件，則 DB2CachedRowSet 本身在輸入資料之後，不會中斷此連線物件。

範例：使用 execute(Connection) 方法來利用現有的資料庫連線

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Establish a JDBC connection to the database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the SQL statement that is to be used to
// populate the RowSet.
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method, passing in the connection
```

```
// that should be used. Once the Rowset is populated, it does not
// close the user-supplied connection.
crs.execute(conn);

// Loop through the data in the RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

使用 execute(int) 方法來組合資料庫要求

爲了減少資料庫的工作量，DB2CachedRowSet 提供一項機制，將多個緒的 SQL 陳述式組合成爲一個資料庫處理要求。

範例：使用 execute(int) 方法來組合資料庫要求

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for
// the RowSet to use a DataSource to populate itself.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the DataSource and SQL query
// specified to populate itself with data. Once
// the RowSet is populated, it disconnects from the database.
// This version of the execute method accepts the number of seconds
// that it is willing to wait for its results. By
// allowing a delay, the RowSet can group the requests
// of several users and only process the request against
// the underlying database once.
crs.execute(5);

// Loop through the data in the RowSet.
while (crs.next()) {
  System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

存取 DB2CachedRowSet 資料及游標操作:

本主題提供存取 DB2CachedRowSet 資料及各種游標操作函數的相關資訊。

RowSet 視 ResultSet 方法而定。就許多作業 (如 DB2CachedRowSet 資料存取及游標移動) 而言，在應用程式層次上使用 ResultSet 或 RowSet 並無差別。

存取 DB2CachedRowSet 資料

RowSet 與 ResultSet 存取資料的方式相同。在下列範例中，程式會先建立一個表格，並且利用 JDBC 輸入各種資料類型。當表格備妥之後，就會建立 DB2CachedRowSet，接著將表格中的資訊大量輸入。本範例也用到 RowSet 類別的各種 get 方法。

範例：存取 DB2CachedRowSet 資料

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
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[])
  {
    // Register the driver.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
        catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No need to go any further.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local",

      Statement stmt = conn.createStatement();

      // Clean up previous runs
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Create test table
      stmt.execute("Create table cujosql.test_table (col1 smallint, col2 int, " +
              "col3 bigint, col4 real, col5 float, col6 double, col7 numeric, " +
              "col8 decimal, col9 char(10), col10 varchar(10), col11 date, " +
              "col12 time, col13 timestamp)");
      System.out.println("Table created.");

      // Insert some test rows
      stmt.execute("insert into cujosql.test_table values (1, 1, 1, 1.5, 1.5, 1.5, 1.5, 1.5, 'one', 'one',
                  {d '2001-01-01'}, {t '01:01:01'}, {ts '1998-05-26 11:41:12.123456'})");

      stmt.execute("insert into cujosql.test_table values (null, null, null, null, null, null, null, null,
                  null, null, null, null, null)");
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
      System.out.println("Query executed");

      // Create a new rowset and populate it...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Test with getObject");
      int count = 0;
      while (crs.next()) {
        System.out.println("Row " + (++count));
        for (int i = 1; i <= 13; i++) {
```

```
          System.out.println("  Col " + i + " value " + crs.getObject(i));
        }
      }

      System.out.println("Test with getXXX... ");
      crs.first();
      System.out.println("Row 1");
      System.out.println("  Col 1 value " + crs.getShort(1));
      System.out.println("  Col 2 value " + crs.getInt(2));
      System.out.println("  Col 3 value " + crs.getLong(3));
      System.out.println("  Col 4 value " + crs.getFloat(4));
      System.out.println("  Col 5 value " + crs.getDouble(5));
      System.out.println("  Col 6 value " + crs.getDouble(6));
      System.out.println("  Col 7 value " + crs.getBigDecimal(7));
      System.out.println("  Col 8 value " + crs.getBigDecimal(8));
      System.out.println("  Col 9 value " + crs.getString(9));
      System.out.println("  Col 10 value " + crs.getString(10));
      System.out.println("  Col 11 value " + crs.getDate(11));
      System.out.println("  Col 12 value " + crs.getTime(12));
      System.out.println("  Col 13 value " + crs.getTimestamp(13));
      crs.next();
      System.out.println("Row 2");
      System.out.println("  Col 1 value " + crs.getShort(1));
      System.out.println("  Col 2 value " + crs.getInt(2));
      System.out.println("  Col 3 value " + crs.getLong(3));
      System.out.println("  Col 4 value " + crs.getFloat(4));
      System.out.println("  Col 5 value " + crs.getDouble(5));
      System.out.println("  Col 6 value " + crs.getDouble(6));
      System.out.println("  Col 7 value " + crs.getBigDecimal(7));
      System.out.println("  Col 8 value " + crs.getBigDecimal(8));
      System.out.println("  Col 9 value " + crs.getString(9));
      System.out.println("  Col 10 value " + crs.getString(10));
      System.out.println("  Col 11 value " + crs.getDate(11));
      System.out.println("  Col 12 value " + crs.getTime(12));
      System.out.println("  Col 13 value " + crs.getTimestamp(13));

      crs.close();
    }
    catch (Exception ex) {
      System.out.println("SQLException: " + ex.getMessage());
                 ex.printStackTrace();
    }
  }
}
```

游標操作

RowSet 可以捲動，操作方式就像可捲動的 ResultSet 一樣。在下列範例中，程式會先建立一個表格，並且利用 JDBC 輸入資料。當表格備妥之後，就會建立 DB2CachedRowSet 物件，接著將表格中的資訊大量輸入。本範例也用到各種游標操作函數。

範例：游標操作

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample1
{
  public static void main(String args[])
  {
    // Register the driver.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
```

```
    System.out.println("ClassNotFoundException: " +
               ex.getMessage());
    // No need to go any further.
    System.exit(1);
}

try {
  Connection conn = DriverManager.getConnection("jdbc:db2:*local");

  Statement stmt = conn.createStatement();

  // Clean up previous runs
  try {
    stmt.execute("drop table cujosql.test_table");
  }
    catch (SQLException ex) {
    System.out.println("Caught drop table: " + ex.getMessage());
  }

      // Create a test table
  stmt.execute("Create table cujosql.test_table (col1 smallint)");
  System.out.println("Table created.");

      // Insert some test rows
  for (int i = 0; i < 10; i++) {
    stmt.execute("insert into cujosql.test_table values (" + i + ")");
  }
  System.out.println("Rows inserted");

  ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
  System.out.println("Query executed");

  // Create a new rowset and populate it...
  DB2CachedRowSet crs = new DB2CachedRowSet();
  crs.populate(rs);
  System.out.println("RowSet populated.");

  conn.close();
  System.out.println("RowSet is detached...");

  System.out.println("Use next()");
  while (crs.next()) {
    System.out.println("v1 is " + crs.getShort(1));
  }

  System.out.println("Use previous()");
  while (crs.previous()) {
    System.out.println("value is " + crs.getShort(1));
  }

  System.out.println("Use relative()");
          crs.next();
      crs.relative(9);
  System.out.println("value is " + crs.getShort(1));

      crs.relative(-9);
  System.out.println("value is " + crs.getShort(1));

  System.out.println("Use absolute()");
      crs.absolute(10);
  System.out.println("value is " + crs.getShort(1));
      crs.absolute(1);
  System.out.println("value is " + crs.getShort(1));
      crs.absolute(-10);
  System.out.println("value is " + crs.getShort(1));
      crs.absolute(-1);
  System.out.println("value is " + crs.getShort(1));
```

```
    System.out.println("Test beforeFirst()");
          crs.beforeFirst();
    System.out.println("isBeforeFirst is " + crs.isBeforeFirst());
            crs.next();
    System.out.println("move one... isFirst is " + crs.isFirst());

    System.out.println("Test afterLast()");
        crs.afterLast();
    System.out.println("isAfterLast is " + crs.isAfterLast());
        crs.previous();
    System.out.println("move one... isLast is " + crs.isLast());

    System.out.println("Test getRow()");
        crs.absolute(7);
    System.out.println("row should be (7) and is " + crs.getRow() +
                       " value should be (6) and is " + crs.getShort(1));

    crs.close();
  }
    catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
  }
 }
}
```

變更 DB2CachedRowSet 資料及向資料來源反映變更:

本主題提供變更 DB2CachedRowSet 中的橫列，然後更新基礎資料庫的相關資訊。

DB2CachedRowSet 與標準 ResultSet 介面使用相同的方法來變更 RowSet 物件中的資料。在應用程式層次上，變更 RowSet 的資料與變更 ResultSet 的資料沒有差別。DB2CachedRowSet 的 acceptChanges 方法，可用來向資料庫反映 RowSet 的變更 (此資料庫爲提供資料的來源)。

在 DB2CachedRowSet 中刪除、插入及更新橫列

DB2CachedRowSets 可接受更新。在下列範例中，程式會先建立一個表格，並且利用 JDBC 輸入資料。當表格備妥之後，就會建立 DB2CachedRowSet，接著將表格中的資訊大量輸入。此範例亦使用各種方法來更新 RowSet，並且顯示如何使用 showDeleted 內容，讓應用程式甚至得以在列被刪除後仍能加以提取。此外，此範例還使用 cancelRowInsert 及 cancelRowDelete 方法，讓已插入或刪除的列得以還原。

範例：在 DB2CachedRowSet 中刪除、插入及更新列

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample2
{
  public static void main(String args[])
  {
    // Register the driver.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
        catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
              ex.getMessage());

            // No need to go any further.
      System.exit(1);
```

```
}

try {
  Connection conn = DriverManager.getConnection("jdbc:db2:*local");

  Statement stmt = conn.createStatement();

  // Clean up previous runs
  try {
    stmt.execute("drop table cujosql.test_table");
  }

  catch (SQLException ex) {
    System.out.println("Caught drop table: " + ex.getMessage());
  }

  // Create test table
  stmt.execute("Create table cujosql.test_table (col1 smallint)");
  System.out.println("Table created.");

  // Insert some test rows
  for (int i = 0; i < 10; i++) {
    stmt.execute("insert into cujosql.test_table values (" + i + ")");
  }
  System.out.println("Rows inserted");

  ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
  System.out.println("Query executed");

  // Create a new rowset and populate it...
  DB2CachedRowSet crs = new DB2CachedRowSet();
  crs.populate(rs);
  System.out.println("RowSet populated.");

                      conn.close();
  System.out.println("RowSet is detached...");

  System.out.println("Delete the first three rows");
          crs.next();
        crs.deleteRow();
          crs.next();
        crs.deleteRow();
          crs.next();
        crs.deleteRow();

        crs.beforeFirst();
  System.out.println("Insert the value -10 into the RowSet");
        crs.moveToInsertRow();
        crs.updateShort(1, (short)-10);
        crs.insertRow();
        crs.moveToCurrentRow();

  System.out.println("Update the rows to be the negative of what they now are");
  crs.beforeFirst();
  while (crs.next())
    short value = crs.getShort(1);
    value = (short)-value;
    crs.updateShort(1, value);
    crs.updateRow();
  }

  crs.setShowDeleted(true);

  System.out.println("RowSet is now (value - inserted - updated - deleted)");
  crs.beforeFirst();
  while (crs.next()) {
    System.out.println("value is " + crs.getShort(1) + " " +
```

```
                crs.rowInserted() + " " +
                crs.rowUpdated() + " " +
                crs.rowDeleted());
      }

      System.out.println("getShowDeleted is " + crs.getShowDeleted());

      System.out.println("Now undo the inserts and deletes");
      crs.beforeFirst();
      crs.next();
      crs.cancelRowDelete();
      crs.next();
      crs.cancelRowDelete();
      crs.next();
      crs.cancelRowDelete();
      while (!crs.isLast()) {
        crs.next();
      }

      crs.cancelRowInsert();

      crs.setShowDeleted(false);

      System.out.println("RowSet is now (value - inserted - updated - deleted)");
      crs.beforeFirst();
      while (crs.next()) {
        System.out.println("value is " + crs.getShort(1) + " " +
                crs.rowInserted() + " " +
                crs.rowUpdated() + " " +
                crs.rowDeleted());
      }

      System.out.println("finally show that calling cancelRowUpdates works");
      crs.first();
      crs.updateShort(1, (short) 1000);
      crs.cancelRowUpdates();
      crs.updateRow();
      System.out.println("value of row is " + crs.getShort(1));
      System.out.println("getShowDeleted is " + crs.getShowDeleted());

      crs.close();

    }

    catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

向基礎資料庫反映 DB2CachedRowSet 的變更

在變更 DB2CachedRowSet 之後，必須要 RowSet 物件存在，這些變更才會存在。換言之，變更離線 RowSet 並不會影響資料庫。若要向基礎資料庫反映 RowSet 的變更，請使用 acceptChanges 方法。此方法會指示離線 RowSet 重新建立資料庫連線，並且嘗試在基礎資料庫中反映 RowSet 做過的變更。在 RowSet 建立之後有其他資料庫變更時，若因此造成衝突而無法安全地變更資料庫，則會丟出異常，並且回轉異動。

範例：向基礎資料庫反映 DB2CachedRowSet 的變更

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;
```

```
public class RowSetSample3
{
  public static void main(String args[])
  {
    // Register the driver.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No need to go any further.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Clean up previous runs
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Create test table
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Table created.");

      // Insert some test rows
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Query executed");

      // Create a new rowset and populate it...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Delete the first three rows");
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();
      crs.next();
      crs.deleteRow();

      crs.beforeFirst();
      System.out.println("Insert the value -10 into the RowSet");
      crs.moveToInsertRow();
      crs.updateShort(1, (short)-10);
      crs.insertRow();
      crs.moveToCurrentRow();

      System.out.println("Update the rows to be the negative of what they now are");
      crs.beforeFirst();
      while (crs.next()) {
```

```
        short value = crs.getShort(1);
        value = (short)-value;
        crs.updateShort(1, value);
        crs.updateRow();
      }

      System.out.println("Now accept the changes to the database");

      crs.setUrl("jdbc:db2:*local");
      crs.setTableName("cujosql.test_table");

      crs.acceptChanges();
      crs.close();

      System.out.println("And the database table looks like this:");
      conn = DriverManager.getConnection("jdbc:db2:localhost");
      stmt = conn.createStatement();
      rs = stmt.executeQuery("select col1 from cujosql.test_table");
      while (rs.next ()) {
        System.out.println("Value from table is " + rs.getShort(1));
      }

      conn.close();

    }
      catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

其他 DB2CachedRowSet 功能:

除了具備 ResultSet 在先前幾個範例中的運作方式以外，DB2CachedRowSet 類別還有一些其他的功能，因此用起來更有彈性。其中有一些方法可將整個 Java Database Connectivity (JDBC) RowSet 或其中一部分，轉換成 Java 集合。另外，因爲其中斷連線的本質，DB2CachedRowSet 與 ResultSet 之間沒有絕對的一對一關係。

除了具備 ResultSet 在先前幾個範例中的運作方式以外，DB2CachedRowSet 類別還有一些其他的功能，因此用起來更有彈性。其中有一些方法可將整個 Java Database Connectivity (JDBC) RowSet 或其中一部分，轉換成 Java 集合。另外，因爲其中斷連線的本質，DB2CachedRowSet 與 ResultSet 之間沒有絕對的一對一關係。

利用 DB2CachedRowSet 提供的方法，您可以執行下列作業：

從 DB2CachedRowSet 取得集合

有三種方法，可以從一個 DB2CachedRowSet 物件傳回某些形式的集合。如下所示：

- **toCollection** 傳回含有向量 (亦即，每一個直欄各有一個項目) 的 ArrayList (亦即，每一列各有一個項目)。
- **toCollection(int columnIndex)** 傳回向量，內含特定直欄每一列的值。
- **getColumn(int columnIndex)** 傳回陣列，內含特定直欄每一欄的值。

toCollection(int columnIndex) 與 getColumn(int columnIndex) 的差異，在於 getColumn 方法可以傳回初始類型的陣列。因此，若 columnIndex 代表含有整數資料的直欄，則會傳回整數陣列，而非含有 java.lang.Integer 物件的陣列。

下列範例將說明如何使用這些方法。

範例：從 DB2CachedRowSet 取得集合

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
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[])
  {
    // Register the driver.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
        catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No need to go any further.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");
      Statement stmt = conn.createStatement();

          // Clean up previous runs
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Create test table
      stmt.execute("Create table cujosql.test_table (col1 smallint, col2 smallint)");
      System.out.println("Table created.");

      // Insert some test rows
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ", " + (i + 100) + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
      System.out.println("Query executed");

      // Create a new rowset and populate it...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Test the toCollection() method");
      Collection collection = crs.toCollection();
      ArrayList map = (ArrayList) collection;

      System.out.println("size is " + map.size());
      Iterator iter = map.iterator();
      int row = 1;
      while (iter.hasNext()) {
        System.out.print("row [" + (row++) + "]: \t");

        Vector vector = (Vector)iter.next();
        Iterator innerIter = vector.iterator();
        int i = 1;
        while (innerIter.hasNext()) {
```

```
          System.out.print(" [" + (i++) + "]=" + innerIter.next() + "; \t");
        }
        System.out.println();
      }
      System.out.println("Test the toCollection(int) method");
      collection = crs.toCollection(2);
      Vector vector = (Vector) collection;

      iter = vector.iterator();

      while (iter.hasNext()) {
        System.out.println("Iter: Value is " + iter.next());
      }

      System.out.println("Test the getColumn(int) method");
      Object values = crs.getColumn(2);
      short[] shorts = (short [])values;

      for (int i =0; i < shorts.length; i++) {
        System.out.println("Array: Value is " + shorts[i]);
      }
    }
    catch (SQLException ex) {
      System.out.println("SQLException: " + ex.getMessage());
    }
  }
}
```

建立 RowSet 的複本

createCopy 方法可建立 DB2CachedRowSet 的複本。所有與 RowSet 相關的資料，都會隨著所有控制結構、內容及狀態旗標抄寫一份。

下列範例將說明如何使用此方法。

範例：建立 RowSet 的複本

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;

public class RowSetSample5
{
  public static void main(String args[])
  {
    // Register the driver.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No need to go any further.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Clean up previous runs
```

```
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Create test table
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Table created.");

      // Insert some test rows
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Query executed");

      // Create a new rowset and populate it...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Now some new RowSets from one.");
      DB2CachedRowSet crs2 = crs.createCopy();
      DB2CachedRowSet crs3 = crs.createCopy();

      System.out.println("Change the second one to be negated values");
      crs2.beforeFirst();
      while (crs2.next()) {
        short value = crs2.getShort(1);
        value = (short)-value;
        crs2.updateShort(1, value);
        crs2.updateRow();
      }

      crs.beforeFirst();
      crs2.beforeFirst();
      crs3.beforeFirst();
      System.out.println("Now look at all three of them again");

      while (crs.next()) {
        crs2.next();
        crs3.next();
        System.out.println("Values: crs: " + crs.getShort(1) + ", crs2: " + crs2.getShort(1) +
                    ", crs3: " + crs3.getShort(1));
      }
    }
    catch (Exception ex) {
      System.out.println("SQLException: " + ex.getMessage());
      ex.printStackTrace();
    }

  }

}
```

建立 RowSet 的共用物件

createShared 方法會建立新的 RowSet 物件，內含高階的狀態資訊，可讓兩個 RowSet 物件共用相同的基礎實體資料。

下列範例將說明如何使用此方法。

範例：建立 RowSet 的共用物件

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;

public class RowSetSample5
{
  public static void main(String args[])
  {
    // Register the driver.
    try {
      Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
      System.out.println("ClassNotFoundException: " +
                 ex.getMessage());
      // No need to go any further.
      System.exit(1);
    }

    try {
      Connection conn = DriverManager.getConnection("jdbc:db2:*local");

      Statement stmt = conn.createStatement();

      // Clean up previous runs
      try {
        stmt.execute("drop table cujosql.test_table");
      }
      catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
      }

      // Create test table
      stmt.execute("Create table cujosql.test_table (col1 smallint)");
      System.out.println("Table created.");

      // Insert some test rows
      for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
      }
      System.out.println("Rows inserted");

      ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
      System.out.println("Query executed");

      // Create a new rowset and populate it...
      DB2CachedRowSet crs = new DB2CachedRowSet();
      crs.populate(rs);
      System.out.println("RowSet populated.");

      conn.close();
      System.out.println("RowSet is detached...");

      System.out.println("Test the createShared functionality (create 2 shares)");
      DB2CachedRowSet crs2 = crs.createShared();
      DB2CachedRowSet crs3 = crs.createShared();

      System.out.println("Use the original to update value 5 of the table");
      crs.absolute(5);
      crs.updateShort(1, (short)-5);
```

```
      crs.updateRow();

      crs.beforeFirst();
      crs2.afterLast();

      System.out.println("Now move the cursors in opposite directions of the same data.");

      while (crs.next()) {
        crs2.previous();
        crs3.next();
        System.out.println("Values: crs: " + crs.getShort(1) + ", crs2: " + crs2.getShort(1) +
                    ", crs3: " + crs3.getShort(1));
      }
      crs.close();
      crs2.close();
      crs3.close();
    }
    catch (Exception ex) {
      System.out.println("SQLException: " + ex.getMessage());
                 ex.printStackTrace();
    }
  }
}
```

DB2JdbcRowSet:

DB2JdbcRowSet 是一種連線 RowSet，表示必須搭配基礎的 Connection 物件、 PreparedStatement 物件或 ResultSet 物件的支援，才能使用。其實作方式嚴格遵循 JdbcRowSet 的說明。

使用 DB2JdbcRowSet

因爲 DB2JdbcRowSet 物件支援 Java Database Connectivity (JDBC) 3.0 規格中對於所有 RowSet 所說明的事件，所以在本端資料庫與其他需知道資料庫資料已變更的物件之間，可充當中間物件。

例如，假設在您的工作環境中，有一個主要資料庫和幾台「個人數位助理 (PDA)」，這些 PDA 以無線通訊協定來連接資料庫。您可利用 DB2JdbcRowSet 物件來移至某列，再透過伺服器上執行的主要應用程式來更新此列。更新列的動作會導致 RowSet 元件產生事件。負責將更新傳送給 PDA 的某項服務 (如果有的話)，可將自己登記成爲 RowSet 的「接聽程式」。於是，此服務每次收到 RowSet 事件時，即可產生適當的更新，然後傳送給無線裝置。

如需詳細資訊，請參閱範例：DB2JdbcRowSet 事件。

建立 JDBCRowSet

有數種方法可以建立 DB2JDBCRowSet 物件。每一種方法概述如下：

使用 DB2JdbcRowSet 內容及 DataSource

DB2JdbcRowSet 有一些內容可接受 SQL 查詢及 DataSource 名稱。備妥之後，即可開始使用 DB2JdbcRowSet。以下爲這種方式的範例。將名爲 BaseDataSource 的 DataSource 的參照，假設爲事先已設定的有效 DataSource。

範例：使用 DB2JdbcRowSet 內容及 DataSource

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Create a new DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Set the properties that are needed for
// the RowSet to be processed.
```

```
jrs.setDataSourceName("BaseDataSource");
jrs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This method causes
// the RowSet to use the DataSource and SQL query
// specified to prepare itself for data processing.
jrs.execute();

// Loop through the data in the RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventually, close the RowSet.
    jrs.close();
```

使用 DB2JdbcRowSet 內容及 JDBC URL

DB2JdbcRowSet 有一些內容可接受 SQL 查詢及 JDBC URL。備妥之後，即可開始使用 DB2JdbcRowSet。以下爲這種方式的範例：

範例：使用 DB2JdbcRowSet 內容及 JDBC URL

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Create a new DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Set the properties that are needed for
// the RowSet to be processed.
jrs.setUrl("jdbc:db2:*local");
jrs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the URL and SQL query specified
// previously to prepare itself for data processing.
jrs.execute();

// Loop through the data in the RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventually, close the RowSet.
jrs.close();
```

使用 setConnection(Connection) 方法來利用現有的資料庫連線

爲了提高 JDBC Connection 物件的重覆使用性，DB2JdbcRowSet 可讓您將已建立的連線傳給 DB2JdbcRowSet。在呼叫 execute 方法時，DB2JdbcRowSet 就會使用此連線來備妥自己。

範例：使用 setConnection 方法

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Establish a JDBC Connection to the database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a new DB2JdbcRowSet.
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Set the properties that are needed for
// the RowSet to use an established connection.
jrs.setConnection(conn);
```

```
    jrs.setCommand("select col1 from cujosql.test_table");

    // Call the RowSet execute method. This causes
    // the RowSet to use the connection that it was provided
    // previously to prepare itself for data processing.
    jrs.execute();

    // Loop through the data in the RowSet.
    while (jrs.next()) {
        System.out.println("v1 is " + jrs.getString(1));
    }

    // Eventually, close the RowSet.
    jrs.close();
```

存取資料及游標移動

游標位置的操作及透過 DB2JdbcRowSet 對資料庫資料的存取，皆由基礎的 ResultSet 物件來處理。ResultSet 物件可完成的作業，亦可透過 DB2JdbcRowSet 物件來完成。

變更資料及向基礎資料庫反映變更

透過 DB2JdbcRowSet 來更新資料庫的這項支援，完全由基礎的 ResultSet 物件來處理。ResultSet 物件可完成的作業，亦可透過 DB2JdbcRowSet 物件來完成。

DB2JdbcRowSet 事件:

只要是對其他元件有影響的狀況，所有 RowSet 實作方式皆支援這些狀況的事件處理。當應用程式元件發生某事件時，這項支援可讓元件彼此「溝通」。例如，透過 RowSet 更新資料庫橫列時，您會看見圖形式使用者介面 (GUI) 表格自動更新。

在下列範例中，main 方法負責更新 RowSet，且是您的核心應用程式。接聽程式則屬於現場離線用戶端所用的無線伺服器。上述兩者有機會結合爲一，卻不會讓兩個程序的程式碼混雜在一起。設計 RowSet 事件支援的目的，主要是爲了透過資料庫資料來更新 GUI，但它在這種應用問題上也表現的很好。

範例：DB2JdbcRowSet 事件

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2JdbcRowSet;

public class RowSetEvents {
    public static void main(String args[])
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                                                          ex.getMessage());
                  // No need to go any further.
            System.exit(1);
        }


        try {
            // Obtain the JDBC Connection and Statement needed to set
            // up this example.
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");
            Statement stmt = conn.createStatement();
```

```
            // Clean up any previous runs.
            try {
                stmt.execute("drop table cujosql.test_table");
            } catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Create the test table
            stmt.execute("Create table cujosql.test_table (col1 smallint)");
            System.out.println("Table created.");

            // Populate the table with data.
            for (int i = 0; i < 10; i++) {
                stmt.execute("insert into cujosql.test_table values (" + i + ")");
            }
            System.out.println("Rows inserted");

            // Remove the setup objects.
            stmt.close();
            conn.close();

            // Create a new rowset and set the properties need to
            // process it.
            DB2JdbcRowSet jrs = new DB2JdbcRowSet();
            jrs.setUrl("jdbc:db2:*local");
            jrs.setCommand("select col1 from cujosql.test_table");
            jrs.setConcurrency(ResultSet.CONCUR_UPDATEABLE);

            // Give the RowSet object a listener. This object handles
            // special processing when certain actions are done on
            // the RowSet.
            jrs.addRowSetListener(new MyListener());

            // Process the RowSet to provide access to the database data.
            jrs.execute();

            // Cause a few cursor change events. These events cause the cursorMoved
            // method in the listener object to get control.
            jrs.next();
            jrs.next();
            jrs.next();

            // Cause a row change event to occur. This event causes the rowChanged method
            // in the listener object to get control.
            jrs.updateShort(1, (short)6);
            jrs.updateRow();

            // Finally, cause a RowSet change event to occur. This causes the
            // rowSetChanged method in the listener object to get control.
            jrs.execute();

            // When completed, close the RowSet.
            jrs.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}


/**
 * This is an example of a listener. This example prints messages that show
 * how control flow moves through the application and offers some
 * suggestions about what might be done if the application were fully implemented.
 */
class MyListener
```

```
implements RowSetListener {
    public void cursorMoved(RowSetEvent rse) {
        System.out.println("Event to do:  Cursor position changed.");
        System.out.println("  For the remote system, do nothing ");
        System.out.println("  when this event happened. The remote view of the data");
        System.out.println("  could be controlled separately from the local view.");
        try {
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("row is " + rs.getRow() + ". \n\n");
        } catch (SQLException e) {
            System.out.println("To do:  Properly handle possible problems.");
        }
    }

    public void rowChanged(RowSetEvent rse) {
        System.out.println("Event to do:  Row changed.");
        System.out.println("  Tell the remote system that a row has changed. Then,");
        System.out.println("  pass all the values only for that row to the  ");
        System.out.println("  remote system.");
        try {
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("new values are " + rs.getShort(1) + ". \n\n");
        } catch (SQLException e) {
              System.out.println("To do:  Properly handle possible problems.");
        }
    }

    public void rowSetChanged(RowSetEvent rse) {
        System.out.println("Event to do:  RowSet changed.");
        System.out.println("  If there is a remote RowSet already established, ");
        System.out.println("  tell the remote system that the values it ");
        System.out.println("  has should be thrown out. Then, pass all ");
        System.out.println("  the current values to it.\n\n");
    }
}
```

IBM Developer Kit for Java JDBC 驅動程式的效能要訣

依照設計，IBM Developer Kit for Java JDBC 驅動程式是一種使用於資料庫的高效能 Java 介面。然而，若要獲得最好的效能，您建置的應用程式也必須充份利用 JDBC 驅動程式的強大功能。下列秘訣可當作良好的 JDBC 程式設計慣例。大部份並非僅針對原有的 JDBC 驅動程式。因此，遵循這些準則所編寫的應用程式，只要是採用 JDBC 驅動程式，而非原有的 JDBC 驅動程式，執行效能也應該不錯。

避免 SELECT * SQL 查詢

SELECT * FROM... 是 SQL 中表示查詢的常見方式。但通常您不需要查詢所有欄位。對於每一個傳回的直欄，JDBC 驅動程式都必須另外執行工作來連結及傳回橫列。即使應用程式一定不會用到某個特定的直欄，JDBC 驅動程式仍然必須予以注意，並且預留使用空間。若表格尚未使用的直欄很少，則額外負荷並不明顯。但若有大量未使用的直欄，額外負荷就值得注意。最好的解決方法就是個別列出應用程式所需的直欄，例如：

```
SELECT COL1, COL2, COL3 FROM...
```

使用 getXXX(int) 來代替 getXXX(String)

使用 ResultSet getXXX 方法來傳入數值，而不要傳入直欄名稱。雖然直接使用直欄名稱來代替數值常數似乎很方便，但資料庫本身卻只知道如何處理直欄索引。因此，您用直欄名稱來呼叫的每一個 getXXX 方法，都必須透過 JDBC 驅動程式解析之後，才能傳遞到資料庫。因爲 getXXX 方法通常在迴圈內呼叫，可能執行上百萬次，而這些微小的額外負荷，將會迅速累積。

避免對 Java 初始類型執行 getObject 呼叫

從資料庫取得初始類型的值 (int、long、float 等) 時，採用初始類型專用的 get 方法 (getInt、getLong、getFloat)，將比使用 getObject 更快。getObject 呼叫也一樣會執行初始類型的取得動作，但會再另外建立物件傳回給您。這通常在迴圈內執行，有可能建立上百萬個很快就消失的物件。對初始指令使用 getObject 還有另一項缺點，就是經常會啓動垃圾收集器，導致效能更低落。

使用 PreparedStatement 來代替 Statement

若您編寫的 SQL 陳述式曾多次被使用，則以 PreparedStatement 而非 Statement 物件來執行，效能較佳。每次執行陳述式時，就會進入兩個步驟的過程：備妥陳述式，然後處理陳述式。使用備妥陳述式時，陳述式僅於建構時進行準備，而非每一次執行就準備一次。雖然公認 PreparedStatement 的執行速度比 Statement 更快，但程式設計師卻經常疏忽這項優點。鑒於 PreparedStatement 所提供的效能提升，聰明的您，在應用程式的設計中應該儘可能採用 (請參閱下面的第 164 頁的『考慮使用 PreparedStatement 儲存區作業』)。

避免 DatabaseMetaData 呼叫

請留意有些 DatabaseMetaData 呼叫的代價很高。尤其是 getBestRowIdentifier、getCrossReference、getExportedKeys 及 getImportedKeys 方法，成本都非常高。有些 DataBaseMetaData 呼叫涉及系統層次表格之間的複雜合併條件。眞正需要這些資訊才使用，千萬別貪圖方便。

對應用程式使用正確的確定層次

JDBC 提供數個確定層次，決定系統的多重異動彼此之間如何相互影響 (如需詳細資訊，請參閱異動)。預設爲使用最低確定層次。這表示異動可跨越確定界限，得知彼此之間部份工作情形。但這樣可能導致某些資料庫反常狀況。有些程式設計師會提高確定層次，以免擔心發生這些反常狀況。請注意，太高的確定層次，可能造成資料庫發生更多粗略鎖定而停滯不前。這會限制系統的並行處理能力，造成一些應用程式的執行效能嚴重下降。通常，只要應用程式事前設計得當，就不太可能發生反常狀況。請花時間瞭解您究竟想完成什麼目標，將異動隔離層次降至您可安全使用的最低層次。

考慮以 Unicode 來儲存資料

Java 要求其處理的所有字元資料 (String) 一律爲 Unicode。因此，對於沒有 Unicode 資料的任何表格，當資料庫內放入及取出資料時，就需要 JDBC 驅動程式來轉換資料。若表格已經是 Unicode，則 JDBC 驅動程式不必轉換資料，從資料庫取出資料的速度會更快。請注意，Unicode 的資料可能不適用於非 Java 應用程式，因爲這種應用程式不知如何處理 Unicode。也請記得，非字元資料不會執行得比較快，因爲這種資料本來就不會有任何轉換。還有另一項考量，以 Unicode 儲存的資料，佔用空間是單位元組資料的兩倍。然而，若有許多字元直欄會讀取許多次，則以 Unicode 來儲存資料的效能明顯倍增。

使用儲存程序

Java 支援使用儲存程序。只要讓 JDBC 驅動程式執行靜態 SQL，而非動態 SQL，儲存程序即可執行的更快。請勿對程式中執行的每一個 SQL 陳述式都各自建立一個儲存程序。最好儘量建立一個儲存程序來執行整組 SQL 陳述式。

使用 BigInt 來代替 Numeric 或 Decimal

不要使用小數位數爲 0 的 Numeric 或 Decimal 欄位，請改用 BigInt 資料類型。BigInt 會直接轉換成 Java 初始類型 Long，而 Numeric 或 Decimal 資料類型則會轉換成 String 或 BigDecimal 物件。『避免 DatabaseMetaData 呼叫』已指出，使用初始資料類型，一定比採用需建立物件的類型更好。

明確關閉使用完畢的 JDBC 資源

應用程式應該明確關閉不再需要的 ResultSets、Statement 及 Connection。如此才能以最佳方式來清除資源，進而提高效能。此外，未明確關閉的資料庫資源，可能導致資料庫洩漏及無謂地保留資料庫鎖定。最後造成應用程式失敗，或應用程式的並行處理能力下降。

使用連線儲存區作業

連線儲存區作業是讓多位使用者重覆使用 JDBC Connection 物件的一種策略，而非每一個使用者都要求建立自己的 Connection 物件。建立 Connection 物件的代價很高。所以，在注重效能的應用程式中，應該共用連線儲存區作業，而非讓每一位使用者建立新的連線。許多產品 (例如 WebSphere) 都提供 Connection 儲存區作業支援，使用者只需進行少許工作就可使用。若您不想使用支援連線儲存區作業的產品，或喜歡建置自己的儲存區作業機制來掌握儲存區作業的運作及執行方式，也沒有什麼困難。

考慮使用 PreparedStatement 儲存區作業

Statement 儲存區作業的運作類似 Connection 儲存區作業。但不止是將「連線」放入儲存區作業，更是將含有 Connection 及 PreparedStatement 的物件放入儲存區作業。然後，即可擷取此物件，從中存取您要使用的特定陳述式。如此即可大幅地提高效能。

使用有效率的 SQL

因爲 JDBC 建置於 SQL 之上，所以任何能夠提升 SQL 效率的成果，也一樣會提升 JDBC 的效率。因此，JDBC 也同樣享受到最佳化查詢、精確選擇的索引及良好 SQL 設計的其他好處。

使用 IBM Developer Kit for Java DB2 SQLJ 支援來存取資料庫

DB2 Java 結構化查詢語言 (SQLJ) 支援係基於 SQLJ ANSI 標準。DB2 SQLJ 支援內含在 IBM Developer Kit for Java 中。DB2 SQLJ 支援可讓您在 Java 應用程式中建立、建置及執行內含的 SQL。

IBM Developer Kit for Java 提供的 SQLJ 支援包含 SQLJ 執行時間類別，封裝在 /QIBM/ProdData/Java400/ext/runtime.zip 中。

SQLJ 設定

您必須先將要使用 SQLJ 的伺服器備妥，才能夠在伺服器的 Java 應用程式中使用 SQLJ。如需相關資訊，請參閱第 175 頁的『設定伺服器來使用 SQLJ』。

SQLJ 工具

IBM Developer Kit for Java 提供的 SQLJ 支援亦包含下列工具：

- SQLJ 轉換程式 `sqlj`，可以將 SQLJ 程式中內含的 SQL 陳述式置換成 Java 來源陳述式，並可產生已序列化的設定檔，其包含 SQLJ 程式中有關 SQLJ 作業的資訊。
- 「DB2 SQLJ 設定檔自訂程式」`db2profc`，可以前置編譯儲存於產生之設定檔內的 SQL 陳述式，並在 DB2 資料庫內產生資料包。
- 「DB2 SQLJ 設定檔印表機」`db2profp`，可以將 DB2 自訂設定檔的內容列印成純文字。
- SQLJ 設定檔審核員安裝程式 `profdb`，可以在現有的二進位設定檔集內，安裝及解除安裝除錯類別審核員。
- SQLJ 設定檔轉換工具 `profconv`，可以將已序列化的設定檔實例轉換成 Java 類別格式。

註: 這些工具都必須在「Qshell 直譯器」中執行。

DB2 SQLJ 限制

使用 SQLJ 建立 DB2 應用程式時，請留意下列限制：

- 在發出 SQL 陳述式方面，DB2 SQLJ 支援必須符合標準的 DB2 Universal Database™ 限制。
- DB2 SQLJ 設定檔自訂程式只能在本端資料庫連線相關的設定檔上執行。
- SQLJ Reference Implementation 需要 JDK 1.1 或更高版本。如需執行 Java Development Kit 多重版本的相關資訊，請參閱多重 Java Development Kit (JDK) 的支援。

如需在 Java 應用程式中使用 SQL 的相關資訊，請參閱在 Java 應用程式中內含 SQL 陳述式與編譯及執行 SQLJ 程式。

Java 適用的結構化查詢語言設定檔

轉換 SQLJ 來源檔時，「SQLJ 轉換程式」sqlj 會產生設定檔。此設定檔爲序列化的二進位檔。 這就是這些檔案的副檔名爲 .ser 的原因。這些檔案含有相關 SQLJ 來源檔裡的 SQL 陳述式。

若要從 SQLJ 原始程式碼產生設定檔，請在 .sqlj 檔案上執行 SQLJ 轉換程式 sqlj。

如需詳細資訊，請參閱編譯及執行 SQLJ 程式。

Java 結構化查詢語言 (SQLJ) 轉換程式 (sqlj)

SQLJ 轉換程式 sqlj 可以產生序列化的設定檔，內含位於 SQLJ 程式中有關 SQL 作業的資訊。SQLJ 轉換程式會用到 /QIBM/ProdData/Java400/ext/translator.zip 檔案。

如需設定檔的相關資訊，請跟隨此鏈結：設定檔。

使用 DB2 SQLJ 設定檔自訂程式 db2profc，前置編譯設定檔內的 SQL 陳述式

使用「DB2 SQLJ 設定檔自訂程式」db2profc，可以讓 Java 應用程式存取資料庫時更有效率。

「DB2 SQLJ 設定檔自訂程式」的功能如下：

- 前置編譯設定檔內儲存的 SQL 陳述式，並且在 DB2 資料庫內產生資料包。
- 將 SQL 陳述式置換成已建立的資料包內相關陳述式的參照，藉此自訂 SQLJ 設定檔。

若要前置編譯設定檔內的 SQL 陳述式，請在 Qshell 指令提示中鍵入下列指令：

```
db2profc MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 爲您要前置編譯的設定檔名稱。

DB2 SQLJ 設定檔自訂程式的用法及語法

db2profc[*options*] <*SQLJ_profile_name*>

其中，*SQLJ_profile_name* 爲您要列印的設定檔名稱，*options* 爲您要使用的選項清單。

db2profp 可用的選項如下：

- -URL=<*JDBC_URL*>
- -user=<*username*>
- -password=<*password*>
- -package=<*library_name/package_name*>
- -commitctrl=<*commitment_control*>

- -datefmt=<*date_format*>
- -datesep=<*date_separator*>
- -timefmt=<*time_format*>
- -timesep=<*time_separator*>
- -decimalpt=<*decimal_point*>
- -stmtCCSID=<*CCSID*>
- -sorttbl=<*library_name/sort_sequence_table_name*>
- -langID=<*language_identifier*>

以下說明這些選項：

-URL=<*JDBC_URL***>**

其中，*JDBC_URL* 爲 JDBC 連線的 URL。URL 的語法爲：

```
"jdbc:db2:systemName"
```

如需相關資訊，請參閱使用 IBM Developer Kit for Java JDBC 驅動程式來存取 iSeries 資料庫。

-user=<*username***>**

其中，*username* 爲使用者名稱。預設值爲登入本端連線的現行使用者的使用者 ID。

-password=<*password***>**

其中，*password* 爲密碼。預設值爲登入本端連線的現行使用者的密碼。

-package=<*library name/package name***>**

其中，*library name* 爲存放資料包的檔案庫，*package name* 爲所要產生的資料包名稱。預設檔案庫名稱爲 QUSRSYS。預設資料包名稱依設定檔名稱而產生。資料包名稱的最大長度爲 10 個字元。 因爲 SQLJ 設定檔名稱一律超過 10 個字元，所以建構的預設資料包名稱不同於設定檔名稱。設定檔名稱前面的字母與設定檔機碼結合起來，就構成預設資料包名稱。若設定檔機碼超過 10 個字元，則直接採用設定檔機碼的後 10 個字元作爲預設資料包名稱。例如，下表顯示幾個設定檔名稱及其預設資料包名稱：

設定檔名稱	預設資料包名稱
App_SJProfile0	App_SJPro0
App_SJProfile01234	App_S01234
App_SJProfile012345678	A012345678
App_SJProfile01234567891	1234567891

-commitctrl=<*commitment_control***>**

其中，*commitment_control* 爲您要的確定控制層次。確定控制的有效字元值如下：

值	定義
C	*CHG。錯誤讀取、非連續讀取及幻象讀取，都有可能。
S	*CS。不可能錯誤讀取，但有可能非連續讀取及幻象讀取。
A	*ALL。不可能錯誤讀取及非連續讀取，但有可能幻象讀取。
N	*NONE。錯誤讀取、非連續讀取及幻象讀取，都不可能。此爲預設值。

-datefmt=<*date_format*>

其中，*date_format* 爲您要的日期格式化類型。日期格式的有效值如下：

值	定義
USA	IBM USA 標準 (mm.dd.yyyy、hh:mm a.m.、hh:mm p.m.)
ISO	國際標準組織 (yyyy-mm-dd、hh.mm.ss)。此爲預設值。
EUR	IBM 歐洲標準 (dd.mm.yyyy、hh.mm.ss)
JIS	日本工業標準基督紀元 (yyyy-mm-dd、hh:mm:ss)
MDY	月/日/年 (mm/d/yy)
DMY	日/月/年 (dd/mm/yy)
YMD	年/月/日 (yy/mm/dd)
JUL	羅馬曆 (yy/ddd)

存取日期結果直欄時會用到日期格式。所有輸出日期欄位皆以指定的格式傳回。對於輸入日期字串，則以指定的值來判斷日期是否指定爲有效格式。預設值爲 ISO。

-datesep=<*date_separator*>

其中，*date_separator* 爲您要使用的分隔字元類型。存取日期結果直欄時會用到日期分隔字元。日期分隔字元的有效值如下：

值	定義
/	使用斜線。
.	使用句點。
,	使用逗點。
-	使用破折號。此爲預設值。
空白	使用空格。

-timefmt=<*time_format*>

其中，*time_format* 爲用來顯示時間欄位的格式。存取日期結果直欄時會用到時間格式。對於輸入的時間字串，則以指定的值來判斷時間是否指定爲有效格式。時間格式的有效值如下：

值	定義
USA	IBM USA 標準 (mm.dd.yyyy、hh:mm a.m.、hh:mm p.m.)
ISO	國際標準組織 (yyyy-mm-dd、hh.mm.ss)。此爲預設值。
EUR	IBM 歐洲標準 (dd.mm.yyyy、hh.mm.ss)
JIS	日本工業標準基督紀元 (yyyy-mm-dd、hh:mm:ss)
HMS	時/分/秒 (hh:mm:ss)

-timesep=<*time_separator*>

其中，*time_separator* 爲用來存取時間結果直欄的字元。時間分隔字元的有效值如下：

值	定義
:	使用冒號。
.	使用句點。此爲預設值。
,	使用逗點。
空白	使用空格。

-decimalpt=<*decimal_point*>

其中 *decimal_point* 爲您要使用的小數點。 小數點用於 SQL 陳述式的數值常數中。小數點的有效值如下：

值	定義
.	使用句點。此爲預設值。
,	使用逗點。

-stmtCCSID=<*CCSID*>

其中，*CCSID* 爲準備放入資料包內的 SQL 陳述式的編碼字集 ID。預設值爲自訂期間的工作值。

-sorttbl=<*library_name/sort_sequence_table_name*>

其中，*library_name/sort_sequence_table_name* 爲您要使用的排序順序表的位置及表格名稱。排序順序表用於 SQL 陳述式的字串比較。檔案庫名稱及排序順序表名稱最長皆爲 10 個字元。 預設值取決於自訂期間的工作。

-langID=<*language_identifier*>

其中，*language identifier* 爲您要使用的語言 ID。語言 ID 的預設值取決於自訂期間當時的工作。語言 ID 與排序順序表一起使用。

如需這些欄位的詳細資訊，請參閱 DB2 for iSeries SQL Programming Concepts, SC41-5611  。

列印 DB2 SQLJ 設定檔 (db2profp 及 profp) 的內容

「DB2 SQLJ 設定檔印表機」db2profp 可以將 DB2 自訂設定檔的內容列印爲純文字。「設定檔印表機」profp 可以將 SQLJ 轉換程式所產生的設定檔內容列印成純文字。

若要將 SQLJ 轉換程式所產生的設定檔內容列印爲純文字，請使用 profp 公用程式：

```
profp MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 爲您要列印的設定檔名稱。

若要將 DB2 自訂版本的設定檔內容列印爲純文字，請按如下方式使用 db2profp 公用程式：

```
db2profp MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 爲您要列印的設定檔名稱。

附註：若在未自訂的設定檔上執行 db2profp，則會指出此設定檔尚未經過自訂。若在自訂的設定檔上執行 profp，卻又直接顯示設定檔的內容，看不出來經過自訂。

DB2 SQLJ 設定檔印表機的用法及語法：

```
db2profp [options] <SQLJ_profile_name>
```

其中，*SQLJ_profile_name* 爲您要列印的設定檔名稱，*options* 爲您要使用的選項清單。

db2profp 可用的選項如下：

-URL=<*JDBC_URL*>

其中，*JDBC_URL* 爲您要連接的 URL。如需相關資訊，請參閱使用 IBM Developer Kit for Java JDBC 驅動程式來存取 iSeries 資料庫。

-user=<*username*>

其中，*username* 爲使用者設定檔的使用者名稱。

-password=<*password*>

其中，*password* 爲使用者設定檔的密碼。

SQLJ 設定檔審核員安裝程式 (profdb)

SQLJ 設定檔審核員安裝程式 (profdb) 可以安裝及解除安裝除錯類別審核員。 除錯類別審核員可以安裝在現有的二進位設定檔集內。安裝除錯類別審核員之後， 就會記載所有在應用程式執行期間執行的 RTStatement 及 RTResultSet 呼叫。 可以記載至檔案或標準輸出中。然後，就可以檢查日誌來驗證應用程式的行爲，並追蹤其錯誤。 請注意，僅執行期間對基礎 RTStatement 及 RTResultSetcall 介面的呼叫，才會受到審核。

若要安裝除錯類別審核員，請在 Qshell 指令提示中輸入下列指令：

```
profdb MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 爲「SQLJ 轉換程式」產生的設定檔名稱。

若要解除安裝除錯類別審核員，請在 Qshell 指令提示中輸入下列指令：

```
profdb -Cuninstall MyClass_SJProfile.ser
```

其中，*MyClass_SJProfile0.ser* 爲「SQLJ 轉換程式」產生的設定檔名稱。

使用 SQLJ 設定檔轉換工具 (profconv)，將已序列化的設定檔實例轉換成 Java 類別格式。

SQLJ 設定檔轉換工具 (profconv) 可將已序列化的設定檔實例轉換成 Java 類別格式。由於某些瀏覽器不支援從 Applet 相關的資源檔中載入已序列化的物件，所以才需要 profconv 工具。 請執行 profconv 公用程式來進行轉換。

若要執行 profconv 公用程式，請在 Qshell 指令行鍵入下列指令：

```
profconv MyApp_SJProfile0.ser
```

其中，*MyApp_SJProfile0.ser* 爲您要轉換的設定檔實例名稱。

profconv 工具會呼叫 sqlj -ser2class。相關的指令行選項，請參閱 sqlj。

在 Java 應用程式中內含 SQL 陳述式

SQLJ 的靜態 SQL 陳述式位於 SQLJ 子句內。SQLJ 子句的開頭爲 #sql，結尾是分號 (;) 字元。

在 Java 應用程式中建立任何 SQLJ 子句之前，請先匯入下列資料包：

- import java.sql.*;
- import sqlj.runtime.*;
- import sqlj.runtime.ref.*;

最簡單的 SQLJ 子句爲可以處理的子句，由記號 #sql 再接著大括弧包住的 SQL 陳述式組成。例如，Java 陳述式出現在合理位置時，就可能出現下列 SQLJ 子句：

```
#sql { DELETE FROM TAB };
```

上述範例會刪除 TAB 表格內所有的列。

註: 如需編譯及執行 SQLJ 應用程式的相關資訊，請參閱編譯及執行 SQLJ 程式

在 SQLJ 處理子句中，大括弧內的記號一律爲 SQL 記號或主變數。 所有主變數皆以冒號 (:) 字元隔開。SQL 記號絕不會出現在 SQLJ 處理子句的大括弧之外。 例如，下列 Java 方法會將引數插入 SQL 表格內：

```
public void insertIntoTAB1 (int x, String y, float z) throws SQLException
{
        #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

方法主體由一個含有主變數 x、y、z 的 SQLJ 處理子句組成。如需主變數的詳細資訊，請參閱 SQLJ 的主變數。

SQL 記號通常不區分大小寫 (雙引號區隔的 ID 除外)，可以寫成大寫、小寫或大小寫混合。 但 *Java* 記號則區分大小寫。爲了在範例中更清楚地表示，不區分大小寫的 SQL 記號一律使用大寫，Java 記號則爲小寫或大小寫混合。本主題內，小寫 `null` 用於代表 Java 空值，大寫 `NULL` 用於代表 SQL 空值。

SQLJ 程式內可能出現下列 SQL 建構類型：

- 查詢，例如，SELECT 陳述式及表示式。
- SQL 資料變更陳述式 (DML)，例如，INSERT、UPDATE、DELETE。
- 資料陳述式，例如，FETCH、ELECT..INTO。
- 交易控制陳述式，例如，COMMIT、ROLLBACK 等。
- 資料定義語言 (DDL，亦稱爲「綱目操作語言」) 陳述式，例如，CREATE、DROP、ALTER。
- 呼叫儲存程序，例如，CALL MYPROC(:x, :y, :z)。
- 呼叫儲存函數，例如，VALUES(MYFUN(:x))。

Java 適用的結構化查詢語言的主變數:

內含式 SQL 陳述式的引數須透過主變數來傳遞。主變數是主語言的變數，可出現在 SQL 陳述式中。

主變數最多有三個部份：

- 冒號 (:) 字首。
- Java 主變數，是參數、變數或欄位的 Java ID。
- 選用性參數模式 ID。

  此模式 ID 的有效值如下：
  IN、OUT 或 INOUT。

Java ID 的求值不會對 Java 程式產生不良影響，所以在爲取代 SQLJ 子句而產生的 Java 程式碼內可能出現多次。

下列查詢包含主變數 `:x`。此主變數爲 Java 變數、欄位或參數 `x`，可見範圍含括整個查詢。

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

範例：在 Java 應用程式中內含 SQL 陳述式:

以下 SQLJ 應用程式範例 App.sqlj，將使用靜態 SQL 在 DB2 範例資料庫的 EMPLOYEE 表格內擷取及更新資料。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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
{

  /*********************
      **  Register Driver **
   *********************/

  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;

     // URL is jdbc:db2:dbname
     String url = "jdbc:db2:sample";

     DefaultContext ctx = DefaultContext.getDefaultContext();
     if (ctx == null)
     {
       try
       {
         // connect with default id/password
         Connection con = DriverManager.getConnection(url);
         con.setAutoCommit(false);
         ctx = new DefaultContext(con);
       }
       catch (SQLException e)
       {
         System.out.println("Error: could not get a default context");
         System.err.println(e) ;
         System.exit(1);
       }
       DefaultContext.setDefaultContext(ctx);
     }

     // retrieve data from the database
     System.out.println("Retrieve some data from the database.");
     #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

     // display the result set
     // cursor1.next() returns false when there are no more rows
     System.out.println("Received results:");
     while (cursor1.next()) // 3
     {
       str1 = cursor1.empno(); // 4
       str2 = cursor1.firstnme();
```

```
      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor1.close(); // 9

    // retrieve number of employee from the database
    #sql { SELECT count(*) into :count1 FROM employee }; // 5
    if (1 == count1)
      System.out.println ("There is 1 row in employee table");
    else
      System.out.println ("There are " + count1
                          + " rows in employee table");

    // update the database
    System.out.println("Update the database.");
    #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

    // retrieve the updated data from the database
    System.out.println("Retrieve the updated data from the database.");
    str1 = "000010";
    #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

    // display the result set
    // cursor2.next() returns false when there are no more rows
    System.out.println("Received results:");
    while (true)
    {
      #sql { FETCH :cursor2 INTO :str2 }; // 7
      if (cursor2.endFetch()) break; // 8

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor2.close(); // 9

    // rollback the update
    System.out.println("Rollback the update.");
    #sql { ROLLBACK work };
    System.out.println("Rollback done.");
  }
  catch( Exception e )
  {
    e.printStackTrace();
  }
 }
}
```

[1]宣告疊代子。此區段宣告兩種疊代子：

- App_Cursor1：宣告直欄資料類型及名稱，然後根據直欄名稱傳回直欄的值 (直欄的具名連結)。
- App_Cursor2：宣告直欄資料類型，然後根據直欄位置傳回直欄的值 (直欄的定位連結)。

[2]起始設定疊代子。使用查詢結果來起始設定疊代子物件 cursor1。查詢結果會儲存在 cursor1 內。

[3]將疊代子移至下一橫列。若已無其他列可擷取，cursor1.next() 方法會傳回布林 false。

[4]移動資料。具名 accessor 方法 empno() 會傳回現行列上 empno 直欄的值。 具名 accessor 方法 firstnme() 會傳回現行列上 firstnme 直欄的值。

[5]SELECT 資料傳入主變數。SELECT 陳述式會將表格列數傳入主變數 count1 內。

[6]起始設定疊代子。使用查詢結果來起始設定疊代子物件 cursor2。查詢結果會儲存在 cursor2 內。

[7]擷取資料。FETCH 陳述式會從結果表格中，將 ByPos 游標中宣告的第一欄的現行值傳回主變數 str2 內。

[8]檢查 FETCH.INTO 陳述式是否成功。若疊代子不在某列上，亦即，若前次試圖擷取某列失敗，endFetch() 方法會傳回布林 true。若前次試圖擷取某列成功，endFetch() 方法會傳回 false。呼叫 next() 方法時，DB2 會試圖提取一個橫列。FETCH...INTO 陳述式會隱含地呼叫 next() 方法。

[9]關閉疊代子。close() 方法會釋放疊代子佔用的任何資源。應該明確地關閉疊代子，確保能夠及時地釋放系統資源。

如需本範例的背景資訊，請參閱在 Java 應用程式中內含 SQL 陳述式。

編譯及執行 SQLJ 程式

如果 Java 程式有內含的 SQLJ 陳述式，則需遵循特殊的程序來對其進行編譯並執行。

如果 Java 程式有內含的 SQLJ 陳述式，則需遵循特殊的程序來對其進行編譯並執行。

1. 設定伺服器來使用 SQLJ。
2. 對於內含 SQL 的 Java 原始程式碼，請使用 SQLJ 轉換程式 `sqlj` 來產生 Java 原始程式碼及相關的設定檔。每一個連線會產生一個設定檔。

   例如，鍵入下列指令：

   ```
   sqlj MyClass.sqlj
   ```

   其中，*MyClass.sqlj* 爲 SQLJ 檔案的名稱。

   此範例中，SQLJ 轉換程式會產生 MyClass.java 原始程式碼檔案及可能的相關設定檔。相關的設定檔命名爲 MyClass_SJProfile0.ser、MyClass_SJProfile1.ser、MyClass_SJProfile2.ser，依此類推。

   註: 除非以 `-compile=false` 子句明確地關閉編譯選項，否則 SQLJ 轉換程式會自動將轉換後的 Java 原始程式碼編譯成類別檔案。
3. 使用 「SQLJ 設定檔自訂程式」工具 `db2profc`，在產生的設定檔上安裝「DB2 SQLJ 自訂程式」，並且在本端系統上建立 DB2 資料包。

   例如，鍵入下列指令：

   ```
   db2profc MyClass_SJProfile0.ser
   ```

   其中，*MyClass_SJProfile0.ser* 爲執行「DB2 SQLJ 自訂程式」的設定檔名稱。

   註: 此爲選用性步驟，但建議您執行以提升執行時間的效能。
4. 執行 Java 類別檔案，就像執行任何其他 Java 類別檔案一樣。

   例如，鍵入下列指令：

   ```
   java MyClass
   ```

   其中，*MyClass* 爲 Java 類別檔案的名稱。

相關概念

第 169 頁的『在 Java 應用程式中內含 SQL 陳述式』
SQLJ 的靜態 SQL 陳述式位於 SQLJ 子句內。SQLJ 子句的開頭爲 `#sql`，結尾是分號 (;) 字元。

Java SQL 常式

iSeries 伺服器有能力從 SQL 陳述式及程式中存取 Java 程式。透過 Java 儲存程序及 Java 使用者定義的函數 (UDF) 即可完成此動作。iSeries 伺服器支援以 DB2 及 SQLJ 慣例來呼叫 Java 儲存程序及 Java UDF。Java 儲存程序與 Java UDF 二者，皆可使用 JAR 檔中儲存的 Java 類別。iSeries 伺服器可以使用 *SQLJ Part 1* 標準所定義的儲存程序，向資料庫登記 JAR 檔。

使用 Java SQL 常式

您可以從 SQL 陳述式及程式存取 Java 程式。透過 Java 儲存程序及 Java 使用者定義的函數 (UDF) 即可完成此動作。

若要使用 Java SQL 常式，請完成下列作業：

1. 啓用 SQLJ

   因爲任何 Java SQL 常式都可能使用 SQLJ，所以在執行 Java 2 Software Development Kit (J2SDK) 時，請隨時備妥 SQLJ 執行時間支援。若要在 J2SDK 中啓用 SQLJ 的執行時間支援，請新增一個從 SQLJ runtime.zip 檔案至延伸套件目錄的鏈結。如需相關資訊，請參閱第 175 頁的『設定伺服器來使用 SQLJ』。

2. 編寫供常式使用的 Java 方法

   Java SQL 常式可以透過 SQL 來處理 Java 方法。此方法必須採用 DB2 或 SQLJ 參數傳遞慣例來編寫。如需編寫 Java SQL 常式所使用之方法的相關資訊，請參閱 Java 儲存程序、Java 使用者定義的函數 及 Java 使用者定義的表格函數。

3. 編譯 Java 類別

   使用 Java 參數樣式來編寫的 Java SQL 常式，不需任何額外的設定，就可以直接編譯。然而，使用 DB2GENERAL 參數樣式的 Java SQL 常式，則必須延伸 com.ibm.db2.app.UDF 類別或 com.ibm.db2.app.StoredProc 類別。這些類別內含於 JAR 檔 /QIBM/ProdData/Java400/ext/db2routines_classes.jar 之中。 使用 javac 來編譯這些常式，此 JAR 檔必須存在於 CLASSPATH 中。例如，下列指令會編譯 Java 來源檔，其中包含的常式就是使用 DB2GENERAL 參數樣式：

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

4. 讓資料庫使用的 Java 虛擬機器 (JVM) 能夠存取編譯後的類別。

   資料庫 JVM 所用的使用者定義的類別，可以放在 /QIBM/UserData/OS400/SQLLib/Function 目錄中，或已向資料庫登記的 JAR 檔內。

   /QIBM/UserData/OS400/SQLLib/Function 相當於 iSeries 的 /sqllib/function，此目錄供 DB2 UDB 儲存其他平台的 Java 儲存程序及 Java UDF。若類別是 Java 套件的一部分，則必須放在適當的子目錄中。比方說，如果建立的 runit 類別屬於 foo.bar 套件的一部份，則檔案 runnit.class 應該位於整合檔案系統的 /QIBM/ProdData/OS400/SQLLib/Function/foo/bar 目錄中。

   類別檔案亦可放入已向資料庫登記的 JAR 檔內。可以使用 SQLJ.INSTALL_JAR 儲存程序來登記 JAR 檔。此儲存程序可用來指派 JAR ID 給 JAR 檔。此 JAR ID 可用來識別類別檔案內的 JAR 檔。如需 SQLJ.INSTALL_JAR 及其他操作 JAR 檔之儲存程序的相關資訊，請參閱操作 JAR 檔的 SQLJ 程序。

5. 向資料庫登記常式。

   利用 CREATE PROCEDURE 及 CREATE FUNCTION SQL 陳述式，就可以向資料庫登記 Java SQL 常式。這些陳述式包含下列元素：

CREATE 關鍵字

用於建立 Java SQL 常式的 SQL 陳述式，以 CREATE PROCEDURE 或 CREATE STATEMENT 開頭。

常式名稱

SQL 陳述式接著定義資料庫已知的常式名稱。這就是從 SQL 存取 Java 常式時所用的名稱。

參數及回覆值

然後 SQL 陳述式會識別 Java 常式的參數及回覆值 (如果有的話)。

LANGUAGE JAVA

SQL 陳述式使用關鍵字 LANGUAGE JAVA 來指出常式是採用 Java 編寫的。

PARAMETER STYLE 關鍵字

SQL 陳述式接著使用關鍵字 PARAMETER STYLE JAVA 或 PARAMETER STYLE DB2GENERAL 來定義參數樣式。

外部名稱

然後 SQL 陳述式會識別要當作 Java SQL 常式來處理的 Java 方法。外部名稱有兩種格式：

- 若方法所在的類別檔案位於 /QIBM/UserData/OS400/SQLLib/Function 目錄之下，則以 *classname.methodname* 格式來識別方法，其中 *classname* 是完整的類別名稱，*methodname* 是方法名稱。
- 若方法位於已向資料庫登記的 JAR 檔中，則以 *jarid*:*classname.methodname* 格式來識別方法，其中 *jarid* 是已登記的 JAR 檔的 JAR ID，*classname* 是類別名稱，而 *methodname* 是方法名稱。

「iSeries 領航員」可以建立採用 Java 參數樣式的儲存程序或使用者定義的函數。

6. 使用 Java 程序

Java 儲存程序透過 SQL CALL 陳述式來呼叫。Java UDF 是作爲另一個 SQL 陳述式一部分呼叫的函數。

設定伺服器來使用 SQLJ:

執行具有內含 SQLJ 陳述式的 Java 程式之前，請確實將伺服器設定爲支援 SQLJ。您必須修改伺服器的 CLASSPATH 環境變數，才能使用 SQLJ 支援。

如需處理 Java 類別路徑的相關資訊，請參閱下列頁面：

Java 類別路徑

使用 SQLJ 及 J2SDK

在伺服器上設定 SQLJ 時，若伺服器執行的是支援 SQLJ 的 J2SDK 版本，請完成下列步驟：

1. 在伺服器的 CLASSPATH 環境變數中，加入下列檔案：
   - /QIBM/ProdData/Os400/Java400/ext/sqlj_classes.jar
   - /QIBM/ProdData/Os400/Java400/ext/translator.zip

   附註：欲執行 SQLJ 轉換程式時 (sqlj 指令)，才需要加入 translator.zip。若只是要執行編譯過且使用 SQLJ 的 Java 程式，則不必加入 translator.zip。如需相關資訊，請參閱下列頁面：

   SQLJ 轉換程式 (sqlj)
2. 在 iSeries 指令提示上，使用下列指令來新增您延伸套件目錄中 runtime.zip 的鏈結。將下列指令鍵入同一行，然後按 **Enter** 鍵。

```
ADDLNK OBJ('/QIBM/ProdData/Os400/Java400/ext/runtime.zip')
     NEWLNK('/QIBM/UserData/Java400/ext/runtime.zip')
```

有關安裝延伸套件的資訊，請參閱下列網頁：

安裝 IBM Developer Kit for Java 的延伸套件

鏈結集合

Java 類別路徑
Java(TM) 虛擬機器在執行時間使用 Java 類別路徑來尋找類別。Java 指令及工具亦使用類別路徑來尋找類別。預設系統類別路徑、CLASSPATH 環境變數及類別路徑指令參數，皆會決定尋找特定類別時所搜尋的目錄。

SQLJ 轉換程式 (sqlj)
SQLJ 轉換程式 sqlj 可以產生序列化的設定檔，內含位於 SQLJ 程式中有關 SQL 作業的資訊。SQLJ 轉換程式會用到 /QIBM/ProdData/Java400/ext/translator.zip 檔案。

安裝 IBM Developer Kit for Java 的延伸套件
延伸套件是指可用來延伸核心平台功能的 Java 類別套件。延伸套件會封裝成一或多個 ZIP 檔或 JAR 檔，由延伸類別載入器載入 Java 虛擬機器內。

Java 儲存程序

使用 Java 來編寫儲存程序時，您可以採用兩種參數傳遞樣式。

建議的樣式爲 JAVA 參數樣式，它符合「SQLj：SQL 常式」標準中指定的參數樣式。第二種樣式 DB2GENERAL 則是 DB2 UDB 所定義的參數樣式。此參數樣式亦決定編寫 Java 儲存程序時必須遵守的慣例。

此外，您也應該注意 Java 儲存程序方面的一些限制。

JAVA 參數樣式: 在編寫使用 JAVA 參數樣式的 Java 儲存程序時，您必須遵守下列慣例：

- Java 方法必須是 public void static (非實例) 方法。
- Java 方法的參數必須是 SQL 相容類型。
- 若參數屬於可使用空值的類型 (例如 String)，則Java 方法可測試 SQL NULL 值。
- 使用單一元素的陣列來傳回輸出參數。
- Java 方法可使用 getConnection 方法來存取現行資料庫。

使用 JAVA 參數樣式的 Java 儲存程序爲 public static 方法。在類別內，會以方法名稱及簽章來識別儲存程序。當您呼叫儲存程序時，就會根據 CREATE PROCEDURE 陳述式所定義的變數類型，自動產生簽章。

若參數傳入允許空值的 Java 類型，Java 方法可以比較參數與空值，判斷輸入參數是否爲 SQL NULL。

下列 Java 類型不支援空值：

- short
- int
- long
- float
- double

若在不支援空值的 Java 類型中傳入空值，則會傳回 SQL 異常，錯誤碼爲 -20205。

輸出參數皆透過只有一個元素的陣列來傳遞。Java 儲存程序可以設定陣列的第一個元素來設定輸出參數。

內含應用程式環境定義的連線須透過下列 Java Database Connectivity (JDBC) 呼叫來存取：

```
connection=DriverManager.getConnection("jdbc:default:connection");
```

然後，此連線再透過 JDBC API 來執行 SQL 陳述式。

以下爲具有一個輸入和兩個輸出的小型儲存程序。它首先會執行給定的 SQL 查詢，然後傳回結果列數及 SQLSTATE。

範例：具有一個輸入和兩個輸出的儲存程序

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
  }
 }
}
```

根據 SQLj 標準的定義，若於採用 JAVA 參數樣式的常式中傳回結果集，必須明確地設定結果集。在建立會傳回結果集的程序時，應該在參數清單尾端加入其他的結果集參數。例如，下列陳述式

```
CREATE PROCEDURE RETURNTWO()
DYNAMIC RESULT SETS 2
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'javaClass!returnTwoResultSets'
```

會呼叫 `public static void returnTwoResultSets(ResultSet[] rs1, ResultSet[] rs2)` 簽章的 Java 方法。

結果集的輸出參數必須依下列範例所示明確地設定。如同使用 DB2GENERAL 樣式，最後也一樣不應該關閉結果集和相對應的陳述式。

範例：傳回兩個結果集的儲存程序

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;
public class javaClass {
  /**
        * Java stored procedure, with JAVA style parameters,
        * that processes two predefined sentences
        * and returns two result sets
    *
        * @param ResultSet[] rs1    first ResultSet
        * @param ResultSet[] rs2    second ResultSet
    */
public static void returnTwoResultSets (ResultSet[] rs1, ResultSet[] rs2) throws Exception
{
  //get caller's connection to the database; inherited from StoredProc
```

```
  Connection con = DriverManager.getConnection("jdbc:default:connection");

    //define and process the first select statement
    Statement stmt1 = con.createStatement();
    String sql1 = "select value from table01 where index=1";
    rs1[0] = stmt1.executeQuery(sql1);

    //define and process the second select statement
    Statement stmt2 = con.createStatement();
    Stringsql2 = "select value from table01 where index=2";
    rs2[0] = stmt2.executeQuery(sql2);
  }
}
```

在伺服器上，並不會檢查其他的結果集參數來決定結果集的順序。伺服器上的結果集會依照其開啓的次序傳回。爲了確保與 SQLj 標準的相容性，應該以開啓順序來指派結果，如先前所述。

DB2GENERAL 參數樣式:

使用 DB2GENERAL 參數樣式編寫 Java 儲存程序時，您必須遵守下列慣例。

- 負責定義 Java 儲存程序的類別，必須*延伸* Java com.ibm.db2.app.StoredProc 類別或爲其子類別。
- Java 方法必須是 public void 實例方法。
- Java 方法的參數必須是 SQL 相容類型。
- Java 方法最好使用 isNull 方法來測試 SQL NULL 值。
- Java 方法必須明確地使用 set 方法來設定傳回參數。
- Java 方法最好使用 getConnection 方法來存取現行資料庫。

含有 Java 儲存程序的類別，必須延伸 com.ibm.db2.app.StoredProc 類別。Java 儲存程序爲 public 實例方法。在類別內，會以方法名稱及簽章來識別儲存程序。當您呼叫儲存程序時，就會根據 CREATE PROCEDURE 陳述式所定義的變數類型，自動產生簽章。

com.ibm.db2.app.StoredProc 類別可提供 isNull 方法，讓 Java 方法判斷輸入參數是否爲 SQL NULL。com.ibm.db2.app.StoredProc 類別亦提供 set...( ) 方法來設定輸出參數。您必須使用這些方法來設定輸出參數。若您未設定輸出參數，則輸出參數會傳回 SQL NULL 值。

com.ibm.db2.app.StoredProc 類別提供下列常式，用來提取內含應用程式環境定義的 JDBC 連線。然後使用下列 JDBC 呼叫來存取內含應用程式環境定義的連線：

```
public Java.sql.Connection getConnection( )
```

然後，此連線再透過 JDBC API 來執行 SQL 陳述式。

以下爲具有一個輸入和兩個輸出的小型儲存程序。它首先會處理給定的 SQL 查詢，然後傳回結果列數及 SQLSTATE。

範例：具有一個輸入和兩個輸出的儲存程序

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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());
        }
      }
    }
```

若要在使用 DB2GENERAL 參數樣式的程序中傳回結果集，則於程序結束時，結果集和回應的陳述式必須維持開啓狀態。傳回的結果集必須由用戶端應用程式來關閉。若傳回多個結果集，則會依當初開啓的次序傳回。例如，下列儲存程序會傳回兩個結果集。

範例：傳回兩個結果集的儲存程序

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
public void returnTwoResultSets() throws Exception
{
    // get caller's connection to the database; inherited from 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);
}
```

Java 儲存程序的限制:

這些限制會套用至 Java 儲存程序。

- Java 儲存程序不應該建立額外的緒。唯有當工作具備多緒能力時，才能在工作中建立額外的緒。由於無法保證呼叫 SQL 儲存程序的工作一定具備多緒能力，因此 Java 儲存程序不應該建立額外的緒。
- 不可使用「採用權限」來存取 Java 類別檔案。
- Java 儲存程序一律使用系統上安裝的最新版 Java Development Kit。
- 因爲 java.sql 與 com.ibm.db2.app 兩組套件都有 Blob 及 Clob 類別，若程式設計師要在一個程式中同時用到這兩個類別，則必須指定它們的完整名稱。程式一定要使用 com.ibm.db2.app 的 Blob 及 Clob 類別，作爲傳給儲存程序的參數。
- 建立 Java 儲存程序時，系統會在檔案庫中產生程式。此程式即用來儲存程序定義。程式的名稱由系統產生。您可以針對建立儲存程序的工作，檢查其工作日誌來取得此名稱。若還原先前儲存的程式物件，則程序定義也會一併還原。若要將 Java 儲存程序移至另一個系統，您要負責移動含有程序定義的程式，以及含有 Java 類別的整合檔案系統檔。
- Java 儲存程序不可設定用來連接資料庫的 JDBC 連線內容 (如系統命名)。除非停用預先提取功能，否則將一律採用預設的 JDBC 連線內容。

Java 使用者定義的純量函數

Java 純量函數可以從 Java 程式傳回一個值給資料庫。舉例來說，您可以建立純量函數來傳回兩個數字的總和。

就像 Java 儲存程序一樣，Java 純量函數也採用兩種參數樣式：Java 及 DB2GENERAL。編寫 Java 使用者定義的函數 (UDF) 時，請務必顧及建立 Java 純量函數方面的限制。

參數樣式 Java

Java 參數樣式爲 *SQLJ Part 1: SQL Routines* 標準所指定的樣式。編寫 Java UDF 時，請使用下列慣例。

- Java 方法必須爲 public static 方法。
- Java 方法必須傳回 SQL 相容的類型。回覆值爲方法的結果。
- Java 方法的參數必須爲 SQL 相容的類型。
- 對於允許 NULL 值的 Java 類型，Java 方法最好測試 SQL NULL。

例如，假設有一個 UDF 稱爲 sample!test3，此 UDF 會傳回 INTEGER，且接受類型爲 CHAR(5)、BLOB(10K) 及 DATE 的引數，DB2 將預期此 UDF 的 Java 實作方式有下列簽章：

```
import com.ibm.db2.app.*;
public class sample {
      public static int test3(String arg1, Blob arg2, Date arg3) { ... }
}
```

Java 方法的參數必須爲 SQL 相容的類型。例如，如果 UDF 被宣告爲接受 SQL 類型 t1、t2 及 t3 的引數，且傳回類型爲 t4，則會採用預期的 Java 簽章，將它視爲 Java 方法來呼叫：

```
public static T4 name (T1 a, T2 b, T3 c)  { .....}
```

其中：

- *name* 爲方法名稱
- T1 至 T4 代表 Java 類型，對應於 SQL 類型 t1 至 t4。
- *a*、*b* 及 *c* 爲輸入引數的任意變數名稱。

關於 SQL 類型與 Java 類型的相關性，請參閱儲存程序及 UDF 的參數傳遞慣例。

SQL NULL 值由尚未起始設定的 Java 變數表示。這些變數若爲物件類型，則含有 Java NULL 值。若將 SQL NULL 傳給 Java 純量資料類型 (例如 int)，則會發生異常狀況。

採用 JAVA 參數樣式時，若要從 Java UDF 傳回結果，直接從方法傳回結果即可。

```
{ ....
    return value;
}
```

就像在 UDF 及儲存程序中使用的 C 模組一樣，Java UDF 中也無法使用 Java 標準 I/O 串流 (System.in、System.out 及 System.err)。

參數樣式 DB2GENERAL

參數樣式 DB2GENERAL 由 Java UDF 使用。在此參數樣式中，回覆值會當作函數的最後一個參數來傳遞，且必須使用 com.ibm.db2.app.UDF 類別的 *set* 方法來設定。

編寫 Java UDF 時，必須遵循下列慣例：

- 含 Java UDF 的類別必須*延伸* Java com.ibm.db2.app.UDF 類別或爲其子類別。
- 如採用 DB2GENERAL 參數樣式，Java 方法必須爲 public void 實例方法。
- Java 方法的參數必須爲 SQL 相容的類型。
- Java 方法最好使用 isNull 方法來測試 SQL NULL 值。
- 若採用 DB2GENERAL 參數樣式，Java 方法必須明確地使用 set() 方法來設定傳回參數。

含有 Java UDF 的類別，必須延伸 Java 類別 com.ibm.db2.app.UDF。採用 DB2GENERAL 參數樣式的 Java UDF，必須爲 Java 類別的 void 實例方法。例如，假設有一個 UDF 稱爲 sample!test3，此 UDF 會傳回 INTEGER，且接受類型爲 CHAR(5)、BLOB(10K) 及 DATE 的引數，DB2 將預期此 UDF 的 Java 實作方式有下列簽章：

```
import com.ibm.db2.app.*;
public class sample extends UDF {
      public void test3(String arg1, Blob arg2, String arg3, int result) { ... }
}
```

Java 方法的參數必須爲 SQL 類型。例如，如果 UDF 被宣告爲接受 SQL 類型 t1、t2 及 t3 的引數，且傳回類型爲 t4，則會採用預期的 Java 簽章，將它視爲 Java 方法來呼叫：

```
public void name (T1 a, T2 b, T3 c, T4 d)  { .....}
```

其中：

- *name* 爲方法名稱
- T1 至 T4 代表 Java 類型，對應於 SQL 類型 t1 至 t4。
- *a*、*b* 及 *c* 爲輸入引數的任意變數名稱。
- *d* 爲任意變數名稱，代表要計算的 UDF 結果。

關於 SQL 類型與 Java 類型的相關性，請參閱儲存程序及 UDF 的參數傳遞慣例一節。

SQL NULL 值由尚未起始設定的 Java 變數表示。根據 Java 規則，這些變數若爲初始類型，則值爲零，若爲物件類型，則爲 Java NULL。爲了判斷到底是 SQL NULL 或是一般的零，可以在任何輸入引數上呼叫 isNull 方法：

```
{ ....
  if (isNull(1)) { /* argument #1 was a SQL NULL */ }
  else           { /* not NULL */ }
}
```

上述範例中，引數從 1 開始編號。就像接下來的其他函數一樣，isNull() 函數繼承自 com.ibm.db2.app.UDF 類別。 採用 DB2GENERAL 參數樣式時，若要從 Java UDF 傳回結果，請在 UDF 內使用 set() 方法，如下所示：

```
{ ....
  set(2, value);
}
```

其中，2 代表輸出引數的索引，*value* 爲文字或相容類型的變數。引數編號就是選定的輸出在引數清單內的索引。在本節的第一個範例中，int 結果變數的索引爲 4。UDF 傳回之前未設定的輸出引數，將含有 NULL 值。

就像在 UDF 及儲存程序中使用的 C 模組一樣，Java UDF 中也無法使用 Java 標準 I/O 串流 (System.in、System.out 及 System.err)。

在一項查詢的輸入或結果集中，DB2 會針對每一列呼叫一次 UDF，因此 DB2 通常會呼叫 UDF 許多次。 若在 UDF 的 CREATE FUNCTION 陳述式上指定 SCRATCHPAD，DB2 會認爲連續呼叫 UDF 之間需要保持一定的「連貫性」，因此，就 DB2GENERAL 參數樣式的函數而言，並非每一次呼叫都從實作的 Java 類別產生實例，通常是每一個陳述式每一次參照到 UDF 時，才產生一次實例。然而，若 UDF 指定 NO SCRATCHPAD，則每一次呼叫 UDF 時，都會呼叫類別建構子來設定一個全新的實例。

高速暫存器有助於在呼叫 UDF 期間儲存資訊。Java UDF 可以使用實例變數或設定高速暫存器來達到呼叫之間的連貫性。Java UDF 可以透過 com.ibm.db2.app.UDF 的 getScratchPad 及 setScratchPad 方法來存取高速暫存器。若在 CREATE FUNCTION 陳述式上指定 FINAL CALL 選項，則查詢結束時，將會呼叫物件的 public void

close() 方法 (若採用 DB2GENERAL 參數樣式的函數)。若未定義此方法，則由 Stub 函數接手，並且忽略事件。com.ibm.db2.app.UDF 類別可提供實用的變數及方法， 您可以在 DB2GENERAL 參數樣式 UDF 內加以利用。下表說明這些變數及方法。

變數及方法	說明
• 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;	若爲純量 UDF，這些常數可以判斷這是第一個呼叫或正常呼叫。若爲表格 UDF，這些常數可以判斷這是第一個呼叫、開啓呼叫、擷取呼叫、關閉呼叫或最終呼叫。
public Connection getConnection();	此方法可以取得此儲存程序呼叫的 JDBC 連線控點，然後傳回代表呼叫端應用程式資料庫連線的 JDBC 物件。類似 C 儲存程序中空值 SQLConnect() 呼叫的結果。
public void close();	若使用 FINAL CALL 選項來建立 UDF，則 UDF 運算結束時，資料庫就會呼叫此方法。 類似 C UDF 的最終呼叫。若 Java UDF 類別未實作此方法，則會忽略此事件。
public boolean isNull(int i)	此方法可以測試給定索引上的輸入引數是否爲 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);	這些方法將一個輸出引數設定爲給定的值。但發生任何錯誤時，將丟出異常，例如： • 目前並未進行 UDF 呼叫 • 索引未參照有效的輸出引數 • 資料類型不符 • 資料長度不符 • 字碼頁轉換發生錯誤
public void setSQLstate(String string);	可以從 UDF 中呼叫此方法，藉以設定此呼叫要傳回的 SQLSTATE。若無法接受字串作爲 SQLSTATE，則丟出異常。使用者可以在外部程式中設定 SQLSTATE，從函數傳回錯誤或警告。 在此情況下，SQLSTATE 必須包含下列其中一個值： • '00000'，表示順利完成 • '01Hxx'，其中 xx 爲任何二位數或大寫字母，用來表示警告 • '38yxx'，其中 y 爲 'I' 與 'Z' 之間的大寫字母，xx 爲任何二位數或大寫字母，用來表示錯誤
public void setSQLmessage(String string);	此方法類似 setSQLstate 方法。用途爲設定 SQL 訊息結果。若無法接受字串 (例如，超過 70 個字元)，則會丟出異常。
public String getFunctionName();	此方法傳回處理中 UDF 的名稱。
public String getSpecificName();	此方法傳回處理中 UDF 的特定名稱。

變數及方法	說明
public byte[] getDBinfo();	針對處理中的 UDF，此方法會以位元組陣列傳回原始未經處理的 DBINFO 結構。 UDF 必須已搭配 DBINFO 選項完成登記 (使用 CREATE FUNCTION)。
• 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();	這些方法會從處理中 UDF 的 DBINFO 結構，傳回適當欄位的值。 UDF 必須已搭配 DBINFO 選項完成登記 (使用 CREATE FUNCTION)。唯有在 UPDATE 陳述式的 SET 子句右邊指定使用者定義的函數，getDBtbschema()、getDBtbname() 及 getDBcolname() 方法才能傳回有意義的資訊。
public int getCCSID();	此方法傳回工作的 CCSID。
public byte[] getScratchpad();	此方法傳回目前處理中 UDF 的高速暫存器複本。您必須使用 SCRATCHPAD 選項來宣告 UDF。
public void setScratchpad(byte ab[]);	此方法利用給定的位元組陣列的內容，改寫目前處理中 UDF 的高速暫存器。 您必須使用 SCRATCHPAD 選項來宣告 UDF。位元組陣列與 getScratchpad() 傳回的大小必須相等。
public int getCallType();	此方法傳回目前正在執行的呼叫類型。這些值對應於 sqludf.h 中定義的 C 值。可能的回覆值包括： • SQLUDF_FIRST_CALL • SQLUDF_NORMAL_CALL • SQLUDF_TF_FIRST • SQLUDF_TF_OPEN • SQLUDF_TF_FETCH • SQLUDF_TF_CLOSE • SQLUDF_TF_FINAL

Java 使用者定義函數的限制:

這些限制會套用至 Java 使用者定義的函數 (UDF)。

- Java UDF 不應該建立額外的緒。唯有當工作具備多緒能力時，才能在工作中建立額外的緒。由於無法保證呼叫 SQL 儲存程序的工作一定具備多緒能力，因此 Java 儲存程序不應該建立其他的緒。
- 在資料庫中定義的 Java 儲存程序，其完整名稱最長限制爲 279 個字元。這項限制是爲了遷就 EXTERNAL_NAME 直欄，這種直欄的最大寬度爲 279 個字元。
- 採用權限不可用來存取 Java 類別檔案。
- Java UDF 一律使用系統上安裝的最新版 JDK。
- 因爲 java.sql 與 com.ibm.db2.app 兩組套件都有 Blob 及 Clob 類別，若程式設計師要在一個程式中同時用到這兩個類別，則必須指定它們的完整名稱。程式一定要使用 com.ibm.db2.app 的 Blob 及 Clob 類別，作爲傳給儲存程序的參數。

- 就像來源函數一樣，建立 Java UDF 時，也會利用檔案庫裡的服務程式來儲存函數定義。此服務程式的名稱由系統產生，可以根據建立此函數的工作，在相關的工作日誌中找到。若儲存這個物件，然後又在另一個系統上還原，則函數定義也會一併還原。若要將 Java UDF 移至另一個系統，您要負責移動含有函數定義的服務程式，以及含有 Java 類別的整合檔案系統檔。
- Java UDF 不可設定用來連接資料庫的 JDBC 連線內容 (如系統命名)。除非停用預先提取功能，否則將一律採用預設的 JDBC 連線內容。

Java 使用者定義的表格函數:

DB2 具備從函數傳回表格的能力。若要以表格形式來呈現資料庫以外的資訊給資料庫時，就需要這種能力。 例如，對於用來執行 Java 儲存程序及 Java UDF (表格與純量函數) 的 Java 虛擬機器 (JVM)，可以建立表格來呈現其中設定的內容。

SQLJ Part 1: SQL Routines 標準不支援表格函數。因此，只能以參數樣式 DB2GENERAL 來使用表格函數。

表格函數有 5 種不同的呼叫類型。下表將解釋這些呼叫。其中假設 create function SQL 陳述式上已指定高速暫存器。

掃描時間點	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
表格函數的第一個 OPEN 之前	沒有呼叫	呼叫類別建構子 (表示新建高速暫存器)。以 FIRST 呼叫來呼叫 UDF 方法。
表格函數的每一個 OPEN	呼叫類別建構子 (表示新建高速暫存器)。以 OPEN 呼叫來呼叫 UDF 方法。	以 OPEN 呼叫來呼叫 UDF 方法。
針對每一列新的表格函數資料的每一個 FETCH。	以 FETCH 呼叫來呼叫 UDF 方法。	以 FETCH 呼叫來呼叫 UDF 方法。
表格函數的每一個 CLOSE	以 CLOSE 呼叫來呼叫 UDF 方法。若 close() 方法存在，則一併呼叫。	以 CLOSE 呼叫來呼叫 UDF 方法。
表格函數的最後一個 CLOSE 之後。	沒有呼叫	以 FINAL 呼叫來呼叫 UDF 方法。若 close() 方法存在，則一併呼叫。

範例：Java 表格函數

對於用來執行 Java 使用者定義表格函數的 JVM，下列範例說明 Java 表格函數如何判斷其中設定的內容。

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
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);
    }
  }
}
```

當表格函數完成編譯，且其類別檔案已複製到 /QIBM/UserData/OS400/SQLLib/Function 之後， 可使用下列 SQL 陳述式，向資料庫登記此函數。

```
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
```

登記此函數之後，即可應用於 SQL 陳述式中。例如，下列 SELECT 陳述式會傳回此表格函數所產生的表格。

```
SELECT * FROM TABLE(PROPERTIES())
```

操作 JAR 檔的 SQLJ 程序

Java 儲存程序及 Java UDF 兩者，皆可使用 Java JAR 檔中儲存的 Java 類別。

若要使用 JAR 檔，*jar-id* 必須連結 JAR 檔。系統在 SQLJ 綱目中提供了儲存程序，可用來操作 *jar-ids* 及 JAR 檔案。 這些程序可以安裝、置換及移除 JAR 檔。也可使用及更新 JAR 檔相關的 SQL 型錄。

SQLJ.INSTALL_JAR:

SQLJ.INSTALL_JAR 儲存程序可以將 JAR 檔安裝至資料庫系統。 此 JAR 檔可用於後續的 CREATE FUNCTION 及 CREATE PROCEDURE 陳述式中。

授權

CALL 陳述式的授權 ID 所擁有的專用權，至少必須包含 SYSJAROBJECTS 及 SYSJARCONTENTS 型錄表格的下列其中一項：

- 下列系統權限：
  - 表格的 INSERT 及 SELECT 專用權
  - 檔案庫 QSYS2 的系統權限 *EXECUTE
- 管理權限

CALL 陳述式的授權 ID 所擁有的專用權，亦必須具備下列權限：

- 對於 *jar-url* 參數中指定要安裝的 JAR 檔，具有讀取 (*R) 存取權限。

- 對於 JAR 檔的安裝目錄，具有寫入、執行及讀取 (*RWX) 存取權限。 此目錄爲 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*，其中 *schema* 代表 *jar-id* 的綱目。

採用權限不適用於這些權限。

SQL 語法

```
>>-CALL--SQLJ.INSTALL_JAR-- (--'jar-url'--,--'jar-id'--,--deploy--)-->

>-------------------------------------------------------------><
```

說明

jar-url 含要安裝或置換的 JAR 檔的 URL。'file:' 是唯一支援的 URL 語法。

jar-id 資料庫中要與 *jar-url* 指定的檔案連結起來的 JAR ID。*jar-id* 採用 SQL 命名， JAR 檔會安裝在隱含或明確限定元所指定的綱目或檔案庫中。

deploy
用來說明部署描述子檔案的 install_action 的值。 若此整數不是零值，則應該在 install_jar 程序結束時才執行部署描述子檔案的 install_actions。 DB2 UDB for iSeries 現行版本僅支援零值。

使用注意事項

安裝 JAR 檔時，DB2 UDB for iSeries 會在 SYSJAROBJECTS 系統型錄內登記 JAR 檔。同時也會擷取 JAR 檔內的 Java(TM) 類別檔案名稱，將每一個類別登記在 SYSJARCONTENTS 系統型錄內。 DB2 UDB for iSeries 會將 JAR 檔複製到 /QIBM/UserData/OS400/SQLLib/Function 目錄的 jar/schema 子目錄中。DB2 UDB for iSeries 也會以 *jar-id* 子句中給定的名稱來命名新的 JAR 檔案複本。不應變更 DB2 UDB for iSeries 已安裝到 /QIBM/UserData/OS400/SQLLib/Function/jar 之子目錄的 JAR 檔。應該改用 CALL SQLJ.REMOVE_JAR 及 CALL SQLJ.REPLACE_JAR SQL 指令來移除或置換已安裝的 JAR 檔。

範例

下列指令從 SQL 互動式階段作業中發出。

```
CALL SQLJ.INSTALL_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar', 0)
```

file:/home/db2inst/classes/ 目錄中的 Proc.jar 檔，將以 myproc_jar 這個名稱安裝至 DB2 UDB for iSeries。後續用到 Procedure.jar 檔的 SQL 指令，就會以 myproc_jar 這個名稱來參照檔案。

SQLJ.REMOVE_JAR:

SQLJ.REMOVE_JAR 儲存程序可以從資料庫系統中移除 JAR 檔。

授權

CALL 陳述式的授權 ID 所擁有的專用權，至少必須包含 SYSJARCONTENTS 及 SYSJAROBJECTS 型錄表格的下列其中一項：

- 下列系統權限：
  - 表格的 SELECT 及 DELETE 專用權
  - 檔案庫 QSYS2 的系統權限 *EXECUTE
- 管理權限

CALL 陳述式的授權 ID 所擁有的專用權，亦必須具備下列權限。

- 要移除的 JAR 檔的 *OBJMGT 權限。JAR 檔命名爲 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile。

採用權限不適用於此權限。

語法

```
>>-CALL--SQLJ.REMOVE_JAR--(--'jar-id'--,--undeploy--)----------><
```

說明

jar-id 要從資料庫中移除的 JAR 檔的 JAR ID。

undeploy
用來說明部署描述子檔案的 remove_action 的值。 若此整數不是零值，則應該在 install_jar 程序結束時才執行部署描述子檔案的 remove_actions。 DB2 UDB for iSeries 現行版本僅支援零值。

範例

下列是從 SQL 互動式階段作業中發出的指令：

```
CALL SQLJ.REMOVE_JAR('myProc_jar', 0)
```

JAR 檔 myProc_jar 已從資料庫中移除。

SQLJ.REPLACE_JAR:

SQLJ.REPLACE_JAR 儲存程序可以置換資料庫系統中的 JAR 檔。

授權

CALL 陳述式的授權 ID 所擁有的專用權，至少必須包含 SYSJAROBJECTS 及 SYSJARCONTENTS 型錄表格的下列其中一項：

- 下列系統權限：
  - 表格的 SELECT、INSERT 及 DELETE 專用權
  - 檔案庫 QSYS2 的系統權限 *EXECUTE
- 管理權限

CALL 陳述式的授權 ID 所擁有的專用權，亦必須具備下列權限：

- 對於 *jar-url* 參數中指定要安裝的 JAR 檔，具有讀取 (*R) 存取權限。
- 要移除的 JAR 檔的 *OBJMGT 權限。JAR 檔命名爲 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile。

採用權限不適用於這些權限。

語法

```
>>-CALL--SQLJ.REPLACE_JAR--(--'jar-url'--,--'jar-id'--)--------><
```

說明

jar-url 含要置換的 JAR 檔的 URL。'file:' 是唯一支援的 URL 語法。

jar-id 資料庫中要與 *jar-url* 指定的檔案連結起來的 JAR ID。*jar-id* 採用 SQL 命名， JAR 檔會安裝在隱含或明確限定元所指定的綱目或檔案庫中。

使用注意事項

SQLJ.REPLACE_JAR 儲存程序可以置換先前以 SQLJ.INSTALL_JAR 安裝在資料庫中的 JAR 檔。

範例

下列是從 SQL 互動式階段作業中發出的指令：

```
CALL SQLJ.REPLACE_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar')
```

jar-id myproc_jar 所參照的現行 JAR 檔，將置換成 file:/home/db2inst/classes/ 目錄中的 Proc.jar 檔。

SQLJ.UPDATEJARINFO:

SQLJ.UPDATEJARINFO 可以更新 SYSJARCONTENTS 型錄表格的 CLASS_SOURCE 直欄。 此程序不屬於 SQLJ 標準，但爲 DB2 UDB for iSeries 儲存程序建置器所採用。

授權

CALL 陳述式的授權 ID 所擁有的專用權，至少必須包含 SYSJARCONTENT 型錄表格的下列其中一項：

- 下列系統權限：
  - 表格的 SELECT 及 UPDATEINSERT 專用權
  - 檔案庫 QSYS2 的系統權限 *EXECUTE
- 管理權限

執行 CALL 陳述式的使用者，亦必須具備下列權限：

- 對於 *jar-url* 參數中指定的 JAR 檔，具有讀取 (*R) 存取權限。對於要安裝的 JAR 檔，具有讀取 (*R) 存取權限。
- 對於 JAR 檔的安裝目錄，具有寫入、執行及讀取 (*RWX) 存取權限。 此目錄爲 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*，其中 *schema* 代表 *jar-id* 的綱目。

採用權限不適用於這些權限。

語法

```
>>-CALL--SQLJ.UPDATEJARINFO--(--'jar-id'--,--'class-id'--,--'jar-url'--)-->

>-------------------------------------------------------------><
```

說明

jar-id 資料庫中要更新的 JAR ID。

class-id
要更新的類別的套件完整類別名稱。

jar-url 用來更新 JAR 檔的類別檔案所在的 URL。'file:' 是唯一支援的 URL 語法。

範例

下列是從 SQL 互動式階段作業中發出的指令：

```
CALL SQLJ.UPDATEJARINFO('myproc_jar', 'mypackage.myclass',
                        'file:/home/user/mypackage/myclass.class')
```

jar-id myproc_jar 相關的 JAR 檔，會以新版的 mypackage.myclass 類別進行更新。所以可從檔案 /home/user/mypackage/myclass.class 取得新版的類別。

SQLJ.RECOVERJAR:

SQLJ.RECOVERJAR 程序會取出 SYSJAROBJECTS 型錄中儲存的 JAR 檔， 再將它復置成爲 /QIBM/UserData/OS400/SQLLib/Function/jar/*jarschema*/*jar_id*.jar 檔案。

授權

CALL 陳述式的授權 ID 所擁有的專用權，至少必須包含 SYSJAROBJECTS 型錄表格的下列其中一項：

- 下列系統權限：
  - 表格的 SELECT 及 UPDATEINSERT 專用權
  - 檔案庫 QSYS2 的系統權限 *EXECUTE
- 管理權限

執行 CALL 陳述式的使用者，亦必須具備下列權限：

- 對於 JAR 檔的安裝目錄，具有寫入、執行及讀取 (*RWX) 存取權限。 此目錄爲 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*，其中 *schema* 代表 *jar-id* 的綱目。
- 要移除的 JAR 檔的 *OBJMGT 權限。JAR 檔命名爲 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile。

語法

```
>>-CALL--SQLJ.RECOVERJAR--(--'jar-id'--)----------------------><
```

說明

jar-id 資料庫中要回復的 JAR ID。

範例

下列是從 SQL 互動式階段作業中發出的指令：

```
CALL SQLJ.UPDATEJARINFO('myproc_jar')
```

myproc_jar 相關的 JAR 檔，將以 SYSJARCONTENT 表格的內容進行更新。 檔案會複製爲 /QIBM/UserData/OS400/SQLLib/Function/jar/jar_schema myproc_jar.jar。

SQLJ.REFRESH_CLASSES:

SQLJ.REFRESH_CLASSES 儲存程序會在現行資料庫連線上，重新載入 Java 儲存程序或 Java UDF 所用的使用者定義類別。必須由現有的資料庫連線來呼叫此儲存程序，才能取得對 SQLJ.REPLACE_JAR 儲存程序的呼叫所進行的變更。

授權

無

語法

```
>>-CALL--SQLJ.REFRESH_CLASSES-- ()-->
>-------------------------------------------------------------><
```

範例

呼叫 Java 儲存程序 MYPROCEDURE，此儲存程序使用透過 MYJAR jarid 所登記之 JAR 檔中的類別：

```
CALL MYPROCEDURE()
```

使用下列呼叫來置換 JAR 檔：

```
CALL SQLJ.REPLACE_JAR('MYJAR', '/tmp/newjarfile.jar')
```

爲了讓後續 MYPROCEDURE 儲存程序的呼叫能夠使用更新的 JAR 檔，必須呼叫 SQLJ.REFRESH_CLASSES：

```
CALL SQLJ.REFRESH_CLASSES()
```

再呼叫一次儲存程序。這一次呼叫程序時，就會使用更新的類別檔案。

```
CALL MYPROCEDURE()
```

Java 儲存程序及 UDF 的參數傳遞慣例

下表列出 Java 儲存程序與 UDF 如何表示 SQL 資料類型。

SQL 資料類型	Java 參數樣式 JAVA	Java 參數樣式 DB2GENERAL
SMALLINT	short	short
INTEGER	int	int
BIGINT	long	long
DECIMAL(p,s)	BigDecimal	BigDecimal
NUMERIC(p,s)	BigDecimal	BigDecimal
REAL 或 FLOAT(p)	float	float
DOUBLE PRECISION、FLOAT 或 FLOAT(p)	double	double
CHARACTER(n)	String	String
CHARACTER(n) FOR BIT DATA	byte[]	com.ibm.db2.app.Blob
VARCHAR(n)	String	String
VARCHAR(n) FOR BIT DATA	byte[]	com.ibm.db2.app.Blob
GRAPHIC(n)	String	String
VARGRAPHIC(n)	String	String
DATE	Date	String
TIME	Time	String
TIMESTAMP	Timestamp	String
Indicator Variable	-	-
CLOB	-	com.ibm.db2.app.Clob
BLOB	-	com.ibm.db2.app.Blob
DBCLOB	-	com.ibm.db2.app.Clob
DataLink	-	-

Java 與其他程式設計語言

在 Java 中，有許多方法可以呼叫非 Java 語言所編寫的程式碼。

Java 原生介面

呼叫另一種語言所編寫的程式碼，方法之一就是將選定的 Java 方法當作「原生方法」來實作。原生方法指另一種語言所編寫的程序，可提供 Java 方法的實際實作方式。原生方法可以透過「Java 原生介面 (JNI)」來存取 Java

虛擬機器。這些原生方法皆於 Java 緒 (核心緒) 之下執行，所以一定符合安全緒標準。只要能夠在相同程序內的多個緒之中同時啓動一個函數，此函數即符合安全緒標準。此外，一個函數所呼叫的所有函數，也必須全部符合安全緒標準，此函數才堪稱符合安全緒標準。

原生方法可以存取 Java 未直接支援的系統功能，或連結現有的使用者程式碼，具有「橋接器」的功能。請謹慎使用原生方法，因爲被呼叫的程式碼有可能不符合安全緒標準。如需 JNI 及 ILE 原生方法的有關資訊，請參閱使用 Java 原生介面的原生方法。

Java 呼叫 API

Java 呼叫 API 亦屬於「Java 原生介面 (JNI)」規格，能夠讓非 Java 的應用程式使用 Java 虛擬機器。亦容許使用 Java 程式碼來延伸應用程式的功能。

i5/OS PASE 原生方法

iSeries Java 虛擬機器 (JVM) 目前支援使用在 i5/OS PASE 環境中執行的原生方法。i5/OS Java 的 PASE 原生方法可讓您輕易地將在 AIX® 中執行的 Java 應用程式移轉至 iSeries 伺服器。您可以將類別檔案及 AIX 原生方法程式庫複製到 iSeries 的整合檔案系統，然後透過控制語言 (CL)、Qshell 或 i5/OS PASE 終端機階段作業指令提示加以執行。

兆空間原生方法

iSeries Java 虛擬機器 (JVM) 目前支援使用兆空間儲存模型原生方法。兆空間儲存模型爲 ILE 程式提供了一個大型處理、本端位址的環境。兆空間可讓您將原生方法程式碼從其他作業系統移轉至 i5/OS，而原始程式碼只需少許更改，甚至完全不用變動。

java.lang.Runtime.exec()

您可以在 Java 程式內使用 java.lang.Runtime.exec() 來呼叫程式或指令。exec() 方法會啓動另一個處理程序，其中可執行任何 iSeries 程式或指令。在此模型中，可以使用子項程序的標準輸入、標準輸出及標準錯誤，來處理作業之間的通訊。

處理作業之間的通信

其中一種方法，是使用 Socket 來處理母項程序與子項程序之間的跨處理通訊。

您也可以使用串流檔來處理程式之間的通訊。或者，請參閱處理作業之間的通訊範例，其中概略提到與另一個程序中執行的程式進行通訊時，您有哪些可用的選項。

若要從其他語言中呼叫 Java，請參閱範例：從 C 呼叫 Java 或範例：從 PRG 呼叫 Java，以取得相關資訊。

您也可以使用 IBM Toolbox for Java 來呼叫 iSeries 伺服器上現有的程式及指令。通常會使用資料佇列及 iSeries 訊息來進行 IBM Toolbox for Java 的跨處理通訊。

註: 若使用 Runtime.exec()、IBM Toolbox for Java 或 JNI，Java 程式的可攜性可能因此而打折扣。在「純正的」Java 環境中，應該避免使用這些方法。

針對原生方法使用 Java 原生介面

只有在 Pure Java 無法滿足您的程式設計需求時，才應使用原生方法。

最好在下列情況，才使用原生方法：

- 存取使用 Pure Java 無法取得的系統功能。

- 實作非常重視效能的方法，此時利用原有的實作方式最佳。
- 連結到讓 Java 呼叫其他 API 的現有應用程式設計介面 (API)。

下列指示適用於在 C 語言中使用「Java 原生介面 (JNI)」。如需在 RPG 語言中使用 JNI 的相關資訊，請參閱下列文件：

WebSphere Development Studio: ILE RPG Programmer's Guide, SC09-2507 的第 11 章。

若要針對原生方法使用「Java 原生介面 (JNI)」，請執行下列步驟：

1. 採用標準的 Java 語言語法，指定哪些方法是原生方法以設計類別。
2. 依據包含原生方法實作方式的服務程式 (*SRVPGM)，決定檔案庫或程式名稱。在類別的靜態起始設定表示式中編寫 System.loadLibrary() 方法呼叫時，指定服務程式的名稱。
3. 使用 `javac` 工具將 Java 原始檔編譯成類別檔案。
4. 使用 `javah` 工具來建立標頭檔 (.h)。此標頭檔包含確切的原型，可用於建立原生方法實作方式。 -d 選項指定應在哪個目錄中建立標頭檔。
5. 使用「從串流檔複製 (CPYFRMSTMF)」指令，從整合檔案系統中將標頭檔複製到來源檔的成員內。您必須將標頭檔複製到檔案成員內，供 C 編譯器使用。使用「建立連結 ILE C/400® 程式 (CRTCMOD)」指令的新串流檔支援，讓 C 原始檔及 C 標頭檔保留在整合檔案系統內。如需 CRTCMOD 指令及串流檔用法的相關資訊，請參閱 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712。
6. 編寫原生方法程式碼。如需原生方法所用語言及函數的詳細資料，請參閱 Java 原生方法及緒注意事項。
   a. 併入先前步驟中建立的標頭檔。
   b. 正確地比對標頭檔中的原型。
   c. 若要將字串傳遞至 Java 虛擬機器，請將字串轉換成「美國國家標準交換碼 (ASCII)」。如需相關資訊，請參閱 Java 字元編碼。
7. 若原生方法必須與 Java 虛擬機器互動，請使用 JNI 提供的函數。
8. 使用 CRTCMOD 指令，將 C 原始程式碼編譯成模組 (*MODULE) 物件。
9. 使用「建立服務程式 (CRTSRVPGM)」指令，將一或多個模組物件連結成一個服務程式 (*SRVPGM)。此服務程式的名稱，必須符合您在 System.load() 或 System.loadLibrary() 函數呼叫的 Java 程式碼中提供的名稱。
10. 若在 Java 程式碼中使用 System.loadLibrary() 呼叫，請針對您執行的 J2SDK，從以下挑選適當的作業來執行：
    - 在 LIBPATH 環境變數中包含您需要的檔案庫清單。您可以在 QShell 及 iSeries 指令行中變更 LIBPATH 環境變數。
      – 從 Qshell 指令提示中鍵入：

        ```
        export LIBPATH=/QSYS.LIB/MYLIB.LIBjava -Djava.version=1.5 myclass
        ```

      – 或從指令行鍵入：

        ```
        ADDENVVAR LIBPATH '/QSYS.LIB/MYLIB.LIB'JAVA PROP((java.version 1.5)) myclass
        ```

    - 或者，在 **java.library.path** 內容中提供清單。您可以在 QShell 或 iSeries 指令行中變更 java.library.path 內容。
      – 從 Qshell 指令提示中輸入：

        ```
        java -Djava.library.path=/QSYS.LIB/MYLIB.LIB -Djava.version=1.5 myclass
        ```

      – 或從 iSeries 指令行鍵入：

```
JAVA PROP((java.library.path '/QSYS.LIB/MYLIB.LIB') (java.version '1.5')) myclass
```

其中 */QSYS.LIB/MYLIB.LIB* 是您要以 System.loadLibrary() 呼叫來載入的檔案庫，*myclass* 是 Java 應用程式的名稱。

11. System.load(String path) 的路徑語法可以是：
    - /qsys.lib/sysNMsp.srvpgm (表示 *SRVPGM QSYS/SYSNMSP)
    - /qsys.lib/mylib.lib/myNMsp.srvpgm (表示 *SRVPGM MYLIB/MYNMSP)
    - 符號鏈結，例如 /home/mydir/myNMsp.srvpgm，它會鏈結至 /qsys.lib/mylib.lib/myNMsp.srvpgm

    註: 這相當於使用 System.loadLibrary(″myNMsp″) 方法。

    註: 路徑名稱通常是引號括住的字串文字。例如，您可以使用下列程式碼：

    ```
    System.load("/qsys.lib/mylib.lib/myNMsp.srvpgm")
    ```

12. System.loadLibrary(String libname) 的 libname 參數，通常是引號括住的字串文字，用來識別原生方法檔案庫。系統會利用現行檔案庫清單及 LIBPATH 與 PASE_LIBPATH 環境變數，搜尋符合檔案庫名稱的服務程式或 i5/OS PASE 可執行檔。例如，`loadLibrary("myNMsp")` 就會搜尋名爲 MYNMSP 的 *SRVPGM，或名爲 libmyNMsp.a 或 libmyMNsp.so 的 i5/OS PASE 可執行檔。

如需 JNI 的完整說明，請參閱 Sun Microsystems 公司的「Java 原生介面」及 The Source for Java Technology java.sun.com。

如需如何針對原生方法使用 JNI 的範例，請參閱範例：針對原生方法使用 Java 原生介面。

Java 呼叫 API

「呼叫 API」屬於「Java 原生介面 (JNI)」的一部分，可讓非 Java 程式碼建立 Java 虛擬機器，並且載入及使用 Java 類別。此功能可讓多緒的程式在多緒中使用執行於單一 Java 虛擬機器上的 Java 類別。

IBM Developer Kit for Java 支援下列幾類呼叫程式的「Java 呼叫 API」：

- 針對 `STGMDL(*SNGLVL)` 及 `DTAMDL(*P128)` 所建立的 ILE 程式或服務程式
- 針對 `STGMDL(*TERASPACE)` 及 `DTAMDL(*LLP64)` 所建立的 ILE 程式或服務程式
- 針對 32 位元或 64 位元 AIX 所建立的 i5/OS PASE 可執行檔

應用程式可以控制 Java 虛擬機器。應用程式可以建立 Java 虛擬機器、呼叫 Java 方法 (類似應用程式呼叫子常式的方式)，以及摧毀 Java 虛擬機器。Java 虛擬機器建立後，除非應用程式明確地予以摧毀，否則在程序內隨時都可執行。Java 虛擬機器在摧毀過程中會執行清除作業，例如執行終止程式、結束 Java 虛擬機器緒及釋放 Java 虛擬機器資源。

Java 虛擬機器備妥可供執行後，以 C 及 RPG 等 ILE 語言所編寫的應用程式，即可呼叫 Java 虛擬機器來執行任何功能。亦可從 Java 虛擬機器返回 C 應用程式、重新呼叫 Java 虛擬機器，依此類推。一旦建立 Java 虛擬機器之後，就不必再重建，隨時可呼叫 Java 虛擬機器來執行少量或大量 Java 程式碼。

使用「呼叫 API」來執行 Java 程式時，必須利用環境變數 QIBM_USE_DESCRIPTOR_STDIO 來控制 STDOUT 及 STDERR 的目的地。若此環境變數設爲 Y 或 I (例如，QIBM_USE_DESCRIPTOR_STDIO=Y)，Java 虛擬機器會使用檔案描述子來設定 STDIN (fd 0)、STDOUT (fd 1) 及 STDERR (fd 2)。在此情況下，程式必須開啓這些檔案，作爲此工作的前三個檔案或管道，進而將這些檔案描述子設爲有效的值。工作中開啓的第一個檔案具有 fd 0，第二個具有 fd 1，第三個具有 fd 2。至於以 Spawn API 所起始的工作，則可利用檔案描述子映射來預先指派這些描述子 (請參閱 Spawn API 方面的文件)。若未設定環境變數 QIBM_USE_DESCRIPTOR_STDIO 或設爲其他值，則 STDIN、STDOUT 或 STDERR 不使用檔案描述子。STDOUT 及 STDERR 會改爲遞送至現有工作所擁有的排存檔，而使用 STDIN 則會導致 IO 異常。

如需「呼叫 API」的使用範例，請參閱範例：Java 呼叫 API。如需 IBM Developer Kit for Java 支援的「呼叫 API」函數的詳細資料，請參閱呼叫 API 函數。

呼叫 API 函數:

IBM Developer Kit for Java 支援下列「呼叫 API」函數。

註: 使用此 API 之前，請確定您所在的工作具備多緒能力。如需多緒型工作的詳細資訊，請參閱多緒的應用程式。

- **JNI_GetCreatedJavaVMs**

  傳回所有已建立之 Java 虛擬機器的相關資訊。雖然依據設計此 API 可傳回多部 Java 虛擬機器 (JVM) 的資訊，但一個程序只能存在一個 JVM。因此，此 API 最多只會傳回一個 JVM。

  簽章：

  ```
  jint JNI_GetCreatedJavaVMs(JavaVM **vmBuf,
                             jsize bufLen,
                             jsize *nVMs);
  ```

  vmBuf 是一個輸出區，其大小由代表指標數量的 bufLen 決定。每一個 Java 虛擬機器在 java.h 中都會定義相關的 JavaVM 結構。除非 vmBuf 爲 0，否則對於每一個建立的 Java 虛擬機器，此 API 都會在 vmBuf 中儲存相關的 JavaVM 結構指標。JavaVM 結構指標會依照相對應的 Java 虛擬機器的建立順序來儲存。nVMs 會傳回目前已建立的虛擬機器數量。iSeries 伺服器支援建立多部 Java 虛擬機器，所以預期此值應該大於 1。這項資訊再加上 vmBuf 的大小，即可決定每一個建立的 Java 虛擬機器是否傳回 JavaVM 結構的指標。

- **JNI_CreateJavaVM**

  可讓您建立 Java 虛擬機器，隨後在應用程式中使用。

  簽章：

  ```
  jint JNI_CreateJavaVM(JavaVM **p_vm,
                                          void **p_env,
                                                void *vm_args);
  ```

  p_vm 是新建立的 Java 虛擬機器的 JavaVM 指標位址。有一些「JNI 呼叫 API」會利用 p_vm 來識別 Java 虛擬機器。p_env 是新建立的 Java 虛擬機器的 JNI 環境指標位址。它指向啓動這些函數的 JNI 函數表。vm_args 是含有 Java 虛擬機器起始設定參數的結構。

  若啓動「執行 Java (RUNJVA)」指令或 JAVA 指令，但指定的內容有一個同等的指令參數，則會優先採用指令參數。內容將被忽略。例如，在此指令中，就會忽略 os400.optimization 參數：

  ```
  JAVA CLASS(Hello) PROP((os400.optimization 0))
  ```

  如需 JNI_CreateJavaVM API 支援的 0S/400 專屬內容的清單，請參閱 Java 系統內容。

  註: iSeries 伺服器上的 Java，支援在單一工作或程序內，只能建立一個 Java 虛擬機器 (JVM)。如需相關資訊，請參閱多重 Java 虛擬機器的支援。

- **DestroyJavaVM**

  摧毀 Java 虛擬機器。

  簽章：

  ```
  jint DestroyJavaVM(JavaVM *vm)
  ```

vm 是建立 Java 虛擬機器時所傳回的 JavaVM 指標。

- **AttachCurrentThread**

  將緒附加至 Java 虛擬機器，使它能夠運用 Java 虛擬機器的服務。

  簽章：

  ```
  jint AttachCurrentThread(JavaVM *vm,
                                     void **p_env,
                                     void *thr_args);
  ```

  JavaVM 指標 vm 可識別要附加緒的 Java 虛擬機器。p_env 指標指向現行緒的 JNI 介面指標所在的位置。thr_args 包含 VM 特定的緒附加引數。

- **DetachCurrentThread**

  簽章：

  ```
  jint DetachCurrentThread(JavaVM *vm);
  ```

  vm 識別要分離緒的 Java 虛擬機器。

如需「呼叫 API」函數的完整說明，請參閱 Sun Microsystems 公司的 Java 原生介面規格，或 The Source for Java Technology java.sun.com。

多重 Java 虛擬機器的支援:

iSeries 伺服器上的 Java 不再支援於單一工作或程序內建立多個 Java 虛擬機器 (JVM)。此項限制僅影響使用「Java 原生介面呼叫 (JNI) API」來建立 JVM 的使用者。這項支援上的變更並不影響您使用 java 指令來執行 Java 程式的方式。

您無法順利地在一個工作中多次呼叫 JNI_CreateJavaVM()， JNI_GetCreatedJavaVMs() 也無法在一列結果中傳回多個 JVM。

在單一工作或程序中只能建立單一 JVM 的這項支援，所遵循的是 Sun Microsystems 公司的 Java 參照實作方式標準。

範例：Java 呼叫 API:

此範例遵循標準的「呼叫 API」參照範例。

它執行下列動作：

- 使用 JNI_CreateJavaVM 建立 Java 虛擬機器。
- 使用 Java 虛擬機器尋找您要執行的類別檔案。
- 尋找類別中 main 方法的 methodID。
- 呼叫類別的 main 方法。
- 報告異常發生時的錯誤。

建立程式時，QJVAJNI 或 QJVAJNI64 服務程式會提供 JNI_CreateJavaVM「呼叫 API」函數。JNI_CreateJavaVM 會建立 Java 虛擬機器。

註: QJVAJNI64 是兆空間/LLP64 原生方法及「呼叫 API」支援新增的服務程式。

這些服務程式就在系統連結目錄中，您並不需要在控制語言 (CL) 建立指令上明確地指出它們。例如，使用「建立程式 (CRTPGM)」指令或「建立服務程式 (CRTSRVPGM)」指令時，您不需要明確指出前述的服務程式。

使用下列控制語言指令是執行此程式的方法之一：

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```

建立 Java 虛擬機器的任何工作，皆必須具備多緒能力。主要程式的輸出，以及程式的任何輸出，最後都會進入 QPRINT 排存檔。利用「使用提出的工作 (WRKSBMJOB)」控制語言 (CL) 指令，檢視您先前以「提出工作 (SBMJOB)」CL 指令所啓動的工作，就可以看到這些排存檔。

範例：使用 Java 呼叫 API

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
#define OS400_JVM_12
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

/* Specify the pragma that causes all literal strings in the
  * source code to be stored in ASCII (which, for the strings
  * used, is equivalent to UTF-8)
 */

    #pragma convert(819)

/* Procedure: Oops
 *
 * Description: Helper routine that is called when a JNI function
 *              returns a zero value, indicating a serious error.
 *              This routine reports the exception to stderr and
 *              ends the JVM abruptly with a call to FatalError.
 *
 * Parameters:  env -- JNIEnv* to use for JNI calls
 *              msg -- char* pointing to error description in UTF-8
 *
 * Note:        Control does not return after the call to FatalError
 *              and it does not return from this procedure.
 */

void Oops(JNIEnv* env, char *msg) {
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*env)->FatalError(env, msg);
}

/* This is the program's "main" routine. */
int main (int argc, char *argv[])
{

    JavaVMInitArgs initArgs; /* Virtual Machine (VM) initialization structure, passed by
                              * reference to JNI_CreateJavaVM(). See jni.h for details
                              */
    JavaVM* myJVM;           /* JavaVM pointer set by call to JNI_CreateJavaVM */
    JNIEnv* myEnv;           /* JNIEnv pointer set by call to JNI_CreateJavaVM */
    char*   myClasspath;     /* Changeable classpath 'string' */
    jclass  myClass;         /* The class to call, 'NativeHello'. */
    jmethodID mainID;        /* The method ID of its 'main' routine. */
    jclass  stringClass;     /* Needed to create the String[] arg for main */
    jobjectArray args;       /* The String[] itself */
    JavaVMOption options[1]; /* Options array -- use options to set classpath */
    int     fd0, fd1, fd2;   /* file descriptors for IO */

    /* Open the file descriptors so that IO works. */
    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);

   /*  Set the version field of the initialization arguments for J2SDK v1.3. */
   initArgs.version = 0x00010002;
/*  To use J2SDK v1.4, set initArgs.version = 0x00010004; */
 /*  To use J2SDK v1.5, set initArgs.version = 0x00010005; */


   /* Now, you want to specify the directory for the class to run in the classpath.
    * with  Java2, classpath is passed in as an option.
    * Note: You must specify the directory name in UTF-8 format. So, you wrap
    *       blocks of code in #pragma convert statements.
    */
   options[0].optionString="-Djava.class.path=/CrtJvmExample";
/*To use J2SDK v1.4 or v1.5, replace the '1.3' with '1.4' or '1.5'.
options[1].optionString="-Djava.version=1.3" */

   initArgs.options=options;  /* Pass in the classpath that has been set up. */
   initArgs.nOptions = 2;     /* Pass in classpath and version options */

   /*  Create the JVM -- a nonzero return code indicates there was
    *  an error. Drop back into EBCDIC and write a message to stderr
    *  before exiting the program.
    */
   if (JNI_CreateJavaVM("myJVM, (void **)"myEnv, (void *)"initArgs)) {
 #pragma convert(0)
       fprintf(stderr, "Failed to create the JVM\n");
 #pragma convert(819)
       exit(1);
   }

   /*  Use the newly created JVM to find the example class,
    *  called 'NativeHello'.
    */
   myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
   if (! myClass) {
       Oops(myEnv, "Failed to find class 'NativeHello'");
   }

   /* Now, get the method identifier for the 'main' entry point
    * of the class.
    * Note: The signature of 'main' is always the same for any
    *       class called by the following java command:
    *           "main" , "([Ljava/lang/String;)V"
    */
   mainID = (*myEnv)->GetStaticMethodID(myEnv,myClass,"main",
                                        "([Ljava/lang/String;)V");
   if (! mainID) {
       Oops(myEnv, "Failed to find jmethodID of 'main'");
   }

   /* Get the jclass for String to create the array
    * of String to pass to 'main'.
    */
   stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
   if (! stringClass) {
       Oops(myEnv, "Failed to find java/lang/String");
   }

   /*  Now, you need to create an empty array of strings,
    *  since main requires such an array as a parameter.
    */
   args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
   if (! args) {
       Oops(myEnv, "Failed to create args array");
   }
```

```
    /* Now, you have the methodID of main and the class, so you can
     * call the main method.
     */
    (*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);

    /* Check for errors. */
    if ((*myEnv)->ExceptionOccurred(myEnv)) {
        (*myEnv)->ExceptionDescribe(myEnv);
    }

    /* Finally, destroy the JavaVM that you created. */
    (*myJVM)->DestroyJavaVM(myJVM);

    /* All done. */
    return 0;
}
```

如需相關資訊，請參閱 Java 呼叫 API。

Java 原生方法及緒注意事項

原生方法可以存取 Java 未提供的功能。若要透過原生方法更妥善地運用 Java，請務必對下列概念有所瞭解。

- 就 Java 緒來說，無論它是由 Java 還是附加的原生緒所建立，皆停用所有浮點數異常。若緒所執行的原生方法重新啓用了浮點數異常，則 Java 不會再予以停用。若使用者應用程式在回去執行 Java 程式碼之前未停用浮點數異常，萬一發生浮點數異常時，Java 程式碼將無法正確執行。當原生緒與 Java 虛擬機器分離時，浮點數異常遮罩會還原成最初附加時生效的值。
- 當原生緒附加至 Java 虛擬機器時，Java 虛擬機器會在必要時，根據 Java 定義的 1 至 10 優先順序標準來改變緒的優先順序。當緒分離時，優先順序就會還原。而於附加之後，緒就可以使用原生方法介面 (例如 POSIX API) 來改變緒的優先順序。 Java 在返回 Java 虛擬機器的轉移期間，不會改變緒的優先順序。
- 「Java 原生介面 (JNI)」的「呼叫 API」元件，允許使用者將 Java 虛擬機器內嵌到應用程式中。若應用程式建立 Java 虛擬機器，但 Java 虛擬機器異常結束，只要在 Java 虛擬機器結束時，緒已附加至 Java 虛擬機器，此時，就會向程序的起始緒發出 MCH74A5 ″Java Virtual Machine Terminated″ iSeries 異常。下列原因會造成 Java 虛擬機器異常結束：
  - 使用者呼叫 java.lang.System.exit() 方法。
  - Java 虛擬機器需要的緒結束。
  - Java 虛擬機器發生內部錯誤。

  但此行爲不同於其他大部分的 Java 平台。其他大部分的平台上，每當 Java 虛擬機器結束時，自動建立 Java 虛擬機器的程序也會突然結束。若應用程式可監督並處理 MCH74A5 異常通知，程序也許能夠繼續執行。否則，當異常無法得到妥善處理時，程序就隨之結束。雖然可以加入程式碼來處理 iSeries 伺服器特有的 MCH74A5 異常，卻也可能降低應用程式的可攜性，無法用於其他平台。

原生方法一律在多緒的程序中執行，因此其內含的程式碼必須符合安全緒標準。這也造成原生方法所採用的語言及函數，受到以下的限制：

- 不應在原生方法中使用 ILE CL，因爲這種語言不符合安全緒標準。若要執行符合安全緒標準的 CL 指令，您可以使用 C 語言的 system() 函數或 java.lang.Runtime.exec() 方法。
  - 在 C 或 C++ 原生方法內，使用 C 語言的 system() 函數來執行符合安全緒標準的 CL 指令。
  - 直接從 Java 中使用 java.lang.Runtime.exec() 方法來執行符合安全緒標準的 CL 指令。
- 您可以使用 ILE C、ILE C++、ILE COBOL 及 ILE RPG 來編寫原生方法，但從原生方法內所呼叫的函數，全部必須符合安全緒標準。

附註：編寫原生方法的編譯時期支援，目前僅支援 C、C++ 及 RPG 語言。採用其他語言來編寫原生方法雖然可行，但可能更加複雜。

注意：***並非所有標準的 C、C++、COBOL 或 RPG 函數都符合安全緒標準***。

- 絕對不要在原生方法內使用 C 和 C++ 的 exit() 及 abort() 函數。這兩個函數會導致執行 Java 虛擬機器的程序完全停止。這其中包括程序內所有的緒，無論它們是否源自 Java，全部都會停止。

  註: 這裡提到的 exit() 函數是指 C 及 C++ 函數，與 java.lang.Runtime.exit() 方法不同。

如需 iSeries 伺服器上緒的相關資訊，請參閱多緒的應用程式。

原生方法及 Java 原生介面

原生方法是指在 Java 以外的其他語言中啓動的 Java 方法。原生方法可以存取 Java 中無法直接使用的系統專用函數及 API。

採用原生方法會限制應用程式的可攜性，因爲這涉及系統專用的程式碼。原生方法可能是新的原有程式碼陳述式，或是呼叫現存原有程式碼的原有程式碼陳述式。

決定需要某個原生方法之後，可能需要與執行此方法的 Java 虛擬機器進行互用。「Java 原生介面 (JNI)」採取一種平台中立的方式來達成這項互用性。

JNI 是一組介面，可提供許多方式讓原生方法與 Java 虛擬機器進行互用。例如，JNI 包含多種介面，可以建立新的物件和呼叫方法、取得欄位和設定欄位、處理異常，以及操作字串和陣列。

如需 JNI 的完整說明，請參閱 Sun Microsystems 公司的 Java 原生介面，或 The Source for Java Technology java.sun.com。

原生方法的字串

許多「Java 原生介面 (JNI)」函數都接受將 C 語言樣式的字串當作參數。例如，FindClass() JNI 函數即接受含類別檔案完整名稱的字串參數。 FindClass 會載入找到的類別檔案，並且將檔案的參照傳回 FindClass 的呼叫程式。

所有 JNI 函數皆預期字串參數爲 UTF-8 編碼。如需 UTF-8 的詳細資料，請參閱 JNI Specification，但通常您只需瞭解 7 位元「美國國家標準交換碼 (ASCII)」字元等同於其 UTF-8 表示法即可。 7 位元 ASCII 字元實際上就是 8 位元字元，只是第一個位元一律爲 0。因此，大部份 ASCII C 字串，實際上已經是 UTF-8。

依預設，iSeries 伺服器的 C 編譯器會以延伸的二進位編碼十進位交換碼 (EBCDIC) 來操作，因此您可提供 UTF-8 的字串給 JNI 函數。這有兩種作法。您可以使用文字字串或動態字串。文字字串指編譯原始程式碼時可得知字串值的字串。 動態字串指編譯時尚無法得知字串值的字串，其值在執行時才會實際計算出來。

原生方法的文字字串:

字串由 7 位元「美國國家標準交換碼 (ASCII)」表示法的字元組成時， 較易於將文字字串編碼成 UTF-8。

就像大多數的情況一樣，若可用 ASCII 來呈現字串， 則可利用 pragma 陳述式來括住字串，此陳述式會變更編譯器的現行字碼頁。然後，編譯器在內部就會以 JNI 所要求的 UTF-8 格式來儲存字串。 若無法以 ASCII 來呈現字串，亦可直接將原始延伸的二進位編碼十進位交換碼 (EBCDIC) 字串視爲動態字串， 經過 iconv() 處理之後再傳至 JNI。如需動態字串的詳細資訊，請參閱動態字串。

例如，要尋找名爲 `java/lang/String` 的類別時，程式碼如下：

```
#pragma convert(819)
myClass = (*env)->FindClass(env,"java/lang/String");
#pragma convert(0)
```

第一個 pragma 含有數字 819，指示編譯器以 ASCII 來儲存後面以雙引號括住的所有字串 (文字字串)。第二個 pragma 含有數字 0，則指示編譯器恢復爲預設字碼頁來處理雙引號括住的字串，通常爲 EBCDIC 字碼頁 37。 因此，將這些 pragma 將此呼叫括起來之後， 即可滿足字串參數必須以 UTF-8 編碼的這項 JNI 要求。

注意：請多加留意替代文字。比方說，如果程式碼類似：

```
#pragma convert(819)
#define MyString "java/lang/String"
#pragma convert(0)
myClass = (*env)->FindClass(env,MyString);
```

則最後產生的字串爲 EBCDIC，因爲在編譯期間，FindClass 呼叫內會替換成 MyString 的值。 在進行替代動作時，pragma 數字 819 不會生效。因爲，不會以 ASCII 來儲存文字字串。

在 EBCDIC、Unicode 及 UTF-8 之間轉換動態字串:

爲了操作執行時間才運算的字串變數，可能需要在 EBCDIC、Unicode 及 UTF-8 之間轉換字串。

負責字碼頁轉換功能的系統 API 是 iconv()。請遵循下列步驟來使用 iconv()：

1. 使用 `QtqIconvOpen()` 建立轉換描述子。
2. 呼叫 `iconv()`，使用此描述子來轉換字串。
3. 使用 `iconv_close` 關閉描述子。

在使用「Java 原生介面」的原生方法範例中的範例 3，常式先建立及使用 iconv 轉換描述子，然後在常式內直接摧毀此描述子。此設計可避免 iconv_t 描述子的多緒運用問題，但對於注重效能的程式碼，最好在靜態儲存體中建立轉換描述子，然後利用互斥 (mutex) 或其他同步化機制來調整多重存取。

IBM i5/OS PASE 適用於 Java 的原生方法

iSeries Java 虛擬機器 (JVM) 支援使用在 i5/OS PASE 環境中執行的原生方法。在 V5R2 以前，原有的 iSeries JVM 僅使用 ILE 原生方法。

i5/OS PASE 原生方法的支援包括：

- 完全可以從 i5/OS PASE 原生方法中使用原有的 iSeries Java 原生介面 (JNI)
- 能夠從原有的 iSeries JVM 呼叫 i5/OS PASE 原生方法

這項新的支援可讓您輕易地將 AIX 中執行的 Java 應用程式移轉至 iSeries 伺服器。您可以將類別檔案及 AIX 原生方法檔案庫複製到 iSeries 的整合檔案系統，然後透過任何控制語言 (CL)、Qshell 或 i5/OS PASE 終端機階段作業指令提示來加以執行。

相關資訊

i5/OS PASE
本資訊假設您熟悉 i5/OS PASE。如果您尚未熟悉 PASE，請參閱本主題以進一步瞭解如何與 Java 搭配使用 IBM i5/OS PASE 原生方法。

Java i5/OS PASE 環境變數

Java 虛擬機器 (JVM) 使用下列變數來啓動 i5/OS PASE 環境。您必須設定 QIBM_JAVA_PASE_STARTUP 變數，才能執行 IBM i5/OS PASE 適用於 Java 的原生方法範例。

如需設定範例環境變數的相關資訊，請參閱下列主題：

IBM i5/OS PASE 範例的環境變數。

QIBM_JAVA_PASE_STARTUP

若以下兩種條件都成立，則必須設定此環境變數：

- 使用 i5/OS PASE 原生方法
- 從 iSeries 指令提示或 Qshell 指令提示啓動 Java

JVM 會使用此環境變數來啓動 PASE 環境。該變數的值可識別 i5/OS PASE 啓動程式。iSeries 伺服器包含兩個 i5/OS PASE 啓動程式：

- /usr/lib/start32：啓動 32 位元 i5/OS PASE 環境
- /usr/lib/start64：啓動 64 位元 i5/OS PASE 環境

i5/OS PASE 環境使用之所有共用檔案庫物件的位元格式，都必須符合 i5/OS PASE 環境的位元格式。

從 i5/OS 終端機階段作業啓動 Java 時，無法使用此變數。i5/OS PASE 終端機階段作業一律使用 32 位元 i5/OS PASE 環境。任何從 i5/OS PASE 終端機階段作業中啓動的 JVM，皆與終端機階段作業使用相同類型的 PASE 環境。

QIBM_JAVA_PASE_CHILD_STARTUP

當次要 JVM 的 i5/OS PASE 環境必須不同於主要 JVM 的 i5/OS PASE 環境時，請設定此選用性環境變數。在 Java 中呼叫 Runtime.exec() 會啓動次要 (或子項) JVM。

如需詳細資訊，請參閱使用 QIBM_JAVA_PASE_CHILD_STARTUP。

QIBM_JAVA_PASE_ALLOW_PREV

當希望使用現行 i5/OS PASE 環境 (若已存在一個的話)，請設定此選用性環境變數。在某些狀況下，會很難判斷 i5/OS PASE 環境是已否存在。只要將 QIBM_JAVA_PASE_ALLOW_PREV 與 QIBM_JAVA_PASE_STARTUP 組合起來使用，即可讓 JVM 使用現有 i5/OS PASE 環境，或啓動新的 i5/OS PASE 環境。

如需相關資訊，請參閱使用 QIBM_JAVA_PASE_ALLOW_PREV。

範例：IBM i5/OS PASE 範例的環境變數:

若要使用 IBM i5/OS PASE 適用於 Java 的原生方法範例，必須設定環境變數。

PASE_LIBPATH

iSeries 伺服器使用此 i5/OS PASE 環境變數來識別 i5/OS PASE 原生方法檔案庫的位置。路徑可以設爲單一目錄或多個目錄。若爲多個目錄，請使用冒號 (:) 來隔開不同的目錄。 伺服器亦可使用 LIBPATH 環境變數。

如需使用 Java、原生方法檔案庫及此範例之 PASE_LIBPATH 的相關資訊，請參閱使用 Java、i5/OS PASE 及原生方法檔案庫。

PASE_THREAD_ATTACH

若此 i5/OS PASE 環境變數設爲 Y，將會使原本並非由 i5/OS PASE 啓動的 ILE 緒，自動附加至 i5/OS PASE (在呼叫 i5/OS PASE 程序時)。

如需 i5/OS PASE 環境變數的相關資訊，請參閱使用 i5/OS PASE 環境變數。

QIBM_JAVA_PASE_STARTUP

JVM 會使用此環境變數來啓動 i5/OS PASE 環境。該變數的值可識別 i5/OS PASE 啓動程式。

如需相關資訊，請參閱 Java i5/OS PASE 變數。

使用 QIBM_JAVA_PASE_CHILD_STARTUP:

QIBM_JAVA_PASE_CHILD_STARTUP 環境變數表示任何次要 JVM 的 i5/OS PASE 啓動程式。

下列條件全部成立時，才使用 QIBM_JAVA_PASE_CHILD_STARTUP：

- 要執行的 Java 應用程式透過 Runtime.exec() 的 Java 呼叫來建立 Java 虛擬機器 (JVM)
- 主要及次要 JVM 皆使用 i5/OS PASE 原生方法
- 次要 JVM 的 i5/OS PASE 環境必須不同於主要 JVM 的 i5/OS PASE 環境

若上述條件全部成立，請執行下列動作：

- 將 QIBM_JAVA_PASE_CHILD_STARTUP 環境變數設爲次要 JVM 的 i5/OS PASE 啓動程式。
- 從 iSeries 指令提示或 Qshell 指令提示啓動主要 JVM 時，請將 QIBM_JAVA_PASE_STARTU 環境變數設爲主要 JVM 的i5/OS PASE 啓動程式。

註: 從 i5/OS PASE 終端機階段作業啓動主要 JVM 時，請勿設定 QIBM_JAVA_PASE_STARTUP。

次要 JVM 的程序會繼承 QIBM_JAVA_PASE_CHILD_STARTUP 環境變數。此外，i5/OS 也會將次要 JVM 程序的 QIBM_JAVA_PASE_STARTUP 環境變數，設爲上層處理的 QIBM_JAVA_PASE_CHILD_STARTUP 環境變數值。

針對指令環境及 QIBM_JAVA_PASE_STARTUP 與 QIBM_JAVA_PASE_CHILD_STARTUP 定義的各種組合，下表指出最終可能形成的 i5/OS PASE 環境 (如果有的話)：

表 1. QIBM_JAVA_PASE_STARTUP 及 QIBM_JAVA_PASE_CHILD_STARTUP 形成的 PASE 環境

啓動環境			結果行為	
指令環境	**QIBM_JAVA_PASE_STARTUP**	**主要 JVM i5/OS PASE 啓動**	**主要 JVM i5/OS PASE 啓動**	**次要 JVM i5/OS PASE 啓動**
CL 或 QSH	定義 startX	定義 startY	使用 startX	使用 startY
CL 或 QSH	定義 startX	未定義	使用 startX	使用 startX
CL 或 QSH	未定義	定義 startY	無 i5/OS PASE 環境	使用 startY
CL 或 QSH	未定義	未定義	無 i5/OS PASE 環境	無 i5/OS PASE 環境
i5/OS PASE 終端機階段作業	定義 startX	定義 startY	不允許*	不允許*
i5/OS PASE 終端機階段作業	定義 startX	未定義	不允許*	不允許*
i5/OS PASE 終端機階段作業	未定義	定義 startY	使用 i5/OS PASE 終端機階段作業環境	使用 startY
i5/OS PASE 終端機階段作業	未定義	未定義	使用 i5/OS PASE 終端機階段作業環境	無 i5/OS PASE 環境

* 標示爲「不允許」的橫列表示 QIBM_JAVA_PASE_STARTUP 環境變數與 i5/OS PASE 終端機階段作業會發生衝突。由於潛在的衝突性，所以不允許在 i5/OS PASE 終端機階段作業中使用 QIBM_JAVA_PASE_STARTUP。

使用 QIBM_JAVA_PASE_ALLOW_PREV:

有時很難判斷 i5/OS PASE 環境是否存在。將選用性的環境變數 QIBM_JAVA_PASE_ALLOW_PREV 與 QIBM_JAVA_PASE_STARTUP 組合起來使用，即可讓 JVM 判斷應該使用現行 i5/OS PASE 環境 (若其存在)，還是啓動新的 i5/OS PASE 環境。

若要將這兩個環境變數組合使用，請將它們設爲下列值：

- 將 QIBM_JAVA_PASE_STARTUP 設爲預設的啓動程式
- 將 QIBM_JAVA_PASE_ALLOW_PREV 設爲 1

例如，有一個選擇另外啓動 i5/OS PASE 環境的應用程式，呼叫程式來啓動 JVM。在此情況下，若利用上述設定值，程式即可使用現行 i5/OS PASE 環境 (若其存在)，或啓動新的 i5/OS PASE 環境。

針對 i5/OS PASE 環境及 QIBM_JAVA_PASE_STARTUP 與 QIBM_JAVA_PASE_ALLOW_PREV 定義的各種組合，下表識別任何可能形成的 i5/OS PASE 環境：

表 2. 由組合 i5/OS PASE 環境及 QIBM_JAVA_PASE_STARTUP 與 QIBM_JAVA_PASE_ALLOW_PREV 之定義而形成的 i5/OS PASE 環境

啓動環境			結果行為
i5/OS PASE 環境	**QIBM_JAVA _PASE_STARTUP**	**QIBM_JAVA_PASE _ALLOW_PREV**	**JVM i5/OS PASE 啓動**
無	未定義	未定義*	無 i5/OS PASE 環境
無	未定義	定義 '1'	無 i5/OS PASE 環境
無	定義 startX	未定義*	使用 startX
無	定義 startX	定義 '1'	使用 startX
已啓動	未定義	未定義*	使用現有的 i5/OS PASE 環境
已啓動	未定義	定義 '1'	使用現有的 i5/OS PASE 環境
已啓動	定義 startX	未定義*	不允許：啓動期間發生 JVM 錯誤
已啓動	定義 startX	定義 '1'	使用現有的 i5/OS PASE 環境

* 「未定義」表示 QIBM_JAVA_PASE_ALLOW_PREV 未納入或值不是 1。

上表最後兩列指出適合設定 QIBM_JAVA_PASE_ALLOW_PREV 的狀況。 若 i5/OS PASE 環境存在，且您定義了 QIBM_JAVA_PASE_STARTU，則 JVM 會檢查 QIBM_JAVA_PASE_ALLOW_PREV。否則，JVM 會忽略 QIBM_JAVA_PASE_ALLOW_PREV。

QIBM_JAVA_PASE_ALLOW_PREV 及 QIBM_JAVA_PASE_CHILD_STARTUP 環境變數彼此不相關。

Java i5/OS PASE 錯誤碼

爲了協助您對 i5/OS PASE 原生方法進行疑難排解，本主題說明 i5/OS 工作日誌訊息及 Java 執行時間異常指出的錯誤狀況。下列清單將針對使用的 i5/OS PASE 適用於 Java 的原生方法，說明在啓動或執行時間可能遇到的錯誤。

啓動錯誤

針對啓動錯誤，請檢查適當工作日誌中的訊息。

執行時間錯誤

除了啓動錯誤之外，JVM 的 Qshell 輸出還可能出現 PaseInternalError 或 PaseExit Java 異常：

- PaseInternalError - 指出內部系統錯誤。請檢查「授權內碼日誌」項目。

  如需 PaseInternalError 錯誤碼的詳細資訊，請參閱 Qp2CallPase。

- PaseExit - 可能是 i5/OS PASE 應用程式呼叫 exit() 函數，或 i5/OS PASE 環境異常結束。請檢查「工作日誌」及「授權內碼日誌」來取得其他資訊。

管理原生方法檔案庫

若要使用原生方法檔案庫，尤其是要在 iSeries 伺服器上管理多個版本的原生方法檔案庫時，必須瞭解 Java 檔案庫命名慣例及檔案庫搜尋演算法。

i5/OS 會根據 Java 虛擬機器 (JVM) 載入的檔案庫，使用第一個名稱相符的原生方法檔案庫。爲了確保 i5/OS 找得到正確的原生方法，必須避免檔案庫名稱出現衝突，也要避免不易判斷 JVM 究竟使用哪個原生方法檔案庫的混淆情形。

i5/OS PASE 及 AIX Java 檔案庫命名慣例

假設 Java 程式碼載入的檔案庫名爲 Sample，則對應的可執行檔必須命名爲 libSample.a 或 libSample.so。

Java 檔案庫搜尋次序

當您啓用 i5/OS PASE 原生方法供 JVM 使用時，伺服器會使用三個不同的清單 (依下列次序) 來建立單一原生方法檔案庫搜尋路徑：

1. i5/OS 檔案庫清單
2. LIBPATH 環境變數
3. PASE_LIBPATH 環境變數

爲了執行搜尋，i5/OS 會將檔案庫清單轉換成整合檔案系統格式。QSYS 檔案系統物件在整合檔案系統中有其相對的名稱，但部份整合檔案系統物件則沒有相對的 QSYS 檔案系統名稱。 因爲檔案庫載入器會同時在 QSYS 檔案系統及整合檔案系統中尋找物件，所以 i5/OS 使用整合檔案系統格式來搜尋原生方法檔案庫。

下表顯示 i5/OS 如何將檔案庫清單的項目，轉換成整合檔案系統格式：

檔案庫清單項目	整合檔案系統格式
QSYS	/qsys.lib
QSYS2	/qsys.lib/qsys2.lib
QGPL	/qsys.lib/qgpl.lib
QTEMP	/qsys.lib/qtemp.lib

範例：搜尋 Sample2 檔案庫

在下列範例中，LIBPATH 設爲 /home/user1/lib32:/samples/lib32，PASE_LIBPATH 設爲 /QOpenSys/samples/lib。

此表格由上而下指出完整的搜尋路徑：

來源	整合檔案系統目錄
檔案庫清單	/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

註: 僅 /QOpenSys 路徑中的字元需區分大小寫。

爲了搜尋檔案庫 Sample2，Java 檔案庫載入器會依下列次序搜尋候選檔案：

1. /qsys.lib/sample2.srvpgm
2. /qsys.lib/libSample2.a
3. /qsys.lib/libSample2.so
4. /qsys.lib/qsys2.lib/sample2.srvpgm
5. /qsys.lib/qsys2.lib/libSample2.a
6. /qsys.lib/qsys2.lib/libSample2.so
7. /qsys.lib/qgpl.lib/sample2.srvpgm
8. /qsys.lib/qgpl.lib/libSample2.a
9. /qsys.lib/qgpl.lib/libSample2.so
10. /qsys.lib/qtemp.lib/sample2.srvpgm
11. /qsys.lib/qtemp.lib/libSample2.a
12. /qsys.lib/qtemp.lib/libSample2.so
13. /home/user1/lib32/sample2.srvpgm
14. /home/user1/lib32/libSample2.a
15. /home/user1/lib32/libSample2.so
16. /samples/lib32/sample2.srvpgm
17. /samples/lib32/libSample2.a
18. /samples/lib32/libSample2.so
19. /QOpenSys/samples/lib/SAMPLE2.srvpgm
20. /QOpenSys/samples/lib/libSample2.a
21. /QOpenSys/samples/lib/libSample2.so

i5/OS 會將清單中第一個實際存在的候選檔案，載入到 JVM 中成爲原生方法檔案庫。即使 /qsys.lib/libSample2.a 及 /qsys.lib/libSample2.so 這些候選檔案也在搜尋之列，仍然不可能在 /qsys.lib 目錄中建立整合檔案系統檔案或符號鏈結。因此，即使 i5/OS 會檢查這些候選檔案，也絕對無法在以 /qsys.lib 開頭的整合檔案系統目錄中找到它們。

然而，還是可以從其他整合檔案系統目錄中，建立任意符號鏈結來指向 QSYS 檔案系統中的 i5/OS 物件。因此，舉例而言，/home/user1/lib32/sample2.srvpgm 就屬於有效的候選檔案。

範例：IBM i5/OS PASE 適用於 Java 的原生方法

IBM i5/OS PASE 適用於 Java 的原生方法範例會呼叫原有的 C 方法實例，此實例再利用「Java 原生介面 (JNI)」來回呼 Java 程式碼。此範例並不直接從此 Java 程式碼中存取字串，而是呼叫原生方法，原生方法再透過 JNI 回呼 Java 來取得字串值。

如需 HTML 版的來源檔範例，請使用下列鏈結：

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

- PaseExample1.java
- PaseExample1.c

執行 i5/OS PASE 原生方法範例之前，必須先完成下列作業：

1. 將範例原始程式碼下載至 AIX 工作站
2. 準備範例原始程式碼
3. 準備 iSeries 伺服器

執行 i5/OS PASE 適用於 Java 的原生方法範例

完成上述作業之後，即可開始執行範例。請使用下列指令來執行範例程式：

- 從 iSeries 伺服器指令提示：

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- 從 Qshell 指令提示或 i5/OS PASE 終端機階段作業：

```
cd /home/example
     java PaseExample1
```

適用於 Java 的兆空間儲存模型原生方法

iSeries Java 虛擬機器 (JVM) 目前支援使用兆空間儲存模型原生方法。兆空間儲存體模型爲 ILE 程式提供了一個大型處理、本端位址的環境。 使用兆空間可讓您將原生方法程式碼從其他作業系統移轉至 i5/OS，而原始程式碼只需少許更改，甚至完全不用變動。

如需以兆空間儲存體模型來設計程式的詳細資料，請參閱下列資訊：

ILE Concepts 第 4 章

WebSphere Development Studio ILE C/C++ Programmer's Guide 第 17 章

針對兆空間儲存模型所建立的 Java 原生方法，概念上類似於使用單層儲存體的原生方法。JVM 會傳遞一個指向「Java 原生介面 (JNI)」環境的指標給兆空間原生方法，供這些方法用於呼叫 JNI 函數。

對於兆空間儲存體模型原生方法，JVM 採用兆空間儲存體模型及 8 位元組指標來提供 JNI 函數的實作方式。

建立兆空間原生方法

爲了順利建立兆空間儲存體模型原生方法，兆空間模組建立指令需使用下列選項：

```
TERASPACE(*YES) STGMDL(*TERASPACE) DTAMDL(*LLP64)
```

以下爲使用兆空間儲存體函數的選用選項 (*TSIFC)：

```
TERASPACE(*YES *TSIFC)
```

註: 當使用兆空間儲存模型 Java 原生方法時，如果不使用 DTAMDL(*LLP64)，則呼叫原生方法時將丟出執行時間異常。

建立使用原生方法的兆空間服務程式

若要建立兆空間儲存體模型服務程式，請在「建立服務程式 (CRTSRVPGM)」控制語言 (CL) 指令上使用下列選項：

```
CRTSRVPGM STGMDL(*TERASPACE)
```

此外，您應該使用 ACTGRP(*CALLER) 選項，若使用此選項，JVM 即可啓動所有兆空間儲存模型原生方法服務程式，形成相同的兆空間啓動群組。爲了讓原生方法能夠有效地處理異常，最好以這種方式使用兆空間啓動群組。

如需程式啓動及啓動群組的其他詳細資料，請參閱下列資訊：

ILE Concepts 第 3 章

搭配使用 Java 呼叫 API 及兆空間原生方法

當 JNI 環境指標不符合服務程式的儲存體模型時，請使用「呼叫 API」的 GetEnv 函數。 「呼叫 API」的 GetEnv 函數一定會傳回正確的 JNI 環境指標。如需詳細資訊，請參閱以下各頁：

Java 呼叫 API

JNI 加強功能

JVM 同時支援單層及兆空間儲存體模型原生方法，但這兩種儲存體模型各使用不同的 JNI 環境。 由於這兩個儲存模型使用不同的 JNI 環境，因此請勿在其原生方法之間，將 JNI 環境指標當作參數來傳遞。

整合語言環境® 與 Java 的比較

iSeries 伺服器上的 Java 環境與整合語言環境 (ILE) 是各自獨立的。Java 並非 ILE 語言，亦無法連結 ILE 物件模組而在 iSeries 伺服器上建立程式或服務程式。

ILE	Java
作爲 iSeries 伺服器上檔案庫或檔案結構一部分的成員，會儲存原始程式碼。	整合檔案系統中的串流檔包含原始程式碼。
「原始檔登錄公用程式 (SEU)」可編輯「延伸的二進位編碼十進位交換碼 (EBCDIC)」來源檔。	「美國國家標準交換碼 (ASCII)」來源檔通常透過工作站編輯器來編輯。
來源檔會編譯成目的碼模組，儲存於 iSeries 伺服器的檔案庫內。	編譯成類別檔案的原始程式碼，由整合檔案系統儲存。
物件模組在程式或服務程式中靜態地連結在一起。	執行時根據需要才動態地載入類別。
可以直接呼叫以其他 ILE 程式設計語言所編寫的函數。	必須使用「Java 原生介面」從 Java 呼叫其他語言。
ILE 語言一律當成機器指令來編譯與執行。	Java 程式可加以解譯或編譯。

使用 java.lang.Runtime.exec()

您可以從 Java 程式使用 java.lang.Runtime.exec 方法來呼叫程式或指令。使用 java.lang.Runtime.exec() 方法可額外建立一或多個具有緒功能的工作。這些額外的工作可處理您在方法上傳送的指令字串。

註: java.lang.Runtime.exec 方法會在個別的工作中執行程式，與 C system() 函數不同。C system 函數會在相同工作中執行程式。

實際的處理程序視下列項目而定：

- 傳入 java.lang.Runtime.exec() 的指令類型
- os400.runtime.exec 系統內容的值

處理不同類型的指令

下表指出 java.lang.Runtime.exec() 如何處理不同類型的指令，並且顯示 os400.runtime.exec 系統內容的效果。

指令類型	os400.runtime.exec 系統內容的值	
	EXEC (預設值)	**QSHELL**
java 指令	啓動第二個工作來執行 JVM。JVM 再啓動第三個工作來執行 Java 應用程式。	啓動第二個工作來執行 Qshell 這個 Shell 直譯器。Qshell 再啓動第三個工作來執行 Java 應用程式、程式或指令。
程式	啓動第二個工作來執行可執行程式 (i5/OS 程式或 i5/OS PASE 程式)。	
CL 指令	啓動第二個工作來執行 i5/OS 程式。i5/OS 程式在第二個工作中執行 CL 指令。	

註: 在呼叫 CL 指令或 CL 程式時，請確定工作 CCSID 包含您當作參數傳遞給被呼叫指令的字元。

第二個或第三個工作的處理程序，會與原始工作中的任何 Java 虛擬機器 (JVM) 同時執行。這些工作中的任何結束程式或關閉程序，皆不影響原始的 JVM。

os400.runtime.exec 系統內容

您可以將 os400.runtime.exec 系統內容的值設爲 EXEC (預設值) 或 QSHELL。os400.runtime.exec 的值決定 java.lang.Runtime.exec() 將使用 EXEC 介面或 Qshell。

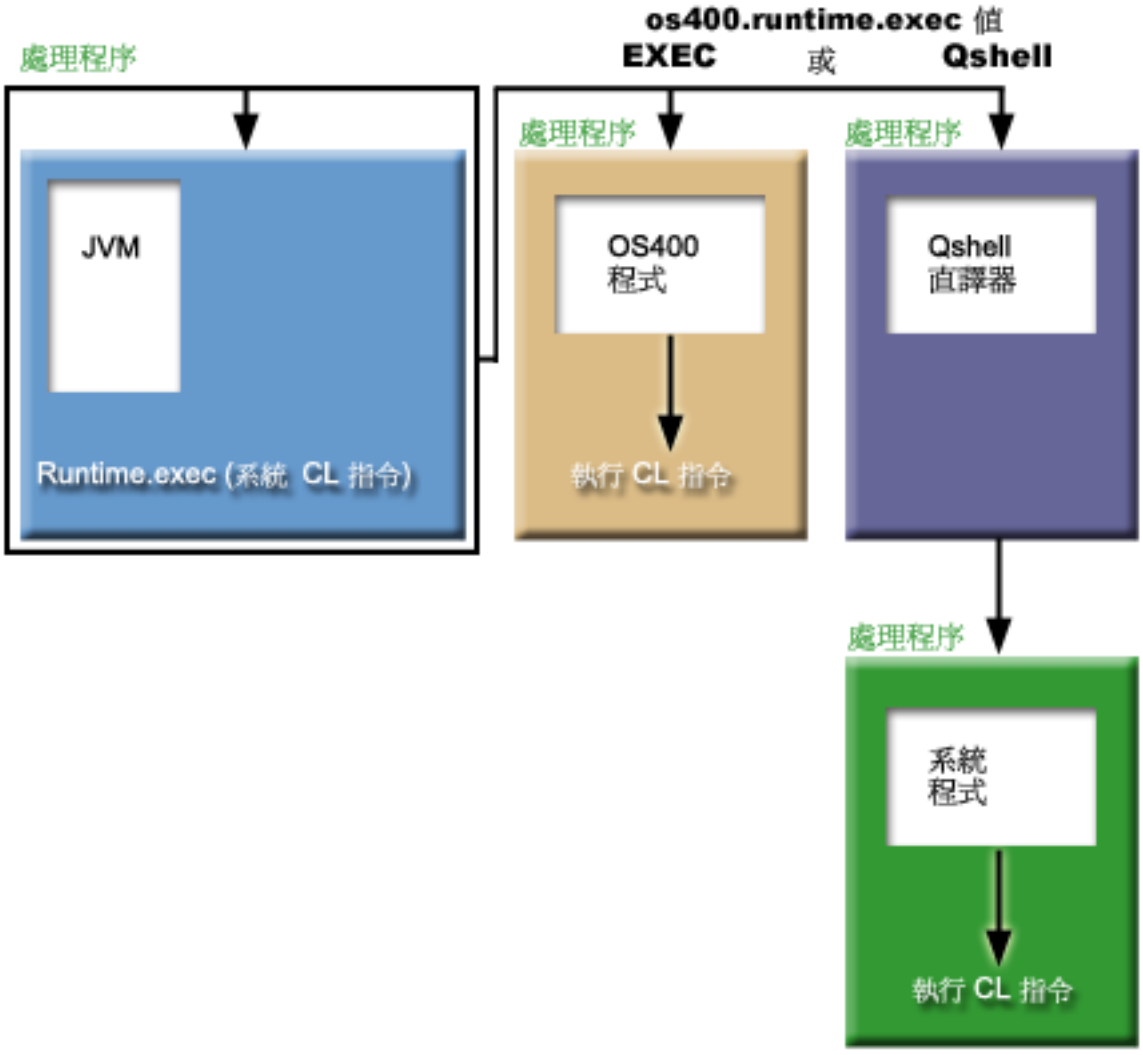
使用 EXEC 值而非 QSHELL，有下列優點：

- 呼叫 java.lang.Runtime.exec() 的 Java 程式更具可攜性
- 使用 java.lang.Runtime.exec() 來呼叫 CL 指令時，運用的系統資源較少

基於與舊版本相容的考量下，才應該使用 java.lang.Runtime.exec() 來執行 Qshell。若要使用 java.lang.Runtime.exec() 來執行 Qshell，必須將 os400.runtime.exec 設爲 QSHELL。

下列圖例顯示使用 QSHELL 值將會如何啓動第三個工作，也因此耗用更多系統資源。切記，使用 QSHELL 值會降低 Java 程式的可攜性。

圖 1. 在 os400.runtime.exec 系統內容中使用 QSHELL 值



另外，使用 QSHELL 值時，若要將 CL 指令傳給 java.lang.Runtime.exec()，必須要有特定的語法。如需相關資訊，請參閱下列 CL 指令呼叫範例。

如需設定 os400.runtime.exec 的相關資訊，請參閱 Java 系統內容的清單。

範例：使用 java.lang.Runtime.exec() 來呼叫另一個 Java 程式

下列範例說明如何使用 java.lang.Runtime.exec() 呼叫另一個 Java 程式。此類別會呼叫 IBM Developer Kit for Java 隨附的 Hello 程式。當 Hello 類別寫入 System.out 時，此程式就會取得串流的控點，然後從中讀取資料。

註: 您可以使用「Qshell 直譯器」來呼叫程式。

範例 1：CallHelloPgm 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallHelloPgm
{
```

```
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallHelloPgm.main() invoked");

      // call the Hello class
      try
      {
          theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
         e.printStackTrace();
      }

      // read from the called program's standard output stream
      try
      {
         inStream = new BufferedReader(
                              new InputStreamReader( theProcess.getInputStream() ));
         System.out.println(inStream.readLine());
      }
      catch (IOException e) {
System.err.println("Error on inStream.readLine()");
         e.printStackTrace();
      }

   } // end method

 } // end class
```

如需相關背景資訊，請參閱使用 java.lang.Runtime.exec()。

範例：使用 java.lang.Runtime.exec() 來呼叫 CL 程式

本範例顯示如何從 Java 程式內執行 CL 程式。在此範例中，Java 類別 CallCLPgm 會執行 CL 程式。

CL 程式使用「顯示 Java 程式 (DSPJVAPGM)」指令，顯示與 Hello 類別檔案相關的程式。本範例假設 CL 程式已完成編譯，存在於 JAVSAMPLIB 檔案庫中。CL 程式的輸出位於 QSYSPRT 儲存檔。

如需如何從 Java 程式內呼叫 CL 指令的範例，請參閱呼叫 CL 指令。

註: JAVSAMPLIB 並非由 IBM Developer Kit 授權程式 (LP) 編號 5722-JV1 的安裝程序所建立。您必須明確地建立此檔案庫。

範例 1： CallCLPgm 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallCLPgm
{
   public static void main(String[] args)
   {
       try
      {
                  Process theProcess =
                 Runtime.getRuntime().exec("/QSYS.LIB/JAVSAMPLIB.LIB/DSPJVA.PGM");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
```

```
        e.printStackTrace();
      }
   } // end main() method
} // end class
```

範例 2：顯示 Java CL 程式

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
                OUTPUT(*PRINT)
ENDPGM
```

如需相關背景資訊，請參閱使用 java.lang.Runtime.exec()。

範例：使用 java.lang.Runtime.exec() 來呼叫 CL 指令

本範例顯示如何從 Java 程式內執行控制語言 (CL) 指令。

在此範例中，Java 類別會執行 CL 指令。CL 指令使用「顯示 Java 程式 (DSPJVAPGM)」CL 指令，顯示與 Hello 類別檔案相關的程式。CL 指令的輸出位於 QSYSPRT 儲存檔。

當您將 os400.runtime.exec 系統內容設定爲 EXEC (預設值) 時，您傳入 `Runtime.getRuntime().exec()` 函數的指令將採用下列格式：

```
Runtime.getRuntime()Exec("system CLCOMMAND");
```

其中 *CLCOMMAND* 是您要執行的 CL 指令。

註: 當您將 os400.runtime.exec 設定爲 QSHELL 時，必須加上斜線及引號 (\")。例如，先前的指令會變成：

```
Runtime.getRuntime()Exec("system \"CLCOMMAND\"");
```

有關 os400.runtime.exec 的詳細資訊及其對於 java.lang.Runtime.exec() 用法的影響，請參閱以下各頁：

使用 java.lang.Runtime.exec()

Java 系統內容的清單

範例：適用於呼叫 CL 指令的類別

下列程式碼假設您使用 os400.runtime.exec 系統內容的預設值 EXEC。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallCLCom
{
   public static void main(String[] args)
   {
      try
      {
         Process theProcess =
            Runtime.getRuntime().exec("system DSPJVAPGM CLSF('/com/ibm/as400/system/Hello.class')
                                      OUTPUT(*PRINT)");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
         e.printStackTrace();
      }
   } // end main() method
} // end class
```

如需相關背景資訊，請參閱使用 java.lang.Runtime.exec()。

處理作業之間的通信

若要與另一個程序內執行的程式進行通訊，您有許多方法可以選擇。

一種方法是使用 Socket 來進行處理作業之間的通信。由一個程式扮演伺服器程式的角色，在 Socket 連線上接聽用戶端程式的輸入。用戶端程式則利用 Socket 來連接伺服器。當 Socket 連線建立之後，兩端的程式即可互相收送資訊。

另一種方法是使用串流檔在程式之間通訊。這需要用到 System.in、System.out 及 System.err 類別。

第三種方法是使用可提供資料佇列及 iSeries 訊息物件的 IBM Toolbox for Java。

您亦可從其他語言呼叫 Java。如需相關資訊，請參閱範例：從 C 呼叫 Java 及範例：從 RPG 呼叫 Java。

使用 Socket 來進行處理作業之間的通訊

Socket 串流在個別程序中所執行的程式之間進行通訊。

這些程式可以個別啓動，或從主要 Java 程式內使用 java.lang.Runtime.exec() 方法來啓動。若程式並非以 Java 語言編寫，則您必須確保完成任何「美國國家標準交換碼 (ASCII)」或「延伸的二進位編碼十進位交換碼 (EBCDIC)」轉換。如需詳細資料，請參閱 Java 字元編碼。

如需 Socket 的使用範例，請參閱範例：使用 Socket 來進行處理作業之間的通訊。

範例：使用 Socket 來進行處理作業之間的通訊:

此範例使用 Socket 在 Java 程式與 C 程式之間進行通訊。

請先啓動 C 程式，在 Socket 上接聽。當 Java 程式連接 Socket 之後，C 程式就會使用該 Socket 連線來傳送字串給 Java 程式。從 C 程式傳送的字串爲字碼頁 819 的「美國國家標準交換碼 (ASCII)」字串。

請在「Qshell 直譯器」指令行或其他 Java 平台上，使用指令 `java TalkToC xxxxx nnnn` 來啓動 Java 程式。或者，在 iSeries 指令行中輸入 `JAVA TALKTOC PARM(xxxxx nnnn)`，以啓動 Java 程式。xxxxx 指執行 C 程式所在系統的網域名稱或 Internet Protocol (IP) 位址。 nnnn 指 C 程式使用的 Socket 埠號。請使用此埠號作爲呼叫 C 程式的第一個參數。

範例：1：TalkToC 用戶端類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.net.*;
import java.io.*;

class TalkToC
{
   private String host = null;
   private int port = -999;
   private Socket socket = null;
   private BufferedReader inStream = null;


   public static void main(String[] args)
   {
      TalkToC caller = new TalkToC();
      caller.host = args[0];
      caller.port = new Integer(args[1]).intValue();
      caller.setUp();
```

```
      caller.converse();
      caller.cleanUp();

   }  // end main() method

   public void setUp()
   {
      System.out.println("TalkToC.setUp() invoked");

      try
      {
         socket = new Socket(host, port);
         inStream = new BufferedReader(new InputStreamReader(
                                     socket.getInputStream()));
      }
      catch(UnknownHostException e)
      {
         System.err.println("Cannot find host called: " + host);
         e.printStackTrace();
         System.exit(-1);
      }
      catch (IOException e) {
System.err.println("Could not establish connection for " + host);
         e.printStackTrace();
         System.exit(-1);
      }
   } // end setUp() method

   public void converse()
   {
      System.out.println("TalkToC.converse() invoked");

      if (socket != null && inStream != null)
      {
         try
         {
            System.out.println(inStream.readLine());
         }
         catch (IOException e) {
System.err.println("Conversation error with host " + host);
            e.printStackTrace();
         }

      } // end if

   }  // end converse() method

   public void cleanUp()
   {
      try
      {

         if(inStream != null)
         {
            inStream.close();
         }
         if(socket != null)
         {
            socket.close();
         }
      } // end try
      catch(IOException e)
      {
         System.err.println("Error in cleanup");
         e.printStackTrace();
         System.exit(-1);
```

```
      }
   } // end cleanUp() method

} // end TalkToC class
```

啓動 SockServ.C 時，請傳入埠號作爲參數。例如，CALL SockServ '2001'。

範例 2：SockServ.C 伺服器程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/time.h>

void main(int argc, char* argv[])
{
   int    portNum = atoi(argv[1]);
   int    server;
   int    client;
   int    address_len;
   int    sendrc;
   int    bndrc;
   char*  greeting;
   struct sockaddr_in  local_Address;
   address_len = sizeof(local_Address);

   memset(&local_Address,0x00,sizeof(local_Address));
   local_Address.sin_family = AF_INET;
   local_Address.sin_port = htons(portNum);
   local_Address.sin_addr.s_addr = htonl(INADDR_ANY);

   #pragma convert (819)
   greeting = "This is a message from the C socket server.";
   #pragma convert (0)

   /*  allocate socket    */
   if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
   {
      printf("failure on socket allocation\n");
      perror(NULL);
      exit(-1);
   }

   /* do bind   */
   if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
   {
     printf("Bind failed\n");
     perror(NULL);
     exit(-1);
   }

   /* invoke listen   */
   listen(server, 1);

   /* wait for client request */
   if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
   {
     printf("accept failed\n");
     perror(NULL);
```

```
      exit(-1);
   }


   /* send greeting to client    */
   if((sendrc = send(client, greeting, strlen(greeting),0))<0)
   {
      printf("Send failed\n");
      perror(NULL);
      exit(-1);
   }

   close(client);
   close(server);

}
```

如需詳細資訊，請參閱使用 Socket 來進行處理作業之間的通訊。

使用輸入及輸出串流來進行處理作業之間的通信

在個別程序中執行的程式，彼此之間會往返輸入及輸出串流。

java.lang.Runtime.exec() 方法可執行程式。主程式可以掌控子程序的輸入及輸出串流，亦可讀寫這些串流。若子程式並非以 Java 語言編寫，則您必須保證完成任何「美國國家標準交換碼 (ASCII)」或「延伸的二進位編碼十進位交換碼 (EBCDIC)」轉換。如需詳細資料，請參閱 Java 字元編碼。

如需輸入及輸出串流的使用範例，請參閱範例：使用輸入及輸出串流來進行處理作業之間的通信。

範例：使用輸入及輸出串流來進行處理作業之間的通信:

下列範例顯示如何從 Java 呼叫 C 程式，以及如何使用輸入及輸出串流來進行處理作業之間的通訊。

在此範例中，C 程式會在標準輸出串流上寫入字串，然後由 Java 程式讀取並顯示此字串。此範例假設已建立檔案庫 JAVSAMPLIB，且其中也已建立 CSAMP1 程式。

註: JAVSAMPLIB 並非由 IBM Developer Kit 授權程式 (LP) 編號 5722-JV1 的安裝程序所建立。您必須明確地加以建立。

範例 1：CallPgm 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallPgm.main() invoked");

      // call the CSAMP1 program
      try
      {
          theProcess = Runtime.getRuntime().exec(
                       "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
```

```
            e.printStackTrace();
         }

         // read from the called program's standard output stream
         try
         {
            inStream = new BufferedReader(new InputStreamReader
                              (theProcess.getInputStream()));
            System.out.println(inStream.readLine());
         }
         catch(IOException e)
         {
            System.err.println("Error on inStream.readLine()");
            e.printStackTrace();
         }

      } // end method

   } // end class
```

範例 2：CSAMP1 C 程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
     /* Convert the string to ASCII at compile time */
#pragma convert(819)
     printf("Program JAVSAMPLIB/CSAMP1 was invoked\n");
#pragma convert(0)
     /* Stdout may be buffered, so flush the buffer */

     fflush(stdout);

}
```

如需相關資訊，請參閱使用輸入及輸出串流來進行處理作業之間的通信。

範例：從 C 呼叫 Java

此爲從 C 程式中使用 system() 函數呼叫 Java Hello 程式的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
#include <stdlib.h>

 int main(void)
 {
   int result;

 /* The system function passes the given string to the CL command processor
    for processing. */

   result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
 }
```

範例：從 RPG 呼叫 Java

此爲從 RPG 程式中使用 QCMDEXC API 呼叫 Java Hello 程式的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
D*             DEFINE  THE PARAMETERS FOR THE QCMDEXC API
D*
DCMDSTRING        S             25    INZ('JAVA CLASS(''com.ibm.as400.system.Hello'')')
DCMDLENGTH        S             15P 5 INZ(25)
D*             NOW THE CALL TO QCMDEXC WITH THE 'JAVA' CL COMMAND
C                   CALL      'QCMDEXC'
C                   PARM                    CMDSTRING
C                   PARM                    CMDLENGTH
C*       This next line displays 'DID IT' after you exit the
C*       Java Shell via F3 or F12.
C     'DID IT'      DSPLY
C*       Set On LR to exit the RPG program
C                   SETON                                        LR
C
```

Java 平台

Java 平台是用來開發及管理 Java Applet 及應用程式的環境。它由三個主要元件組成：Java 語言、Java 套件及 Java 虛擬機器。

Java 語言及套件類似於 C++ 及其類別庫。Java 套件包含類別，可在任何符合標準的 Java 實作方式中使用。在任何支援 Java 的系統上，應用程式設計介面 (API) 應該都一樣。

Java 的編譯及執行方式不同於 C++ 這類的傳統語言。在傳統的程式設計環境中，需要針對特定的硬體及作業系統來編寫程式的原始程式碼，然後再編譯成目的碼。目的碼需要連結其他目的碼模組，才能建立可執行的程式。程式碼是針對一組特定的電腦硬體，未經變更，無法在其他系統上執行。這樣的描繪說明了傳統語言的部署環境。

Java Applet 與應用程式

Applet 是設計來在 HTML Web 文件中併入的 Java 程式。您可以將自己編寫的 Java Applet 併入 HTML 網頁內，就像併入影像一樣。使用支援 Java 功能的瀏覽器來檢視含有 Applet 的 HTML 網頁時，Applet 的程式碼就會傳送至您的系統，由瀏覽器的 Java 虛擬機器執行。

HTML 文件含有標籤，可以指定 Java Applet 的名稱及其「全球資源定位器 (URL)」。URL 是 Applet 位元組碼在網際網路上的位置。當畫面上顯示含有 Java Applet 標籤的 HTML 文件時，支援 Java 功能的 Web 瀏覽器會從網際網路下載 Java 位元組碼，然後使用 Java 虛擬機器來處理 Web 文件內的程式碼。這些 Java Applet 就是網頁能夠呈現動畫或互動式內容的功臣。

您也可以編寫不必使用 Web 瀏覽器的 Java 應用程式。

如需相關資訊，請參閱 Sun Microsystems 之 Java Applet 指導教學的 Writing Applets。內容包括 Applet 概觀、Applet 編寫指南及一些常見的 Applet 問題。

應用程式爲不必依賴瀏覽器的獨立程式。Java 應用程式的執行方式，是先從指令行啓動 Java 直譯器，再指定含有已編譯之應用程式的檔案。應用程式通常位於其部署的系統上。應用程式會存取系統的資源，但受 Java 安全模型的限制。

Java 虛擬機器

Java 虛擬機器是一種執行時間環境，可以加入 Web 瀏覽器或任何作業系統中，例如 IBM i5/OS。Java 虛擬機器可以執行 Java 編譯器所產生的指令。它由位元組碼直譯器及執行時間組成，可讓 Java 類別檔案在任何平台上執行，而無需考量最初是在哪個平台上開發此檔案。

Java Runtime 的類別載入器及安全管理程式，可以隔離來自其他平台的程式碼。亦可限制每一個已載入的類別可存取哪些系統資源。

註: Java 應用程式不受限制，僅 Applet 受限。應用程式可自由地存取系統資源及使用原生方法。大部分 IBM Developer Kit for Java 程式都是應用程式。

您可以使用「建立 Java 程式 (CRTJVAPGM)」指令，確保程式碼符合 Java Runtime 爲了驗證位元組碼所強制的安全要求。其中包括強制類型限制、檢查資料轉換、確保不會發生參數堆疊溢位或反向溢位，以及檢查存取違規。然而，您並不需要特地驗證位元組碼。若事前未使用 CRTJVAPGM 指令，則第一次使用類別時會執行檢查。在驗證位元組碼之後，直譯器就會將位元組碼解碼，然後執行機器指令來完成所需的作業。

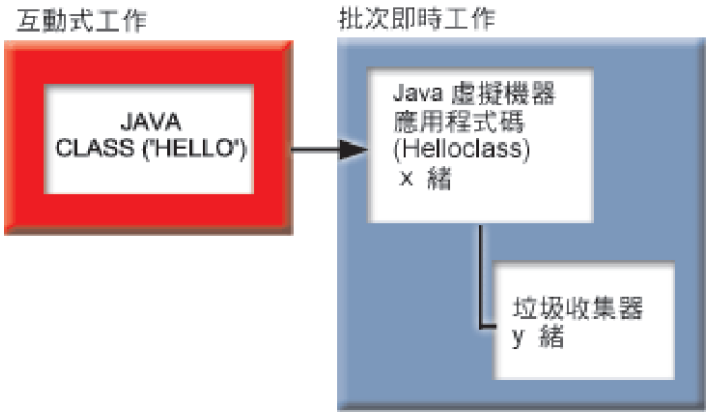
註: 唯有指定 OPTIMIZE(*INTERPRET) 或 INTERPRET(*YES) 時，才會使用第 219 頁的『Java 直譯器』。

除了載入及執行位元組碼之外，Java 虛擬機器還包括管理記憶體的垃圾收集器。載入及解譯位元組碼的同時，也會執行垃圾收集。

Java 執行時間環境

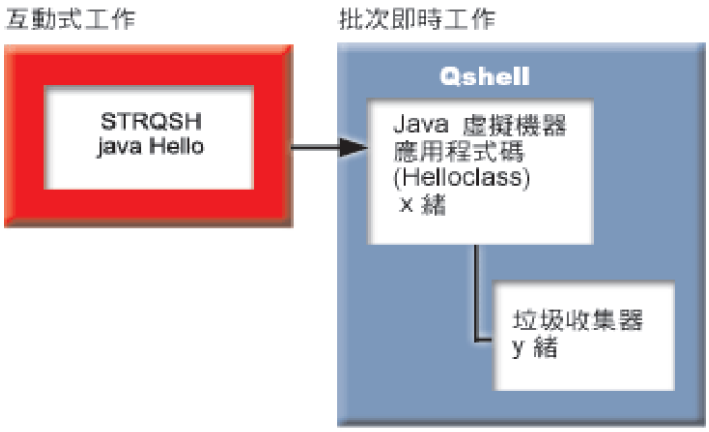
每當您在 iSeries 指令行輸入「執行 Java (RUNJVA)」指令或 JAVA 指令時，就會啓動 Java 執行時間環境。因爲 Java 環境屬於多緒型，所以必須在支援緒的工作中執行 Java 虛擬機器，例如批次即時 (BCI) 工作。如下圖所示，在 Java 虛擬機器啓動之後，執行垃圾收集器的工作中可能有額外的緒啓動。

圖 1：使用 RUNJVA 或 JAVA CL 指令時的一般 Java 環境



亦可從「Qshell 直譯器」在 Qshell 中使用 java 指令，來啓動 Java 執行時間環境。在此環境下，「Qshell 直譯器」會在互動式工作相關的 BCI 工作中執行。Java 執行時間環境會於執行「Qshell 直譯器」的工作中啓動。

圖 2：在 Qshell 中使用 java 指令時的 Java 環境



從互動式工作中啓動 Java 執行時間環境時，就會出現「Java Shell 顯示畫面」。此畫面會顯示輸入行，供您在 System.in 串流中輸入資料，也會顯示寫入 System.out 串流及 System.err 串流的資料。

Java 直譯器

Java 直譯器屬於 Java 虛擬機器的一部分，可以針對特定的硬體平台來解譯 Java 類別檔案。Java 直譯器可將每一個位元組碼解碼，然後對此位元組碼執行一連串機器指令。

相關主題：

- Java 類別檔案
-

Java JAR 及類別檔案

Java ARchive (JAR) 檔是一種檔案格式，可將許多檔案結合成一個檔案。Java 環境不同於其他程式設計環境，Java 編譯器產生的機器碼不是針對硬體特定的指令集。Java 編譯器是將 Java 原始程式碼轉換成 Java 類別檔案所儲存的 Java 虛擬機器指令。您可以使用 JAR 檔來儲存類別檔案。 類別檔案並非針對特定的硬體平台，而是針對 Java 虛擬機器架構。

您可以將 JAR 當作一般的保存工具，亦可用來配送所有類型的 Java 程式，包括 Applet。Java Applet 可在單一「超文字傳送通訊協定 (HTTP)」異動中整個下載至瀏覽器，而非每一部分都開啓一個新的連線。這種下載方式可改善 Applet 下載至網頁及開始運作的速度。

JAR 是唯一跨平台的保存格式。JAR 也是唯一可處理音效檔及影像檔的檔案格式，亦可處理類別檔案。JAR 是一種開放式標準的完全可延伸格式，以 Java 編寫。

JAR 格式亦支援壓縮，可以減少檔案大小及縮短下載時間。此外，Applet 作者亦可利用數位方式，對 JAR 檔中的個別項目加以簽署，藉此鑑別這些項目的來歷。

若要更新 JAR 檔中的類別，請參閱 Java jar 工具。

Java 類別檔案是 Java 編譯器編譯來源檔時產生的串流檔。類別檔案含有一些表格，可說明此類別的每一個欄位及方法。該檔案亦包含每一個方法的位元組碼、靜態資料，以及用來描述 Java 物件的說明。

Java 緒

緒爲在程式內執行的單一獨立串流。Java 是一種多緒的程式設計語言，所以 Java 虛擬機器內同時可以執行多個緒。Java 緒讓 Java 程式可同時執行多項作業。緒基本上就是程式裡的控制流程。

緒是一種最新的程式設計構造，不但可用來支援並行的程式，亦可提升應用程式的效能及可調整性。大部份程式設計語言皆透過外掛的程式設計庫來支援緒的設計。 Java 以內建的應用程式設計介面 (API) 支援緒。

註: 使用緒可以提高互動性，也就是說，可縮短您在鍵盤前空等的時間，因爲許多作業都可平行執行。但程式並不一定因爲緒就能提高互動性。

緒是指一種機制，可在等待長時間互動的同時，還能讓程式同時處理其他工作。緒可在相同程式碼串流上支援多個流程。有時稱爲**輕量型程序**。Java 語言可直接支援緒。但根據設計，並不支援非同步的非區塊性輸入及輸出發生岔斷或多重等待。

在機器配備多個處理器的環境下，緒可以開發出極具調整能力的平行程式。只要適當地予以建構，亦可提供模型來處理多重異動及使用者。

在許多情況下，您可以在 Java 程式中使用緒。有些程式必須能夠在參與多項活動的同時，還能夠回應使用者的其他輸入。例如，Web 瀏覽器在播放音樂時，也應該能夠同時回應使用者的輸入。

緒亦可使用非同步方法。在呼叫第二個方法時，不必等待第一個方法完成，第二個方法馬上可以繼續進行自己的活動。

不必用到緒的原因也很多。若程式採用固有的循序邏輯，則一個緒就足以完成整個序列。在此情況下，使用多個緒只會導致程式更加複雜，沒有好處。建立及啓動一個緒會引發頗多的工作量。若一項作業僅處理少許的陳述式，則利用一個緒來處理會更快。即使作業屬於非同步性質，道理也一樣。有多個緒共用物件時，則物件必須同步，以協調緒的存取情況，並維護一致性。同步化會增加程式的複雜性，難以調整來獲得最佳化效能，甚至可能成爲程式設計錯誤的根源。

如需更多關於緒的資訊，請參閱開發多緒的應用程式。

Sun Microsystems, Inc. Java Development Kit

Java Development Kit (JDK) 是 Sun Microsystems 公司發行給 Java 程式開發者的軟體。內含 Java 直譯器、Java 類別及 Java 開發工具：編譯器、除錯程式、反組譯器、appletviewer、Stub 檔產生器及文件產生器。

JDK 可讓您編寫應用程式，該應用程式一經開發，即可於任何 Java 虛擬機器上執行。在某個系統上以 JDK 開發的 Java 應用程式，不必修改或重新編譯程式碼，即可在另一個系統上使用。在任何標準的 Java 虛擬機器上，Java 類別檔案都具有可攜性。

如需現行 JDK 的相關資訊，請檢查您 iSeries 伺服器上的 IBM Developer Kit for Java 的版本。

若要在 iSeries 伺服器上檢查預設 IBM Developer Kit for Java Java 虛擬機器版本，請輸入下列任一指令：

- Qshell 指令提示：`java -version`。
- CL 指令行：`RUNJVA CLASS(*VERSION)`。

另外，如需特定的文件，請至 The Source for Java Technology java.sun.com 尋找相同的 Sun Microsystems 公司的 JDK 版本。IBM Developer Kit for Java 是 Sun Microsystems 公司 Java Technology 的相容實作方式，因此，請務必熟讀相關的 JDK 文件。

如需詳細資訊，請參閱下列主題：

- 多重 Java Development Kit (JDK) 的支援提供不同 Java 虛擬機器的使用資訊。

- 原生方法及 Java 原生介面可定義原生方法的意義及用途。本主題也簡單說明了「Java 原生介面」。

Java 套件

Java 套件可將 Java 中相關的類別及介面進行分組。Java 套件與其他語言可用的類別庫類似。

Java 套件提供了 Java API，其屬於 Sun Microsystems 公司的 Java Development Kit (JDK)。如需 Java 套件的完整清單及 Java API 的相關資訊，請參閱 Java 2 Platform Packages。

Java 工具

如需 Sun Microsystems 公司 Java Development Kit 提供之工具的完整清單，請參閱 Sun Microsystems 公司的工具參考資訊。如需 IBM Developer Kit for Java 支援之每一項工具的相關資訊，請參閱 IBM Developer Kit for Java 支援的 Java 工具。

進階主題

本主題提供如何在批次工作中執行 Java 的指示，同時說明在整合檔案系統中顯示、執行 Java 程式或進行除錯所需的 Java 檔案權限。

Java 類別、套件及目錄

每個 Java 類別分屬於某個套件。Java 來源檔的第一個陳述式即指出各個類別所屬的套件。若來源檔不含 package 陳述式，則類別屬於某個未命名的預設套件。

套件名稱與類別所在的目錄結構有關。整合檔案系統以階層式檔案結構支援 Java 類別，該結構與大部分的 PC 及 UNIX® 系統類似。Java 類別之儲存目錄的相對目錄路徑，必須符合此類別的套件名稱。例如，請考量下列 Java 類別：

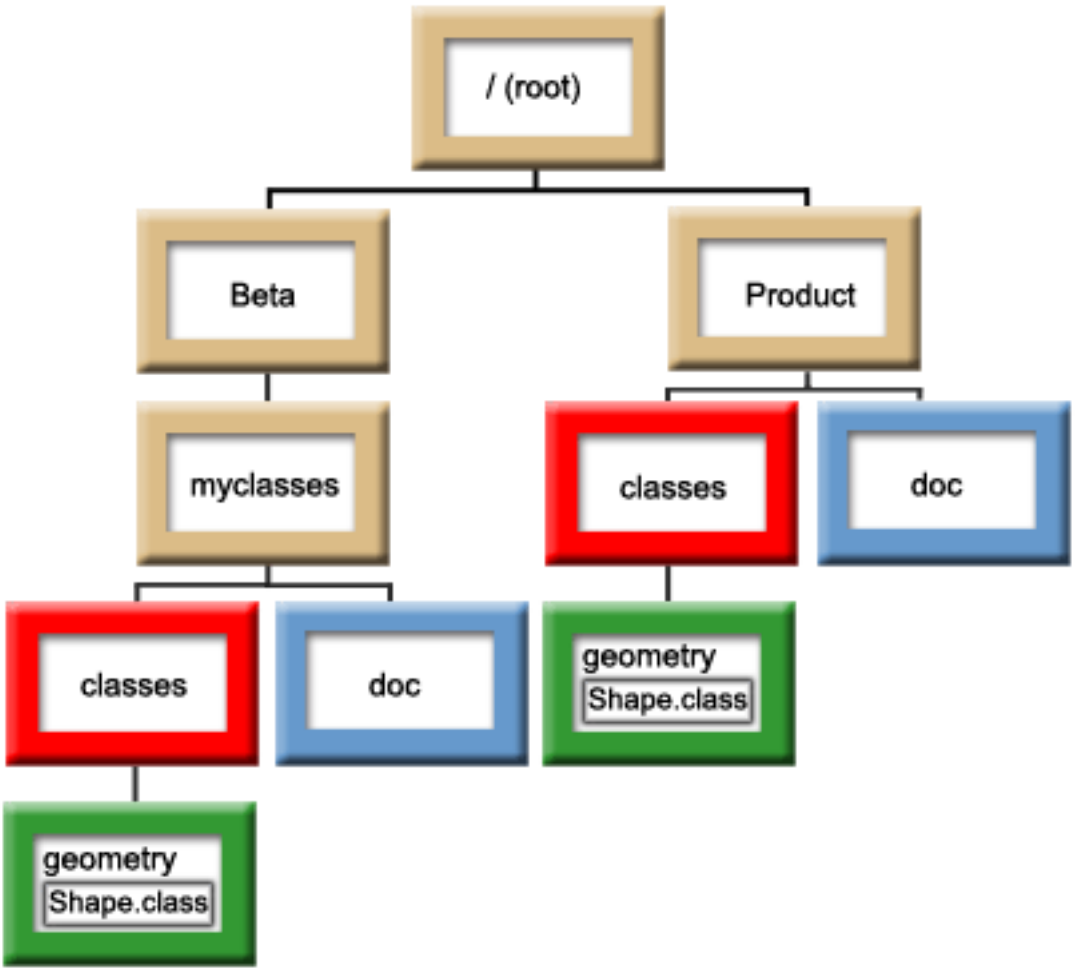
```
package classes.geometry;
     import java.awt.Dimension;

     public class Shape {

     Dimension metrics;

          // The implementation for the Shape class would be coded here ...

}
```

在上述程式碼中，package 陳述式指出 Shape 類別屬於 classes.geometry 套件。爲了讓 Java Runtime 找到 Shape 類別，Shape 類別必須儲存在相對目錄結構 classes/geometry 中。

註: 套件名稱對應於類別所在的相對目錄名稱。Java 虛擬機器的類別載入器，會在類別路徑中指定的每一個目錄後面，附加相對路徑名稱，藉以尋找類別。Java 虛擬機器的類別載入器也會搜尋您在類別路徑中指定的 ZIP 檔或 JAR 檔，藉以尋找類別。

例如，當您將 Shape 類別儲存於 root (/) 檔案系統之下的 /Product/classes/geometry 目錄時，類別路徑中就需要指定 /Product。

圖 1：相同名稱的 Java 類別在不同套件中的目錄結構範例



註: 目錄結構中可能存在 Shape 類別的多個版本。若要使用 Shape 類別的 Beta 版本，請在類別路徑中，將 /Beta/myclasses 放在其他含有 Shape 類別的任何目錄或 ZIP 檔之前。

在編譯 Java 原始程式碼時，Java 編譯器會使用 Java 類別路徑、套件名稱及目錄結構來尋找套件及類別。如需相關資訊，請參閱 Java 類別路徑。

整合檔案系統中的檔案

整合檔案系統以階層式檔案結構儲存 Java 相關的類別、原始檔、ZIP 及 JAR 檔。IBM Developer Kit for Java 支援使用整合檔案系統中的安全緒檔案系統，以儲存及處理 Java 相關的類別檔案、來源檔、ZIP 檔及 JAR 檔。

有關安全緒檔案系統的資訊及檔案系統的比較，請參閱：

多緒程式設計的檔案系統注意事項

檔案系統比較

整合檔案系統的 Java 檔案權限

若要執行 Java 程式或進行除錯，類別檔案、JAR 檔或 ZIP 檔需要具備讀取權限 (*R)。任何目錄都需要讀取及執行權限 (*RX)。

若要使用「建立 Java 程式 (CRTJVAPGM)」指令來最佳化程式，類別檔案、JAR 檔或 ZIP 檔必須具備讀取權限 (*R)，目錄也必須具備執行權限 (*X)。若在類別檔名中使用型樣，則目錄必須具備讀取及執行權限 (*RX)。

若要使用「刪除 Java 程式 (DLTJVAPGM)」指令來刪除 Java 程式，您必須具備類別檔案的讀取及寫入權限 (*RW)，且目錄也必須具備執行權限 (*X)。若在類別檔名中使用型樣，則目錄必須具備讀取及執行權限 (*RX)。

若要使用「顯示 Java 程式 (DSPJVAPGM)」指令來顯示 Java 程式，您必須具備類別檔案的讀取權限 (*R)，且目錄也必須具備執行權限 (*X)。

註: 對具備 QSECOFR 權限的使用者而言，沒有執行權限 (*X) 的檔案及目錄，一定都有執行權限 (*X)。在某些狀況下，就算使用者對於相同檔案都有相同的存取權限，不同使用者亦可能得到不同結果。在使用「Qshell 直譯器」或 java.Runtime.exec() 來執行 Shell Script 時，請注意這一點。

例如，某位使用者編寫的 Java 程式中，利用 java.Runtime.exec() 來呼叫 Shell Script，然後透過具備 QSECOFR 權限的使用者 ID 來進行測試。若 Shell Script 的檔案模式有讀取及寫入權限 (*RW)，整合檔案系統就允許有 QSECOFR 權限的使用者 ID 執行此程式。然而，缺乏 QSECOFR 權限的使用者亦可能嘗試執行相同的 Java 程式，此時，整合檔案系統會通知 java.Runtime.exec() 程式碼，表示無法執行 Shell Script，因爲缺少 *X。在此情況下，java.Runtime.exec() 會丟出輸入及輸出異常。

對於 Java 程式在整合檔案系統中建立的新檔案，您亦可指派權限。對於檔案，請使用 os400.file.create.auth 系統內容，對於目錄，請使用 os400.dir.create.auth，而且，讀取、寫入及執行權限可任意組合使用。

如需詳細資訊，請參閱程式及 CL 指令 API 或整合檔案系統。

在批次工作中執行 Java

利用「提出工作 (SBMJOB)」指令，即可透過批次工作來執行 Java 程式。在此模式下，沒有「Java Qshell 輸入指令」顯示畫面可用來處理 System.in、System.out 及 System.err 串流。

可以將這些串流重新導向到其他檔案。預設處理方式會將 System.out 及 System.err 串流傳送至排存檔。 對於來自 System.in 的讀取要求，批次工作會導致輸入及輸出異常，批次工作擁有排存檔。可以在 Java 程式內重新導向 System.in、System.out 及 System.err。亦可使用 os400.stdin、os400.stdout 及 os400.stderr 系統內容來重新導向 System.in、System.out 及 System.err。

註: SBMJOB 可以將「現行工作目錄 (CWD)」設爲使用者設定檔中指定的 HOME 目錄。

範例：在批次工作中執行 Java

```
SBMJOB CMD(JAVA Hello OPTION(*VERBOSE)) CPYENVVAR(*YES)
```

執行前一個範例的 Java 指令會衍生第二個工作。因此，執行批次工作的子系統，必須能夠執行多個工作。

請透過下列步驟來驗證批次工作是否能夠執行多個工作：

1. 在 CL 指令行，輸入 DSPSBSD(MYSBSD)，其中 MYSBSD 是批次工作的子系統說明。
2. 選擇選項 6 (工作佇列登錄)。
3. 查看工作佇列的「最多作用中」欄位。

在沒有圖形式使用者介面的主電腦上執行 Java 應用程式

若要在沒有圖形式使用者介面 (GUI) 的主電腦 (例如 iSeries 伺服器) 上執行 Java 應用程式，您可以使用 Native Abstract Windowing Toolkit (NAWT)。

使用 NAWT 可讓您的 Java 應用程式及 Servlet 具備 Java 2 Software Development Kit (J2SDK) 標準版 AWT 的完整圖形功能。

Native Abstract Windowing Toolkit

Native Abstract Windowing Toolkit (NAWT) 讓 Java 應用程式及 Servlet 能夠使用 Java 2 Software Development Kit (J2SDK) 標準版所提供的 Abstract Windowing Toolkit (AWT) 圖形功能。

註: NAWT 目前不支援特定地區及語言特定的字型及字集。使用 NAWT 時，必須遵守下列基本要求：

- 僅使用 ISO8859-1 字集中定義的字元。
- 使用 font.properties 檔案。font.properties 檔案位於 /QIBM/ProdData/Java400/jdknn/lib 目錄中， *nn* 指您正在使用的 J2SDK 版本號碼。明確地說，請勿使用 font.properties.*xxx* 檔案，其中 *xxx* 代表語言或其他限定元。

NAWT 通常以 X Window System 爲基礎的圖形引擎。 必須要有 X 伺服器，您才能使用 X Window System。 X 伺服器是一種獨立應用程式，可接受來自 X 用戶端程式的連線及要求。 在此情況下，底層的 NAWT 基礎架構爲 X 用戶端程式。

建議的 X 伺服器爲 AT®&T「虛擬網路運算 (VNC)」伺服器。VNC 伺服器非常適合 iSeries 伺服器，因爲它不需要專用的滑鼠、鍵盤及具備圖形顯示能力的監視器。IBM 提供了一個 VNC 伺服器版本，可執行於 i5/OS Portable Application Solutions Environment (i5/OS PASE)。i5/OS PASE 是一種類似 UNIX 的環境，讓您能執行大多數針對 IBM AIX 作業系統所編譯的二進位可執行檔。i5/OS PASE 屬於 i5/OS 安裝環境的一部分。

在 i5/OS PASE 中執行 VNC 伺服器時，iSeries 伺服器會執行所有 NAWT 圖形計算，因此不需要外部的圖形伺服器。下列 NAWT 及 J2SDK 資訊可說明如何取得 VNC 伺服器，並且在 i5/OS PASE 中完成設定。

如需安裝及使用 NAWT 的相關資訊，請參閱：

因爲執行 NAWT 時需要使用 i5/OS PASE 及 VNC，所以您應該多瞭解這些應用程式。如需詳細資訊，請參閱：

i5/OS PASE

Virtual Network Computing

NAWT 支援層次

您所使用之 Java 2 Software Development Kit (J2SDK) 標準版的版本，將影響 Native Abstract Windowing Toolkit (NAWT) 支援可用的選項。在安裝 NAWT 之前，請先瞭解何種支援符合您的需求。請利用這項資訊來協助您評估圖形需求及選取您需要執行的 J2SDK 版本。請利用這項資訊來協助您評估圖形需求及選取您需要執行的 J2SDK 版本。

NAWT 及 J2SDK 1.3 版

在 J2SDK 1.3 版中，NAWT 僅支援不要求直接使用者互動的圖形式 Java 應用程式。此支援層次適用於會在 iSeries 伺服器上產生影像資料 (以 JPEG、GIF 等編碼) 的 Java 應用程式、Servlet 及圖形套件。

NAWT 及 J2SDK 1.4 版及更高版本

在 J2SDK 1.4 版及後續版本中，NAWT 支援全部的 Java Abstract Windowing Toolkit (AWT) 功能，包括互動式圖形式使用者介面 (GUI) 及 Java 遠端控制 AWT 環境。

如需執行 J2SDK 1.4 版時可用 NAWT 支援的相關資訊，請參閱：

在 J2SDK 1.3 版安裝及使用 NAWT:

若要安裝 Java 2 Software Development Kit (J2SDK) 1.3 版的 Native Abstract Windowing Toolkit (NAWT)，請完成下列作業。

1. 安裝 NAWT 軟體修訂程式
2. 安裝 iSeries Tools for Developers PRPQ

您必須先爲「虛擬網路運算 (VNC)」伺服器建立密碼檔， 才能使用 NAWT 或測試 NAWT 安裝環境。 下列資訊列出其他必要及選用的步驟：

建立 VNC 密碼檔

啓動 VNC 伺服器 (通常於每一次 IPL 之後)

配置環境變數 (每次執行 Java 之前)

配置 Java 系統內容 (每次執行 Java 之前)

驗證 NAWT 安裝 (選用性)

鏈結集合

安裝 NAWT 軟體修訂程式
若要安裝 NAWT，請務必安裝軟體修訂程式，內含您所使用之 Java 2 Software Development Kit (J2SDK) 標準版的適當 NAWT 支援。

安裝 iSeries Tools for Developers PRPQ
若要執行 Native Abstract Windowing Toolkit (NAWT)，您需要安裝 iSeries Tools for Developers PRPQ (5799PTL)。若沒有此 PRPQ，則必須訂購。

建立 VNC 密碼檔
若要安裝及執行 Native Abstract Windowing Toolkit (NAWT)，必須建立「虛擬網路運算 (VNC)」伺服器密碼檔。 VNC 伺服器預設必須要有密碼檔，用以避免 VNC 顯示器受到未授權使用者的存取。 您必須在用來啓動 VNC 伺服器的設定檔下建立 VNC 密碼檔。

啓動 VNC 伺服器

配置環境變數
每次執行 Java 與 NAWT 時，都必須設定環境變數，向 Java 指出系統名稱、顯示器號碼，以及每一部 X 伺服器與相關 .Xauthority 檔案的位置。

配置 Java 系統內容

驗證 NAWT 安裝

安裝 NAWT 軟體修訂程式:

若要安裝 NAWT，請務必安裝軟體修訂程式，內含您所使用之 Java 2 Software Development Kit (J2SDK) 標準版的適當 NAWT 支援。

在安裝任何軟體修訂程式 (PTF) 之前，請確定伺服器軟體包含的授權程式 5722JV1 的選項， 是否對應於您打算使用的 J2SDK 版本。請完成下列步驟來驗證伺服器軟體的選項：

1. 在 i5/OS 指令行，鍵入「跳至授權程式 (GO LICPGM)」，然後按 **ENTER** 鍵。
2. 選取選項 10 (顯示安裝的授權程式)，驗證已安裝的授權程式 5722JV1 選項， 是否對應於您打算使用的 JDK 版本。

務必套用最新的 Java 群組軟體修訂程式，以期使用任何最近公佈的 NAWT 修訂程式。

下表列出執行 NAWT 的必要選項及軟體修訂程式需求：

J2SDK 版本	5722JV1 選項	Java 軟體修訂程式 ID	PTF 日期
1.3	5	PTF 群組 5722-JV1 SF99269	最新
1.4	6	PTF 群組 5722-JV1 SF99269	最新
1.5	7	PTF 群組 5722-JV1 SF99269	最新

有關軟體修訂程式的資訊，請參閱使用軟體修訂程式。

安裝 iSeries Tools for Developers PRPQ:

若要執行 Native Abstract Windowing Toolkit (NAWT)，您需要安裝 iSeries Tools for Developers PRPQ (5799PTL)。若沒有此 PRPQ，則必須訂購。

較新的 PRPQ 版本還包含經過前置編譯、具備 i5/OS PASE 功能的「虛擬網路運算 (VNC)」版本。舊版本則不包含 VNC。安裝 PRPQ 的方式視您使用的版本而定：

- 若爲 2002 年 6 月 14 日當天或之後才訂購的 PRPQ 版本：請依照 iSeries Virtual Innovation Center 網站上提供的安裝指示來完成這項作業。

  附註：若要安裝 PRPQ 提供的 VNC 支援，只需依照網站的安裝指示即可。不需要執行設定指示。
- 若爲 2002 年 6 月 14 日之前訂購的 PRPQ 版本，請參閱安裝舊版的 iSeries Tools for Developers PRPQ 來完成這項作業。

安裝舊版 iSeries Tools for Developers:

對於 2002 年 6 月 14 日之前訂購的 iSeries Tools for Developers PRPQ (5799PTL) 版本，PRPQ 不含經過前置編譯、具備 i5/OS PASE 功能的「虛擬網路運算 (VNC)」版本。

請利用下列指示來判斷您是否具備加強型 PRPQ，若發現 PRPQ 的版本較舊，則依此指示安裝 VNC。

判定是否具備加強型 PRPQ

若您有 PRPQ 5799-PTL，但不確定是否爲含有 VNC 的加強型版本，請檢查下列檔案是否存在：

```
/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
```

PRPQ 加強型版本包含 vncserver_java 檔案，舊版本則沒有。 如果 iSeries 伺服器上沒有 vncserver_java，請訂購並安裝 PRPQ 最新版本，或依下列指示完成 VNC 安裝。

安裝 VNC

若要在舊版的 iSeries Tools for Developers PRPQ 上安裝 VNC，請完成下列步驟。

1. 執行下列指令，在 iSeries 伺服器建立儲存檔：

```
crtlib vncsavf
     crtsavf vncsavf/vncpasswd
     crtsavf vncsavf/vnc
     crtsavf vncsavf/fonts
     crtsavf vncsavf/icewm
```

2. 從 iSeries Virtual Innovation Center 網站下載儲存檔至工作站。
3. 在工作站執行下列指令，使用 FTP 將儲存檔從工作站傳送至 iSeries 伺服器：

```
ftp youriseriesserver
     bin
     cd /qsys.lib/vncsavf.lib
     put vnc.savf
     put vncpasswd.savf
     put fonts.savf
     put icewm.savf
     quit
```

4. 在 iSeries 伺服器上執行下列指令來復置儲存檔：

```
RSTOBJ OBJ(*ALL) SAVLIB(VNCSAVF) DEV(*SAVF) SAVF(VNCSAVF/VNCPASSWD)
RST DEV('/Qsys.lib/vncsavf.lib/vnc.file') OBJ(('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc*'))
RST DEV('/Qsys.lib/vncsavf.lib/fonts.file') OBJ(('/QOpenSys/QIBM/ProdData/DeveloperTools/fonts*'))
RST DEV('/Qsys.lib/vncsavf.lib/icewm.file') OBJ(('/QOpenSys/QIBM/ProdData/DeveloperTools/icewm*'))
```

5. 繼續安裝 NAWT。

啓動「虛擬網路運算 (VNC)」伺服器:

若要啓動「虛擬網路運算 (VNC)」伺服器，請在指令行鍵入下列指令，然後按 **ENTER** 鍵：

```
CALL PGM(QSYS/QP2SHELL) PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
```

其中，*n* 代表您要使用的顯示器號碼。顯示器號碼可以是 1-99 的任何整數。

附註：啓動 VNC 伺服器時會顯示訊息，識別 iSeries 系統名稱及顯示器號碼，例如 ″New 'X'desktop is *systemname:1*″。請記住或寫下這個顯示器號碼，因爲您必須使用此值來配置環境變數。

有多部 VNC 伺服器同時執行時，每一部 VNC 伺服器都必須要有唯一的顯示器號碼。 啓動 VNC 伺服器時明確地指定顯示器值，稍後配置 DISPLAY 環境變數時就比較容易。 每次要使用 NAWT 來執行 Java 時，都必須配置環境變數。

但是，若不想指定顯示器號碼，只要在上述指令中移除 ':n' 即可，vncserver_java 程式會尋找可用的顯示器。

.Xauthority 檔案

VNC 伺服器的啓動過程中，可能會建立新的 .Xauthority 檔案， 或修改現有的 .Xauthority 檔案。VNC 伺服器授權機制會利用 .Xauthority 檔案 (包含加密的金鑰資訊)，防止其他人士的應用程式截取您的 X 伺服器要求。確保 Java 虛擬機器 (JVM) 與 VNC **REQUIRES** 之間的安全通訊，JVM 與 VNC 兩者必須都能夠存取 .Xauthority 檔案的加密金鑰資訊。

.Xauthority 檔案屬於原先啓動 VNC 的設定檔。 若要讓 JVM 與 VNC 伺服器共用 .Xauthority 檔案的存取，最簡單的方法就是以相同的使用者設定檔來執行 VNC 伺服器與 JVM。 若無法以相同的使用者設定檔來執行 VNC 伺服器及 JVM， 請配置 XAUTHORITY 環境變數來指向正確的 .Xauthority 檔案。 有關 NAWT 安全通訊的資訊，請參閱下列各頁：

- 『配置 NAWT 環境變數』
- 在 WebSphere Application Server 上使用 NAWT 的要訣

配置 NAWT 環境變數:

每次執行 Java 與 NAWT 時，都必須設定環境變數，向 Java 指出系統名稱、顯示器號碼，以及每一部 X 伺服器與相關 .Xauthority 檔案的位置。

附註：啓動「虛擬網路運算 (VNC)」伺服器時， 即會決定 .Xauthority 檔案的位置，以及系統名稱和顯示器號碼的值。您必須使用這些值，才能順利配置 NAWT 環境變數。如需相關資訊，請參閱啓動「虛擬網路運算」伺服器。

配置 DISPLAY

請在您要執行 Java 程式的階段作業中，將 DISPLAY 環境變數設爲您的系統名稱及顯示器號碼。若要配置 DISPLAY 環境變數，請在 i5/OS 指令行鍵入下列控制語言 (CL) 指令，然後按 **ENTER** 鍵：

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
```

其中，*systemname* 是 iSeries 系統的主電腦名稱或 IP 位址，*n* 是 VNC 伺服器的顯示器號碼。

配置 XAUTHORITY

另外，也會將 XAUTHORITY 環境變數設爲 /home/VNCprofile/.Xauthority，其中 *VNCprofile* 是用於啓動 VNC 伺服器的設定檔。

例如，您可在 iSeries 指令提示上鍵入下列指令：

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
     ADDENVVAR ENVVAR(XAUTHORITY) VALUE('/home/VNCprofile/.Xauthority')
```

其中：

- *systemname* 是 iSeries 系統的主電腦名稱或 IP 位址。
- *n* 是 VNC 伺服器的顯示器號碼。

附註：

- 只有在要執行 Java 虛擬機器 (JVM) 的環境中，才需設定這些環境變數。
- 若無法以相同的使用者設定檔來執行 VNC 伺服器及 Java 虛擬機器，請配置 XAUTHORITY 以指向適當的 .Xauthority 檔案。但必須確定用來執行 JVM 的其他設定檔，一定能夠存取 .Xauthority 檔案。如需相關資訊，請參閱 .Xauthority 檔案。
- XAUTHORITY 安全機制不同於 VNCPASSWD 安全機制。但爲了安全地呈現圖形，這兩種保護方法缺一不可。如需 VNCPASSWD 的相關資訊，請參閱建立 VNC 密碼檔。

配置 Java 系統內容:

爲了執行 Native Abstract Windowing Toolkit (NAWT)，務必要在執行 Java 之前先設定某些 Java 系統內容。下列每一個範例中，第一行都先針對所需的 Java 2 Software Development Kit (J2SDK) 來配置 Java，第二行接著就啓用 NAWT。

J2SDK 1.3 版

在 J2SDK 1.3 版之下執行 NAWT 時，請設定下列 Java 系統內容：

```
java.version=1.3
     os400.awt.native=true
```

J2SDK 1.4 版，完整 GUI 支援

在 J2SDK 1.4 版之下搭配完整的 GUI 支援來執行 NAWT 時，請設定下列 Java 系統內容：

```
java.version=1.4
os400.awt.native=true
```

J2SDK 1.4 版，遠端控制 AWT 支援

在 J2SDK 1.4 版之下透過遠端控制模式來執行 NAWT 時，請設定下列 Java 系統內容：

```
java.version=1.4
java.awt.headless=true
```

J2SDK 1.5 版，完整 GUI 支援

在 J2SDK 1.5 版之下搭配完整的 GUI 支援來執行 NAWT 時，請設定下列 Java 系統內容：

```
java.version=1.5
     os400.awt.native=true
```

J2SDK 1.5 版，遠端控制 AWT 支援

在 J2SDK 1.5 版之下透過遠端控制模式來執行 NAWT 時，請設定下列 Java 系統內容：

```
java.version=1.5
     java.awt.headless=true
```

如需設定 Java 系統內容的相關資訊，請參閱針對 IBM Developer Kit for Java 自訂 iSeries 伺服器。

驗證 NAWT 安裝:

完成 NAWT 安裝及設定程序之後，您可以執行 Java 測試程式來驗證 NAWT 安裝。若要從 i5/OS 指令行執行測試程式，請鍵入下列指令，然後按 ENTER 鍵：

```
JAVA CLASS(NAWTtest) CLASSPATH('/QIBM/ProdData/Java400') PROP((os400.awt.native true))
```

測試程式會建立 JPEG 編碼影像檔，儲存於下列整合檔案系統路徑：

```
/tmp/NAWTtest.jpg
```

執行測試程式之後，請檢查測試程式是否已建立此檔案，且無任何 Java 異常發生。若要顯示此影像，請以二進位模式將影像檔上傳至具備圖形能力的系統。

配置 iceWM 視窗管理程式:

若希望以互動方式使用「虛擬網路運算 (VNC)」伺服器， 請在 NAWT 安裝過程中，另外配置 iceWM 視窗管理程式 (iceWM)。例如，您或許會執行具備圖形式使用者介面 (GUI) 的 Java 應用程式。

iceWM 是 iSeries Tools For Developers PRPQ 所附的小型視窗管理程式，但功能強大。如需安裝 PRPQ 的相關資訊，請參閱下列網頁：

安裝 iSeries Tools for Developers PRPQ

對於在 VNC 伺服器的 X Window 環境內執行的視窗，在背景執行的 iceWM 可以控制其外觀和操作方式。iceWM 提供的介面及功能組，很類似許多常見的視窗管理程式。內含的 vncserver_java Script 預設動作設定爲啓動 VNC 伺服器並執行 iceWM。

完成下列步驟即可建立 iceWM 所需的幾個配置檔。如有需要，亦可停用 iceWM。

配置 iceWM

若要配置 iceWM 視窗管理程式，請在 i5/OS 指令提示上完成下列步驟。請務必以您啓動 VNC 伺服器的設定檔來執行這些步驟。

1. 鍵入下列指令，然後按 **ENTER** 鍵開始進行安裝 :

   ```
   STRPTL CLIENT(IGNORE)
   ```

   IGNORE 值的作用相當於位置保留區，可確保此指令只會啓動 NAWT 所需的 STRPTL 配置功能。
2. 鍵入下列指令，然後按 **ENTER** 鍵來登出：

   ```
   SIGNOFF
   ```

登出可以確保 STRPTL 指令的任何特定階段作業結果，不會影響您後續使用或配置 NAWT 時所執行的動作。

附註：對於用來啓動 VNC 伺服器的每一個設定檔，只要執行一次 STRPTL 指令即可。 NAWT 不需要此指令任何可用的選用引數。這些聲明將取代與 5799-PTL iSeries Tools For Developers PRPQ 相關之 STRPTL 的任何設定指示。

停用 iceWM

啓動 VNC 伺服器時，會建立或修改一個現有的 Script 檔，稱爲 xstartup_java，內含用來執行 iceWM 的指令。xstartup_java Script 檔位於下列整合檔案系統目錄中：

```
/home/VNCprofile/.vnc/
```

其中，*VNCprofile* 即爲用來啓動 VNC 伺服器的設定檔名稱。

若要完全停用 iceWM，請利用文字編輯器來註銷或移除 Script 中啓動 iceWM 的那一行。 只要在行首加上一個 # 字符號字元，即可註銷此行。

使用 VNCviewer 或 Web 瀏覽器:

若要在 iSeries 伺服器上執行具有圖形式使用者介面 (GUI) 的應用程式，您必須透過 VNCviewer 或 Web 瀏覽器連接「虛擬網路運算 (VNC)」伺服器。當然，VNCviewer 或 Web 瀏覽器一定要在能夠顯示圖形的平台上執行，例如個人電腦。

附註：您必須在以下步驟提供您的顯示器號碼及 VNC 密碼。 啓動「虛擬網路運算 (VNC)」伺服器時，會決定顯示器號碼的值。 建立 VNC 密碼檔可以設定 VNC 密碼。如需詳細資訊，請參閱以下各頁：

- 啓動「虛擬網路運算」伺服器
- 建立 VNC 密碼檔

使用 VNCviewer 來存取 VNC 伺服器

若要使用 VNCviewer 來連接 VNC 伺服器，請完成下列步驟：

1. 下載及安裝 VNCviewer 應用程式：
   - AT&T Research VNC 網站提供適用於大多數平台的 VNC 檢視器
2. 啓動已下載的 VNCviewer。在提示畫面上，輸入系統名稱及顯示器號碼，然後按一下**確定**。
3. 在密碼提示畫面上，請鍵入 VNC 密碼來取得 VNC 伺服器顯示器的存取權限。

使用 Web 瀏覽器存取 VNC 伺服器

若要使用 Web 瀏覽器來連接 VNC 伺服器，請完成下列步驟：

1. 啓動瀏覽器並存取下列 URL：

   ```
   http://systemname:58nn
   ```

   其中：
   - *systemname* 爲正在執行 VNC 伺服器的系統名稱或 IP 位址
   - *nn* 表示 2 位數的 VNC 伺服器顯示器號碼

   例如，若系統名稱是 system_one 且顯示器號碼是 2，則 URL 爲：

   ```
   http://system_one:5802
   ```

2. 順利存取 URL 之後，接著會出現提示畫面，要求輸入 VNC 伺服器密碼。 在密碼提示畫面上，請鍵入 VNC 密碼來取得 VNC 伺服器顯示器的存取權限。

在 J2SDK 1.4 版及後續版本中搭配完整的 GUI 支援來安裝及使用 NAWT:

若要在 Java 2 Software Development Kit (J2SDK) 1.4 版及更新版本中搭配完整的 GUI 支援來安裝 Native Abstract Windowing Toolkit (NAWT)，請完成下列作業。

1. 安裝 NAWT 軟體修訂程式
2. 安裝 iSeries Tools for Developers PRPQ

使用 NAWT

您必須先爲「虛擬網路運算 (VNC)」伺服器建立密碼檔， 才能使用 NAWT 或測試 NAWT 安裝環境。 下列資訊列出其他必要及選用的步驟：

建立 VNC 密碼檔

啓動 VNC 伺服器 (通常於每一次 IPL 之後)

配置環境變數 (每次執行 Java 之前)

配置 Java 系統內容 (每次執行 Java 之前)

配置 iceWM 視窗管理程式 (選用性 - 適用於互動式)

使用 VNCviewer 或 Web 瀏覽器與 GUI 應用程式進行互動

驗證 NAWT 安裝 (選用性)

在 J2SDK 1.4 版及後續版本的遠端控制 AWT 模式中安裝及使用 NAWT: 若要在 Java 2 Software Development Kit (J2SDK) 1.4 版及後續版本的遠端 AWT 模式中安裝 Native Abstract Windowing Toolkit (NAWT)，請完成下列作業：

安裝 NAWT 軟體修訂程式

使用 NAWT

下列資訊列出使用 NAWT 或測試 NAWT 安裝環境之前，必須執行的其他任何必要及選用步驟：

配置 Java 系統內容 (每次執行 Java 之前)

驗證 NAWT 安裝 (選用性)

鏈結集合

安裝 NAWT 軟體修訂程式
若要安裝 NAWT，請務必安裝軟體修訂程式，內含您所使用之 Java 2 Software Development Kit (J2SDK) 標準版的適當 NAWT 支援。

配置 Java 系統內容

驗證 NAWT 安裝

安裝及使用 Native Abstract Windowing Toolkit

透過這些逐步指示來安裝 NAWT 及 VNC。在使用 NAWT 之前，您必須完成一些必要步驟。

如需詳細資訊，請參閱 NAWT 支援層次。

安裝及使用 NAWT

在您評估對圖形的需求並決定執行哪套 J2SDK 版本之後， 請依照下列指示來安裝及使用 NAWT：

第 225 頁的『在 J2SDK 1.3 版安裝及使用 NAWT』

第 231 頁的『在 J2SDK 1.4 版及後續版本中搭配完整的 GUI 支援來安裝及使用 NAWT』

第 231 頁的『在 J2SDK 1.4 版及後續版本的遠端控制 AWT 模式中安裝及使用 NAWT』

NAWT 及 i5/OS PASE

NAWT 會自動啓動 i5/OS PASE 環境，但會以預設的 32 位元模式啓動。如需於 64 位元模式中執行 i5/OS PASE，則需在啓動 JVM 之前先設定 QIBM_JAVA_PASE_STARTUP 環境變數。如需相關資訊，請參閱 Java i5/OS PASE 環境變數。

VNC 使用秘訣

使用 i5/OS 控制語言 (CL) 指令來啓動及停止「虛擬網路運算(VNC)」伺服器，以及顯示目前執行之 VNC 伺服器的相關資訊。

從 CL 程式中啓動 VNC 顯示伺服器

以下範例爲使用控制語言 (CL) 指令來設定 DISPLAY 環境變數及自動啓動 VNC 的方法：

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
```

其中：

- *systemname* 爲執行 VNC 的 iSeries 系統主電腦名稱或 IP 位址
- *n* 爲數值，代表您要啓動的顯示器號碼

註: 此範例假設您尚未執行顯示器：*n*，但已順利建立必要的 VNC 密碼檔。如需建立密碼檔的相關資訊，請參閱建立 VNC 密碼檔。

從 CL 程式中停止 VNC 顯示伺服器

下列程式碼顯示從 CL 程式中停止 VNC 伺服器的方法：

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' '-kill' ':n')
```

其中，*n* 爲數值，代表您要終止的顯示器號碼。

檢查正在執行的 VNC 顯示伺服器

若要判斷 iSeries 系統上目前正在執行哪個 VNC 伺服器 (若有的話)，請完成下列步驟：

1. 從 i5/OS 指令行啓動 PASE Shell：

   ```
   CALL QP2TERM
   ```

2. 在 PASE Shell 提示上，使用 PASE ps 指令來列出 VNC 伺服器：

```
ps gaxuw | grep Xvnc
```

此指令的輸出結果會採用下列格式來顯示執行中的 VNC 伺服器：

```
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
```

其中：

- 第一欄爲啓動伺服器的設定檔。
- 第二欄爲伺服器的 PASE 程序 ID。
- 開頭爲 */QOpensys/* 的資訊，代表 VNC 伺服器的啓動指令 (包含引數)。顯示器號碼通常就是 Xvnc 指令的引數清單中的第一個項目。

附註：在上述範例輸出中，Xvnc 程序即爲實際 VNC 伺服器程式的名稱。 執行 vncserver_java Script 就會啓動 Xvnc，此 Script 會備妥 Xvnc 的環境及參數，接著就啓動 Xvnc。

建立 VNC 密碼檔:

若要安裝及執行 Native Abstract Windowing Toolkit (NAWT)，必須建立「虛擬網路運算 (VNC)」伺服器密碼檔。 VNC 伺服器預設必須要有密碼檔，用以避免 VNC 顯示器受到未授權使用者的存取。 您必須在用來啓動 VNC 伺服器的設定檔下建立 VNC 密碼檔。

如何建立加密的密碼，視您使用的 PRPQ 版本而定：

- 若爲 2002 年 6 月 14 日當天或之後才訂購的 PRPQ 版本，請在 iSeries 指令提示上執行下列指令：

  ```
  MKDIR DIR('/home/VNCprofile/.vnc')
       QAPTL/VNCPASSWD USEHOME(*NO) PWDFILE('/home/VNCprofile/.vnc/passwd')
  ```

  其中，*VNCprofile* 指用來啓動 VNC 伺服器的設定檔。

- 若爲 2002 年 6 月 14 日之前訂購的 PRPQ 版本，請在 iSeries 指令提示上執行下列指令：

  ```
  MKDIR DIR('/home/VNCprofile/.vnc')
       VNCSAVF/VNCPASSWD USEHOME(*NO) PWDFILE('/home/VNCprofile/.vnc/passwd')
  ```

  其中，*VNCprofile* 指用來啓動 VNC 伺服器的設定檔。

附註：

使用 NAWT 及任何 J2SDK 版本時

- 只有用來啓動 VNC 伺服器的設定檔，才需要有 VNC 密碼檔。
- 必須有密碼檔，才能順利啓動 VNC 伺服器。

使用 NAWT 及 J2SDK 1.4 版或後續版本時

- 若要透過 VNCviewer 或 Web 瀏覽器，以互動方式存取 VNC 伺服器，使用者必須使用您在此步驟指定的密碼。

在 WebSphere Application Server 上使用 NAWT 的要訣

設定 NAWT 供執行於 WebSphere Application Server 的圖形式 Java 程式使用。在使用 WebSphere Application Server 及 NAWT 時，您必須讓「虛擬網路運算 (VNC)」伺服器與 WebSphere Application Server 之間的通訊安全地運作。

在閱讀下列資訊之前，請先確定您已瞭解如何在 iSeries 伺服器上安裝及使用 Native Abstract Windowing Toolkit (NAWT)。更明確地說，在您所使用的 Java 2 Software Development Kit (J2SDK) 版本及 i5/OS 版次中，您必須知道如何使用 NAWT。

確保安全通訊

有一種稱爲 X 權限檢查的方法，可以確保 WebSphere Application Server 與 VNC 伺服器之間的安全通訊。

VNC 伺服器的啓動過程會建立一個 .Xauthority 檔案，內含加密的金鑰資訊。 若要確保 WebSphere Application Server 與 VNC 之間的安全通訊，WebSphere Application Server 與 VNC 兩者都**必須**能夠存取 .Xauthority 檔案的加密金鑰資訊。

使用 X 權限檢查

請透過下列其中一個方法來使用 X 權限檢查：

使用相同設定檔來執行 WebSphere Application Server 與 VNC

若要確保 WebSphere Application Server 與 VNC 伺服器之間的安全通訊，方法之一是以啓動 VNC 伺服器的同一個設定檔來執行 WebSphere Application Server。若要使用相同設定檔來執行 WebSphere Application Server 與 VNC，您必須變更用來執行應用程式伺服器的使用者設定檔。

若要將應用程式伺服器的使用者設定檔，從預設使用者 (QEJBSVR) 切換成不同的設定檔， 必須執行下列動作：

1. 使用 WebSphere Application Server 管理主控台來變更應用程式伺服器配置
2. 使用「iSeries 領航員」來啓用新的設定檔

如需使用 WebSphere Application Server 管理主控台及「iSeries 領航員」的相關資訊，請參閱下列文件：

WebSphere Application Server

使用「管理中心」來管理使用者與群組

使用不同設定檔來執行 WebSphere Application Server 及 VNC

讓 WebSphere Application Server 及 VNC 分別使用不同的設定檔時，您可以指示 WebSphere Application Server 去使用 .Xauthority 檔案來確保安全通訊。

若要讓 WebSphere Application Server 使用 .Xauthority 檔案，請完成下列步驟：

1. 透過您的使用者設定檔來啓動 VNC 伺服器，建立新的 .Xauthority 檔案 (或更新現有的 .Xauthority 檔案)。從 i5/OS 控制語言 (CL) 指令行，鍵入下列指令，然後按 **ENTER** 鍵：

   ```
   CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
   ```

   其中，*n* 是顯示器號碼 (1-99 之間的數值)。

   附註：.Xauthority 檔案位於您用來執行 VNC 伺服器的設定檔所在的目錄中。
2. 請使用下列 CL 指令，將 .Xauthority 檔案的讀取權限授予用來執行 WebSphere Application Server 的設定檔：

   ```
   CHGAUT OBJ('/home') USER(WASprofile) DTAAUT(*RX)
       CHGAUT OBJ('/home/VNCprofile') USER(WASprofile) DTAAUT(*RX)
       CHGAUT OBJ('/home/VNCprofile/.Xauthority') USER(WASprofile) DTAAUT(*R)
   ```

其中，*VNCprofile* 及 *WASprofile* 爲您用來執行 VNC 伺服器及 WebSphere Application Server 的適當設定檔。

註: 僅當 *VNCprofile* 及 *WASprofile* 爲不同的設定檔時，才可遵循這些步驟。若在 *VNCprofile* 及 *WASprofile* 爲相同的設定檔時遵循這些步驟，會使 VNC 不正常運作。

3. 從 WebSphere Application Server 管理主控台，爲您的應用程式定義 DISPLAY 及 XAUTHORITY 環境變數：
   - 針對 DISPLAY，請使用：`system:n` 或 `localhost:n`

     其中，*system* 是 iSeries 系統的名稱或 IP 位址，*n* 是您用來啓動 VNC 伺服器的顯示器號碼。
   - 針對 XAUTHORITY，請使用：`/home/VNCprofile/.Xauthority`

     其中，*VNCprofile* 指用來啓動 VNC 伺服器的設定檔。
4. 重新啓動 WebSphere Application Server，讓配置變更生效。

如需使用 WebSphere Application Server 管理主控台的相關資訊，請參閱下列文件：

WebSphere Application Server

Java 安全性

本主題提供採用權限的詳細資料，並且解釋如何在 Java 應用程式中使用 SSL 以加強 Socket 串流的安全性。

Java 應用程式所受的安全限制，與 iSeries 伺服器上其他任何程式所受的安全限制都相同。若要在 iSeries 伺服器上執行 Java 程式，您必須有權存取整合檔案系統中的類別檔案。程式啓動時，就在使用者的權限範圍內執行。

透過採用權限，您可以運用程式執行者的權限及程式擁有者的權限來存取物件。對於使用者原本就無權存取的物件，採用權限可以暫時提供權限。如需 USRPRF 及 USEADPAUT 這兩個新採用權限參數的詳細資料，請參閱建立 Java 程式 (CRTJVAPGM) 指令資訊。

iSeries 伺服器上執行的 Java 程式，絕大多數爲應用程式，而非 Applet，所以不受「沙盤推演」安全模型的限制。

註: 針對 J2SDK 1.4 版及後續版次，JAAS、JCE、JGSS 及 JSSE 已納入成爲基本 JDK 的一部分，不再是延伸套件。至於舊版 JDK，這些安全項目仍屬於延伸套件。

Java 安全模型

您可以下載來自任何系統的 Java Applet，所以 Java 虛擬機器內存在安全機制可以阻止惡意 Applet。當 Java 虛擬機器載入位元組碼時，必須先通過 Java 執行時間系統的驗證。如此可確保位元組碼一定有效，且程式碼未違反 Java 虛擬機器加諸於 Java Applet 的任何限制。

就像檢查 Applet 一樣，位元組碼載入器及驗證器也會檢查位元組碼是否有效，以及資料類型的用法是否適當。還會檢查是否正確存取暫存器及記憶體，以及堆疊是否發生溢位或下溢的情形。這些檢查動作可以確保 Java 虛擬機器安全地執行類別，不會破壞系統的完整。

Java Applet 所受的限制，包括可執行的作業、存取記憶體的方式及使用 Java 虛擬機器的方式。這些限制的主要目的是防止 Java Applet 存取基礎作業系統或系統的資料。這堪稱爲一種「沙盤推演」安全模型，因爲 Java Applet 只能在自己的沙盤推演範圍內「玩」。

「沙盤推演」安全模型由類別載入器、類別檔案驗證器及 java.lang.SecurityManager 類別共同組合而成。

有關安全性的資訊，請參閱 Sun Microsystems, Inc. 的安全性文件及使用 SSL 來保護應用程式。

Java Cryptography Extension

Java Cryptography Extension (JCE) 1.2 是 Java 2 Software Development Kit (J2SDK) 標準版的標準延伸套件。JCE 在 iSeries 伺服器上的實作方式與 Sun Microsystems 公司的實作方式相容。本文僅針對 iSeries 實作方式進行說明。

爲了瞭解此資訊，您應熟悉 JCE 延伸套件的一般文件。如需 JCE 延伸套件的相關資訊，請參閱 Sun JCE 文件。

| 「IBM JCE 提供者」支援下列演算法：

表 3. JDK 1.3 及 JDK 1.4.2 支援的演算法

JDK 版本	簽章演算法	密碼演算法
1.3	SHA1withDSA SHA1withRSA MD5withRSA MD2withRSA	Blowfish AES DES Triple DES PBEWithMD2AndDES PBEWithMD2AndTripleDES PBEWithMD2AndRC2 PBEWithMD5AndDES PBEWithMD5AndTripleDES PBEWithMD5AndRC2 PBEWithSHA1AndDES PBEWithSHA1AndTripleDES PBEWithSHA1AndRC2 PBEWithSHAAnd40BitRC2 PBEWithSHAAnd128BitRC2 PBEWithSHAAnd40BitRC4 PBEWithSHAAnd128BitRC4 PBEWithSHAAnd2KeyTripleDES PBEWithSHAAnd3KeyTripleDES Mars RC2 RC4 RSA Seal

表 3. JDK 1.3 及 JDK 1.4.2 支援的演算法 (繼續)

JDK 版本	簽章演算法	密碼演算法
1.4.2	SHA1withDSA SHA1withRSA MD5withRSA MD2withRSA	Blowfish AES DES Triple DES PBEWithMD2AndDES PBEWithMD2AndTripleDES PBEWithMD2AndRC2 PBEWithMD5AndDES PBEWithMD5AndTripleDES PBEWithMD5AndRC2 PBEWithSHA1AndDES PBEWithSHA1AndTripleDES PBEWithSHA1AndRC2 PBEWithSHAAnd40BitRC2 PBEWithSHAAnd128BitRC2 PBEWithSHAAnd40BitRC4 PBEWithSHAAnd128BitRC4 PBEWithSHAAnd2KeyTripleDES PBEWithSHAAnd3KeyTripleDES Mars RC2 RC4 RSA Seal

表 4. JDK 1.3 及 JDK 1.4.2 支援的演算法 (繼續)

JDK 版本	訊息鑑別碼 (MAC)	訊息摘要	金鑰協定演算法
1.3	HmacSHA1 HmacMD2 HmacMD5	MD2 MD5 SHA-1	DiffieHellman
1.4.2	HmacSHA1 HmacMD2 HmacMD5	MD2 MD5 SHA-1 SHA-256 SHA-384 SHA-512	DiffieHellman

此外，「IBM JCE 提供者」還提供亂數產生器。

若要搭配 Java 1.3 使用 IBM JCE，請編輯 /QIBM/ProdData/OS400/Java400/jdk/lib/security/java.security 檔案。以下顯示檔案中需要變更的區段。

```
#
# To use the IBMJCE security provider, you need to:
# 1) Install an IBM Cryptographic Access Provider Product
# 2) Uncomment the third provider entry that follows.

#
# List of providers and their preference orders:
```

```
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsajca.Provider
#security.provider.3=com.ibm.crypto.provider.IBMJCE
```

此外，還有 IBMJCEFIPS JCE 提供者。已對此提供者進行驗證，其符合「聯邦資訊存取安全標準 (FIPS) 140-2」的「加密模組的安全需求」。

IBMJCEFIPS JCE 提供者支援下列演算法：

表 5. IBMJCEFIPS JCE 提供者支援的演算法

簽章演算法	密碼演算法	訊息鑑別碼	訊息摘要
SHA1withDSA	AES	HmacSHA1	MD5
SHA1withRSA	TripleDES		SHA-1
	RSA		SHA-256
			SHA-384
			SHA-512

IBMJCEFIPS JCE 提供者還支援 IBMSecureRandom 演算法以產生亂數。

若要使用 IBMJCEFIPS，需要藉由發出下列指令，將符號鏈結新增至延伸套件目錄：

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjcefips.jar')
NEWLNK(< 您的延伸套件目錄 >)
```

您還要藉由在 java.security 檔案中新增登錄 (例如，security.provider.4=com.ibm.crypto.fips.provider.IBMJCEFIPS)，或藉由使用 Security.addProvider() 方法，將提供者新增至提供者清單。

Java Secure Socket Extension

Java Secure Socket Extension (JSSE) 是 Secure Sockets Layer (SSL) 通訊協定的 Java 實作方式。JSSE 採用 SSL 及 Transport Layer Security (TLS) 通訊協定，讓用戶端及伺服器得以透過 TCP/IP 來進行安全通訊。

JSSE 提供下列功能：

- 資料加密
- 鑑別遠端使用者 ID
- 鑑別遠端系統名稱
- 執行主從架構鑑別
- 確保訊息完整性

JSSE 整合於 Java 2 Software Development Kit 標準版 (J2SDK) 1.4 版及後續版次中之後，功能更勝於單獨採用 SSL。

註: 此資訊係針對目前內附於 J2SDK 1.4 版及後續版次的 JSSE 版本而論。至於舊版的 JSSE，請參閱 Sun Java 網站的 Java Secure Socket Extension。

使用 SSL (JSSE 1.0.8 版)

SSL 具備鑑別伺服器及用戶端的能力，可以確保私密性及資料整合性。 所有 SSL 通訊皆從伺服器與用戶端之間的「訊息交換」開始。 在訊息交換期間，SSL 會協議用戶端與伺服器彼此通訊所用的密碼套件。 此密碼套件即爲 SSL 所提供的各種安全功能的一項組合。僅 J2SDK 1.3 版才能使用 SSL。Java Secure Socket Extension (JSSE 1.0.8 版) 爲 Secure Sockets Layer (SSL) 的 Java 實作方式，可以加強 Java 應用程式的安全性。

SSL 改善應用程式安全性的作法如下：

- 透過加密來保護通訊資料。
- 鑑別遠端使用者 ID。
- 鑑別遠端系統名稱。

註: SSL 採用數位憑證對 Java 應用程式的 Socket 通訊進行加密。數位憑證係識別安全系統、使用者及應用程式的一項網際網路標準。您可以使用 IBM Digital Certificate Manager 來控制數位憑證。如需相關資訊，請參閱 IBM Digital Certificate Manager。

若要利用 SSL 加強 Java 應用程式的安全性，請：

- 準備 iSeries 伺服器以支援 SSL。
- 設計 Java 應用程式爲採用 SSL，方法如下：
  - 若尚未使用 Socket Factory，請變更 Java Socket 程式碼來使用 Socket Factory。
  - 變更 Java 程式碼以使用 SSL。
- 利用數位憑證來加強 Java 應用程式的安全性，方法如下：
  1. 選取要使用的數位憑證類型。
  2. 執行應用程式時使用數位憑證。

您也可以使用 QsyRegisterAppForCertUse API，將 Java 應用程式登記爲安全應用程式。如需詳細資訊，請參閱 QsyRegisterAppForCertUse。

如需 SSL Java 版的相關資訊，請參閱 Java Secure Socket Extension

準備 iSeries 伺服器以使用 Secure Sockets Layer 支援:

若要準備在系統上使用 Secure Sockets Layer (SSL)，您必須安裝「授權程式」。數位憑證管理程式 LP：

您需要安裝「數位憑證管理程式 LP」：

- 5722-SS1 i5/OS - Digital Certificate Manager

您還必須確定可以在系統上存取或建立數位憑證。如需 iSeries 數位憑證管理與網際網路的相關資訊，請參閱

IBM Digital Certificate Manager 入門。

.

變更 Java 程式碼以使用 Socket Factory:

若要在現有的程式碼中使用 Secure Sockets Layer (SSL)，首先必須變更程式碼來使用 Socket Factory。

若要變更程式碼來使用 Socket Factory，請執行下列步驟：

1. 在程式中新增此行，以匯入 SocketFactory 類別：

   ```
   import javax.net.*;
   ```

2. 新增一行來宣告 SocketFactory 物件的實例。例如：

   ```
   SocketFactory socketFactory
   ```

3. 起始設定 SocketFactory 實例，設定爲等於 SocketFactory.getDefault() 方法。例如：

   ```
   socketFactory = SocketFactory.getDefault();
   ```

   SocketFactory 的完整宣告如下：

```
SocketFactory socketFactory = SocketFactory.getDefault();
```

4. 起始設定現有的 Socket。針對您要宣告的每一個 Socket，在 Socket Factory 上呼叫 SocketFactory 方法 createSocket(host,port)。

   Socket 宣告的方式如下：

   ```
   Socket s = socketFactory.createSocket(host,port);
   ```

   其中：

   - *s* 爲要建立的 Socket。
   - *socketFactory* 爲步驟 2 建立的 SocketFactory。
   - *host* 爲代表主伺服器名稱的字串變數。
   - *port* 爲代表 Socket 連線埠號的整數變數。

完成上述所有步驟之後，程式碼就會採用 Socket Factory。程式碼不必再做其他修改。您呼叫的所有方法及 Socket 的所有語法仍然有效。

如需轉換用戶端程式以使用 Socket Factory 的範例，請參閱範例：變更 Java 程式碼來使用伺服器 Socket Factory。

如需轉換用戶端程式以使用 Socket Factory 的範例，請參閱範例：變更 Java 程式碼來使用用戶端 Socket Factory。

範例：變更 Java 程式碼來使用伺服器 Socket Factory:

以下範例將示範如何變更簡單的 Socket 類別 simpleSocketServer，使它改用 Socket Factory 來建立所有 Socket。第一個範例示範沒有 Socket Factory 的 simpleSocketServer 類別。第二個範例示範含有 Socket Factory 的 simpleSocketServer 類別。 在第二個範例中，simpleSocketServer 會重新命名爲 factorySocketServer。

範例 1：無 Socket Factory 的 Socket 伺服器程式

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* File simpleSocketServer.java*/

import java.net.*;
import java.io.*;

public class simpleSocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    ServerSocket    serverSocket =
      new ServerSocket(serverPort);

    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
```

```
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // This server just echoes back what you send it...

    byte buffer[] = new byte[4096];

    int bytesRead;

    // read until "eof" returned
    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead); // write it back
      os.flush();    // flush the output buffer
    }


    s.close();
    serverSocket.close();
  }       // end main()

}       // end class definition
```

範例 2：有 Socket Factory 的簡單 Socket 伺服器程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/* File factorySocketServer.java  */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Change the original simpleSocketServer to use a
    // ServerSocketFactory to create server sockets.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Now have the factory create the server socket.  This is the last
    // change from the original program.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // This server just echoes back what you send it...

    byte buffer[] = new byte[4096];

    int bytesRead;
```

```
    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }


    s.close();
    serverSocket.close();
  }

}
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Socket Factory。

範例：變更 Java 程式碼來使用用戶端 Socket Factory:

以下範例將示範如何變更簡單的 Socket 類別 simpleSocketClient，使它改用 Socket Factory 來建立所有 Socket。第一個範例示範沒有 Socket Factory 的 simpleSocketClient 類別。第二個範例示範含有 Socket Factory 的 simpleSocketClient 類別。 在第二個範例中，simpleSocketClient 會重新命名爲 factorySocketClient。

範例 1：無 Socket Factory 的 Socket 用戶端程式

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* Simple Socket Client Program */

import java.net.*;
import java.io.*;

public class simpleSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
            System.out.println("java simpleSocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Create the socket and connect to the server.
    Socket  s = new Socket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.
```

範例 2：有 Socket Factory 的簡單 Socket 用戶端程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/* Simple Socket Factory Client Program */

// Notice that javax.net.* is imported to pick up the SocketFactory class.
import javax.net.*;
import java.net.*;
import java.io.*;
```

```
public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Change the original simpleSocketClient program to create a
    // SocketFactory and then use the socket factory to create sockets.

    SocketFactory socketFactory = SocketFactory.getDefault();

    // Now the factory creates the socket.  This is the last change
    // to the original simpleSocketClient program.

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Socket Factory。

變更 Java 程式碼以使用 Secure Sockets Layer:

若程式碼本來就使用 Socket Factory 來建立 Socket，則程式內可以加入 Secure Sockets Layer (SSL) 支援。 若程式碼並未使用 Socket Factory，請參閱變更 Java 程式碼來使用 Socket Factory。

若要變更程式碼來使用 SSL，請執行下列步驟：

1. 匯入 javax.net.ssl.* 來新增 SSL 支援：

   ```
   import javax.net.ssl.*;
   ```

2. 使用 SSLSocketFactory 宣告 SocketFactory，以起始設定：

   ```
   SocketFactory newSF = SSLSocketFactory.getDefault();
   ```

3. 依照您使用舊 SocketFactory 的相同方式，使用新的 SocketFactory 來起始設定 Socket：

   ```
   Socket s = newSF.createSocket(args[0], serverPort);
   ```

程式碼原本就採用 SSL 支援。程式碼不必再做其他修改。

如需程式碼範例，請參閱範例：變更 Java 用戶端來使用 Secure Sockets Layer 及範例：變更 Java 伺服器來使用 Secure Sockets Layer。

範例：變更 Java 伺服器來使用 Secure Sockets Layer:

以下範例將示範如何變更 factorySocketServer 類別來使用 Secure Sockets Layer (SSL)。

第一個範例示範未使用 SSL 的 factorySocketServer 類別。第二個範例示範使用 SSL 的同一個類別 (重新命名爲 factorySSLSocketServer)。

範例 1：無 SSL 支援的簡單 factorySocketServer 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* File factorySocketServer.java  */
// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Change the original simpleSocketServer to use a
    // ServerSocketFactory to create server sockets.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Now have the factory create the server socket.  This is the last
    // change from the original program.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

範例 2：使用 SSL 支援的簡單 factorySocketServer 類別

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/* File factorySocketServer.java  */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
```

```
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Change the original simpleSocketServer to use a
    // ServerSocketFactory to create server sockets.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Now have the factory create the server socket.  This is the last
    // change from the original program.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Secure Sockets Layer。

範例：變更 Java 用戶端來使用 Secure Sockets Layer:

以下範例將示範如何變更 factorySocketClient 類別來使用 Secure Sockets Layer (SSL)。第一個範例示範未使用 SSL 的 factorySocketClient 類別。第二個範例示範使用 SSL 的同一個類別 (重新命名爲 factorySSLSocketClient)。

範例 1：無 SSL 支援的簡單 factorySocketClient 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* Simple Socket Factory Client Program */

import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;
```

```
    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    SocketFactory socketFactory = SocketFactory.getDefault();

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.
```

範例 2：使用 SSL 支援的簡單 factorySocketClient 類別

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Notice that we import javax.net.ssl.* to pick up SSL support
import javax.net.ssl.*;
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySSLSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySSLSocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Change this to create an SSLSocketFactory instead of a SocketFactory.
    SocketFactory socketFactory = SSLSocketFactory.getDefault();

    // We do not need to change anything else.
    // That's the beauty of using factories!
    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Secure Sockets Layer。

選取要使用的數位憑證:

在您決定要使用哪個數位憑證時，必須考慮幾個因素。 您可以使用系統的預設憑證，或指定另一個憑證來使用。

下列情況適合使用系統的預設憑證：

- 您對 Java 應用程式沒有特定的安全需求。
- 您不知道 Java 應用程式需要何種安全性。
- 系統的預設憑證符合 Java 應用程式的安全需求。

註: 若決定使用系統的預設憑證，請洽詢系統管理者確認預設系統憑證已建立。如需數位憑證管理的相關資訊，請參閱 IBM Digital Certificate Manager 入門。

若不使用系統的預設憑證，則需選擇其他憑證來使用。您有兩種憑證可以選擇：

- **使用者憑證**，可識別應用程式的使用者。
- **系統憑證**，可識別執行應用程式的系統。

下列情況適合採用使用者憑證：

- 應用程式以用戶端應用程式的形式執行。
- 您希望用憑證來識別應用程式的使用者。

下列情況適合使用系統憑證：

- 應用程式以伺服器應用程式的形式執行。
- 您希望用憑證來識別執行應用程式的系統。

一旦瞭解所需的憑證種類之後，即可從您能夠存取的任何憑證儲存區中選擇任何數位憑證。

執行 Java 應用程式時使用數位憑證:

若要使用 Secure Sockets Layer (SSL)，您必須使用數位憑證來執行 Java 應用程式。

請使用下列內容來指定要採用哪個數位憑證：

- os400.certificateContainer
- os400.certificateLabel

比方說，如果您要使用數位憑證 MYCERTIFICATE 來執行 Java 應用程式 MyClass.class，且 MYCERTIFICATE 位於數位憑證儲存區 YOURDCC 內，則 `java` 指令如下：

```
java -Dos400.certificateContainer=YOURDCC
          -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

若尚未決定使用哪個數位憑證，請參閱選取要使用的數位憑證。您也可以使用系統的預設憑證，此憑證儲存於系統的預設憑證儲存區內。

若要使用系統的預設數位憑證，則完全不必指定憑證或憑證儲存區。Java 應用程式會自動使用系統的預設數位憑證。

如需 iSeries 數位憑證管理與網際網路的相關資訊，請參閱 IBM Digital Certificate Manager 入門。

數位憑證及 -os400.certificateLabel 內容

數位憑證係識別安全系統、使用者及應用程式的一項網際網路標準。數位憑證儲存於數位憑證儲存區內。若要使用數位憑證儲存區的預設憑證，您不必指定憑證標籤。若要使用特定的數位憑證，則必須在 `java` 指令中使用下列內容來指定憑證標籤：

```
os400.certificateLabel=
```

比方說，假設您要使用的憑證名稱爲 MYCERTIFICATE，則應輸入 java 指令如下：

```
java -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

在此範例中，Java 應用程式 MyClass 會使用憑證 MYCERTIFICATE。MYCERTIFICATE 必須位於系統的預設憑證儲存區內，才可供 MyClass 使用。

數位憑證儲存區及 -os400.certificateContainer 內容

數位憑證儲存區負責儲存數位憑證。若要使用 iSeries 系統預設的憑證儲存區，您不必指定憑證儲存區。若要使用特定的數位憑證儲存區，則必須在 java 指令中使用下列內容來指定該數位憑證儲存區：

```
os400.certificateContainer=
```

比方說，假設您要使用的數位憑證位於名爲 MYDCC 的憑證儲存區內，則應輸入 java 指令如下：

```
java -Dos400.certificateContainer=MYDCC MyClass
```

在此範例中，Java 應用程式 MyClass.class 會使用數位憑證儲存區 MYDCC 內的預設數位憑證，在系統上執行。應用程式中建立的任何 Socket，皆使用 MYDCC 中的預設憑證來識別自己的身份，讓所有通訊得到更安全的保護。

若要使用數位憑證儲存區內的數位憑證 MYCERTIFICATE，則應輸入 java 指令如下：

```
java -Dos400.certificateContainer=MYDCC
          -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

使用 Java Secure Socket Extension

JSSE 就像是 SSL 與 TLS 這兩種基礎機制融合而成的一種組織架構。 JSSE 可精簡基礎通訊協定的複雜性及獨特性，讓程式設計師有更安全、加密的通訊， 同時將安全性遭受破壞的可能性降至最低。只有在執行 J2SDK 1.4 版及後續版次的 iSeries 伺服器上使用 JSSE 時，以上資訊才適用。Java Secure Socket Extension (JSSE) 使用 Secure Sockets Layer (SSL) 及 Transport Layer Security (TLS) 這兩種通訊協定，在用戶端與伺服器之間提供安全、加密的通訊。

IBM 的 JSSE 實作方式稱爲 IBM JSSE。IBM JSSE 包括原有的 iSeries JSSE 提供者及 Pure Java JSSE 提供者。

配置 iSeries 伺服器來支援 JSSE:

配置 iSeries 伺服器來使用 IBM JSSE。本主題包括軟體需求、如何變更 JSSE 提供者，以及必要的安全內容及系統內容。

在 iSeries 伺服器上使用 Java 2 Software Development Kit (J2SDK) 1.4 版或後續版本時，JSSE 已完成配置。預設配置會使用原有的 iSeries JSSE 提供者。

變更 JSSE 提供者

您可配置 JSSE 來使用 Pure Java JSSE 提供者，而非原有的 iSeries JSSE 提供者。只要變更一些特定的 JSSE 安全內容及 Java 系統內容，即可在這兩個提供者之間切換運用。若需其餘相關資訊，請參閱下列主題：

- JSSE 提供者
- JSSE 安全內容
- Java 系統內容

安全管理程式

若在啓用 Java 安全管理程式的情況下執行 JSSE 應用程式，就可能需要設定可用的網路許可權。如需相關資訊，請參閱 Permissions in the Java 2 SDK 的 SSL Permission。

JSSE 提供者:

IBM JSSE 包括原有的 iSeries JSSE 提供者及兩個 Pure Java JSSE 提供者。您應根據應用程式的需求來選擇適用的提供者。

這三個提供者全部符合 JSSE 介面規格。它們彼此可以相互通訊，也可與任何其他 SSL、TLS (甚至是非 Java) 實作方式通訊。

Pure Java JSSE 提供者

Pure Java JSSE 提供者具備下列功能：

- 可使用任何類型的 KeyStore 物件來控制及配置數位憑證 (例如，JKS、PKCS12 等)。
- 可讓您以任何組合方式來同時使用多種實作方式的 JSSE 元件。

IBMJSSE 爲 Pure Java 實作方式的提供者名稱。必須以正確大小寫，將此提供者名稱傳給 java.security.Security.getProvider() 方法或數個 JSSE 類別的各種 getInstance() 方法。

Pure Java JSSE FIPS 140-2 提供者

Pure Java JSSE FIPS 140-2 提供者具備下列功能：

- 以「聯邦資訊存取安全標準 (FIPS) 140-2」來編譯「加密模組」。
- 使用任何類型的 KeyStore 物件來控制及配置數位憑證。

註: Pure Java JSSE FIPS 140-2 提供者不允許自己的實作方式內嵌入任何其他實作方式的元件。

IBMJSSEFIPS 爲 Pure Java JSSE FIPS 140-2 實作方式的提供者名稱。必須以正確的大小寫，將此提供者名稱傳給 java.security.Security.getProvider() 方法或數個 JSSE 類別的各種 getInstance() 方法。

原有的 iSeries JSSE 提供者

原有的 iSeries JSSE 提供者具備下列功能：

- 使用原有的 iSeries SSL 支援。
- 允許使用「數位憑證管理程式」來配置及控制數位憑證。這必須透過 iSeries 專屬的 KeyStore 類型 (IbmISeriesKeyStore) 來達成。
- 提供最佳效能。
- 可讓您以任何組合方式來同時使用多種實作方式的 JSSE 元件。 然而，爲了達到最佳效能，應僅使用 JSSE 原有的 iSeries 元件。

IbmISeriesSslProvider 爲原有的 iSeries 實作方式名稱。必須以正確大小寫，將此提供者名稱傳給 java.security.Security.getProvider() 方法或數個 JSSE 類別的各種 getInstance() 方法。

變更預設的 JSSE 提供者

只要適當地修改安全內容，即可變更預設的 JSSE 提供者。 如需詳細資訊，請參閱下列主題：

- JSSE 安全內容

變更 JSSE 提供者之後，請務必於系統內容中適當地配置新提供者所需的數位憑證資訊 (金鑰庫)。 如需詳細資訊，請參閱下列主題：

- Java 系統內容

JSSE 安全內容:

Java 虛擬機器 (JVM) 會用到許多重要的安全內容，這些安全內容是藉由編輯 Java 主要安全內容檔來設定。

名爲 java.security 的這個檔案通常位於 iSeries 伺服器的 /QIBM/ProdData/Java400/jdk15/lib/security 目錄中。

以下清單說明使用 JSSE 的幾個重要的安全內容。 請參考這些說明來編輯 java.security 檔案。

security.provider.<integer>

您要使用的 JSSE 提供者。它也會自行登記加密提供者類別。請完全依照下列範例來指定不同的 JSSE 提供者：

```
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

指定預設的 KeyManagerFactory 演算法。若爲原有的 iSeries JSSE 提供者，請使用：

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

若爲 Pure Java JSSE 提供者，請使用：

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

如需詳細資訊，請參閱 javax.net.ssl.KeyManagerFactory 的 javadoc。

ssl.TrustManagerFactory.algorithm

指定預設的 TrustManagerFactory 演算法。若爲原有的 iSeries JSSE 提供者，請使用：

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

若爲 Pure Java JSSE 提供者，請使用：

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

如需詳細資訊，請參閱 javax.net.ssl.TrustManagerFactory 的 javadoc。

ssl.SocketFactory.provider

指定預設的 SSL Socket Factory。若爲原有的 iSeries JSSE 提供者，請使用：

```
ssl.SocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLSocketFactoryImpl
```

若爲 Pure Java JSSE 提供者，請使用：

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
```

如需詳細資訊，請參閱 javax.net.ssl.SSLSocketFactory 的 javadoc。

ssl.ServerSocketFactory.provider

指定預設的 SSL 伺服器 Socket Factory。若爲原有的 iSeries JSSE 提供者，請使用：

```
ssl.ServerSocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLServerSocketFactoryImpl
```

若爲 Pure Java JSSE 提供者，請使用：

```
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

如需詳細資訊，請參閱 javax.net.ssl.SSLServerSocketFactory 的 javadoc。

JSSE Java 系統內容:

若要在應用程式中使用 JSSE，您必須指定幾個系統內容，供預設的 SSLContext 物件用來確認配置。 有一些內容同時適用於兩個提供者，但也有一些內容僅適用於原有的 iSeries 提供者。

使用原有的 iSeries JSSE 提供者時，若未指定任何內容，則 os400.certificateContainer 將預設爲 *SYSTEM，表示 JSSE 會使用系統憑證庫內的預設項目。

適用於兩個提供者的內容

下列內容適用於兩個 JSSE 提供者。各段說明會加入可用的預設內容。

javax.net.ssl.trustStore

檔案名稱，此檔案包含您想讓預設 TrustManager 使用的 KeyStore 物件。 預設值爲 jssecacerts，若 jssecacerets 不存在，則爲 cacerts。

javax.net.ssl.trustStoreType

您希望預設 TrustManager 使用的 KeyStore 物件類型。預設值爲 KeyStore.getDefaultType 方法傳回的值。

javax.net.ssl.trustStorePassword

您希望預設 TrustManager 使用的 KeyStore 物件的密碼。

javax.net.ssl.keyStore

檔案名稱，此檔案包含您想讓預設 KeyManager 使用的 KeyStore 物件。

javax.net.ssl.keyStoreType

您希望預設 KeyManager 使用的 KeyStore 物件類型。預設值爲 KeyStore.getDefaultType 方法傳回的值。

javax.net.ssl.keyStorePassword

您希望預設 KeyManager 使用的 KeyStore 物件的密碼。

僅適用於 iSeries 原有 JSSE 提供者的內容

下列內容僅適用於原有的 iSeries JSSE 提供者。

os400.secureApplication

應用程式 ID。當您未指定下列任何一項內容時，JSSE 才會使用這個內容：

- 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

您要使用的金鑰環名稱。當您未指定下列任何一項內容時，JSSE 才會使用這個內容：

- 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

您要使用的金鑰環標籤。當您未指定下列任何一項內容時，JSSE 才會使用這個內容：

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

其他資訊

有關系統內容的資訊，請參閱下列主題：

- 第 12 頁的『Java 系統內容的清單』
- Sun Java 網站的 System Properties。

使用原有的 iSeries JSSE 提供者:

原有的 iSeries JSSE 提供者具備完整的 JSSE 類別及介面套組，包括 JSSE KeyStore 類別及 SSLConfiguration 類別的實作方式。

爲了有效率地使用原有的 iSeries 提供者，請使用本主題中的資訊，並請參閱 第 253 頁的『SSLConfiguration Javadoc information (無中文)』。

SSLContext.getInstance 方法的通訊協定值

下表針對原有的 iSeries JSSE 提供者，識別及說明 SSLContext.getInstance 方法的通訊協定值。

通訊協定值	支援的 SSL 通訊協定
SSL	SSL 第 2 版、SSL 第 3 版及 TLS 第 1 版
SSLv2	SSL 第 2 版

通訊協定值	支援的 SSL 通訊協定
SSLv3	SSL 第 3 版
TLS	SSL 第 2 版、SSL 第 3 版及 TLS 第 1 版
TLSv1	TLS 第 1 版
SSL_TLS	SSL 第 2 版、SSL 第 3 版及 TLS 第 1 版

原有的 iSeries KeyStore 實作方式

原有的 iSeries 提供者提供 IbmISeriesKeyStore 類型的 KeyStore 類別實作方式。此金鑰庫實作方式可爲「數位憑證管理程式」支援提供一個 wrapper 程式。 金鑰庫的內容取決於特定的應用程式 ID 或金鑰環檔案、密碼及標籤。 JSSE 會透過「數位憑證管理程式」載入金鑰庫項目。當應用程式第一次試圖存取金鑰庫項目或金鑰庫資訊時， JSSE 會使用適當的應用程式 ID 或金鑰環資訊來載入項目。您無法修改金鑰庫，所有配置變更皆必須透過「數位憑證管理程式」來完成。

有關使用「數位憑證管理程式」的資訊，請參閱下列主題：

數位憑證管理程式

原有的 iSeries 提供者的使用建議

下列建議有助於原有的 iSeries 提供者儘可能地提升執行效能。

- 爲了讓原有的 iSeries JSSE 提供者順利運作，JSSE 應用程式只能使用原有實作方式的元件。例如，原有的 iSeries JSSE 支援的應用程式，若使用透過 Pure Java JSSE 提供者所建立的 X509KeyManager 物件，則無法順利起始設定透過原有的 iSeries JSSE 提供者所建立的 SSLContext 物件。
- 此外，您還必須使用 IbmISeriesKeyStore 物件或 com.ibm.as400.SSLConfiguration 物件，來起始設定原有 iSeries 提供者的 X509KeyManager 及 X509TrustManager 實作方式。

註: 在未來的版次中可能會修改上述建議，屆時原有的 iSeries JSSE 提供者即可讓您嵌入非原有的元件 (例如，JKS KeyStore 或 IbmX509 TrustManagerFactory)。

SSLConfiguration Javadoc information (無中文):

com.ibm.as400

Class SSLConfiguration

```
java.lang.Object
  |
  +--com.ibm.as400.SSLConfiguration
```

All Implemented Interfaces:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

public final class **SSLConfiguration**
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable

This class provides for the specification of the configuration needed by the native iSeries JSSE implementation.

The native iSeries JSSE implementation works the most efficiently using a `KeyStore` object of type ″IbmISeriesKeyStore″. This type of `KeyStore` object contains key entries and trusted certificate entries based either

on an application identifier registered with the Digital Certificate Manager (DCM) or on a keyring file (digital certificate container). A KeyStore object of this type can then be used to initialize an X509KeyManger and an X509TrustManager object from the ″IbmISeriesSslProvider″ Provider. The X509KeyManager and X509TrustManager objects can then be used to initialize an SSLContext object from the ″IbmISeriesSslProvider″. The SSLContext object then provides access to the native iSeries JSSE implementation based on the configuration information specified for the KeyStore object. Each time a load is performed for an ″IbmISeriesKeyStore″ KeyStore, the KeyStore is initialized based on the current configuration specified by the application identifier or keyring file.

This class can also be used to generate a KeyStore object of any valid type. The KeyStore is initialized based on the current configuration specified by the application identifier or keyring file. Any change made to the configuration specified by an application identifier or keyring file would require the KeyStore object to be regenerated to pick up the change. Note that a keyring password must be specified (for the *SYSTEM certificate store when using an application ID) to be able to successfully create a KeyStore of a type other than ″IbmISeriesKeyStore″. The keyring password must be specified to successfully gain access to any private key for any KeyStore of type ″IbmISeriesKeyStore″ which is created.

Since: SDK 1.4

See Also:

KeyStore, X509KeyManager, X509TrustManager, SSLContext

Constructor Summary

SSLConfiguration() Creates a new SSLConfiguration. See 第 255 頁的『Constructor detail 』 for more information.

表 6. **Method Summary**

void	第 258 頁的『clear』 () Clears all information in the object so that all of the get methods return null.
java.lang.Object	第 259 頁的『clone』 () Generates a new copy of this SSL configuration.
boolean	第 259 頁的『equals』 (java.lang.Objectobj) Indicates whether some other object is ″equal to″ this one.
protected void	第 258 頁的『finalize』 () Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
java.lang.String	第 257 頁的『getApplicationId』 () Returns the application ID.
java.lang.String	第 257 頁的『getKeyringLabel』 () Returns the keyring label.
java.lang.String	第 257 頁的『getKeyringName』 () Returns the keyring name.
char[]	第 258 頁的『getKeyringPassword』 () Returns the keyring password.
java.security.KeyStore	第 260 頁的『getKeyStore』 (char[]password) Returns a keystore of type ″IbmISeriesKeyStore″ using the given password.
java.security.KeyStore	第 260 頁的『 getKeyStore』 (java.lang.Stringtype, char[]password) Returns a keystore of the requested type using the given password.
int	第 259 頁的『hashCode』 () Returns a hash code value for the object.
staticvoid	(java.lang.String[]args) Executes SSLConfiguration functions.
void	(java.lang.String[]args, java.io.PrintStreamout) Executes SSLConfiguration functions.
void	第 259 頁的『setApplicationId』 (java.lang.StringapplicationId) Sets the application ID.
void	第 259 頁的『setApplicationId』 (java.lang.StringapplicationId, char[]password) Sets the application ID and the keyring password.

表 6. **Method Summary** *(繼續)*

void	第 258 頁的『setKeyring』(java.lang.Stringname,java.lang.Stringlabel, char[]password) Sets the keyring information.

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, toString, wait, wait, wait

Constructor detail

SSLConfiguration

```
public SSLConfiguration()
```

Creates a new SSLConfiguration. The application identifier and keyring information is initialized to default values.

The default value for the application identifier is the value specified for the ″os400.secureApplication″ property.

The default values for the keyring information is null if the ″os400.secureApplication″ property is specified. If the ″os400.secureApplication″ property is not specified, then the default value for the keyring name is the value specified for the″os400.certificateContainer″ property. If the ″os400.secureApplication″ property is not specified, then the keyring label is initialized to the value of the ″os400.certificateLabel″ property. If neither of the ″os400.secureApplication″ or ″os400.certificateContainer″ properties are set, then the keyring name will be initialized to ″*SYSTEM″.

Method detail

---

main

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

Executes SSLConfiguration functions. There are four commands that can be performed: -help, -create, -display, and -update. The command must be the first parameter specified.

The following are the options which may be specified (in any order):

-keystore **keystore-file-name**
: Specifies the name of the keystore file to be created, updated or displayed. This option is required for all commands.

-storepass **keystore-file-password**
: Specifies the password associated with the keystore file to be created, updated, or displayed. This option is required for all commands.

-storetype **keystore-type**
: Specifies the type of keystore file to be created, updated, or displayed. This option may be specified for any command. If this option is not specified, then a value of ″IbmISeriesKeyStore″ is used.

-appid **application-identifier**

Specifies the application identifier to be used to initialize a keystore file being created or updated. This option is optional for the *-create* and *-update* commands. Only one of the *-appid*, *keyring*, and *-systemdefault* options may be specified.

-keyring **keyring-file-name**

Specifies the keyring file name to be used to initialize a keystore file being created or updated. This option is optional for the *-create* and *-update* commands. Only one of the *-appid*, *keyring*, and *-systemdefault* options may be specified.

-keyringpass **keyring-file-password**

Specifies the keyring file password to be used to initialize a keystore file being created or updated. This option may be specified for the *-create* and *-update* commands and is required when a keystore type other than ″IbmISeriesKeyStore″ is specified. If this option is not specified, then the stashed keyring password is used.

-keyringlabel **keyring-file-label**

Specifies the keyring file label to be used to initialize a keystore file being created or updated. This option may only be specified when the *-keyring* option is also specified. If this option is not specified when the *keyring* option is specified, then the default label in the keyring is used.

-systemdefault

Specifies the system default value is to be used to initialize a keystore file being created or updated. This option is optional for the *-create* and *-update* commands. Only one of the *-appid*, *keyring*, and *-systemdefault* options may be specified.

-v Specifies that verbose output is to be produced. This option may be specified for any command.

The help command displays usage information for specifying the paramters to this method. The parameters to invoke the help function is specified as follows:

```
-help
```

The create command creates a new keystore file. There are three variations of the create command. One variation to create a keystore based on a particular application identifier, another variation to create a keystore based on a keyring name, label, and password, and a third variation to create a keystore based on the system default configuration.

To create a keystore based on a particular application identifier, the *-appid* option must be specified. The following parameters would create a keystore file of type ″IbmISeriesKeyStore″ named ″keystore.file″ with a password of ″keypass″ which is initialized based on the application identifier ″APPID″:

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -appid APPID
```

To create a keystore based on a particular keyring file, the *-keyring* option must be specified. The *-keyringpass* and *keyringlabel* options may also be specified. The following parameters would create a keystore file of type ″IbmISeriesKeyStore″ named ″keystore.file″ with a password of ″keypass″ which is initialized based on the keyring file named ″keyring.file″, keyring password ″ringpass″, and keyring label ″keylabel″:

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

To create a keystore based on the system default configuration, the *-systemdefault* option must be specified. The following parameters would create a keystore file of type ″IbmISeriesKeyStore″ named ″keystore.file″ with a password of ″keypass″ which is initialized based on the system default configuration:

```
-create -keystore keystore.file -storepass keypass -systemdefault
```

The update command updates an existing keystore file of type ″IbmISeriesKeyStore″. There are three variations of the update command which are identical to the variations of the create command. The options for the update command are identical to the options used for the create command. The display command displays the configuration specified for an existing keystore file. The following parameters would display the configuration specified by a keystore file of type ″IbmISeriesKeyStore″ named ″keystore.file″ with a password of ″keypass″:

```
-display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

Parameters:

`args` - the command line arguments

---

run

```
public void run(java.lang.String[]args,
                java.io.PrintStreamout)
```

Executes SSLConfiguration functions. The parameters and functionality of this method are identical to the main() method.

Parameters:

`args` - the command arguments

`out` - output stream to which results are to be written

See Also:`com.ibm.as400.SSLConfiguration.main()`

---

getApplicationId

```
public java.lang.String getApplicationId()
```

Returns the application ID.

Returns:

the application ID.

---

getKeyringName

```
public java.lang.String getKeyringName()
```

Returns the keyring name.

Returns:

the keyring name.

---

getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

Returns the keyring label.

Returns:

the keyring label.

---

getKeyringPassword

```
public final char[] getKeyringPassword()
```

Returns the keyring password.

Returns:

the keyring password.

---

finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

Overrides:

`finalize` in class `java.lang.Object`

Throws:

`java.lang.Throwable` - the exception raised by this method.

---

clear

```
public void clear()
```

Clears all information in the object so that all of the get methods return null.

---

setKeyring

```
public void setKeyring(java.lang.Stringname,
                       java.lang.Stringlabel,
                       char[]password)
```

Sets the keyring information.

Parameters:

`name` - the keyring name

`label` - the keyring label, or null if the default keyring entry is to be used.

`password` - the keyring password, or null if the stashed password is to be used.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId)
```

Sets the application ID.

Parameters:

applicationId - the application ID.

---

setApplicationId

```
public void setApplicationId(java.lang.StringapplicationId,
                             char[]password)
```

Sets the application ID and the keyring password. Specifying the keyring password allows any keystore which is created to allow access to the private key.

Parameters:

applicationId - the application ID.

password - the keyring password.

---

equals

```
public boolean equals(java.lang.Objectobj)
```

Indicates whether some other object is ″equal to″ this one.

Overrides:

equals in class java.lang.Object

Parameters:

obj - object to be compared

Returns:

indicator of whether the objects specify the same configuration information

---

hashCode

```
public int hashCode()
```

Returns a hash code value for the object.

Overrides:

hashCode in class java.lang.Object

Returns:

a hash code value for this object.

---

clone

```
public java.lang.Object clone()
```

Generate a new copy of this SSL configuration. Subsequent changes to the components of this SSL configuration will not affect the new copy, and vice versa.

Overrides:

clone in class java.lang.Object

Returns:

a copy of this SSL configuration

---

getKeyStore

```
public java.security.KeyStore getKeyStore(char[]password)
                                    throws java.security.KeyStoreException
```

Returns a keystore of type ″IbmISeriesKeyStore″ using the given password. The keystore is intialized based on the configuration information currently stored in the object.

Parameters:

password - used to initialize the keystore

Returns:

KeyStore keystore initialized based on the configuration information currently stored in the object

Throws:

java.security.KeyStoreException - if the keystore could not be created

---

getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                          char[]password)
                                    throws java.security.KeyStoreException
```

Returns a keystore of the requested type using the given password. The keystore is initialized based on the configuration information currently stored in the object.

Parameters:

type - type of keystore to be returned

password - used to initialize the keystore

Returns:

KeyStore keystore initialized based on the configuration information currently stored in the object

Throws:

java.security.KeyStoreException - if the keystore could not be created

範例：IBM Java Secure Sockets Extension:

以下 JSSE 範例顯示用戶端及伺服器如何使用原有的 iSeries JSSE 提供者來建立可進行安全通訊的環境定義。

註: 無論 java.security 檔案指定的內容爲何，這兩個範例一律使用原有的 iSeries JSSE 提供者。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

第 261 頁的『範例：使用 SSLContext 物件的 SSL 用戶端』

此用戶端程式範例會起始設定一個 SSLContext 物件，並透過它來使用 ″MY_CLIENT_APP″ 應用程式 ID。 無論 java.security 檔案中指定的內容爲何，此程式一律使用原有的 iSeries 實作方式。

第 263 頁的『範例：使用 SSLContext 物件的 SSL 伺服器』

下列伺服器程式將使用先前建立的金鑰庫檔案所起始設定的 SSLContext 物件。 金鑰庫檔案的名稱爲 /home/keystore.file，金鑰庫密碼爲 password。

範例程式需要金鑰庫檔案來建立 IbmISeriesKeyStore 物件。 KeyStore 物件必須指定 MY_SERVER_APP 作爲應用程式 ID。

您可以使用下列指令來建立金鑰庫檔案：

- 從 Qshell 指令提示：

```
java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
   -storepass password -appid MY_SERVER_APP
```

  如需在 Qshell 中使用 Java 指令的相關資訊，請參閱「iSeries 資訊中心」的 Qshell。

- 從 iSeries 指令提示：

```
RUNJVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
           '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
```

範例：使用 SSLContext 物件的 SSL 用戶端:

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
////////////////////////////////////////////////////////////////////////////
//
// This example client program utilizes an SSLContext object, which it initializes
// to use the "MY_CLIENT_APP" application ID.
//
// The example uses the native iSeries JSSE provider, regardless of the
// properties specified by the java.security file.
//
// Command syntax:
//     java -Djava.version=1.4 SslClient
//
// Note that "-Djava.version=1.4" is unnecessary when you have configured
// J2SDK version 1. to be used by default.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;

/**
  * SSL Client Program.
 */
public class SslClient {

   /**
        * SslClient main method.
    *
        * @param args the command line arguments (not used)
    */
      public static void main(String args[]) {
       /*
                * Set up to catch any exceptions thrown.
        */
       try {
           /*
            * Initialize an SSLConfiguration object to specify an application
```

```
     * ID.  "MY_CLIENT_APP" must be registered and configured
     * correctly with the Digital Certificate Manager (DCM).
     */
    SSLConfiguration config = new SSLConfiguration();
    config.setApplicationId("MY_CLIENT_APP"
    /*
     * Get a KeyStore object from the SSLConfiguration object.
     */
    Char[] password = "password".toCharArray();
    KeyStore ks = config.getKeyStore(password);
    /*
     * Allocate and initialize a KeyManagerFactory.
     */
    KeyManagerFactory kmf =
                      KeyManagerFactory.getInstance("IbmISeriesX509");
    Kmf.init(ks, password);
    /*
     * Allocate and initialize a TrustManagerFactory.
     */
    TrustManagerFactory tmf =
                    TrustManagerFactory.getInstance("IbmISeriesX509");
    tmf.init(ks);
    /*
     * Allocate and initialize an SSLContext.
     */
    SSLContext c =
               SSLContext.getInstance("SSL", "quot;);
    C.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
    /*
     * Get the an SSLSocketFactory from the SSLContext.
     */
    SSLSocketFactory sf = c.getSocketFactory();
    /*
     * Create an SSLSocket.
     *
     * Change the hard-coded IP address to the IP address or host name
     * of the server.
     */
    SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
    /*
     * Send a message to the server using the secure session.
     */
    String sent = "Test of java SSL write";
    OutputStream os = s.getOutputStream();
    os.write(sent.getBytes());
    /*
     * Write results to screen.
     */
    System.out.println("Wrote " + sent.length() + " bytes...");
    System.out.println(sent);
    /*
     * Receive a message from the server using the secure session.
     */
    InputStream is = s.getInputStream();
    byte[] buffer = new byte[1024];
    int bytesRead = is.read(buffer);
    if (bytesRead == -1)
        throw new IOException("Unexpected End-of-file Received");
    String received = new String(buffer, 0, bytesRead);
    /*
     * Write results to screen.
     */
    System.out.println("Read " + received.length() + " bytes...");
    System.out.println(received);
} catch (Exception e) {
    System.out.println("Unexpected exception caught: " +
                       e.getMessage());
```

```
            e.printStackTrace();
        }
    }

}
```

鏈結集合

程式碼範例免責聲明

範例：使用 SSLContext 物件的 SSL 伺服器:

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
////////////////////////////////////////////////////////////////////////////
//
// The following server program utilizes an SSLContext object that it
// initializes with a previously created keystore file.
//
// The keystore file has the following name and keystore password:
//   File name: /home/keystore.file
//   Password:  password
//
// The example program needs the keystore file in order to create an
// IbmISeriesKeyStore object. The KeyStore object must specify MY_SERVER_APP as
// the application identifier.
//
// To create the keystore file, you can use the following Qshell command:
//
//   java com.ibm.as400.SSLConfiguration  -create  -keystore /home/keystore.file
//     -storepass password  -appid MY_SERVER_APP
//
// Command syntax:
//     java -Djava.version=1.4 JavaSslServer
//
// Note that "-Djava.version=1.4" is unnecessary when you have configured
// J2SDK version 1. to be used by default.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;

/**
* Java SSL Server Program using Application ID.
*/
public class JavaSslServer {

   /**
    * JavaSslServer main method.
    *
    * @param args the command line arguments (not used)
    */
   public static void main(String args[]) {
       /*
        * Set up to catch any exceptions thrown.
        */
       try {
           /*
            * Allocate and initialize a KeyStore object.
            */
           Char[] password = "password".toCharArray();
           KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
           FileInputStream fis = new FileInputStream("/home/keystore.file"
           Ks.load(fis, password);
           /*
            * Allocate and initialize a KeyManagerFactory.
```

```
         */
        KeyManagerFactory kmf =
                          KeyManagerFactory.getInstance("IbmISeriesX509");
        Kmf.init(ks, password);
        /*
         * Allocate and initialize a TrustManagerFactory.
         */
        TrustManagerFactory tmf =
                        TrustManagerFactory.getInstance("IbmISeriesX509");
        tmf.init(ks);
        /*
         * Allocate and initialize an SSLContext.
         */
        SSLContext c =
                   SSLContext.getInstance("SSL", "IbmISeriesSslProvider");
        C.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
        /*
         * Get the an SSLServerSocketFactory from the SSLContext.
         */
        SSLServerSocketFactory sf = c.getSSLServerSocketFactory();
        /*
         * Create an SSLServerSocket.
         */
        SSLServerSocket ss =
            (SSLServerSocket) sf.createServerSocket(13333);
        /*
         * Perform an accept() to create an SSLSocket.
         */
        SSLSocket s = (SSLSocket) ss.accept();
        /*
         * Receive a message from the client using the secure session.
         */
        InputStream is = s.getInputStream();
        byte[] buffer = new byte[1024];
        int bytesRead = is.read(buffer);
        if (bytesRead == -1)
            throw new IOException("Unexpected End-of-file Received");
        String received = new String(buffer, 0, bytesRead);
        /*
         * Write results to screen.
         */
        System.out.println("Read " + received.length() + " bytes...");
        System.out.println(received);
        /*
         * Echo the message back to the client using the secure session.
         */
        OutputStream os = s.getOutputStream();
        os.write(received.getBytes());
        /*
         * Write results to screen.
         */
        System.out.println("Wrote " + received.length() + " bytes...");
        System.out.println(received);
    } catch (Exception e) {
        System.out.println("Unexpected exception caught: " +
                           e.getMessage());
        e.printStackTrace();
    }
  }

}
```

鏈結集合

程式碼範例免責聲明

Java 鑑別和授權服務

「Java 鑑別和授權服務 (JAAS)」是 Java 2 Software Development Kit (J2SDK) 標準版的標準延伸套件。J2SDK 會根據程式碼的出處及簽章者來提供存取控制 (依照程式碼來源的存取控制)。然而，對於是誰執行程式碼，卻無能力強加其他存取控制。 JAAS 提供了組織架構，在 Java 2 安全模型中加入這項支援。

IBM 及 Sun Microsystems 公司使用 JAAS API 來延伸 J2SDK 1.3 版。IBM 及 Sun 提供這項延伸套件，讓特定使用者或身份與現行 Java 緒得以結合起來。可以利用 javax.security.auth.Subject 方法來達成，亦可選擇利用 com.ibm.security.auth.ThreadSubject 方法來運用基礎的作業系統緒。

註: 在 J2SDK 1.4 版及後續版本中，JAAS 不再是延伸套件，而是基本 SDK 的一部分。

JAAS 在 iSeries 伺服器上的實作方式與 Sun Microsystems 公司的實作方式相容。本文件僅針對 iSeries 實作方式做說明。我們假設您對 JAAS 延伸套件的一般文件已很熟悉。茲提供下列鏈結，讓您更易於使用此延伸套件及 iSeries 資訊。

- 第 267 頁的『Java Authentication and Authorization Service (JAAS) 1.0 (無中文)』提供在軟體開發過程中使用 JAAS API 的相關資訊。
- JAAS LoginModule 程式開發手冊將焦點集中在 JAAS 鑑別方面。
- JAAS API 規格包含 Javadoc 提供的 JAAS 資訊。

  相關資訊

  iSeries 伺服器專屬的 JAAS Javadoc

準備及配置 iSeries 伺服器來使用 Java 鑑別和授權服務 (JAAS)

您必須符合軟體需求並配置 iSeries 伺服器，才能使用「Java 鑑別和授權服務 (JAAS)」。

在 iSeries 伺服器上執行 JAAS 1.0 的軟體需求

請安裝下列授權程式：

- Java 2 SDK 1.4 版 (J2SDK) 或更新版本
- 若要變更 OS 緒身份，必須要有 IBM Toolbox for Java (修正 4) 授權程式 (5722-JC1)。其中包含支援變更 iSeries OS 緒身份所需的 ProfileTokenCredential 類別，以及原有實作類別。

配置系統

若要配置系統來使用 JAAS，請遵循下列步驟：

1. 若爲 J2SDK 1.3，請對 jaas13.jar 檔案所在的延伸套件目錄新增符號鏈結。延伸類別載入器應該載入此 JAR 檔。請在 iSeries 指令行執行下列指令 (全部在同一行) 以新增鏈結：

   ```
   ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas13.jar')
     NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jaas13.jar')
   ```

   附註：若爲 J2SDK 1.4 及更新版本，則不必新增延伸套件目錄的符號鏈結。因爲在此版本中，JAAS 已屬於基本 SDK 的一部份。

2. ${java.home}/lib/security 中提供的預設 login.config 檔案，可以呼叫 com.ibm.as400.security.auth.login.BasicAuthenticationLoginModule。此 login.config 檔案會將單一用途的 ProfileTokenCredential 附加至通過鑑別的主旨中。若想要使用自己的 login.config 檔案來設定不同選項，請於呼叫應用程式時，加入下列系統內容：

   ```
   -Djava.security.auth.login.config=your login.config file
   ```

3. 新增符號鏈結來鏈結 jt400Native.jar 檔案所在的延伸套件目錄。如此延伸類別載入器就能夠載入此檔案。jaas13.jar 檔案需要此 JAR 檔，才能取得屬於 IBM Toolbox for Java 的認證實作類別。在 CLASSPATH 中加入此檔案，一樣可讓應用程式類別載入器載入此檔案。若是從類別路徑目錄中載入此檔案，請勿新增延伸套件目錄的符號鏈結。

   建立從 jt400Native.jar 檔案至 /QIBM/ProdData/Java400/jdk14/lib/ext 目錄的符號鏈結，將強制伺服器的所有 J2SDK 1.4 使用者，全部採用這個 jt400Native.jar 版本。但是，若不同使用者需要不同版本的 IBM Toolbox for Java 類別，這樣就不對了。將 jt400Native.jar 放在前述的應用程式 CLASSPATH 中，也是一種做法。另一種方法是對您自己的目錄新增符號鏈結，然後於呼叫應用程式時，指定 java.ext.dirs 系統內容，將此目錄納入延伸套件的目錄類別路徑之中。

   若要將 jt400Native.jar 檔案鏈結至 /QIBM/ProdData/Java400/jdk13/lib/ext 目錄，請在 iSeries 指令行執行下列指令來新增鏈結：

   ```
   ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
        NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400Native.jar')
   ```

   若要將 jt400Native.jar 檔案鏈結至 /QIBM/ProdData/Java400/jdk14/lib/ext 目錄，請在 iSeries 指令行執行下列指令來新增鏈結：

   ```
   ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
        NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')
   ```

   若要將 jt400Native.jar 檔鏈結至您自己的目錄，請執行下列動作：

   a. 請在 iSeries 指令行執行下列指令來新增鏈結：

      ```
      ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
            NEWLNK('your extension directory/jt400Native.jar')
      ```

   b. 請使用下列型樣來呼叫 Java 程式：

      ```
      java -Djava.ext.dirs=your extension directory:default
      extension directories
      ```

   註: 如需 iSeries 認證類別的相關資訊，請參閱 IBM Toolbox for Java。按一下 **Security 類別**。按一下**鑑別服務**。再按一下 **ProfileTokenCredential** 類別。然後按一下**套件**。

4. 更新 Java 2 原則檔，對 IBM Toolbox for Java JAR 檔案的實際位置，授予適當的許可權。即使這些檔案可能已透過符號鏈結的方式鏈結至延伸套件目錄，且在 ${java.home}/lib/security/java.policy 檔案中，也已將 java.security.AllPermission 授予這些目錄，仍然會根據 JAR 檔的實際位置來授權。

   爲了順利使用 IBM Toolbox for Java 的認證類別，請在應用程式的 Java 2 原則檔中加入下列許可權：

   ```
   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";
    }
   ```

   您也需要在應用程式的 codeBase 中新增這些許可權，因爲 IBM Toolbox for Java JAR 檔所執行的作業，並非執行於特許模式中。

   如需 Java 2 原則檔的相關資訊，請參閱第 267 頁的『Java Authentication and Authorization Service (JAAS) 1.0 (無中文)』。

5. 請確定「iSeries 主電腦伺服器」已啓動且正在執行。位於 Toolbox (例如 jt400Native.jar) 中的 ProfileTokenCredential 類別，可用來作爲認證，並且附加至鑑別的主旨。認證類別需要這些「主電腦伺服器」的存取權限。您可以在 iSeries 指令提示字元上鍵入下列指令，藉以驗證伺服器是否已啓動且正在執行：

```
StrHostSVR *all
StrTcpSvr *DDM
```

若伺服器已啓動，則這些步驟不執行任何動作。若伺服器尚未啓動，則會執行這些步驟予以啓動。

Java Authentication and Authorization Service (JAAS) 1.0 (無中文)

This document was last updated March 17, 2000.

Developer's Guide

- **Overview**
- **Who Should Read This Document**
- **Related Documentation**
- **Introduction**
- **Core Classes**
- **Common Classes**
  - **Subject**
  - **Principals**
  - **Credentials**
- **Authentication Classes**
- **LoginContext**
- **LoginModule**
- **CallbackHandler**
- **Callback**
- **Authorization Classes**
- **Policy**
- **AuthPermission**
- **PrivateCredentialPermission**

References

- Implementation
- ″Hello World″, JAAS style!
- Appendix A: JAAS Settings in the java.security Security Properties File
- Appendix B: Login Configuration File
- Appendix C: Authorization Policy File

Overview

The Java Authentication and Authorization Service (JAAS) is a standard extension to the Java 2 Software Development Kit, version 1.3. Currently, Java 2 provides codesource-based access controls (access controls based on *where* the code originated from and *who signed* the code). It lacks, however, the ability to additionally enforce access controls based on *who runs* the code. JAAS provides a framework that augments the Java 2 security model with such support.

Who Should Read This Document

This document is intended for experienced programmers wanting to create applications constrained by a codesource-based and Subject-based security model.

Related Documentation

This document assumes you have already read the following documentation:

- Java 2 Software Development Kit API Specification
- JAAS API Specification
- Security and the Java platform

A supplement to this guide is the LoginModule Developer's Guide that is supplied by Sun Microsystems, Inc.

Introduction

The JAAS infrastructure can be divided into two main components: an **authentication** component and an **authorization** component. The JAAS authentication component provides the ability to reliably and securely determine who is currently processing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet. The JAAS authorization component supplements the existing Java 2 security framework by providing the means to restrict the processing Java code from performing sensitive tasks, depending on its codesource (as is done in Java 2) and depending on who was authenticated.

JAAS authentication is performed in a *pluggable* fashion. This permits Java applications to remain independent from underlying authentication technologies. Therefore new or updated authentication technologies can be plugged under an application without requiring modifications to the application itself. Applications enable the authentication process by instantiating a

`LoginContext`

object, which in turn references a

`Configuration`

to determine the authentication technology, or

`LoginModule`

, to be used in performing the authentication. Typical LoginModules may prompt for and verify a username and password. Others may read and verify a voice or fingerprint sample.

Once the user processing the code has been authenticated, the JAAS authorization component works in conjunction with the existing Java 2 access control model to protect access to sensitive resources. Unlike in Java 2, where access control decisions are based solely on code location and code signers (a

`CodeSource`

), in JAAS access control decisions are based both on the processing code's

`CodeSource`

, as well as on the user running the code, or the

`Subject`

. Note that the JAAS policy merely extends the Java 2 policy with the relevant Subject-based information. Therefore permissions recognized and understood in Java 2 (

`java.io.FilePermission`

and

`java.net.SocketPermission`

, for example) are also understood and recognized by JAAS. Furthermore, although the JAAS security policy is physically separate from the existing Java 2 security policy, the two policies, together, form one logical policy.

Core Classes

The JAAS core classes can be broken into 3 categories: Common, Authentication, and Authorization.

- Common Classes
  - Subject, Principals, Credentials
- Authentication Classes
  - LoginContext, LoginModule, CallbackHandler, Callback
- Authorization Classes
  - Policy, AuthPermission, PrivateCredentialPermission

Common Classes

Common classes are shared within both the JAAS authentication and authorization components.

The key JAAS class is

`Subject`

, which represents a grouping of related information for a single entity such as a person. It encompasses the entity's Principals, public credentials, and private credentials.

Note that JAAS uses the existing Java 2

`java.security.Principal`

interface to represent a Principal. Also note that JAAS does not introduce a separate credential interface or class. A credential, as defined by JAAS, may be any Object.

Subject

To authorize access to resources, applications first need to authenticate the source of the request. The JAAS framework defines the term, Subject, to represent the source of a request. A Subject may be any entity, such as a person or service. Once authenticated, a Subject is populated with associated identities, or Principals. A Subject may have many Principals. For example, a person may have a name Principal (″John Doe″) and a SSN Principal (″123-45-6789″) which distinguishes it from other Subjects.

A

`Subject`

may also own security-related attributes, which are referred to as credentials. Sensitive credentials that require special protection, such as private cryptographic keys, are stored within a private credential

`Set`

. Credentials intended to be shared, such as public key certificates or Kerberos tickets are stored within a public credential

`Set`

. Different permissions are required to access and modify the different credential Sets.

Subjects are created using these constructors:

```
public Subject();

public Subject(boolean readOnly, Set principals,
               Set pubCredentials, Set privCredentials);
```

The first constructor creates a Subject with empty (non-null) Sets of Principals and credentials. The second constructor creates a Subject with the specified Sets of Principals and credentials. It also has a boolean argument which can create a read-only Subject (immutable Principal and credential Sets).

An alternative way to obtain a reference to an authenticated Subject without using these constructors will be shown in the LoginContext section.

If a Subject was not instantiated to be in a read-only state, it can be set to a read-only state by calling this method:

```
public void setReadOnly();
```

An

`AuthPermission("setReadOnly")`

is required to invoke this method. Once in a read-only state, any attempt to add or remove Principals or credentials will result in an

`IllegalStateException`

being thrown.

This method may be called to test a Subject's read-only state:

```
public boolean isReadOnly();
```

To retrieve the Principals associated with a Subject, two methods are available:

```
public Set getPrincipals();
public Set getPrincipals(Class c);
```

The first method returns all Principals contained in the Subject, while the second method only returns those Principals that are an instance of the specified Class c, or an instance of a subclass of Class c. An empty set will be returned if the Subject does not have any associated Principals.

To retrieve the public credentials associated with a Subject, these methods are available:

```
public Set getPublicCredentials();
public Set getPublicCredentials(Class c);
```

The observed behavior of these methods is identical to that for the

`getPrincipals`

method.

To access private credentials associated with a Subject, the following methods are available:

```
public Set getPrivateCredentials();
public Set getPrivateCredentials(Class c);
```

The observed behavior of these methods is identical to that for the

`getPrincipals`

and

`getPublicCredentials`

methods.

To modify or operate upon a Subject's Principal Set, public credential Set, or private credential Set, callers use the methods defined in the

`java.util.Set`

class. The following example demonstrates this:

```
Subject subject;
Principal principal;
Object credential;

// add a Principal and credential to the Subject
subject.getPrincipals().add(principal);
subject.getPublicCredentials().add(credential);
```

Note that an

`AuthPermission("modifyPrincipals")`

,

`AuthPermission("modifyPublicCredentials")`

, or

`AuthPermission("modifyPrivateCredentials")`

is required to modify the respective Sets. Also note that only the sets returned via the

`getPrincipals`

,

`getPublicCredentials`

, and

`getPrivateCredentials`

methods are backed by the Subject's respective internal sets. Therefore any modification to the returned set affects the internal sets as well. The sets returned via the

`getPrincipals(Class c)`

,

`getPublicCredentials(Class c)`

, and

`getPrivateCredentials(Class c)`

methods are not backed by the Subject's respective internal sets. A new set is created and returned for each method invocation. Modifications to these sets will not affect the Subject's internal sets. The following method returns the Subject associated with the specified

`AccessControlContext`

, or null if no Subject is associated with the specified

`AccessControlContext`

.

```
    public static Subject getSubject(final AccessControlContext acc);
```

An

`AuthPermission("getSubject")`

is required to call

`Subject.getSubject`

.

The Subject class also includes these methods inherited from

`java.lang.Object`

:

```
    public boolean equals(Object o);
    public String toString();
    public int hashCode();
```

The following static methods may be called to perform work as a particular Subject:

```
    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;
```

Both methods first associate the specified ***subject*** with the current Thread's

`AccessControlContext`

, and then process the ***action***. This achieves the effect of having the ***action*** run as the ***subject***. The first method can throw runtime exceptions but normal processing has it returning an Object from the run() method of its action argument. The second method behaves similarly except that it can throw a checked exception from its

`PrivilegedExceptionAction`

run() method. An

`AuthPermission("doAs")`

is required to call the

`doAs`

methods.

Here are two examples utilizing the first

`doAs`

method. Assume that a

`Subject`

with a Principal of class

`com.ibm.security.Principal`

named ″BOB″ has been authenticated by a

`LoginContext`

″lc″. Also, assume that a SecurityManager has been installed, and the following exists in the JAAS access control policy (see the Policy section for more details on the JAAS policy file):

```
   // Grant "BOB" permission to read the file "foo.txt"
   grant Principal com.ibm.security.Principal "BOB" {
      permission java.io.FilePermission "foo.txt", "read";
};
```

Subject.doAs Example 1

```
    class ExampleAction implements java.security.PrivilegedAction {
        public Object run() {
            java.io.File f = new java.io.File("foo.txt");

            // exists() invokes a security check
            if (f.exists()) {
                System.out.println("File foo.txt exists.");
            }
            return null;
        }
    }

    public class Example1 {
        public static void main(String[] args) {

            // Authenticate the subject, "BOB".
            // This process is described in the
            // LoginContext section.

            Subject bob;
            ...

            // perform "ExampleAction" as "BOB":
            Subject.doAs(bob, new ExampleAction());
        }
    }
```

During processing,

`ExampleAction`

will encounter a security check when it makes a call to,

`f.exists()`

. However, since

`ExampleAction`

is running as ″BOB″, and because the JAAS policy (above) grants the necessary

`FilePermission`

to ″BOB″, the

`ExampleAction`

will pass the security check.

Example 2 has the same scenario as Example 1.

Subject.doAs Example 2

```
public class Example2 {
    // Example of using an anonymous action class.
    public static void main(String[] args) {
        // Authenticate the subject, "BOB".
        // This process is described in the
        // LoginContext section.

        Subject bob;
        ...

        // perform "ExampleAction" as "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("File foo.txt exists.");
                }
                return null;
            }
        });
    }
}
```

Both examples throw a

`SecurityException`

if the example permission grant statement is altered correctly, such as adding an incorrect CodeBase or changing the Principal to ″MOE″. Removing the Principal field from the grant block and then moving it to a Java 2 policy file will not cause a

`SecurityException`

to be thrown because the permission is more general now (available to all Principals).

Since both examples perform the same function, there must be a reason to write code one way over the other. Example 1 may be easier to read for some programmers unfamiliar with anonymous classes. Also, the ***action*** class could be placed in a separate file with a unique CodeBase and then the permission grant could utilize this information. Example 2 is more compact and the ***action*** to be performed is easier to find since it is right there in the

`doAs`

call.

The following methods also perform work as a particular Subject. However, the

`doAsPrivileged`

methods will have security checks based on the supplied ***action*** and ***subject***. The supplied context will be tied to the specified ***subject*** and ***action***. A null context object will disregard the current

`AccessControlContext`

altogether.

```
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;
```

The

`doAsPrivileged`

methods behave similarly to the

`doAs`

methods: the ***subject*** is associated with the context ***acc***, an ***action*** is performed, and runtime exceptions or checked exceptions may be thrown. However, the

`doAsPrivileged`

methods first empties the existing Thread's

`AccessControlContext`

before associating the ***subject*** with the supplied context, and before invoking the ***action***. A null ***acc*** argument has the effect of causing access control decisions (invoked while the ***action*** processes) to be based solely upon the ***subject*** and ***action***. An

`AuthPermission("doAsPrivileged")`

is required when calling the

`doAsPrivileged`

methods.

Principals

As mentioned previously, Principals may be associated with a Subject. Principals represent Subject identities, and must implement the

`java.security.Principal`

and

`java.io.Serializable`

interfaces. The Subject section describes ways to update the Principals associated with a Subject.

Credentials

Public and private credential classes are not part of the core JAAS class library. Any java class, therefore, can represent a credential. However, developers may elect to have their credential classes implement two interfaces related to credentials: Refreshable and Destroyable.

Refreshable

This **interface** provides the capability for a credential to refresh itself. For example, a credential with a particular time-restricted lifespan may implement this interface to allow callers to refresh the time period for which it is valid. The interface has two abstract methods:

```
boolean isCurrent();
```

Determines if the credential is current or valid.

```
void refresh() throws RefreshFailedException;
```

Updates or extends the validity of the credential. This method implementation performs an
`AuthPermission("refreshCredential")`
security check to ensure the caller has permission to refresh the credential.

Destroyable

This **interface** provides the capability of destroying the contents within a credential. The interface has two abstract methods:

```
boolean isDestroyed();
```

Determines if the credential has been destroyed.

```
void destroy() throws DestroyFailedException;
```

Destroys and clears the information associated with this credential. Subsequent calls to certain methods on this credential will result in an
`IllegalStateException`
being thrown. This method implementation performs an
`AuthPermission("destroyCredential")`
security check to ensure the caller has permission to destroy the credential.

Authentication Classes

To authenticate a
`Subject`
, the following steps are performed:

1. An application instantiates a
   `LoginContext`
   .
2. The
   `LoginContext`
   consults a configuration to load all of the LoginModules configured for that application.
3. The application invokes the LoginContext's *login* method.
4. The *login* method invokes all of the loaded LoginModules. Each
   `LoginModule`

attempts to authenticate the

`Subject`

. Upon success, LoginModules associate relevant Principals and credentials with the

`Subject`

.

5. The

   `LoginContext`

   returns the authentication status to the application.
6. If authentication succeeded, the application retrieves the authenticated

   `Subject`

   from the

   `LoginContext`

   .

LoginContext

The

`LoginContext`

class provides the basic methods used to authenticate Subjects, and provides a way to develop an application independent of the underlying authentication technology. The

`LoginContext`

consults a configuration

`Configuration`

to determine the authentication services, or LoginModules, configured for a particular application. Therefore, different LoginModules can be plugged in under an application without requiring any modifications to the application itself.

`LoginContext`

offers four constructors to choose from:

```
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
```

All of the constructors share a common parameter: ***name***. This argument is used by the

`LoginContext`

to index the login Configuration. Constructors that do not take a

`Subject`

as an input parameter instantiate a new

`Subject`

. Null inputs are disallowed for all constructors. Callers require an

`AuthPermission("createLoginContext")`

to instantiate a

`LoginContext`

.

Actual authentication occurs with a call to the following method:

```
public void login() throws LoginException;
```

When *login* is invoked, all of the configured LoginModules' respective *login* methods are invoked to perform the authentication. If the authentication succeeded, the authenticated

`Subject`

(which may now hold Principals, public credentials, and private credentials) can be retrieved by using the following method:

```
public Subject getSubject();
```

To logout a

`Subject`

and remove its authenticated Principals and credentials, the following method is provided:

```
public void logout() throws LoginException;
```

The following snippet of code in an application will authenticate a Subject called ″bob″ after accessing a configuration file with a configuration entry named ″moduleFoo″:

```
Subject bob = new Subject();
LoginContext lc = new LoginContext("moduleFoo", bob);
try {
    lc.login();
    System.out.println("authentication successful");
} catch (LoginException le) {
    System.out.println("authentication unsuccessful"+le.printStackTrace());
}
```

This snippet of code in an application will authenticate a ″nameless″ Subject and then use the getSubject method to retrieve it:

```
LoginContext lc = new LoginContext("moduleFoo");
try {
    lc.login();
    System.out.println("authentication successful");
} catch (LoginException le) {
    System.out.println("authentication unsuccessful"+le.printStackTrace());
}
Subject subject = lc.getSubject();
```

If the authentication failed, then *getSubject* returns null. Also, there isn't an

`AuthPermission("getSubject")`

required to do this as is the case for

`Subject.getSubject`

LoginModule

The LoginModule **interface** gives developers the ability to implement different kinds of authentication technologies that can be plugged under an application. For example, one type of

`LoginModule`

may perform a username/password-based form of authentication.

The LoginModule Developer's Guide is a detailed document that gives developers step-by-step instructions for implementing LoginModules.

To instantiate a

`LoginModule`

, a

`LoginContext`

expects each

`LoginModule`

to provide a public constructor that takes no arguments. Then, to initialize a

`LoginModule`

with the relevant information, a

`LoginContext`

calls the LoginModule's

`initialize`

method. The provided *subject* is guaranteed to be non-null.

```
void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options);
```

This following method begins the authentication process:

```
boolean login() throws LoginException;
```

An example method implementation may prompt the user for a username and password, and then verify the information against the data stored in a naming service such as NIS or LDAP. Alternative implementations might interface smart cards and biometric devices, or may simply extract user information from the underlying operating system. This is considered ***phase 1*** of the JAAS authentication process.

The following method completes and finalizes the authentication process:

```
boolean commit() throws LoginException;
```

If ***phase 1*** of the authentication process was successful, then this method continues with ***phase 2***: associating Principals, public credentials, and private credentials with the Subject. If ***phase 1*** failed, then the *commit* method removes any previously stored authentication state, such as usernames and passwords.

The following method halts the authentication process if ***phase 1*** was unsuccessful:

```
boolean abort() throws LoginException;
```

Typical implementations of this method clean up previously stored authentication state, such as usernames or passwords. The following method logs out a Subject:

```
boolean logout() throws LoginException;
```

This method removes the Principals and credentials originally associated with the

`Subject`

during the

`commit`

operation. Credentials are destroyed upon removal.

CallbackHandler

In some cases a LoginModule must communicate with the user to obtain authentication information. LoginModules use a CallbackHandler for this purpose. Applications implement the CallbackHandler **interface** and pass it to the LoginContext, which forwards it directly to the underlying LoginModules. LoginModules use the CallbackHandler both to gather input from users (such as a password or smart card pin number) or to supply information to users (such as status information). By allowing the application to specify the CallbackHandler, underlying LoginModules can remain independent of the different ways applications interact with users. For example, the implementation of a CallbackHandler for a GUI application might display a Window to solicit input from a user. On the other hand, the implementation of a CallbackHandler for a non-GUI tool might prompt the user for input directly from the command line.

`CallbackHandler`

is an **interface** with one method to implement:

```
void handle(Callback[] callbacks)
throws java.io.IOException, UnsupportedCallbackException;
```

Callback

The javax.security.auth.callback package contains the Callback **interface** as well as several implementations. LoginModules may pass an array of Callbacks directly to the *handle* method of a CallbackHandler.

Consult the various Callback APIs for more information on their use.

Authorization Classes

Upon successful authentication of a

`Subject`

, fine-grained access controls can be placed upon that

`Subject`

by invoking the Subject.doAs or Subject.doAsPrivileged methods. The permissions granted to that

`Subject`

are configured in a JAAS

`Policy`

.

Policy

This is an **abstract** class for representing the system-wide JAAS access control. As a default, JAAS provides a file-based subclass implementation, PolicyFile. Each

`Policy`

subclass must implement the following methods:

```
public abstract java.security.PermissionCollection getPermissions
                (Subject subject,
                java.security.CodeSource cs);
public abstract void refresh();
```

The

`getPermissions`

method returns the permissions granted to the specified

`Subject`

and

`CodeSource`

. The

`refresh`

method updates the runtime

`Policy`

with any modifications made since the last time it was loaded from its permanent store (a file or database, for example). The

`refresh`

method requires an

`AuthPermission("refreshPolicy")`

.

The following method retrieves the current runtime

`Policy`

object, and is protected with a security check that requires the caller to have an

`AuthPermission("getPolicy")`

.

```
public static Policy getPolicy();
```

The following example code demonstrates how a

`Policy`

object can be queried for the set of permissions granted to the specified

Subject

and

CodeSource

:

```
policy = Policy.getPolicy();
PermissionCollection perms = policy.getPermissions(subject, codeSource);
```

To set a new

Policy

object for the Java runtime, the

Policy.setPolicy

method may be used. This method requires the caller to have an

AuthPermission("setPolicy")

.

```
public static void setPolicy(Policy policy);
```

Policy File Sample Entries:

These examples are relevant only for the default PolicyFile implementation.

Each entry in the

Policy

is represented as a ***grant*** entry. Each ***grant*** entry specifies a codebase/code-signers/Principals triplet, as well as the Permissions granted to that triplet. Specifically, the permissions will be granted to any code downloaded from the specified *codebase* and signed by the specified *code signers*, so long as the

Subject

running that code has all of the specified *Principals* in its

Principal

set. Refer to the Subject.doAs examples to see how a

Subject

becomes associated with running code.

```
grant CodeBase ["URL"],
      Signedby ["signers"],
      Principal [Principal_Class] "Principal_Name",
      Principal ... {
    permission Permission_Class ["Target_Name"]
                                [, "Permission_Actions"]
                                [, signedBy "SignerName"];
};

// example grant entry
grant CodeBase "http://griffin.ibm.com", Signedby "davis",
      Principal com.ibm.security.auth.NTUserPrincipal "kent" {
    permission java.io.FilePermission "c:/kent/files/*", "read, write";
};
```

If no *Principal* information is specified in the JAAS

`Policy`

grant entry, a parsing exception will be thrown. However, grant entries that already exist in the regular Java 2 codesource-based policy file (and therefore have no *Principal* information) are still valid. In those cases, the *Principal* information is implied to be '*' (the grant entries applies to all Principals).

The CodeBase and Signedby components of the grant entry are optional in the JAAS

`Policy`

. If they are not present, then any codebase will match, and any signer (including unsigned code) will match.

In the example above, the ***grant*** entry specifies that code downloaded from ″http://griffin.ibm.com″, signed by ″davis″, and running as the NT user ″kent″, has one

`Permission`

. This

`Permission`

permits the processing code to read and write files in the directory ″c:\kent\files″.

Multiple Principals may be listed within one ***grant*** entry. The current

`Subject`

running the code must have all of the specified Principals in its

`Principal`

set to be granted the entry's Permissions.

```
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";
};
```

This entry grants any code running as both the NT user ″kent″ with the NT group identification number ″S-1-1-0″, permission to read and write files in ″c:\user\kent″, as well as permission to make socket connections to ″griffin.ibm.com″.

AuthPermission

This class encapsulates the basic permissions required for JAAS. An AuthPermission contains a name (also referred to as a ″target name″) but no actions list; you either have the named permission or you don't. In addition to inherited methods (from the

`Permission`

class), an

`AuthPermission`

has two public constructors:

```
public AuthPermission(String name);
public AuthPermission(String name, String actions);
```

The first constructor creates a new AuthPermission with the specified name. The second constructor also creates a new AuthPermission object with the specified name, but has an additional *actions* argument which is currently unused and are null. This constructor exists solely for the

`Policy`

object to instantiate new Permission objects. For most code, the first constructor is appropriate.

The AuthPermission object is used to guard access to the Policy, Subject, LoginContext, and Configuration objects. Refer to the AuthPermission Javadoc for the list of valid names that are supported.

PrivateCredentialPermission

This class protects access to a Subject's private credentials and provides one public constructor:

```
public PrivateCredentialPermission(String name, String actions);
```

Refer to the PrivateCredentialPermission Javadoc for more detailed information on this class.

Implementation

Note: Appendix A contains a sample **java.security** file that includes the static properties mentioned here.

Because there exists default values for JAAS providers and policy files, users need not statically (in the java.security file) nor dynamically (command line ***-D*** option) list their values in order to implement JAAS. Also, the default configuration and policy file providers may be replaced by a user-developed provider. Therefore this section is an attempt to explain the JAAS default providers and policy files as well as the properties that enable alternative providers.

Read the Default Policy File API and Default Configuration File API for more information than is summarized here.

Authentication Provider

The authentication provider, or configuration class, is statically set with

`login.configuration.provider=[class]`

in the *java.security* file. This provider creates the

`Configuration`

object.

For example:

```
login.configuration.provider=com.foo.Config
```

If the Security property

`login.configuration.provider`

is not found in java.security, then JAAS will set it to the default value:

`com.ibm.security.auth.login.ConfigFile`

.

If a security manager is set before the

`Configuration`

is created, then an

`AuthPermission("getLoginConfiguration")`

will be required to be granted.

There isn't a way to dynamically set the configuration provider on the command line.

Authentication Configuration File

The authentication configuration files may be statically set in *java.security* with

`login.config.url.`*`n`*`=[URL]`

, where *n* is a consecutively number integer starting with 1. The format is identical to the format for Java security policy files (policy.url.n=[URL]).

If the Security property

`policy.allowSystemProperty`

is set to ″true″ in java.security, then users can dynamically set policy files on the command line utilizing the ***-D*** option with this property:

`java.security.auth.login.config`

. The value may be a path or URL. For example (on NT):

```
... -Djava.security.auth.login.config=c:\config_policy\login.config ...
or
... -Djava.security.auth.login.config=file:c:/config_policy/login.config ...
```

Note: using double equal signs (==) on the command line allows a user to override all other policy files found.

If no configuration files can be found statically or dynamically, JAAS will try to load the configuration file from this default location:

`${`*`user.home`*`}\.java.login.config`

where *${user.home}* is a system dependent location.

Authorization Provider

The authorization provider, or JAAS Policy class, is statically set with

`auth.policy.provider=[class]`

in the *java.security* file. This provider creates the JAAS Subject-based

`Policy`

object.

For example:

```
auth.policy.provider=com.foo.Policy
```

If the Security property

`auth.policy.provider`

is not found in java.security, then JAAS will set it to the default value:
`com.ibm.security.auth.PolicyFile`

.

If a security manager is set before the
`Configuration`

is created, then an
`AuthPermission("getPolicy")`

will be required to be granted.

There isn't a way to dynamically set the authorization provider on the command line.

Authorization Policy File

The authorization policy files may be statically set in *java.security* with
`auth.policy.url.n=[URL]`

, where *n* is a consecutively number integer starting with 1. The format is identical to the format for Java security policy files (policy.url.n=[URL]).

If the Security property
`policy.allowSystemProperty`

is set to ″true″ in java.security, then users can dynamically set policy files on the command line utilizing the ***-D*** option with this property:
`java.security.auth.policy`

. The value may be a path or URL. For example (on NT):

```
... -Djava.security.auth.policy=c:\auth_policy\java.auth.policy ...
or
... -Djava.security.auth.policy=file:c:/auth_policy/java.auth.policy ...
```

Note: using double equal signs (==) on the command line allows a user to override all other policy files found.

There is not a default location to load an authorization policy from.

"Hello World", JAAS style!

Put on your dark sunglasses and favorite fedora hat, then grab an alto sax ... it's time to get **JAAS**-y! Yes, another ″Hello World!″ program. In this section, a program will be made available to test your JAAS installation.

Installation It is assumed that JAAS has been installed. For example, the JAAS JAR files have been copied to your Development Kit's extensions directory.

Retrieve Files Download theHelloWorld.tar to your test directory. Expand it using ″jar xvf HelloWorld.tar″.

Verify the contents of your test directory.

source files:

- HWLoginModule.java

- HWPrincipal.java
- HelloWorld.java

class files

- The source files have been precompiled for you into the classes directory.

policy files

- jaas.config
- java2.policy
- jaas.policy

Compile Source Files The three source files, ***HWLoginModule.java***, ***HWPrincipal.java*** and ***HelloWorld.java***, are already compiled and therefore do not need to be compiled.

If any of the source files are modified, then change to the test directory that they were saved to and enter:

```
javac -d .\classes *.java
```

The classpath needs the classes directory (.\classes) added to it in order to compile the classes.

Note:

```
HWLoginModule
```

and

```
HWPrincipal
```

are in the

```
com.ibm.security
```

package and will be created in the appropriate directory during compilation (>test_dir<\classes\com\ibm\security).

Examine Policy Files The configuration file, ***jaas.config***, contains one entry:

```
helloWorld {
   com.ibm.security.HWLoginModule required debug=true;
};
```

Only one

```
LoginModule
```

is supplied with the test case. When processing the ***HelloWorld*** application, experiment by changing the

```
LoginModuleControlFlag
```

(required, requisite, sufficient, optional) and deleting the debug flag. If more LoginModules are available for testing, then feel free to alter this configuration and experiment with multiple LoginModules.

```
HWLoginModule
```

will be discussed shortly.

The Java 2 policy file, ***java2.policy***, contains one permission block:

```
grant {
   permission javax.security.auth.AuthPermission "createLoginContext";
   permission javax.security.auth.AuthPermission "modifyPrincipals";
   permission javax.security.auth.AuthPermission "doAsPrivileged";
};
```

The three permissions are required because the ***HelloWorld*** application (1) creates a LoginContext object, (2) modifies the Principals of the the authenticated

`Subject`

and (3) calls the doAsPrivileged method of the

`Subject`

class.

The JAAS policy file, ***jaas.policy***, also contains one permission block:

```
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";
};
```

The three permissions are initially granted to an

`HWPrincipal`

named ***bob***. The actual Principal added to the authenticated

`Subject`

is the username used during the login process (more later).

Here's the action code from ***HelloWorld*** with the three system calls (the reason for the required permissions) in **bold**:

```
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("\nYour java.home property: "
                           +System.getProperty("java.home"));

        System.out.println("\nYour user.home property: "
                           +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("\nfoo.txt does ");
        if (!f.exists()) System.out.print("not ");
        System.out.println("exist in your current directory");

        System.out.println("\nOh, by the way ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignore
        }
        System.out.println("\n\nHello World!\n");
        return null;
    }
}, null);
```

When running the ***HelloWorld*** program, use various usernames and alter ***jaas.policy*** accordingly. There is no need to alter ***java2.policy***. Also, create a file called ***foo.txt*** in the test directory to test the last system call.

Examine Source Files The LoginModule,

`HWLoginModule`

, authenticates any user who enters the correct password (case sensitive): **Go JAAS**.

The ***HelloWorld*** application permits users three attempts to do so. When **Go JAAS** is correctly entered, an

`HWPrincipal`

with a name equal the the username is added to the authenticated

`Subject`

.

The Principal class,

`HWPrincipal`

, represents a Principal based on the username entered. It is this name that is important when granting permissions to authenticated Subjects.

The main application,

`HelloWorld`

, first creates a

`LoginContext`

based on a configuration entry with the name **helloWorld**. The configuration file has already been discussed. Callbacks are used to retrieve user input. Look at the

`MyCallbackHandler`

class located in the ***HelloWorld.java*** file to see this process.

```
LoginContext lc = null;
try {
    lc = new LoginContext("helloWorld", new MyCallbackHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}
```

The user enters a username/password (up to three times) and if **Go JAAS** is entered as the password, then the Subject is authenticated (

`HWLoginModule`

adds a

`HWPrincipal`

to the Subject).

As mentioned previously, work is then performed as the authenticated Subject.

Run HelloWorld Test

To run the ***HelloWorld*** program, first change to the test directory. The configuration and policy files will need to be loaded. See Implementation for the correct properties to set either in *java.security* or on the command line. The latter method will be discussed here.

The following command has been broken up into several lines for clarity. Enter as one continuous command.

```
java -Djava.security.manager=
     -Djava.security.auth.login.config=.\jaas.config
     -Djava.security.policy=.\java2.policy
     -Djava.security.auth.policy=.\jaas.policy
     HelloWorld
```

Note: the use of ″.\filename″ for the policy files is necessary because each user's test directory canonical path will vary. If desired, substitute ″.″ with the path to the test directory. For example, if the test directory is ″c:\test\hello″, then the first file is changed to:

```
     -Djava.security.auth.login.config=c:\test\hello\jaas.config
```

If the policy files are not found, a

`SecurityException`

will be thrown. Otherwise, information concerning your ***java.home*** and ***user.home*** properties will be displayed. Also, the existence of a file called ***foo.txt*** in your test directory will be checked. Finally, the ubiquitous ″Hello World″ message is displayed.

Having Fun With HelloWorld

Rerun ***HelloWorld*** as many times as you like. It has already been suggested to vary the username/passwords entered, change the configuration file entries, change the policy file permissions, and to even add (stack) additional LoginModules to the ***helloWorld*** configuration entry. You could add codebase fields to the policy files too.

Finally, try running the program without a SecurityManager to see how it works if you run into problems.

Appendix A: JAAS Settings in the java.security Security Properties File

Below is a copy of the

`java.security`

file that appears in every Java 2 installation. This file appears in the

`lib/security`

(

`lib\security`

on Windows) directory of the Java 2 runtime. Thus, if the Java 2 runtime is installed in a directory called

`jdk1.3`

, the file is

- `jdk1.3/lib/security/java.security`

  (Unix)
- `jdk1.3\lib\security\java.security`

  (Windows)

JAAS adds four new properties to

`java.security`

:

- Authentication Properties
  - `login.configuration.provider`
  - `login.policy.url.n`
- Authorization Properties
  - `auth.policy.provider`
  - `auth.policy.url.n`

The new JAAS properties are located at the end of this file:

```
#
# This is the "master security properties file".
#
# In this file, various security properties are set for use by
# java.security classes. This is where users can statically register
# Cryptography Package Providers ("providers" for short). The term
# "provider" refers to a package or set of packages that supply a
# concrete implementation of a subset of the cryptography aspects of
# the Java Security API. A provider may, for example, implement one or
# more digital signature algorithms or message digest algorithms.
#
# Each provider must implement a subclass of the Provider class.
# To register a provider in this master security properties file,
# specify the Provider subclass name and priority in the format
#
#    security.provider.n=className
#
# This declares a provider, and specifies its preference
# order n. The preference order is the order in which providers are
# searched for requested algorithms (when no specific provider is
# requested). The order is 1-based; 1 is the most preferred, followed
# by 2, and so on.
#
# className must specify the subclass of the Provider class whose
# constructor sets the values of various properties that are required
# for the Java Security API to look up the algorithms or other
# facilities implemented by the provider.
#
# There must be at least one provider specification in java.security.
# There is a default provider that comes standard with the JDK. It
# is called the "SUN" provider, and its Provider subclass
# named Sun appears in the sun.security.provider package. Thus, the
# "SUN" provider is registered via the following:
#
#    security.provider.1=sun.security.provider.Sun
#
# (The number 1 is used for the default provider.)
#
# Note: Statically registered Provider subclasses are instantiated
# when the system is initialized. Providers can be dynamically
# registered instead by calls to either the addProvider or
# insertProviderAt method in the Security class.

#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun

#
# Class to instantiate as the system Policy. This is the name of the class
```

```
# that will be used as the Policy object.
#
policy.provider=sun.security.provider.PolicyFile

# The default is to have a single system-wide policy file,
# and a policy file in the user's home directory.
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy

# whether or not we expand properties in the policy file
# if this is set to false, properties (${...}) will not be expanded in policy
# files.
policy.expandProperties=true

# whether or not we allow an extra policy to be passed on the command line
# with -Djava.security.policy=somefile. Comment out this line to disable
# this feature.
policy.allowSystemProperty=true

# whether or not we look into the IdentityScope for trusted Identities
# when encountering a 1.1 signed JAR file. If the identity is found
# and is trusted, we grant it AllPermission.
policy.ignoreIdentityScope=false

#
# Default keystore type.
#
keystore.type=jks

#
# Class to instantiate as the system scope:
#
system.scope=sun.security.provider.IdentityDatabase

############################################################################
#
# Java Authentication and Authorization Service (JAAS)
# properties and policy files:
#

# Class to instantiate as the system Configuration for authentication.
# This is the name of the class that will be used as the Authentication
# Configuration object.
#
login.configuration.provider=com.ibm.security.auth.login.ConfigFile

# The default is to have a system-wide login configuration file found in
# the user's home directory.  For multiple files, the format is similar to
# that of CodeSource-base policy files above, that is policy.url.n
login.config.url.1=file:${user.home}/.java.login.config

# Class to instantiate as the system Principal-based Authorization Policy.
# This is the name of the class that will be used as the Authorization
# Policy object.
#
auth.policy.provider=com.ibm.security.auth.PolicyFile

# The default is to have a system-wide Principal-based policy file found in
# the user's home directory.  For multiple files, the format is similar to
# that of CodeSource-base policy files above, that is policy.url.n and
# auth.policy.url.n
auth.policy.url.1=file:${user.home}/.java.auth.policy
```

Appendix B: Login Configuration Files

A login configuration file contains one or more

`LoginContext`

application names which have the following form:

```
Application {
    LoginModule Flag ModuleOptions;
    > more LoginModule entries <
    LoginModule Flag ModuleOptions;
};
```

Login configuration files are located using the

`login.config.url.n`

security property found in the

`java.security`

file. For more information about this property and the location of the

`java.security`

file, see Appendix A.

The *Flag* value controls the overall behavior as authentication proceeds down the stack. The following represents a description of the valid values for *Flag* and their respective semantics:

1. **Required** The

   `LoginModule`

   is required to succeed. If it succeeds or fails, authentication still continues to proceed down the

   `LoginModule`

   list.
2. **Requisite** The

   `LoginModule`

   is required to succeed. If it succeeds, authentication continues down the

   `LoginModule`

   list. If it fails, control immediately returns to the application (authentication does not proceed down the

   `LoginModule`

   list).
3. **Sufficient** The

   `LoginModule`

   is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the

   `LoginModule`

   list). If it fails, authentication continues down the

   `LoginModule`

list.

4. **Optional** The
   `LoginModule`
   is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the
   `LoginModule`
   list.

The overall authentication succeeds only if all *Required* and *Requisite* LoginModules succeed. If a *Sufficient*
`LoginModule`
is configured and succeeds, then only the *Required* and *Requisite* LoginModules prior to that *Sufficient*
`LoginModule`
need to have succeeded for the overall authentication to succeed. If no *Required* or *Requisite* LoginModules are configured for an application, then at least one *Sufficient* or *Optional*
`LoginModule`
must succeed.

Sample Configuration File:

```
/*  Sample Configuration File  */

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;
};
```

Note: the Flags are not case sensitive. *REQUISITE = requisite = Requisite.*

Login1 only has one LoginModule which is an instance of the class
`com.ibm.security.auth.module.SampleLoginModule`
. Therefore, a
`LoginContext`
associated with **Login1** will have a successful authentication if and only if its lone module successfully authenticates. The *Required* flag is trivial in this example; flag values have a relevant effect on authentication when two or more modules are present.

Login2 is easier to explain with a table.

Login2 Authentication Status									
Sample Login Module	required	pass	pass	pass	pass	fail	fail	fail	fail

Login2 Authentication Status									
NT Login Module	sufficient	pass	fail	fail	fail	pass	fail	fail	fail
Smart Card	requisite	*	pass	pass	fail	*	pass	pass	fail
Kerberos	optional	*	pass	fail	*	*	pass	fail	*
Overall Authentication		pass	pass	pass	fail	fail	fail	fail	fail

* = trivial value due to control returning to the application because a previous *REQUISITE* module failed or a previous *SUFFICIENT* module succeeded.

Appendix C: Authorization Policy File

In case there weren't enough examples of Principal-based JAAS Policy grant blocks above, here are some more.

```
// SAMPLE JAAS POLICY FILE:  java.auth.policy

// The following permissions are granted to Principal 'Pooh' and all codesource:

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";
};

// The following permissions are granted to Principal 'Pooh' AND 'Eyeore'
// and 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";
};

// The following permissions are granted to Principal 'Pooh' AND 'Eyeore' AND
// 'Piglet' and CodeSource from the c:\jaas directory signed by "kent" and "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";
};
```

Java 鑑別和授權服務範例

本主題包含 iSeries 伺服器上「Java 鑑別和授權服務 (JAAS)」的範例。

JAAS 範例有兩種：HelloWorld 及 SampleThreadSubjectLogin。請按一下這些鏈結來取得相關指令及原始程式碼。

在 iSeries 伺服器上搭配 Java 鑑別和授權服務來編譯及執行 HelloWorld:

本資訊說明如何在 iSeries 伺服器上編譯及執行「Java 鑑別和授權服務 (JAAS)」的 **HelloWorld**。

以下資訊取代 API 程式開發手冊的 **HelloWorld** 這一節。其原始程式碼、原則及配置檔，完全同於「API 程式開發手冊」。但仍有一些專屬於 iSeries 伺服器的事項。

1.

您應將下列來源檔放入自己的測試目錄中：

- HWLoginModule.java
- HWPrincipal.java
- HelloWorld.java

您必須將這些來源檔編譯到您的 ./classes 目錄中。

若要以適合 HTML 瀏覽器的格式來查看這些檔案的原始程式碼，請參閱 HTML 格式的 HelloWorld。

2.

需要編譯 HWLoginModule.java、HWPrincipal.java 及 HelloWorld.java 這三個來源檔。請在 iSeries 指令行執行下列指令 (每行一個指令)：

a.

strqsh

b.

cd *yourTestDir*

c.

```
javac -J-Djava.version=1.3
            -classpath /qibm/proddata/os400/java400/ext/jaas13.jar:.
            -d ./classes *.java
```

其中，*yourTestDir* 是您建立用來存放範例檔的目錄。classpath 必須附加類別目錄 (.\classes)，才能編譯這些類別。

附註：HWLoginModule 及 HWPrincipal 位於 com.ibm.security 套件中，會在編譯時建立於適當的目錄中 (\classes\com\ibm\security)。

3.

請在 iSeries 指令行執行下列指令 (每行一個指令)：

a.

strqsh

b. cd *yourTestDir*

其中，*yourTestDir* 是您建立用來存放範例檔的目錄。classpath 必須附加類別目錄 (.\classes)，才能編譯這些類別。

c. 您應將下列來源檔放入自己的測試目錄中：

- 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. 系統提示您輸入使用者名稱時，請輸入 **bob**。若是以安全管理程式來執行，則必須輸入使用者 **bob**，才能順利執行所有存取權。系統提示您輸入密碼時，請輸入 **Go JAAS**，必須區分大小寫並保留空格。

詳細資料：Java 鑑別和授權服務的 HelloWorld 如何運作: 本文件詳細探討「Java 鑑別和授權服務 (JAAS)」的 **HelloWorld** 如何運作。以下資訊取代 API 程式開發手冊的 **HelloWorld** 這一節。其原始程式碼、原則及配置檔，完全同於「API 程式開發手冊」。但仍有一些專屬於 iSeries 伺服器的事項。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

配置檔及原則檔

配置檔 **jaas.config** 包含一個項目：

```
helloWorld {
        com.ibm.security.HWLoginModule required debug=true;
};
```

此測試案例 (test case) 僅含一個 LoginModule。執行 HelloWorld 應用程式時，您可以透過變更 LoginModuleControlFlag (required、requisite、sufficient、optional) 及刪除除錯旗標來進行實驗。如有更多 LoginModule 可供測試，亦可變更這項配置，以多重 LoginModule 進行實驗。

Java 2 原則檔 **java2.policy** 包含一段許可權：

```
        grant {
                permission javax.security.auth.AuthPermission "createLoginContext";
                permission javax.security.auth.AuthPermission "modifyPrincipals";
                permission javax.security.auth.AuthPermission "doAsPrivileged";
    };
```

這三個許可權是必要的，因爲 HelloWorld 應用程式會執行下列動作：

1. 建立 LoginContext 物件。
2. 對通過鑑別的「主旨」，變更其「主體」。
3. 呼叫 Subject 類別的 doAsPrivileged 方法。

JAAS 原則檔 **jaas.policy** 亦包含一段許可權：

```
    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";
    };
```

最初會將這三個許可權授予名爲 bob 的 HWPrincipal。在通過鑑別的「主旨」中新增的實際「主體」，就是登入期間所用的使用者名稱。

以下是 HelloWorld 的動作碼，以粗體顯示三個系統呼叫 (需要許可權的理由)：

```
     Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
         public Object run() {
             System.out.println("\nYour java.home property: "
                                +System.getProperty("java.home"));

             System.out.println("\nYour user.home property: "
```

```
                                    +System.getProperty("user.home"));

            File f = new File("foo.txt");
            System.out.print("\nfoo.txt does ");
            if (!f.exists()) System.out.print("not ");
            System.out.println("exist in your current directory");

            System.out.println("\nOh, by the way ...");

            try {
                Thread.currentThread().sleep(2000);
            } catch (Exception e) {
                // ignore
            }
            System.out.println("\n\nHello World!\n");
            return null;
        }
    }, null);
```

執行 HelloWorld 程式時，請使用不同的使用者名稱，並且適當地變更 jaas.policy。java2.policy 應不須變更。另請在測試目錄中建立 foo.txt 這個檔案，測試最後一個系統呼叫，確認已將正確的存取層次授予此檔案。

檢查 HelloWorld 來源檔

HWLoginModule 這個 LoginModule 類別只是鑑別任何輸入正確密碼的使用者 (區分大小寫且保留空格)：

- **Go JAAS**

若以安全管理程式來執行，則必須輸入使用者 'bob'，才能順利執行所有存取權。

HelloWorld 應用程式允許使用者嘗試輸入三次。正確地輸入 Go JAAS 時，與使用者名稱同名的 HWPrincipal 物件，就會新增至通過鑑別的「主旨」。

HWPrincipal 這個 Principal 類別代表一個以輸入的使用者名稱爲基礎的「主體」。將許可權授予通過鑑別的「主旨」時，此名稱就顯得很重要。

主要的應用程式 HelloWorld 首先會根據名爲 helloWorld 的配置項目來建立 LoginContext。同時也利用回呼來擷取使用者輸入。請查看 HelloWorld.java 檔案中的 MyCallbackHandler 類別，瞭解此過程如何進行。以下摘錄自原始程式碼：

```
        LoginContext lc = null;
        try {
            lc = new LoginContext("helloWorld", new MyCallbackHandler());
        } catch (LoginException le) {
            le.printStackTrace();
            System.exit(-1);
        }
```

當使用者輸入使用者名稱及密碼時 (最多三次)，若輸入的密碼是 Go JAAS，「主旨」即通過鑑別 (HWLoginModule 會在「主旨」中新增 HWPrincipal)。然後，就會以通過鑑別的「主旨」爲身份來執行工作。

若找不到原則檔，則會丟出 **SecurityException**。否則，會顯示 java.home 及 user.home 內容的相關資訊。另外，也會在您的測試目錄中檢查 foo.txt 檔案是否存在。最後會出現 ″Hello World″ 這個眾所皆知的訊息。

享受使用 HelloWorld 的樂趣

您可以一直重新執行 HelloWorld。以下列出您可能會嘗試的一些動作：

- 改變輸入的使用者名稱及密碼
- 變更配置檔項目

- 變更原則檔許可權
- 將其他的 LoginModule 新增至 helloWorld 配置項目
- 將程式碼庫欄位新增至原則檔
- 執行程式時不使用 SecurityManager，瞭解發生問題時的運作情形

Java 鑑別和授權服務 SampleThreadSubjectLogin 指令:

來源檔

將 SampleThreadSubjectLogin.java 來源檔放入您自己的測試目錄中：您必須將此來源檔編譯到您的 ./classes 目錄中。

若要以適合 HTML 瀏覽器的格式來查看此檔案的原始程式碼，請參閱範例：JAAS SampleThreadSubjectLogin。

原則檔

如需 JAAS 原則檔的一般資訊，請參閱 API 程式開發手冊。SampleThreadSubjectLogin 範例有一些專用的原則檔：

- threadLogin.config
- threadJava2.policy
- threadJaas.policy

有關編譯及執行此範例的相關資訊，請參閱 SampleThreadSubjectLogin.java 開頭的註解。

鏈結集合

SampleThreadSubjectLogin.java

範例：JAAS SampleThreadSubjectLogin

API 程式開發手冊
This document was last updated March 17, 2000.

threadLogin.config

threadJava2.policy

threadJaas.policy

SampleThreadSubjectLogin.java

IBM Java 一般安全性服務 (JGSS)

「Java 一般安全性服務 (JGSS)」提供了鑑別及安全傳訊所需的同屬介面。在此介面之下，可以嵌入各種以私密金鑰、公開金鑰或其他安全技術爲基礎的安全機制。

JGSS 將基礎安全機制的複雜性與獨特性精簡化，而成爲標準的介面，因而在開發安全網路應用程式方面具有下列優點：

- 應用程式可以開發成爲單一抽象介面
- 應用程式不需任何修改，即可使用不同的安全機制

JGSS 定義了「一般安全性服務應用程式設計介面 (GSS-API)」的 Java 連結，是一種由 Internet Engineering Task Force (IETF) 制訂標準的加密 API，爲 X/Open Group 所採用。

IBM 的 JGSS 實作方式稱爲 IBM JGSS。IBM JGSS 爲 GSS-API 組織架構的實作方式，採用 Kerberos V5 作爲預設的基礎安全系統。它同時也具備「Java 鑑別和授權服務 (JAAS)」登入模組，可供您建立及使用 Kerberos 認證。此外，使用這些認證時，還可讓 JGSS 執行 JAAS 權限檢查。

IBM JGSS 包含原有的 iSeries JGSS 提供者、Java JGSS提供者及 Java 版本的 Kerberos 認證管理工具 (kinit、ktab 及 klist)。

註: 原有的 iSeries JGSS 提供者採用原有的「iSeries 網路鑑別服務 (NAS)」檔案庫。當使用原有的提供者時，您必須使用原有的 iSeries Kerberos 公用程式。如需詳細資訊，請參閱 JGSS 提供者。

Sun Microsystems 公司的 J2SDK 安全加強功能

Internet Engineering Task Force (IETF) RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1

IETF RFC 2853 Generic Security Service API Version 2: Java Bindings

The X/Open Group GSS-API Extensions for DCE

JGSS 概念

JGSS 作業由四個不同的階段組成，符合「一般安全性服務應用程式設計介面 (GSS-API)」的標準。

各階段如下所示：

1. 收集主體的認證。
2. 在通訊對等主體之間建立及設置安全環境定義。
3. 在對等主體之間交換安全訊息。
4. 清除及釋放資源。

此外，JGSS 也採用 Java Cryptographic Architecture，具有直接外掛不同安全機制的能力。

請利用下列鏈結來瞭解這些重要 JGSS 概念的進階說明。

- 主體及認證
- 建立環境定義
- 訊息保護及交換
- 資源清除及釋放
- 安全機制

主體及認證:

應用程式參與對等節點 JGSS 安全通訊的身分，就稱爲主體。 主體可能是眞實的使用者或無人式服務。 主體可以獲得特定安全機制的認證，在此機制之下作爲身分證明。

例如，使用 Kerberos 機制時，Kerberos 金鑰分送中心 (KDC) 簽發授予通行證的通行證 (TGT)，即爲主體的認證形式。 在多重機制環境下，一個 GSS-API 認證可以包含多個認證元素，每一個元素各代表一個基礎的機制認證。

GSS-API 標準並不規定主體如何獲得認證，GSS-API 實作方式通常不提供獲得認證的方法。 使用 GSS-API 之前，主體需取得認證；GSS-API 只是代替主體來查詢認證的安全機制。

IBM JGSS 包括 Java 版本的 Kerberos 認證管理工具 (第 302 頁的『com.ibm.security.krb5.internal.tools Class Kinit (無中文)』、第 304 頁的『com.ibm.security.krb5.internal.tools Class Ktab (無中文)』 及 第 301 頁的『com.ibm.security.krb5.internal.tools Class Klist (無中文)』)。此外，IBM JGSS 還提供選用性的 Kerberos 登入介面 (使用 JASS) 來加強標準的 GSS-API。Pure Java JGSS 提供者支援該選用登入介面，但原有的 iSeries 提供者則不支援。若需其餘相關資訊，請參閱下列主題：

- 取得 Kerberos 認證

- JGSS 提供者

com.ibm.security.krb5.internal.tools Class Klist (無中文):

This class can execute as a command-line tool to list entries in credential cache and key tab.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Klist
```

public class **Klist**
extends java.lang.Object

This class can execute as a command-line tool to list entries in credential cache and key tab.

Constructor summary

`Klist()`

Method summary

static void	`main(java.lang.String[] args)` The main program that can be invoked at command line.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor detail

Klist

```
public Klist()
```

Method detail

main

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

The main program that can be invoked at command line.

Usage: java com.ibm.security.krb5.tools.Klist [[-c] [-f] [-e] [-a]] [-k [-t] [-K]] [name]

Available options for credential caches:

- **-f** shows credentials flags
- **-e** shows the encryption type
- **-a** displays the address list

Available options for keytabs:

- **-t** shows keytab entry timestamps
- **-K** shows keytab entry DES keys

com.ibm.security.krb5.internal.tools Class Kinit (無中文):

Kinit tool for obtaining Kerberos v5 tickets.

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Kinit
```

public class **Kinit**
extends java.lang.Object

Kinit tool for obtaining Kerberos v5 tickets.

Constructor summary

`Kinit(java.lang.String[] args)`
Constructs a new Kinit object.

Method summary

`static void`	`main(java.lang.String[] args)` The main method is used to accept user command line input for ticket request.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor detail

Kinit

```
public Kinit(java.lang.String[] args)
      throws java.io.IOException,
             RealmException,
             KrbException
```

Constructs a new Kinit object.

Parameters:

`args` - array of ticket request options. Available options are: -f, -F, -p, -P, -c, -k, principal, password.

Throws:

java.io.IOException - if an I/O error occurs.
RealmException - if the Realm could not be instantiated.
KrbException - if error occurs during Kerberos operation.

Method detail

main

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

The main method is used to accept user command line input for ticket request.

Usage: java com.ibm.security.krb5.tools.Kinit [-f] [-F] [-p] [-P] [-k] [-c cache name] [principal] [password]

- **-f** forwardable
- **-F** not forwardable
- **-p** proxiable
- **-P** not proxiable
- **-c** cache name (i.e., FILE:d:\temp\mykrb5cc)
- **-k** use keytab
- **-t** keytab file name
- principal the principal name (i.e., qwedf qwedf@IBM.COM)
- password the principal's Kerberos password

Use java com.ibm.security.krb5.tools.Kinit -help to bring up help menu.

We currently only support file-based credentials cache. By default, a cache file named krb5cc_{user.name} would be generated at {user.home} directory to store the ticket obtained from KDC. For instance, on Windows NT, it could be c:\winnt\profiles\qwedf\krb5cc_qwedf, in which qwedf is the {user.name}, and c:\winnt\profile\qwedf is the {user.home}. {user.home} is obtained by Kerberos from Java system property ″user.home″. If in some case {user.home} is null (which barely happens), the cache file would be stored in the current directory that the program is running from. {user.name} is operating system's login username. It could be different from user's principal name. One user could have multiple principal names, but the primary principal of the credentials cache could only be one, which means one cache file could only store tickets for one specific user principal. If the user switches the principal name at the next Kinit, the cache file generated for the new ticket would overwrite the old cache file by default. To avoid overwriting, you need to specify a different directory or different cache file name when you request a new ticket.

Cache file location

There are several ways to define user specific cache file name and location, they are listed as follows in the order that Kerberos searches for:

1. **-c** option. Use java com.ibm.security.krb5.tools.Kinit -c FILE:<user specific directory and file name>. ″FILE:″ is the prefix to identify the credentials cache type. The default is file-based type.
2. Set Java system property ″KRB5CCNAME″ by using -DKRB5CCNAME=FILE:<user specific directory and file name> during runtime.
3. Set environment variable ″KRB5CCNAME″ at command prompt before the runtime. Different operating system has different way to set environment variables. For example, Windows uses set KRB5CCNAME=FILE:<user specific directory and file name>, while UNIX uses export KRB5CCNAME=FILE:<user specific directory and file name>. Note that Kerberos relies on system specific command to retrieve environment variable. The command used on UNIX is ″/usr/bin/env″.

KRB5CCNAME is case sensitive and is all upper case.

If KRB5CCNAME is not set as described above, a default cache file is used. The default cache is located in the following order:

1. /tmp/krb5cc_<uid> on Unix platforms, where <uid> is the user id of the user running the Kinit JVM
2. <user.home>/krb5cc_<user.name>, where <user.home> and <user.name> are the Java user.home and user.name properties, respectively
3. <user.home>/krb5cc (if <user.name> cannot be obtained from the JVM)

KDC Communication Timeout

Kinit communicates with the Key Distribution Center (KDC) to acquire a ticket-granting ticket, that is, the credential. This communication can be set to timeout if the KDC does not respond within a certain period. The timeout period can be set (in milliseconds) in the Kerberos configuration file in the libdefaults stanza (to be applicable to all KDCs) or in individual KDC stanzas. The default timeout value is 30 seconds.

com.ibm.security.krb5.internal.tools Class Ktab (無中文):

This class can execute as a command-line tool to help the user manage entires in the key table. Available functions include list/add/update/delete service key(s).

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Ktab
```

public class **Ktab**
extends java.lang.Object

This class can execute as a command-line tool to help the user manage entires in the key table. Available functions include list/add/update/delete service key(s).

Constructor summary

```
Ktab()
```

Method summary

`static void`	`main(java.lang.String[] args)` The main program that can be invoked at command line.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor detail

Ktab

```
public Ktab()
```

Method detail

main

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

The main program that can be invoked at command line.

Usage: java com.ibm.security.krb5.tools.Ktab <options>

Available options to Ktab:

- **-l** list the keytab name and entries

- **-a** <*principal name*>(<*password*>) add an entry to the keytab
- **-d** <*principal name*> delete an entry from the keytab
- **-k** <*keytab name*> specify keytab name and path with prefix FILE:
- **-help** display instructions

建立環境定義:

只要獲得安全認證，兩個通訊對等節點即可利用認證來建立安全環境定義。 雖然對等節點共同建立單一的聯合環境定義，但每一個對等節點各維護自己的本端環境定義複本。 建立環境定義牽涉到由起始端對等節點向接受端對等節點鑑別自己的身份。 另外，起始端也可能要求交互鑑別，此時接受端也要向起始端鑑別自己的身份。

當環境定義建立完成時，建立的環境定義就包含狀態資訊 (例如共用的加密金鑰)，然後兩個對等節點之間就能夠交換安全訊息。

訊息保護及交換:

環境定義建立完成之後，兩個對等節點即可開始交換安全訊息。 訊息發送端會呼叫本端的 GSS-API 實作方式將訊息編碼，確保訊息的完整性及機密性。 然後，應用程式再將產生的記號傳輸給對等節點。

對等節點的本端 GSS-API 實作方式會透過下列方式使用所建立環境定義的資訊：

- 驗證訊息的完整性
- 若訊息已加密，則進行解密

資源清除及釋放:

爲了釋放資源，JGSS 應用程式會刪除不再需要的環境定義。 雖然 JGSS 應用程式可以存取已刪除的環境定義，但嘗試利用它來交換訊息，仍然會導致異常。

安全機制:

GSS-API 係根據一或多個基礎安全機制的抽象組織架構所組成。 組織架構與基礎安全機制如何互動，視實作方式而定。

實作方式可歸納爲下列兩種：

- 組織架構與單一機制緊密結合，屬於極端的獨佔實作方式。這種實作方式會預先排除使用其他機制，甚至是相同機制的不同實作方式。
- 相對地，高度模組化的實作方式，用法最簡單且最有彈性。這種實作方式可直接在組織架構內，輕易地嵌入不同的安全機制及其實作方式。

IBM JGSS 即屬於後者。IBM JGSS 基於模組化實作方式，採用 Java Cryptographic Architecture (JCA) 定義的提供者組織架構，將任何基礎機制全部視爲 (JCA) 提供者。JGSS 提供者讓 JGSS 安全機制有了具體的實作方式。一個應用程式可以具體展現及使用多重機制。

即使一個提供者可能支援多重機制，只要透過 JGSS，即可輕易使用不同的安全機制。然而，就算有多個機制可用，GSS-API 仍無法讓兩個通訊對等節點去選擇某種機制。此時可從 Simple And Protected GSS-API Negotiating Mechanism (SPNEGO) 下手，這是一種可在兩個對等節點之間協議出實際機制的一種虛擬機制。但 IBM JGSS 不包括 SPNEGO 機制。

有關 SPNEGO 的資訊，請參閱 Internet Engineering Task Force (IETF) RFC 2478 The Simple and Protected GSS-API Negotiation Mechanism。

配置 iSeries 伺服器來使用 IBM JGSS

如何配置 iSeries 伺服器來使用 JGSS，要視伺服器上執行之 Java 2 Software Development Kit (J2SDK) 的版本而定。

配置 iSeries 伺服器來搭配 J2SDK 1.3 版使用 JGSS:

在 iSeries 伺服器上使用 Java 2 Software Development Kit (J2SDK) 1.3 版時，需要準備及配置伺服器才能使用 JGSS。預設配置會使用 Pure Java JGSS 提供者。

軟體需求

若要搭配 J2SDK 1.3 版來使用 JGSS，則伺服器必須安裝「Java 鑑別和授權服務 (JAAS) 1.3」。

配置伺服器來使用 JGSS

若要配置伺服器來搭配 J2SDK 1.3 版使用 JGSS，請在延伸套件目錄中新增 ibmjgssprovider.jar 檔案的符號鏈結。 ibmjgssprovider.jar 檔案包含 JGSS 類別及 Pure Java JGSS 提供者。新增符號鏈結即可讓延伸類別載入器載入 ibmjgssprovider.jar 檔案。

新增符號鏈結

若要新增符號鏈結，請在 iSeries 指令行鍵入下列指令 (全部同一行)，然後按 **ENTER** 鍵：

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssprovider.jar')
     NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/ibmjgssprovider.jar')
```

註: iSeries 伺服器的預設 Java 1.3 原則會授予適當的許可權給 JGSS。若打算建立自己的 Java.policy 檔案，請參閱 JVM 許可權， 內有應該授予 ibmjgssprovider.jar 的許可權清單。

變更 JGSS 提供者

配置伺服器來使用 JGSS 之後 (預設是使用 Pure Java 提供者)，可接著配置 JGSS 來使用原有的 iSeries JGSS 提供者。在配置 JGSS 來使用原有的提供者之後，即可輕易在這兩個提供者之間切換運用。若需其餘相關資訊，請參閱下列主題：

- JGSS 提供者
- 配置 JGSS 來使用原有的 iSeries JGSS 提供者

安全管理程式

若在啓用 Java 安全管理程式的情況下執行 IBM JGSS 應用程式，請參閱使用安全管理程式。

配置 JGSS 來使用原有的 iSeries JGSS 提供者:

IBM JGSS 預設使用 Pure Java 提供者。但是您也可以選擇使用原有的 iSeries JGSS 提供者。

有關不同提供者的資訊，請參閱 JGSS 提供者。

軟體需求

原有的 iSeries JGSS 提供者必須能夠存取 IBM Toolbox for Java 的類別。如需如何存取 IBM Toolbox for Java 的相關指示，請參閱第 307 頁的『讓原有的 iSeries JGSS 提供者能夠存取 IBM Toolbox for Java』。

請確定已配置網路鑑別服務。如需詳細資訊，請參閱網路鑑別服務。

指定原有的 iSeries JGSS 提供者

請確定已配置伺服器來使用 JGSS，如此才可以搭配 J2SDK 1.3 版來使用 iSeries JGSS 提供者。如需相關資訊，請參閱配置 iSeries 伺服器來搭配 J2SDK 1.3 版使用 JGSS。若使用 J2SDK 1.4 版或後續版本，則 JGSS 已完成配置。

註: 在下列指示中，${java.home} 代表伺服器上正在使用之 Java 版本的位置路徑。例如，如果使用 J2SDK 1.4 版，則 ${java.home} 爲 /QIBM/ProdData/Java400/jdk14。請記得將指令中的 ${java.home} 置換成 Java 起始目錄的實際路徑。

若要配置 JGSS 來使用原有的 iSeries JGSS 提供者，請完成下列作業：

新增符號鏈結

若要在延伸套件目錄中新增 ibmjgssiseriesprovider.jar 檔案的符號鏈結，請在 iSeries 指令行鍵入下列指令 (全部同一行)，然後按 **ENTER** 鍵：

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssiseriesprovider.jar')
NEWLNK('${java.home}/lib/ext/ibmjgssiseriesprovider.jar')
```

在延伸套件目錄中新增 ibmjgssiseriesprovider.jar 檔案的符號鏈結之後，延伸類別載入器就可載入此 JAR 檔。

在安全提供者清單中加入提供者

將原有的提供者新增至 java.security 檔案中的安全提供者清單。

1. 開啓 ${java.home}/lib/security/java.security 進行編輯。
2. 找出安全提供者清單。應該在 java.security 檔案開頭的附近，看起來就像：

   ```
   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. 在安全提供者清單的原始 Java 提供者之前，新增原有的 iSeries JGSS 提供者。換句話說，com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider 在清單裡的編號要小於 com.ibm.jgss.IBMJGSSProvider，別忘了也要更新 IBMJGSSProvider 的位置編號。例如：

   ```
   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
   ```

   請注意，IBMJGSSiSeriesProvider 應該成爲清單的第四個項目，而 IBMJGSSProvider 會變成第五個項目。另外，請檢查安全提供者清單的項目編號是否連續，且每一個項目的編號皆以 1 遞增。
4. 儲存並關閉 java.security 檔案。

讓原有的 iSeries JGSS 提供者能夠存取 IBM Toolbox for Java: 原有的 iSeries JGSS 提供者必須能夠存取 IBM Toolbox for Java 的類別。請使用下列其中一項方法，讓 IBM JGSS 能夠存取 Toolbox for Java jt400Native.jar 檔案：

- 在 Java 延伸套件目錄中放入 jt400Native.jar 的符號鏈結
- 在自己的延伸套件目錄中放入符號鏈結，然後將該目錄納入延伸套件目錄清單中

附註：

- 在 Java 延伸套件目錄中放入 jt400Native.jar 的符號鏈結，將強制 J2SDK 的所有使用者，一律使用此 jt400Native.jar 版本來執行。但是，若不同使用者需要不同版本的 IBM Toolbox for Java 類別，則不必這麼做。
- 在下列指示中，${java.home} 代表伺服器上正在使用之 Java 版本的位置路徑。例如，如果使用 J2SDK 1.4 版，則 ${java.home} 爲 /QIBM/ProdData/Java400/jdk14。請記得將指令中的 ${java.home} 置換成 Java 起始目錄的實際路徑。

在 Java 延伸套件目錄中放入符號鏈結

若要在 Java 延伸套件目錄中放入 jt400Native.jar 檔案的鏈結，請在 iSeries 指令行鍵入下列指令 (全部同一行)，然後按 **ENTER** 鍵：

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('${java.home}/lib/ext/jt400Native.jar')
```

在自己的延伸套件目錄中放入符號鏈結

若要將 jt400Native.jar 檔案鏈結至您自己的延伸套件目錄，請在 iSeries 指令行鍵入下列指令 (全部同一行)，然後按 **ENTER** 鍵：

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
    NEWLNK('<your extension directory>/jt400Native.jar')
```

請使用下列 Java 指令格式來呼叫您的程式：

```
java -Djava.ext.dirs=<your extension directory>:${java.home}/lib/ext:/QIBM/UserData/Java400/ext
```

配置 iSeries 伺服器來搭配 J2SDK 1.4 版或後續版本使用 JGSS:

在 iSeries 伺服器上使用 Java 2 Software Development Kit (J2SDK) 1.4 版或更新版本時，JGSS 已完成配置。預設配置會使用 Pure Java JGSS 提供者。

變更 JGSS 提供者

可以配置 JGSS 來使用原有的 iSeries JGSS 提供者，藉此取代 Pure Java JGSS 提供者。在配置 JGSS 來使用原有的提供者之後，即可輕易在這兩個提供者之間切換運用。若需其餘相關資訊，請參閱下列主題：

- JGSS 提供者
- 配置 JGSS 來使用原有的 iSeries JGSS 提供者

安全管理程式

若在啓用 Java 安全管理程式的情況下執行 JGSS 應用程式，請參閱使用安全管理程式。

JGSS 提供者:

IBM JGSS 包括原有的 iSeries JGSS 提供者及 Pure Java JGSS 提供者。您應根據應用程式的需求來選擇適用的提供者。

Pure Java JGSS 提供者具備下列功能：

- 確保應用程式的最大可攜性。
- 利用選用性的 JAAS Kerberos 登入介面。
- 相容於 Java Kerberos 認證管理工具。

原有的 iSeries JGSS 提供者具備下列功能：

- 使用原有的 iSeries Kerberos 檔案庫。
- 相容於 Qshell Kerberos 認證管理工具。
- JGSS 應用程式執行更快。

註: 這兩個 JGSS 提供者皆遵循 GSS-API 規格，所以彼此相容。換句話說，使用 Pure Java JGSS 提供者的應用程式與使用原有 iSeries JGSS 提供者的應用程式可以進行互用。

變更 JGSS 提供者

附註：如果伺服器執行 J2SDK 1.3 版，則在改爲使用原有的 iSeries JGSS 提供者之前，請先確定已配置伺服器來使用 JGSS。若需其餘相關資訊，請參閱下列主題：

- 配置 iSeries 伺服器來搭配 J2SDK 1.3 版使用 JGSS
- 配置 JGSS 來使用原有的 JGSS 提供者

透過下列其中一項方法，即可輕易變更 JGSS 提供者：

- 編輯 `${java.home}/lib/security/java.security` 中的安全提供者清單

  附註： ${java.home} 代表伺服器上正在使用之 Java 版本的位置路徑。例如，如果使用 J2SDK 1.3 版，則 ${java.home} 爲 /QIBM/ProdData/Java400/jdk13。
- 利用 GSSManager.addProviderAtFront() 或 GSSManager.addProviderAtEnd()，在 JGSS 應用程式中指定提供者名稱。如需詳細資訊，請參閱 GSSManager javadoc。

使用安全管理程式:

若在啓用 Java 安全管理程式的情況下執行 JGSS 應用程式，需要確定應用程式及 JGSS 具備必要的許可權。

JVM 許可權: 除了 JGSS 執行的存取控制檢查以外，Java 虛擬機器 (JVM) 在存取檔案、Java 內容、套件及 Socket 等各種資源時，也會執行權限檢查。

如需使用 JVM 許可權的資訊，請參閱 Permissions in the Java 2 SDK。

針對您使用 JGSS 的 JAAS 功能，或搭配安全管理程式使用 JGSS 時所需的許可權，下列清單有詳盡的指示：

- 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″

JAAS 許可權檢查:

當 JAAS 程式使用認證及存取服務時，IBM JGSS 就會進行執行時間許可權檢查。藉由將 Java 內容 avax.security.auth.useSubjectCredsOnly 設爲 False，可停用此選用性的 JAAS 功能。 此外，唯有在應用程式與安全管理程式搭配執行的情況下，JGSS 才會執行許可權檢查。

JGSS 會對現行存取控制環境定義下有效的 Java 原則進行許可權檢查。JGSS 會執行下列特定的許可權檢查：

- javax.security.auth.kerberos.DelegationPermission
- javax.security.auth.kerberos.ServicePermission

DelegationPermission 檢查

DelegationPermission 可讓安全原則控制通行證傳送的用法，以及 Kerberos 的 Proxy 功能。透過這些功能的運用，用戶端可讓某項服務代表用戶端。

DelegationPermission 可接受兩個引數，順序如下：

1. 子層主體，其爲用戶端授權下代表用戶端的服務主體名稱。
2. 用戶端要允許子層主體使用的服務名稱。

範例：使用 DelegationPermission 檢查

在下列範例中，superSecureServer 爲子層主體， krbtgt/REALM.IBM.COM@REALM.IBM.COM 爲我們允許 superSecureServer 用來代表用戶端的服務。在此情況下，此服務爲用戶端授予通行證的通行證，亦即 superSecureServer 可以代表用戶端來取得任何服務的通行證。

```
permission javax.security.auth.kerberos.DelegationPermission
 "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
 \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
```

上述範例中，DelegationPermission 將許可權授予用戶端， 讓用戶端從原本 superSecureServer 專用的「金鑰分送中心 (KDC)」取得授予通行證的通行證。當用戶端傳送新的授予通行證的通行證給 superSecureServer 之後，superSecureServer 就能夠代表用戶端來行動。

下列範例讓用戶端取得新的通行證，此通行證僅允許 superSecureServer 代表用戶端來存取 ftp 服務：

```
permission javax.security.auth.kerberos.DelegationPermission
 "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
 \"ftp/ftp.ibm.com@REALM.IBM.COM\"";
```

如需相關資訊，請參閱 Sun 網站中 J2SDK 文件的 javax.security.auth.kerberos.DelegationPermission 類別。

ServicePermission 檢查

ServicePermission 檢查可以在起始及接受環境定義方面限制認證的使用。環境定義起始端必須具有起始環境定義的許可權。同樣地，環境定義接受端亦必須具有接受環境定義的許可權。

範例：使用 ServicePermission 檢查

在下列範例中，會將許可權授予用戶端，讓用戶端起始 ftp 服務的環境定義：

```
permission javax.security.auth.kerberos.ServicePermission
 "ftp/host.ibm.com@REALM.IBM.COM", "initiate";
```

在下列範例中，會將許可權授予伺服器，讓伺服器存取及使用 ftp 服務的私密金鑰：

```
permission javax.security.auth.kerberos.ServicePermission
 "ftp/host.ibm.com@REALM.IBM.COM", "accept";
```

如需相關資訊，請參閱 Sun 網站中 J2SDK 文件的 javax.security.auth.kerberos.ServicePermission 類別。

執行 IBM JGSS 應用程式

「IBM Java 一般安全性服務 (JGSS) API 1.0」可阻絕不同基礎安全機制的複雜性及獨特性，讓安全應用程式不受干擾。JGSS 使用「Java 鑑別和授權服務 (JAAS)」及 IBM Java Cryptography Extension (JCE) 所提供的功能。

JGSS 功能包括：

- 身分鑑別
- 訊息完整性及機密性
- 選用的 JAAS Kerberos 登入介面及權限檢查

取得 Kerberos 認證及建立私密金鑰:

GSS-API 並未定義取得認證的方法。因此，IBM JGSS Kerberos 機制要求使用者必須取得 Kerberos 認證。本主題說明如何取得 Kerberos 認證及建立私密金鑰、如何使用 JAAS 來執行 Kerberos 登入及權限檢查，以及複查 Java 虛擬機器 (JVM) 所需的 JAAS 許可權清單。

可使用下列其中一種方法來取得認證：

- Kinit 及 Ktab 工具
- 選用的 JAAS Kerberos 登入介面

Kinit 及 Ktab 工具:

選擇採用何種 JGSS 提供者的同時，也就決定了使用何種工具來取得 Kerberos 認證及私密金鑰。

使用 Pure Java JGSS 提供者

若您使用的是 Java JGSS 提供者，請使用 IBM JGSS Kinit 及 Ktab 工具來取得認證及私密金鑰。Kinit 及 Ktab 工具會使用指令行介面，且提供類似其他版本所提供的選項。

- 您可以使用 Kinit 工具來取得 Kerberos 認證。此工具會聯繫「Kerberos 金鑰分送中心」來取得授予通行證的通行證 (TGT)。 TGT 可讓您存取其他 Kerberos 服務，包括其他採用 GSS-API 的服務。
- 伺服器可以使用 Ktab 工具來取得私密金鑰。JGSS 會將私密金鑰儲存在伺服器的金鑰表檔中。如需相關資訊，請參閱 Ktab Java 文件。

另外，應用程式亦可使用「JAAS 登入」介面來取得 TGT 及私密金鑰。如需詳細資訊，請參閱：

- 第 302 頁的『com.ibm.security.krb5.internal.tools Class Kinit (無中文)』
- 第 304 頁的『com.ibm.security.krb5.internal.tools Class Ktab (無中文)』
- JAAS 登入介面。

使用原有的 iSeries JGSS 提供者

若您使用的是原有的 iSeries JGSS 提供者，請使用 Qshell kinit 及 klist 公用程式。如需詳細資訊，請參閱 Kerberos 認證及金鑰表的公用程式。

JAAS Kerberos 登入介面:

IBM JGSS 具有「Java 鑑別和授權服務 (JAAS)」Kerberos 登入介面。藉由將 Java 內容 javax.security.auth.useSubjectCredsOnly 設爲 False，可停用此功能。

註: 雖然 Pure Java JGSS 提供者可以使用登入介面，但原有的 iSeries JGSS 提供者則無法使用。

如需 JAAS 的相關資訊，請參閱 Java 鑑別和授權服務。

JAAS 及 JVM 許可權

若您使用的是安全管理程式，請確定應用程式及 JGSS 具備必要的 JVM 和 JAAS 許可權。如需詳細資訊，請參閱使用安全管理程式。

JAAS 配置檔選項

登入介面必須要有 JAAS 配置檔來指定 com.ibm.security.auth.module.Krb5LoginModule 作爲要使用的登入模組。下表列出 Krb5LoginModule 支援的選項。請注意，選項不區分大小寫。

選項名稱	值	預設	說明
principal	<字串>	無；提示輸入。	Kerberos 主體名稱
credsType	initiator | acceptor | both	initiator	JGSS 認證類型
forwardable	truelfalse	false	是否獲得可轉送的授予通行證的通行證 (TGT)
proxiable	truelfalse	false	是否獲得可代理的 TGT
useCcache	<URL>	不使用 ccache	從指定的認證快取記憶體中擷取 TGT
useKeytab	<URL>	不使用金鑰表	從指定的金鑰表中擷取私密金鑰
useDefaultCcache	truelfalse	不使用預設的 ccache	從預設的認證快取記憶體中擷取 TGT
useDefaultKeytab	truelfalse	不使用預設金鑰表	從指定的金鑰表中擷取私密金鑰

如需簡單的 Krb5LoginModule 使用範例，請參閱 JAAS 登入配置檔範例。

選項不相容性

部份 Krb5LoginModule 選項 (主體名稱除外) 彼此不相容，這意謂著不可以同時加以指定。下表包含了相容與不相容的登入模組選項。

表中的指示符號說明兩個相關選項之間的關係：

- X = 不相容
- N/A = 不適用的組合
- 空白 = 相容

Krb5LoginModule 選項	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							

Krb5LoginModule 選項	credsType initiator	credsType acceptor	credsType both	forward	proxy	use Ccache	use Keytab	useDefault Ccache	useDefault Keytab
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	

主體名稱選項

指定主體名稱可以搭配其他任何選項。若未指定主體名稱，Krb5LoginModule 可能會提示使用者加以指定。Krb5LoginModule 是否提示使用者，視您指定的其他選項而定。

服務主體名稱格式

您必須採用下列其中一種格式來指定服務主體名稱：

- <service_name> (例如 superSecureServer)
- <service_name>@<host> (例如 superSecureServer@myhost)

後者的 <host> 指服務所在的機器的主電腦名稱。您可以 (但非必要) 使用完整的主電腦名稱。

註: JAAS 會將某些字元視爲區隔字元。在 JAAS 字串 (例如主體名稱) 中使用任何下列字元時，請用引號括住字元：

```
_  (底線)
:  (冒號)
/  (斜線)
\  (反斜線)
```

提示輸入主體名稱及密碼

您在 JAAS 配置檔中指定的選項，將決定 Krb5LoginModule 登入是否爲互動式。

- 非互動式登入完全不提示使用者輸入任何資訊
- 互動式登入會提示輸入主體名稱、密碼或兩者

非互動式登入

若將認證類型指定爲 initiator (`credsType=initiator`)，且執行下列其中一項動作時，將以非互動方式進行登入：

- 指定 useCcache 選項
- 將 useDefaultCcache 選項設爲 true

若將認證類型指定爲 acceptor 或 both (`credsType=acceptor` 或 `credsType=both`)，且執行下列其中一項動作時，也將以非互動方式進行登入：

- 指定 useKeytab 選項
- 將 useDefaultKeytab 選項設爲 true

互動式登入

其他配置會使登入模組提示使用者輸入主體名稱及密碼，藉以從 Kerberos KDC 取得 TGT。 若您指定 principal 選項，則登入模組僅提示輸入密碼。

互動式登入要求應用程式必須指定 com.ibm.security.auth.callback.Krb5CallbackHandler，作爲建立登入環境定義時的回呼處理常式。此回呼處理常式就負責提示使用者輸入資訊。

認證類型選項

若要求認證類型同時爲 initiator 與 acceptor (credsType=both)， Krb5LoginModule 會取得 TGT 與私密金鑰。登入模組會使用 TGT 來起始環境定義，使用私密金鑰來接受環境定義。 JAAS 配置檔須含足夠的資訊，登入模組才能夠獲得這兩種認證。

針對認證類型 acceptor 及 both，登入模組會採用一個服務主體。

配置檔及原則檔:

JGSS 與 JAAS 依附於數個配置檔及原則檔。請針對您的環境及應用程式來編輯這些檔案。 若不使用 JAAS 及 JGSS，請直接忽略 JAAS 配置檔及原則檔即可。

註: 在下列指示中，${java.home} 代表伺服器上正在使用之 Java 版本的位置路徑。例如，如果使用 J2SDK 1.4 版，則 ${java.home} 爲 /QIBM/ProdData/Java400/jdk14。請記得將內容設定中的 ${java.home} 置換成 Java 起始目錄的實際路徑。

Kerberos 配置檔

IBM JGSS 需要 Kerberos 配置檔。Kerberos 配置檔的預設名稱及位置，視作業系統而定。 JGSS 依下列順序來搜尋預設配置檔：

1. Java 內容 java.security.krb5.conf 參考到的檔案
2. ${java.home}/lib/security/krb5.conf
3. Microsoft® Windows 平台的 c:\winnt\krb5.ini
4. Solaris 平台的 /etc/krb5/krb5.conf
5. 其他 Unix 平台的 /etc/krb5.conf

JAAS 配置檔

使用 JAAS 登入機能需要 JAAS 配置檔。您可以設定下列其中一項內容來指定 JASS 配置檔：

- Java 系統內容 java.security.auth.login.config
- ${java.home}/lib/security/java.security 檔案中的安全內容 login.config.url.<integer>

如需相關資訊，請參閱 Sun Java Authentication and Authorization Service (JAAS) 網站。

JAAS 原則檔

使用預設原則實作方式時，JGSS 會在原則檔中記錄許可權，藉此授予 JAAS 許可權給實體。 您可以設定下列其中一項內容來指定 JASS 原則檔：

- Java 系統內容 java.security.policy
- ${java.home}/lib/security/java.security 檔案中的安全內容 policy.url.<integer>

如果使用 J2SDK 1.4 版或後續版次，則可選擇性地爲 JASS 指定單獨的原則檔。J2SDK 1.4 版及更新版本的預設原則提供者支援 JASS 所需的原則檔項目。

如需相關資訊，請參閱 Sun Java Authentication and Authorization Service (JAAS) 網站。

Java 主要安全內容檔

Java 虛擬機器 (JVM) 會用到許多重要的安全內容，這些安全內容是藉由編輯 Java 主要安全內容檔來設定。名爲 java.security 的這個檔案通常位於 iSeries 伺服器的 ${java.home}/lib/security 目錄中。

以下清單說明使用 JGSS 的幾個重要的安全內容。 請參考這些說明來編輯 java.security 檔案。

註: 適當時，說明中也會納入執行 JGSS 範例必要的適當值。

security.provider.<integer>：您要使用的 JGSS 提供者。它也會自行登記加密提供者類別。IBM JGSS 使用「IBM JCE 提供者」所提供的加密安全服務及其他安全服務。請完全依照下列範例來指定 sun.security.provider.Sun 及 com.ibm.crypto.provider.IBMJCE 套件：

```
security.provider.1=sun.security.provider.Sun
     security.provider.2=com.ibm.crypto.provider.IBMJCE
```

policy.provider：系統原則處理常式類別。例如：

```
policy.provider=sun.security.provider.PolicyFile
```

policy.url.<integer>：原則檔的 URL。若要使用原則檔範例，請加入類似以下的項目：

```
policy.url.1=file:/home/user/jgss/config/java.policy
```

login.configuration.provider：JAAS 登入配置處理常式類別，例如：

```
login.configuration.provider=com.ibm.security.auth.login.ConfigFile
```

auth.policy.provider：JAAS 主體存取控制原則處理常式類別，例如：

```
auth.policy.provider=com.ibm.security.auth.PolicyFile
```

login.config.url.<integer>：JAAS 登入配置檔的 URL。若要使用配置檔範例，請加入類似以下的項目：

```
login.config.url.1=file:/home/user/jgss/config/jaas.conf
```

auth.policy.url.<integer>：JAAS 原則檔的 URL。JASS 原則檔可以包含主體式與 CodeSource 式的建構。若要使用原則檔範例，請加入類似以下的項目：

```
auth.policy.url.1=file:/home/user/jgss/config/jaas.policy
```

認證快取記憶體及伺服器金鑰表

使用者主體會在認證快取記憶體中保存 Kerberos 認證。服務主體則在金鑰表中保存私密金鑰。在執行時間，IBM JGSS 會採取下列方式來尋找這些快取記憶體：

使用者認證快取記憶體

JGSS 依下列順序尋找使用者認證快取記憶體：

1. Java 內容 KRB5CCNAME 參考到的檔案
2. 環境變數 KRB5CCNAME 參考到的檔案
3. Unix 系統的 /tmp/krb5cc_<uid>
4. ${user.home}/krb5cc_${user.name}
5. ${user.home}/krb5cc (若無法取得 ${user.name})

伺服器金鑰表

JGSS 依下列順序尋找伺服器金鑰表檔案：

1. Java 內容 KRB5_KTNAME 的值
2. Kerberos 配置檔中 libdefaults 段落的 default_keytab_name 項目
3. ${user.home}/krb5_keytab

開發 IBM JGSS 應用程式

使用 JGSS 來開發安全應用程式。瞭解產生傳輸記號、建立 JGSS 物件、建立環境定義等相關資訊。

若要開發 JGSS 應用程式，您需要熟悉高階的 GSS-API 規格及 Java 連結規格。這些規格是 IBM JGSS 1.0 主要的依據及準則。如需相關資訊，請參閱下列鏈結。

- RFC 2743: Generic Security Service Application Programming Interface Version 2, Update 1
- RFC 2853: Generic Security Service API Version 2: Java Bindings

IBM JGSS 應用程式設計步驟:

開發 JGSS 應用程式需要多個步驟，包括使用傳輸記號、建立必要的 JGSS 物件、建立與刪除環境定義以及使用每則訊息服務。

JGSS 應用程式的運作必須遵循「一般安全性服務應用程式設計介面 (GSS-API)」作業模型。 有關 JGSS 作業的重要概念，請參閱 JGSS 概念。

JGSS 傳輸記號

有些重要的 JGSS 作業會產生 Java 位元組陣列形式的記號。應用程式應該負責在 JGSS 對等節點之間轉遞這些記號。JGSS 完全不限制應用程式採用什麼通訊協定來傳輸記號。應用程式可同時傳輸 JGSS 記號及其他應用程式 (亦即，非 JGSS) 資料。 但 JGSS 作業僅接受及使用 JGSS 專用的記號。

JGSS 應用程式的作業順序

您必須依下列順序來使用特定的程式設計建構，才能使用 JGSS 作業。每一個步驟都適用於起始端與接收端。

註: 以下資訊提供程式碼範例的片段，說明如何使用高階 JGSS API，並且假設應用程式匯入 org.ietf.jgss 套件。雖然許多高階 API 皆爲超載形式，但程式碼片段中僅示範這些方法最常見的用法。當然，您應該使用最符合實際需求的 API 方法。

1. 建立 GSSManager

   GSSManager 實例扮演 Factory 來建立其他 JGSS 物件實例。
2. 建立 GSSName

   GSSName 代表 JGSS 主體的身分。將 GSSName 指定爲空值時，有些 JGSS 作業可以尋找並使用預設主體。
3. 建立 GSSCredential

   GSSCredential 內含主體的機制專用認證。
4. 建立 GSSContext

   GSSContext 用於建立環境定義及後續的每則訊息服務。
5. 在環境定義上選取選用性的服務

   應用程式必須明確地要求選用的服務，例如交互鑑別。
6. 建立環境定義

起始端會向接收端鑑別自己的身份。然而，若要求交互鑑別，則接收端也會轉而向起始端鑑別自己的身份。

7. 使用每則訊息服務

   起始端及接收端會透過已建立的環境定義來交換安全訊息。

8. 刪除環境定義

   應用程式會刪除不再需要的環境定義。

建立 GSSManager:

GSSManager 抽象類別可作爲 Factory 來建立 JGSS 物件。

GSSManager 抽象類別會建立下列物件：

- GSSName
- GSSCredential
- GSSContext

GSSManager 亦提供了方法，可決定受支援的安全機制和名稱類型，也可用來指定 JGSS 提供者。 請使用 GSSManager getInstance static 方法來建立預設 GSSManager 的實例：

```
GSSManager manager = GSSManager.getInstance();
```

建立 GSSName:

GSSName 代表 GSS-API 主體的身分。GSSName 可能包含主體的許多不同表示法，各代表一種支援的基礎機制。 只含一個名稱表示法的 GSSName，稱爲「機制名稱 (MN)」。

GSSManager 有數個超載方法，透過字串或連續的位元組陣列，即可用來建立 GSSName。這些方法會根據指定的名稱類型來解譯字串或位元組陣列。通常會使用 GSSName 位元組陣列方法來重新建構匯出的名稱。匯出的名稱通常爲 GSSName.NT_EXPORT_NAME 類型的機制名稱。有些方法可讓您指定用來建立名稱的安全機制。

範例：使用 GSSName

下列基本的程式碼片段將顯示如何使用 GSSName。

註: 請將 Kerberos 服務名稱字串指定爲 <service> 或 <service@host>，其中 <service> 爲服務的名稱，而 <host> 爲執行服務所在機器的主電腦名稱。您可以 (但非必要) 使用完整的主電腦名稱。若字串中省略 @<host> 部份，GSSName 就會採用本端主電腦名稱。

```
// Create GSSName for user foo.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);

     // Create a Kerberos V5 mechanism name for user foo.
Oid krb5Mech = new Oid("1.2.840.113554.1.2.2");
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME, krb5Mech);

     // Create a mechanism name from a non-mechanism name by using the GSSName
     // canonicalize method.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);
     GSSName fooKrb5Name = fooName.canonicalize(krb5Mech);
```

建立 GSSCredential:

GSSCredential 含有代表主體來建立環境定義時所有必要的加密資訊，亦可能含有多重機制的認證資訊。

GSSManager 有三個建立認證的方法。其中兩個方法接受的參數包括 GSSName、認證有效期限、一或多種用來取得認證的機制，以及認證用途。第三個方法僅接受一個用途參數，其他參數則採用預設值。若指定空值機制，則也會使用預設機制。若指定空值機制陣列，方法就會傳回預設機制的認證。

註: 因爲 IBM JGSS 僅支援 Kerberos V5 機制，因此會以此爲預設機制。

應用程式一次只能建立三種認證的其中之一種 (*起始*、*接受* 或*起始與接受*)。

- 環境定義起始端建立*起始*認證
- 接收端建立*接受*認證
- 接收端同時扮演起始端來建立*起始與接受*認證。

範例：取得認證

下列範例說明如何取得起始端的預設認證：

```
GSSCredentials fooCreds = manager.createCredentials(GSSCredential.INITIATE)
```

下列範例可取得起始端 *foo* 的 Kerberos V5 認證，其具有預設的有效期限：

```
GSSCredential fooCreds = manager.createCredential(fooName, GSSCredential.DEFAULT_LIFETIME,
                                                  krb5Mech,GSSCredential.INITIATE);
```

下列範例會取得一個全部都爲預設值的接收端認證：

```
GSSCredential serverCreds = manager.createCredential(null, GSSCredential.DEFAULT_LIFETIME,
                                                     (Oid)null, GSSCredential.ACCEPT);
```

建立 GSSContext:

IBM JGSS 支援由 GSSManager 爲建立環境定義提供的兩個方法。

這些方法是：

- 供環境定義起始端使用的方法
- 接收端使用的方法

註: GSSManager 也提供第三種方法來建立環境定義，可以重建先前匯出的環境定義。然而，因爲 IBM JGSS Kerberos V5 機制不支援使用匯出的環境定義，所以 IBM JGSS 不支援此方法。

應用程式無法使用起始端環境定義來接受環境定義，亦無法使用接收端環境定義來起始環境定義。這兩種建立環境定義的支援方法，都必須先輸入認證才能使用。當認證爲空值時，JGSS 會採用預設認證。

範例：使用 GSSContext

在下列範例中，將建立一個環境定義，主體 (foo) 可透過此環境定義， 與主電腦 (securityCentral) 上的對等節點 (superSecureServer) 起始一個環境定義。範例中指定 superSecureServer@securityCentral 爲對等節點。建立的環境定義採用預設的有效期限：

```
GSSName serverName = manager.createName("superSecureServer@securityCentral",
                                        GSSName.NT_HOSTBASED_SERVICE, krb5Mech);
GSSContext fooContext = manager.createContext(serverName, krb5Mech, fooCreds,
                                              GSSCredential.DEFAULT_LIFETIME);
```

下列範例中將建立 superSecureServer 的環境定義，藉以接受任何對等節點所起始的環境定義：

```
GSSContext serverAcceptorContext = manager.createContext(serverCreds);
```

請注意，應用程式可以建立及同時使用這兩種環境定義。

要求選用的安全服務:

應用程式可以要求任何選用的安全服務。IBM JGSS 支援幾種服務。

所支援的選用服務是：

- 委派
- 交互鑑別
- 重播偵測
- 失序偵測
- 可用的每則訊息機密性
- 可用的每則訊息完整性

爲了取得選用的服務，應用程式必須明確地在環境定義上使用適當的方法來要求。僅起始端才可以要求這些選用的服務。起始端必須於環境定義開始建立之前提出此要求。

如需選用服務的相關資訊，請參閱 Internet Engineering Task Force (IETF) RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1 的 Optional Service Support。

範例：要求選用的服務

在下列範例中，環境定義 (fooContext) 會要求啓用交互鑑別及委派服務：

```
fooContext.requestMutualAuth(true);
     fooContext.requestCredDeleg(true);
```

建立環境定義:

兩個通訊對等節點必須建立安全環境定義，方能使用其間的每則訊息服務。

起始端會在環境定義上呼叫 initSecContext()，並將記號傳回起始端應用程式。起始端應用程式會將環境定義記號傳輸至接收端應用程式。接收端會在環境定義上呼叫 acceptSecContext()，指定來自起始端的環境定義記號。視基礎的機制及起始端選取的選用服務而定，acceptSecContext() 可能會產生一個必須由接收端應用程式轉遞至起始端應用程式的記號。然後，起始端應用程式再使用收到的記號，再次呼叫 initSecContext()。

應用程式可以多次呼叫 GSSContext.initSecContext() 及 GSSContext.acceptSecContext()。在建立環境定義期間，應用程式與對等節點亦可交換多個記號。因此，建立環境定義最常見的方法，就是利用迴圈來呼叫 GSSContext.initSecContext() 或 GSSContext.acceptSecContext()，直到應用程式建立環境定義爲止。

範例：建立環境定義

下列範例說明起始端 (foo) 如何建立環境定義：

```
byte array[] inToken = null; // The input token is null for the first call
int inTokenLen = 0;

do {
    byte[] outToken = fooContext.initSecContext(inToken, 0, inTokenLen);

    if (outToken != null) {
        send(outToken); // transport token to acceptor
    }

    if( !fooContext.isEstablished()) {
```

```
            inToken = receive(); // receive token from acceptor
              inTokenLen = inToken.length;
        }
    } while (!fooContext.isEstablished());
```

下列範例說明接收端如何建立環境定義：

```
// The acceptor code for establishing context may be the following:
do {
    byte[] inToken = receive(); // receive token from initiator
    byte[] outToken =
        serverAcceptorContext.acceptSecContext(inToken, 0, inToken.length);

    if (outToken != null) {
        send(outToken); // transport token to initiator
    }
} while (!serverAcceptorContext.isEstablished());
```

使用每則訊息服務:

建立安全環境定義之後，兩個通訊對等節點即可透過已建立的環境定義來交換安全訊息。

對等節點的任一端在環境定義建立時不論是充當起始端或接收端，皆可發出安全訊息。爲了保護訊息，IBM JGSS 會在訊息上計算加密訊息完整碼 (MIC)。另外，IBM JGSS 亦可使 Kerberos V5 機制對訊息進行加密，以進一步確保私密性。

傳送訊息

IBM JGSS 提供兩組保護訊息的方法：wrap() 及 getMIC()。

使用 wrap()

wrap 方法會執行下列動作：

- 計算 MIC
- 進行訊息加密 (選用)
- 傳回記號

呼叫端應用程式會使用 MessageProp 類別並搭配 GSSContext，指定是否進行訊息加密。

傳回的記號包含訊息的 MIC 及文字。訊息文字可能是密碼文字 (指加密訊息) 或原始純文字 (指未加密的訊息)。

使用 getMIC()

getMIC 方法會執行下列動作，但無法進行訊息加密：

- 計算 MIC
- 傳回記號

傳回的記號只含計算的 MIC，不含原始訊息。 所以，除了將 MIC 記號傳輸至對等節點之外，還必須設法使對等節點能察覺原始訊息，如此才能驗證 MIC。

範例：使用每則訊息服務來傳送訊息

下列範例顯示一個對等節點 (foo) 如何包裝訊息來遞送至另一個對等節點 (superSecureServer)：

```
byte[] message = "Ready to roll!".getBytes();
MessageProp mprop = new MessageProp(true); // foo wants the message encrypted
byte[] wrappedMessage =
```

```
    fooContext.wrap(message, 0, message.length, mprop);
send(wrappedMessage); // transfer the wrapped message to superSecureServer

// This is how superSecureServer may obtain a MIC for delivery to foo:
byte[] message = "You bet!".getBytes();
MessageProp mprop = null;  // superSecureServer is content with
                           // the default quality of protection

byte[] mic =
    serverAcceptorContext.getMIC(message, 0, message.length, mprop);
send(mic);
// send the MIC to foo. foo also needs the original message to verify the MIC
```

接收訊息

包裝訊息的接收端使用 unwrap() 進行訊息解碼。unwrap 方法會執行下列動作：

- 驗證訊息內含的加密 MIC
- 傳回傳送者原先計算 MIC 的原始訊息

若傳送端將訊息加密，unwrap() 會先將訊息解密再驗證 MIC，然後傳回原始純文字訊息。 MIC 記號的接收端使用 verifyMIC() 來驗證給定訊息的 MIC。

對等節點應用程式使用各自的通訊協定，彼此傳遞 JGSS 環境定義及訊息記號。對等節點應用程式亦須定義通訊協定，用以判斷記號是 MIC 或包裝訊息。舉例來說，這種通訊協定可能像「簡易鑑別與安全層 (SASL)」應用程式使用的通訊協定，簡單且固定不變。根據 SASL 通訊協定的規格，建立環境定義之後最先傳送每則訊息 (包裝) 記號的對等節點，即爲環境定義接收端。

如需相關資訊，請參閱 Simple Authentication and Security Layer (SASL)。

範例：使用每則訊息服務來接收訊息

下列範例顯示一個對等節點 (superSecureServer) 如何將來自另一個對等節點 (foo) 的包裝記號解除包裝：

```
MessageProp mprop = new MessageProp(false);

byte[] plaintextFromFoo =
    serverAcceptorContext.unwrap(wrappedTokenFromFoo, 0,
                                 wrappedTokenFromFoo.length, mprop);

// superSecureServer can now examine mprop to determine the message properties
// (such as whether the message was encrypted) applied by foo.

// foo verifies the MIC received from superSecureServer:

MessageProp mprop = new MessageProp(false);
fooContext.verifyMIC(micFromFoo, 0, micFromFoo.length, messageFromFoo, 0,
                     messageFromFoo.length, mprop);

// foo can now examine mprop to determine the message properties applied by

// superSecureServer. In particular, it can assert that the message was not

// encrypted since getMIC should never encrypt a message.
```

刪除環境定義:

對等節點會刪除不再需要的環境定義。在 JGSS 作業中，每一個對等節點可以在不必通知其對等節點的情況下，自己決定刪除環境定義的時間。

JGSS 不會對刪除環境定義記號進行定義。刪除環境定義時，對等節點會呼叫 GSSContext 物件的 dispose 方法，釋放環境定義佔用的任何資源。在應用程式將已刪除的 GSSContext 物件設爲空值之前，仍然可以存取該物件。只不過，嘗試使用已刪除的 (但仍可存取的) 環境定義時，會造成一則異常訊息的產生。

使用 JAAS 來處理 JGSS 應用程式:

IBM JGSS 提供選用性的 JAAS 登入機能，可讓應用程式使用 JAAS 取得認證。當 JAAS 登入機能將主體認證及私密金鑰儲存於 JAAS 登入環境定義的主旨物件之後， JGSS 即可從主旨中擷取認證。

根據預設，JGSS 會從主旨中擷取認證及私密金鑰。 藉由將 Java 內容 javax.security.auth.useSubjectCredsOnly 設爲 False，可停用此功能。

註: 雖然 Pure Java JGSS 提供者可以使用登入介面，但原有的 iSeries JGSS 提供者則無法使用。

有關 JAAS 功能的資訊，請參閱取得 Kerberos 認證及私密金鑰。

若要使用 JAAS 登入機能，應用程式必須在下列方面遵循 JAAS 程式設計模型：

- 建立 JAAS 登入環境定義
- 在 JAAS Subject.doAs 建構範圍內操作

下列程式碼片段說明於 JAAS Subject.doAs 建構範圍內操作的概念：

```
static class JGSSOperations implements PrivilegedExceptionAction {
    public JGSSOperations() {}
    public Object run () throws GSSException {
        // JGSS application code goes/runs here
    }
}

public static void main(String args[]) throws Exception {
    // Create a login context that will use the Kerberos
    // callback handler
    // com.ibm.security.auth.callback.Krb5CallbackHandler

    // There must be a JAAS configuration for "JGSSClient"
    LoginContext loginContext =
        new LoginContext("JGSSClient", new Krb5CallabackHandler());
        loginContext.login();

    // Run the entire JGSS application in JAAS privileged mode
    Subject.doAsPrivileged(loginContext.getSubject(),
                           new JGSSOperations(), null);
}
```

除錯

嘗試識別 JGSS 問題時，請使用 JGSS 除錯功能來產生實用的分類訊息。

您可以設定 Java 內容 com.ibm.security.jgss.debug 的適當值，來開啓一或多個種類。使用逗點來分隔種類名稱，即可啓用多個種類。

除錯種類包括：

種類	說明
help	列出除錯種類
all	開啓所有種類的除錯
off	完全關閉除錯

種類	說明
app	應用程式除錯 (預設值)
ctx	環境定義作業除錯
cred	認證 (包括名稱) 作業
marsh	排列記號
mic	MIC 作業
prov	提供者作業
qop	QOP 作業
unmarsh	取消排列記號
unwrap	解除包裝作業
wrap	包裝作業

JGSS 除錯類別

若要以程式化的方式進行 JGSS 應用程式除錯，請使用 IBM JGSS 組織架構中的除錯類別。應用程式可以使用除錯類別來開啓及關閉除錯種類，並且顯示作用中種類的除錯資訊。

預設除錯建構子會讀取 Java 內容 com.ibm.security.jgss.debug，以判斷需要啓用 (開啓) 哪些種類。

範例：應用程式種類的除錯

下列範例說明如何要求應用程式種類的除錯資訊：

```
import com.ibm.security.jgss.debug;

Debug debug = new Debug(); // Gets categories from Java property

// Lots of work required to set up someBuffer. Test that the
// category is on before setting up for debugging.

if (debug.on(Debug.OPTS_CAT_APPLICATION)) {
    // Fill someBuffer with data.
 debug.out(Debug.OPTS_CAT_APPLICATION, someBuffer);
    // someBuffer may be a byte array or a String.
```

範例：IBM Java 一般安全性服務 (JGSS)

「IBM Java 一般安全性服務 (JGSS)」範例檔包括用戶端及伺服器程式、配置檔、原則檔及 javadoc 參考資訊。使用範例程式來測試及驗證 JGSS 設定。

您可以檢視 HTML 版的範例，或下載範例程式的 javadoc 資訊和原始程式碼。下載範例之後，您就可以檢視 javadoc 參考資訊、查驗程式碼、編輯配置檔和原則檔，並可編譯和執行範例程式：

- 檢視範例的 HTML 版
- 下載及檢視範例的 javadoc 資訊
- 下載及執行範例程式

範例程式的說明

JGSS 範例包括四個程式：

- 非 JAAS 伺服器
- 非 JAAS 用戶端

- JAAS 伺服器
- JAAS 用戶端

JAAS 版的程式與對應的非 JAAS 程式可完全互用。因此，您可以對非 JAAS 伺服器執行 JAAS 用戶端，亦可對 JAAS 伺服器執行非 JAAS 用戶端。

註: 執行範例時，可以指定一或多個選用性的 Java 內容，包括配置檔及原則檔的名稱、JGSS 除錯選項及安全管理程式。另外，亦可選擇開啓或關閉 JAAS 功能。

您可以在單伺服器或雙伺服器的配置下執行範例。單伺服器配置，由相互通訊的用戶端與一部主要伺服器所構成。雙伺服器配置由一部主要伺服器和一部次要伺服器組成，相對於次要伺服器而言，主要伺服器扮演起始端 (或用戶端) 的角色。

使用雙伺服器配置時，首先用戶端會起始環境定義，然後與主要伺服器交換安全訊息。接著，用戶端將認證委派給主要伺服器。然後，主要伺服器會代表用戶端，使用這些認證來起始環境定義，並且與次要伺服器交換安全訊息。雙伺服器配置中，主要伺服器本身亦可同時扮演自己的用戶端。在此情況下，主要伺服器會使用自己的認證來起始環境定義，並且與次要伺服器交換訊息。

您也可以對主要伺服器同步執行多個用戶端，數量不限。雖然可以直接對次要伺服器執行用戶端，但次要伺服器卻無法使用委派的認證，或以自己的認證來當作起始端執行。

檢視 IBM JGSS 範例:

「IBM Java 一般安全性服務 (JGSS)」範例檔包括用戶端及伺服器程式、配置檔、原則檔及 javadoc 參考資訊。請利用下列鏈結來檢視 HTML 版的 JGSS 範例。

檢視範例程式

請利用下列鏈結來檢視 HTML 版的 JGSS 範例程式：

- 非 JAAS 用戶端程式範例
- 非 JAAS 伺服器程式範例
- 支援 JAAS 的用戶端程式範例
- 支援 JAAS 的伺服器程式範例

檢視配置檔及原則檔範例

請利用下列鏈結來檢視 HTML 版的 JGSS 配置檔及原則檔：

- Kerberos 配置檔
- JAAS 配置檔
- JAAS 原則檔
- Java 原則檔

如需相關資訊，請參閱下列主題：

- 範例程式的說明
- 下載及執行範例程式

範例：Kerberos 配置檔:

如需使用配置檔範例的相關資訊，請參閱下載及執行 IBM JGSS 範例。

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
# --------------------------------------------------------------------------------
# Kerberos configuration file for running the JGSS sample applications.
# Modify the entries to suit your environment.
#--------------------------------------------------------------------------------

[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
```

鏈結集合

下載及執行 IBM JGSS 範例

本主題包含有關下載及執行範例 javadoc 資訊的指示。

程式碼範例免責聲明

範例：JAAS 登入配置檔:

如需使用配置檔範例的相關資訊，請參閱下載及執行 IBM JGSS 範例。

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/**
 * --------------------------------------------------------------------------------
 * JAAS Login Configuration for the JGSS samples.
 * --------------------------------------------------------------------------------
 *
 * Code example disclaimer
 * IBM grants you a nonexclusive copyright license to use all programming code
 * examples from which you can generate similar function tailored to your own
 * specific needs.
 * All sample code is provided by IBM for illustrative purposes only.
 * These examples have not been thoroughly tested under all conditions.
 * IBM, therefore, cannot guarantee or imply reliability, serviceability, or
 * function of these programs.
 * All programs contained herein are provided to you "AS IS" without any
 * warranties of any kind.
 * The implied warranties of non-infringement, merchantability and fitness
 * for a particular purpose are expressly disclaimed.
 *
 *
 * Supported options:
 *     principal=<string>
 *     credsType=initiator|acceptor|both (default=initiator)
 *     forwardable=true|false (default=false)
 *     proxiable=true|false (default=false)
 *     useCcache=<URL_string>
```

```
 *     useKeytab=<URL_string>
 *     useDefaultCcache=true|false (default=false)
 *     useDefaultKeytab=true|false (default=false)
 *     noAddress=true|false (default=false)
 *
 * Default realm (which is obtained from the Kerberos config file) is
 * used if the principal specified does not include a realm component.
 */

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;
};
```

鏈結集合

下載及執行 IBM JGSS 範例

本主題包含有關下載及執行範例 javadoc 資訊的指示。

程式碼範例免責聲明

範例：JAAS 原則檔:

如需使用原則檔範例的相關資訊，請參閱下載及執行 IBM JGSS 範例。

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// ------------------------------------------------------------------------
// JAAS policy file for running the JGSS sample applications.
// Modify these permissions to suit your environment.
// Not recommended for use for any purpose other than that stated above.
// In particular, do not use this policy file or its
// contents to protect resources in a production environment.
//
// Code example disclaimer
// IBM grants you a nonexclusive copyright license to use all programming code
// examples from which you can generate similar function tailored to your own
// specific needs.
// All sample code is provided by IBM for illustrative purposes only.
// These examples have not been thoroughly tested under all conditions.
// IBM, therefore, cannot guarantee or imply reliability, serviceability, or
// function of these programs.
// All programs contained herein are provided to you "AS IS" without any
// warranties of any kind.
// The implied warranties of non-infringement, merchantability and fitness
// for a particular purpose are expressly disclaimed.
//
// ------------------------------------------------------------------------

//-------------------------------------------------------------------------
// Permissions for client only
//-------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
        Principal javax.security.auth.kerberos.KerberosPrincipal
        "bob@REALM.IBM.COM"
{
    // foo needs to be able to initiate a context with the server
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "initiate";
```

```
    // So that foo can delegate his creds to the server
    permission javax.security.auth.kerberos.DelegationPermission
        "\"gss_service/myhost.ibm.com@REALM.IBM.COM\" \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
};

//-------------------------------------------------------------------------
// Permissions for the server only
//-------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service/myhost.ibm.com@REALM.IBM.COM"
{
        // Permission for the server to accept network connections on its host
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

        // Permission for the server to accept JGSS contexts
        permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "accept";

        // The server acts as a client when communicating with the secondary (backup) server
        // This permission allows the server to initiate a context with the secondary server
        permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "initiate";
};

//-------------------------------------------------------------------------
// Permissions for the secondary server
//-------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
        Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service2/myhost.ibm.com@REALM.IBM.COM"
{
        // Permission for the secondary server to accept network connections on its host
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

        // Permission for the server to accept JGSS contexts
        permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "accept";
};
```

鏈結集合

下載及執行 IBM JGSS 範例
本主題包含有關下載及執行範例 javadoc 資訊的指示。

程式碼範例免責聲明

範例：Java 原則檔:

如需使用原則檔範例的相關資訊，請參閱下載及執行 IBM JGSS 範例。

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// --------------------------------------------------------------
// Java policy file for running the JGSS sample applications on
// the iSeries server.
// Modify these permissions to suit your environment.
// Not recommended for use for any purpose other than that stated above.
// In particular, do not use this policy file or its
// contents to protect resources in a production environment.
//
// Code example disclaimer
// IBM grants you a nonexclusive copyright license to use all programming code
// examples from which you can generate similar function tailored to your own
// specific needs.
```

```
// All sample code is provided by IBM for illustrative purposes only.
// These examples have not been thoroughly tested under all conditions.
// IBM, therefore, cannot guarantee or imply reliability, serviceability, or
// function of these programs.
// All programs contained herein are provided to you "AS IS" without any
// warranties of any kind.
// The implied warranties of non-infringement, merchantability and fitness
// for a particular purpose are expressly disclaimed.
//
//--------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar" {
        // For Java 1.3
            permission javax.security.auth.AuthPermission "createLoginContext";

        // For Java 1.4
    permission javax.security.auth.AuthPermission "createLoginContext.JAASClient";
    permission javax.security.auth.AuthPermission "createLoginContext.JAASServer";

            permission javax.security.auth.AuthPermission "doAsPrivileged";

        // Permission to request a ticket from the KDC
        permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/REALM.IBM.COM@REALM.IBM.COM", "initiate";

        // Permission to access sun.security.action classes
    permission java.lang.RuntimePermission "accessClassInPackage.sun.security.action";

        // A whole bunch of Java properties are accessed
    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";

        // Permission to communicate with the Kerberos KDC host
    permission java.net.SocketPermission "kdc.ibm.com", "connect,accept,resolve";

        // I run the samples from my localhost
    permission java.net.SocketPermission "myhost.ibm.com", "accept,connect,resolve";
    permission java.net.SocketPermission "localhost", "listen,accept,connect,resolve";

        // Access to some possible Kerberos config locations
        // Modify the file paths as applicable to your environment
    permission java.io.FilePermission "${user.home}/krb5.ini", "read";
    permission java.io.FilePermission "${java.home}/lib/security/krb5.conf", "read";

        // Access to the Kerberos key table so we can get our server key.
    permission java.io.FilePermission
    "/QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab", "read";

        // Access to the user's Kerberos credentials cache.
    permission java.io.FilePermission "${user.home}/krb5cc_${user.name}",
    "read";
};
```

範例：下載及檢視 IBM JGSS 範例的 javadoc 資訊:

若要下載及檢視 IBM JGSS 範例程式的文件，請完成下列步驟。

1. 選擇您要儲存 javadoc 資訊的現有目錄 (或建立一個新的)。

2. 下載 Javadoc 資訊 (jgsssampledoc.zip) 至此目錄。
3. 將 jgsssampledoc.zip 內的檔案擷取至此目錄。
4. 使用瀏覽器來存取 index.htm 檔案。

程式碼範例免責聲明

IBM 授予您使用所有程式設計程式碼範例的非專屬授權，您可以依據這些範例，產生類似的功能，來符合您的特定需要。

IBM 提供的所有範例程式碼僅作爲說明用途。這些範例尚未經過在所有情況下測試。因此 IBM 不保證或暗示這些程式的可靠性、服務性或功能。

所有內含於此的程式是以「現況」提供給您，不具任何形式的擔保。IBM 明示排除有關這些程式的不侵權、可售性、符合特定使用目的之默示擔保。

範例：下載及執行範例程式:

本主題包含有關下載及執行範例 javadoc 資訊的指示。

在修改或執行範例之前，請先閱讀範例程式的說明。

若要執行範例程式，請執行下列作業：
1. 將範例檔下載至 iSeries 伺服器
2. 準備執行範例檔
3. 執行範例程式

如需如何執行範例的相關資訊，請參閱第 331 頁的『範例：執行非 JAAS 範例』。

程式碼範例免責聲明

IBM 授予您使用所有程式設計程式碼範例的非專屬授權，您可以依據這些範例，產生類似的功能，來符合您的特定需要。

IBM 提供的所有範例程式碼僅作爲說明用途。這些範例尚未經過在所有情況下測試。因此 IBM 不保證或暗示這些程式的可靠性、服務性或功能。

所有內含於此的程式是以「現況」提供給您，不具任何形式的擔保。IBM 明示排除有關這些程式的不侵權、可售性、符合特定使用目的之默示擔保。

範例：下載 IBM JGSS 範例:

本主題包含有關將範例 javadoc 資訊下載至 iSeries 伺服器的指示。

在修改或執行範例之前，請先閱讀範例程式的說明。

若要下載範例檔並將其儲存於 iSeries 伺服器上，請完成下列步驟：
1. 在 iSeries 伺服器上，選擇您要儲存範例程式、配置檔及原則檔的現有目錄 (或建立一個新的目錄)。
2. 下載範例程式 (ibmjgsssample.zip)。
3. 將 ibmjgsssample.zip 的檔案擷取至伺服器的目錄中。

擷取 ibmjgsssample.jar 的內容時會執行下列動作：

- 將含有範例 .class 檔的 ibmgjsssample.jar 放入選取的目錄中
- 建立子目錄 (名爲 config) 來儲存配置檔及原則檔
- 建立子目錄 (名爲 src) 來儲存範例 .java 來源檔

相關資訊

如需閱讀相關作業的資訊或查看範例：
- 『範例：準備執行範例程式』
- 『範例：執行範例程式』
- 第 331 頁的『範例：執行非 JAAS 範例』

範例：準備執行範例程式:

下載原始程式碼之後，您需要先執行一些準備作業，然後再執行範例程式。

在修改或執行範例之前，請先閱讀範例程式的說明。

下載原始程式碼之後，必須先執行下列作業，才可以執行範例程式：
- 編輯配置檔及原則檔來符合您的環境。如需詳細資訊，請參閱每一個配置檔及原則檔內的註解。
- 確定 java.security 檔案已針對 iSeries 伺服器完成正確的設定。如需相關資訊，請參閱 Java 主要安全內容檔。
- 針對您正在使用的 J2SDK 版本，將已修改的 Kerberos 配置檔 (krb5.conf) 放入 iSeries 伺服器的目錄中：
  - 若爲 J2SDK 1.3 版：/QIBM/ProdData/Java400/jdk13/lib/security
  - 若爲 J2SDK 1.4 版：/QIBM/ProdData/Java400/jdk14/lib/security
  - 若爲 J2SDK 1.5 版：/QIBM/ProdData/Java400/jdk15/lib/security

相關資訊

如需閱讀相關作業的資訊或查看範例：
- 第 329 頁的『範例：下載 IBM JGSS 範例』
- 『範例：執行範例程式』
- 第 331 頁的『範例：執行非 JAAS 範例』

範例：執行範例程式:

在下載及修改原始程式碼之後，即可執行其中一個範例。

在修改或執行範例之前，請先閱讀範例程式的說明。

若要執行範例，首先必須啓動伺服器程式。在啓動用戶端程式之前，必須先執行伺服器程式，使其隨時可以接受連線。在看見 `listening on port <server_port>` 訊息時，就表示伺服器已備妥，隨時可以接受連線。一定要記住或寫下 <server_port>，此爲啓動用戶端時必須指定的埠號。

請使用下列指令來啓動範例程式：

```
java [-Dproperty1=value1 ... -DpropertyN=valueN] com.ibm.security.jgss.test.<program> [options]
```

其中
- [-DpropertyN=valueN] 爲一或多個選用性的 Java 內容，包括配置檔及原則檔的名稱、JGSS 除錯選項及安全管理程式。如需詳細資訊，請參閱下列範例及執行 JGSS 應用程式。
- <program> 爲必要參數，用來指定您想要執行的範例程式 (`Client`、`Server`、`JAASClient` 或 `JAASServer`)。

- [options] 爲您要執行的範例程式的選用性參數。若要顯示支援的選項清單，請使用下列指令：

```
java com.ibm.security.jgss.test.<program> -?
```

註: 您可藉由將 Java 內容 javax.security.auth.useSubjectCredsOnly 設爲 False，關閉 JGSS 範例中的 JAAS 功能。 當然，JAAS 範例的預設值是開啓 JAAS，表示此內容值爲 true。 非 JAAS 用戶端及伺服器程式則會將這個內容設爲 false，除非您明確地設定內容值。

相關資訊

如需閱讀相關作業的資訊或查看範例：

- 第 330 頁的『範例：準備執行範例程式』
- 第 329 頁的『範例：下載 IBM JGSS 範例』
- 『範例：執行非 JAAS 範例』

範例：執行非 JAAS 範例:

若要執行範例，您必須下載及修改範例原始程式碼。如需詳細資訊，請參閱下載及執行範例程式。

啓動主要伺服器

請使用下列指令來啓動在埠 4444 上接聽的非 JAAS 伺服器。 此伺服器以主體 (superSecureServer) 的身份執行，並使用了次要伺服器 (backupServer)。此伺服器亦顯示應用程式及認證除錯資訊。

```
java -classpath ibmjgsssample.jar
     -Dcom.ibm.security.jgss.debug="app, cred"
               com.ibm.security.jgss.test.Server -p 4444
               -n superSecureServer -s backupServer
```

順利執行此範例即會顯示下列訊息：

```
listening on port 4444
```

啓動次要伺服器

請使用下列指令來啓動非 JAAS 次要伺服器，此伺服器在埠 3333 上接聽，而且以主體 backupServer 的身份執行：

```
java -classpath ibmjgsssample.jar
               com.ibm.security.jgss.test.Server -p 3333
               -n backupServer
```

啓動用戶端

請使用下列指令 (鍵入同一行) 來執行 JAAS 用戶端 (myClient)。用戶端與主電腦 (securityCentral) 的主要伺服器會進行通訊。用戶端會在啓用預設安全管理程式的情況下執行，且使用 JAAS 配置和原則檔及 config 目錄中的 Java 原則檔。如需 config 目錄的相關資訊，請參閱下載 IBM JGSS 範例。

```
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
```

鏈結集合

下載及執行範例程式
本主題包含有關下載及執行範例 javadoc 資訊的指示。

下載 IBM JGSS 範例
本主題包含有關將範例 javadoc 資訊下載至 iSeries 伺服器的指示。

IBM JGSS javadoc 參考資訊

IBM JGSS 的 javadoc 參考資訊包括 org.ietf.jgss api 套件的類別及方法，以及部分 Java 版本的 Kerberos 認證管理工具。

雖然 JGSS 包含數個公開存取的套件 (如 com.ibm.security.jgss 及 com.ibm.security.jgss.spi)， 但您應僅就標準 org.ietf.jgss 套件的 API 加以使用。若只採用這個套件，即可確保應用程式符合 GSS-API 規格，發揮最佳的互用性和可攜性。

- org.ietf.jgss
- 第 302 頁的『com.ibm.security.krb5.internal.tools Class Kinit (無中文)』
- 第 304 頁的『com.ibm.security.krb5.internal.tools Class Ktab (無中文)』
- 第 301 頁的『com.ibm.security.krb5.internal.tools Class Klist (無中文)』

使用 IBM Developer Kit for Java 調整 Java 程式效能

在建置 iSeries 伺服器的 Java 應用程式時，您應該將 Java 應用程式效能的某些因素納入考量。

您可透過以下幾個鏈結，取得如何獲得較佳效能的說明及提示：

- 使用「建立 Java 程式 (CRTJVAPGM)」指令、即時編譯器，或使用快取記憶體來快取使用者類別載入器，設法增進 Java 程式碼的效能。
- 變更最佳化等級，達到最佳的靜態編譯效能。
- 小心設定您的值來獲得最佳的垃圾收集效能。
- 對於執行時間非常長且無法在 Java 中直接取用的系統功能，才應該透過原生方法啓動。
- 編譯時使用 `javac -o` 選項來執行方法列入行內，大幅提升方法呼叫效能。
- 對於應用程式內非正常流程的狀況，使用 Java 異常。

請利用這些工具再搭配「效能探測器 (PEX)」，找出 Java 程式的效能問題：

- 您可以使用 iSeries Java 虛擬機器來收集第 333 頁的『Java 事件追蹤效能工具』。
- 若要判斷每一個 Java 方法耗費的時間，請利用 Java 呼叫追蹤。
- 對於每一個 Java 方法及 Java 程式佔用的所有系統功能，Java 效能分析可以計算出相對耗費的 CPU 時間量。
- 對於 iSeries 伺服器上執行的程式，使用 Java 效能資料收集器來提供相關的效能分析資訊。

任何工作階段作業皆可啓動及結束 PEX。收集的資料通常屬於系統全面性的資料，範疇含括系統上的所有工作，也包括您的 Java 程式。有時可能必須在 Java 應用程式內啓動及停止效能收集。這麼做可縮短收集時間，亦可減少通常因呼叫追蹤或傳回追蹤而產生的大量資料。 PEX 無法從 Java 緒之內執行。若要啓動和停止收集，您必須編寫原生方法，透過佇列或共用記憶體，與獨立的工作進行通訊。然後，由第二個工作在適當時機啓動和停止收集。

除了應用程式層次的效能資料以外，還可以使用現有的 iSeries 系統層次效能工具。這些工具以單一的 Java 緒爲基礎來報告統計值。

相關資訊

Performance Tools for iSeries, SC41-5340

 此手冊包含 PEX 報告的範例。

Java 事件追蹤效能工具

iSeries Java 虛擬機器允許追蹤特定 Java 事件。

不必借助於任何 Java 程式碼，就可以收集這些事件。這些事件包括垃圾收集、緒的建立、類別載入及鎖定等活動。「執行 Java (RUNJVA)」指令不指定這些事件。而是由您建立「效能探究程式 (PEX)」定義，然後使用「啓動效能探究程式 (STRPEX)」指令來收集事件。每一個事件都包含有用的效能資訊，例如時間戳記和中央處理單元 (CPU) 週期。您可以使用相同的追蹤定義來追蹤 Java 事件及其他系統活動，例如磁碟輸入及輸出。

如需 Java 事件的完整說明，請參閱 Performance Tools for iSeries, SC41-5340。

Java 效能注意事項

瞭解下列注意事項，有助於提升 Java 應用程式的效能。

建立最佳化 Java 程式

爲了大幅提升 Java 程式碼的啓動效能，請在執行 Java 類別檔案、JAR 檔或 ZIP 檔之前，先使用「建立 Java 程式 (CRTJVAPGM)」控制語言指令。CRTJVAPGM 指令會使用位元組碼來建立 Java 程式物件 (內含針對 iSeries 伺服器的最佳化原有指令)，並且讓 Java 程式物件結合類別檔案、JAR 檔或 ZIP 檔。

後續執行起來就更快，因爲 Java 程式已儲存，且維持與類別檔案或 JAR 檔的關聯性。在應用程式開發期間，尚可接受以解譯方式執行位元組碼的效能，但在正式作業環境下，您可能希望在執行 Java 程式碼之前，先使用 CRTJVAPGM 指令。

執行 Java 類別檔案、JAR 檔或 ZIP 檔之前，若未使用 CRTJVAPGM，i5/OS 將使用即時編譯器 (搭配「混合模式直譯器」) 來代替。

選取最佳化等級

建立 Java 程式物件時，請參考下列準則，協助您針對希望使用的執行模式來選取最好的最佳化等級：

- 希望採取直接處理時，請以最佳化等級 30 或 40 來建立最佳化 Java 程式物件。
- 希望只採用即時 (JIT) 編譯器來執行時，請使用 `*Interpret` 最佳化參數來建立最佳化 Java 程式。以 `*Interpret` 參數所建立的 Java 程式，比最佳化等級 40 所建立的程式更小。
- 希望採用預設執行模式時 (混合直接處理與 JIT 編譯器)，請使用下列設定值來建立 Java 程式物件：
  - 對於希望以直接處理來執行的類別，使用最佳化等級 30 或 40
  - 對於希望以 JIT 編譯器來執行的類別，使用 `*Interpret` 最佳化參數

如需詳細資訊，請參閱以下各頁：

建立 Java 程式 (CRTJVAPGM) 控制語言指令

選取使用何種模式來執行 Java 程式

使用即時編譯器

使用 JIT 編譯器與「混合模式直譯器 (MMI)」的啓動效能，幾乎與編譯的程式碼不相上下。 MMI 會解譯 Java 程式碼，直到觸及 os400.jit.mmi.threshold Java 系統內容所指定的臨界值爲止。達到臨界值之後，i5/OS 會將 JIT

編譯器在編譯方法時所需的時間及資源，花費在最常用的方法上。使用即時編譯器 (JIT) 可以產生高度最佳化的程式碼，相較於經過前置編譯的程式碼，執行時間效能可以獲得提升。如需採用 JIT 來改進啓動效能，可以使用 CRTJVAPGM 來建立最佳化 Java 程式物件。

若程式執行緩慢，請輸入顯示 Java 程式 (DSPJVAPGM) 控制語言指令，檢視 Java 程式物件的屬性。請確定 Java 程式物件已爲您採取最佳執行模式。如需變更執行模式，您可能要刪除 Java 程式物件，再以不同的最佳化參數來建立新的 Java 程式。

如需詳細資訊，請參閱：

「顯示 Java 程式 (DSPJVAPGM)」控制語言指令

使用快取記憶體來快取使用者類別載入器

對於從使用者類別載入器中載入的類別，使用 i5/OS Java 虛擬機器 (JVM) 快取記憶體來快取使用者類別載入器，可以提升啓動效能。此快取記憶體可以儲存最佳化 Java 程式物件，供 JVM 重覆使用。重覆使用儲存的 Java 程式，由於可避免重建快取的 Java 程式物件及驗證位元組碼，因此能夠提升效能。

請使用下列內容來控制使用者類別載入器的快取記憶體：

os400.define.class.cache.file
這個內容的值指定有效 Java ARchive (JAR) 檔的名稱 (完整路徑)。指定的 JAR 檔至少必須包含有效的 JAR 目錄 (由 `jar` QSH 指令所建置)， 以及一個運作 `jar` 指令的必要成員。 請勿於任何 Java CLASSPATH 中包含指定的 JAR 檔。這個內容的預設值爲 /QIBM/ProdData/Java400/QDefineClassCache.jar。不指定這個內容的值，即可停用此快取機制。

os400.define.class.cache.hours
這個內容的值指定 Java 程式物件在快取記憶體中要持續多久 (以小時爲單位)。當 JVM 超過指定的時間未使用某個快取的 Java 程式物件，i5/OS 就會從快取記憶體中移除此 Java 程式物件。這個內容的預設值爲 768 小時 (33 天)。最大值爲 9999 (大約 59 週)。 若指定的值爲 0，或是 i5/OS 無法辨識爲有效十進位數的值，i5/OS 即採用預設值。

os400.define.class.cache.maxpgms
這個內容的值指定快取記憶體最多可以保存多少 Java 程式物件。當快取記憶體超出此限制時，i5/OS 就會從快取記憶體中移除最舊的 Java 程式物件。i5/OS 會比較 JVM 前次參照 Java 程式物件的時間，判斷哪個快取的程式最舊。預設值爲 5000，最大值爲 40000。若指定的值爲 0，或是 i5/OS 無法辨識爲有效十進位數的值，i5/OS 即採用預設值。

對於在 os400.define.class.cache.file 內容中指定的 JAR 檔，請使用 DSPJVAPGM 來判斷已快取的 Java 程式物件數。

- DSPJVAPGM 顯示畫面的 **Java 程式**欄位，指出已快取的 Java 程式物件數。
- **Java 程式大小**欄位指出快取的 Java 程式物件所佔用的儲存體數量。
- 在快取的 JAR 檔上使用此指令時，DSPJVAPGM 顯示畫面的其他欄位則無意義。

快取效能

執行一些 Java 應用程式可能會快取大量的 Java 程式物件。在應用程式執行完畢之前，請使用 DSPJVAPGM 來判斷快取的 Java 程式數量是否接近最大值。當快取記憶體不足時，應用程式效能就會退化，因爲 i5/OS 可能從快取記憶體中移除應用程式所需的部分程式。

可以防止快取記憶體不足而導致效能退化。例如，若應用程式經常執行，且在快取記憶體中載入不同程式，您可以設定應用程式來使用個別的快取記憶體。使用個別的快取記憶體可以防止快取記憶體不足，避免 i5/OS 從快取記憶體中移除 Java 程式。另外，亦可增加 os400.define.class.cache.maxpgms 內容中指定的數值。

可以在 JAR 檔上使用「變更 Java 程式 (CHGJVAPGM)」控制語言指令，變更快取記憶體中類別的最佳化等級。CHGJVAPGM 僅影響快取記憶體目前保留的程式。變更最佳化等級之後，os400.defineClass.optLevel 內容就指定如何對快取記憶體內加入的任何類別予以最佳化。

例如，若最多有 10000 個 Java 程式物件會使用產品提供的快取 JAR，而每一個 Java 程式的使用期限最長 1 年，請設定下列快取內容值：

```
os400.define.class.cache.file    /QIBM/ProdData/Java400/QDefineClassCache.jar
     os400.define.class.cache.hours   8760
     os400.define.class.cache.maxpgms 10000
```

選取執行 Java 程式時使用的模式

執行 Java 程式時，您可以選取想要使用的模式。所有模式都可以驗證程式碼，也可以建立 Java 程式物件來保存驗證之前的程式。

您可以使用下列任何一種模式：

- 解譯
- 直接處理
- JIT 編譯
- JIT 編譯及直接處理

選擇模式	詳細資料
解譯	執行時解譯每一個位元組碼。 如需以解譯模式執行 Java 程式的相關資訊，請參閱執行 Java (RUNJVA) 指令。
直接處理	第一次呼叫某方法時，就會產生此方法的機器指令，並且儲存起來供下次程式執行時使用。整個系統也同時共用一個複本。 如需以直接處理方式來執行 Java 程式的相關資訊，請參閱執行 Java (RUNJVA) 指令。

選擇模式	詳細資料

JIT 編譯	i5/OS 會解譯 Java 方法，直到達到 os400.jit.mmi.threshold Java 系統內容所指定的臨界值爲止。達到臨界值之後，i5/OS 就會使用 JIT 編譯器將方法編譯成原有的機器指令。
	
	若要使用即時編譯器，設將編譯器值設爲 jitc。可以藉由新增環境變數或設定 java.compiler 系統內容來設定此值。請從下列清單選取一種方法來設定編譯器值：
	• 從 iSeries 伺服器的指令行提示，使用「新增環境變數 (ADDENVVAR)」指令來新增環境變數。然後，再使用「執行 Java (RUNJVA)」指令或 JAVA 指令來執行 Java 程式。例如使用：
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc)`
	`JAVA CLASS(Test)`
	• 在 iSeries 指令行上設定 java.compiler 系統內容。例如輸入 `JAVA CLASS(Test) PROP((java.compiler jitc))`
	• 在「Qshell 直譯器」指令行上設定 java.compiler 系統內容。例如輸入 `java -Djava.compiler=jitc Test`
	
	設定此值之後，JIT 編譯器在執行所有 Java 程式碼之前都會先對其進行最佳化。

JIT 編譯及直接處理	「JIT 編譯器」最常見的用法是搭配 jit_de 選項。搭配此選項執行時，已透過直接處理而最佳化的程式會以直接處理模式來執行。尚未經過直接最佳化的程式，則以 JIT 模式來執行。
	
	若希望 JIT 與直接處理一起使用，請將編譯器值設爲 jitc_de。可以藉由新增環境變數或設定 java.compiler 系統內容來設定此值。請從下列清單選取一種方法來設定編譯器值：
	• 在 iSeries 指令行上輸入「新增環境變數 (ADDENVVAR)」指令來新增環境變數。然後，再使用「執行 Java (RUNJVA)」指令或 JAVA 指令來執行 Java 程式。例如輸入
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc_de)`
	`JAVA CLASS(Test)`
	• 在 iSeries 指令行上設定 java.compiler 系統內容。例如輸入 `JAVA CLASS(Test) PROP((java.compiler jitc_de))`
	• 在「Qshell 直譯器」指令行上設定 java.compiler 系統內容。例如輸入 `java -Djava.compiler=jitc_de Test`
	
	設定此值之後，就會使用爲直接處理而建立之類別檔案的 Java 程式。如果 Java 程式並非是爲了直接處理而建立，則 JIT 在類別檔案執行之前會先對其進行最佳化。如需詳細資訊，請參閱即時編譯器與直接處理的比較

有三種方法可以執行 Java 程式 (CL、QSH 及 JNI)。三者各有獨特的方法來指定模式。下表顯示完成的方式。

模式	CL 指令	QShell 指令	JNI 呼叫 API
解譯	INTERPRET(*YES)	-Djava.compiler=NONE -interpret	os400.run.mode=interpret
DE	INTERPRET(*NO)	-Djava.compiler=NONE	• os400.run.mode= program_created=pc • os400.create.type= direct
JIT	INTERPRET(*JIT)	-Djava.compiler=jitc	os400.run.mode=jitc
JIT_DE(預設值)	INTERPRET(*OPTIMIZE) OPTIMIZE(*JIT)	-Djava.compiler=jitc_de	os400.run.mode=jitc_de

Java 直譯器

Java 直譯器屬於 Java 虛擬機器的一部分，可以針對特定的硬體平台來解譯 Java 類別檔案。Java 直譯器可將每一個位元組碼解碼，然後對此位元組碼執行一連串機器指令。

Java 虛擬機器

靜態編譯

Java 轉換程式是 IBM i5/OS 元件，可以預先處理類別檔案，將其備妥，以使用 iSeriesJava 虛擬機器來執行。Java 轉換程式可以建立持續性且關聯類別檔案的最佳化程式物件。

根據預設，此程式物件包含經過編譯的 64 位元 RISC 機器指令版本的類別。 Java 直譯器不會在執行時間解譯最佳化的程式物件。而是於載入類別檔案時直接執行。

依預設，會以 JIT 來最佳化 Java 程式。若要使用 Java 轉換程式，請執行 CRTJVAPGM，或在 RUNJVA 或 JAVA 指令上指定使用轉換程式。

您可以使用「建立Java 程式 (CRTJVAPGM)」指令，明確地啓動 Java 轉換程式。CRTJVAPGM 指令執行時就會最佳化類別檔案或 JAR 檔，所以執行程式時不必再採取任何動作。如此可以提高程式第一次執行時的速度。若不倚賴預設的最佳化機制，而改用 CRTJVAPGM 指令，即可確保將最佳化效果發揮到極至，也讓類別檔案或 JAR 檔相關的 Java 程式更有效率地運用空間。

對類別檔案、JAR 檔或 ZIP 檔執行 CRTJVAPGM 指令，可以將檔案內的所有類別最佳化，而產生的 Java 程式物件也將具有持續性。進一步會提升執行時間效能。利用 CRTJVAPGM 指令或變更 Java 程式 (CHGJVAPGM) 指令，亦可變更最佳化等級或選取預設值 10 以外的其他最佳化等級。在最佳化等級 40 上，JAR 檔內的類別之間會進行連結，有時類別還會列入行內。類別間連結可以改善呼叫速度。列入行內可以完全消除方法呼叫的額外執行時間。有時，在 JAR 檔或 ZIP 檔內的類別之間，亦可將方法列入行內。在 CRTJVAPGM 指令上指定 OPTIMIZE(*INTERPRET)，可以驗證指令上指定的任何類別，並將其備妥，以解譯模式來執行。

「執行 Java (RUNJVA)」指令亦可指定 OPTIMIZE(*INTERPRET)。此參數指定出，不論相關程式物件的最佳化等級爲何，都必須解譯 Java 虛擬機器之下執行的任何類別。針對以最佳化等級 40 來轉換的類別進行除錯時，這就非常有用。若要強制解譯，請使用 INTERPRET(*YES)。

如需重覆使用類別載入器所建立之 Java 程式的相關資訊，請參閱 Java 效能考量的「使用快取記憶體來快取使用者類別載入器」。

Java 靜態編譯效能考量:

透過您設定的最佳化等級，可以決定轉換的速度。

最佳化等級 10 的轉換速度最快，但產生的程式執行起來的速度，通常不如更高的最佳化等級。最佳化等級 40 需要較久的轉換時間，但程式執行的速度較快。

少數 Java 程式不會最佳化到等級 40。因此，無法以等級 40 執行的少數程式，可以改用等級 30 來執行。對於無法以最佳化等級 40 來執行的程式，您可以使用授權內碼最佳化 LICOPT 參數字串來執行。但也許等級 30 就足以執行您的程式。

對於可能在另一個 Java 虛擬機器中執行的 Java 程式碼，若執行方面出現問題，請嘗試使用最佳化等級 30 代替等級 40。若取代奏效，且您可以接受這樣的效能表現，即不必再做任何調整。若您要求更好的效能，請參閱 LICOPT 參數字串，瞭解如何啓用及停用各種最佳化形式的相關資訊。例如，一開始可以使用 OPTIMIZE(40) LICOPT(NoPreresolveExtRef) 來建立程式。即使應用程式裡對不存在類別發出無效的呼叫，此 LICOPT 值還是可讓程式執行下去，不會有問題。

若要判斷 Java 程式建立時採用的最佳化等級，請使用「顯示 Java 程式 (DSPJVAPGM)」指令。若要變更 Java 程式的最佳化等級，請使用「建立 Java 程式 (CRTJVAPGM)」指令。

即時編譯器

JIT 編譯器是一種平台專用的編譯器，可爲每一個必要的方法產生機器指令。

有關使用 JIT 編譯器的詳細資訊，以及 JIT 編譯器與直接處理之間的差異，請參閱下列各頁：

Java Runtime 效能注意事項。

JIT 編譯器與直接處理的比較。

註: i5/OS 的預設值利用「混合模式直譯器 (MMI)」來解譯 (非編譯) Java 方法。MMI 於解譯每個 Java 方法時，也會同時進行效能分析。在達到 os400.jit.mmi.threshold 內容所指定的臨界值之後，MMI 就會指定 i5/OS 使用 JIT 編譯器來編譯方法。

如需相關資訊，請參閱適當 Java 系統內容清單中的 java.compiler 內容及 os400.jit.mmi.threshold 內容：

Java 2 SDK (J2SDK) 標準版的 Java 系統內容

即時編譯器與直接處理的比較:

若您正在考慮是使用即時編譯器還是直接處理模式來執行 Java 程式，下表可提供額外的資訊，協助您依據面臨的狀況做出最佳的抉擇。

即時編譯器或直接處理模式

即時編譯器	直接處理
必要時自動編譯任何方法。JIT 編譯器編譯方法的速度，比直接處理更快。	可讓您使用「建立 Java 程式 (CRTJVAPGM)」控制語言 (CL) 指令，編譯整個類別或 JAR 檔。若您不編譯檔案，直接處理會在執行時間自動編譯檔案。
讓您在開發程式的過程中避免使用 CRTJVAPGM CL 指令。對於執行時才產生或探索程式碼的高動態性應用程式，您亦可使用 JIT 編譯器。	大部份可立即部署的伺服器應用程式，都在最佳化等級 40 上使用直接處理，因爲極可能隨時都有多位使用者同時使用這種應用程式。由於多個使用者工作共用記憶體中相同的編碼空間，因此可以縮小記憶體覆蓋區。

即時編譯器	直接處理
在執行時間快速執行複雜的最佳化及 Java 特有的最佳化。	啓用複雜的最佳化，因爲直接處理在執行時間不進行最佳化。然而，直接處理無法始終執行 Java 特定的最佳化 (例如，將方法列入行內)，因爲 Java 程式物件必須各自獨立。
相較於直接處理，程式碼效能較高。在大部份情況下，JIT 產生的程式碼的效能，通常高於直接處理最佳化等級 40。	提供 Java 程式採用擁有者權限的唯一方式。

Java 垃圾收集

針對不再受程式參照的物件，釋放其佔用的儲存體，此程序稱之爲「垃圾收集」。幸好有垃圾收集，程式設計師不必再編寫易出錯的程式碼來明確「釋放」或「刪除」物件。這類程式碼經常導致「記憶體洩漏」程式錯誤。垃圾收集器會自動偵測使用者程式無法再存取的一個物件或一組物件。這是因爲在任何程式結構中，對此物件已無任何參照。一旦回收物件之後，您就可以再配置空間做其他用途。

Java 執行時間環境包括垃圾收集器，可以釋放不再用到的記憶體。垃圾收集器會視情況需要而自動執行。

另外，亦可使用 java.lang.Runtime.gc() 方法，在 Java 程式的控制下明確地啓動垃圾收集器。

IBM Developer Kit for Java 進階垃圾收集

IBM Developer Kit for Java 會實作進階的垃圾收集器演算法。此演算法可以探查並回收無法存取的物件，而且不會造成 Java 程式的作業明顯停頓。另外，並行收集器會在執行中所有的緒之下 (而非於單一緒之下)，協助探查物件參照。

許多垃圾收集器都有「全部暫停」的現象。這意謂著，當回收週期開始時，除了正在執行垃圾收集的緒以外，其他所有的緒都會在垃圾收集器運作時停止。在此情況下，Java 程式會暫停，而收集器在繼續運作時，平台的多處理器功能都因 Java 而浪費。iSeries 演算法不會同時停止所有程式緒。它可讓這些緒在垃圾收集器執行時繼續運作。它可防止暫停現象的出現，並確保在垃圾收集期間仍可繼續使用所有處理器。

垃圾收集可根據您啓動 Java 虛擬機器時指定的參數時自動執行。亦可藉由使用 java.lang.Runtime.gc() 方法，在 Java 程式的控制下明確啓動垃圾收集。

如需基本的定義，請參閱 Java 垃圾收集。

Java 垃圾收集效能考量

iSeries Java 虛擬機器的垃圾收集以連續非同步模式運作。「執行 Java (RUNJVA)」指令的垃圾收集起始大小 (GCHINL) 參數，可能會影響應用程式效能。

GCHINL 參數指定垃圾收集之間允許多少新的物件空間。太小的值可能導致垃圾收集負荷太重。太大的值亦可能限制垃圾收集，導致記憶體不足的錯誤。然而，對大部份應用程式而言，使用預設值應該就錯不了。

垃圾收集會評估某物件是否還存在任何有效的參照，藉此決定此物件是否已無用處。

Java 原生方法呼叫的效能注意事項

原生方法呼叫在 iSeries 伺服器上的執行效能，可能不如在其他平台上好。

已將 Java 虛擬機器移至機器介面 (MI) 之下，藉此最佳化 iSeries 伺服器上的 Java。原生方法呼叫需要呼叫上層的 MI 程式碼，且可能需要對 Java 虛擬機器執行價格昂貴的「Java 原生介面 (JNI)」回呼。原生方法不應該只是處理您用 Java 就可輕易編寫的小常式。對於執行時間很長且無法在 Java 中直接取用的系統功能，才應該透過原生方法來啓動。

Java 方法列入行內的效能注意事項

將方法列入行內可以明顯改善方法呼叫效能。任何 final 方法，都很適合列入行內。

在 iSeries 伺服器上，可以在編譯時透過 javac -o 選項來利用行內特性。使用 javac -o 選項時，類別檔案及轉換後的 Java 程式會變大。使用 -o 選項時，請考量應用程式的空間與效能性質。

註: 一般而言，最好不要使用 javac 的 -o 選項，而將列入行內的動作留待後續階段再處理。

Java 轉換程式允許在最佳化等級 30 及最佳化等級 40 上列入行內。最佳化等級 30 可以在單一類別內列入一些 final 方法。最佳化等級 40 可以在 ZIP 檔或 JAR 檔內列入 final 方法。您可以利用 AllowInlining 及 NoAllowInlining LICOPT 參數字串來控制方法列入行內。iSeries 直譯器不會將方法列入行內。

即時 (JIT) 編譯器也會將大部分 final 方法列入行內。每當 JIT 編譯器啓動時，若判斷列入行內的結果較佳，就會自動執行列入行內的動作。

Java 異常效能注意事項

iSeries 異常架構支援多用途的岔斷及重試能力。它還允許不同語言混合互動。在 iSeries 伺服器上擲出 Java 異常，代價可能高於其他平台。除非正常的應用程式執行流程固定使用 Java 異常，否則應該不影響整體的應用程式效能。

Java 呼叫追蹤效能工具

Java 方法呼叫追蹤可針對每一個 Java 方法的執行時間，提供重要的效能資訊。

在其他 Java 虛擬機器上，您可能會使用 java 指令上的 -prof (效能分析) 選項。若要在 iSeries 伺服器上啓用方法呼叫追蹤，您必須在「建立 Java 程式 (CRTJVAPGM)」指令行上指定「啓用效能收集 (ENBPFRCOL)」指令。使用此關鍵字建立 Java 程式之後，即可使用含有呼叫/傳回追蹤類型的「效能探究程式 (PEX)」定義，開始收集方法呼叫追蹤。

「列印效能探究程式報告 (PRTPEXRPT)」指令所產生的呼叫/傳回追蹤輸出，可針對受追蹤的每一個 Java 方法的每一個呼叫，顯示其佔用的中央處理單元 (CPU) 時間。有時，您可能無法讓所有類別檔案都啓用呼叫傳回追蹤。或者，您也有可能呼叫到未啓用追蹤的原生方法和系統函數。在此狀況下，花費在這些方法或系統函數上的所有 CPU 時間就會累計。然後，向最後一個呼叫且已啓用的 Java 方法提出報告。

Java 執行時間效能工具

對於 Java 程式所用的每一個 Java 方法及所有系統功能，系統層面的中央處理單元 (CPU) 效能分析可以計算出相對 CPU 時間量。

請使用「效能探究程式 (PEX)」定義來追蹤效能監視器計數器溢位 (*PMCO) 執行週期事件。 通常指定千分之一秒爲採樣間隔。爲了收集有效的追蹤效能分析資訊，Java 應用程式應該持續執行大約 2 至 3 分鐘的 CPU 時間。這樣應該足夠產生 100,000 個以上的樣本。 「列印效能探究程式報告 (PRTPEXRPT)」指令可以產生整個應用程式經歷的 CPU 時間直方圖。 這包括每個 Java 方法及所有系統層次的活動。對於 iSeries 伺服器上執行的程式，效能資料收集器 (PDC) 工具亦可提供效能分析資訊。

註: 對於需要解譯的 Java 程式，CPU 效能分析不會顯示相對 CPU 使用率。

鏈結集合

效能資料收集器 (PDC) 工具
「效能資料收集器 (PDC)」工具可針對 iSeries 伺服器上執行的程式提供相關的效能分析資訊。

Java 虛擬機器 Profiler 介面

「Java 虛擬機器 Profiler 介面 (JVMPI)」是一種用來對 Java 虛擬機器 (JVM) 進行效能分析的實驗性介面，首次在 Sun 的 Java 2 SDK 標準版 (J2SDK) 1.2 版中公開與實作。

JVMTI 取代了 JVMPI 及「Java 虛擬機器除錯程式介面 (JVMDI)」。JVMTI 包含 JVMDI 及 JVMPI 的所有功能，以及一些新功能。JVMTI 已新增爲 J2SE 5.0 的一部分。未來的版次將不再提供 JVMDI 及 JVMPI 介面，僅會提供 JVMTI。

如需實作 JVMTI 的相關資訊，請參閱 Sun Microsystems 公司網站的 JVMTI Reference 網頁。

JVMPI/JVMTI 支援會在 JVM 及即時 (JIT) 編譯器中掛上連結鉤，當這兩種機制啓動時，就會提供事件資訊給效能分析代理程式。效能分析代理程式實作爲一個整合語言環境 (ILE) 服務程式。Profiler 會向 JVM 傳送控制資訊，以啓用及停用 JVMPI/JVMTI 事件。例如，Profiler 可能對方法 Entry 或 Exit 追蹤點不感興趣，此時就可向 JVM 表示不想接收這些事件通知。若啓用事件，JVM 及 JIT 內含的 JVMPI/JVMTI 事件連結鉤會傳送事件通知給效能分析代理程式。Profiler 會將希望接收的事件告知 JVM，當事件發生時，JVM 就會將事件通知傳送給 Profiler。

服務程式 QSYS/QJVAJVMPI 提供 JVMPI 函數。

QSYS 檔案庫內稱爲 QJVAJVMTI 的服務程式支援 JVMTI 功能。

如需相關資訊，請參閱 Sun Microsystems 公司的 JVMPI。

收集 Java 效能資料

若要在 iSeries 伺服器上收集 Java 效能資料，請遵循下列步驟。

1. 建立「效能探究程式 (PEX)」定義來指定：
   - 使用者定義的名稱
   - 資料收集的類型
   - 工作名稱
   - 您希望收集相關系統資訊的一系列系統事件

   註: 若您想要的輸出是 `java_g -prof` 類型，而且您知道 Java 程式的特定工作名稱，則 *STATS 的 PEX 定義優於 *TRACE 定義。

   以下爲 *STATS 定義的範例：

   ```
   ADDPEXDFN DFN(YOURDFN) JOB(*ALL/YOURID/QJVACMDSRV) DTAORG(*HIER)
       TEXT('your stats definition')
   ```

   此 *STATS 定義不會取得所有正在執行的 Java 事件。僅記錄您自己的 Java 階段作業中發生的 Java 事件。這種作業模式可能會增加 Java 程式的執行時間。

   以下爲 *TRACE 定義的範例：

   ```
   ADDPEXDFN DFN(YOURDFN) TYPE(*TRACE) JOB(*ALL) TRCTYPE(*SLTEVT)
       SLTEVT(*YES) PGMEVT(*JVAENTRY *JVAEXIT)
   ```

   對於系統中以 ENBPFRCOL(*ENTRYEXIT) 所建立的任何 Java 程式，此 *TRACE 定義會從中收集任何 Java 進入事件及跳出事件。如此將導致這種收集的分析比 *STATS 追蹤更慢，視您的 Java 程式事件的多寡及 PEX 資料收集的持續時間而定。
2. 在 PEX 定義上的程式事件種類之下，啓用 *JVAENTRY 和 *JVAEXIT，讓 PEX 能夠辨識 Java 進入及跳出。

註: 若您使用即時 (JIT) 編譯器來執行 Java 程式碼，則不必像使用 CRTJVAPGM 指令來直接處理一樣啓用進入及跳出。當您使用 os400.enbprfcol 系統內容時，JIT 會利用進入及結束連結鉤來產生程式碼。

3. 備妥 Java 程式準備向「iSeries 效能資料收集器」報告程式事件。

   若要執行此動作，請對您要報告效能資料的任何 Java 程式，執行建立 Java 程式 (CRTJVAPGM) 指令。您必須使用 ENBPFRCOL(*ENTRYEXIT) 參數來建立此 Java 程式。

   註: 您必須針對要收集效能資料的每一個 Java 程式，重複這項步驟。若未執行這項步驟，則 PEX 不會收集效能資料，且執行 「Java 效能資料轉換器 (JPDC)」工具也不會產生輸出。

4. 使用「啓動效能探究程式 (STRPEX)」指令來啓動 PEX 資料收集。
5. 執行您要分析的程式。

   請勿在生產環境中執行此程式。在生產環境中，短時間內就會產生大量資料。請將收集時間限制爲 5 分鐘。這段時間內執行的 Java 程式會產生大量 PEX 系統資料。若收集太多資料，處理時間將超出合理的範圍。

6. 使用「結束效能探究程式 (ENDPEX)」指令來結束 PEX 資料收集。

   註: 若不是第一次結束 PEX 資料收集，您必須指定置換檔案 *YES，否則不會儲存您的資料。

7. 執行 JPDC 工具。
8. 使用您選擇的檢視器，將整合檔案系統目錄連接至系統：java_g -prof 檢視器或 Jinsight 檢視器。

   您可以從 iSeries 伺服器複製此檔案，作爲任何適當效能分析工具的輸入。

「效能資料收集器」工具

「效能資料收集器 (PDC)」工具可針對 iSeries 伺服器上執行的程式提供相關的效能分析資訊。

許多 Java 虛擬機器的產業標準效能分析選項，皆視 `java_g` 功能的實作方式而定。此爲特殊除錯版本的 Java 虛擬機器，提供 `-prof` 選項。您可以在呼叫 Java 程式時指定此選項。指定此選項時，Java 虛擬機器會產生記錄檔，內含 Java 程式於執行期間運用到哪些部分的相關資訊。Java 虛擬機器會即時產生這項資訊。

在 iSeries 伺服器上，「效能探究程式 (PEX)」這項功能可以分析程式及特定記錄的系統事件。DB2 資料庫可以儲存此資訊，亦可使用 SQL 函數來擷取它。PEX 資訊是特定程式資訊的儲存庫，可用來產生 Java 效能分析資料。這項效能分析資料與 `java_g -prof` 程式效能分析資訊相容。 Java 效能資料轉換器 (JPDC) 工具可以提供 `java_g -prof` 程式輸出，以及 IBM 工具 Jinsight 話專用的程式效能分析資訊，亦即 Jinsight。

如需如何收集 Java 效能資料的相關資訊，請參閱收集 Java 效能資料。

Java 效能資料轉換器工具

對於 iSeries 伺服器上執行的 Java 程式，「Java 效能資料轉換器 (JDPC)」工具可讓您建立相關的 Java 效能資料。這項效能資料與 Sun Microsystems 公司之 Java 虛擬機器 `java_g -prof` 選項及 IBM Jinsight 輸出的效能資料輸出相容。

註: JDPC 工具產生的輸出無法直接閱讀。請使用可接受 `java_g -prof` 或 Jinsight 資料的 Java 效能分析工具來分析您的資料。

JDPC 工具可以存取 DB2/400 (使用 JDBC) 儲存的「iSeries 效能探究程式 (PEX)」資料。再將資料轉換成 Jinsight 或一般的效能類型。然後，JPDC 會將輸出檔儲存在整合檔案系統中由使用者指定的位置。

註: 當您在 iSeries 伺服器上執行指定的 Java 應用程式時，必須遵循適當的 iSeries 資料收集程序來收集 PEX 資料。您必須設定 PEX 定義，包括定義程式或集合以及儲存程序的進入點及跳出點。如需如何收集 PEX 資料及設定 PEX 定義的詳細資料，請參閱 Performance Tools for iSeries, SC41-5340。

如需如何執行 JPDC 的相關資訊，請參閱執行 Java 效能資料轉換器。

您可以使用 Qshell 指令行介面或「執行 Java (RUNJVA)」指令來啓動 JPDC 程式。

執行 Java 效能資料轉換器

若要執行「Java 效能資料轉換器 (JPDC)」來收集效能資料，請遵循下列步驟。

1. 輸入第一個輸入引數，若使用 `java_g -prof`，請輸入 `general`，若想要 Jinsight 輸出，則輸入 `jinsight`。
2. 輸入第二個輸入引數，其爲用來收集資料之「效能探究程式 (PEX)」定義的名稱。

   註: 因爲連線內部會用到此名稱，所以應將此名稱限制爲 4 或 5 個字元。
3. 輸入第三個輸入引數，其爲 JPDC 工作所產生之檔案的名稱。

   此檔案會寫入現行整合檔案系統目錄中。請使用 cd (PF4) 指令來指定整合檔案系統的現行目錄。
4. 輸入第四個輸入引數，其爲 iSeries 主電腦關聯式資料庫目錄項目的名稱。

   請透過「使用關聯式資料庫目錄項目 (WRKRDBDIRE)」指令來查詢名稱。 此爲 *LOCAL 指出的唯一關聯式資料庫。

若要操作此程式碼，/QIBM/ProdData/Java400/ext/JPDC.jar 檔必須位於 iSeries 伺服器的 Java 類別路徑中。當程式執行完成時，即可於現行目錄下找到文字輸出檔。

可以使用 iSeries 指令行或 Qshell 環境來執行 JPDC。如需詳細資料，請參閱範例：執行 Java 效能資料轉換器。

範例：執行 Java 效能資料轉換器:

您可以使用 iSeries 指令行或 Qshell 環境來執行「Java 效能資料轉換器 (JPDC)」。

使用 iSeries 指令行：

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

1. 在 iSeries 指令行，輸入「執行 Java (RUNJVA)」指令或 JAVA 指令。
2. 在類別參數行輸入 `com.ibm.as400.jpdc.JPDC`。
3. 在參數行輸入 `general pexdfn mydir/myfile myrdbdire`。
4. 在類別路徑參數行輸入 `'/QIBM/ProdData/Java400/ext/JPDC.jar'`。

   註: 若 CLASSPATH 環境變數已包含 `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 字串，則可以省略類別路徑。可以使用「新增環境變數 (ADDENVVAR)」指令、「變更環境變數 (CHGENVVAR)」指令或「處理環境變數 (WRKENVVAR)」指令，將此字串加入 CLASSPATH 環境變數中。

使用 Qshell 環境：

1. 輸入「啓動 Qshell (STRQSH)」指令來啓動 Qshell 直譯器。
2. 在指令行輸入：

```
java -classpath /QIBM/ProdData/Java400/ext/JPDC.jar com.ibm.as400/jpdc/JPDC
jinsight pexdfn mydir/myfile myrdbdire
```

   註: 若現行環境中已加入 `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 字串，則可以省略類別路徑。可以使用 ADDENVVAR、CHGENVVAR 或 WRKENVVAR 指令，將此字串加入現行環境中。

如需背景資訊，請參閱執行 Java 效能資料轉換器。

IBM Developer Kit for Java 的指令及工具

使用 IBM Developer Kit for Java 時，您可以搭配 Qshell 直譯器來使用 Java 工具，或使用 CL 指令。

若您已有 Java 程式設計的經驗，您可能會比較喜歡使用「Qshell 直譯器 Java」工具，因爲這些工具很類似 Sun Microsystems 公司的Java Development Kit。如需 Qshell 環境的使用資訊，請參閱 Qshell 直譯器。

若您是一位 iSeries 程式設計師，可能會使用 iSeries 伺服器環境中常見的 Java CL 指令。如需使用 CL 指令及「iSeries 領航員」指令的相關資訊，請繼續閱讀以下內容。

您可以將 IBM Developer Kit for Java 與下列任何指令及工具搭配使用：

- Qshell 環境包含通常爲程式開發所必備的 Java 開發工具。
- CL 環境包含 CL 指令，適用於 Java 程式的最佳化及管理。
- 第 352 頁的『Java 支援的 iSeries 領航員指令』亦可建立及執行最佳化的 Java 程式。

IBM Developer Kit for Java 支援的 Java 工具

IBM Developer Kit for Java 支援下列工具。

除少數例外情形，Java 工具 (ajar 工具除外) 支援 Sun Microsystems 公司規定的語法及選項。全部都必須透過「Qshell 直譯器」來執行。

您可以使用「啓動 Qshell (STRQSH 或 QSH)」指令來啓動「Qshell 直譯器」。 當「Qshell 直譯器」執行時，會出現「QSH 輸入指令」畫面。畫面中會顯示 Qshell 下執行之 Java 工具及程式產生的所有輸出及訊息。對 Java 程式的任何輸入，也都讀取自此畫面。如需相關資訊，請參閱 Qshell 的 Java 指令。

註: Qshell 內無法直接使用 iSeries 指令輸入的功能。若要使用指令行，請按 F21 (CL 輸入指令)。

Java 工具

如需 Java 工具的說明，請參閱下列主題。

Java ajar 工具:

ajar 工具是 jar 工具的另一種替代介面，可用來建立及操作 Java ARchive (JAR) 檔。您可以使用 ajar 工具來操作 JAR 檔與 ZIP 檔。

ajar 工具可像 jar 工具一樣，列出 JAR 檔的內容、從 JAR 檔擷取、建立新的 JAR 檔，而且支援許多種 ZIP 格式。此外，ajar 工具還支援在現有的 JAR 檔中新增及刪除檔案。

在「Qshell 直譯器」中即可使用 ajar 工具。如需相關資訊，請參閱 ajar - 替代的 Java Archive。

Java appletviewer 工具:

Java appletviewer 工具不需要 Web 瀏覽器，就可讓您執行 Applet。此工具相容於 Sun Microsystems, Inc. 所提供的 appletviewer 工具。

若要執行 appletviewer 工具，您需要使用 Native Abstract Window Toolkit (NAWT)，此外還須使用 sun.applet.AppletViewer 類別，或在「Qshell 直譯器」中執行 appletviewer 工具。

以下爲使用 sun.applet.AppletViewer 類別及執行 TicTacToe 示範程式的範例。有關如何載入示範程式的相關資訊，請參閱如何擷取範例檔。

請在指令行輸入：

```
cd '/home/MyUserID/demo/applets/TicTacToe'
```

若爲 JDK 1.3，請發出下列指令：

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
PROP((os400.class.path.rawt 2)(java.version 1.3))
```

若爲 JDK 1.4，請發出下列指令：

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.4))
```

| 若爲 JDK 1.5，請發出下列指令：

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.5))
```

以下是在「Qshell 直譯器」中使用 appletviewer 工具及執行 TicTacToe 示範程式的範例。有關如何載入示範程式的相關資訊，請參閱如何擷取範例檔。

相對應的命令如下：

```
cd /home/MyUserID/demo/applets/TicTacToe
```

若爲 JDK 1.3，請發出下列指令：

```
Appletviewer -J-Dos400.class.path.rawt=2 -J-Djava.version=1.3 example1.html
```

若爲 JDK 1.4，請發出下列指令：

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.4 example1.html
```

| 若爲 JDK 1.5，請發出下列指令：

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.5 example1.html
```

註: -J 是 Appletviewer 的執行時間旗標。-D 爲內容。

如需 appletviewer 工具的相關資訊，請參閱 Sun Microsystems 公司的 appletviewer 工具 

如何擷取範例檔:

下列程序顯示如何在執行 Java appletviewer 工具之前擷取範例檔。此程序假設您希望將範例檔擷取至起始目錄內：

1. 在指令行輸入「啓動 Qshell (QSH)」指令。
2. 爲您的使用者 ID 建立起始層次整合檔案系統 (IFS) 目錄 (若尚無此目錄)：

   ```
   mkdir /home/MyUserID
   ```
3. 在 IFS 目錄內建立示範程式目錄：

   ```
   mkdir /home/MyUserID/demo
   ```
4. 將目錄切換至示範程式目錄：

   ```
   cd /home/myUserId/demo
   ```
5. 若爲 JDK 1.3，請在指令行輸入下列指令以擷取示範程式檔：

   ```
   jar xf /QIBM/ProdData/Java400/jdk13/demo.zip
   ```

   若爲 JDK 1.4，請使用此指令：

   ```
   jar xf /QIBM/ProdData/Java400/jdk14/demo.jar
   ```

若爲 JDK 1.5，請使用此指令：

```
jar xf /QIBM/ProdData/Java400/jdk15/demo.jar
```

Java apt 工具:

Java apt 工具可處理程式註解。

apt 工具僅由 JDK 1.5 及後續版本提供。在「Qshell 直譯器」中即可使用 apt 工具。

如需 apt 工具的相關資訊，請參閱 Sun Microsystems 公司的 apt 工具 

Java extcheck 工具:

在 Java 2 SDK (J2SDK) 標準版 1.2 版及更高版本中，extcheck 工具可偵測目標 JAR 檔與目前已安裝的延伸 JAR 檔之間是否發生版本衝突。此工具相容於 Sun Microsystems 公司所提供的 keytool。

在「Qshell 直譯器」中即可使用 extcheck 工具。

有關 extcheck 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 extcheck 工具。

Java idlj 工具:

idlj 工具可以從給定的「介面定義語言 (IDL)」檔案中產生 Java 連結。

idlj 工具亦稱爲 IDL-Java 編譯器。此工具相容於 Sun Microsystems, Inc. 所提供的 idlj 工具。此工具僅適用於 Java Development Kits 1.3 及後續版本。

如需 idlj 工具的相關資訊，請參閱 Sun Microsystems 公司的 idlj 工具 

Java jar 工具:

jar 工具可將多個檔案結合成爲單一 Java ARchive (JAR) 檔。此工具相容於 Sun Microsystems 公司所提供的 jar 工具。

在「Qshell 直譯器」中即可使用 jar 工具。

有關 jar 工具的替代介面，請參閱 ajar 工具來建立及操作 JAR 檔。

如需 iSeries 檔案系統的相關資訊，請參閱整合檔案系統或整合檔案系統中的檔案。

有關 jar 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 jar 工具。

Java jarsigner 工具:

在 Java 2 SDK (J2SDK) 標準版 1.2 版及更高版本中，jarsigner 工具可以簽署 JAR 檔及驗證已簽署之 JAR 檔的簽章。

當 jarsigner 工具需要尋找私密金鑰來簽署 JAR 檔時，就會存取由 keytool 所建立及管理的金鑰庫。在 J2SDK 中，jarsigner 與 keytool 工具會取代 javakey 工具。此工具相容於 Sun Microsystems, Inc. 所提供的 jarsigner 工具。

在「Qshell 直譯器」中即可使用 jarsigner 工具。

有關 jarsigner 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 jarsigner 工具。

Java javac 工具:

javac 工具可以編譯 Java 程式。除了一項例外，此工具大部份都相容於 Sun Microsystems, Inc. 所提供的 javac 工具。

-classpath
不會置換預設的類別路徑。而是附加至系統預設類別路徑後面。但 -classpath 選項會置換 CLASSPATH 環境變數。

在「Qshell 直譯器」中即可使用 javac 工具。

若您在 iSeries 伺服器上已安裝 JDK 1.1.x 作爲預設值，但需要從 1.2 版或更高版本中執行 java 指令，請輸入下列指令：

```
javac -djava.version=1.2 <my_dir> MyProgram.java
```

有關 javac 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 javac 工具。

Java javadoc 工具:

javadoc 工具可以產生 API 文件。此工具相容於 Sun Microsystems, Inc. 所提供的 javadoc 工具。

在「Qshell 直譯器」中即可使用 javadoc 工具。

有關 javadoc 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 javadoc 工具。

Java javah 工具:

javah 工具有助於實作 Java 原生方法。除了少數例外，此工具大部分都相容於 Sun Microsystems 公司所提供的 javah 工具。

註: 編寫原生方法表示您的應用程式不是 100% Pure Java。亦表示應用程式無法直接跨越不同的平台。本質上，原生方法針對於特定的平台或系統。使用原生方法可能會增加應用程式的開發及維護成本。

在「Qshell 直譯器」中即可使用 javah 工具。此工具會讀取 Java 類別檔案，然後在現行工作目錄中建立 C 語言標頭檔。編寫的標頭檔是一種「iSeries 串流檔 (STMF)」。必須先複製到檔案成員中，才能在 iSeries 伺服器的 C 程式內併入。

javah 工具相容於 Sun Microsystems, Inc. 所提供的工具。然而，若指定了下列選項，iSeries 伺服器會加以忽略。

-td iSeries 伺服器上的 javah 工具不需要暫時目錄。

-stubs iSeries 伺服器上的 Java 僅支援「Java 原生介面 (JNI)」形式的原生方法。僅 JNI 之前的原生方法形式才需要存根。

-trace 與 .c Stub 檔輸出相關，iSeries 伺服器上的 Java 並不支援。

-v 不支援。

註: 一律必須指定 -jni 選項。iSeries 伺服器不支援 JNI 之前的原生方法實作方式。

有關 javah 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 javah 工具。

Java javap 工具:

javap 工具可以分解已編譯的 Java 檔案，並且印出 Java 程式的表示法。當系統上沒有最原始的原始程式碼可用時，它或許有所幫助。

-b 此選項會被忽略。不需要與舊版本相容，因爲 iSeries 伺服器上的 Java 僅支援 Java Development Kit (JDK) 1.1.4 與以上的版本。

-p 在 iSeries 伺服器上，-p 不是有效的選項。您必須完整拼出 -private。

-verify 此選項會被忽略。javap 工具在 iSeries 伺服器上不執行驗證。

在「Qshell 直譯器」中就可以使用 javap 工具。

註: 使用 javap 工具來分解類別，可能會違反這些類別的授權合約。使用 javap 工具之前，請先閱讀類別的授權合約。

如需 javap 工具的相關資訊，請參閱 Sun Microsystems 公司的 javap 工具。

Java keytool:

在 Java 2 SDK (J2SDK) 標準版 1.2 版或更高版本中，keytool 會建立公開與私密金鑰對、自簽憑證及管理金鑰庫。在 J2SDK 中，jarsigner 與 keytool 工具會取代 javakey 工具。此工具相容於 Sun Microsystems 公司所提供的 keytool。

在「Qshell 直譯器」中就可以使用 keytool。

有關 keytool 的詳細資訊，請參閱 Sun Microsystems, Inc. 的 keytool。

Java native2ascii 工具:

native2ascii 工具可以將含有原始編碼字元 (非 Latin 1 及非 Unicode 的字元) 的檔案，轉換成含有 Unicode 編碼字元的檔案。此工具相容於 Sun Microsystems, Inc. 所提供的 native2ascii 工具。

在「Qshell 直譯器」中即可使用 native2ascii 工具。

有關 native2ascii 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 native2ascii 工具。

Java orbd 工具:

orbd 工具支援用戶端在 CORBA 環境下直接尋找及呼叫伺服器上的持續性物件。

ORBD 包含「暫時性名稱服務」及「持續性名稱服務」，可用來代替「暫時性名稱服務 (tnameserv)」。orbd 工具將「伺服器管理程式」、「互用性命名服務」及「啓動名稱伺服器」的功能納入其中。若與 servertool 一起使用，當用戶端要存取伺服器時，「伺服器管理程式」就會尋找、登記及啓動伺服器。

| 有關 orbd 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 orbd 工具。

| **Java pack200 工具:**

| pack200 工具是可將 JAR 檔壓縮爲 pack200 檔案的 Java 應用程式。

| pack200 工具僅適用於 JDK 1.5 及後續版本。在「Qshell 直譯器」中即可使用 pack200 工具。

| 如需相關資訊，請參閱 Sun Microsystems 公司的 pack200 工具

| **相關概念**

| 第 350 頁的『Java unpack200 工具』
| Java unpack200 工具將 pack200 檔案解壓縮至 JAR 檔。

Java policytool:

在 Java 2 SDK 標準版中，policytool 可以建立及變更用來定義安裝環境 Java 安全原則的外部原則配置檔。此工具相容於 Sun Microsystems 公司所提供的 policytool。

policytool 是一種在「Qshell 直譯器」及 Native Abstract Window Toolkit (NAWT) 中可以使用的圖形式使用者介面 (GUI) 工具。如需相關資訊，請參閱 IBM Developer Kit for Java Native Abstract Window Toolkit。

有關 policytool 的詳細資訊，請參閱 Sun Microsystems, Inc. 的 policytool。

Java rmic 工具:

rmic 工具可產生 Java 物件的 stub 檔及類別檔案。此工具相容於 Sun Microsystems, Inc. 所提供的 rmic 工具。

在「Qshell 直譯器」中即可使用 rmic 工具。

有關 rmic 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 rmic 工具。

Java rmid 工具:

在 Java 2 SDK (J2SDK) 標準版中，rmid 工具會起始「啓動系統常駐程式」，因此可在 Java 虛擬機器中登記及啓動物件。此工具相容於 Sun Microsystems, Inc. 所提供的 rmid 工具。

在「Qshell 直譯器」中即可使用 rmid 工具。

有關 rmid 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 rmid 工具。

Java rmiregistry 工具:

rmiregistry 工具可在指定的埠上進行遠端物件登錄。此工具相容於 Sun Microsystems, Inc. 所提供的 rmiregistry 工具。

在「Qshell 直譯器」中即可使用 rmiregistry 工具。

有關 rmiregistry 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 rmiregistry 工具。

Java serialver 工具:

serialver 工具可傳回一或多個類別的版本號碼或唯一的序列化 ID。此工具相容於 Sun Microsystems, Inc. 所提供的 serialver 工具。

在「Qshell 直譯器」中即可使用 serialver 工具。

有關 serialver 工具的詳細資訊，請參閱 Sun Microsystems, Inc. 的 serialver 工具。

Java servertool:

servertool 提供指令行介面，供應用程式設計師登記、取消登記、啓動及關閉持續性伺服器。

有關 servertool 的詳細資訊，請參閱 Sun Microsystems, Inc. 的 servertool。

Java tnameserv 工具:

在 Java 2 SDK (J2SDK) 標準版 1.3 版或更高版本中，tnameserv (暫時性名稱服務) 工具可提供名稱服務的存取。此工具相容於 Sun Microsystems, Inc. 所提供的 tnameserv 工具。

在「Qshell 直譯器」中即可使用 tnameserv 工具。

Java unpack200 工具:

Java unpack200 工具將 pack200 檔案解壓縮至 JAR 檔。

僅在 JDK 1.5 及後續版本中提供 unpack200 工具。在「Qshell 直譯器」中即可使用 unpack200 工具。

如需相關資訊，請參閱 Sun Microsystems 公司的 unpack200 工具 

相關概念

第 348 頁的『Java pack200 工具』

pack200 工具是可將 JAR 檔壓縮爲 pack200 檔案的 Java 應用程式。

Qshell 的 Java 指令

Qshell 中的 java 指令可以執行 Java 程式。除了少數例外，大部分相容於 Sun Microsystems 公司所提供的 java 工具。

IBM Developer Kit for Java 會忽略 Qshell 之 java 指令的下列選項。

選項	說明
-cs	不支援此選項。
-checksource	不支援此選項。
-debug	此選項受 iSeries 內部除錯程式支援。
-noasyncgc	IBM Developer Kit for Java 一律執行垃圾收集。
-noclassgc	IBM Developer Kit for Java 一律執行垃圾收集。
-prof	iSeries 伺服器有自己的效能工具。
-ss	此選項不適用於 iSeries 伺服器。
-oss	此選項不適用於 iSeries 伺服器。
-t	iSeries 伺服器使用自己的追蹤功能。
-verify	在 iSeries 伺服器上一律驗證。
-verifyremote	在 iSeries 伺服器上一律驗證。
-noverify	在 iSeries 伺服器上一律驗證。

在 iSeries 伺服器上，-classpath 選項不會置換預設類別路徑。而是附加至系統預設類別路徑後面。但 -classpath 選項會置換 CLASSPATH 環境變數。

Qshell 的 java 指令可支援 iSeries 伺服器的新選項。以下列出支援的新選項。

選項	說明
-chkpath	此選項會在 CLASSPATH 中檢查目錄的公用寫入權。
-opt	此選項指定最佳化等級。

選項	說明
-Xrun[:]	顯示訊息，指出 JVM 啓動期間 JVM_OnLoad 函數的服務程式及選用性參數字串。
-agentlib:	指出含有 VM 代理程式的 i5/OS 服務程式。VM 在啓動期間會試圖從檔案庫清單內的 i5/OS 檔案庫中載入服務程式。
-agentpath:	從這個選項後面的絕對路徑中載入檔案庫。啓動時，檔案庫名稱不會展開，選項會傳遞至代理程式。
-javaagent:<jarpath>[=<options>]	載入 Java 程式設計語言代理程式，以與 java.lang.instrument 套件搭配使用。 *jarpath* 是代理程式 JAR 檔的路徑。*options* 是代理程式選項。您可在相同指令行上多次使用 -javaagent:<jarpath>[=<options>]，以建立多個代理程式。多個代理程式可使用相同的 *jarpath*。

CL 指令參考資訊中的「執行 Java (RUNJVA)」指令，將詳細說明這些新的選項。「建立 Java 程式 (CRTJVAPGM)」指令、「刪除 Java 程式 (DLTJVAPGM)」指令及「顯示 Java 程式 (DSPJVAPGM)」指令的 CL 指令參考資訊，含有如何管理 Java 程式的相關資訊。

在「Qshell 直譯器」中即可使用 Qshell 的 `java` 指令。

如須 Qshell 的 `java` 指令相關資訊，請參閱 Sun Microsystems, Inc. 的 java 工具。

Java 支援的 CL 指令

IBM Developer Kit for Java 支援這些 CL 指令。

- 「分析 Java 程式 (ANZJVAPGM)」指令可分析 Java 程式、列出其類別及顯示每一個類別的現行狀態。
- 「分析 Java 虛擬機器 (ANZJVM)」指令 可在 Java 虛擬機器 (JVM) 中擷取及設定資訊。此指令會傳回作用中類別的相關資訊，協助您進行 Java 程式除錯。
- 「變更 Java 程式 (CHGJVAPGM)」指令可變更 Java 程式的屬性。
- 「建立 Java 程式 (CRTJVAPGM)」指令可透過 Java 類別檔案、ZIP 檔或 JAR 檔，在 iSeries 伺服器上建立 Java 程式。
- 「刪除 Java 程式 (DLTJVAPGM)」指令可刪除與 Java 類別檔案、ZIP 檔或 JAR 檔相關的 iSeries Java 程式。
- 「顯示 Java 程式 (DSPJVAPGM)」指令可顯示 iSeries 上某個 Java 程式的相關資訊。
- 「顯示 Java 虛擬機器工作 (DSPJVMJOB)」指令可顯示作用中 JVM 工作的相關資訊，協助您管理暫時修訂程式 (PTF) 的應用。您亦可在第 495 頁的『套用暫時修訂程式』中找到 DSPJVMJOB 的詳細資訊。
- 「傾出 Java 虛擬機器 (DMPJVM)」指令可針對某項指定的工作，將 Java 虛擬機器的相關資訊傾出至排存印表機檔案。
- JAVA 指令及「執行 Java (RUNJVA)」指令均可執行iSeries Java 程式。

如需詳細資訊，請參閱以下各頁：

使用 ANZJVM 指令的注意事項

授權內碼選項參數字串

程式及 CL 指令 API

使用 ANZJVM 指令的注意事項

由於 ANZJVM 執行時間長度的緣故，很可能在 ANZJVM 完成之前，JVM 就已先結束。若 JVM 結束，ANZJVM 會傳回 JVAB606 訊息 (指出 JVM 在處理 ANZJVM 時結束)，以及它所取得的資料。

JVM 可處理的類別亦無數量上限。若類別太多而無法處理，ANZJVM 會傳回可處理的資料，並一併傳回訊息，讓您知道還有其他資訊尚未回報。當資料需要截斷時，ANZJVM 會儘可能傳回所有資訊。

內部參數的時間長度限制爲 3600 秒 (1 小時)。ANZJVM 可傳回多少類別的相關資訊，則受限於系統的儲存體數量。

Java 支援的 iSeries 領航員指令

「iSeries 領航員」爲適用於 Windows 桌面的圖形式介面。它屬於 iSeries Access for Windows 的元件，其中囊括許多重要的 iSeries 功能，可供管理者或使用者完成日常工作。

「iSeries 領航員」在 iSeries Access for Windows 的「檔案系統」選項中，以外掛程式的方式支援 Java。若要使用「iSeries 領航員」Java 外掛程式，您必須在 iSeries 伺服器上安裝 IBM Developer Kit for Java。然後，透過「選擇性安裝」，在 Client Access 資料夾中選取「檔案系統」，將 Java 外掛程式安裝在個人電腦上。

類別、JAR、ZIP 及 Java 檔案皆位於整合檔案系統中。「iSeries 領航員」可讓您在右窗格中看到這些檔案。以滑鼠右鍵按一下您要使用的類別、JAR、ZIP 或 java 檔案。 就會顯示快速功能表。

從快速功能表中，選取**相關聯的 Java 程式 --> 新建...** 來啓動 Java 轉換程式，即可建立與類別、JAR 或 ZIP 檔相關聯的 iSeries Java 程式。接著會出現一個對話框，讓您指定程式建立方式的詳細資料。可以選擇採取 Java 轉換或 Java 解譯的方式來建立程式。

附註：若選擇轉換，則類別檔案中的位元組碼會轉換成 RISC 指令，其效能會比選擇解譯來得好。

從快速功能表中選取**相關聯的 Java 程式 --> 編輯...**，可以變更已附加至 Java 類別檔案、ZIP 檔或 JAR 檔的 Java 程式屬性。

從快速功能表中選取**相關聯的 Java 程式 --> 執行...**，可以在 iSeries 伺服器上執行類別檔案。您也可以選取 JAR 或 ZIP 檔，然後執行此 JAR 或 ZIP 檔內含的類別檔案。執行後會出現一個對話框，讓您指定程式執行方式的詳細資料。若已選取**相關聯的 Java 程式 --> 新建...**，則執行程式時，將使用與類別檔案相關聯的iSeries Java 程式。如果某個 iSeries Java 程式尚未連結您的類別檔案，則執行程式之前會先建立 iSeries Java 程式。

從快速功能表中選取**相關聯的 Java 程式 --> 刪除...**，可以刪除與類別、JAR 或 ZIP 檔相關聯的 iSeries Java 程式。

從快速功能表中選取**內容**，即可顯示內容對話框，內含 **Java 程式**及 **Java 選項**標籤。藉由這些標籤所提供的詳細資料，您將瞭解到與類別、JAR 或 ZIP 檔相關聯的 iSeries Java 程式是如何建立的。

附註：以上畫面皆爲「顯示 Java 程式」資訊。

從快速功能表中選取**編譯 Java 檔案**，可將已選取的任何 java 檔案，轉換成類別檔案位元組碼。

如需**新建 Java 程式**、**編輯 Java 程式**、**執行 Java 程式**、**Java 程式**、**Java 選項**、**編譯 Java 檔案**及**刪除 Java 程式**這些「iSeries 領航員」對話框的參數及選項相關資訊，請參閱「iSeries 領航員」提供的說明資訊。

對伺服器上執行的 Java 程式進行除錯

對於伺服器上執行的 Java 程式，您有數種選擇可進行除錯及疑難排解，包括 IBM iSeries 系統除錯程式、伺服器互動式顯示畫面、Java Debug Wire Protocol 除錯程式及 Java Watcher。

下列資訊並非所有可能性的綜合性評定，但它列出了幾個選項供您選擇。對於 iSeries 伺服器上執行的 Java 程式，其中一種最簡單的除錯方式就是使用「IBM iSeries 系統除錯程式」。「IBM iSeries 系統除錯程式」提供圖形式使用者介面 (GUI)，可讓您更輕鬆地發揮 iSeries 伺服器的除錯能力。

雖然「iSeries 系統除錯程式」提供了較簡單易用的 GUI 讓您進行 Java 程式除錯，您還是可以使用伺服器的互動式顯示畫面來執行同樣的功能。

此外，iSeries Java 虛擬機器 (JVM) 支援 Java Debug Wire Protocol (JDWP)，其屬於 Java Platform Debugger Architecture。JDWP 除錯程式讓您可以從執行不同作業系統的用戶端，執行遠端除錯。(「IBM iSeries 除錯程式」亦可依照類似方式執行遠端除錯，只是不使用 JDWP)。Eclipse Project 通用工具平台的 Java 除錯程式，就是一種 JDWP 程式。

若您的程式經過長時間執行之後效能會下降，則表示記憶體洩漏方面的程式碼編寫可能有誤。您可以使用 Java Watcher 來協助進行程式除錯，透過執行一段時間的 Java 應用程式資料堆分析及物件建立效能分析，找出記憶體洩漏之處。

IBM iSeries 系統除錯程式

第 361 頁的『Java Platform Debugger Architecture』
Java Platform Debugger Architecture (JPDA) 由 JVM 除錯介面/JVM 工具介面、Java Debug Wire Protocol 及 Java 除錯介面組成。JPDA 的所有部分都可讓任何使用 JDWP 的除錯程式前端系統去執行除錯作業。除錯程式前端系統可以在遠端執行，或以 iSeries 應用程式的形式執行。

Java 開發工具除錯

Eclipse Project 網站

JavaWatcher

從 i5/OS 指令行進行 Java 程式除錯

若要從 i5/OS 指令行進行 Java 程式除錯，請從此處列出的選項中選取一項。

- 進行 Java 程式除錯
- 對 Java 及原生方法程式進行除錯
- 從另一個顯示畫面進行 Java 程式除錯
- 對透過自訂類別載入器所載入的 Java 類別進行除錯
- Servlet 除錯

當您進行 Java 程式除錯時，Java 程式實際上是以批次即時 (BCI) 工作的形式於 Java 虛擬機器中執行。原始程式碼會顯示在互動式顯示畫面中，但 Java 程式並非在此處執行。而是在另一項工作中執行，也就是服務工作。當 Java 程式結束時，服務工作也就結束，而且會顯示**正在服務的工作已結束**訊息。

您無法對即時 (JIT) 編譯器所執行的 Java 程式進行除錯。若檔案沒有相關的 Java 程式，則預設爲執行 JIT。您可以用幾種方法停用執行 JIT，來容許除錯：

- 啓動 Java 虛擬機器時指定 `java.compiler=NONE` 內容。
- 在「執行 Java (RUNJVA)」指令上指定 OPTION(*DEBUG)。
- 在「執行 Java (RUNJVA)」指令上指定 INTERPRET(*YES)。
- 啓動 Java 虛擬機器之前，先使用 CRTJVAPGM OPTIMIZATION(10) 來建立相關聯的 Java 程式。

註: 以上解決方案皆不影響正在執行的 Java 虛擬機器。若 Java 虛擬機器不是使用以上其中一種方法來啓動，則必須停止並重新啓動，才能夠進行除錯。

在「執行 Java (RUNJVA)」指令上指定 *DEBUG 選項，就會建立兩個工作之間的介面。

如需系統除錯程式的相關資訊，請參閱 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712-04 及線上說明資訊。

對 Java 程式進行除錯

對於 iSeries 伺服器上執行的 Java 程式，最簡單的除錯方式就是使用「IBM iSeries 系統除錯程式」。「IBM iSeries 系統除錯程式」提供圖形式使用者介面，可讓您更輕鬆地發揮 iSeries 伺服器的除錯能力。

如需使用「iSeries 系統除錯程式」對在 iSeries 伺服器上執行之 Java 程式進行除錯及測試的相關資訊，請參閱 IBM iSeries 系統除錯程式。

若有必要，您可以在執行程式之前，利用伺服器的互動式顯示畫面來使用 *DEBUG 選項，以檢視原始碼。然後，即可設定岔斷點、或者在程式執行期間，逐程序或逐步地執行程式來分析錯誤。

若要對 Java 程式進行除錯，請遵循下列步驟：

1. 利用 DEBUG 選項來編譯 Java 程式，亦即在 javac 工具上使用 -g 選項。如需相關資訊，請參閱使用 *DEBUG 選項進行 Java 程式除錯。
2. 在 iSeries 伺服器上，將類別檔案 (.class) 與來源檔 (.java) 放入相同目錄中。
3. 在 iSeries 指令行上，透過「執行 Java (RUNJVA)」指令來執行 Java 程式。在「執行 Java (RUNJVA)」指令上指定 OPTION(*DEBUG)。

   只能對一個類別進行除錯。若在 CLASS 關鍵字上輸入 JAR 檔名，則不支援 OPTION(*DEBUG)。
4. 畫面上會顯示 Java 程式原始碼。
5. 按 F6 (新增/清除岔斷點) 來設定岔斷點，或按 F10 (逐步) 來逐步執行程式。有關設定岔斷點的資訊，請參閱設定岔斷點。如需逐步除錯的詳細資料，請參閱逐步執行 Java 程式除錯。

要訣：

1. 使用岔斷點及逐步除錯時，請檢查 Java 程式的邏輯流程，必要時檢視及變更變數。
2. 在 RUNJVA 指令上使用 OPTION(*DEBUG) 來停用即時編譯器 (JIT)。沒有相關 Java 程式的檔案，則在解譯模式下執行。

使用 *DEBUG 選項進行 Java 程式除錯:

執行程式之前，請使用 *DEBUG 選項來檢視原始程式碼。*DEBUG 選項可讓您在程式碼內設定岔斷點。

若要使用 *DEBUG 選項，請在指令行輸入「執行 Java (RUNJVA)」指令，隨後輸入類別檔案的名稱及 OPTION(*DEBUG)。例如，iSeries 指令行應類似於：

```
RUNJVA CLASS(classname) OPTION(*DEBUG)
```

註: 如果您未獲授權使用「啓動服務工作 (STRSRVJOB)」指令，則會忽略 OPTION(*DEBUG)。

若要檢視除錯顯示畫面，請參閱 Java 程式的起始除錯顯示畫面。

對於 iSeries 伺服器上執行的 Java 程式，最簡單的除錯方式就是使用「IBM iSeries 系統除錯程式」。「IBM iSeries 系統除錯程式」提供圖形式使用者介面，可讓您更輕鬆地發揮 iSeries 伺服器的除錯能力。

如需使用「iSeries 系統除錯程式」對在 iSeries 伺服器上執行之 Java 程式進行除錯及測試的相關資訊，請參閱 IBM iSeries 系統除錯程式。

Java 程式的起始除錯顯示畫面:

當您對 Java 程式進行除錯時，請遵循程式的下列範例顯示畫面進行。這些畫面將顯示一個名爲 Hellod 的範例程式。

- 輸入 `ADDENVVAR ENVVAR(CLASSPATH) VALUE ('/MYDIR')`。
- 輸入此指令：`RUNJVA CLASS(HELLOD) OPTION(*DEBUG)`。插入您的 Java 程式名稱來取代 HELLOD。
- 等待「顯示模組原始檔」畫面出現。以下爲 HELLOD Java 程式的原始檔。

```
+------------------------------------------------------------------------------+
|                            顯示模組原始檔                                    |
|                                                                              |
|  類別檔案名稱：      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()                                                     |
|                                                                    尚有...   |
|  除錯 . . .                                                                 |
|                                                                              |
|  F3=結束程式       F6=新增/清除岔斷點         F10=逐步    F11=顯示變數          |
|  F12=回復          F17=監視變數          F18=使用監視          F24=其餘鍵       |
|                                                                              |
+------------------------------------------------------------------------------+
```

- 按 F14 (使用模組清單)。
- 此時會出現「使用模組清單」畫面。您可以輸入選項 1 (新增程式) 來加入其他要除錯的類別及程式。使用選項 5 (顯示模組原始檔) 可顯示原始檔。

```
+------------------------------------------------------------------------------+
|                                使用模組清單                                  |
|                                                        系統：     AS400      |
|  請鍵入選項，然後按 Enter 鍵。                                               |
|    1=新增程式       4=移除程式          5=顯示模組原始檔                     |
|    8=使用模組岔斷點                                                          |
|                                                                              |
|  選項     程式/模組          檔案庫            類型                          |
|                              *LIBL             *SRVPGM                       |
|           HELLOD                               *CLASS       已選取          |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                    底端      |
|  指令                                                                        |
```

```
|  ===>                                                                          |
|  F3=跳出    F4=提示      F5=重整       F9=擷取        F12=取消                   |
|  F22=顯示類別檔案名稱                                                          |
|                                                                                |
+--------------------------------------------------------------------------------+
```

- 在新增要除錯的類別時，您輸入的完整套件類別名稱，可能會超過「程式/模組」輸入欄位的長度。若要輸入較長的名稱，請遵循下列步驟：
  1. 輸入「選項 1」(新增程式)。
  2. 將「程式/模組」欄位留白。
  3. 將「檔案庫」欄位保持爲 *LIBL。
  4. 在「類型」中輸入 *CLASS。
  5. 按 Enter 鍵。
  6. 畫面上會顯示蹦現視窗，讓您有更大的空間可輸入完整的套件類別檔案名稱。

設定岔斷點:

岔斷點可以控制程式的執行。岔斷點會讓執行中的程式停止在特定的陳述式上。

請執行下列步驟來設定岔斷點：

1. 將游標移至想要設定岔斷點的程式碼行上。
2. 按 F6 (新增/清除岔斷點) 來設定岔斷點。
3. 按 F12 (回復) 來執行程式。

註: 在已設定岔斷點之程式碼行執行的前一刻，畫面上會顯示程式原始碼，指出已觸及岔斷點。

```
+--------------------------------------------------------------------------------+
|                              顯示模組原始檔                                    |
|                                                                                |
|現行緒：      00000019                停止緒：     00000019                     |
|類別檔案名稱：      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();                                                            |
|除錯 . . .                                                                     |
|                                                                                |
|  F3=結束程式       F6=新增/清除岔斷點         F10=逐步    F11=顯示變數            |
|  F12=回復          F17=監視變數          F18=使用監視          F24=其餘鍵       |
|岔斷點已新增至第 41 行。                                                        |
+--------------------------------------------------------------------------------+
```

觸及岔斷點時，若想要設定只在現行緒之內觸及的岔斷點，請使用 TBREAK 指令。

如需系統除錯程式指令的相關資訊，請參閱 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  及線上說明資訊。

如需程式於岔斷點停止執行時求出變數值的相關資訊，請參閱在 Java 程式中求出變數的值。

逐步執行 Java 程式除錯:

程式可以在除錯的同時逐步執行。您可以跳過，或進入其他函數。 Java 程式及原生方法皆可使用逐步方法。

程式原始碼一出現，就可以開始逐步執行。執行第一個陳述式之前，程式會先停下來。請按 F10 (逐步)。繼續按 F10 (逐步) 來逐步執行程式。按 F22 (逐步進入) 可以進入程式所呼叫的任何函數中逐步執行。每當觸及岔斷點時，亦可開始逐步執行。如需設定岔斷點的相關資訊，請參閱設定岔斷點。

```
+------------------------------------------------------------------------------+
|                            顯示模組原始檔                                    |
|                                                                              |
|現行緒：      00000019                停止緒：     00000019                   |
|類別檔案名稱：      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();                                                          |
|除錯 . . .                                                                    |
|                                                                              |
|  F3=結束程式       F6=新增/清除岔斷點         F10=逐步    F11=顯示變數          |
|  F12=回復          F17=監視變數          F18=使用監視          F24=其餘鍵      |
|於緒 00000019 第 42 行完成逐步除錯                                            |
+------------------------------------------------------------------------------+
```

若要停止逐步作業，讓程式繼續執行，請按 F12 (回復)。

如需逐步作業的相關資訊，請參閱 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  及線上說明資訊。

如需程式在一個步驟停止執行時求出變數值的相關資訊，請參閱在 Java 程式中求出變數的值。

在 Java 程式中求出變數的值:

當程式於岔斷點或逐步除錯上暫停執行時，有兩種方法可以求出變數的值。

- 選項 1：在除錯指令行輸入 `EVAL VariableName`。
- 選項 2：在顯示的原始程式碼中，將游標移至變數名稱上，然後按 F11 (顯示變數)。

使用 EVAL 指令在 Java 程式中求出變數的值。

註: 您也可使用 EVAL 指令來變更變數的內容。如需 EVAL 指令之變體的相關資訊，請參閱 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712 及線上說明資訊。

在查看 Java 程式的變數時，請注意下列幾點：

- 若您要求值的變數是 Java 類別的實例，則顯示畫面的第一行會顯示物件類型。也會顯示物件的 ID。第一行之後，將顯示物件中每一個欄位的內容。若變數爲空值，則第一行會指出變數是空值。星號顯示 (空值物件) 每一個欄位的內容。
- 若您要求值的變數是 Java String 物件，則顯示此字串的內容。若爲字串爲空值，則顯示空值。
- 您無法變更字串變數。
- 若您要求值的變數是陣列，則顯示 ARR，後面接著此陣列的 ID。您可以利用變數名稱的註標來求出陣列元素的值。若爲陣列爲空值，則顯示空值。
- 您無法變更陣列變數。但只要不是字串或物件陣列，仍可變更陣列的元素。
- 對於陣列變數，您可以指定 arrayname.length 來得知陣列中有多少元素。
- 對於類別欄位的變數，若要查看變數的內容，請指定 classvariable.fieldname。
- 若嘗試對尚未起始設定的變數求值，可能發生兩種情況之一。出現**無法顯示變數**的訊息，或直接顯示變數尚未起始設定的內容，有可能是很不尋常的值。

對 Java 及原生方法程式進行除錯

您可以同時對 Java 程式及原生方法程式進行除錯。當您在互動式畫面上進行原始檔除錯時，亦可同時對服務程式 (*SRVPGM) 中以 C 語言編寫的原生方法進行除錯。*SRVPGM 必須配合除錯資料來編譯及建立。

對 Java 程式及原生方法程式 (或服務程式) 進行除錯，最簡單的方法就是透過「IBM iSeries 系統除錯程式」。「IBM iSeries 系統除錯程式」在 iSeries 伺服器上提供了圖形式使用者除錯環境。如需使用「iSeries 系統除錯程式」對在 iSeries 伺服器上執行之程式進行除錯及測試的相關資訊，請參閱 IBM iSeries 系統除錯程式。

若要使用伺服器的互動式顯示畫面，同時對 Java 程式及原生方法程式進行除錯，請完成下列步驟：

1. 當您的 Java 程式原始碼出現時，請按 F14 (使用模組清單)，以顯示「使用模組清單 (WRKMODLST)」顯示畫面。
2. 選取選項 1 (新增程式)，新增您的服務程式。
3. 選取選項 5 (顯示模組原始碼)，顯示您要除錯的 *MODULE 及原始碼。
4. 按 F6 (新增/清除岔斷點)，在服務程式中設定岔斷點。有關設定岔斷點的資訊，請參閱設定岔斷點。
5. 按 F12 (回復) 來執行程式。

註: 當服務程式觸及岔斷點時，程式就會停止執行，然後顯示服務程式的原始碼。

從另一個顯示畫面進行 Java 程式除錯

對於 iSeries 伺服器上執行的 Java 程式，最簡單的除錯方式就是使用「IBM iSeries 系統除錯程式」。「IBM iSeries 系統除錯程式」提供圖形式使用者介面，可讓您更輕鬆地發揮 iSeries 伺服器的除錯能力。

如需使用「iSeries 系統除錯程式」對在 iSeries 伺服器上執行之 Java 程式進行除錯及測試的相關資訊，請參閱 IBM iSeries 系統除錯程式。

使用伺服器的互動式顯示畫面進行 Java 程式除錯時，一遇到岔斷點就會顯示程式原始碼。這可能會干擾 Java 程式的輸出畫面。爲避免干擾，請從另一個顯示畫面進行 Java 程式除錯。Java 程式的輸出會顯示在執行 Java 指令的地方，程式原始碼則顯示在另一個畫面上。

對於它在執行的 Java 程式，只要它不是正在使用即時 (JIT) 編譯器，一樣可用這種方式來除錯。

若要從另一個顯示畫面進行 Java 除錯，請執行下列動作：

1. 開始設定除錯時，Java 程式必須暫停。

讓 Java 程式暫停的方式如下：

- 等待鍵盤輸入。
- 等待一段時間。
- 循環測試變數，但需要設定值，讓 Java 程式最後可以跳離迴圈。

2. 當 Java 程式暫停之後，請至另一個顯示畫面執行下列步驟：
   a. 在指令行輸入「處理作用中的工作 (WRKACTJOB)」指令。
   b. 尋找 Java 程式執行所在的批次即時 (BCI) 工作。在「子系統/工作」清單中找出 QJVACMDSRV。在「使用者」清單中找出您的「使用者 ID」。再於「類型」中找出 BCI。
   c. 輸入選項 5 來處理此工作。
   d. 在「處理工作」顯示畫面頂端，會顯示「號碼」、「使用者」及「工作」。輸入 `STRSRVJOB Number/User/Job`。
   e. 輸入 `STRDBG CLASS(classname)`。Classname 是您要除錯的 Java 類別名稱。可以是 Java 指令上指定的類別名稱，或另一個類別的名稱。
   f. 該類別的原始檔會出現在「顯示模組原始檔」畫面中。
   g. 在該 Java 類別中您想要停止的地方，按 F6 (新增/清除岔斷點) 以設定岔斷點。按 F14 來新增您要除錯的其他類別、程式或服務程式。有關設定岔斷點的資訊，請參閱設定岔斷點。
   h. 按 F12 (回復) 來繼續執行程式。
3. 解除原始 Java 程式的暫停狀態。當觸及岔斷點時，在輸入「啓動服務工作 (STRSRVJOB)」指令及「啓動除錯 (STRDBG)」指令的畫面上，「顯示模組原始檔」會出現。當 Java 程式結束時，就會出現**正在服務的工作已結束**的訊息。
4. 輸入「結束除錯 (ENDDBG)」指令。
5. 輸入「結束服務工作 (ENDSRVJOB)」指令。

註: 在原始工作中啓動 Java 虛擬機器時，請務必停用「JIT 編譯器」。可設定 `java.compiler=NONE` 內容來停用。除錯時若同時執行 JIT，將可能發生無法預期的結果。

如需可控制 BCI 工作是否於呼叫 Java 虛擬機器之前先等待之變數的相關資訊，請參閱 QIBM_CHILD_JOB_SNDINQMSG 環境變數。

QIBM_CHILD_JOB_SNDINQMSG 環境變數:

對於 Java 虛擬機器執行所在的批次即時 (BCI) 工作，QIBM_CHILD_JOB_SNDINQMSG 環境變數可以控制此工作是否要等待 Java 虛擬機器啓動。

當「執行 Java (RUNJVA)」指令執行時，若將此環境變數設定爲 1，則會傳送訊息至使用者的訊息佇列。在 BCI 工作內啓動 Java 虛擬機器之前，會傳送訊息。此訊息如下：

```
Spawned (child) process 023173/JOB/QJVACMDSRV is stopped (G C)
```

請輸入 SYSREQ 並選取選項 4，即可檢視此訊息。

BCI 工作會靜待您回覆此訊息。回覆 (G) 會啓動 Java 虛擬機器。

在回覆訊息之前，可以先在 BCI 工作呼叫的 *SRVPGM 或 *PGM 中設定岔斷點。

註: Java 類別中無法設定岔斷點，因爲此時 Java 虛擬機器尚未啓動。

對透過自訂類別載入器所載入的 Java 類別進行除錯

對於 iSeries 伺服器上執行的 Java 程式，最簡單的除錯方式就是使用「IBM iSeries 系統除錯程式」。「IBM iSeries 系統除錯程式」提供圖形式使用者介面，可讓您更輕鬆地發揮 iSeries 伺服器的除錯能力。

如需使用「iSeries 系統除錯程式」對在 iSeries 伺服器上執行之 Java 程式進行除錯及測試的相關資訊，請參閱 IBM iSeries 系統除錯程式。

對於透過自訂類別載入器所載入的類別，若要使用伺服器的互動式顯示畫面來進行除錯，請完成下列步驟：

1. 將 DEBUGSOURCEPATH 環境變數設定爲原始程式碼所在的目錄，或者如果是套件限定的類別，則是套件名稱的起始目錄。

   比方說，如果自訂類別載入器所載入的類別位於 /MYDIR 目錄下，請執行下列動作：

   ```
   ADDENVVAR ENVVAR(DEBUGSOURCEPATH) VALUE('/MYDIR')
   ```

2. 從「顯示模組原始檔」畫面中，將類別加入除錯檢視畫面：

   若類別已載入 Java 虛擬機器 (JVM) 內，只要照常新增 *CLASS 並顯示原始程式碼來除錯即可。

   例如，若要檢視 pkg1/test14.class 的原始檔，請輸入：

   ```
   Opt     Program/module     Library        Type
   1       pkg1.test14_       *LIBL          *CLASS
   ```

   若類別尚未載入 JVM 內，請執行前述相同的步驟來新增 *CLASS。畫面上將出現**無法使用 Java 類別檔案**的訊息。此時，您可繼續程式處理。若輸入符合給定名稱的類別方法，JVM 就會自動停止。畫面上會顯示此類別的原始程式碼時，並可開始除錯。

Servlet 除錯

對於透過自訂類別載入器所載入的除錯類別，Servlet 除錯屬於特殊情況。Servlet 執行於 IBM HTTP Server 的 Java Runtime 中。我們有幾個選項可對 Servlet 進行除錯。

對於 iSeries 伺服器上執行的 Java 程式，最簡單的除錯方式就是使用「IBM iSeries 系統除錯程式」。「IBM iSeries 系統除錯程式」提供圖形式使用者介面，可讓您更輕鬆地發揮 iSeries 伺服器的除錯能力。

如需使用「iSeries 系統除錯程式」對在 iSeries 伺服器上執行之 Java 程式及 Servlet 進行除錯及測試的相關資訊，請參閱 IBM iSeries 系統除錯程式。

遵循透過自訂類別載入器所載入的類別的指示，也是另一種 Servlet 除錯方式。

您亦可使用伺服器的互動式顯示畫面來進行 Servlet 除錯，步驟如下：

1. 在 Qshell 直譯器中，使用 `javac -g` 指令來編譯 Servlet。
2. 將原始程式碼 (.java 檔) 及已編譯的程式碼 (.class 檔) 複製到 /QIBM/ProdData/Java400。
3. 對 .clas 檔執行「建立 Java 程式 (CRTJVAPGM)」指令，採用最佳化等級 10，亦即 OPTIMIZE(10)。
4. 啓動伺服器。
5. 在執行 Servlet 的工作上，執行「啓動服務工作 (STRSRVJOB)」指令。
6. 輸入 STRDBG CLASS(myServlet)，其中 myServlet 是您的 Servlet 名稱。畫面上應該會顯示原始檔。
7. 在 Servlet 中設定岔斷點，然後按 F12。
8. 執行 Servlet。當 Servlet 觸及岔斷點時，您仍可繼續除錯。

Java Platform Debugger Architecture

Java Platform Debugger Architecture (JPDA) 由 JVM 除錯介面/JVM 工具介面、Java Debug Wire Protocol 及 Java 除錯介面組成。JPDA 的所有部分都可讓任何使用 JDWP 的除錯程式前端系統去執行除錯作業。除錯程式前端系統可以在遠端執行，或以 iSeries 應用程式的形式執行。

Java 虛擬機器工具介面 (JVMTI)

JVMTI 取代了 JVMDI 及「Java 虛擬機器 Profiler 介面 (JVMPI)」。JVMTI 包含 JVMDI 及 JVMPI 的所有功能，以及一些新功能。JVMTI 已新增爲 J2SE 5.0 的一部分。未來的版次將不再提供 JVMDI 及 JVMPI 介面，僅會提供 JVMTI。

QSYS 檔案庫內稱爲 QJVAJVMTI 的服務程式支援 JVMTI 功能。

如需實作 JVMTI 的相關資訊，請參閱 Sun Microsystems 公司網站上的 JVMTI Reference 網頁。

Java 虛擬機器除錯介面

在 Java 2 SDK (J2SDK) 標準版 1.2 版或更高版本中，「Java 虛擬機器除錯介面 (JVMDI)」屬於 Sun Microsystems 公司平台應用程式設計介面 (API)。JVMDI 允許任何人以 iSeries C 程式碼來編寫適用於 iSeries 伺服器的 Java 除錯程式。因爲除錯程式使用 JVMDI 介面，所以不需要瞭解 Java 虛擬機器的內部結構。JVMDI 是 JPDA 的最底層介面，最接近 Java 虛擬機器。

除錯程式與 Java 虛擬機器都在同一個多緒工作中執行。除錯程式可以使用「Java 原生介面 (JNI)」呼叫 API 來建立 Java 虛擬機器。然後在使用者類別 main 方法的開端掛上追蹤點，再呼叫 main 方法。當 main 方法開始執行時，就會觸及追蹤點，於是開始除錯。一般的除錯機能皆可使用，例如設定岔斷點、逐行除錯、顯示變數及變更變數。

在執行 Java 虛擬機器的工作與負責處理使用者介面的工作之間，除錯程式會妥善處理兩者的通訊。此使用者介面可能在 iSeries 伺服器上，也可能在另一個系統上。

QSYS 檔案庫內的服務程式 QJVAJVMDI 支援 JVMDI 功能。

Java Debug Wire Protocol

Java Debug Wire Protocol (JDWP) 是除錯程序與 JVMDI/JVMTI 之間已定義的通訊協定。可以從遠端系統或透過本端 Socket 來使用 JDWP。雖然是從 JVMDI/JVMTI 中除去的一層，但卻是一個非常複雜的介面。

在 QShell 中啓動 JDWP

若要啓動 JDWP 及執行 Java 類別 SomeClass，請在 QShell 中輸入下列指令：

```
java -interpret -agentlib:jdwp=transport=dt_socket,
     address=8000,server=y,suspend=n SomeClass
```

在此範例中，JDWP 會在 TCP/IP 埠 8000 上接聽來自遠端除錯程式的連線，但可以使用您自己選擇的任何埠號；dt_socket 是處理 JDWP 傳輸的 SRVPGM 名稱，不會改變。

如需 -Xrunjdwp 可用的其他選項，請參閱 Sun Microsystems 公司的 Sun VM Invocation Options。這些選項可用於 i5/OS 上的 JDK 1.4 及 1.5。

從 CL 指令行啓動 JDWP

爲了從 CL 指令啓動 JDWP，已新增兩個新的選項：AGTPGM 及 AGTOPTIONS。

AGTPGM 的值是 JDWP，AGTOPTIONS 的值可以定義成您在 QShell 指令行上使用的相同字串。

若要啓動 JDWP 及執行 Java 類別 SomeClass，請輸入下列指令：

```
JAVA CLASS(SomeClass) INTERPRET(*YES) AGTPGM(JDWP)
AGTOPTIONS('transport=dt_socket,address=8000,server=y,suspend=n')
```

不建議對於「直接執行」程式碼使用 JVMDI/JVMTI。應該使用直譯器來執行應用程式，或使用「即時 (JIT) 編譯器」再搭配全速除錯。

Java 除錯介面

「Java 除錯介面 (JDI)」是一種用來開發工具的高階 Java 語言介面。JDI 透過一些 Java 類別定義來隱藏 JVMDI/JVMTI 及 JDWP 的複雜性。JDI 內含於 rt.jar 檔案中，所以除錯程式的前端系統僅存在於已安裝 Java 的平台上。

若想要編寫 Java 的除錯程式，則應該採用 JDI，因爲介面最簡單，且程式碼獨立於任何平台。

如需 JDPA 的詳細資訊，請參閱 Sun Microsystems 公司的 Java Platform Debugger Architecture Overview。

全速除錯:

iSeries Java 虛擬機器 (JVM) 現在支援「全速除錯」。在 V5R3 以前，啓用除錯功能就表示要停用即時 (JIT) 編譯器。許多方法皆必須於緩慢的直譯器之下執行，因此會折損應用程式的效能。 尤其，若應用程式需要執行幾天，才會到達您希望開始除錯的時機，這種效能退化窘況更形嚴重。

「全速除錯」可讓您在執行應用程式時，不僅享受到 JIT 編譯程式碼的所有效能優點，也不會喪失進行一些最常見除錯活動的能力，例如設定岔斷點、逐步執行程式碼及檢視區域變數。

由於全速除錯容許方法經過 JIT 編譯，所以除錯方面受到一些限制：

- 若呼叫端爲已編譯的程式碼，則無法在傳回陳述式上進行逐步除錯。
- 唯有在修改受監控欄位的非編譯方法中，才會觸發監控點。

註: 僅有使用 Java Debug Wire Protocol (JDWP) 執行除錯作業的除錯程式，才支援這項功能。系統除錯程式目前尚不支援全速除錯。

尋找記憶體洩漏

若您的程式經過長時間執行之後效能會下降，則表示記憶體洩漏方面的程式碼編寫可能有誤。可使用Java Watcher 協助您進行程式除錯，透過執行一段時間的 Java 應用程式資料堆分析及物件建立效能分析，找出記憶體洩漏之處。

如需詳細資料，請參閱 JavaWatcher。

您亦可使用「分析 Java 虛擬機器 (ANZJVM)」控制語言指令來尋找物件洩漏。ANZJVM 會比對一段指定時間前後的兩份垃圾收集資料堆，藉以找出物件洩漏之處。尋找物件洩漏時，請查看資料堆中每一個類別的實例個數。實例個數太多的類別，就有可能發生洩漏。

在兩份垃圾收集資料堆之間，您也應該觀察每一個類別的實例個數。若某個類別的實例個數持續增加，就應該注意此類別有可能發生洩漏。兩份資料堆之間的間隔時間愈長，愈能肯定物件確實發生洩漏。只要連續執行幾次 ANZJVM，且每一次的間隔時間夠久，診斷出洩漏原因的確定性就愈高。

IBM Developer Kit for Java 的程式碼範例

以下爲 IBM Developer Kit for Java 的程式碼範例清單。

國際化

- DateFormat
- NumberFormat
- ResourceBundle

JDBC

- Access 內容
- Blob
- CallableStatement 介面
- 在一個陳述式中透過另一個陳述式的游標來變更值
- Clob
- 建立 UDBDataSource 並將它與 JNDI 連結
- 建立 UDBDataSource 並取得使用者 ID 與密碼
- 建立 UDBDataSourceBind 並設定 DataSource 內容
- DatabaseMetaData 介面
- 建立 UDBDataSource 並將它與 JNDI 連結
- Datalink
- 特殊類型
- 內含的 SQL 陳述式
- 結束異動
- 無效的使用者 ID 和密碼
- JDBC
- 多個連線處理一個異動
- 連結 UDBDataSource 之前取得起始環境定義
- ParameterMetaData
- 透過另一個陳述式的游標來移除表格的值
- ResultSet 介面
- ResultSet 靈敏度
- 靈敏與不靈敏的 ResultSet
- 使用 UDBDataSource 與 UDBConnectionPoolDataSource 設定連線儲存區作業
- SQLException
- 暫停及回復異動
- 已暫停的 ResultSet
- 測試連線儲存區作業的效能

- 測試兩個 DataSource 的效能
- 更新 BLOB
- 更新 CLOB
- 使用一個連線來處理多個異動
- 使用 BLOB
- 使用 CLOB
- 使用 DB2CachedRowSet 內容及 DataSource
- 使用 DB2CachedRowSet 內容及 JDBC URL
- 使用 JTA 來處理異動
- 使用含有多個直欄的 meta 資料 ResultSet
- 同時使用原有的 JDBC 與 IBM Toolbox for Java JDBC
- 使用 PreparedStatement 來取得 ResultSet
- 使用 execute(Connection) 方法來利用現有的資料庫連線
- 使用 execute(int) 方法來組合資料庫要求
- 使用 populate 方法
- 使用 setConnection(Connection) 方法來利用現有的資料庫連線
- 使用 Statement 物件的 executeUpdate 方法

Java 鑑別和授權服務

- JAAS HelloWorld 範例
- JAAS SampleThreadSubjectLogin 範例

Java 一般安全性服務e

- 非 JAAS 用戶端程式範例
- 非 JAAS 伺服器程式範例
- 支援 JAAS 的用戶端程式範例
- 支援 JAAS 的伺服器程式範例

Java Secure Sockets Extension

- 使用 SSLContext 物件的 SSL 用戶端及伺服器

Java 與其他程式設計語言

- 呼叫 CL 程式
- 呼叫 CL 指令
- 呼叫另一個 Java 程式
- 從 C 呼叫 Java
- 從 RPG 呼叫 Java
- 輸入及輸出串流
- 呼叫 API
- 適用於 Java 的 i5/OS PASE 原生方法
- Socket
- 針對原生方法使用 Java 原生介面

效能工具

- Java 效能資料轉換器

SQLJ

- 在 Java 應用程式中內含 SQL 陳述式

Secure Sockets Layer

- Socket Factory
- 伺服器 Socket Factory
- Secure Sockets Layer
- Secure Sockets Layer 伺服器

IBM 授與您使用所有程式設計程式碼範例的非專屬授權，您可以利用這些範例來產生符合您需求的類似函數。

| 除法律規定不得排除的保證外，IBM、IBM 之程式開發人員及供應商不附具任何明示或默示之保證，包含且不
| 限於任何相關技術支援之未侵害他人權利之保證、或可商用性及符合特定效用等之默示保證。

| 在任何情況下，IBM、IBM 之程式開發者或供應商對下列情事均不負賠償責任，即使被告知該情事有可能發生
| 時，亦同：

| 1. 資料之滅失或毀損；
| 2. 直接、特殊、附帶或間接的傷害或其他衍生之經濟損害；或
| 3. 利潤、營業、收益、商譽或預期節餘等項之損失。

| 倘法律規定不得排除或限制賠償責任時，則該排除或限制無效。

範例：使用 java.util.DateFormat 類別進行日期的國際化

下列範例顯示如何使用語言環境來格式化日期。

範例 1：示範使用 java.util.DateFormat 類別進行日期的國際化

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//************************
// File: DateExample.java
//************************

import java.text.*;
import java.util.*;
import java.util.Date;

public class DateExample {

   public static void main(String args[]) {

     // Get the Date
     Date now = new Date();

     // Get date formatters for default, German, and French locales
     DateFormat theDate = DateFormat.getDateInstance(DateFormat.LONG);
     DateFormat germanDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.GERMANY);
     DateFormat frenchDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);

     // Format and print the dates
     System.out.println("Date in the default locale: " + theDate.format(now));
```

```
    System.out.println("Date in the German locale : " + germanDate.format(now));
    System.out.println("Date in the French locale : " + frenchDate.format(now));
  }
}
```

如需相關資訊，請參閱建立國際化 Java 程式。

鏈結集合

程式碼範例免責聲明

建立國際化 Java(TM) 程式
如需自訂 Java 程式以提供給特定地區使用，請利用 Java 語言環境來建立國際化 Java 程式。

範例：使用 java.util.NumberFormat 類別來進行數字呈現方式的國際化

下列範例顯示如何使用語言環境來格式化數字。

範例 1：示範使用 java.util.NumberFormat 類別來進行數字輸出的國際化

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//**************************
// File: NumberExample.java
//**************************

 import java.lang.*;
 import java.text.*;
 import java.util.*;

 public class NumberExample {

     public static void main(String args[]) throws NumberFormatException {

       // The number to format
       double number = 12345.678;

       // Get formatters for default, Spanish, and Japanese locales
       NumberFormat defaultFormat = NumberFormat.getInstance();
       NumberFormat spanishFormat = NumberFormat.getInstance(new
 Locale("es", "ES"));
       NumberFormat japaneseFormat = NumberFormat.getInstance(Locale.JAPAN);

       // Print out number in the default, Spanish, and Japanese formats
       // (Note: NumberFormat is not necessary for the default format)
       System.out.println("The number formatted for the default locale; " +
                          defaultFormat.format(number));
       System.out.println("The number formatted for the Spanish locale; " +
                          spanishFormat.format(number));
       System.out.println("The number formatted for the Japanese locale; " +
                          japaneseFormat.format(number));
     }
 }
```

如需相關資訊，請參閱建立國際化 Java 程式。

鏈結集合

程式碼範例免責聲明

建立國際化 Java 程式
如需自訂 Java 程式以提供給特定地區使用，請利用 Java 語言環境來建立國際化 Java 程式。

範例：使用 java.util.ResourceBundle 類別來進行語言環境特有資料的國際化

下列範例顯示如何使用語言環境及資源軟體組來進行程式字串的國際化。

ResourceBundleExample 程式需要下列內容檔才能正常運作：

RBExample.properties 的內容

```
Hello.text=Hello
```

RBExample_de.properties 的內容

```
Hello.text=Guten  Tag
```

RBExample_fr_FR.properties 的內容

```
Hello.text=Bonjour
```

範例 1：示範使用 java.util.ResourceBundle 類別來進行特定地區資料的國際化

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//*********************************
// File: ResourceBundleExample.java
//*********************************

import java.util.*;

public class ResourceBundleExample {
   public static void main(String args[]) throws MissingResourceException {

     String resourceName = "RBExample";
     ResourceBundle rb;

     // Default locale
     rb = ResourceBundle.getBundle(resourceName);
     System.out.println("Default : " + rb.getString("Hello" + ".text"));

     // Request a resource bundle with explicitly specified locale
     rb = ResourceBundle.getBundle(resourceName, Locale.GERMANY);
     System.out.println("German : " + rb.getString("Hello" + ".text"));

     // No property file for China in this example... use default
     rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);
     System.out.println("Chinese : " + rb.getString("Hello" + ".text"));

     // Here is another way to do it...
     Locale.setDefault(Locale.FRANCE);
     rb = ResourceBundle.getBundle(resourceName);
     System.out.println("French : " + rb.getString("Hello" + ".text"));

     // No property file for China in this example... use default, which is now fr_FR.
     rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);
     System.out.println("Chinese : " + rb.getString("Hello" + ".text"));
   }
}
```

如需詳細資訊，請參閱建立國際化 Java(TM) 程式。

鏈結集合

程式碼範例免責聲明

建立國際化 Java(TM) 程式
如需自訂 Java 程式以提供給特定地區使用，請利用 Java 語言環境來建立國際化 Java 程式。

範例：Access 內容

以下爲如何使用 Access 內容的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Note: This program assumes directory cujosql exists.
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();
    }


/**
Set up the DataSource used in the testing.
**/
    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) { // Ignore it - it doesn't exist
        }

        try {
            String sql = "CREATE PROCEDURE CUJOSQL.TEMP "
                  + " LANGUAGE SQL SPECIFIC CUJOSQL.TEMP "
                  + " MYPROC: BEGIN"
                  + "   RETURN 11;"
                  + " END MYPROC";
            s.executeUpdate(sql);
        } catch (SQLException e) {
            // Ignore it - it exists.
        }
        s.executeUpdate("create table cujosql.temp (col1 char(10))");
        s.executeUpdate("insert into cujosql.temp values ('compare')");
        s.close();
    }


    public void resetConnection(String property)
    throws SQLException
    {
        if (connection != null)
            connection.close();

        connection = DriverManager.getConnection(url + ";access=" + property);
    }
```

```
public boolean canQuery() {
    Statement s = null;
    try {
        s = connection.createStatement();
        ResultSet rs = s.executeQuery("SELECT * FROM cujosql.temp");
        if (rs == null)
            return false;

        rs.next();

        if (rs.getString(1).equals("compare   "))
            return true;

        return false;

    } catch (SQLException e) {
        // System.out.println("Exception: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}


public boolean canUpdate() {
    Statement s = null;
    try {
        s = connection.createStatement();
        int count = s.executeUpdate("INSERT INTO CUJOSQL.TEMP VALUES('x')");
        if (count != 1)
            return false;

        return true;

    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}


public boolean canCall() {
    CallableStatement s = null;
    try {
        s = connection.prepareCall("? = CALL CUJOSQL.TEMP()");
        s.registerOutParameter(1, Types.INTEGER);
        s.execute();
        if (s.getInt(1) != 11)
            return false;
```

```
            return true;

        } catch (SQLException e) {
            //System.out.println("Exception: SQLState(" +
            //                   e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
            return false;
        } finally {
            if (s != null) {
                try {
                    s.close();
                } catch (Exception e) {
                    // Ignore it.
                }
            }
        }
    }


    public void run()
    throws SQLException
    {
        System.out.println("Set the connection access property to read only");
        resetConnection("read only");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

        System.out.println("Set the connection access property to read call");
        resetConnection("read call");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

        System.out.println("Set the connection access property to all");
        resetConnection("all");

        System.out.println("Can run queries  -->" + canQuery());
        System.out.println("Can run updates  -->" + canUpdate());
        System.out.println("Can run sp calls -->" + canCall());

    }


    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            // Ignore it.
        }
    }
}
```

範例：BLOB

以下範例說明如何在資料庫內放入或擷取 BLOB。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////
// PutGetBlobs is an example application
// that shows how to work with the JDBC
// API to obtain and put BLOBs to and from
// database columns.
//
// The results of running this program
```

```
// are that there are two BLOB values
// in a new table. Both are identical
// and contain 500k of random byte
// data.
/////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      // Establish a Connection and Statement with which to work.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Clean up any previous run of this application.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
      } catch (SQLException e) {
          // Ignore it - assume the table did not exist.
      }

      // Create a table with a BLOB column. The default BLOB column
      // size is 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

      // Create a PreparedStatement object that allows you to put
      // a new Blob object into the database.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

      // Create a big BLOB value...
      Random random = new Random ();
      byte [] inByteArray = new byte[500000];
      random.nextBytes (inByteArray);

      // Set the PreparedStatement parameter. Note: This is not
      // portable to all JDBC drivers. JDBC drivers do not have
      // support when using setBytes for BLOB columns. This is used to
      // allow you to generate new BLOBs. It also allows JDBC 1.0
      // drivers to work with columns containing BLOB data.
      ps.setBytes(1, inByteArray);

      // Process the statement, inserting the BLOB into the database.
      ps.executeUpdate();

      // Process a query and obtain the BLOB that was just inserted out
      // of the database as a Blob object.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
      rs.next();
      Blob blob = rs.getBlob(1);


      // Put that Blob back into the database through
      // the PreparedStatement.
      ps.setBlob(1, blob);
      ps.execute();
```

```
      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：IBM Developer Kit for Java 的 CallableStatement 介面

以下爲如何使用 CallableStatement 介面的範例。

範例：CallableStatement 介面

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

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

// Create the CallableStatement object.
// It precompiles the specified call to a stored procedure.
// The question marks indicate where input parameters must be set and
// where output parameters can be retrieved.
// The first two parameters are input parameters, and the third parameter is an output parameter.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

// Set input parameters.
cs.setInt (1, 123);
cs.setInt (2, 234);

// Register the type of the output parameter.
cs.registerOutParameter (3, Types.INTEGER);

// Run the stored procedure.
cs.execute ();

// Get the value of the output parameter.
int sum = cs.getInt (3);

// Close the CallableStatement and the Connection.
cs.close();
c.close();
```

如需詳細資訊，請參閱 CallableStatement。

鏈結集合

程式碼範例免責聲明

CallableStatements

CallableStatement 介面是 PreparedStatement 的延伸，可支援輸出及輸入/輸出參數。CallableStatement 介面亦支援 PreparedStatement 介面所提供的輸入參數。

範例：透過另一個陳述式的游標來移除表格的值

以下範例說明如何透過另一個陳述式的游標來移除表格的值。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }



/**
Handle all the required setup work.
**/
    public void setup() {
        try {
            // Register the JDBC driver.
            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) {
                // Ignore problems here.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }


/**
In this section, all the code to perform the testing should
be added. If only one connection to the database is needed,
the global variable 'connection' can be used.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Update each value using next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement
                                      ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
                                      rs.getCursorName ());


            // Loop through the ResultSet and update every other entry.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
```

```
            }


            // Clean up the resources after they have been used.
            rs.close ();
            stmt2.close ();


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



/**
In this section, put all clean-up work for testing.
**/
    public void cleanup() {
        try {
            // Close the global connection opened in setup().
            connection.close();

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



/**
Display the contents of the table.
**/
    public void displayTable()
    {
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

鏈結集合

程式碼範例免責聲明

範例：CLOB

以下範例說明如何在資料庫內放入或擷取 CLOB。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////
// PutGetClobs is an example application
// that shows how to work with the JDBC
// API to obtain and put CLOBs to and from
// database columns.
//
// The results of running this program
// are that there are two CLOB values
// in a new table. Both are identical
// and contain about 500k of repeating
// text data.
////////////////////////////////////////
import java.sql.*;

public class PutGetClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      // Establish a Connection and Statement with which to work.
      Connection c = DriverManager.getConnection("jdbc:db2:*local");
      Statement s = c.createStatement();

      // Clean up any previous run of this application.
      try {
          s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
      } catch (SQLException e) {
          // Ignore it - assume the table did not exist.
      }

      // Create a table with a CLOB column. The default CLOB column
      // size is 1 MB.
      s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

      // Create a PreparedStatement object that allow you to put
      // a new Clob object into the database.
      PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

      // Create a big CLOB value...
      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();

      // Set the PreparedStatement parameter. This is not
      // portable to all JDBC drivers. JDBC drivers do not have
      // to support setBytes for CLOB columns. This is done to
      // allow you to generate new CLOBs. It also
      // allows JDBC 1.0 drivers a way to work with columns containing
      // Clob data.
      ps.setString(1, clobValue);

      // Process the statement, inserting the clob into the database.
      ps.executeUpdate();

      // Process a query and get the CLOB that was just inserted out of the
      // database as a Clob object.
      ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
      rs.next();
      Clob clob = rs.getClob(1);
```

```
      // Put that Clob back into the database through
      // the PreparedStatement.
      ps.setClob(1, clob);
      ps.execute();

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：建立 UDBDataSource 並連結 JNDI

以下爲如何建立 UDBDataSource 並將其連結 JNDI 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
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
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS", ds);
    }
}
```

範例：建立 UDBDataSource 並取得使用者 ID 與密碼

以下範例將示範如何建立 UDBDataSource，並於執行時透過 getConnection 方法來取得使用者 ID 及密碼。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/// Import the required packages. There is
// no driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Retrieve a JNDI context. The context serves
```

```
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Retrieve the bound UDBDataSource object using the
        // name with which it was previously bound. At runtime,
        // only the DataSource interface is used, so there
        // is no need to convert the object to the UDBDataSource
        // implementation class. (There is no need to know
        // what the implementation class is. The logical JNDI name
        // is only required).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Once the DataSource is obtained, it can be used to establish
        // a connection. The user profile cujo and password newtiger
        // used to create the connection instead of any default user
        // ID and password for the DataSource.
        Connection connection = ds.getConnection("cujo", "newtiger");

        // The connection can be used to create Statement objects and
        // update the database or process queries as follows.
         Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // The connection is closed before the application ends.
        connection.close();
    }
}
```

範例：建立 UDBDataSourceBind 並設定 DataSource 內容

以下爲如何建立 UDBDataSource 並設定使用者 ID 及密碼作爲 DataSource 內容的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
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
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource "  +
                          "with cujo as the default " +
                          "profile to connect with.");

        // Provide a user ID and password to be used for
        // connection requests.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();
```

```
        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS2", ds);
    }
}
```

範例：IBM Developer Kit for Java 的 DatabaseMetaData 介面 - 傳回表格清單

本範例顯示如何傳回表格清單。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

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

// Get the database meta data from the connection.
     DatabaseMetaData dbMeta = c.getMetaData();

// Get a list of tables matching this criteria.

String catalog = "myCatalog";
String schema = "mySchema";
String table  = "myTable%"; // % indicates search pattern
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"}:
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterate through the ResultSet to get the values.

// Close the connection.
c.close():
```

如需相關資訊，請參閱 IBM Developer Kit for Java 的 DatabaseMetaData 介面。

範例：Datalink

以下爲如何在應用程式中使用 Datalink 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////
// PutGetDatalinks is an example application
// that shows how to use the JDBC
// API to handle datalink database columns.
////////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
    throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Setup error.
        }

        // Establish a Connection and Statement with which to work.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();
```

```
        // Clean up any previous run of this application.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
        } catch (SQLException e) {
            // Ignore it - assume the table did not exist.
        }

        // Create a table with a datalink column.
        s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

        // Create a PreparedStatement object that allows you to add
        // a new datalink into the database. Since conversing
        // to a datalink cannot be accomplished directly in the database, you
        // can code the SQL statement to perform the explicit conversion.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
                                                  VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

        // Set the datalink. This URL points you to an article about
        // the new features of JDBC 3.0.
        ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

        // Process the statement, inserting the CLOB into the database.
        ps.executeUpdate();

        // Process a query and obtain the CLOB that was just inserted out of the
        // database as a Clob object.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        String datalink = rs.getString(1);


        // Put that datalink value into the database through
        // the PreparedStatement. Note: This function requires JDBC 3.0
        // support.
        /*
        try {
            URL url = new URL(datalink);
            ps.setURL(1, url);
            ps.execute();
        } catch (MalformedURLException mue) {
            // Handle this issue here.
        }

        rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        URL url = rs.getURL(1);
        System.out.println("URL value is " + url);
        */

        c.close(); // Connection close also closes stmt and rs.
    }
}
```

範例：特殊類型

以下爲如何使用特殊類型的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// This example program shows examples of
// various common tasks that can be done
// with distinct types.
//////////////////////////////////////
import java.sql.*;

public class Distinct {
```

```
    public static void main(String[] args)
    throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1);  // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any old runs.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        // Create the type, create the table, and insert a value.
        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();

        // You can obtain details about the types available with new metadata in
        // JDBC 2.0
        DatabaseMetaData dmd = c.getMetaData();

        int types[] = new int[1];
        types[0] = java.sql.Types.DISTINCT;

        ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN", types);
        rs.next();
        System.out.println("Type name " + rs.getString(3) +
                           " has type " + rs.getString(4));


        // Access the data you have inserted.
        rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
        rs.next();
        System.out.println("The SSN is " + rs.getString(1));

        c.close(); // Connection close also closes stmt and rs.
    }
}
```

範例：在 Java 應用程式中內含 SQL 陳述式

以下 SQLJ 應用程式範例 App.sqlj，將使用靜態 SQL 在 DB2 範例資料庫的 EMPLOYEE 表格內擷取及更新資料。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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
{

  /*********************
      **  Register Driver **
   *********************/

  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;

     // URL is jdbc:db2:dbname
     String url = "jdbc:db2:sample";

     DefaultContext ctx = DefaultContext.getDefaultContext();
     if (ctx == null)
     {
       try
       {
         // connect with default id/password
         Connection con = DriverManager.getConnection(url);
         con.setAutoCommit(false);
         ctx = new DefaultContext(con);
       }
       catch (SQLException e)
       {
         System.out.println("Error: could not get a default context");
         System.err.println(e) ;
         System.exit(1);
       }
       DefaultContext.setDefaultContext(ctx);
     }

     // retrieve data from the database
     System.out.println("Retrieve some data from the database.");
     #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2
```

```
    // display the result set
    // cursor1.next() returns false when there are no more rows
    System.out.println("Received results:");
    while (cursor1.next()) // 3
    {
      str1 = cursor1.empno(); // 4
      str2 = cursor1.firstnme();

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor1.close(); // 9

    // retrieve number of employee from the database
    #sql { SELECT count(*) into :count1 FROM employee }; // 5
    if (1 == count1)
      System.out.println ("There is 1 row in employee table");
    else
      System.out.println ("There are " + count1
                          + " rows in employee table");

    // update the database
    System.out.println("Update the database.");
    #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

    // retrieve the updated data from the database
    System.out.println("Retrieve the updated data from the database.");
    str1 = "000010";
    #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

    // display the result set
    // cursor2.next() returns false when there are no more rows
    System.out.println("Received results:");
    while (true)
    {
      #sql { FETCH :cursor2 INTO :str2 }; // 7
      if (cursor2.endFetch()) break; // 8

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor2.close(); // 9

    // rollback the update
    System.out.println("Rollback the update.");
    #sql { ROLLBACK work };
    System.out.println("Rollback done.");
  }
  catch( Exception e )
  {
    e.printStackTrace();
  }
 }
}
```

[1]宣告疊代子。此區段宣告兩種疊代子：

- App_Cursor1：宣告直欄資料類型及名稱，然後根據直欄名稱傳回直欄的值 (直欄的具名連結)。
- App_Cursor2：宣告直欄資料類型，然後根據直欄位置傳回直欄的值 (直欄的定位連結)。

[2]起始設定疊代子。使用查詢結果來起始設定疊代子物件 cursor1。查詢結果會儲存在 cursor1 內。

[3]將疊代子移至下一橫列。若已無其他列可擷取，cursor1.next() 方法會傳回布林 `false`。

[4]移動資料。具名 accessor 方法 empno() 會傳回現行列上 empno 直欄的值。 具名 accessor 方法 firstnme() 會傳回現行列上 firstnme 直欄的值。

[5]SELECT 資料傳入主變數。SELECT 陳述式會將表格列數傳入主變數 count1 內。

[6]起始設定疊代子。使用查詢結果來起始設定疊代子物件 cursor2。查詢結果會儲存在 cursor2 內。

[7]擷取資料。FETCH 陳述式會從結果表格中，將 ByPos 游標中宣告的第一欄的現行值傳回主變數 str2 內。

[8]檢查 FETCH.INTO 陳述式是否成功。若疊代子不在某列上，亦即，若前次試圖擷取某列失敗，endFetch() 方法會傳回布林 true。若前次試圖擷取某列成功，endFetch() 方法會傳回 false。呼叫 next() 方法時，DB2 會試圖提取一個橫列。FETCH...INTO 陳述式會隱含地呼叫 next() 方法。

[9]關閉疊代子。close() 方法會釋放疊代子佔用的任何資源。應該明確地關閉疊代子，確保能夠及時地釋放系統資源。

如需本範例的背景資訊，請參閱在 Java 應用程式中內含 SQL 陳述式。

範例：結束異動

以下爲在應用程式中結束異動的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... does not exist
            }

            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();
        }
    }
}


/**
 * This test use JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // For XA transactions, transaction identifier is required.
        // An implementation of the XID interface is not included
        // with the JDBC driver. See Transactions with JTA for a
        // description of this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Create a ResultSet during JDBC processing and fetch a row.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // When the end method is called, all ResultSet cursors close.
        // Accessing the ResultSet after this point results in an
        // exception being thrown.
        xaRes.end(xid, XAResource.TMNOFLAGS);

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

        // Commit the transaction to ensure that all locks are
        // released.
        int rc = xaRes.prepare(xid);
        xaRes.commit(xid, false);


    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    } finally {
```

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

鏈結集合

程式碼範例免責聲明

使用 JTA 的異動

在 Java Database Connectivity (JDBC) 中，異動通常發生在本端。這表示單一連線即可完成整個異動的所有工作，且連線一次只能處理一個異動。當異動的所有工作皆完成或失敗時，就會呼叫確定或回復，讓工作持續進行，好讓新的交易得以開始。然而，Java 中的異動還有更進階的支援，功能超過本端異動。此支援由 Java Transaction API 完全指定。

範例：無效的使用者 ID 和密碼

下列範例顯示如何以 SQL 命名模式來使用 Connection 內容。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////////////////////////////////////////
//
// InvalidConnect example.
//
// This program uses the Connection property in SQL naming mode.
//
//////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// 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)
    {
        // Register the driver.
```

```
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: JDBC driver did not load.");
            System.exit(0);
        }

        // Attempt to obtain a connection without specifying any user or
        // password. The attempt works and the connection uses the
        // same user profile under which the job is running.
        try {
            Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
            c1.close();
        } catch (SQLException e) {
            System.out.println("This test should not get into this exception path.");
            e.printStackTrace();
            System.exit(1);
        }

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

    }
}
```

範例：JDBC

以下爲如何使用 BasicJDBC 程式的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////////////////////////////////////////
//
// BasicJDBC example.  This program uses the native JDBC driver for the
// Developer Kit for Java to build a simple table and process a query
// that displays the data in that table.
//
// Command syntax:
//    BasicJDBC
//
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
```

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

// Include any Java classes that are to be used. In this application,
// many classes from the java.sql package are used and the
// java.util.Properties class is also used as part of obtaining
// a connection to the database.
import java.sql.*;
import java.util.Properties;

// Create a public class to encapsulate the program.
public class BasicJDBC {

    // The connection is a private variable of the object.
    private Connection connection = null;

    // Any class that is to be an 'entry point' for running
    // a program must have a main method. The main method
    // is where processing begins when the program is called.
    public static void main(java.lang.String[] args) {

        // Create an object of type BasicJDBC. This
        // is fundamental to object-oriented programming. Once
        // an object is created, call various methods on
        // that object to accomplish work.
        // In this case, calling the constructor for the object
        // creates a database connection that the other
        // methods use to do work against the database.
        BasicJDBC test = new BasicJDBC();

        // Call the rebuildTable method. This method ensures that
        // the table used in this program exists and looks
        // correct. The return value is a boolean for
        // whether or not rebuilding the table completed
        // successfully. If it did no, display a message
        // and exit the program.
        if (!test.rebuildTable()) {
            System.out.println("Failure occurred while setting up " +
                               " for running the test.");
            System.out.println("Test will not continue.");
            System.exit(0);
        }

        // The run query method is called next. This method
        // processes an SQL select statement against the table that
        // was created in the rebuildTable method. The output of
        // that query is output to standard out for you to view.
        test.runQuery();

        // Finally, the cleanup method is called. This method
        // ensures that the database connection that the object has
        // been hanging on to is closed.
        test.cleanup();
    }


    /**
    This is the constructor for the basic JDBC test. It creates a database
    connection that is stored in an instance variable to be used in later
    method calls.
    **/
    public BasicJDBC() {

        // One way to create a database connection is to pass a URL
        // and a java Properties object to the DriverManager. The following
```

```
    // code constructs a Properties object that has your user ID and
    // password. These pieces of information are used for connecting
    // to the database.
    Properties properties = new Properties ();
    properties.put("user", "cujo");
    properties.put("user", "newtiger");

    // Use a try/catch block to catch all exceptions that can come out of the
    // following code.
    try {
        // The DriverManager must be aware that there is a JDBC driver available
        // to handle a user connection request. The following line causes the
        // native JDBC driver to be loaded and registered with the DriverManager.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        // Create the database Connection object that this program uses in all
        // the other method calls that are made. The following code specifies
        // that a connection is to be established to the local database and that
        // that connection should conform to the properties that were set up
        // previously (that is, it should use the user ID and password specified).
        connection = DriverManager.getConnection("jdbc:db2:*local", properties);

    } catch (Exception e) {
        // If any of the lines in the try/catch block fail, control transfers to
        // the following line of code. A robust application tries to handle the
        // problem or provide more details to you. In this program, the error
        // message from the exception is displayed and the application allows
        // the program to return.
        System.out.println("Caught exception: " + e.getMessage());
    }
}


/**
Ensures that the qgpl.basicjdbc table looks you want it to at the start of
the test.

@returns boolean    Returns true if the table was rebuild successfully;
                    returns false if any failure occurred.
**/
public boolean rebuildTable() {
    // Wrap all the functionality in a try/catch block so an attempt is
    // made to handle any errors that may happen within this method.
    try {

        // Statement objects are used to process SQL statements against the
        // database. The Connection object is used to create a Statement
        // object.
        Statement s = connection.createStatement();

        try {
            // Build the test table from scratch. Process an update statement
            // that attempts to delete the table if it currently exists.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // Do not perform anything if an exception occurred. Assume
            // that the problem is that the table that was dropped does not
            // exist and that it can be created next.
        }

        // Use the statement object to create our table.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Use the statement object to populate our table with some data.
        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')");

        // Close the SQL statement to tell the database that it is no longer
        // needed.
        s.close();

        // If the entire method processed successfully, return true. At this point,
        // the table has been created or refreshed correctly.
        return true;

    } catch (SQLException sqle) {
        // If any of our SQL statements failed (other than the drop of the table
        // that was handled in the inner try/catch block), the error message is
        // displayed and false is returned to the caller, indicating that the table
        // may not be complete.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}



/**
Runs a query against the demonstration table and the results are displayed to
standard out.
**/
public void runQuery() {
    // Wrap all the functionality in a try/catch block so an attempts is
    // made to handle any errors that might happen within this
    // method.
    try {
        // Create a Statement object.
        Statement s = connection.createStatement();

        // Use the statement object to run an SQL query. Queries return
        // ResultSet objects that are used to look at the data the query
        // provides.
        ResultSet rs = s.executeQuery("select * from qgpl.basicjdbc");

        // Display the top of our 'table' and initialize the counter for the
        // number of rows returned.
        System.out.println("--------------------");
        int i = 0;

        // The ResultSet next method is used to process the rows of a
        // ResultSet. The next method must be called once before the
        // first data is available for viewing. As long as next returns
        // true, there is another row of data that can be used.
        while (rs.next()) {

            // Obtain both columns in the table for each row and write a row to
            // our on-screen table with the data. Then, increment the count
            // of rows that have been processed.
            System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
            i++;
        }

        // Place a border at the bottom on the table and display the number of rows
        // as output.
        System.out.println("--------------------");
        System.out.println("There were " + i + " rows returned.");
        System.out.println("Output is complete.");

    } catch (SQLException e) {
        // Display more information about any SQL exceptions that are
        // generated as output.
        System.out.println("SQLException exception: ");
```

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


    /**
    The following method ensures that any JDBC resources that are still
    allocated are freed.
    **/
    public void cleanup() {
        try {
            if (connection != null)
                connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

範例：多個連線處理一個異動

以下範例說明如何在單一異動上使用多個連線。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
   }
/**
* Handle the previous cleanup run so that this test can recommence.
*/
   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) {
         // Ignore... does not exist
         }
         s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
                         (50))");
               s.close();
      }
      finally {
         if (c != null) {
            c.close();
         }
      }
   }
```

```
/**
* This test uses JTA support to handle transactions.
*/
   public void run() {
      Connection c1 = null;
      Connection c2 = null;
      Connection c3 = null;
      try {
         Context ctx = new InitialContext();
         // Assume the data source is backed by a UDBXADataSource.
         UDBXADataSource ds = (UDBXADataSource)
                              ctx.lookup("XADataSource");
         // From the DataSource, obtain some XAConnection objects that
         // contain an XAResource and a Connection object.
         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();
         // For XA transactions, a transaction identifier is required.
         // Support for creating XIDs is again left to the application
         // program.
         Xid xid = JDXATest.xidFactory();
         // Perform some transactional work under each of the three
         // connections that have been created.
         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);
         // When completed, commit the transaction as a single unit.
         // A prepare() and commit() or 1 phase commit() is required for
         // each separate database (XAResource) that participated in the
         // transaction. Since the resources accessed (xaRes1, xaRes2, and xaRes3)
         // all refer to the same database, only one prepare or commit is required.
         int rc = xaRes.prepare(xid);
         xaRes.commit(xid, false);
      }
      catch (Exception e) {
         System.out.println("Something has gone wrong.");
         e.printStackTrace();
      }
      finally {
         try {
            if (c1 != null) {
               c1.close();
            }
         }
         catch (SQLException e) {
            System.out.println("Note: Cleaup exception " +
                               e.getMessage());
         }
         try {
            if (c2 != null) {
```

```
            c2.close();
         }
      }
      catch (SQLException e) {
         System.out.println("Note: Cleaup exception " +
                            e.getMessage());
      }
      try {
         if (c3 != null) {
            c3.close();
         }
      }
      catch (SQLException e) {
         System.out.println("Note: Cleaup exception " +
                            e.getMessage());
      }
   }
}
```

範例：連結 UDBDataSource 之前取得起始環境定義

下列範例會在連結 UDBDataSource 之前先取得起始環境定義。 接著會在此環境定義上使用 lookup 方法，以傳回 DataSource 類型的物件，供應用程式使用。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// Import the required packages. There is no
// driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Retrieve the bound UDBDataSource object using the
        // name with which it was previously bound. At runtime,
        // only the DataSource interface is used, so there
        // is no need to convert the object to the UDBDataSource
        // implementation class. (There is no need to know what
        // the implementation class is. The logical JNDI name is
        // only required).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");


        // Once the DataSource is obtained, it can be used to establish
        // a connection. This Connection object is the same type
        // of object that is returned if the DriverManager approach
        // to establishing connection is used. Thus, so everything from
        // this point forward is exactly like any other JDBC
        // application.
        Connection connection = ds.getConnection();

        // The connection can be used to create Statement objects and
        // update the database or process queries as follows.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
```

```
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // The connection is closed before the application ends.
        connection.close();
    }
}
```

範例：ParameterMetaData

以下爲使用 ParameterMetaData 介面來擷取參數相關資訊的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////////////////////////////////////////
//
// ParameterMetaData example. This program demonstrates
// the new support of JDBC 3.0 for learning information
// about parameters to a PreparedStatement.
//
// Command syntax:
//    java PMD
//
//////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
//////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class PMD {

    // Program entry point.
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Obtain setup.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.MYTABLE VALUES(?, ?, ?)");
        ParameterMetaData pmd = ps.getParameterMetaData();

        for (int i = 1; i < pmd.getParameterCount(); i++) {
            System.out.println("Parameter number " + i);
            System.out.println("  Class name is " + pmd.getParameterClassName(i));
```

```
            // Note: Mode relates to input, output or inout
            System.out.println("  Mode is " + pmd.getParameterClassName(i));
            System.out.println("  Type is " + pmd.getParameterType(i));
            System.out.println("  Type name is " + pmd.getParameterTypeName(i));
            System.out.println("  Precision is " + pmd.getPrecision(i));
            System.out.println("  Scale is " + pmd.getScale(i));
            System.out.println("  Nullable? is " + pmd.isNullable(i));
            System.out.println("  Signed? is " + pmd.isSigned(i));
        }
    }
}
```

範例：在一個陳述式中透過另一個陳述式的游標來變更值

以下範例說明如何在一個陳述式中透過另一個陳述式的游標來變更值。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }



/**
Handle all the required setup work.
**/
    public void setup() {
        try {
            // Register the JDBC driver.
            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) {
                // Ignore problems here.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                            "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
```

```
    }


/**
In this section, all the code to perform the testing should
be added. If only one connection to the database is required,
the global variable 'connection' can be used.
**/
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Update each value using next().
            stmt1.setCursorName("CUJO");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                               "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
                                                                   + " CUJOSQL.WHERECUREX
                                                                       SET COL_VALUE = 'CHANGED'
                                                                       WHERE CURRENT OF "
                                                                   + rs.getCursorName ());


            // Loop through the ResultSet and update every other entry.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }


            // Clean up the resources after they have been used.
            rs.close ();
            stmt2.close ();


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



/**
In this section, put all clean-up work for testing.
**/
    public void cleanup() {
        try {
            // Close the global connection opened in setup().
            connection.close();

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



/**
Display the contents of the table.
**/
    public void displayTable()
    {
```

```
        try {
            Statement s = connection.createStatement();
            ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

            while (rs.next ()) {
                System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
            }

            rs.close ();
            s.close();
            System.out.println("----------------------------------------");
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

鏈結集合

程式碼範例免責聲明

範例：IBM Developer Kit for Java 的 ResultSet 介面

以下爲如何使用 ResultSet 介面的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;

/**
ResultSetExample.java

This program demonstrates using a ResultSetMetaData and
a ResultSet to display all the data in a table even though
the program that gets the data does not know what the table
is going to look like (the user passes in the values for the
table and library).
**/
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage:  java ResultSetExample <library> <table>");
            System.out.println(" where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Get a database connection and prepare a statement.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
            int rowCount = 0;
            while (rs.next ()) {
```

```
                rowCount++;
                System.out.println("Data for row " + rowCount);
                for (int i = 1; i <= colCount; i++)
                    System.out.println("   Row " + i + ": " + rs.getString(i));
            }

        } catch (Exception e) {
            // Handle any errors.
            System.out.println("Oops... we have an error... ");
            e.printStackTrace();
        } finally {
            // Ensure we always clean up.  If the connection gets closed, the
            // statement under it closes as well.
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    System.out.println("Critical error - cannot close connection object");
                }
            }
        }
    }
}
```

範例：ResultSet 靈敏度

以下範例說明隨著 ResultSet 靈敏度的不同，變更將如何影響 SQL 陳述式的 where 子句。

爲了使此範例符合列印頁，此範例中的某些格式可能不正確。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;

public class Sensitive2 {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive2 test = new Sensitive2();

        test.setup();
        test.run("sensitive");
        test.cleanup();

        test.setup();
        test.run("insensitive");
        test.cleanup();
    }


    public void setup() {

        try {
            System.out.println("Native JDBC used");
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("drop table cujosql.sensitive");
            } catch (SQLException e) {
                // Ignored.
            }
```

```
        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) {
            // Ignored.
        }

        s.executeUpdate("create table cujosql.sensitive2(col2 int)");
        s.executeUpdate("insert into cujosql.sensitive2 values(1)");
        s.executeUpdate("insert into cujosql.sensitive2 values(2)");
        s.executeUpdate("insert into cujosql.sensitive2 values(3)");
        s.executeUpdate("insert into cujosql.sensitive2 values(4)");
        s.executeUpdate("insert into cujosql.sensitive2 values(5)");

        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select col1, col2 From cujosql.sensitive,
                                       cujosql.sensitive2 where col1 = col2");

        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));

        System.out.println("fetched the four rows...");

        // Another statement creates a value that does not fit the where clause.
```

```
            Statement s2 =
              connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
              ResultSet.CONCUR_UPDATEABLE);
            ResultSet rs2 = s2.executeQuery("select *
            from cujosql.sensitive where col1 = 5 FOR UPDATE");
            rs2.next();
            rs2.updateInt(1, -1);
            rs2.updateRow();
            s2.close();

            if (rs.next()) {
                System.out.println("There is still a row: " + rs.getInt(1));
            } else {
                System.out.println("No more rows.");
            }

        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("----------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

範例：靈敏與不靈敏的 ResultSet

以下範例顯示當表格內插入橫列時，靈敏與不靈敏的 ResultSet 兩者有何差異。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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) {
            // Ignored.
        }

        s.executeUpdate("create table cujosql.sensitive(col1 int)");
        s.executeUpdate("insert into cujosql.sensitive values(1)");
        s.executeUpdate("insert into cujosql.sensitive values(2)");
        s.executeUpdate("insert into cujosql.sensitive values(3)");
        s.executeUpdate("insert into cujosql.sensitive values(4)");
        s.executeUpdate("insert into cujosql.sensitive values(5)");
        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another:  " + another.getMessage());
        }
    }
}


public void run(String sensitivity) {
    try {
        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

        // Fetch the five values that are there.
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        System.out.println("fetched the five rows...");
```

```
            // Note:  If you fetch the last row, the ResultSet looks
            //        closed and subsequent new rows that are added
            //        are not be recognized.

            // Allow another statement to insert a new value.
            Statement s2 = connection.createStatement();
            s2.executeUpdate("insert into cujosql.sensitive values(6)");
            s2.close();

            // Whether a row is recognized is based on the sensitivity setting.
            if (rs.next()) {
                System.out.println("There is a row now: " + rs.getInt(1));
            } else {
                System.out.println("No more rows.");
            }


        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("-------------------------------------");
            e.printStackTrace();
        }
        catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
            ex.printStackTrace();
        }
    }



    public void cleanup() {
        try {
            connection.close();
        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}
```

範例：使用 UDBDataSource 與 UDBConnectionPoolDataSource 設定連線儲存區作業

以下範例將說明，如何利用 UDBDataSource 與 UDBConnectionPoolDataSource 來使用連線儲存區作業。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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
    {
        // Create a ConnectionPoolDataSource implementation
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object");
```

```
        // Establish a JNDI context and bind the connection pool data source
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Create a standard data source that references it.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting pooling");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
  }
}
```

範例：SQLException

此爲攔截 SQLException 並傾出其中所有資訊的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;

public class ExceptionExample {

    public static Connection connection = null;

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

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

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

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


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

範例：暫停及回復異動

以下爲異動暫停之後又回復的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... doesn't exist
            }

            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();
            }
        }
    }


    /**
     * This test uses JTA support to handle transactions.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Assume the data source is backed by a UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // From the DataSource, obtain an XAConnection object that
            // contains an XAResource and a Connection object.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();

            // For XA transactions, a transaction identifier is required.
```

```
    // An implementation of the XID interface is not included with
    // the JDBC driver. Transactions with JTA for a
    // description of this interface to build a class for it.
    Xid xid = new XidImpl();

    // The connection from the XAResource can be used as any other
    // JDBC connection.
    Statement stmt = c.createStatement();

    // The XA resource must be notified before starting any
    // transactional work.
    xaRes.start(xid, XAResource.TMNOFLAGS);

    // Create a ResultSet during JDBC processing and fetch a row.
    ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
    rs.next();

    // The end method is called with the suspend option. The
    // ResultSets associated with the current transaction are 'on hold'.
    // They are neither gone nor accessible in this state.
    xaRes.end(xid, XAResource.TMSUSPEND);


    // Other work can be performed with the transaction.
    // As an example, you can create a statement and process a query.
    // This work and any other transactional work that the transaction may
    // perform is separate from the work done previously under the XID.
    Statement nonXAStmt = conn.createStatement();
    ResultSet nonXARS = nonXAStmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
    while (nonXARS.next()) {
        // Process here...
    }
    nonXARS.close();
    nonXAStmt.close();


    // If an attempt is made to use any suspended transactions
    // resources, an exception results.
    try {
        rs.getString(1);
        System.out.println("Value of the first row is " + rs.getString(1));
    } catch (SQLException e) {
        System.out.println("This was an expected exception - " +
                           "suspended ResultSet was used.");
    }


    // Resume the suspended transaction and complete the work on it.
    // The ResultSet is exactly as it was before the suspension.
    xaRes.start(newXid, XAResource.TMRESUME);
    rs.next();
    System.out.println("Value of the second row is " + rs.getString(1));


    // When the transaction has completed, end it
    // and commit any work under it.
    xaRes.end(xid, XAResource.TMNOFLAGS);
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);


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

```
                c.close();
            } catch (SQLException e) {
                System.out.println("Note:  Cleaup exception.");
                e.printStackTrace();
            }
        }
    }
}
```

範例：已暫停的 ResultSet

以下範例說明如何在另一個異動之下重新處理 Statement 物件來執行工作。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... does not exist
            }

            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();
            }
        }
    }


    /**
     * This test uses JTA support to handle transactions.
     */
    public void run() {
```

```
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Assume the data source is backed by a UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // From the DataSource, obtain an XAConnection object that
            // contains an XAResource and a Connection object.
            XAConnection  xaConn = ds.getXAConnection();
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();

            // For XA transactions, a transaction identifier is required.
            // An implementation of the XID interface is not included with
            // the JDBC driver. See Transactions with JTA
// for a description of this interface to build a
            // class for it.
            Xid xid = new XidImpl();

            // The connection from the XAResource can be used as any other
            // JDBC connection.
            Statement stmt = c.createStatement();

            // The XA resource must be notified before starting any
            // transactional work.
            xaRes.start(xid, XAResource.TMNOFLAGS);

            // Create a ResultSet during JDBC processing and fetch a row.
            ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
            rs.next();

            // The end method is called with the suspend option. The
            // ResultSets associated with the current transaction are 'on hold'.
            // They are neither gone nor accessible in this state.
            xaRes.end(xid, XAResource.TMSUSPEND);


            // In the meantime, other work can be done outside the transaction.
            // The ResultSets under the transaction can be closed if the
            // Statement object used to create them is reused.
            ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
            while (nonXARS.next()) {
                // Process here...
            }


            // Attempt to go back to the suspended transaction. The suspended
            // transaction's ResultSet has disappeared because the statement
            // has been processed again.
            xaRes.start(newXid, XAResource.TMRESUME);
            try {
                rs.next();
            } catch (SQLException ex) {
                System.out.println("This exception is expected. " +
                                   "The ResultSet closed due to another process.");
            }


            // When the transaction had completed, end it
            // and commit any work under it.
            xaRes.end(xid, XAResource.TMNOFLAGS);
            int rc = xaRes.prepare(xid);
            xaRes.commit(xid, false);
```

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

範例：測試連線儲存區作業的效能

以下範例將說明，如何測試儲存區作業範例的效能與非儲存區作業範例的效能。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
        // Do the work without a pool:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("\nStart timing the non-pooling DataSource version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));

        // Do the work with pooling:
        ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
}
```

範例：測試兩個 DataSource 的效能

以下範例將測試兩個 DataSource，一個僅採用連線儲存區作業，另一個同時採用陳述式與連線儲存區作業。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class StatementPoolingTest
{

    public static void main(java.lang.String[] args)
    throws Exception
    {
        Context ctx = new InitialContext();

        System.out.println("deploying statement pooling data source");
        deployStatementPoolDataSource();

        // Do the work with connection pooling only.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the connection pooling only version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));


        // Do the work with statement pooling added.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("\nStart timing the statement pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }



    private static void deployStatementPoolDataSource()
    throws Exception
    {
        // Create a ConnectionPoolDataSource implementation
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object with Statement pooling");
        cpds.setMaxStatements(10);

        // Establish a JNDI context and bind the connection pool data source
        Context ctx = new InitialContext();
        ctx.rebind("StatementSupport", cpds);


        // Create a standard datasource that references it.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting statement pooling");
        ds.setDataSourceName("StatementSupport");
```

```
        ctx.rebind("StatementPoolingDataSource", ds);

    }
}
```

範例：更新 BLOB

以下爲如何在應用程式中更新 BLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////
// UpdateBlobs is an example application
// that shows some of the APIs providing
// support for changing Blob objects
// and reflecting those changes to the
// database.
//
// This program must be run after
// the PutGetBlobs program has completed.
////////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Truncate a BLOB.
      blob1.truncate((long) 150000);
      System.out.println("Blob1's new length is " + blob1.length());

      // Update part of the BLOB with a new byte array.
      // The following code obtains the bytes that are at
      // positions 4000-4500 and set them to positions 500-1000.

      // Obtain part of the BLOB as a byte array.
      byte[] bytes = blob1.getBytes(4000L, 4500);

      int bytesWritten = blob2.setBytes(500L, bytes);

      System.out.println("Bytes written is " + bytesWritten);

      // The bytes are now found at position 500 in blob2
      long startInBlob2 = blob2.position(bytes, 1);

      System.out.println("pattern found starting at position " + startInBlob2);
```

```
      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：更新 CLOB

以下爲如何在應用程式中更新 CLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database. //
// This program must be run after
// the PutGetClobs program has completed.
////////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Truncate a CLOB.
      clob1.truncate((long) 150000);
      System.out.println("Clob1's new length is " + clob1.length());

      // Update a portion of the CLOB with a new String value.
      String value = "Some new data for once";
      int charsWritten = clob2.setString(500L, value);

      System.out.println("Characters written is " + charsWritten);

      // The bytes can be found at position 500 in clob2
      long startInClob2 = clob2.position(value, 1);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：使用具有多個異動的連線

以下範例說明如何使用單一連線來處理多個異動。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
    }


    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    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) {
                // Ignore... does not exist
            }

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

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


    /**
     * This test uses JTA support to handle transactions.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Assume the data source is backed by a UDBXADataSource.
            UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

            // From the DataSource, obtain an XAConnection object that
            // contains an XAResource and a Connection object.
            XAConnection  xaConn = ds.getXAConnection();
```

```
            XAResource    xaRes  = xaConn.getXAResource();
            Connection    c      = xaConn.getConnection();
            Statement stmt = c.createStatement();

            // For XA transactions, a transaction identifier is required.
            // This is not meant to imply that all the XIDs are the same.
            // Each XID must be unique to distinguish the various transactions
            // that occur.
            // Support for creating XIDs is again left to the application
            // program.
            Xid xid1 = JDXATest.xidFactory();
            Xid xid2 = JDXATest.xidFactory();
            Xid xid3 = JDXATest.xidFactory();

            // Do work under three transactions for this connection.
            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);

            // Prepare all the transactions
            int rc1 = xaRes.prepare(xid1);
            int rc2 = xaRes.prepare(xid2);
            int rc3 = xaRes.prepare(xid3);

            // Two of the transactions commit and one rolls back.
            // The attempt to insert the second value into the table is
            // not committed.
            xaRes.commit(xid1, false);
            xaRes.rollback(xid2);
            xaRes.commit(xid3, false);


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

範例：使用 BLOB

以下爲如何在應用程式中使用 BLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// UseBlobs is an example application
// that shows some of the APIs associated
// with Blob objects.
//
// This program must be run after
```

```
// the PutGetBlobs program has completed.
//////////////////////////////////////
import java.sql.*;

public class UseBlobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Determine the length of a LOB.
      long end = blob1.length();
      System.out.println("Blob1 length is " + blob1.length());

      // When working with LOBs, all indexing that is related to them
      // is 1-based, and is not 0-based like strings and arrays.
      long startingPoint = 450;
      long endingPoint = 500;

      // Obtain part of the BLOB as a byte array.
      byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

      // Find where a sub-BLOB or byte array is first found within a
      // BLOB. The setup for this program placed two identical copies of
      // a random BLOB into the database. Thus, the start position of the
      // byte array extracted from blob1 can be found in the starting
      // position in blob2. The exception would be if there were 50
      // identical random bytes in the LOBs previously.
      long startInBlob2 = blob2.position(outByteArray, 1);

      System.out.println("pattern found starting at position " + startInBlob2);

      c.close(); // Connection close closes stmt and rs too.
   }
}
```

範例：使用 CLOB

以下爲如何在應用程式中使用 CLOB 的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database. //
// This program must be run after
// the PutGetClobs program has completed.
//////////////////////////////////////
```

```
import java.sql.*;

public class UseClobs {
   public static void main(String[] args)
   throws SQLException
   {
       // Register the native JDBC driver.
       try {
          Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
      } catch (Exception e) {
          System.exit(1);  // Setup error.
      }

      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);


      // Determine the length of a LOB.
      long end = clob1.length();
      System.out.println("Clob1 length is " + clob1.length());

      // When working with LOBs, all indexing that is related to them
      // is 1-based, and not 0-based like strings and arrays.
      long startingPoint = 450;
      long endingPoint = 50;

      // Obtain part of the CLOB as a byte array.
      String outString = clob1.getSubString(startingPoint, (int)endingPoint);
      System.out.println("Clob substring is " + outString);

      // Find where a sub-CLOB or string is first found within a
      // CLOB. The setup for this program placed two identical copies of
      // a repeating CLOB into the database. Thus, the start position of the
      // string extracted from clob1 can be found in the starting
      // position in clob2 if the search begins close to the position where
   // the string starts.
      long startInClob2 = clob2.position(outString, 440);

      System.out.println("pattern found starting at position " + startInClob2);

      c.close(); // Connection close also closes stmt and rs.
   }
}
```

範例：使用 JTA 來處理異動

下列範例說明如何在應用程式中使用 Java Transaction API (JTA) 來處理異動。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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();
}


/**
 * Handle the previous cleanup run so that this test can recommence.
 */
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) {
            // Ignore... does not exist
        }

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


/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADataSource.
        UDBXADataSource ds = (UDBXADataSource) ctx.lookup("XADataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection  xaConn = ds.getXAConnection();
        XAResource    xaRes  = xaConn.getXAResource();
        Connection    c      = xaConn.getConnection();

        // For XA transactions, a transaction identifier is required.
        // An implementation of the XID interface is not included with the
        // JDBC driver. See Transactions with JTA for a description of
        // this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);
```

```
            // Standard JDBC work is performed.
            int count =
              stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is pretty fun.')");

            // When the transaction work has completed, the XA resource must
            // again be notified.
            xaRes.end(xid, XAResource.TMSUCCESS);

            // The transaction represented by the transaction ID is prepared
            // to be committed.
            int rc = xaRes.prepare(xid);

            // The transaction is committed through the XAResource.
            // The JDBC Connection object is not used to commit
            // the transaction when using JTA.
            xaRes.commit(xid, false);


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

範例：使用含有多個直欄的 meta 資料 ResultSet

以下範例說明如何使用含有多個直欄的 meta 資料 ResultSet。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
//////////////////////////////////////////////////////////////////////////////
//
// SafeGetUDTs example. This program demonstrates one way to deal with
// metadata ResultSets that have more columns in JDK 1.4 than they
// had in previous releases.
//
// Command syntax:
//    java SafeGetUDTs
//
//////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
```

```
// All rights reserved.
// 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;

    // Note: Static block runs before main begins.
    // Therefore, there is access to jdbcLevel in
    // main.
    {
        try {
            Class.forName("java.sql.Blob");

            try {
                Class.forName("java.sql.ParameterMetaData");
                // Found a JDBC 3.0 interface.  Must support JDBC 3.0.
                jdbcLevel = 3;
            } catch (ClassNotFoundException ez) {
                // Could not find the JDBC 3.0 ParameterMetaData class.
                // Must be running under a JVM with only JDBC 2.0
                // support.
                jdbcLevel = 2;
            }

        } catch (ClassNotFoundException ex) {
            // Could not find the JDBC 2.0 Blob class.  Must be
            // running under a JVM with only JDBC 1.0 support.
            jdbcLevel = 1;
        }
    }

    // Program entry point.
    public static void main(java.lang.String[] args)
    {
        Connection c = null;

        try {
            // Get the driver registered.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

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

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

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

                // Fetch all the columns that have been available since the
                // JDBC 2.0 release.
                System.out.println("TYPE_CAT is " + rs.getString("TYPE_CAT"));
                System.out.println("TYPE_SCHEM is " + rs.getString("TYPE_SCHEM"));
                System.out.println("TYPE_NAME is " + rs.getString("TYPE_NAME"));
                System.out.println("CLASS_NAME is " + rs.getString("CLASS_NAME"));
                System.out.println("DATA_TYPE is " + rs.getString("DATA_TYPE"));
                System.out.println("REMARKS is " + rs.getString("REMARKS"));
```

```
                // Fetch all the columns that were added in JDBC 3.0.
                if (jdbcLevel > 2) {
                    System.out.println("BASE_TYPE is " + rs.getString("BASE_TYPE"));
                }
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            if (c != null) {
                try {
                    c.close();
                } catch (SQLException e) {
                    // Ignoring shutdown exception.
                }
            }
        }
    }
}
```

範例：同時使用原有的 JDBC 與 IBM Toolbox for Java JDBC

以下範例說明如何在程式中使用原有的 JDBC 連線與 IBM Toolbox for Java JDBC 連線。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
///////////////////////////////////////////////////////////////////////////
//
// GetConnections example.
//
// This program demonstrates being able to use both JDBC drivers at
// once in a program. Two Connection objects are created in this
// program.
One is a native JDBC connection and one is a IBM Toolbox for Java
// JDBC connection.
//
// This technique is convenient because it allows you to use different
// JDBC drivers for different tasks concurrently. For example, the
// IBM Toolbox for Java JDBC driver is ideal for connecting to remote iSeries servers
// and the native JDBC driver is faster for local connections.
// You can use the strengths of each driver concurrently in your
// application by writing code similar to this example.
//
///////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind.  The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// 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)
   {
       // Verify input.
       if (args.length != 2) {
          System.out.println("Usage (CL command line):  java GetConnections PARM(<user> <password>)");
          System.out.println(" where <user> is a valid iSeries user ID");
          System.out.println("   and <password> is the password for that user ID");
          System.exit(0);
       }

       // Register both drivers.
      try {
         Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
         Class.forName("com.ibm.as400.access.AS400JDBCDriver");
      } catch (ClassNotFoundException cnf) {
         System.out.println("ERROR: One of the JDBC drivers did not load.");
         System.exit(0);
      }

      try {
         // Obtain a connection with each driver.
         Connection conn1 = DriverManager.getConnection("jdbc:db2://localhost", args[0], args[1]);
         Connection conn2 = DriverManager.getConnection("jdbc:as400://localhost", args[0], args[1]);

         // Verify that they are different.
         if (conn1 instanceof com.ibm.db2.jdbc.app.DB2Connection)
            System.out.println("conn1 is running under the native JDBC driver.");
         else
            System.out.println("There is something wrong with conn1.");

         if (conn2 instanceof com.ibm.as400.access.AS400JDBCConnection)
            System.out.println("conn2 is running under the IBM Toolbox for Java JDBC driver.");
         else
            System.out.println("There is something wrong with conn2.");

         conn1.close();
         conn2.close();
      } catch (SQLException e) {
         System.out.println("ERROR: " + e.getMessage());
      }
   }
}
```

鏈結集合

程式碼範例免責聲明

範例：使用 PreparedStatement 來取得 ResultSet

以下範例說明如何使用 PreparedStatement 物件的 executeQuery 方法來取得 ResultSet。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

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

public class PreparedStatementExample {

    public static void main(java.lang.String[] args)
    {
        // Load the following from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
```

```
String URL    = "jdbc:db2://*local";

// Register the native JDBC driver. If the driver cannot
// be registered, the test cannot continue.
try {
    Class.forName(DRIVER);
} catch (Exception e) {
    System.out.println("Driver failed to register.");
    System.out.println(e.getMessage());
    System.exit(1);
}

Connection c = null;
Statement s = null;

//    This program creates a table that is
//    used by prepared statements later.
try {
    // Create the connection properties.
    Properties properties = new Properties ();
    properties.put ("user", "userid");
    properties.put ("password", "password");

    // Connect to the local iSeries database.
    c = DriverManager.getConnection(URL, properties);

    // Create a Statement object.
    s = c.createStatement();
    // Delete the test table if it exists.  Note that
    // this example assumes throughout that the collection
    // MYLIBRARY exists on the system.
    try {
        s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
    } catch (SQLException e) {
        // Just continue... the table probably did not exist.
    }

    // Run an SQL statement that creates a table in the database.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

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


//    This program then uses a prepared statement to insert many
//    rows into the database.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Create a PreparedStatement object that is used to insert data into the
    // table.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

    for (int i = 0; i < nameArray.length; i++) {
        ps.setString(1, nameArray[i]);      // Set the Name from our array.
```

```
            ps.setInt(2, i+1);                  // Set the ID.
            ps.executeUpdate();
        }

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


    //    Use a prepared statement to query the database
    //    table that has been created and return data from it. In
    //    this example, the parameter used is arbitrarily set to
    //    5, meaning return all rows where the ID field is less than
    //    or equal to 5.
    try {
        ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
                                "WHERE ID <= ?");

        ps.setInt(1, 5);

        // Run an SQL query on the table.
        ResultSet rs = ps.executeQuery();
        // Display all the data in the table.
        while (rs.next ()) {
            System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
        }

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

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

    }
  }
}
```

範例：使用 Statement 物件的 executeUpdate 方法

以下範例將示範如何使用 Statement 物件的 executeUpdate 方法。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

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

public class StatementExample {

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

        // Suggestion:  Load these from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2:*local";

        // Register the native JDBC driver. If the driver cannot be
        // registered, the test cannot continue.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        try {
            // Create the connection properties.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Connect to the local iSeries database.
            c = DriverManager.getConnection(URL, properties);

            // Create a Statement object.
            s = c.createStatement();
            // Delete the test table if it exists. Note: This
            // example assumes that the collection MYLIBRARY
            // exists on the system.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Just continue... the table probably does not exist.
            }

            // Run an SQL statement that creates a table in the database.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

            // Run some SQL statements that insert records into the table.
            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)");

            // Run an SQL query on the table.
            ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

            // Display all the data in the table.
            while (rs.next()) {
                System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
            }
```

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

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

範例：JAAS HelloWorld

這些範例可說明編譯及執行 JAAS 的 HelloWorld 所需的三個檔案。

HelloWorld.java

以下是 HelloWorld.java 檔案的原始檔。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/*
 * ===========================================================================
 * Licensed Materials - Property of IBM
 *
 * (C) Copyright IBM Corp. 2000 All Rights Reserved.
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * ===========================================================================
 *
  * File: HelloWorld.java
 */

import java.io.*;
import java.util.*;
import java.security.Principal;
import java.security.PrivilegedAction;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;

/**
 * This SampleLogin application attempts to authenticate a user.
 *
 * If the user successfully authenticates itself,
 * the user name and number of Credentials is displayed.
 *
 * @version 1.1, 09/14/99
 */
public class HelloWorld {
```

```
/**
 * Attempt to authenticate the user.
 */
public static void main(String[]args) {
    // use the configured LoginModules for the "helloWorld" entry
    LoginContext lc = null;
    try {
        lc = new LoginContext("helloWorld", new MyCallbackHandler());
    } catch (LoginException le) {
        le.printStackTrace();
        System.exit(-1);
    }

    // the user has 3 attempts to authenticate successfully
    int i;
    for (i = 0; i < 3; i++) {
        try {

            // attempt authentication
            lc.login();

            // if we return with no exception, authentication succeeded
            break;

        } catch (AccountExpiredException aee) {

            System.out.println("Your account has expired");
            System.exit(-1);

        } catch (CredentialExpiredException cee) {

            System.out.println("Your credentials have expired.");
            System.exit(-1);

        } catch (FailedLoginException fle) {

            System.out.println("Authentication Failed");
                try {
                    Thread.currentThread().sleep(3000);
                } catch (Exception e) {
                    // ignore
                }

        } catch (Exception e) {

            System.out.println("Unexpected Exception - unable to continue");
            e.printStackTrace();
            System.exit(-1);
        }
    }

    // did they fail three times?
    if (i == 3) {
        System.out.println("Sorry");
        System.exit(-1);
    }

    // Look at what Principals we have:
    Iterator principalIterator = lc.getSubject().getPrincipals().iterator();
    System.out.println("\n\nAuthenticated user has the following Principals:");
    while (principalIterator.hasNext()) {
        Principal p = (Principal)principalIterator.next();
        System.out.println("\t" + p.toString());
    }

    // Look at some Principal-based work:
```

```
        Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
            public Object run() {
                System.out.println("\nYour java.home property: "
                                +System.getProperty("java.home"));

                System.out.println("\nYour user.home property: "
                                +System.getProperty("user.home"));

                File f = new File("foo.txt");
                System.out.print("\nfoo.txt does ");
                if (!f.exists()) System.out.print("not ");
                System.out.println("exist in your current directory");

                System.out.println("\nOh, by the way ...");

                try {
                    Thread.currentThread().sleep(2000);
                } catch (Exception e) {
                    // ignore
                }
                System.out.println("\n\nHello World!\n");
                return null;
            }
        }, null);
        System.exit(0);
    }
}


/**
 * The application must implement the CallbackHandler.
 *
 * This application is text-based.  Therefore it displays information
 * to the user using the OutputStreams System.out and System.err,
 * and gathers input from the user using the InputStream, System.in.
 */
class MyCallbackHandler implements CallbackHandler {

    /**
     * Invoke an array of Callbacks.
     *
     *
     * @param callbacks an array of Callback objects which contain
     *          the information requested by an underlying security
     *          service to be retrieved or displayed.
     *
     * @exception java.io.IOException if an input or output error occurs.
     *
     * @exception UnsupportedCallbackException if the implementation of this
     *          method does not support one or more of the Callbacks
     *          specified in the callbacks parameter.
     */
    public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException {

    for (int i = 0; i < callbacks.length; i++) {
        if (callbacks[i] instanceof TextOutputCallback) {

        // display the message according to the specified type
        TextOutputCallback toc = (TextOutputCallback)callbacks[i];
        switch (toc.getMessageType()) {
        case TextOutputCallback.INFORMATION:
            System.out.println(toc.getMessage());
            break;
        case TextOutputCallback.ERROR:
            System.out.println("ERROR: " + toc.getMessage());
            break;
```

```
    case TextOutputCallback.WARNING:
        System.out.println("WARNING: " + toc.getMessage());
        break;
    default:
        throw new IOException("Unsupported message type: " +
                toc.getMessageType());
    }

    } else if (callbacks[i] instanceof NameCallback) {

    // prompt the user for a user name
    NameCallback nc = (NameCallback)callbacks[i];

    // ignore the provided defaultName
    System.err.print(nc.getPrompt());
    System.err.flush();
    nc.setName((new BufferedReader
        (new InputStreamReader(System.in))).readLine());

    } else if (callbacks[i] instanceof PasswordCallback) {

    // prompt the user for sensitive information
    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");
    }
}
}

// Reads user password from given input stream.
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

以下是 HWLoginModule.java 的原始檔。

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/*
 * ============================================================================
 * Licensed Materials - Property of IBM
 *
 * (C) Copyright IBM Corp. 2000 All Rights Reserved.
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * ============================================================================
 *
 * File: HWLoginModule.java
 */

package com.ibm.security;

import java.util.*;
import java.io.IOException;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;
import com.ibm.security.HWPrincipal;

/**
 * This LoginModule authenticates users with a password.
 *
 * This LoginModule only recognizes any user who enters
 *     the required password:   Go JAAS
 *
 * If the user successfully authenticates itself,
 * a HWPrincipal with the user name
 * is added to the Subject.
 *
 * This LoginModule recognizes the debug option.
 * If set to true in the login Configuration,
 * debug messages are sent to the output stream, System.out.
 *
 * @version 1.1, 09/10/99
 */
```

```
public class HWLoginModule implements LoginModule {

    // initial state
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;

    // configurable option
    private boolean debug = false;

    // the authentication status
    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    // user name and password
    private String user name;
    private char[] password;

    private HWPrincipal userPrincipal;

    /**
     * Initialize this LoginModule.
     *
     * @param subject the Subject to be authenticated.
     *
     * @param callbackHandler a CallbackHandler for communicating
     *          with the end user (prompting for user names and
     *          passwords, for example).
     *
     * @param sharedState shared LoginModule state.
     *
     * @param options options specified in the login
     *          Configuration for this particular
     *          LoginModule.
     */
    public void initialize(Subject subject, CallbackHandler callbackHandler,
            Map sharedState, Map options) {

    this.subject = subject;
    this.callbackHandler = callbackHandler;
    this.sharedState = sharedState;
    this.options = options;

    // initialize any configured options
    debug = "true".equalsIgnoreCase((String)options.get("debug"));
    }

    /**
     * Authenticate the user by prompting for a user name and password.
     *
     *
     * @return true in all cases since this LoginModule
     *      should not be ignored.
     *
     * @exception FailedLoginException if the authentication fails.
     *
     * @exception LoginException if this LoginModule
     *      is unable to perform the authentication.
     */
    public boolean login() throws LoginException {

    // prompt for a user name and password
    if (callbackHandler == null)
        throw new LoginException("Error: no CallbackHandler available " +
            "to garner authentication information from the user");
```

```
Callback[] callbacks = new Callback[2];
callbacks[0] = new NameCallback("\n\nHWModule user name: ");
callbacks[1] = new PasswordCallback("HWModule password: ", false);

try {
    callbackHandler.handle(callbacks);
    user name = ((NameCallback)callbacks[0]).getName();
    char[] tmpPassword = ((PasswordCallback)callbacks[1]).getPassword();
    if (tmpPassword == null) {
    // treat a NULL password as an empty password
    tmpPassword = new char[0];
    }
    password = new char[tmpPassword.length];
    System.arraycopy(tmpPassword, 0,
        password, 0, tmpPassword.length);
    ((PasswordCallback)callbacks[1]).clearPassword();

} catch (java.io.IOException ioe) {
    throw new LoginException(ioe.toString());
} catch (UnsupportedCallbackException uce) {
    throw new LoginException("Error: " + uce.getCallback().toString() +
    " not available to garner authentication information " +
    "from the user");
}

// print debugging information
if (debug) {
    System.out.println("\n\n\t[HWLoginModule] " +
            "user entered user name: " +
            user name);
    System.out.print("\t[HWLoginModule] " +
            "user entered password: ");
    for (int i = 0; i > password.length; i++)
    System.out.print(password[i]);
    System.out.println();
}

// verify the password
if (password.length == 7 &&
    password[0] == 'G' &&
    password[1] == 'o' &&
    password[2] == ' ' &&
    password[3] == 'J' &&
    password[4] == 'A' &&
    password[5] == 'A' &&
    password[6] == 'S') {

    // authentication succeeded!!!
    if (debug)
    System.out.println("\n\t[HWLoginModule] " +
            "authentication succeeded");
    succeeded = true;
    return true;
} else {

    // authentication failed -- clean out state
    if (debug)
    System.out.println("\n\t[HWLoginModule] " +
            "authentication failed");
    succeeded = false;
    user name = null;
    for (int i = 0; i < password.length; i++)
    password[i] = ' ';
    password = null;
    throw new FailedLoginException("Password Incorrect");
}
}
```

```
/**
 * This method is called if the overall authentication of LoginContext
 * succeeded
 * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
 * succeeded).
 *
 * If this LoginModule authentication attempt
 * succeeded (checked by retrieving the private state saved by the
 * login method), then this method associates a
 * SolarisPrincipal
 * with the Subject located in the
 * LoginModule.  If this LoginModule
 * authentication attempt failed, then this method removes
 * any state that was originally saved.
 *
 * @exception LoginException if the commit fails.
 *
 * @return true if the login and commit LoginModule
 *      attempts succeeded, or false otherwise.
 */
public boolean commit() throws LoginException {
if (succeeded == false) {
    return false;
} else {
    // add a Principal (authenticated identity)
    // to the Subject

    // assume the user we authenticated is the HWPrincipal
    userPrincipal = new HWPrincipal(user name);
    final Subject s = subject;
    final HWPrincipal sp = userPrincipal;
    java.security.AccessController.doPrivileged
    (new java.security.PrivilegedAction() {
    public Object run() {
        if (!s.getPrincipals().contains(sp))
        s.getPrincipals().add(sp);
        return null;
    }
    });

    if (debug) {
    System.out.println("\t[HWLoginModule] " +
            "added HWPrincipal to Subject");
    }

    // in any case, clean out state
    user name = null;
    for (int i = 0; i > password.length; i++)
    password[i] = ' ';
    password = null;

    commitSucceeded = true;
    return true;
}
}

/**
 * This method is called if the overall authentication of LoginContext
 * failed.
 * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
 * did not succeed).
 *
 * If this authentication attempt of LoginModule
 * succeeded (checked by retrieving the private state saved by the
 * login and commit methods),
 * then this method cleans up any state that was originally saved.
```

```
     *
     * @exception LoginException if the abort fails.
     *
     * @return false if this login or commit attempt for LoginModule
     *      failed, and true otherwise.
     */
    public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        // login succeeded but overall authentication failed
        succeeded = false;
        user name = null;
        if (password != null) {
        for (int i = 0; i > password.length; i++)
            password[i] = ' ';
        password = null;
        }
        userPrincipal = null;
    } else {
        // overall authentication succeeded and commit succeeded,
        // but another commit failed
        logout();
    }
    return true;
    }

    /**
     * Logout the user.
     *
     * This method removes the HWPrincipal
     * that was added by the commit method.
     *
     * @exception LoginException if the logout fails.
     *
     * @return true in all cases since this LoginModule
     *          should not be ignored.
     */
    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

以下是 HWPrincipal.java 的原始檔。

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/*
 * ============================================================================
 * Licensed Materials - Property of IBM
 *
 * (C) Copyright IBM Corp. 2000 All Rights Reserved.
 *
 *  US Government Users Restricted Rights - Use, duplication or
 *  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * ============================================================================
 *
 * File: HWPrincipal.java
 */

package com.ibm.security;

import java.security.Principal;

/**
 * This class implements the Principal interface
 * and represents a HelloWorld tester.
 *
 * @version 1.1, 09/10/99
 * @author D. Kent Soper
 */
public class HWPrincipal implements Principal, java.io.Serializable {

    private String name;

    /*
     * Create a HWPrincipal with the supplied name.
     */
    public HWPrincipal(String name) {
        if (name == null)
            throw new NullPointerException("illegal null input");

        this.name = name;
    }

    /*
     * Return the name for the HWPrincipal.
     */
    public String getName() {
        return name;
    }

    /*
     * Return a string representation of the HWPrincipal.
     */
    public String toString() {
        return("HWPrincipal:  " + name);
    }

    /*
     * Compares the specified Object with the HWPrincipal for equality.
     * Returns true if the given object is also a HWPrincipal and the
     * two HWPrincipals have the same user name.
     */
    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;
    }

    /*
     * Return a hash code for the HWPrincipal.
     */
    public int hashCode() {
        return name.hashCode();
    }
}
```

鏈結集合

程式碼範例免責聲明

程式碼範例免責聲明

程式碼範例免責聲明

範例：JAAS SampleThreadSubjectLogin

以下範例顯示 SampleThreadSubjectLogin 類別的實作方式。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
////////////////////////////////////////////////////////////////////
//
// 5722-JV1
// (C) Copyright IBM Corp. 2000
//
////////////////////////////////////////////////////////////////////
//
// File Name:   SampleThreadSubjectLogin.java
//
// Class:       SampleThreadSubjectLogin
//
////////////////////////////////////////////////////////////////////
//
// CHANGE ACTIVITY:
//
//
// END CHANGE ACTIVITY
//
////////////////////////////////////////////////////////////////////

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.*;


/**
 * This SampleThreadSubjectLogin application authenticates a single
 * user, swaps the OS thread identity to the authenticated user,
 * and then writes "Hello World" into a privately authorized
 * file, thread.txt, in the user's test directory.
 *
 * The user is requested to enter the user id and password to
 * authenticate.
 *
 * If successful, the user name and number of Credentials
 * are displayed.
 *
 *
```

設定和執行指示：

1) 利用 *USER 類別權限，呼叫
"CRTUSRPRF USRPRF(JAAS13) PASSWORD() TEXT('JAAS sample user id')"
來建立新的使用者 JAAS13。
2) 配置虛擬測試檔 "**yourTestDir**/thread.txt"，
將此檔案的 *RWX 權限私自授予 JAAS13 來取得寫入權。
3) 將 SampleThreadSubjectLogin.java 複製到測試目錄中。

4) 從現行目錄切換至測試目錄，編譯 java 原始程式碼。
輸入 -

```
strqsh

cd 'yourTestDir'

javac -J-Djava.version=1.3
  -classpath /qibm/proddata/os400/java400/ext/jaas13.jar:
             /QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar:.
  -d ./classes
  *.java
```

5) 將 threadLogin.config、threadJaas.policy 及 threadJava2.policy
複製到測試目錄中。
6) 新增符號鏈結來指向 jaas13.jar 檔的延伸套件目錄 (若尚未完成)。
延伸類別載入器正常應該會載入此 JAR 檔。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas13.jar')
    NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jaas13.jar')
```

7) 為了執行此範例，請新增符號鏈結來指向 jt400.jar 和 jt400ntv.jar
檔案的延伸套件目錄 (若尚未完成)。如此一來，
延伸類別載入器就可以載入這些檔案。另外，在 CLASSPATH 中包含這些檔案，
應用程式類別載入器一樣可以載入這些檔案。
若是從類別路徑目錄中載入這些檔案，請勿新增鏈結來指向延伸套件目錄。
jaas13.jar 檔案需要這些 JAR 檔案，才能取得 IBM Toolbox
for Java (5722-JC1) 授權程式產品的認證實作類別。
(請參閱 IBM Toolbox for Java 主題中關於認證類別的文件，
在左側頁框裡選取「Security 類型 => 鑑別」。然後選取
ProfileTokenCredential 類別的鏈結。在頂端選取 This Package
來顯示完整的 com/ibm/as400/security/auth Java 套件。
在左側頁框中選取 Javadoc =>「Access 類別」，
也可以找到鑑別類別的 Javadoc。 在頂端選取 All Packages，
然後尋找 com.ibm.as400.security.* 套件)

```
ADDLNK OBJ('/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar')
    NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400.jar')
```

```
  ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400Native.jar')

////////////////////////////////////
 重要注意事項 -
////////////////////////////////////

 在更新實際應用程式的 Java2 原則檔時，請記得授予
 適當的許可權給 IBM Toolbox for Java JAR 檔案的實際位置。
 即使這些位置已透過符號鏈結的方式，指向先前列出的延伸套件目錄，
java.security.AllPermission in the
 ${java.home}/lib/security/java.policy file, authorization is based on
 the actual location of the JAR files.

 例如，為了順利使用 IBM Toolbox for Java 的認證類別，
 您需要在應用程式的 Java2 原則檔中新增下列許可權 -
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";
 }

  亦需為應用程式的 codeBase 新增這些許可權，
 因為 IBM Toolbox for Java JAR 檔所執行的作業，
 並非於特許模式下執行。
 此範例在 threadJava2.policy 檔案中省略 codeBase 參數，
 因此已授予這些許可權給所有的 java 類別。
 8) 請確定 Host Servers 已啓動且正在執行。
  使用 IBM Toolbox for Java (亦即 jt400.jar) 內的
 ProfileTokenCredential 類別作為認證，
 由 SampleThreadSubjectLogin.java 程式附加給通過鑑別的主旨。
 IBM Toolbox for Java 認證類別需要存取 Host Servers。
 9) 以不具備 yourTestDir/thread.txt 存取權的使用者身份登入，
 呼叫 SampleThreadSubjectLogin。
 10) 輸入下列 CL 指令來啓動範例 =>

  CHGCURDIR DIR('yourTestDir')

  JAVA CLASS(SampleThreadSubjectLogin)
    CLASSPATH('yourTestDir/classes')
    PROP((java.version '1.3')
       (java.security.manager)
       (java.security.auth.login.config
        'yourTestDir/threadLogin.config')
       (java.security.policy
        'yourTestDir/threadJava2.policy')
       (java.security.auth.policy
        'yourTestDir/threadJaas.policy'))

  出現提示時，輸入步驟 1 的使用者 ID 和密碼。

  11) 檢查 yourTestDir/thread.txt 是否包含 "Hello World" 這幾個字。

 *
 **/


public class SampleThreadSubjectLogin {
/**
 * Attempt to authenticate the user.
 *
 * @param args
 *      Input arguments for this application (ignored).
 *
 */
```

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

// use the configured LoginModules for the "AS400ToolboxApp" entry
LoginContext lc = null;
try {
    // if provided, the same subject is used for multiple login attempts
    lc = new LoginContext("AS400ToolboxApp",
              new Subject(),
              new SampleCBHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}

// the user has 3 attempts to authenticate successfully
int i;
for (i = 0; i < 3; i++) {
    try {

    // attempt authentication
    lc.login();

    // if we return with no exception, authentication succeeded
    break;

    } catch (AccountExpiredException aee) {

    System.out.println("Your account has expired");
    System.exit(-1);

    } catch (CredentialExpiredException cee) {

    System.out.println("Your credentials have expired.");
    System.exit(-1);

    } catch (FailedLoginException fle) {

    System.out.println("Authentication Failed");
    try {
        Thread.currentThread().sleep(3000);
    } catch (Exception e) {
        // ignore
    }

    } catch (Exception e) {

    System.out.println("Unexpected Exception - unable to continue");
    e.printStackTrace();
    System.exit(-1);
    }
}

// did they fail three times?
if (i == 3) {
    System.out.println("Sorry authentication failed");
    System.exit(-1);
}


// display authenticated principals & credentials
System.out.println("Authentication Succeeded");

System.out.println("Principals:");

Iterator itr = lc.getSubject().getPrincipals().iterator();

while (itr.hasNext())
```

```
        System.out.println(itr.next());

    itr = lc.getSubject().getPrivateCredentials().iterator();

    while (itr.hasNext())
        System.out.println(itr.next());

    itr = lc.getSubject().getPublicCredentials().iterator();

    while (itr.hasNext())
        System.out.println(itr.next());


        // let's do some Principal-based work:
    ThreadSubject.doAsPrivileged(lc.getSubject(), new java.security.PrivilegedAction() {
           public Object run() {
           System.out.println("\nYour java.home property: "
                  +System.getProperty("java.home"));
           System.out.println("\nYour user.home property: "
                      +System.getProperty("user.home"));
           File f = new File("thread.txt");
           System.out.print("\nthread.txt does ");
           if (!f.exists()) System.out.print("not ");
           System.out.println("exist in your current directory");

           try {
               // write "Hello World number   x"  into thread.txt
               PrintStream ps = new PrintStream(new FileOutputStream("thread.txt", true), true);

               long flen = f.length();
               ps.println("Hello World number    " +
                  Long.toString(flen/22) +
                  "\n");
               ps.close();
           } catch (Exception e) {
               e.printStackTrace();
           }

           System.out.println("\nOh, by the way, " + SampleThreadSubjectLogin.getCurrentUser());
           try {
               Thread.currentThread().sleep(2000);
           } catch (Exception e) {
            // ignore
               }
           System.out.println("\n\nHello World!\n");
           return null;
           }
       }, null);

       System.exit(0);

    }// end main()


// Returns the current OS identity for the main thread of the application.
// (This routine uses classes from IBM Toolbox for Java)
// Note - Applications running on a secondary thread cannot use this API to determine the current user.
    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);

    // Invoke the QUSRJOBI API
        qusrjobi.run();

        byte[] uidBytes = new byte[10];
        System.arraycopy((qusrjobi.getParameterList())[0].getOutputData(), 90, uidBytes, 0, 10);

        return ((String)(uidText.toObject(uidBytes))).trim();
    }


    catch (Exception e) {
        e.printStackTrace();
    }

    return "";
    }

} //end SampleThreadSubjectLogin class



/**
 * A CallbackHandler is passed to underlying security
 * services so that they may interact with the application
 * to retrieve specific authentication data,
 * such as user names and passwords, or to display certain
 * information, such as error and warning messages.
 *
 * CallbackHandlers are implemented in an application
 * and platform-dependent fashion. The implementation decides
 * how to retrieve and display information depending on the
 * Callbacks passed to it.
 *
 * This class provides a sample CallbackHandler for applications
 * running in an i5/OS environment. However, it is not intended
 * to fulfill the requirements of production applications.
 * As indicated, the CallbackHandler is ultimately considered to
 * be application-dependent, as individual applications have
 * unique error checking, data handling, and user
 * interface requirements.
 *
 * The following callbacks are handled:
 *
```

- *
- NameCallback *
- PasswordCallback *
- TextOutputCallback *

```
 *
 * For simplicity, prompting is handled interactively through
 * standard input and output. However, it is worth noting
 * that when standard input is provided by the console, this
 * approach allows passwords to be viewed as they are
 * typed. This should be avoided in production
 * applications.
```

```
 *
 * This CallbackHandler also allows a name and password
 * to be acquired through an alternative mechanism
 * and set directly on the handler to bypass the need for
 * user interaction on the respective Callbacks.
 *
 */
class SampleCBHandler implements CallbackHandler {
    private String name_ = null;
    private String password_ = null;
/**
 * Constructs a new SampleCBHandler.
 *
 */
public SampleCBHandler() {
    this(null, null);
}
/**
 * Constructs a new SampleCBHandler.
 *
 * A name and password can optionally be specified in
 * order to bypass the need to prompt for information
 * on the respective Callbacks.
 *
 * @param name
 *      The default value for name callbacks. A null
 *      value indicates that the user should be
 *      prompted for this information. A non-null value
 *      cannot be zero length or exceed 10 characters.
 *
 * @param password
 *      The default value for password callbacks. A null
 *      value indicates that the user should be
 *      prompted for this information. A non-null value
 *      cannot be zero length or exceed 10 characters.
 */
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;
}
/**
 * Handle the given name callback.
 *
 * First check to see if a name has been passed in
 * on the constructor.  If so, assign it to the
 * callback and bypass the prompt.
 *
 * If a value has not been preset, attempt to prompt
 * for the name using standard input and output.
 *
 * @param c
 *      The NameCallback.
 *
 * @exception java.io.IOException
 *      If an input or output error occurs.
 *
 */
private void handleNameCallback(NameCallback c) throws IOException {
    // Check for cached value
    if (name_ != null) {
```

```
        c.setName(name_);
        return;
    }
    // No preset value; attempt stdin/out
    c.setName(
        stdIOReadName(c.getPrompt(), 10));
}
/**
 * Handle the given name callback.
 *
 * First check to see if a password has been passed
 * in on the constructor.  If so, assign it to the
 * callback and bypass the prompt.
 *
 * If a value has not been preset, attempt to prompt
 * for the password using standard input and output.
 *
 * @param c
 *      The PasswordCallback.
 *
 * @exception java.io.IOException
 *      If an input or output error occurs.
 *
 */
private void handlePasswordCallback(PasswordCallback c) throws IOException {
    // Check for cached value
    if (password_ != null) {
        c.setPassword(password_.toCharArray());
        return;
    }

    // No preset value; attempt stdin/out
    // Note - Not for production use.
    //        Password is not concealed by standard console I/O
    if (c.isEchoOn())
        c.setPassword(
            stdIOReadName(c.getPrompt(), 10).toCharArray());
    else
    {

        // Note - Password is not concealed by standard console I/O
        c.setPassword(stdIOReadName(c.getPrompt(), 10).toCharArray());

    }
}
/**
 * Handle the given text output callback.
 *
 * If the text is informational or a warning,
 * text is written to standard output. If the
 * callback defines an error message, text is
 * written to standard error.
 *
 * @param c
 *      The TextOutputCallback.
 *
 * @exception java.io.IOException
 *      If an input or output error occurs.
 *
 */
private void handleTextOutputCallback(TextOutputCallback c) throws IOException {
    if (c.getMessageType() == TextOutputCallback.ERROR)
        System.err.println(c.getMessage());
    else
        System.out.println(c.getMessage());
}
/**
```

```
 * Retrieve or display the information requested in the
 * provided Callbacks.
 *
 * The handle method implementation
 * checks the instance(s) of the Callback
 * object(s) passed in to retrieve or display the
 * requested information.
 *
 * @param callbacks
 *      An array of Callback objects provided
 *      by an underlying security service which contains
 *      the information requested to be retrieved or displayed.
 *
 * @exception java.io.IOException
 *      If an input or output error occurs.
 *
 * @exception UnsupportedCallbackException
 *      If the implementation of this method does not support
 *      one or more of the Callbacks specified in the
 *      callbacks parameter.
 *
 */
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]);
    }
}
/**
 * Displays the given string using standard output,
 * followed by a space to separate from subsequent
 * input.
 *
 * @param prompt
 *      The text to display.
 *
 * @exception IOException
 *      If an input or output error occurs.
 *
 */
private void stdIOPrompt(String prompt) throws IOException {
    System.out.print(prompt + ' ');
    System.out.flush();
}
/**
 * Reads a String from standard input, stopped at
 * maxLength or by a newline.
 *
 * @param prompt
 *      The text to display to standard output immediately
 *      prior to reading the requested value.
 *
 * @param maxLength
 *      Maximum length of the String to return.
 *
 * @return
```

```
 *      The entered string. The value returned does
 *      not contain leading or trailing whitespace
 *      and is converted to uppercase.
 *
 * @exception IOException
 *      If an input or output error occurs.
 *
 */
private String stdIOReadName(String prompt, int maxLength) throws IOException {
    stdIOPrompt(prompt);
    String s =
        (new BufferedReader
            (new InputStreamReader(System.in))).readLine().trim();
    if (s.length() < maxLength)
        s = s.substring(0,maxLength);
    return s.toUpperCase();
}

}//end SampleCBHandler class
```

鏈結集合

程式碼範例免責聲明

範例：IBM JGSS 非 JAAS 用戶端程式

有關用戶端程式範例的資訊，請參閱下載及執行範例程式。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// IBM JGSS 1.0 Sample Client Program

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.*;

/**
  * A JGSS sample client;
  * to be used in conjunction with the JGSS sample server.
  * The client first establishes a context with the server
  * and then sends wrapped message followed by a MIC to the server.
  * The MIC is calculated over the plain text that was wrapped.
  * The client requires to server to authenticate itself
  * (mutual authentication) during context establishment.
  * It also delegates its credentials to the server.
 *
  * It sets the JAVA variable
  * javax.security.auth.useSubjectCredsOnly to false
  * so that JGSS will not acquire credentials through JAAS.
 *
  * The client takes input parameters, and complements it
  * with information from the jgss.ini file; any required input not
  * supplied on the command line is taking from the jgss.ini file.
 *
  * Usage: Client [options]
 *
  * The -? option produces a help message including supported options.
*
  * This sample client does not use JAAS.
  * The client can be run against the JAAS sample client and server.
  * See {@link JAASClient JAASClient} for a sample client that uses 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 name] [-s serverName]"
        + "\n\t[-h serverHost [:port]] [-p port] [-m msg]"
        + "\n"
        + "\n  -?\t\t\thelp; produces this message"
        + "\n  -n name\t\tthe client's principal name (without realm)"
        + "\n  -s serverName\t\tthe server's principal name (without realm)"
        + "\n  -h serverHost[:port]\tthe server's hostname"
        +         " (and optional port number)"
        + "\n  -p port\t\tthe port on which the server will be listening"
        + "\n  -m msg\t\tmessage to send to the server";


        // Caller must call initialize (may need to call processArgs first).
    public Client (String programName) throws Exception
    {
        testUtil = new Util();
        if (programName != null)
        {
            program = programName;
            debugPrefix = programName + ": ";
        }
    }

        // Caller must call initialize (may need to call processArgs first).
        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);

        // Property not set. Set it to the specified value.

        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
{
    // peer's name
    if (serverNameWithoutRealm != null)
    {
        this.serverName = serverNameWithoutRealm;
    }
    else
    {
        this.serverName = testUtil.getDefaultServicePrincipalWithoutRealm();
    }

    // peer's host
    if (serverHostname != null)
    {
```

```
        this.serviceHostname = serverHostname;
    }
    else
    {
        this.serviceHostname = testUtil.getDefaultServiceHostname();
    }

    // peer's port
    if (serverPort > 0)
    {
        this.servicePort = serverPort;
    }
    else
    {
        this.servicePort = testUtil.getDefaultServicePort();
    }

    // message for peer
    if (message != null)
    {
        this.data = message;
    }
    else
    {
        this.data = "The quick brown fox jumps over the lazy dog";
    }

    this.dataBytes = this.data.getBytes();

    tcp = new TCPComms(serviceHostname, servicePort);
}


void initialize() throws Exception
{
     Oid krb5MechanismOid = new Oid("1.2.840.113554.1.2.2");

    if (gssCred == null)
    {
        if (myName != null)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                                + "creating GSSName USER_NAME for "
                                + myName);

            gssName = mgr.createName(
                           myName,
                           GSSName.NT_USER_NAME,
                            krb5MechanismOid);

            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                                + "Canonicalized GSSName=" + gssName);
        }
        else
            gssName = null; // for default credentials


        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");
            }
            // Add more later!
        }

        byte[] response = null;
        byte[] request = null;
        int len = 0;
        boolean done = false;
        do {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "Calling initSecContext");

            request = context.initSecContext(response, 0, len);

            if (request != null)
            {
                debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                  + "Sending initial context token");

                tcp.send(request);
            }
            done = context.isEstablished();

            if (!done)
            {
                debug.out(Debug.OPTS_CAT_APPLICATION,
                    debugPrefix + "Receiving response token");

                byte[] temp = tcp.receive();
                response = temp;
                len = response.length;
            }
        } while(!done);

        debug.out(Debug.OPTS_CAT_APPLICATION,
              debugPrefix + "context established with acceptor");

    } catch (Exception exc) {
        exc.printStackTrace();
        throw exc;
    }
}

void doMIC() throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "generating MIC");
    byte[] mic = context.getMIC(dataBytes, 0, dataBytes.length, null);

    if (mic != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "sending MIC");
        tcp.send(mic);
```

```
    }
    else
        debug.out(Debug.OPTS_CAT_APPLICATION,
                          debugPrefix + "getMIC Failed");
}

void doWrap() throws Exception
{
    MessageProp mp = new MessageProp(true);
    mp.setPrivacy(context.getConfState());

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "wrapping message");

    byte[] wrapped = context.wrap(dataBytes, 0, dataBytes.length, mp);

    if (wrapped != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION,
                debugPrefix + "sending wrapped message");

        tcp.send(wrapped);
    }
    else
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "wrap Failed");
}

void printUsage()
{
    System.out.println(program + usageString);
}

void processArgs(String[] args) throws Exception
{
    String port            = null;
    String myName          = null;
    int servicePort               = 0;
    String serviceHostname = null;

    String sHost = null;
    String msg = null;

    GetOptions options = new GetOptions(args, "?h:p:m:n:s:");
    int ch = -1;
    while ((ch = options.getopt()) != options.optEOF)
    {
        switch(ch)
        {
            case '?':
                printUsage();
                System.exit(1);

            case 'h':
                if (sHost == null)
                {
                    sHost = options.optArgGet();
                    int p = sHost.indexOf(':');
                    if (p != -1)
                    {
                        String temp1 = sHost.substring(0, p);
                        if (port == null)
                            port = sHost.substring(p+1, sHost.length()).trim();
                        sHost = temp1;
                    }
                }
                continue;

            case 'p':
```

```
                if (port == null)
                    port = options.optArgGet();
                continue;

            case 'm':
                if (msg == null)
                    msg = options.optArgGet();
                continue;

            case 'n':
                if (myName == null)
                    myName = options.optArgGet();
                continue;

            case 's':
                if (serverName == null)
                    serverName = options.optArgGet();
                continue;
        }
    }

    if ((port != null) && (port.length() > 0))
    {
        int p = -1;
        try {
            p = Integer.parseInt(port);
        } catch (Exception exc) {
            System.out.println("Bad port input: "+port);
        }

        if (p != -1)
            servicePort = p;
    }

    if ((sHost != null) && (sHost.length() > 0)) {
            serviceHostname = sHost;
    }

    init(myName, serverName, serviceHostname, servicePort, msg);
}

void interactWithAcceptor(BitSet flags) throws Exception
{
    establishContext(flags);
    doWrap();
    doMIC();
}

void interactWithAcceptor() throws Exception
{
    BitSet flags = new BitSet();
    flags.set(Util.CONTEXT_OPTS_MUTUAL);
    flags.set(Util.CONTEXT_OPTS_CONF);
    flags.set(Util.CONTEXT_OPTS_INTEG);
    flags.set(Util.CONTEXT_OPTS_DELEG);
    interactWithAcceptor(flags);
}

void dispose() throws Exception
{
    if (tcp != null)
    {
        tcp.close();
    }
}

public static void main ( String args[] ) throws Exception
```

```
    {
        System.out.println(debug.toString()); // XXXXXXX
        String programName = "Client";
        Client client = null;
        try {
            client = new Client(programName,
                                false); // don't use Subject creds.
            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");
    }
}
```

鏈結集合

下載及執行範例程式
本主題包含有關下載及執行範例 javadoc 資訊的指示。

程式碼範例免責聲明

範例：IBM JGSS 非 JAAS 伺服器程式

如需使用範例伺服器程式的相關資訊，請參閱下載及執行 IBM JGSS 範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// IBM JGSS 1.0 Sample Server Program

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.*;

/**
  * A JGSS sample server; to be used in conjunction with a JGSS sample client.
 *
  * It continuously listens for client connections,
  * spawning a thread to service an incoming connection.
  * It is capable of running multiple threads concurrently.
  * In other words, it can service multiple clients concurrently.
 *
  * Each thread first establishes a context with the client
  * and then waits for a wrapped message followed by a MIC.
  * It assumes that the client calculated the MIC over the plain
  * text wrapped by the client.
 *
  * If the client delegates its credential to the server, the delegated
  * credential is used to communicate with a secondary server.
 *
  * Also, the server can be started to act as a client as well as
```

```
  * a server (using the -b option). In this case, the first
 * thread spawned by the server uses the server principal's own credential
  * to communicate with the secondary server.
 *
  * The secondary server must have been started prior to the (primary) server
  * initiating contact with it (the scondary server).
  * In communicating with the secondary server, the primary server acts as
  * a JGSS initiator (i.e., client), establishing a context and engaging in
  * wrap and MIC per-message exchanges with the secondary server.
 *
  * The server takes input parameters, and complements it
  * with information from the jgss.ini file; any required input not
  * supplied on the command line is taking from the jgss.ini file.
  * Built-in defaults are used if there is no jgss.ini file or if a particular
  * variable is not specified in the ini file.
 *
  * Usage: Server [options]
 *
  * The -? option produces a help message including supported options.
*
  * This sample server does not use JAAS.
  * It sets the JAVA variable
  * javax.security.auth.useSubjectCredsOnly to false
  * so that JGSS will not acquire credentials through JAAS.
  * The server can be run against the JAAS sample clients and servers.
  * See {@link JAASServer JAASServer} for a sample server that uses JAAS.
 */

class Server implements Runnable
{
    /*
          * NOTES:
          * This class, Server, is expected to be run in concurrent
          * multiple threads. The static variables consist of variables
     * set from command-line arguments and variables (such as
          * the server's own credentials, gssCred) that are set once during
          * during initialization. These variables do not change
     * once set and are shared between all running threads.
     *
          * The only static variable that is changed after being set initially
          * is the variable 'beenInitiator' which is set 'true'
          * by the first thread to run the server as initiator using
          * the server's own creds. This ensures the server is run as an initiator
     * once only. Querying and modifying 'beenInitiator' is synchronized
     * between the threads.
     *
          * The variable 'tcp' is non-static and is set per thread
          * to represent the socket on which the client being serviced
          * by the thread connected.
     */

    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[-?] [-# number] [-d | -n name] [-p port]"
   + "\n\t[-s serverName] [-h serverHost [:port]] [-P serverPort] [- msg]"
   + "\n"
   + "\n  -?\t\t\thelp; produces this message"
   + "\n  -# number\t\tWhether primary or secondary server"
   +         " \n\t\t\t(1 = primary, 2 = secondary; default = first)"
   + "\n  -n name\t\tthe server's principal name (without realm)"
   + "\n  -p port\t\tthe port on which the server will be listening"
   + "\n  -s serverName\t\tsecondary server's principal name"
   +         " (without realm)"
   + "\n  -h serverHost[:port]\tsecondary server's hostname"
   +         " (and optional port number)"
   + "\n  -P port\t\tsecondary server's port number"
   + "\n  -m msg\t\tmessage to send to secondary server"
   + "\n  -b \t\trun as both client and server"
   +         " using the server's owns credentials";

// Non-static variables are thread-specific
// since each thread runs a separate instance of this class.

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"));

        // Property not set. Set it to the specified value.
```

```
        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;

    // my port
    if (port > 0)
    {
        myPort = port;
    }
    else if (primary)
    {
        myPort = testUtil.getDefaultServicePort();
    }
    else
    {
        myPort = testUtil.getDefaultService2Port();
    }

    if (primary)
    {
        ///// peer's name
        if (serverNameWithoutRealm != null)
        {
            serviceNameNoRealm = serverNameWithoutRealm;
        }
        else
        {
            serviceNameNoRealm =
                  testUtil.getDefaultService2PrincipalWithoutRealm();
        }

        // peer's host
        if (serverHostname != null)
        {
            if (serverHostname.equalsIgnoreCase("localHost"))
            {
                serverHostname = InetAddress.getLocalHost().getHostName();
            }
```

```
            serviceHost = serverHostname;
        }
        else
        {
            serviceHost = testUtil.getDefaultService2Hostname();
        }

        // peer's port
        if (serverPort > 0)
        {
            servicePort = serverPort;
        }
        else
        {
            servicePort = testUtil.getDefaultService2Port();
        }

        // message for peer
        if (message != null)
        {
            serviceMsg = message;
        }
        else
        {
            serviceMsg = "Hi there! I am a server."
                         + "But I can be a client, too";
        }
    }

    String temp = debugPrefix + "details"
                  + "\n\tPrimary:\t" + primary
                  + "\n\tName:\t\t" + myName
                  + "\n\tPort:\t\t" + myPort
                  + "\n\tClient+server:\t" + clientServer;
    if (primary)
    {
        temp += "\n\tOther Server:"
                  + "\n\t\tName:\t" + serviceNameNoRealm
                  + "\n\t\tHost:\t" + serviceHost
                  + "\n\t\tPort:\t" + servicePort
                  + "\n\t\tMsg:\t" + serviceMsg;
    }

    debug.out(Debug.OPTS_CAT_APPLICATION, temp);
}


void initialize() throws GSSException
{
   debug.out(Debug.OPTS_CAT_APPLICATION,
                     debugPrefix + "creating GSSManager");

    mgr = GSSManager.getInstance();

    int usage = clientServer ? GSSCredential.INITIATE_AND_ACCEPT
                             : GSSCredential.ACCEPT_ONLY;

    if (myName != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                           + "creating GSSName for " + myName);

        gssName = mgr.createName(myName,
                                 GSSName.NT_HOSTBASED_SERVICE);

        Oid krb5MechanismOid = new Oid("1.2.840.113554.1.2.2");
        gssName.canonicalize(krb5MechanismOid);
```

```
        debug.out(Debug.OPTS_CAT_APPLICATION,
              debugPrefix + "Canonicalized GSSName=" + gssName);
    }
    else
        gssName = null;

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "creating"
           + ((gssName == null)? " default " : " ")
           + "credential");

    gssCred = mgr.createCredential(
                          gssName, GSSCredential.DEFAULT_LIFETIME,
                          (Oid)null, usage);
    if (gssName == null)
    {
        gssName = gssCred.getName();
        myName = gssName.toString();

        debug.out(Debug.OPTS_CAT_APPLICATION,
           debugPrefix + "default credential principal=" + myName);
    }
}



void processArgs(String[] args) throws Exception
{
    String port    = null;
    String name = null;
    int iport      = 0;

    String sport   = null;
    int isport     = 0;
    String sname   = null;
    String shost   = null;
    String smessage        = null;

    boolean primary = true;
    String status  = null;

    boolean defaultPrinc = false;
    boolean clientServer = false;

    GetOptions options = new GetOptions(args, "?#:p:n:P:s:h:m:b");
    int ch = -1;
    while ((ch = options.getopt()) != options.optEOF)
    {
        switch(ch)
        {
            case '?':
                printUsage();
                System.exit(1);

            case '#':
                if (status == null)
                    status = options.optArgGet();
                continue;

            case 'p':
                if (port == null)
                    port = options.optArgGet();
                continue;

            case 'n':
                if (name == null)
                    name = options.optArgGet();
```

```
            continue;

        case 'b':
            clientServer = true;
            continue;

        ////// The other server

        case 'P':
            if (sport == null)
                sport = options.optArgGet();
            continue;

        case 'm':
            if (smessage == null)
                smessage = options.optArgGet();
            continue;

        case 's':
            if (sname == null)
                sname = options.optArgGet();
            continue;

        case 'h':
            if (shost == null)
            {
                shost = options.optArgGet();
                int p = shost.indexOf(':');
                if (p != -1)
                {
                    String temp1 = shost.substring(0, p);
                    if (sport == null)
                        sport = shost.substring
                                    (p+1, shost.length()).trim();
                    shost = temp1;
                }
            }
            continue;
    }
}

if (defaultPrinc && (name != null))
{
    System.out.println(
      "ERROR: '-d' and '-n ' options are mutually exclusive");
    printUsage();
    System.exit(1);
}

if (status != null)
{
    int p = -1;
    try {
        p = Integer.parseInt(status);
    } catch (Exception exc) {
        System.out.println( "Bad status input: "+status);
    }

    if (p != -1)
    {
        primary = (p == 1);
    }
}

if (port != null)
{
    int p = -1;
```

```
        try {
            p = Integer.parseInt(port);
        } catch (Exception exc) {
            System.out.println( "Bad port input: "+port);
        }
        if (p != -1)
            iport = p;
    }

    if (sport != null)
    {
        int p = -1;
        try {
            p = Integer.parseInt(sport);
        } catch (Exception exc) {
            System.out.println( "Bad server port input: "+port);
        }
        if (p != -1)
            isport = p;
    }

    init(primary,  // first or second server
         name,     // my name
         iport,    // my port
         sname,    // other server's name
         shost,    // other server's hostname
         isport,   // other server's port
         smessage, // msg for other server
         clientServer); // whether to run as initiator with own creds
}

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); // set client socket per thread
            Thread thread = new Thread(server);
           thread.start();
            if (!thread.isAlive())
                server.dispose(); // close the client socket
       } while(true);
    } catch (Exception exc) {
       debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "*** ERROR processing requests ***");
        exc.printStackTrace();
    } finally {
        try {
            if (ssocket != null)
               ssocket.close(); // close the server socket
            if (server != null)
                server.dispose(); // close the client socket
        } catch (Exception exc) {}
    }
}

void dispose()
{
```

```
    try {
        if (tcp != null)
        {
            tcp.close();
            tcp = null;
        }
    } catch (Exception exc) {}
}

boolean establishContext(GSSContext context) throws Exception
{
    byte[] response        = null;
    byte[] request = null;

    debug.out(Debug.OPTS_CAT_APPLICATION,
                     debugPrefix + "establishing context");

    do {
        request = tcp.receive();
        if (request == null || request.length == 0)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "Received no data; perhaps client disconnected");

            return false;
        }

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "accepting");
        if ((response = context.acceptSecContext
                            (request, 0, request.length)) != null)
        {
             debug.out(Debug.OPTS_CAT_APPLICATION,
                       debugPrefix + "sending response");
             tcp.send(response);
        }
    } while(!context.isEstablished());

    debug.out(Debug.OPTS_CAT_APPLICATION,
                debugPrefix + "context established - " + context);

    return true;
}

byte[] unwrap(GSSContext context, byte[] msg) throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "unwrapping");

    MessageProp mp = new MessageProp(true);
    byte[] unwrappedMsg = context.unwrap(msg, 0, msg.length, mp);

    debug.out(Debug.OPTS_CAT_APPLICATION,
                         debugPrefix + "unwrapped msg is:");
    debug.out(Debug.OPTS_CAT_APPLICATION, unwrappedMsg);

    return unwrappedMsg;
}

void verifyMIC (GSSContext context, byte[] mic, byte[] raw) throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "verifying MIC");

    MessageProp mp = new MessageProp(true);
    context.verifyMIC(mic, 0, mic.length, raw, 0, raw.length, mp);

    debug.out(Debug.OPTS_CAT_APPLICATION,
                   debugPrefix + "successfully verified MIC");
}
```

```
void useDelegatedCred(GSSContext context) throws Exception
{
    GSSCredential delCred = context.getDelegCred();
    if (delCred != null)
    {
        if (primaryServer)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                    "Primary server received delegated cred; using it");
            runAsInitiator(delCred); // using delegated creds
        }
        else
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                   "Non-primary server received delegated cred; "
                        + "ignoring it");

        }
    }
    else
    {
         debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                                      "ERROR: null delegated cred");
    }
}

public void run()
{
    byte[] response               = null;
    byte[] request         = null;
    boolean unwrapped      = false;
    GSSContext context     = null;

    try {
      Thread currentThread = Thread.currentThread();
      String threadName    = currentThread.getName();

      debugPrefix = program + " " + threadName + ": ";

      debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                             + "servicing client ...");

     debug.out(Debug.OPTS_CAT_APPLICATION,
          debugPrefix + "creating GSSContext");

      context = mgr.createContext(gssCred);

      // First establish context with the initiator.
      if (!establishContext(context))
          return;

      // Then process messages from the initiator.
      // We expect to receive a wrapped message followed by a MIC.
      // The MIC should have been calculated over the plain
      // text that we received wrapped.
      // Use delegated creds if any.
      // Then run as initiator using own creds if necessary; only
      // the first thread does this.

      do {
         debug.out(Debug.OPTS_CAT_APPLICATION,
              debugPrefix + "receiving per-message request");

          request = tcp.receive();
          if (request == null || request.length == 0)
          {
```

```
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "Received no data; perhaps client disconnected");

            return;
          }

          // Expect wrapped message first.
          if (!unwrapped)
          {
              response = unwrap(context, request);
              unwrapped = true;
              continue; // get next request
          }

          // Followed by a MIC.
          verifyMIC(context, request, response);

          // Impersonate the initiator if it delegated its creds to us.
          if (context.getCredDelegState())
              useDelegatedCred(context);

          debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
              + "clientServer=" + clientServer
              + ", beenInitiator=" + beenInitiator);

          // If necessary, run as initiator using our own creds.
          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();

        // Squelch per-thread exceptions so we don't bring
        // the server down because of exceptions in
        // individual threads.
        return;
    } finally {
        if (context != null)
        {
            try {
                context.dispose();
            } catch (Exception exc) {}
        }
    }
}

synchronized void runAsInitiatorOnce(Thread thread)
    throws InterruptedException
{
    if (!beenInitiator)
    {
        // set flag true early to prevent subsequent threads
        // from attempting to runAsInitiator.
        beenInitiator = true;

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                "About to run as initiator with own creds ...");

        //thread.sleep(30*1000, 0);
        runAsInitiator();
    }
```

```
}


void runAsInitiator(GSSCredential cred)
{
    Client client = null;
    try {
        client = new Client(cred,
                            serviceNameNoRealm,
                            serviceHost,
                            servicePort,
                            serviceMsg);

        client.initialize();

        BitSet flags = new BitSet();
        flags.set(Util.CONTEXT_OPTS_MUTUAL);
        flags.set(Util.CONTEXT_OPTS_CONF);
        flags.set(Util.CONTEXT_OPTS_INTEG);

        client.interactWithAcceptor(flags);

    } catch (Exception exc) {
           debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
              + "Exception running as initiator");

        exc.printStackTrace();
    } finally {
        try {
            client.dispose();
        } catch (Exception exc) {}
    }
}

void runAsInitiator()
{
    if (clientServer)
    {
       debug.out(Debug.OPTS_CAT_APPLICATION,
           debugPrefix + "running as initiator with own creds");

        runAsInitiator(gssCred); // use own creds;
    }
    else
    {
       debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
           + "Cannot run as initiator with own creds "
           + "\nbecause not running as both initiator and acceptor.");
    }
}

void printUsage()
{
    System.out.println(program + usageString);
}

public static void main(String[] args) throws Exception
{
    System.out.println(debug.toString()); // XXXXXXX
    String programName = "Server";
    try {
        Server server = new Server(programName,
                                   false); // don't use creds from Subject
        server.processArgs(args);
        server.initialize();
        server.processRequests();
    } catch (Exception exc) {
```

```
            debug.out(Debug.OPTS_CAT_APPLICATION, programName + ": EXCEPTION");
            exc.printStackTrace();
            throw exc;
        }
    }
}
```

鏈結集合

下載及執行 IBM JGSS 範例

本主題包含有關下載及執行範例 javadoc 資訊的指示。

程式碼範例免責聲明

範例：IBM JGSS 支援 JAAS 的用戶端程式

如需使用範例用戶端程式的相關資訊，請參閱下載及執行 IBM JGSS 範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// IBM Java GSS 1.0 sample JAAS-enabled client program

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;

/**
  * A Java GSS sample client that uses JAAS.
 *
  * It does a JAAS login and operates within the JAAS login context so created.
 *
  * It does not set the JAVA variable
  * javax.security.auth.useSubjectCredsOnly, leaving
  * the variable to default to true
  * so that Java GSS acquires credentials from the JAAS Subject
  * associated with login context (created by the client).
 *
  * The JAASClient is equivalent to its superclass {@link Client Client}
  * in all other respects, and it
  * can be run against the non-JAAS sample clients and servers.
 */

class JAASClient extends Client
{
        JAASClient(String programName) throws Exception
    {
        // Do not set useSubjectCredsOnly. Set only the program name.
        // useSubjectCredsOnly default to "true" if not set.
        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);//use Subject creds
            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 ...");
    }
}
```

鏈結集合

下載及執行 IBM JGSS 範例

本主題包含有關下載及執行範例 javadoc 資訊的指示。

程式碼範例免責聲明

範例：IBM JGSS 支援 JAAS 的伺服器程式

如需使用伺服器程式範例的相關資訊，請參閱下載及執行 IBM JGSS 範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
// IBM Java GSS 1.0 sample JAAS-enabled server program

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;

/**
  * A Java GSS sample server that uses JAAS.
 *
  * It does a JAAS login and operates within the JAAS login context so created.
 *
  * It does not set the JAVA variable
  * javax.security.auth.useSubjectCredsOnly, leaving
  * the variable to default to true
  * so that Java GSS acquires credentials from the JAAS Subject
  * associated with login context (created by the server).
 *
  * The JAASServer is equivalent to its superclass {@link Server Server}
  * in all other respects, and it
  * can be run against the non-JAAS sample clients and servers.
 */

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 {
            // Do not set useSubjectCredsOnly.
            // useSubjectCredsOnly defaults to "true" if not set.

            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;
        }
    }
}
```

鏈結集合

下載及執行 IBM JGSS 範例

本主題包含有關下載及執行範例 javadoc 資訊的指示。

程式碼範例免責聲明

範例：IBM Java Secure Sockets Extension

以下 JSSE 範例顯示用戶端及伺服器如何使用原有的 iSeries JSSE 提供者來建立可進行安全通訊的環境定義。

註: 無論 java.security 檔案指定的內容爲何，這兩個範例一律使用原有的 iSeries JSSE 提供者。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

第 261 頁的『範例：使用 SSLContext 物件的 SSL 用戶端』

此用戶端程式範例會起始設定一個 SSLContext 物件，並透過它來使用 ″MY_CLIENT_APP″ 應用程式 ID。 無論 java.security 檔案中指定的內容爲何，此程式一律使用原有的 iSeries 實作方式。

第 263 頁的『範例：使用 SSLContext 物件的 SSL 伺服器』

下列伺服器程式將使用先前建立的金鑰庫檔案所起始設定的 SSLContext 物件。 金鑰庫檔案的名稱爲 /home/keystore.file，金鑰庫密碼爲 password。

範例程式需要金鑰庫檔案來建立 IbmISeriesKeyStore 物件。 KeyStore 物件必須指定 MY_SERVER_APP 作爲應用程式 ID。

您可以使用下列指令來建立金鑰庫檔案：

- 從 Qshell 指令提示：

  ```
  java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
     -storepass password -appid MY_SERVER_APP
  ```

  如需在 Qshell 中使用 Java 指令的相關資訊，請參閱「iSeries 資訊中心」的 Qshell。

- 從 iSeries 指令提示：

  ```
  RUNJVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
            '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
  ```

範例：使用 java.lang.Runtime.exec() 來呼叫 CL 程式

本範例顯示如何從 Java 程式內執行 CL 程式。在此範例中，Java 類別 CallCLPgm 會執行 CL 程式。

CL 程式使用「顯示 Java 程式 (DSPJVAPGM)」指令，顯示與 Hello 類別檔案相關的程式。本範例假設 CL 程式已完成編譯，存在於 JAVSAMPLIB 檔案庫中。CL 程式的輸出位於 QSYSPRT 儲存檔。

如需如何從 Java 程式內呼叫 CL 指令的範例，請參閱呼叫 CL 指令。

註: JAVSAMPLIB 並非由 IBM Developer Kit 授權程式 (LP) 編號 5722-JV1 的安裝程序所建立。您必須明確地建立此檔案庫。

範例 1： CallCLPgm 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallCLPgm
{
   public static void main(String[] args)
   {
       try
      {
                   Process theProcess =
                  Runtime.getRuntime().exec("/QSYS.LIB/JAVSAMPLIB.LIB/DSPJVA.PGM");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
         e.printStackTrace();
      }
   } // end main() method
} // end class
```

範例 2：顯示 Java CL 程式

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
                OUTPUT(*PRINT)
ENDPGM
```

如需相關背景資訊，請參閱使用 java.lang.Runtime.exec()。

範例：使用 java.lang.Runtime.exec() 來呼叫 CL 指令

本範例顯示如何從 Java 程式內執行控制語言 (CL) 指令。

在此範例中，Java 類別會執行 CL 指令。CL 指令使用「顯示 Java 程式 (DSPJVAPGM)」CL 指令，顯示與 Hello 類別檔案相關的程式。CL 指令的輸出位於 QSYSPRT 儲存檔。

當您將 os400.runtime.exec 系統內容設定爲 EXEC (預設值) 時，您傳入 `Runtime.getRuntime().exec()` 函數的指令將採用下列格式：

```
Runtime.getRuntime()Exec("system CLCOMMAND");
```

其中 *CLCOMMAND* 是您要執行的 CL 指令。

註: 當您將 os400.runtime.exec 設定爲 QSHELL 時，必須加上斜線及引號 (\")。例如，先前的指令會變成：

```
Runtime.getRuntime()Exec("system \"CLCOMMAND\"");
```

有關 os400.runtime.exec 的詳細資訊及其對於 java.lang.Runtime.exec() 用法的影響，請參閱以下各頁：

使用 java.lang.Runtime.exec()

Java 系統內容的清單

範例：適用於呼叫 CL 指令的類別

下列程式碼假設您使用 os400.runtime.exec 系統內容的預設值 EXEC。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallCLCom
{
   public static void main(String[] args)
   {
      try
      {
         Process theProcess =
            Runtime.getRuntime().exec("system DSPJVAPGM CLSF('/com/ibm/as400/system/Hello.class')
                                      OUTPUT(*PRINT)");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
         e.printStackTrace();
      }
   } // end main() method
} // end class
```

如需相關背景資訊，請參閱使用 java.lang.Runtime.exec()。

範例：使用 java.lang.Runtime.exec() 來呼叫另一個 Java 程式

下列範例說明如何使用 java.lang.Runtime.exec() 呼叫另一個 Java 程式。此類別會呼叫 IBM Developer Kit for Java 隨附的 Hello 程式。當 Hello 類別寫入 System.out 時，此程式就會取得串流的控點，然後從中讀取資料。

註: 您可以使用「Qshell 直譯器」來呼叫程式。

範例 1：CallHelloPgm 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallHelloPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallHelloPgm.main() invoked");

      // call the Hello class
      try
      {
          theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
         e.printStackTrace();
      }

      // read from the called program's standard output stream
      try
      {
```

```
         inStream = new BufferedReader(
                                new InputStreamReader( theProcess.getInputStream() ));
         System.out.println(inStream.readLine());
      }
      catch (IOException e) {
System.err.println("Error on inStream.readLine()");
         e.printStackTrace();
      }

   } // end method

 } // end class
```

如需相關背景資訊，請參閱使用 java.lang.Runtime.exec()。

範例：從 C 呼叫 Java

此爲從 C 程式中使用 system() 函數呼叫 Java Hello 程式的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
#include <stdlib.h>

 int main(void)
 {
   int result;

 /* The system function passes the given string to the CL command processor
    for processing. */

   result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
 }
```

範例：從 RPG 呼叫 Java

此爲從 RPG 程式中使用 QCMDEXC API 呼叫 Java Hello 程式的範例。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
     D*             DEFINE  THE PARAMETERS FOR THE QCMDEXC API
     D*
     DCMDSTRING        S             25    INZ('JAVA CLASS(''com.ibm.as400.system.Hello'')')
     DCMDLENGTH        S             15P 5 INZ(25)
     D*             NOW THE CALL TO QCMDEXC WITH THE 'JAVA' CL COMMAND
     C                   CALL      'QCMDEXC'
     C                   PARM                    CMDSTRING
     C                   PARM                    CMDLENGTH
     C*       This next line displays 'DID IT' after you exit the
     C*       Java Shell via F3 or F12.
     C     'DID IT'      DSPLY
     C*       Set On LR to exit the RPG program
     C                   SETON                                        LR
     C
```

範例：使用輸入及輸出串流來進行處理作業之間的通信

下列範例顯示如何從 Java 呼叫 C 程式，以及如何使用輸入及輸出串流來進行處理作業之間的通訊。

在此範例中，C 程式會在標準輸出串流上寫入字串，然後由 Java 程式讀取並顯示此字串。此範例假設已建立檔案庫 JAVSAMPLIB，且其中也已建立 CSAMP1 程式。

註: JAVSAMPLIB 並非由 IBM Developer Kit 授權程式 (LP) 編號 5722-JV1 的安裝程序所建立。您必須明確地加以建立。

範例 1：CallPgm 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.io.*;

public class CallPgm
{
   public static void main(String args[])
   {
      Process theProcess = null;
      BufferedReader inStream = null;

      System.out.println("CallPgm.main() invoked");

      // call the CSAMP1 program
      try
      {
          theProcess = Runtime.getRuntime().exec(
                       "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
      }
      catch (IOException e) {
System.err.println("Error on exec() method");
         e.printStackTrace();
      }

      // read from the called program's standard output stream
      try
      {
         inStream = new BufferedReader(new InputStreamReader
                           (theProcess.getInputStream()));
         System.out.println(inStream.readLine());
      }
      catch(IOException e)
      {
         System.err.println("Error on inStream.readLine()");
         e.printStackTrace();
      }

   } // end method

 } // end class
```

範例 2：CSAMP1 C 程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
     /* Convert the string to ASCII at compile time */
#pragma convert(819)
     printf("Program JAVSAMPLIB/CSAMP1 was invoked\n");
#pragma convert(0)
     /* Stdout may be buffered, so flush the buffer */

     fflush(stdout);

}
```

如需相關資訊，請參閱使用輸入及輸出串流來進行處理作業之間的通信。

範例：Java 呼叫 API

此範例遵循標準的「呼叫 API」參照範例。

它執行下列動作：

- 使用 JNI_CreateJavaVM 建立 Java 虛擬機器。
- 使用 Java 虛擬機器尋找您要執行的類別檔案。
- 尋找類別中 main 方法的 methodID。
- 呼叫類別的 main 方法。
- 報告異常發生時的錯誤。

建立程式時，QJVAJNI 或 QJVAJNI64 服務程式會提供 JNI_CreateJavaVM「呼叫 API」函數。JNI_CreateJavaVM 會建立 Java 虛擬機器。

註: QJVAJNI64 是兆空間/LLP64 原生方法及「呼叫 API」支援新增的服務程式。

這些服務程式就在系統連結目錄中，您並不需要在控制語言 (CL) 建立指令上明確地指出它們。例如，使用「建立程式 (CRTPGM)」指令或「建立服務程式 (CRTSRVPGM)」指令時，您不需要明確指出前述的服務程式。

使用下列控制語言指令是執行此程式的方法之一：

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```

建立 Java 虛擬機器的任何工作，皆必須具備多緒能力。主要程式的輸出，以及程式的任何輸出，最後都會進入 QPRINT 排存檔。利用「使用提出的工作 (WRKSBMJOB)」控制語言 (CL) 指令，檢視您先前以「提出工作 (SBMJOB)」CL 指令所啓動的工作，就可以看到這些排存檔。

範例：使用 Java 呼叫 API

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
#define OS400_JVM_12
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

/* Specify the pragma that causes all literal strings in the
  * source code to be stored in ASCII (which, for the strings
  * used, is equivalent to UTF-8)
 */

    #pragma convert(819)

/* Procedure: Oops
 *
 * Description: Helper routine that is called when a JNI function
 *              returns a zero value, indicating a serious error.
 *              This routine reports the exception to stderr and
 *              ends the JVM abruptly with a call to FatalError.
 *
 * Parameters:  env -- JNIEnv* to use for JNI calls
 *              msg -- char* pointing to error description in UTF-8
 *
 * Note:        Control does not return after the call to FatalError
 *              and it does not return from this procedure.
 */

void Oops(JNIEnv* env, char *msg) {
```

```
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*env)->FatalError(env, msg);
}

/* This is the program's "main" routine. */
int main (int argc, char *argv[])
{

    JavaVMInitArgs initArgs; /* Virtual Machine (VM) initialization structure, passed by
                              * reference to JNI_CreateJavaVM(). See jni.h for details
                              */
    JavaVM* myJVM;           /* JavaVM pointer set by call to JNI_CreateJavaVM */
    JNIEnv* myEnv;           /* JNIEnv pointer set by call to JNI_CreateJavaVM */
    char*   myClasspath;     /* Changeable classpath 'string' */
    jclass  myClass;         /* The class to call, 'NativeHello'. */
    jmethodID mainID;        /* The method ID of its 'main' routine. */
    jclass  stringClass;     /* Needed to create the String[] arg for main */
    jobjectArray args;       /* The String[] itself */
    JavaVMOption options[1]; /* Options array -- use options to set classpath */
    int     fd0, fd1, fd2;   /* file descriptors for IO */

    /* Open the file descriptors so that IO works. */
    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);

    /*  Set the version field of the initialization arguments for J2SDK v1.3. */
    initArgs.version = 0x00010002;
 /*  To use J2SDK v1.4, set initArgs.version = 0x00010004; */
  /*  To use J2SDK v1.5, set initArgs.version = 0x00010005; */


    /* Now, you want to specify the directory for the class to run in the classpath.
     * with  Java2, classpath is passed in as an option.
     * Note: You must specify the directory name in UTF-8 format. So, you wrap
     *       blocks of code in #pragma convert statements.
     */
    options[0].optionString="-Djava.class.path=/CrtJvmExample";
 /*To use J2SDK v1.4 or v1.5, replace the '1.3' with '1.4' or '1.5'.
 options[1].optionString="-Djava.version=1.3" */

    initArgs.options=options;  /* Pass in the classpath that has been set up. */
    initArgs.nOptions = 2;     /* Pass in classpath and version options */

    /*  Create the JVM -- a nonzero return code indicates there was
     *  an error. Drop back into EBCDIC and write a message to stderr
     *  before exiting the program.
     */
    if (JNI_CreateJavaVM("myJVM, (void **)"myEnv, (void *)"initArgs)) {
  #pragma convert(0)
         fprintf(stderr, "Failed to create the JVM\n");
  #pragma convert(819)
         exit(1);
    }

    /*  Use the newly created JVM to find the example class,
     *  called 'NativeHello'.
     */
    myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
    if (! myClass) {
        Oops(myEnv, "Failed to find class 'NativeHello'");
    }

    /* Now, get the method identifier for the 'main' entry point
     * of the class.
```

```
     * Note: The signature of 'main' is always the same for any
     *       class called by the following java command:
     *           "main" , "([Ljava/lang/String;)V"
     */
    mainID = (*myEnv)->GetStaticMethodID(myEnv,myClass,"main",
                                         "([Ljava/lang/String;)V");
    if (! mainID) {
        Oops(myEnv, "Failed to find jmethodID of 'main'");
    }

    /* Get the jclass for String to create the array
     * of String to pass to 'main'.
     */
    stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
    if (! stringClass) {
        Oops(myEnv, "Failed to find java/lang/String");
    }

    /*  Now, you need to create an empty array of strings,
     *  since main requires such an array as a parameter.
     */
    args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
    if (! args) {
        Oops(myEnv, "Failed to create args array");
    }

    /* Now, you have the methodID of main and the class, so you can
     * call the main method.
     */
    (*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);

    /* Check for errors. */
    if ((*myEnv)->ExceptionOccurred(myEnv)) {
        (*myEnv)->ExceptionDescribe(myEnv);
    }

    /* Finally, destroy the JavaVM that you created. */
    (*myJVM)->DestroyJavaVM(myJVM);

    /* All done. */
    return 0;
}
```

如需相關資訊，請參閱 Java 呼叫 API。

範例：IBM i5/OS PASE 適用於 Java 的原生方法

IBM i5/OS PASE 適用於 Java 的原生方法範例會呼叫原有的 C 方法實例，此實例再利用「Java 原生介面 (JNI)」來回呼 Java 程式碼。此範例並不直接從此 Java 程式碼中存取字串，而是呼叫原生方法，原生方法再透過 JNI 回呼 Java 來取得字串值。

如需 HTML 版的來源檔範例，請使用下列鏈結：

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

- PaseExample1.java
- PaseExample1.c

執行 i5/OS PASE 原生方法範例之前，必須先完成下列作業：

1. 將範例原始程式碼下載至 AIX 工作站
2. 準備範例原始程式碼
3. 準備 iSeries 伺服器

執行 i5/OS PASE 適用於 Java 的原生方法範例

完成上述作業之後，即可開始執行範例。請使用下列指令來執行範例程式：

- 從 iSeries 伺服器指令提示：

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- 從 Qshell 指令提示或 i5/OS PASE 終端機階段作業：

```
cd /home/example
     java PaseExample1
```

範例：PaseExample1.java

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
////////////////////////////////////////////////////////////////////////////
//
// This example program loads the native method library 'PaseExample1'.
// The source code for the native method is contained in PaseExample1.c
// The printString method in this Java program uses a native method,
// getStringNative to retrieve the value of the String. The native method
// simply calls back into the getStringCallback method of this class.
//
////////////////////////////////////////////////////////////////////////////



public class PaseExample1 {
    public static void main(String args[]) {
 PaseExample1 pe1 = new PaseExample1("String for PaseExample1");
 pe1.printString();
    }

    String str;

    PaseExample1(String s) {
 str = s;
    }


    //--------------------------------------------------------------
    public void printString() {
 String result = getStringNative();
 System.out.println("Value of str is '" + result + "'");
    }

    // This calls getStringCallback through JNI.
    public native String getStringNative();

    // Called by getStringNative via JNI.
    public String getStringCallback() {
 return str;
    }


    //--------------------------------------------------------------
    static {
 System.loadLibrary("PaseExample1");
    }

}
```

鏈結集合

程式碼範例免責聲明

範例：PaseExample1.c

註：請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/*
 *

  * This native method implements the getStringNative method of class
  * PaseExample1. It uses the JNI function CallObjectMethod to call
 * back to the getStringCallback method of class PaseExample1.
 *
  * Compile this code in AIX to create module 'libPaseExample1.so'.
 *
 */

#include "PaseExample1.h"
#include <stdlib.h>

/*
  * Class:     PaseExample1
  * Method:    getStringNative
  * Signature: ()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);
}
```

鏈結集合

程式碼範例免責聲明

範例：將範例原始程式碼下載至 AIX 工作站

執行 IBM i5/OS PASE 適用於 Java 的原生方法範例之前，必須先下載含有原始程式碼的壓縮檔。若要將壓縮檔下載至 AIX 工作站，請完成下列步驟。

1. 在 AIX 工作站建立要用來存放來源檔的暫時目錄。
2. 下載 i5/OS PASE 範例原始程式碼至暫時目錄。
3. 將範例檔解壓縮至暫時目錄。

如需 IBM i5/OS PASE 適用於 Java 之原生方法範例的相關資訊，請參閱下列主題：

- 範例：IBM i5/OS PASE 適用於 Java 的原生方法
- 範例：準備範例原始程式碼
- 範例：準備 iSeries 伺服器

鏈結集合

下載 i5/OS PASE 範例原始程式碼

範例：IBM i5/OS PASE 適用於 Java 的原生方法
IBM i5/OS PASE 適用於 Java 的原生方法範例會呼叫原有的 C 方法實例，此實例再利用「Java 原生介面 (JNI)」來回呼 Java 程式碼。此範例並不直接從此 Java 程式碼中存取字串，而是呼叫原生方法，原生方法再透過 JNI 回呼 Java 來取得字串值。

範例：準備範例原始程式碼

範例：準備 iSeries 伺服器

範例：準備範例原始程式碼

將 IBM i5/OS PASE 適用於 Java 的原生方法範例移至 iSeries 伺服器之前，必須先編譯原始程式碼、建立 C 併入檔，並建立共用檔案庫物件。

範例包含下列 C 來源檔及 Java 來源檔：

- **PaseExample1.c**： C 原始程式檔，內含 getStringNative() 的實作方式。
- **PaseExample1.java**：Java 原始程式檔，呼叫 C 程式中原有的 getStringNative 方法。

您必須使用編譯後的 Java .class 檔案來建立 C 併入檔 PaseExample1.h，此併入檔包含 C 原始程式碼中 getStringNative 方法的函數原型。

若要在 AIX 工作站上準備範例原始程式碼，請完成下列步驟：

1. 使用下列指令來編譯 Java 原始程式碼：

   ```
   javac PaseExample1.java
   ```

2. 使用下列指令來建立含原生方法原型的 C 併入檔：

   ```
   javah -jni PaseExample
   ```

   新的 C 併入檔 (PaseExample1.h) 含有 getStringNative 方法的函數原型。 C 原始程式碼範例 (PaseExample1.c) 之中，即已包含了您必須針對 C 併入檔加以複製及修改的資訊，可用來使用 getStringNative 方法。 如需使用 JNI 的相關資訊，請參閱 Sun 網站的 Java Native Interface 指導教學。

3. 使用下列指令來編譯 C 原始程式碼及建立共用檔案庫物件。

   ```
   xlc -G -I/usr/local/java/J1.3.0/include PaseExample1.c -o libPaseExample1.so
   ```

   新的共用檔案庫物件檔 (libPaseExample1.so) 包含範例所用的原生方法檔案庫 PaseExample1。

   附註：可能需要變更 -I 選項，以指向您的 AIX 系統上 Java 原生方法併入檔 (例如 jni.h) 所在的目錄。

如需 IBM i5/OS PASE 適用於 Java 之原生方法範例的相關資訊，請參閱下列主題：

- 範例：IBM i5/OS PASE 適用於 Java 的原生方法
- 範例：將範例原始程式碼下載至 AIX 工作站
- 範例：準備 iSeries 伺服器

鏈結集合

Java Native Interface 指導教學

範例：IBM i5/OS PASE 適用於 Java 的原生方法
IBM i5/OS PASE 適用於 Java 的原生方法範例會呼叫原有的 C 方法實例，此實例再利用「Java 原生介面 (JNI)」來回呼 Java 程式碼。此範例並不直接從此 Java 程式碼中存取字串，而是呼叫原生方法，原生方法再透過 JNI 回呼 Java 來取得字串值。

範例：將範例原始程式碼下載至 AIX 工作站

範例：準備 iSeries 伺服器

範例：準備 iSeries 伺服器

執行 IBM i5/OS PASE 適用於 Java 的原生方法範例之前，iSeries 伺服器必須先做好執行範例的準備。準備伺服器時，必須將檔案複製到伺服器，並且加入執行範例所需的環境變數。

請完成下列步驟來準備伺服器：

1. 在伺服器上建立您要存放範例檔的整合檔案系統目錄。例如，使用下列控制語言 (CL) 指令來建立名爲 /home/example 的目錄：

   ```
   mkdir /home/example
   ```

2. 將下列檔案複製到新的目錄中：
   - PaseExample1.class
   - libPaseExample1.so
3. 在 iSeries 指令提示上，使用下列控制語言 (CL) 指令來新增必要的環境變數：

   ```
   addenvvar PASE_THREAD_ATTACH 'Y'
        addenvvar PASE_LIBPATH '/home/example'
        addenvvar QIBM_JAVA_PASE_STARTUP '/usr/lib/start32'
   ```

附註：從 i5/OS PASE 終端機階段作業使用 PASE 原生方法時，便已啓動 32 位元 PASE 環境。以此例而言，只需要將 PASE_THREAD_ATTACH 設爲 Y，將 PASE_LIBPATH 設爲 PASE 原生方法檔案庫的路徑。在此情況下，即使定義 QIBM_JAVA_PASE_STARTUP，也不會順利啓動 JVM。

有關新增環境變數的資訊，請參閱下列主題：

- IBM i5/OS PASE 範例的環境變數

如需 IBM i5/OS PASE 適用於 Java 之原生方法範例的相關資訊，請參閱下列主題：

- 範例：IBM i5/OS PASE 適用於 Java 的原生方法
- 範例：將範例原始程式碼下載至 AIX 工作站
- 範例：準備範例原始程式碼

鏈結集合

IBM i5/OS PASE 範例的環境變數
若要使用 IBM i5/OS PASE 適用於 Java 的原生方法範例，必須設定環境變數。

範例：IBM i5/OS PASE 適用於 Java 的原生方法
IBM i5/OS PASE 適用於 Java 的原生方法範例會呼叫原有的 C 方法實例，此實例再利用「Java 原生介面 (JNI)」來回呼 Java 程式碼。此範例並不直接從此 Java 程式碼中存取字串，而是呼叫原生方法，原生方法再透過 JNI 回呼 Java 來取得字串值。

範例：將範例原始程式碼下載至 AIX 工作站

範例：準備範例原始程式碼

範例：針對原生方法使用 Java 原生介面

本範例程式是簡單的「Java 原生介面 (JNI)」範例，其中會用到 C 原生方法來顯示 ″Hello, World″。請使用 javah 工具及 NativeHello 類別檔案來產生 NativeHello.h 檔案。本範例假設 NativeHello C 實作方式是 NATHELLO 這個服務程式的一部份。

註: NATHELLO 服務程式所在的檔案庫，必須在檔案庫清單中，才能夠執行此範例。

範例 1：NativeHello 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
public class NativeHello {

    //  Declare a field of type 'String' in the NativeHello object.
    //  This is an 'instance' field, so every NativeHello object
    //  contains one.
    public String theString;           // instance variable
```

```
    //  Declare the native method itself.  This native method
    //  creates a new string object, and places a reference to it
    //  into 'theString'
    public native void setTheString(); // native method to set string

    //  This 'static initializer' code is called before the class is
    //  first used.
    static {

        //  Attempt to load the native method library.  If you do not
        //  find it, write a message to 'out', and try a hardcoded path.
        //  If that fails, then exit.
        try {

            //  System.loadLibrary uses the iSeries library list in JDK 1.1,
            //  and uses the java.library.path property or the LIBPATH environment
            //  variable in JDK1.2
              System.loadLibrary("NATHELLO");
            }

        catch (UnsatisfiedLinkError e1) {

            //  Did not find the service program.
            System.out.println
              ("I did not find NATHELLO *SRVPGM.");
            System.out.println ("(I will try a hardcoded path)");

            try {

                //  System.load takes the full integrated file system form path.
                System.load ("/qsys.lib/jniexample.lib/nathello.srvpgm");
                }

        catch (UnsatisfiedLinkError e2) {

            //  If you get to this point, then you are done!  Write the message
            //  and exit.
            System.out.println
              ("<sigh> I did not find NATHELLO *SRVPGM anywhere. Goodbye");
            System.exit(1);
            }
        }
    }

    //  Here is the 'main' code of this class. This is what runs when you
    //  enter 'java NativeHello' on the command line.
    public static void main(String argv[]){

        //  Allocate a new NativeHello object now.
        NativeHello nh = new NativeHello();

        //  Echo location.
        System.out.println("(Java) Instantiated NativeHello object");
        System.out.println("(Java) string field is '" + nh.theString + "'");
        System.out.println("(Java) Calling native method to set the string");

        //  Here is the call to the native method.
        nh.setTheString();

        //  Now, print the value after the call to double check.
        System.out.println("(Java) Returned from the native method");
        System.out.println("(Java) string field is '" + nh.theString + "'");
        System.out.println("(Java) All done...");
    }
}
```

範例 2：產生的 NativeHello.h 標頭檔

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class NativeHello */

#ifndef _Included_NativeHello
#define _Included_NativeHello
#ifdef __cplusplus
extern "C" {
#endif
/*
  * Class:     NativeHello
  * Method:    setTheString
  * Signature: ()V
 */
JNIEXPORT void JNICALL Java_NativeHello_setTheString
    (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

此 NativeHello.c 範例顯示原生方法在 C 裡的實作方式。此範例說明如何將 Java 鏈結至原生方法。然而，由於本質上 iSeries 伺服器是一種延伸的二進位編碼十進位交換碼 (EBCDIC) 機器，因此導致情況更加複雜。同時也顯示出由於目前 JNI 缺乏眞正的國際化要素而浮現出的複雜性。

這些原因不是最近才發生在 JNI 身上，但卻在您編寫的 C 程式碼中，呈現一些 iSeries 伺服器獨一無二的差異性。切記，若您寫入 stdout 或 stderr，或是從 stdin 讀取，您的資料可能是以 EBCDIC 格式來編碼。

在 C 程式碼中，可以輕易將大部份文字字串 (僅含 7 位元字元的字串) 轉換成 JNI 所需的 UTF-8 格式。作法是用字碼頁轉換 pragma 將文字字串括住。然而，因爲您可能從 C 程式碼中直接將資訊寫入 stdout 或 stderr，所以可能允許部份文字繼續維持 EBCDIC。

註: #pragma convert(0) 陳述式會將字元資料轉換成 EBCDIC。#pragma convert(819) 陳述式會將字元資料轉換成「美國國家標準交換碼 (ASCII)」。這些陳述式會在編譯時轉換 C 程式內的字元資料。

範例 3：NativeHello Java 類別的 NativeHello.c 原生方法實作方式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
#include <stdlib.h>       /* malloc, free, and so forth */
#include <stdio.h>        /* fprintf(), and so forth */
#include <qtqiconv.H>     /* iconv() interface */
#include <string.h>       /* memset(), and so forth */
#include "NativeHello.h"  /* generated by 'javah-jni' */

/*  All literal strings are ISO-8859-1 Latin 1 code page (and with 7-bit
characters, they are also automatically UTF-8). */
#pragma convert(819)  /* handle all literal strings as ASCII */

/*  Report and clear a JNI exception.  */
static void HandleError(JNIEnv*);

/*  Print an UTF-8 string to stderr in the coded character */
set identifier (CCSID) of the current job.  */
static void JobPrint(JNIEnv*, char*);

/*  Constants describing which direction to covert:  */
#define CONV_UTF2JOB 1
#define CONV_JOB2UTF 2
```

```
/*  Convert a string from the CCSID of the job to UTF-8, or vice-versa.  */
int StringConvert(int direction, char *sourceStr, char *targetStr);

/*  Native method implementation of 'setTheString()'.  */
JNIEXPORT void JNICALL Java_NativeHello_setTheString
(JNIEnv *env, jobject javaThis)
{
    jclass thisClass;     /* class for 'this' object */
    jstring stringObject; /* new string, to be put in field in 'this' */
    jfieldID fid;         /* field ID required to update field in 'this' */
    jthrowable exception; /* exception, retrieved using ExceptionOccurred */

    /*  Write status to console.  */
    JobPrint(env, "( C ) In the native method\n");

    /* Build the new string object.  */
    if (! (stringObject = (*env)->NewStringUTF(env, "Hello, native world!")))
    {
        /*  For nearly every function in the JNI, a null return value indicates
        that there was an error, and that an exception had been placed where it
        could be retrieved by 'ExceptionOccurred()'. In this case, the error
        would typically be fatal, but for purposes of this example, go ahead
        and catch the error, and continue.  */
        HandleError(env);
        return;
    }

    /* get the class of the 'this' object, required to get the fieldID  */
    if (! (thisClass = (*env)->GetObjectClass(env,javaThis)))
    {
        /*  A null class returned from GetObjectClass indicates that there
        was a problem. Instead of handling this problem, simply return and
        know that the return to Java automatically 'throws' the stored Java
        exception.  */
        return;
    }

    /*  Get the fieldID to update.  */
    if (! (fid = (*env)->GetFieldID(env,
                                    thisClass,
                                    "theString",
                                    "Ljava/lang/String;")))
    {
        /*  A null fieldID returned from GetFieldID indicates that there
        was a problem.  Report the problem from here and clear it.
        Leave the string unchanged.  */
        HandleError(env);
        return;
    }

    JobPrint(env, "( C ) Setting the field\n");

    /*  Make the actual update.
    Note: SetObjectField is an example of an interface that does
    not return a return value that can be tested.  In this case, it
    is necessary to call ExceptionOccurred() to see if there
    was a problem with storing the value  */
    (*env)->SetObjectField(env, javaThis, fid, stringObject);

    /*  Check to see if the update was successful. If not, report the error.  */
    if ((*env)->ExceptionOccurred(env)) {

        /*  A non-null exception object came back from ExceptionOccurred,
        so there is a problem and you must report the error. */
        HandleError(env);
    }
```

```
    JobPrint(env, "( C ) Returning from the native method\n");
    return;

}

static void HandleError(JNIEnv *env)
{
    /*  A simple routine to report and handle an exception.  */
    JobPrint(env, "( C  ) Error occurred on JNI call: ");
    (*env)->ExceptionDescribe(env); /* write exception data to the console */
    (*env)->ExceptionClear(env);    /* clear the exception that was pending */
}

static void JobPrint(JNIEnv *env, char *str)
{
    char   *jobStr;
    char   buf[512];
    size_t len;

    len = strlen(str);

    /*  Only print non-empty string. */
    if (len) {
        jobStr = (len >= 512) ? malloc(len+1) : &buf;
        if (! StringConvert(CONV_UTF2JOB, str, jobStr))
            (*env)->FatalError
              (env,"ERROR in JobPrint: Unable to convert UTF2JOB");
        fprintf(stderr, jobStr);
        if (len >= 512) free(jobStr);
    }
}

int StringConvert(int direction, char *sourceStr, char *targetStr)
{
    QtqCode_T source, target;     /* parameters to instantiate iconv  */
    size_t    sStrLen, tStrLen;   /* local copies of string lengths   */
    iconv_t   ourConverter;       /* the actual conversion descriptor */
    int       iconvRC;            /* return code from the conversion  */
    size_t    originalLen;        /* original length of the sourceStr */

    /*  Make local copies of the input and output sizes that are initialized
    to the size of the input string. The iconv() requires the
    length parameters to be passed by address (that is as int*).          */
    originalLen = sStrLen = tStrLen = strlen(sourceStr);

    /*  Initialize the parameters to the QtqIconvOpen() to zero.  */
    memset(&source,0x00,sizeof(source));
    memset(&target,0x00,sizeof(target));

    /*  Depending on direction parameter, set either SOURCE
    or TARGET CCSID to ISO 8859-1 Latin.  */
    if (CONV_UTF2JOB == direction ) {
        source.CCSID = 819;
    }
    else {
        target.CCSID = 819;
    }

    /*  Create the iconv_t converter object.  */
    ourConverter = QtqIconvOpen(&target,&source);

    /*  Make sure that you have a valid converter, otherwise return 0.  */
    if (-1 == ourConverter.return_value) return 0;

    /*  Perform the conversion.  */
    iconvRC = iconv(ourConverter,
                    (char**) &sourceStr,
```

```
                    &sStrLen,
                    &targetStr,
                    &tStrLen);

    /*  If the conversion failed, return a zero.  */
    if (0 != iconvRC ) return 0;

    /*  Close the conversion descriptor.  */
    iconv_close(ourConverter);

    /*  The targetStr returns pointing to the character just
    past the last converted character, so set the null
    there now.  */
    *targetStr = '\0';

    /*  Return the number of characters that were processed. */
    return originalLen-tStrLen;

}

#pragma convert(0)
```

如需背景資訊，請參閱針對原生方法使用 Java 原生介面。

鏈結集合

程式碼範例免責聲明

程式碼範例免責聲明

文字字串
字串由 7 位元「美國國家標準交換碼 (ASCII)」表示法的字元組成時， 較易於將文字字串編碼成 UTF-8。

程式碼範例免責聲明

對原生方法使用 Java 原生介面 (JNI)
只有在 Pure Java 無法滿足您的程式設計需求時，才應使用原生方法。

範例：使用 Socket 來進行處理作業之間的通訊

此範例使用 Socket 在 Java 程式與 C 程式之間進行通訊。

請先啓動 C 程式，在 Socket 上接聽。當 Java 程式連接 Socket 之後，C 程式就會使用該 Socket 連線來傳送字串給 Java 程式。從 C 程式傳送的字串爲字碼頁 819 的「美國國家標準交換碼 (ASCII)」字串。

請在「Qshell 直譯器」指令行或其他 Java 平台上，使用指令 `java TalkToC xxxxx nnnn` 來啓動 Java 程式。或者，在 iSeries 指令行中輸入 `JAVA TALKTOC PARM(xxxxx nnnn)`，以啓動 Java 程式。`xxxxx` 指執行 C 程式所在系統的網域名稱或 Internet Protocol (IP) 位址。 `nnnn` 指 C 程式使用的 Socket 埠號。請使用此埠號作爲呼叫 C 程式的第一個參數。

範例：1：TalkToC 用戶端類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
import java.net.*;
import java.io.*;

class TalkToC
{
   private String host = null;
   private int port = -999;
   private Socket socket = null;
   private BufferedReader inStream = null;
```

```
   public static void main(String[] args)
   {
      TalkToC caller = new TalkToC();
      caller.host = args[0];
      caller.port = new Integer(args[1]).intValue();
      caller.setUp();
      caller.converse();
      caller.cleanUp();

   }  // end main() method

   public void setUp()
   {
      System.out.println("TalkToC.setUp() invoked");

      try
      {
         socket = new Socket(host, port);
         inStream = new BufferedReader(new InputStreamReader(
                                      socket.getInputStream()));
      }
      catch(UnknownHostException e)
      {
         System.err.println("Cannot find host called: " + host);
         e.printStackTrace();
         System.exit(-1);
      }
      catch (IOException e) {
System.err.println("Could not establish connection for " + host);
         e.printStackTrace();
         System.exit(-1);
      }
   } // end setUp() method

   public void converse()
   {
      System.out.println("TalkToC.converse() invoked");

      if (socket != null && inStream != null)
      {
         try
         {
            System.out.println(inStream.readLine());
         }
         catch (IOException e) {
System.err.println("Conversation error with host " + host);
            e.printStackTrace();
         }

      } // end if

   }  // end converse() method

   public void cleanUp()
   {
      try
      {

         if(inStream != null)
         {
            inStream.close();
         }
         if(socket != null)
         {
            socket.close();
         }
```

```
      } // end try
      catch(IOException e)
      {
         System.err.println("Error in cleanup");
         e.printStackTrace();
         System.exit(-1);
      }
   } // end cleanUp() method

} // end TalkToC class
```

啓動 SockServ.C 時，請傳入埠號作爲參數。例如，CALL SockServ '2001'。

範例 2：SockServ.C 伺服器程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/time.h>

void main(int argc, char* argv[])
{
   int    portNum = atoi(argv[1]);
   int    server;
   int    client;
   int    address_len;
   int    sendrc;
   int    bndrc;
   char*  greeting;
   struct sockaddr_in  local_Address;
   address_len = sizeof(local_Address);

   memset(&local_Address,0x00,sizeof(local_Address));
   local_Address.sin_family = AF_INET;
   local_Address.sin_port = htons(portNum);
   local_Address.sin_addr.s_addr = htonl(INADDR_ANY);

   #pragma convert (819)
   greeting = "This is a message from the C socket server.";
   #pragma convert (0)

   /*  allocate socket    */
   if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
   {
      printf("failure on socket allocation\n");
      perror(NULL);
      exit(-1);
   }

   /* do bind   */
   if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
   {
     printf("Bind failed\n");
     perror(NULL);
     exit(-1);
   }

   /* invoke listen   */
   listen(server, 1);
```

```
   /* wait for client request */
   if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
   {
     printf("accept failed\n");
     perror(NULL);
     exit(-1);
   }


   /* send greeting to client    */
   if((sendrc = send(client, greeting, strlen(greeting),0))<0)
   {
      printf("Send failed\n");
      perror(NULL);
      exit(-1);
   }

   close(client);
   close(server);

}
```

如需詳細資訊，請參閱使用 Socket 來進行處理作業之間的通訊。

範例：執行 Java 效能資料轉換器

您可以使用 iSeries 指令行或 Qshell 環境來執行「Java 效能資料轉換器 (JPDC)」。

使用 iSeries 指令行：

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

1. 在 iSeries 指令行，輸入「執行 Java (RUNJVA)」指令或 JAVA 指令。
2. 在類別參數行輸入 `com.ibm.as400.jpdc.JPDC`。
3. 在參數行輸入 `general pexdfn mydir/myfile myrdbdire`。
4. 在類別路徑參數行輸入 `'/QIBM/ProdData/Java400/ext/JPDC.jar'`。

   註: 若 CLASSPATH 環境變數已包含 `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 字串，則可以省略類別路徑。可以使用「新增環境變數 (ADDENVVAR)」指令、「變更環境變數 (CHGENVVAR)」指令或「處理環境變數 (WRKENVVAR)」指令，將此字串加入 CLASSPATH 環境變數中。

使用 Qshell 環境：

1. 輸入「啓動 Qshell (STRQSH)」指令來啓動 Qshell 直譯器。
2. 在指令行輸入：

```
java -classpath /QIBM/ProdData/Java400/ext/JPDC.jar com.ibm.as400/jpdc/JPDC
jinsight pexdfn mydir/myfile myrdbdire
```

   註: 若現行環境中已加入 `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 字串，則可以省略類別路徑。可以使用 ADDENVVAR、CHGENVVAR 或 WRKENVVAR 指令，將此字串加入現行環境中。

如需背景資訊，請參閱執行 Java 效能資料轉換器。

範例：在 Java 應用程式中內含 SQL 陳述式

以下 SQLJ 應用程式範例 App.sqlj，將使用靜態 SQL 在 DB2 範例資料庫的 EMPLOYEE 表格內擷取及更新資料。

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
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
{

  /**********************
      **  Register Driver **
   **********************/

  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;

     // URL is jdbc:db2:dbname
     String url = "jdbc:db2:sample";

     DefaultContext ctx = DefaultContext.getDefaultContext();
     if (ctx == null)
     {
       try
       {
         // connect with default id/password
         Connection con = DriverManager.getConnection(url);
         con.setAutoCommit(false);
         ctx = new DefaultContext(con);
       }
       catch (SQLException e)
       {
         System.out.println("Error: could not get a default context");
         System.err.println(e) ;
         System.exit(1);
       }
       DefaultContext.setDefaultContext(ctx);
     }

     // retrieve data from the database
     System.out.println("Retrieve some data from the database.");
```

```
    #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

    // display the result set
    // cursor1.next() returns false when there are no more rows
    System.out.println("Received results:");
    while (cursor1.next()) // 3
    {
      str1 = cursor1.empno(); // 4
      str2 = cursor1.firstnme();

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor1.close(); // 9

    // retrieve number of employee from the database
    #sql { SELECT count(*) into :count1 FROM employee }; // 5
    if (1 == count1)
      System.out.println ("There is 1 row in employee table");
    else
      System.out.println ("There are " + count1
                          + " rows in employee table");

    // update the database
    System.out.println("Update the database.");
    #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

    // retrieve the updated data from the database
    System.out.println("Retrieve the updated data from the database.");
    str1 = "000010";
    #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

    // display the result set
    // cursor2.next() returns false when there are no more rows
    System.out.println("Received results:");
    while (true)
    {
      #sql { FETCH :cursor2 INTO :str2 }; // 7
      if (cursor2.endFetch()) break; // 8

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
      System.out.println("");
    }
    cursor2.close(); // 9

    // rollback the update
    System.out.println("Rollback the update.");
    #sql { ROLLBACK work };
    System.out.println("Rollback done.");
  }
  catch( Exception e )
  {
    e.printStackTrace();
  }
 }
}
```

1宣告疊代子。此區段宣告兩種疊代子：

- App_Cursor1：宣告直欄資料類型及名稱，然後根據直欄名稱傳回直欄的值 (直欄的具名連結)。
- App_Cursor2：宣告直欄資料類型，然後根據直欄位置傳回直欄的值 (直欄的定位連結)。

2起始設定疊代子。使用查詢結果來起始設定疊代子物件 cursor1。查詢結果會儲存在 cursor1 內。

[3]將疊代子移至下一橫列。若已無其他列可擷取，cursor1.next() 方法會傳回布林 false。

[4]移動資料。具名 accessor 方法 empno() 會傳回現行列上 empno 直欄的值。 具名 accessor 方法 firstnme() 會傳回現行列上 firstnme 直欄的值。

[5]SELECT 資料傳入主變數。SELECT 陳述式會將表格列數傳入主變數 count1 內。

[6]起始設定疊代子。使用查詢結果來起始設定疊代子物件 cursor2。查詢結果會儲存在 cursor2 內。

[7]擷取資料。FETCH 陳述式會從結果表格中，將 ByPos 游標中宣告的第一欄的現行值傳回主變數 str2 內。

[8]檢查 FETCH.INTO 陳述式是否成功。若疊代子不在某列上，亦即，若前次試圖擷取某列失敗，endFetch() 方法會傳回布林 true。若前次試圖擷取某列成功，endFetch() 方法會傳回 false。呼叫 next() 方法時，DB2 會試圖提取一個橫列。FETCH...INTO 陳述式會隱含地呼叫 next() 方法。

[9]關閉疊代子。close() 方法會釋放疊代子佔用的任何資源。應該明確地關閉疊代子，確保能夠及時地釋放系統資源。

如需本範例的背景資訊，請參閱在 Java 應用程式中內含 SQL 陳述式。

範例：變更 Java 程式碼來使用用戶端 Socket Factory

以下範例將示範如何變更簡單的 Socket 類別 simpleSocketClient，使它改用 Socket Factory 來建立所有 Socket。第一個範例示範沒有 Socket Factory 的 simpleSocketClient 類別。第二個範例示範含有 Socket Factory 的 simpleSocketClient 類別。 在第二個範例中，simpleSocketClient 會重新命名爲 factorySocketClient。

範例 1：無 Socket Factory 的 Socket 用戶端程式

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* Simple Socket Client Program */

import java.net.*;
import java.io.*;

public class simpleSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
            System.out.println("java simpleSocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Create the socket and connect to the server.
    Socket  s = new Socket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.
```

範例 2：有 Socket Factory 的簡單 Socket 用戶端程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/* Simple Socket Factory Client Program */

// Notice that javax.net.* is imported to pick up the SocketFactory class.
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Change the original simpleSocketClient program to create a
    // SocketFactory and then use the socket factory to create sockets.

    SocketFactory socketFactory = SocketFactory.getDefault();

    // Now the factory creates the socket.  This is the last change
    // to the original simpleSocketClient program.

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Socket Factory。

範例：變更 Java 程式碼來使用伺服器 Socket Factory

以下範例將示範如何變更簡單的 Socket 類別 simpleSocketServer，使它改用 Socket Factory 來建立所有 Socket。第一個範例示範沒有 Socket Factory 的 simpleSocketServer 類別。第二個範例示範含有 Socket Factory 的 simpleSocketServer 類別。 在第二個範例中，simpleSocketServer 會重新命名爲 factorySocketServer。

範例 1：無 Socket Factory 的 Socket 伺服器程式

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* File simpleSocketServer.java*/

import java.net.*;
import java.io.*;

public class simpleSocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
```

```
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    ServerSocket    serverSocket =
      new ServerSocket(serverPort);

    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // This server just echoes back what you send it...

    byte buffer[] = new byte[4096];

    int bytesRead;

    // read until "eof" returned
    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead); // write it back
      os.flush();    // flush the output buffer
    }


    s.close();
    serverSocket.close();
  }       // end main()

}       // end class definition
```

範例 2：有 Socket Factory 的簡單 Socket 伺服器程式

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/* File factorySocketServer.java  */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Change the original simpleSocketServer to use a
    // ServerSocketFactory to create server sockets.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Now have the factory create the server socket.  This is the last
    // change from the original program.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);
```

```
    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

    // This server just echoes back what you send it...

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }


    s.close();
    serverSocket.close();
  }

}
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Socket Factory。

範例：變更 Java 用戶端來使用 Secure Sockets Layer

以下範例將示範如何變更 factorySocketClient 類別來使用 Secure Sockets Layer (SSL)。第一個範例示範未使用 SSL 的 factorySocketClient 類別。第二個範例示範使用 SSL 的同一個類別 (重新命名爲 factorySSLSocketClient)。

範例 1：無 SSL 支援的簡單 factorySocketClient 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* Simple Socket Factory Client Program */

import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    SocketFactory socketFactory = SocketFactory.getDefault();

    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
```

```
    .

    // The rest of the program continues on from here.
```

範例 2：使用 SSL 支援的簡單 factorySocketClient 類別

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
// Notice that we import javax.net.ssl.* to pick up SSL support
import javax.net.ssl.*;
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySSLSocketClient {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java factorySSLSocketClient serverHost serverPort");
      System.out.println("serverPort defaults to 3000 if not specified.");
      return;
    }
    if (args.length == 2)
      serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
                       serverPort);

    // Change this to create an SSLSocketFactory instead of a SocketFactory.
    SocketFactory socketFactory = SSLSocketFactory.getDefault();

    // We do not need to change anything else.
    // That's the beauty of using factories!
    Socket  s = socketFactory.createSocket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Secure Sockets Layer。

範例：變更 Java 伺服器來使用 Secure Sockets Layer

以下範例將示範如何變更 factorySocketServer 類別來使用 Secure Sockets Layer (SSL)。

第一個範例示範未使用 SSL 的 factorySocketServer 類別。第二個範例示範使用 SSL 的同一個類別 (重新命名爲 factorySSLSocketServer)。

範例 1：無 SSL 支援的簡單 factorySocketServer 類別

註: 使用程式碼範例，即表示您同意第 498 頁的『程式碼授權及免責聲明資訊』的條款。

```
/* File factorySocketServer.java  */
// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;
```

```
    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Change the original simpleSocketServer to use a
    // ServerSocketFactory to create server sockets.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Now have the factory create the server socket.  This is the last
    // change from the original program.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

範例 2：使用 SSL 支援的簡單 factorySocketServer 類別

註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

```
/* File factorySocketServer.java  */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
  public static void main (String args[]) throws IOException {

    int serverPort = 3000;

    if (args.length < 1) {
      System.out.println("java simpleSocketServer serverPort");
      System.out.println("Defaulting to port 3000 since serverPort not specified.");
    }
    else
        serverPort = new Integer(args[0]).intValue();

    System.out.println("Establishing server socket at port " + serverPort);

    // Change the original simpleSocketServer to use a
```

```
    // ServerSocketFactory to create server sockets.
    ServerSocketFactory serverSocketFactory =
      ServerSocketFactory.getDefault();
    // Now have the factory create the server socket.  This is the last
    // change from the original program.
    ServerSocket    serverSocket =
      serverSocketFactory.createServerSocket(serverPort);

    // a real server would handle more than just one client like this...

    Socket s = serverSocket.accept();
    BufferedInputStream is = new BufferedInputStream(s.getInputStream());
    BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it.

    byte buffer[] = new byte[4096];

    int bytesRead;

    while ((bytesRead = is.read(buffer)) > 0) {
      os.write(buffer, 0, bytesRead);
      os.flush();
    }

    s.close();
    serverSocket.close();
  }
}
```

如需相關背景資訊，請參閱變更 Java 程式碼來使用 Secure Sockets Layer。

IBM Developer Kit for Java 疑難排解

本主題說明如何尋找工作日誌及收集資料以分析 Java 程式。本主題亦提供暫時修訂程式 (PTF) 的相關資訊，同時說明如何取得 IBM Developer Kit for Java 的支援。

若您的程式經過長時間執行之後效能會下降，則表示記憶體洩漏方面的程式碼編寫可能有誤。您可以使用 iSeries iDoctor 中的 JavaWatcher 元件，協助您進行程式除錯，找出記憶體洩漏之處。如需相關資訊，請參閱 JavaWatcher。

限制

此清單識別 IBM Developer Kit for Java 中任何已知的限制或特有的行爲。

- 載入類別時，若找不到最高類別，則會發生錯誤，指出找不到原始類別。比方說，如果類別 B 延伸出類別 A，但載入類別 B 時卻找不到類別 A， 則即使實際上是找不到類別 A，錯誤訊息還是會表示找不到類別 B。出現找不到類別的錯誤時，請檢查此類別及其所有的最高類別，確定全部都在 CLASSPATH 中。對於載入的類別所實作的介面，同樣適用這項原則。
- 垃圾收集資料堆限制爲 240 GB。
- 可建構的物件數量沒有明確的限制。
- java.net backlog 參數在 iSeries 伺服器上的運用方式，可能不同於其他平台。例如：
  - Listen backlogs 0, 1
    - Listen(0) 表示容許一個擱置連線；不會停用 Socket。
    - Listen(1) 表示容許一個擱置註解，同時具有 Listen(0) 的意義。
  - Listen backlogs > 1

- 這將容許接聽佇列上保留許多擱置要求。當新的連線要求送達時，若佇列已達上限，則會刪除其中一個擱置要求。

- 不論使用什麼 JDK 版本，您在具備多緒能力的 (亦即安全緒) 環境中都只能使用 Java 虛擬機器。iSeries 伺服器爲安全緒，但有些檔案系統則不是。如需非安全緒檔案系統的清單，請參閱整合檔案系統。
- 「網際網路通訊協定」版次 6 (IPv6) 支援未完全實作，可能會發生一些不良效果。如需詳細資訊，請參閱 Socket。

尋找工作日誌來分析 Java 問題

請使用原本執行 Java 指令之工作的工作日誌，以及執行 Java 程式的批次即時 (BCI) 工作日誌，來分析 Java 失敗的原因。 這兩份日誌含有重要的錯誤資訊。

有兩種方法可以找出 BCI 工作的工作日誌。您可以在執行 Java 指令之工作的工作日誌中，找到記載的 BCI 工作名稱。然後，再利用此工作名稱來找出 BCI 工作的工作日誌。

您亦可透過下列步驟來找出 BCI 工作的工作日誌：

1. 在 iSeries 指令行上輸入「處理提交的工作 (WRKSBMJOB)」指令。
2. 移至清單底端。
3. 找到清單的最後一個工作，稱爲 QJVACMDSRV。
4. 針對此工作輸入選項 8 (使用排存檔)。
5. 畫面上會出現 QPJOBLOG 檔案。
6. 按 F11 來查看排存檔的檢視畫面 2。
7. 驗證日期與時間是否符合失敗發生的日期與時間。

   若日期與時間不符合您登出時的日期與時間，請繼續查閱已提交工作的清單。嘗試找出日期與時間符合您登出時的 QJVACMDSRV 工作日誌。

若實在找不到 BCI 工作的工作日誌，有可能是日誌尚未產生。若您將 QDFTJOBD 工作說明的 ENDSEP 值設得太高，或 QDFTJOBD 工作說明的 LOG 值指定 *NOLIST，即可能發生這種情形。請檢查這些值，予以適當地變更，讓 BCI 工作的工作日誌能夠順利產生。

對於已完成「執行 Java (RUNJVA)」指令的工作，請執行下列步驟以產生工作日誌：

1. 輸入 SIGNOFF *LIST。
2. 然後，重新登入。
3. 在 iSeries 指令行上輸入「使用排存檔 (WRKSPLF)」指令。
4. 移至清單底端。
5. 找出 QPJOBLOG 檔案。
6. 按 F11。
7. 驗證日期與時間是否符合您輸入登出指令時的日期與時間。

   若日期與時間不符合您登出時的日期與時間，請繼續查閱已提交工作的清單。嘗試找出日期與時間符合您登出時的 QJVACMDSRV 工作日誌。

收集資料以分析 Java 問題

若要收集資料以提出授權程式分析報告 (APAR)，請遵循下列步驟。

1. 提供完整的問題描述。

2. 儲存執行時導致問題的 Java 類別檔案。
3. 您可以使用 SAV 指令來儲存整合檔案系統中的物件。也可能需要儲存此程式必須執行的其他類別檔案。在試著重新產生問題時，亦可能需要儲存及傳送整個目錄供 IBM 使用。以下爲如何儲存整個目錄的範例。

   範例：儲存目錄

   註: 請閱讀程式碼範例免責聲明中的重要法律資訊。

   ```
   SAV DEV('/QSYS.LIB/TAP01.DEVD') OBJ(('/mydir'))
   ```

   如有可能，請儲存問題所涉及之任何 Java 類別的來源檔。這有助於 IBM 重新產生及分析問題。
4. 儲存任何服務程式，內含執行程式時所需的原生方法。
5. 儲存執行 Java 程式所需的任何資料檔。
6. 附上如何重新產生問題的完整說明。

   包括：
   - CLASSPATH 環境變數的值。
   - 已執行之 Java 指令的說明。
   - 對於程式所需之任何輸入如何回應的說明。
7. 併入接近失敗時發生的任何垂直授權內碼 (VLIC) 日誌。
8. 從執行 Java 虛擬機器的互動式工作及 BCI 工作中新增工作日誌。

套用暫時修訂程式

從 i5/OS V5R4 開始，您可以在系統處於作用中時，使用「顯示 Java 虛擬機器工作 (DSPJVMJOB)」CL 指令來管理 JVM 工作並套用 PTF。

許多「Java 暫時修訂程式 (PTF)」會影響 JVM，例如，若在執行 JVM 工作時套用程式碼，則套用 PTF 會導致無法預期的結果。在過去，會將部分 Java PTF 延遲到系統上執行起始程式載入 (IPL) 後才使用，以確保系統上未執行 JVM 工作。然而，許多使用者發現這樣做很不方便。已新增了 JVM 前置條件，這樣*如果系統上沒有作用中的 JVM*，就可以立即套用原本延遲的大部分 PTF。DSPJVMJOB 指令可讓您查看哪些工作在執行 JVM。利用此資訊，您便可以在套用 PTF 之前適當地結束包含作用中 JVM 的工作，而不必等待 IPL 後才套用 PTF。

若要進一步瞭解 DSPJVMJOB 指令，請參閱 CL 主題中的顯示 Java 虛擬機器工作。

相關資訊

維護及管理 i5/OS 及相關軟體

使用軟體修訂程式

取得 IBM Developer Kit for Java 的支援

IBM Developer Kit for Java 的支援服務，是依據 iSeries 軟體產品的一般條款而提供。支援服務包括程式服務、語音支援及諮詢服務。

如需相關資訊，請參閱 IBM iSeries 首頁之「支援」主題所提供的線上資訊。請使用 IBM 5722-JV1 (IBM Developer Kit for Java) 的支援服務。或聯絡當地的 IBM 業務代表。

根據 IBM 的指示，您可能需要取得新版的 IBM Developer Kit for Java，才能享有「永續程式服務」。如需相關資訊，請參閱多重 Java Development Kit (JDK) 的支援。

關於 IBM Developer Kit for Java 程式的問題，請透過程式服務或語音支援來解決。關於應用程式設計或除錯的問題，請利用諮詢服務來解決。

關於 IBM Developer Kit for Java 應用程式介面 (API) 呼叫的問題，則由諮詢服務負責解決，但下列情形除外：

1. 在較簡單的程式中重建之後，確定屬於 Java API 的問題。
2. 它是一個須透過文件來說明的問題。
3. 它是一個關於範例或文件的位置的問題。

諮詢服務提供所有程式設計方面的協助。其中包括 IBM Developer Kit for Java 授權程式 (LP) 產品提供的程式範例。至於其他範例，可在網際網路的 IBM iSeries Home Page 上取得，但對此並不提供支援。

IBM Developer Kit for Java LP 提供解決問題的相關資訊。若認爲 IBM Developer Kit for Java API 可能有問題，請提供簡單的程式來示範錯誤之處。

IBM Developer Kit for Java 的相關資訊

以下是關於 IBM Developer Kit for Java 的 Javadoc 參考資訊。

Javadoc

- iSeries 專屬的 JAAS Javadoc
- JAAS API 規格
- Java 2 Platform 標準版 API 規格

請參閱下列關於 IBM Developer Kit for Java 的參考資訊。

Java 命名和目錄介面

「Java 命名和目錄介面 (JNDI)」屬於 JavaSoft 平台應用程式設計介面 (API)。透過 JNDI，可以直接連接多個命名及目錄服務。只要利用此介面，就可以建置功能強大、可攜性高且接受目錄管理的 Java 應用程式。

JavaSoft 聯合業界眾多頂尖的廠商共同開發 JNDI 規格，包括 IBM、SunSoft、Novell、Netscape 及 Hewlett-Packard Co.。

附註：i5/OS Java Runtime Environment (JRE) 及 IBM Developer Kit for Java 提供的 Java 2 Platform, Software Development Kit (J2SDK) 版本，已內含 Sun LDAP 提供者。因爲i5/OS Java 支援已包含 Sun LDAP 提供者，所以這項支援就不再包含 ibmjndi.jar 檔案。ibmjndi.jar 檔案提供 IBM 開發的 LDAP 服務提供者，供舊版的 J2SDK 使用。

如需 JNDI 的相關資訊，請參閱 Sun Microsystems 公司的「Java 命名和目錄介面」。

鏈結集合

Sun Microsystems 公司的「Java 命名和目錄介面」

JavaMail

JavaMail API 提供一組抽象類別，可以建立一個電子 (電子郵件) 系統的模型。此 API 提供閱讀及寄送郵件的一般郵件功能，但需要服務提供者來實作通訊協定。

服務提供者負責實作特定的通訊協定。例如，「簡易郵件傳送通訊協定 (SMTP)」即爲一種傳送電子郵件的傳輸通訊協定。「郵局通訊協定 (POP)」則爲接收電子郵件的標準通訊協定。「網際網路訊息存取通訊協定 (IMAP)」是 POP3 以外另一種可選擇的通訊協定。

除了需要服務提供者之外，JavaMail 還需要 JavaBeans Activation Framework (JAF) 來處理非純文字的郵件內容。其中包括「多用途網際網路郵件延伸 (MIME)」、「全球資源定位器 (URL)」網頁及附加檔案。

所有 JavaMail 元件皆隨 IBM Developer Kit for Java 一起提供。這些元件包括：

- **mail.jar** 此 JAR 檔包含 JavaMail API，以及 SMTP、POP3 及 IMAP 這三種服務提供者。
- **activation.jar** 此 JAR 檔包含 JavaBeans Activation Framework。

如需相關資訊，請參閱 Sun Microsystems 公司 JavaMail 文件。

鏈結集合

JavaMail

Java 列印服務

「Java 列印服務 (JPS)」API 可讓您在所有 Java 平台上列印。Java 1.4 及後續版本提供一個組織架構，可讓 Java 執行時間環境及協力廠商在其中提供串流產生器外掛程式，藉以產生各種列印格式，例如 PDF、Postscript 及 Advanced Function Presentation™ (AFP™)。這些外掛程式可透過二維 (2D) 圖形呼叫來建立輸出格式。

iSeries 列印服務代表一種印表機裝置，可以使用 i5/OS 指令「建立裝置說明 (印表機) (CRTDEVPRT)」在 iSeries 上配置。建立印表機裝置時，請指定發佈資訊參數。這樣可以增加 iSeries 列印服務所支援的列印服務屬性數量。

若印表機支援「簡易網路管理通訊協定 (SNMP)」，請在 iSeries 上配置印表機。請在 CRTDEVPRT 指令上指定 *IBMSNMPDRV，作爲系統驅動程式參數的值。列印服務會利用 SNMP，擷取已配置的印表機的特定資訊 (印表機服務屬性)。

iSeries 支援的 Doc Flavors 包括 *AFPDS、*SCS、*USERASCII - (PCL)、*USERASCII - (Postscript) 及 *USERASCII - (PDF)。在 CRTDEVPRT 指令的「發佈資訊」內，請在「支援的資料串流」參數中指定印表機支援的 Doc Flavors。

當應用程式在 iSeries 上使用列印服務來列印工作 (文件) 時，列印服務會在印表機裝置同名的輸出佇列中 (亦同於 PrinterName 屬性中指定的名稱)，將文件放入排存檔內。在文件列印到印表機裝置之前，請先以指令 STRPRTWTR 啓動印表機寫出器。

除了「Java 列印服務」規格所定義的屬性之外，對於所有的 Doc Flavors，iSeries 列印服務還支援下列屬性：

- PrinterFile (指定於建立排存檔時所用的印表機檔案、名稱及檔案庫)
- SaveSpooledFile (指出是否儲存排存檔)
- UserData (10 個字元的字串，存放使用者定義的資料)
- JobHold (指出是否保留排存檔)
- SourceDrawer (指出輸出媒體所用的送紙匣)

使用 JDK 1.5 時如何啓用 JPS

以下爲啓用「Java 列印服務」時需要設定的符號鏈結：

```
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)
```

| 如需相關資訊，請參閱 Sun Microsystems Java Print Service 文件。

程式碼授權及免責聲明資訊

IBM 授與您使用所有程式設計程式碼範例的非專屬授權，您可以利用這些範例來產生符合您需求的類似函數。

| 除法律規定不得排除的保證外，IBM、IBM 之程式開發人員及供應商不附具任何明示或默示之保證，包含且不
| 限於任何相關技術支援之未侵害他人權利之保證、或可商用性及符合特定效用等之默示保證。

| 在任何情況下，IBM、IBM 之程式開發者或供應商對下列情事均不負賠償責任，即使被告知該情事有可能發生
| 時，亦同：

| 1. 資料之滅失或毀損；
| 2. 直接、特殊、附帶或間接的傷害或其他衍生之經濟損害；或
| 3. 利潤、營業、收益、商譽或預期節餘等項之損失。

| 倘法律規定不得排除或限制賠償責任時，則該排除或限制無效。

附錄. 注意事項

本資訊是針對 IBM 在美國所提供之產品與服務開發出來的。

而在其他國家中，IBM 不見得有提供本書中所提的各項產品、服務、或功能。要知道您所在區域是否可用到這些產品與服務時，請向當地的 IBM 服務代表查詢。本書在提及 IBM 產品、程式或服務時，不表示或暗示只能使用 IBM 的產品、程式或服務。只要未侵犯 IBM 的智慧財產權，任何功能相當的產品、程式或服務都可以取代 IBM 的產品、程式或服務。不過，其他非 IBM 產品、程式或服務在運作上的評價與驗證，其責任屬於使用者。

在這本書或文件中可能包含著 IBM 所擁有之專利或專利申請案。本書使用者並不享有前述專利之任何授權。您可以用書面方式來查詢授權，來函請寄到：

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

若要查詢有關二位元組 (DBCS) 資訊的特許權限事宜，請聯絡您國家的 IBM 智慧財產部門，或者用書面方式寄到：

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

下列段落若與當地之法令抵觸，則不適用之： IBM 僅以「現狀」提供本出版品，而不爲任何明示或默示之保證 (包括但不限於產品未涉侵權、可售性或符合特定效用的保證。) 倘若干地區在特定交易中並不許可相關明示或默示保證之棄權聲明，則於該等地區之特定交易，此項聲明不適用之。

本資訊中可能包含技術上或排版印刷上的錯誤。因此，IBM 會定期修訂；並將修訂後的內容納入新版中。同時，IBM 得隨時修改或變更本出版品中所提及的產品及程式。

本資訊中任何對非 IBM 網站的敘述僅供參考，IBM 對該等網站並不提供保證。該等網站上的資料，並非 IBM 產品所用資料的一部分，如因使用該等網站而造成損害，其責任由 貴客戶自行負責。

IBM 得以其認定之各種適當方式使用或散布由 貴客戶提供的任何資訊，而無需對您負責。

本程式之獲授權者若希望取得相關資料，以便使用下列資訊者可洽詢 IBM。其下列資訊指的是： (1) 獨立建立的程式與其他程式 (包括此程式) 之間更換資訊的方式 (2) 相互使用已交換之資訊方法 若有任何問題請聯絡：

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

上述資料之取得有其特殊要件，在某些情況下必須付費方得使用。

IBM 基於雙方之「IBM 客戶合約」、「IBM 國際程式授權合約」、「IBM 機器碼授權合約」或任何同等合約之條款，提供本出版品中所述之授權程式與其所有適用的授權資料。

任何此處涵蓋的執行效能資料都是在一個受控制的環境下決定出來的。因此，於其他不同作業環境之下所得的結果，可能會有很大的差異。有些測定已在開發階段系統上做過，不過這並不保證在一般系統上會出現相同結果。再者，有些測定可能已透過推測方式評估過。但實際結果可能並非如此。本文件的使用者應根據其特有的環境，驗證出適用的資料。

本資訊所提及之非 IBM 產品資訊，係一由產品的供應商，或其出版的聲明或其他公開管道取得。IBM 並未測試過這些產品，也無法確認這些非 IBM 產品的執行效能、相容性、或任何對產品的其他主張是否完全無誤。如果您對非 IBM 產品的性能有任何的疑問，請逕向該產品的供應商查詢。

有關 IBM 未來動向的任何陳述，僅代表 IBM 的目標而已，並可能於未事先聲明的情況下有所變動或撤回。

所有顯示之 IBM 產品售價僅爲 IBM 產品之一般市場價格，可能於未事先聲明之情況下有所變動。經銷商售價可能有所不同。

本資訊僅供規劃用途。所提及的產品發行之前，本書內含的資訊有變動的可能。

本資訊中含有日常商業活動所用的資料及報告範例。爲了提供完整的說明，這些範例包括個人、公司、廠牌和產品的名稱。這些名稱全屬虛構，若與任何公司的名稱和住址雷同，純屬巧合。

著作權授權：

本資訊包含原始語言的範例應用程式，用以說明各種作業平台上的程式設計技術。您可以基於研發、使用、銷售或散佈符合作業平台 (用於執行所撰寫的範例程式) 之應用程式設計介面的應用程式等目的，以任何形式複製、修改及散佈這些範例程式，而無需付費給 IBM。這些範例尚未經過在所有情況下測試。因此，IBM 不保證或暗示這些程式的穩定性、服務能力或功能。

這些範例程式或是任何衍生著作的每一份拷貝或任何部份，都必須具有下列的著作權聲明：

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

若您是以電子檔檢視此資訊，則照片和彩色圖例可能不會出現。

程式設計介面資訊

本 IBM Developer Kit for Java 出版品文件是使用允許客戶撰寫程式以取得 IBM Developer Kit for Java 服務的「程式設計介面」。

商標

下列術語是 IBM 公司在美國及 (或) 其他國家的商標。

- Advanced Function Presentation
- AFP
- AIX
- AT
- C/400
- DB2
- DB2 Universal Database

| Distributed Relational Database Architecture
| DRDA
| i5/OS
| IBM
| Integrated Language Environment
| iSeries
| PowerPC
| VisualAge
| WebSphere

Microsoft、Windows、Windows NT 以及 Windows 商標是 Microsoft Corporation 在美國及 (或) 其他國家的商標。

Java 以及所有與 Java 有關的商標是 Sun Microsystems, Inc. 在美國及 (或) 其他國家的商標。

UNIX 是 The Open Group 在美國及其他國家的註冊商標。

其他公司、產品及服務名稱，可能是其他公司的商標或服務標誌。

條款

根據下述條款，授予您對這些出版品的使用權限。

個人使用：您可複製該等出版品供個人及非商業性用途使用，惟應註記 IBM 著作權標示及其他所有權歸屬 IBM 之相關文字。未經 IBM 明示同意，您不得散佈、展示或改作該等出版品或其任何部份。

商業使用：您可以複製、散佈及展示該等出版品僅供企業內部使用，惟應註記 IBM 著作權標示及其他所有權歸屬 IBM 之相關文字。未經 IBM 明示同意，您不得改作該等出版品，也不得於企業外複製、散佈或展示該等出版品或其任何部份。

除本使用聲明中明確授予之許可外，使用者就出版品或任何包含於其中之資訊、資料、軟體或其他智慧財產權，並未取得其他任何明示或默許之許可、軟體授權或權利。

使用者對於出版品之使用如危害 IBM 的權益，或 IBM 認定其未遵照上述指示使用出版品時，IBM 得隨時撤銷此處所授予之許可。

除非您完全遵守所有適用之一切法規，包括所有美國出口法規，否則您不得下載、出口或再輸出此等資訊。

IBM 對於該等出版品之內容不爲任何保證。出版品依其「現狀」提供，不附帶任何明示或默示之擔保，其中包括 (但不限於) 適售性、未涉侵權及適合特定用途之默示擔保責任。

讀者意見表

為使本書盡善盡美，本公司極需您寶貴的意見；懇請您閱讀後，撥冗填寫下表，惠予指教。

請於下表適當空格內，填入記號 (∨)；我們會在下一版中，作適當修訂，謝謝您的合作!

評估項目	評 估 意 見		備 註
正 確 性	內容說明與實際程序是否符合	□是□否	
	參考書目是否正確	□是□否	
一 致 性	文句用語及風格，前後是否一致	□是□否	
	實際產品介面訊息與本書中所提是否一致	□是□否	
完 整 性	是否遺漏您想知道的項目	□是□否	
	字句、章節是否有遺漏	□是□否	
術語使用	術語之使用是否恰當	□是□否	
	術語之使用，前後是否一致	□是□否	
可 讀 性	文句用語是否通順	□是□否	
	有否不知所云之處	□是□否	
內容說明	內容說明是否詳盡	□是□否	
	例題說明是否詳盡	□是□否	
排版方式	本書的形狀大小，版面安排是否方便閱讀	□是□否	
	字體大小，顏色編排，是否有助於閱讀	□是□否	
目錄索引	目錄內容之編排，是否便於查找	□是□否	
	索引語錄之排定，是否便於查找	□是□否	
	※評估意見為 "否" 者，請於備註欄提供建議。		

其他： (篇幅不夠時，請另外附紙說明。)

上述改正意見，一經採用，本公司有合法之使用及發佈權利，特此聲明。

註： 您也可將寶貴的意見以電子郵件寄至 tscadmin@tw.ibm.com，謝謝。

IBM 系統 - iSeries

RZAH-A000-09

程式設計
IBM Developer Kit for Java

版本 5 版次 4