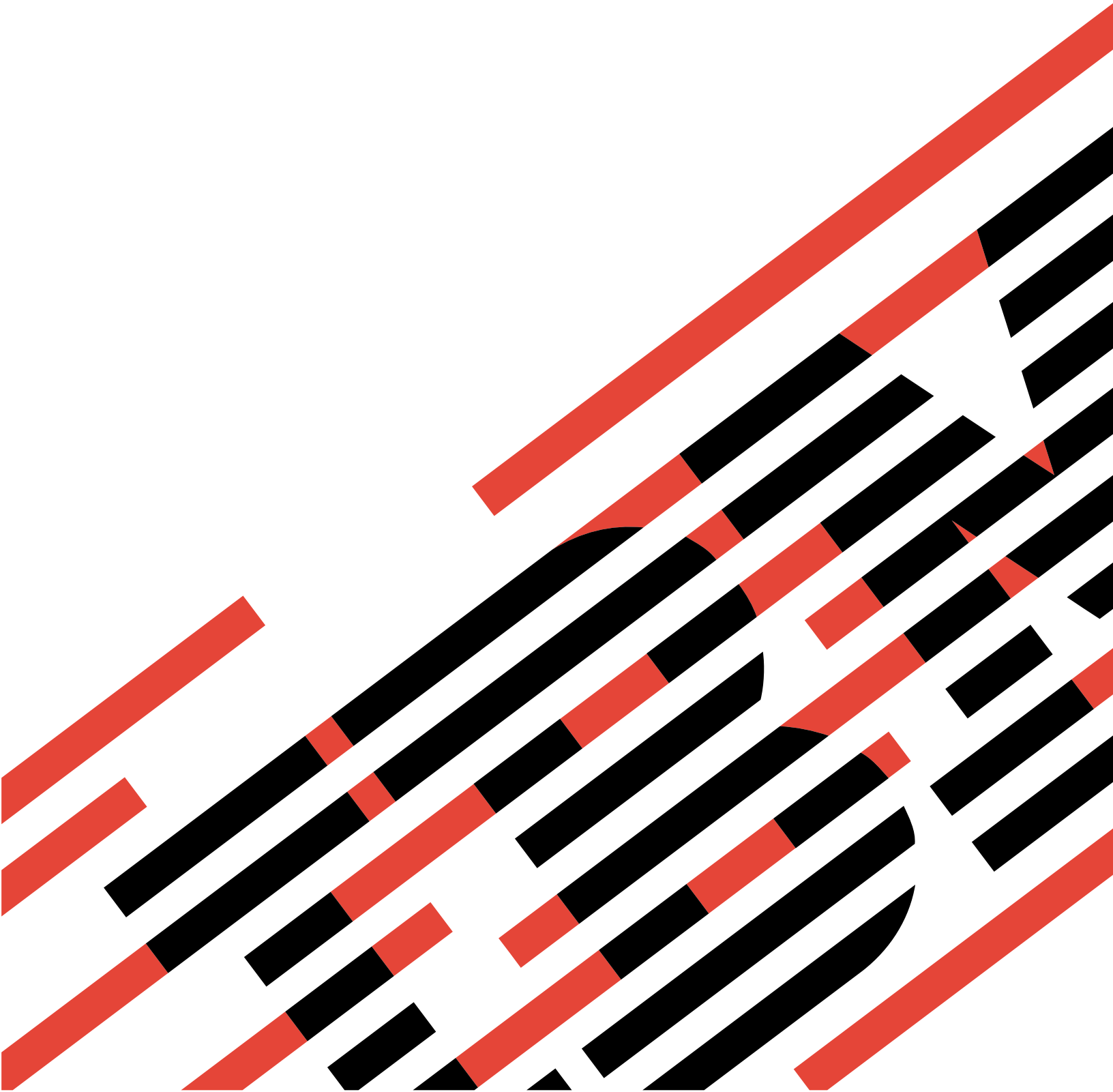
IBM

eserver

iSeries

IBM Developer Kit for Java

버전 6



IBM

@server

iSeries

IBM Developer Kit for Java

버전 6

목차

제 1 장 IBM Developer Kit for Java

IBM(R) Developer Kit for Java(TM)는 iSeries(TM) 서버 환경에서 사용하도록 최적화되었습니다. Java 프로그래밍과 사용자 인터페이스의 호환성을 활용하므로 iSeries 서버를 위한 사용자 자신의 어플리케이션을 개발할 수 있습니다.

IBM Developer Kit for Java는 iSeries 서버에서 Java 프로그램을 작성하고 실행할 수 있습니다. IBM Developer Kit for Java는 Sun Microsystems, Inc. Java Technology와 호환이 가능한 것으로서 사용자가 JDK(Java Development Kit) 문서에 익숙한 것으로 가정합니다. 정보를 쉽게 사용할 수 있도록 Sun Microsystems, Inc. 정보에 대한 링크를 제공합니다.

Sun Microsystems, Inc.의 Java Development Kit 문서에 대한 링크가 작동하지 않는 경우 필요한 정보에 대해 HTML 참조 문서를 참조하십시오.

WWW의 The Source for Java Technology java.sun.com 에서 이 정보를 찾을 수 있습니다.

IBM Developer Kit for Java를 사용하는 방법에 대한 자세한 내용은 다음의 주제 중 하나를 선택하십시오.

- 이 주제 인쇄는 IBM Developer Kit for Java의 인쇄 가능한 PDF 파일이나 압축된 패키지를 다운로드하는 방법에 대한 설명을 제공합니다.
- V5R2의 새로운 사항은 최신제품 및 정보 갱신을 강조합니다.
- 시작에서는 간단한 Hello World Java 프로그램의 설치, 구성, 작성 및 수행 방법, 다운로드 및 설치, 릴리스간 호환성에 대한 정보를 제공합니다.
- 사용자 정의는 서버에서 시간대, 시스템 등록 정보 및 서버의 클래스 경로 구성을 사용자 정의하는 방법에 대한 지침을 제공합니다.
- 호환성은 릴리스 간 Java 클래스 파일 호환성에 대한 정보를 제공합니다.
- 데이터베이스 액세스는 IBM Developer Kit for Java가 Java 프로그램으로 하여금 iSeries 데이터베이스 파일에 액세스할 수 있도록 허용하는 방법을 설명합니다.
- 기타 프로그래밍 언어에서는 JNI(Java 원시 인터페이스), java.lang.Runtime.exec(), 프로그램간 통신 및 Java 호출 API를 사용하여 Java 이외의 언어로 작성한 코드를 호출하는 방법을 보여줍니다.
- Java 플랫폼은 Java 애플릿 및 어플리케이션의 개발 및 관리를 위한 환경을 설명하고 Java 언어, Java 패키지 및 JVM(Java Virtual Machine)으로 구성되어 있습니다.
- 확장 주제는 일괄처리 작업에서 Java를 실행하는 방법에 대한 지침을 제공하며 통합 파일 시스템에서 Java 프로그램을 표시, 실행 또는 디버그하는데 필요한 Java 파일 권한을 설명합니다.

- GUI 없이 호스트에서 실행에는 Remote AWT(Abstract Window Toolkit), CBJ(Class Broker for Java) 또는 NAWT(Native Abstract Windowing Toolkit)로 Java 프로그램을 설정 및 실행하는 방법에 대한 정보가 들어 있습니다.
- 보안에서는 허용한 권한에 대한 자세한 내용과 Java 어플리케이션에서 소켓 스트림을 보안하기 위해 SSL을 사용하는 방법을 설명합니다.
- 성능에서는 Java 성능을 조정하는 방법을 설명합니다.
- 명령 및 툴에서는 Java 명령과 Java 툴 사용법에 대해 자세히 설명합니다.
- 선택적 패키지는 Java 어플리케이션을 개발하기 위해 선택적으로 사용할 수 있는 JNDI(Java Naming and Directory Service) 및 JavaMail과 같은 패키지를 나열합니다.
- 디버깅에서는 Java 프로그램을 디버그하는 방법을 설명합니다.
- 문제점 해결에서는 작업 기록부를 찾고 Java 프로그램 분석을 위한 자료를 수집하는 방법을 설명합니다. 이 주제는 프로그램 임시 수정(PTF) 및 IBM Developer Kit for Java에 대한 지원을 얻는 방법에 대한 정보도 제공합니다.
- 코드 예는 이 정보에 있는 모든 코드 예로 직접 링크합니다.
- 참조는 모든 Javadoc 및 API 참조 정보에 직접 링크합니다.

IBM Developer Kit for Java의 V5R2 새로운 사항

다음 주제는 V5R2에서 IBM Developer Kit for Java(TM)의 변경사항 요점입니다. JDK(Java Development Kit) 1.1.8 및 J2SDK(Java 2 Software Development Kit), Standard Edition, 버전 1.4의 특정 변경사항이 정리되어 있습니다. V5R2 일반 릴리스 이후 갱신사항은 다음 목록의 맨 아래 표시됩니다.

시작하기

- 복수 JDK 지원에 IBM이 지원하는 각 JDK의 정보가 있습니다.
- iSeries 서버에 네트워크 드라이브 맵핑 및 iSeries 서버에서 디렉토리 작성은 시작하기로 이동되었습니다.
- Hello World Java 프로그램 작성, 컴파일 및 실행에 약간의 변경이 있습니다.

사용자 정의

- 436 페이지의 『클래스 로더 캐시』에 대한 등록 정보를 포함하여 새로운 시스템 등록 정보가 있습니다.

데이터베이스 액세스

- JDBC 절에는 중점적인 변경 내용이 나옵니다.
- Java 저장 프로시듀어 및 Java 사용자 정의 스칼라 함수 섹션이 추가되었습니다.

GUI가 없는 호스트에서 실행

- 갱신 사항은 Remote AWT(Abstract Window Toolkit)를 사용하여 Java 프로그램 실행 주제를 참조하십시오.

명령 및 툴

- ANZJVM(JVM 분석) 명령이 CL 명령 섹션에 추가되었습니다.
- 453 페이지의 『Java idlj 툴』이 추가되었습니다.
- Java가 지원하는 iSeries Navigator 명령에 iSeries Navigator에 대한 몇 가지 변경사항을 반영시켰습니다.

선택적 패키지

- JNDI LDAP 제공자 프로그래밍 안내서가 추가되었습니다.

디버깅

- 사용자 정의 클래스 로더를 통해 로드한 Java 클래스에 대해 새로운 디버그 지원이 있습니다.

코드 예

- 더 많은 코드 예를 추가했습니다.

이 주제 인쇄

- IBM Developer Kit for Java 정보의 PDF는 이 주제 인쇄에 있습니다.

참조서

- IBM Developer Kit for Java 참조서 섹션이 추가되었으며 Javadoc 및 API 참조 정보가 들어 있습니다.

특정 버전에 대한 변경사항

사용자가 선택한 버전에 해당하는 정보는 아래의 링크를 클릭하십시오.

- Java Development Kit, 버전 1.1.8
- Java 2 Software Development Kit, 표준판, 버전 1.4

2002년 9월 26일 릴리스의 새로운 사항

복수 JDK(Java Development Kit) 지원

이 기술 갱신사항은 하나 이상의 JDK를 설치할 때 OS/400(R)이 디폴트 JDK를 판별하기 위해 사용하는 우선순위를 구분합니다.

Java 프로그램에서 제어 언어(CL) 호출

이 기술 갱신사항은 Java 프로그램에서 제어 언어(CL) 명령을 호출할 때 다른 JDK에 필요한 분리문자 정보를 추가합니다.

2002년 8월 30 릴리스의 새로운 사항

NAWT(Native Abstract Windowing Toolkit)

NAWT(Native Abstract Windowing Toolkit) 변경사항 정보에는 JDK(Java Development Kit) 버전 1.3 지원과 NAWT 설치 지침이 들어 있습니다. 설치 지침은 현재 NAWT가 사용하는 PRPQ 5799-PTL의 확장 기능이 반영된 것입니다.

JGSS(Java Generic Security Service)
JGSS(Java Generic Security Service)는 인증 및 보안 메세징을 위한 일반 인터페이스인 IBM JGSS 정보를 제공하는 새 주제입니다. 이 인터페이스하에서는 비밀키, 공용키 또는 기타 보안 기법을 기반으로 다양한 보안 메카니즘을 접목할 수 있습니다.

새로운 사항 또는 변경사항을 보는 방법

기술적으로 변경된 위치를 찾을 때 다음 정보를 사용하십시오.

- 신규 정보나 변경 정보가 시작되는 위치를 표시하는 이미지.

- 
  신규 정보나 변경 정보가 끝나는 위치를 표시하는 이미지.

이 릴리스의 새로운 사항 또는 변경된 사항에 대한 다른 정보를 찾으려면 사용자 메모  를 참조하십시오.

JDK(Java Development Kit) 1.1.8, V5R2의 새로운 사항

JDK(Java(TM) Development Kit) 1.1.8의 V5R2에는 고유한 변경사항이 없습니다.

J2SDK(Java 2 Software Development Kit), 표준판, 버전 1.4의 V5R2 새로운 사항

다음 주제는 J2SDK(Java 2 Software Development Kit), 표준판 버전 1.4의 V5R2에서 IBM Developer Kit for Java(TM)의 변경사항 요점입니다.

주: 이 절에서는 J2SDK, 버전 1.4에 고유하거나 관심이 있는 변경사항만 논의합니다. JDK(Java Development Kit) 1.1.8의 새로운 사항 주제의 갱신 관련 정보도 J2SDK, 버전 1.4에 적용됩니다.

사용자 정의

- 시간대 환경 변수는 iSeries 서버에서의 시간대 변수 설정과 같은 고유 특징을 보여줍니다. Java 2 SDK(J2SDK), Standard Edition, 버전 1.3 이상의 일부인 원시 **getSystemTimeZoneID()**를 사용하려면 이를 설정해야 합니다.
- 신규 J2SDK용 시스템 등록 정보가 추가되었습니다.

보안

- JAAS(Java Authentication and Authorization Service)는 Java 2 Software Development Kit, v 1.3(JDK 1.3)이상의 표준 부가 제품입니다. 현재, Java 2는 코드 기점과 코드를 서명한 사람에 기초하여 액세스 제어(코드 소스 기본 액세스 제어)를 제공합니다. 그러나 코드를 실행하는 사람에 기초하여 추가 액세스 제어를 시행하는 능력은 부족합니다. JAAS는 Java 2 보안 모델에 이 지원을 추가하는 구조를 제공합니다.

- JCE(Java Cryptography Extension) 1.2는 J2SDK(Java 2 Software Development Kit), Standard Edition에 대한 표준 부가 제품입니다. iSeries 서버에서의 JCE 구현은 Sun Microsystems, Inc.의 구현과 호환됩니다. 이 문서에서는 iSeries 구현의 고유한 특징을 다룹니다. 여기에서는 사용자가 JCE 부가 제품에 대한 일반 문서에 익숙한 것으로 가정합니다.

선택적 패키지

- Java 인쇄 서비스 API를 사용하면 모든 Java 플랫폼에서 인쇄가 가능합니다. Java 런타임 환경 및 써드 파티는 PDF, Postscript 및 AFP와 같이 다양한 인쇄 형식을 작성하기 위한 스트림 생성기 플러그인을 제공할 수 있습니다.
- JavaMail API는 전자(전자 우편) 시스템을 모델화하는 추상 클래스 세트입니다. API는 Java 기반 전자 우편 및 메세징 어플리케이션을 빌드할 수 있도록 플랫폼과 상관 없으며 프로토콜과도 상관 없는 구조를 제공합니다.

이 주제 인쇄

PDF 버전을 보거나 다운로드하려면 IBM Developer Kit for Java(TM)(2159KB 또는 450 페이지 정도)를 선택하십시오.

PDF 파일 저장

워크스테이션에 PDF를 저장하려면 다음과 같이 하십시오.

1. 브라우저에서 PDF를 마우스 오른쪽 버튼으로 클릭하십시오(위의 링크를 마우스 오른쪽 버튼으로 클릭).
2. **다른 이름으로 대상 저장...**을 클릭하십시오.
3. PDF를 저장할 디렉토리로 가십시오.
4. **저장**을 클릭하십시오.

Adobe Acrobat Reader 다운로드

이러한 PDF를 보거나 인쇄하기 위해 Adobe Acrobat Reader가 필요하면 Adobe 웹 사이트 (www.adobe.com/products/acrobat/readstep.html) 에서 사본을 다운로드할 수 있습니다.

IBM Developer Kit for Java 시작하기

IBM Developer Kit for Java(TM)를 처음 사용하는 경우, 다음 단계를 수행하여 간단한 Hello World Java 프로그램을 설치, 구성하고 실행을 연습할 수 있습니다.

1. IBM Developer Kit for Java 정보에 이미 익숙한 경우 최신 제품 갱신사항과 정보로 링크하려면, 새로운 사항을 참조하십시오.
2. IBM Developer Kit for Java를 설치하십시오.
3. 시스템을 구성하십시오.

4. 이 정보를 처음 보는 것이며 IBM Developer Kit for Java를 사용해본 적이 없으면 처음 Hello World Java 프로그램 실행을 참조하십시오. 이 주제는 IBM Developer Kit for Java를 사용하여 간단한 Hello World Java 프로그램을 실행하는 두 가지 방법을 나타냅니다. 이것은 IBM Developer Kit for Java가 올바로 설치되었는지 알 수 있는 편리한 방법입니다.
5. 이제 Hello World Java 프로그램을 작성, 컴파일 및 실행할 준비가 되었습니다. 사용법에 대해서는 Hello World Java 프로그램 작성, 컴파일 및 실행을 참조하십시오.
6. 사용자의 Java 어플리케이션 작성에 관심이 있으면 다음 주제를 읽으십시오.
   - Java 소스 파일 작성 및 편집은 Java 소스 파일을 작성하고 편집할 수 있는 세 가지 방법을 보여줍니다.
   - iSeries 서버에 Java 패키지 다운로드 및 설치는 보다 효율적으로 Java 패키지를 사용할 수 있도록 도와줍니다. 이 항목은 ZIP 파일 처리 및 JAR 파일 처리 뿐만 아니라, GUI(그래픽 사용자 인터페이스), 통합 파일 시스템 및 대소문자 구별의 패키지에 대한 세부사항을 제공합니다.
   - 릴리스 호환성은 한 릴리스에서 다른 릴리스로의 호환성에 대한 내용이 있습니다.

IBM Developer Kit for Java 설치

IBM Developer Kit for Java(TM)를 설치하면 iSeries 서버에서 Java 프로그램을 작성하고 실행할 수 있습니다.

IBM Developer Kit for Java를 설치하려면 다음 단계를 수행하십시오.

1. 명령 행에 GO LICPGM(사용권 프로그램으로 이동) 명령을 입력하십시오.
2. 옵션 11(사용권 프로그램 설치)을 선택하십시오.
3. LP(사용권 프로그램) 5722-JV1 *BASE에 대해 옵션 1(설치)을 선택하고 설치하려는 JDK(Java Development Kit)와 일치하는 옵션을 선택하십시오. 설치하려는 옵션이 리스트에 표시되지 않으면 옵션 필드에 옵션 1(설치)을 입력하여 리스트에 추가할 수 있습니다. 사용권 프로그램 필드에 5722JV1을 입력하고 제품 옵션 필드에 옵션 번호를 입력하십시오.

   주: 둘 이상의 옵션을 한 번에 설치할 수 있습니다.

iSeries 서버에 IBM Developer Kit for Java를 설치했으면 사용자 정의 하도록 선택할 수 있습니다.

IBM Developer Kit for Java 시작에 대한 내용은 처음 Hello World Java 프로그램 실행을 참조하십시오.

사용권 프로그램 복원 명령을 사용한 사용권 프로그램 설치

사용권 프로그램 설치 화면에 나열된 프로그램들은 사용자의 서버가 새로운 서버일 때 LICPGM 설치에서 지원하는 프로그램입니다. 때때로 서버에서 사용권 프로그램으로 나열되지 않은 새로운 프로그램이 사용 가능해집니다. 설치하려는 프로그램이 이러한 경우에 해당하면 RSTLICPGM(사용권 프로그램 복원) 명령을 사용하여 설치해야 합니다.

RSTLICPGM(사용권 프로그램 복원) 명령을 사용하여 사용권 프로그램을 설치하려면 다음의 단계를 따르십시오.

1. 사용권 프로그램이 포함된 테이프나 CD-ROM을 적합한 드라이브에 넣으십시오.

2. iSeries 명령행에서 다음을 입력하십시오.

   RSTLICPGM

   Enter 키를 누르십시오.

   RSTLICPGM(사용권 프로그램 복원) 화면이 나타납니다.
3. *제품* 필드에 설치하려는 사용권 프로그램의 ID 번호를 입력하십시오.
4. *장치* 필드에 설치 장치를 지정하십시오.

   주: 테이프 드라이브에서 설치하는 경우에 장치 ID의 형식은 대개 **TAP*XX***이며 여기서 ***XX***는 **01**과 같은 번호입니다.
5. *사용권 프로그램 복원* 화면에서 다른 매개변수에 대한 디폴트 설정은 그대로 두십시오. **Enter** 키를 누르십시오.
6. 추가 매개변수가 나타납니다. 이 디폴트 설정도 그대로 두십시오. **Enter** 키를 누르십시오. 프로그램은 설치를 시작합니다.

사용권 프로그램의 설치를 완료한 후에 *사용권 프로그램 복원* 화면이 다시 나타납니다.

복수 JDK(Java Development Kits) 지원

iSeries 서버는 복수 JDK(Java Development Kits) 및 J2SDK(Java 2 SDK), 표준판을 지원합니다. iSeries 서버는 복수 JDK의 동시 사용을 지원하지만 복수 JVM(Java Virtual Machine)을 통해서만 지원합니다. 단일 JVM은 하나의 지정된 JDK를 실행합니다.

사용중이거나 사용하려는 JDK를 찾아 설치할 조정 옵션을 선택하십시오. 한 번에 둘 이상의 JDK를 설치할 수 있습니다. java.version 시스템 등록 정보는 실행할 JDK를 판별합니다. JVM이 시작되어 실행되고 있으면 java.version 시스템 등록 정보의 변경은 아무런 효력이 없습니다.

주: V5R2에서, 옵션 1(JDK 1.1.6)과 2(JDK 1.1.7)는 더 이상 사용할 수 없습니다. 설치하거나 사용할 수 없습니다.

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

주: 버전 1.3은 J2SDK(Java 2 SDK), 표준판, 버전 1.3과 동일합니다.

«

예를 들어, 다음은 설치할 옵션의 결과 및 입력할 명령입니다.

설치	입력	결과
옵션 3(1.2)	`java Hello`	J2SDK, 표준판, 버전 1.2 실행.
옵션 4(1.1.8)	`java Hello`	하나의 JDK가 설치되었으며 이것이 디폴트이므로 JDK 1.1.8이 실행됩니다.
옵션 4(1.1.8) 및 옵션 3(1.2)	`java Hello`	가장 높은 번호이기 때문에 J2SDK, 표준판, 버전 1.2가 실행됩니다.
» 4개의 옵션이 모두 설치됩니다.	`java Hello`	J2SDK, 표준판, 버전 1.3이 실행됩니다. «
옵션 3(1.2) 및 옵션 5(1.3)	`java Hello`	가장 높은 번호이기 때문에 J2SDK, 표준판, 버전 1.3이 실행됩니다.
옵션 4(1.1.8) 및 옵션 5(1.3)	`java -Djava.version=1.1.8 Hello`	지정되었기 때문에 JDK 1.1.8이 실행됩니다.
» 옵션 5(1.3) 및 옵션 6(1.4)	`java Hello`	J2SDK, 표준판, 버전 1.3이 실행됩니다. 1.4가 보다 높은 숫자이지만 1.3이 우선입니다. «

주: 하나의 JDK만을 설치하는 경우, 설치한 JDK가 디폴트 JDK입니다.

»

하나 이상의 JDK를 설치한 경우, 다음 우선순위중 하나로 디폴트 JDK가 결정됩니다.

1. 옵션 5(1.3)
2. 옵션 3(1.2)
3. 옵션 6(1.4)
4. 옵션 4(1.1.8)

«

IBM Developer Kit for Java 확장 기능 설치

확장 기능은 핵심 플랫폼의 기능을 확장시키기 위해 사용할 수 있는 Java(TM) 클래스 패키지입니다. 확장 기능은 하나 이상의 ZIP 파일이나 JAR 파일에 패키지되며, 확장 기능 클래스 로더에 의해 JVM(Java Virtual Machine)에 로드됩니다.

확장 메카니즘을 사용하면 JVM(Java Virtual Machine)이 가상 기계가 시스템 클래스를 사용하는 것과 같은 방법으로 확장 클래스를 사용할 수 있습니다. 확장 메카니즘은 또한 확장 기능이 J2SDK, 버전 1.2 이상이나

Java 2 Runtime Environment, 표준판, 버전 1.2 이상에 아직 설치되지 않았을 때 지정된 URL(Uniform Resource Locator)에서 확장 기능을 검색하는 방법도 제공합니다.

»

확장 기능을 위한 일부 JAT 파일이 iSeries 서버와 함께 제공됩니다.

«

이러한 확장 기능 중 하나를 설치하려면 다음의 명령을 입력하십시오.

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

여기서 `extensionToInstall.jar`은 설치하려는 확장 기능이 들어 있는 ZIP 또는 JAR 파일의 이름입니다.

주: IBM이 제공하지 않는 확장 기능의 JAR 파일은 /QIBM/UserData/Java400/ext 디렉토리에 넣을 수 있습니다.

/QIBM/UserData/Java400/ext 디렉토리에서 확장 기능에 대한 링크를 작성하거나 파일을 추가하면 확장 기능 클래스 로더가 탐색하는 파일의 리스트가 *iSeries 서버에서 실행 중인 모든 JVM(Java Virtual Machine)*에 대해 변경됩니다. iSeries 서버의 다른 JVM(Java Virtual Machine)에 대한 확장 기능 클래스 로더에 영향을 주지 않으면서 확장 기능에 대한 링크를 작성하거나 iSeries 서버과 함께 IBM이 제공하지 않는 확장 기능을 설치하려면 다음의 단계를 따르십시오.

1. 확장 기능을 설치할 디렉토리를 작성하십시오.
   iSeries 명령행에서 MKDIR(디렉토리 작성) 명령을 사용하거나 Qshell 인터프리터에서 `mkdir` 명령을 사용하십시오.
2. 작성된 디렉토리에 확장 기능 JAR 파일을 저장하십시오.
3. java.ext.dirs 등록 정보에 신규 디렉토리를 추가하십시오.
   iSeries 명령행에서 JAVA 명령의 PROP 필드를 사용하여 java.ext.dirs 등록 정보에 새로운 디렉토리를 추가할 수 있습니다.

»

새로운 디렉토리의 이름이 /home/username/ext이고, 확장 파일 이름은 extensionToInstall.jar이고, Java 프로그램의 이름이 Hello이면, 입력하는 명령은 다음과 비슷합니다.

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

CPY OBJ('/productA/extensionToInstall.jar') TODIR('/home/username/ext') or
copy the file to /home/username/ext using FTP (file transfer protocol).

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

iSeries 서버에서 Java 패키지 다운로드 및 설치

iSeries 서버에서 보다 효과적으로 Java(TM) 패키지를 다운로드, 설치하고 사용하려면 다음을 참조하십시오.

- 그래픽 사용자 인터페이스가 있는 패키지(10 페이지 참조)
- 대소문자 구분 및 통합 파일 시스템(10 페이지 참조)
- ZIP 파일 처리 및 JAR 파일 처리(10 페이지 참조)
- Java 확장 구조(11 페이지 참조)

그래픽 사용자 인터페이스가 있는 패키지

그래픽 사용자 인터페이스가 사용된 Java 프로그램은 그래픽 화면 기능이 있는 표시 장치를 사용해야 합니다. 예를 들면, 퍼스널 컴퓨터, 기술적인 워크스테이션 또는 네트워크 컴퓨터를 사용할 수 있습니다. iSeries 서버는 Remote Abstract Window Toolkit(AWT) 기능을 제공합니다. 이 기능은 적절한 TCP/IP 연결 화면에서 전체 범위의 그래픽 기능을 사용하여 iSeries 서버에서 어플리케이션을 실행합니다. 특정 설치, 설정 및 전반적인 사용에 대한 내용은 Remote Abstract Window Toolkit 설정을 참조하십시오.

대소문자 구분 및 통합 파일 시스템

통합 파일 시스템은 대소문자를 구분하는데 파일명은 대소문자를 구분하지 않는 파일 시스템을 제공합니다. QOpenSys는 통합 파일 시스템 내의 대소문자를 구분하는 파일 시스템의 예입니다. 루트, '/'는 대소문자를 구분하지 않는 파일 시스템의 예입니다. 통합 파일 시스템에 대한 자세한 정보는 통합 파일 시스템 주제에서 파일 시스템 정보를 참조하십시오.

JAR 또는 클래스는 대소문자를 구분하지 않는 파일 시스템에서 찾을 수 있지만 Java는 여전히 대소문자 구분 언어입니다. `wrklnk '/home/Hello.class'`와 `wrklnk '/home/hello.class'`는 동일한 결과를 산출하지만 `JAVA CLASS(Hello)`와 `JAVA CLASS(hello)`는 다른 클래스를 호출합니다.

ZIP 파일 처리 및 JAR 파일 처리

ZIP 파일과 JAR 파일에는 Java 클래스 세트가 있습니다. 이들 파일 중 하나에 대해 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하면 클래스를 확인하여 내부 기계 양식으로 변환한 다음 지정된 경우 iSeries 기계 코드로 변환됩니다. ZIP 파일과 JAR 파일을 다른 개별 클래스 파일과 같이 취급할 수 있습니다. 내부 기계 양식이 해당 파일에 연관되는 경우 파일과 연관된 채 남아있습니다. 내부 기계 양식은 성능 개선을 위해 이후 실행시에 클래스 파일 대신에 사용됩니다. 현재 Java 프로그램이 클래스 파일이나 JAR 파일과 연관되는지 확실하지 않은 경우 DSPJVAPGM(Java 프로그램 표시) 명령을 사용하여 iSeries 서버의 Java 프로그램에 대한 정보를 표시하십시오.

IBM Developer Kit for Java의 이전 릴리스에서는 JAR 파일이나 ZIP 파일을 변경한 경우 접속된 Java 프로그램이 사용 불가능하게 되기 때문에 Java 프로그램을 다시 작성했어야 합니다. 이제는 더 이상 그럴 필요

가 없습니다. 많은 경우에서 JAR 파일이나 ZIP 파일을 변경하더라도 Java 프로그램은 여전히 유효하고 다시 작성할 필요가 없습니다. 단일 클래스 파일이 JAR 파일에서 갱신될 때처럼 부분적으로 변경하는 경우 해당 JAR 파일에서 영향을 받는 클래스 파일만 다시 작성하면 됩니다.

Java 프로그램은 JAR 파일에 대한 가장 일반적인 변경 후에도 JAR 파일에 연결되어 있습니다. 예를 들어, Java 프로그램은 다음과 같은 경우에 JAR 파일에 연결되어 있습니다.

- 452 페이지의 『Java ajar 툴』을 사용하여 JAR 파일을 변경하거나 재작성.
- 454 페이지의 『Java jar 툴』을 사용하여 JAR 파일을 변경하거나 재작성.
- 0S/400 COPY 명령 또는 Qshell `cp` 유틸리티를 사용하여 JAR 파일을 대체하는 경우.

Windows용 iSeries Access 또는 PC(퍼스널 컴퓨터)에 맵핑된 드라이브에서 통합 파일 시스템의 JAR 파일에 액세스할 때 Java 프로그램은 다음과 같은 경우에 JAR 파일에 연결되어 있습니다.

- 기존의 통합 파일 시스템 JAR 파일로 다른 JAR 파일을 끌어서 놓는 경우.
- 454 페이지의 『Java jar 툴』을 사용하여 통합 파일 시스템 JAR 파일을 변경 또는 재작성하는 경우.
- PC 복사 명령을 사용하여 통합 파일 시스템 JAR 파일을 대체하는 경우.

JAR 파일이 변경되거나 대체될 때 해당 파일에 접속되는 Java 프로그램은 더 이상 현재의 것이 아닙니다.

Java 프로그램이 JAR 파일에 연결되지 않는 한 가지 예외가 있습니다. 연결된 Java 프로그램이 FTP(파일 전송 프로토콜)를 사용하여 JAR 파일을 대체하는 경우 손상을 받습니다. 예를 들면, FTP `put` 명령을 사용하여 JAR 파일을 대체하는 경우입니다.

JAR 파일의 성능 등록 정보에 대한 자세한 내용은 Java 런타임 성능을 참조하십시오.

Java 확장 구조

Java 2 SDK, 표준판, 버전 1.2 이상에서 부가 제품은 코어 플랫폼의 기능성을 확장하기 위해 사용할 수 있는 Java 클래스의 패키지입니다. 부가 제품이나 어플리케이션은 하나 이상의 JAR 파일에 패키지됩니다. 확장 메카니즘을 사용하면 JVM(Java Virtual Machine)이 가상 기계가 시스템 클래스를 사용하는 것과 같은 방법으로 확장 클래스를 사용할 수 있습니다. 확장 메카니즘은 또한 부가 제품이 JDK(Java Development Kit) 또는 Java 2 Runtime Environment, 표준판에 아직 설치되지 않았을 때 지정된 URL에서 부가 제품을 검색하는 방법을 제공합니다.

부가 제품 설치에 대한 정보는 IBM Developer Kit for Java 부가 제품 설치를 참조하십시오.

Hello World Java 프로그램 실행

Hello World Java(TM) 프로그램을 호출하여 다음 방법 중 하나로 실행할 수 있습니다.

1. 간단하게 IBM Developer Kit for Java와 함께 제공된 Hello World Java 프로그램을 실행할 수 있습니다.

   프로그램을 실행하려면 다음 단계를 수행하십시오.

a. GO LICPGM(사용권 프로그램으로 이동) 명령을 입력하여 IBM Developer Kit for Java가 설치되어 있는지 확인하십시오. 그런 다음, 옵션 10(설치된 사용권 프로그램 표시)을 선택하십시오. 사용권 프로그램 5722-JV1 *BASE 및 옵션 중 최소한 하나가 설치된 것으로 나열되는지 확인하십시오.

b. iSeries 기본 메뉴 명령행에 `java Hello`를 입력하십시오. Enter 키를 눌러 Hello World Java 프로그램을 실행하십시오.

c. IBM Developer Kit for Java가 올바로 설치된 경우 Java 쉘화면에 `Hello World`가 나타납니다. 명령 입력 화면으로 리턴하려면 F3(나감) 또는 F12(나감) 키를 누르십시오.

d. Hello World 클래스가 실행되지 않으면 설치가 성공적으로 완료되었는지 확인하거나 서비스 정보에 대해 IBM Developer Kit for Java에 대한 지원 확보를 참조하십시오.

2. 사용자 Hello Java 프로그램을 실행할 수도 있습니다. 사용자 Hello Java 프로그램을 작성하는 방법에 대한 자세한 내용은 Hello World Java 프로그램 작성, 컴파일 및 실행을 참조하십시오.

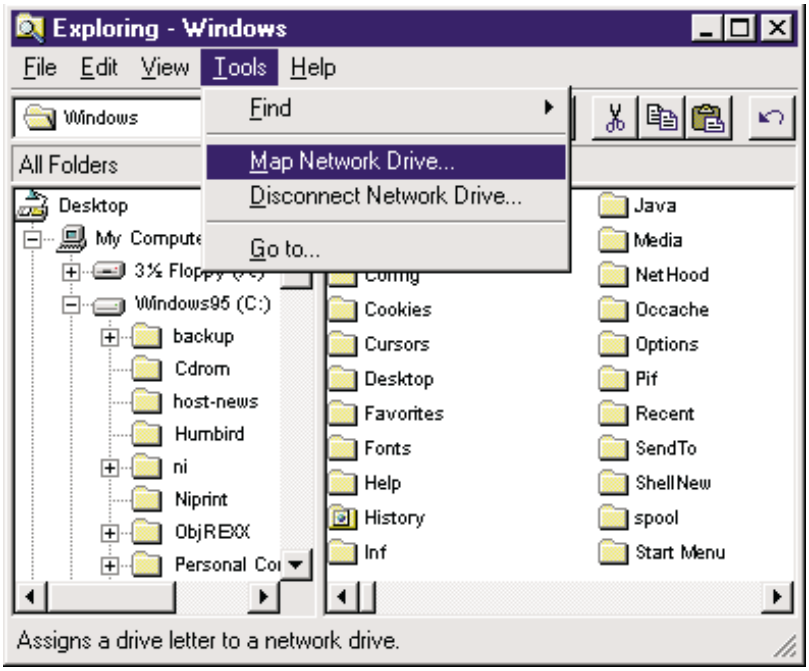
iSeries 서버에 네트워크 드라이브 맵핑

네트워크 드라이브를 iSeries 서버에 맵핑하려면 서버와 워크스테이션에 Windows용 iSeries Access를 설치했는지 확인하십시오. Windows용 iSeries Access를 설치하고 구성하는 방법에 대한 자세한 정보는 Windows용 iSeries Access 설치를 참조하십시오.

네트워크 드라이브를 맵핑하려면 먼저 iSeries 서버를 위해 구성된 연결이 있어야 합니다.

네트워크 드라이브를 맵핑시키려면 이 단계를 따르십시오.

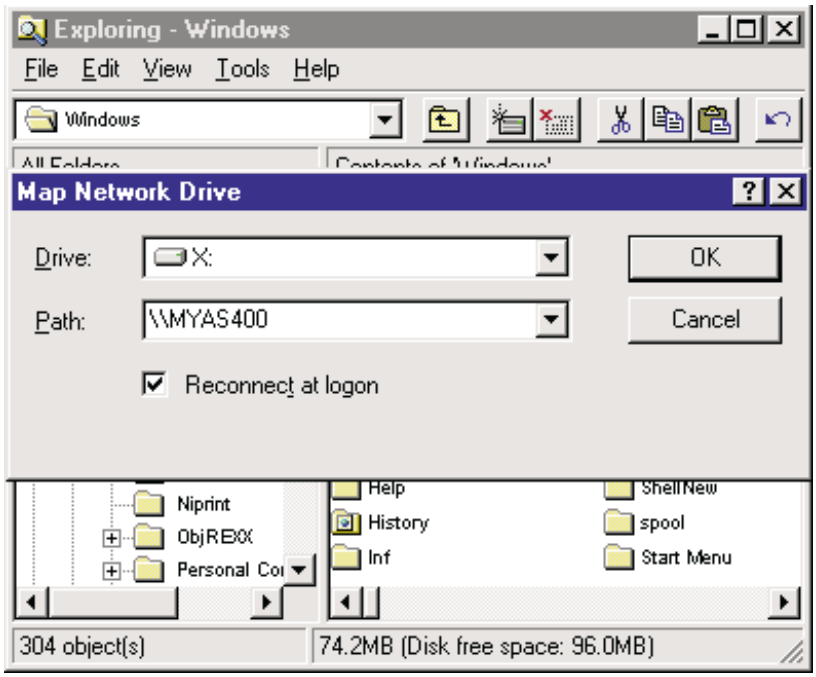
1. Windows(R) 탐색기를 여십시오.
   a. Windows 작업바에서 **시작** 버튼을 마우스 오른쪽 버튼으로 클릭하십시오.
   b. 메뉴에서 **탐색**을 클릭하십시오.
2. 툴 메뉴에서 **네트워크 드라이브 맵핑**을 선택하십시오.



3. iSeries 서버에 연결하는데 사용하려는 드라이브를 선택하십시오.
4. 서버에 경로명을 입력하십시오. 예를 들면 다음과 같습니다.

   \\MYSERVER

   여기서 ***MYSERVER***는 iSeries 서버의 이름입니다.



5. **로그온 시 재연결** 상자가 비어 있으면 선택하십시오.

6. 완료하기 위해 **확인**을 클릭하십시오.

맵핑시킨 드라이브는 이제 Windows 탐색기의 **모든 폴더** 섹션에 나타납니다.

iSeries 서버에서 디렉토리 작성

iSeries 서버에서 Java(TM) 어플리케이션을 저장할 수 있는 디렉토리를 작성해야 합니다. 이를 실행하는데는 두 가지 방법이 있습니다.

- iSeries Navigator를 사용하여 디렉토리 작성
  Windows용 iSeries Access를 설치했으면 이 옵션을 선택하십시오. iSeries Navigator를 사용하여 Java 프로그램을 컴파일, 최적화 및 실행할 예정이면 이 옵션을 선택하여 프로그램이 이러한 조작을 수행할 수 있도록 올바른 위치에 저장되었는지 확인해야 합니다.
- 명령 항목 행을 사용하여 디렉토리 작성
  Windows용 iSeries Access를 설치하지 않았으면 이 옵션을 선택하십시오.

설치 정보를 비롯하여 iSeries Navigator에 대한 정보는 iSeries Navigator 시작하기를 참조하십시오.

명령 항목 행을 사용하여 디렉토리 작성

iSeries 서버에서 디렉토리를 작성하려면 다음의 단계를 따르십시오.

1. iSeries 서버에 사인 온하십시오.
2. 명령행에서 다음을 입력하십시오.

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

   여기서 *mydir*은 작성 중인 디렉토리의 이름입니다.

   Enter 키를 누르십시오.

   "**디렉토리가 작성되었습니다.**"라는 메세지가 화면의 맨 아래에 나타납니다.

iSeries Navigator를 사용하여 디렉토리 작성

iSeries 서버에서 디렉토리를 작성하려면 이 단계를 따르십시오.

1. iSeries Navigator를 여십시오.
2. **사용자 연결** 창에서 서버의 이름을 더블 클릭하여 사인 온하십시오.
   사용자 연결 창에 사용자의 서버가 없으면 다음의 단계에 따라 추가하십시오.

a. **파일 —> 연결 추가....**를 클릭하십시오.

b. **시스템** 필드에 사용자 서버의 이름을 입력하십시오.

c. **다음**을 클릭하십시오.

d. 아직 입력하지 않았으면 **디폴트 사용자 ID 사용, 필요에 따라 프롬트 표시** 필드에 사용자 ID를 입력하십시오.

e. **다음**을 클릭하십시오.

f. **연결 확인**을 클릭하십시오. 그러면 서버에 연결할 수 있음이 확실해집니다.

g. **완료**를 클릭하십시오.

3. 사용하려는 연결 아래의 폴더를 여십시오. 이름이 **파일 시스템**인 폴더를 찾으십시오. 이 폴더를 찾을 수 없으면 iSeries Navigator 설치 중 파일 시스템을 설치하는 옵션을 선택하지 않은 것입니다. **시작 —> 프로그램 —> Windows용 iSeries Access —> 선택적 설치**를 선택하여 iSeries Navigator의 파일 시스템 옵션을 설치해야 합니다.
4. **파일 시스템** 폴더를 열고 **통합 파일 시스템** 폴더를 찾으십시오.
5. **통합 파일 시스템** 폴더를 열고 **루트** 폴더를 여십시오. **루트** 폴더를 열면 iSeries 명령행에서 WRKLNK ('/') 명령을 수행할 때와 동일한 구조를 볼 수 있습니다.
6. 서브디렉토리를 추가하려는 폴더를 마우스 오른쪽 버튼으로 클릭하십시오. **신규 폴더**를 선택하고 작성할 서브디렉토리의 이름을 입력하십시오.

HelloWorld Java 프로그램 작성, 컴파일 및 실행

간단한 Hello World Java(TM) 프로그램을 작성하는 것은 IBM Developer Kit for Java에 익숙해질 때 시작하는 매우 적합한 작업입니다.

Hello World Java 프로그램을 작성, 컴파일 및 실행하려면 다음 단계를 수행하십시오.

1. 
   iSeries 서버에 네트워크 드라이브를 맵핑하십시오.
2. Java 어플리케이션에 대한 디렉토리를 iSeries 서버에서 작성하십시오.

3. 통합 파일 시스템에 ASCII(미국 표준 정보 교환 코드) 텍스트 파일로 소스 파일을 작성하십시오.

IDE(integrated development environment) 제품이나 Windows(R) 메모장과 같은 텍스트 기반 편집기를 사용하여 Java 어플리케이션을 코딩할 수 있습니다.

a. 텍스트 파일을 HelloWorld.java로 명명하십시오. 파일 작성 및 편집에 대한 자세한 정보는 Java 소스 파일 작성 및 편집을 참조하십시오.

b. 다음 소스 코드가 파일에 들어 있는지 확인하십시오.

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

4. 소스 파일을 컴파일하십시오.

   a. WRKENVVAR(환경 변수에 대한 작업) 명령을 입력하여 CLASSPATH 환경 변수를 체크하십시오. CLASSPATH 변수가 없으면 변수를 추가한 후 '.'(현재 디렉토리)로 설정하십시오. CLASSPATH 변수가 있으면 '.'가 경로명 리스트의 시작 부분에 있는지 확인하십시오. CLASSPATH 환경 변수에 관한 상세한 정보는 Java classpath를 참조하십시오.

   b. STRQSH(Qshell 시작) 명령을 입력하여 Qshell 인터프리터를 시작하십시오.

   c. cd(디렉토리 변경) 명령을 사용하여 현재 디렉토리를 HelloWorld.java 파일이 있는 통합 파일 시스템 디렉토리로 변경하십시오.

   d. 디스크에 저장한 파일명 앞에 javac를 입력하십시오. 예를 들어, javac HelloWorld.java를 입력하십시오.

5. 

   통합 파일 시스템의 클래스 파일에 파일 권한을 설정하십시오.

6. Java 어플리케이션을 최적화하십시오.

   a. *QSH 명령 입력* 행에서 다음을 입력하십시오.

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

      여기서 *mydir*은 Java 어플리케이션이 저장된 디렉토리의 경로 이름이고 *myclass*는 컴파일된 Java 어플리케이션의 이름입니다.

      주: 최대 40으로 최적화 레벨을 지정할 수 있습니다. 최적화 레벨 40은 Java 어플리케이션의 효율성을 증가시키지만 디버그 기능을 제한하기도 합니다. Java 어플리케이션을 개발하는 이전 단계에서는 최적화 레벨을 20으로 설정하여 보다 쉽게 어플리케이션을 디버그할 수 있습니다. 자세한 정보는 CRTJVAPGM 명령 및 OPTIMIZE 매개변수를 참조하십시오.

   b. **Enter** 키를 누르십시오.

      Java 프로그램이 사용자의 클래스에 대해 작성되었음을 나타내는 메세지가 표시됩니다.

7. 클래스 파일을 실행하십시오.

   a. Java classpath가 올바르게 설정되었는지 확인하십시오.

b. JVM(Java Virtual Machine)에서 HelloWorld.class를 실행하려면 Qshell 명령행에서 java를 입력하고 그 뒤에 HelloWorld를 입력하십시오. 예를 들어, java HelloWorld를 입력할 수 있습니다. 또한 Java 실행(RUNJVA) 명령을 iSeries 서버에서 사용하여 HelloWorld.class를 실행할 수 있습니다.

c. 모두가 제대로 입력되면 "Hello World"가 사용자 화면에 인쇄됩니다.

Qshell이 다른 명령을 위해 준비되었음을 나타내는 쉘 프롬트(디폴트는 $)가 나타납니다.

d. 명령 입력 화면으로 리턴하려면 F3(나감) 또는 F12(단절) 키를 누르십시오.

iSeries 서버에서 타스크를 수행하기 위한 그래픽 사용자 인터페이스인 iSeries Navigator를 사용하여 Java 어플리케이션을 쉽게 컴파일, 최적화 및 실행할 수도 있습니다. 지침은 iSeries Navigator를 사용한 Java 어플리케이션에 대한 작업을 참조하십시오. 설치 정보를 비롯하여 iSeries Navigator에 대한 자세한 정보는 iSeries Navigator 시작하기를 참조하십시오.

Java 소스 파일 작성 및 편집

다음과 같은 여러 방법으로 Java(TM) 소스 파일을 작성하고 편집할 수 있습니다.

- 『Windows용 iSeries Access』.
- 『워크스테이션』.
- 18 페이지의 『EDTF』.
- 18 페이지의 『소스 입력 유틸리티(SEU)』.

Windows용 iSeries Access

Java 소스 파일은 iSeries 서버의 통합 파일 시스템에 있는 미국 표준 정보 교환 코드(ASCII) 텍스트 파일입니다.

Windows용 iSeries Access와 워크스테이션 기반 편집기로 Java 소스 파일을 작성하고 편집할 수 있습니다.

워크스테이션

워크스테이션에서 Java 소스 파일을 작성할 수 있습니다. 그런 다음, FTP(파일 전송 프로토콜)를 사용하여 파일을 통합 파일 시스템으로 전송하십시오.

워크스테이션에서 Java 소스 파일을 작성하고 편집하려면 다음과 같이 하십시오.

1. 선택한 편집기를 사용하여 워크스테이션에서 ASCII 파일을 작성하십시오.

2. FTP로 iSeries 서버에 연결하십시오.
3. 통합 파일 시스템 내 2진 파일 디렉토리에 소스 파일을 전송해서 파일을 ASCII 형식으로 남겨놓으십시오.

EDTF

EDTF CL 명령을 사용하여 어느 파일 시스템에서나 파일을 편집할 수 있습니다. 이것은 스트림 파일이나 데이터베이스 파일을 편집하기 위한 소스 입력 유틸리티(SEU)와 유사한 편집기입니다. 자세한 정보는 EDTF CL 명령을 참조하십시오.

소스 입력 유틸리티(SEU)

SEU를 사용하여 Java 소스 파일을 텍스트 파일로 작성할 수 있습니다.

SEU를 사용하여 Java 소스 파일을 텍스트 파일로 작성하려면 다음 단계를 수행하십시오.

1. SEU를 사용하여 소스 파일 멤버를 작성하십시오.
2. CPYTOSTMF(스트림 파일로 복사) 명령을 사용하여 자료를 ASCII로 변환하면서 소스 파일 멤버를 통합 파일 시스템 스트림 파일에 복사하십시오.

소스 코드를 변경해야 할 경우 SEU를 사용하여 데이터베이스 멤버를 변경하고 파일을 다시 복사하십시오.

파일 저장에 대한 내용은 통합 파일 시스템의 파일을 참조하십시오.

iSeries Navigator를 사용한 Java 어플리케이션에 대한 작업

iSeries Navigator에서는 간단한 포인트 및 클릭을 통해 Java(TM) 어플리케이션을 컴파일, 최적화하고 실행할 수 있습니다.

설정 요구사항

iSeries Navigator를 사용하여 Java 어플리케이션에 대해 작업하려면 다음의 조건이 충족되었는지 확인하십시오.

- Windows용 iSeries Access의 일부인 iSeries Navigator를 워크스테이션에 설치해야 합니다. Windows용 iSeries Access를 아직 설치하지 않았으면 iSeries Navigator 시작하기에서 다운로드 정보를 참조하십시오.
- iSeries 서버의 특정 디렉토리에 Java 어플리케이션을 저장해야 합니다. Java 어플리케이션을 저장하기 위한 올바른 경로는 다음과 같습니다.

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

  여기서 **myserver**는 iSeries 서버의 이름이고 **mydir**은 Java 어플리케이션을 저장한 디렉토리의 이름입니다. Java 어플리케이션을 저장할 디렉토리 작성에 대한 자세한 내용은 iSeries Navigator를 사용한 디렉토리 작성을 참조하십시오.

iSeries Navigator를 사용하여 Java 어플리케이션 컴파일

Java 어플리케이션을 컴파일하려면 다음 단계를 따르십시오.

1. **myfile.java** 오른쪽을 클릭하십시오. 여기서 **myfile**은 Java 어플리케이션의 이름입니다.
2. **Java 파일 컴파일**을 선택하십시오.

3. 새 창에서 JDK 버전을 선택합니다. 이 안내서에 제공된 지침에 따라 클래스 경로를 이미 설정했으면 이 창에서 클래스 경로를 지정할 필요가 없습니다.
4. **확인**을 클릭하십시오.

프로그램에서 오류가 발견되었으면 오류 리스트가 있는 창이 열립니다. 그렇지 않으면 **하나의 Java 파일 중 하나를 컴파일했습니다.** 메세지가 iSeries Navigator 창의 맨 아래에 나타납니다. **myfile.class**라고 하는 새로운 파일을 작성합니다.

iSeries Navigator를 사용하여 Java 어플리케이션 최적화 및 실행

Java 어플리케이션을 최적화하려면 다음 단계를 따르십시오.

1. **myfile.class**를 마우스 오른쪽 버튼으로 클릭하십시오.
2. 

   연관된 Java 프로그램을 선택한 후 **실행...**을 클릭하여 Java 프로그램을 실행하십시오.

   
3. **확장**을 클릭하십시오. 요구하는 최적화 레벨을 선택하십시오.

   주: 최대 40으로 최적화 레벨을 지정할 수 있습니다. 최적화 레벨 40은 Java 어플리케이션의 효율성을 증가시키지만 디버그 기능을 제한하기도 합니다. Java 어플리케이션을 개발하는 이전 단계에서는 최적화 레벨을 20으로 설정하여 보다 쉽게 어플리케이션을 디버그할 수 있습니다. 최적화에 대한 자세한 정보는 최적화 레벨을 참조하십시오. just-in-time 컴파일러를 나타내는 JIT라고 하는 옵션도 있습니다. JIT는 필요할 때 코드를 컴파일하므로 직접 처리보다 효율적입니다. JIT에 대한 자세한 정보는 Java 프로그램을 실행할 때 사용할 모드 선택을 참조하십시오.
4. **확인**을 클릭하여 확장 옵션 창을 닫으십시오.
5. **확인**을 클릭하여 Java 프로그램을 실행하십시오.

프로그램의 출력을 새로운 창에 표시합니다. 프로그램이 실행을 완료했으면 **Java 프로그램 완료** 메세지가 나타납니다.

iSeries Navigator 기능에 대한 자세한 정보는 iSeries Navigator 화면에서 **도움말** 메뉴를 참조하십시오.

IBM Developer Kit for Java에 대한 iSeries 서버 사용자 정의

일단 iSeries 서버에 IBM Developer Kit for Java(TM)를 설치했으면, 사용자 서버를 사용자 정의하도록 선택할 수 있습니다.

시간대 구성

Java 프로그램이 시간에 대해 민감한 경우, 시간대 구성이 필요할 수 있습니다.

QUTCOFFSET(조정된 범용 시간 오프셋) 시스템 값이 디폴트(+00:00)로 설정된 경우, Java는 현재 시간으로 iSeries 400 시간을 사용합니다. user.timezone Java 시스템 등록 정보 설정의 디폴트 값은 UTC입니다.

다음 경우에는 QUTCOFFSET 시스템 값과 갱신 로케일이 필요합니다.

- QUTCOFFSET이 디폴트 값으로 설정되지 않았기 때문에 시간대에 대해 민감한 경우.
- Java 코드에서 user.timezone 시스템 등록 정보가 UTC가 아닌 다른 값으로 디폴트 처리되기를 기대하는 경우.
- `java` 명령 실행시 user.timezone Java 시스템 등록 정보를 지정하는 경우.

로케일의 LC_TOD 범주에는 사용자의 시간대와 일치하는 같은 값으로 설정해야 하는 tname 필드가 포함됩니다. 로케일을 작성하고 tname 필드를 형식화하는 방법에 대한 자세한 내용은 OS/400 국제화를 참조하십시오.

시스템 등록 정보 구성

Java 시스템 등록 정보는 Java 프로그램이 실행되는 환경을 판별합니다. 이 등록 정보는 OS/400의 시스템 값 또는 환경 변수와 같습니다. JVM(Java Virtual Machine)을 시작할 때 여러 개의 등록 정보가 설정됩니다. 다음 단계 중 하나를 수행하여 시스템 등록 정보 디폴트 값을 사용하기 위해 선택하거나 자신만의 디폴트 등록 정보 값 세트를 지정할 수 있습니다.

1. /QIBM/UserData/Java400의 파일, SystemDefault.properties를 제공합니다. 이 파일에 지정한 등록 정보 값을 IBM이 제공하는 시스템 디폴트 값으로 대체합니다. 이 파일은 iSeries 서버 시스템에서 실행되는 모든 JVM(Java Virtual Machine)에 대한 디폴트 시스템 등록 정보를 설정합니다.
2. 또는 user.home 경로에 SystemDefault.properties 파일을 넣습니다. 포함된 파일 및 등록 정보는 /QIBM/UserData/Java400/SystemDefault.properties의 값으로 대체합니다. 이 파일은 실행하는 모든 JVM에 대한 디폴트 시스템 등록 정보를 설정합니다.

IBM Developer Kit for Java 시작에 대한 내용은 처음 Hello World Java 프로그램 실행을 참조하십시오.

Java classpath

Java(TM) 가상 기계는 런타임 동안 Java clsaapath를 사용하여 클래스를 찾습니다. Java 명령과 툴(tool)도 클래스를 찾기 위해 classpath를 사용합니다. 디폴트 시스템 classpath, CLASSPATH 환경 변수 및 classpath 명령 매개변수 모두가 특정 클래스를 찾을 때 탐색할 디렉토리를 판별합니다.

주: J2SDK(Java 2 Software Development Kit), 표준판, 버전 1.2 이상에서는 로드될 부가 제품에 대한 classpath를 java.ext.dirs 등록 정보가 판별합니다. 자세한 정보는 IBM Developer Kit for Java 부가 제품 설치를 참조하십시오.

디폴트 시스템 classpath는 시스템이 정의하며 사용자가 변경할 수 없습니다. iSeries 서버에서는 디폴트 classpath가 IBM Developer Kit, Remote Abstract Window Toolkit(AWT) 및 다른 시스템 클래스의 일부인 클래스를 찾을 곳을 지정합니다.

시스템의 기타 모든 클래스를 찾으려면 CLASSPATH 환경 변수 또는 classpath 매개변수를 사용하여 탐색할 classpath를 지정해야 합니다. 툴 또는 명령에서 사용되는 classpath 매개변수가 CLASSPATH 환경 변수에 지정되는 값을 대체합니다.

WRKENVVAR(환경 변수에 대한 작업) 명령을 사용하여 CLASSPATH 환경 변수에 대한 작업을 할 수 있습니다. WRKENVVAR 화면에서 CLASSPATH 환경 변수를 추가하거나 변경할 수 있습니다. ADDENVVAR(환경 변수 추가) 또는 CHGENVVAR(환경 변수 변경) 명령은 CLASSPATH 환경 변수를 추가하거나 변경합니다.

CLASSPATH 환경 변수 값은 콜론(:)으로 분리되는 경로명 리스트로서 특정 클래스를 찾기 위해 탐색됩니다. 경로명은 0(zero) 이상의 디렉토리명의 순서입니다. 이 디렉토리명 다음에 통합 파일 시스템에서 탐색되는 디렉토리명, ZIP 파일 또는 JAR 파일이 옵니다. 경로명의 구성요소는 슬래시(/) 문자로 분리됩니다. 현재 작업 디렉토리를 지정하려면 마침표(.)를 사용하십시오.

Qshell 인터프리터를 사용하여 이루어지는 내보내기 유틸리티를 사용하여 Qshell 환경에 CLASSPATH 변수를 설정할 수 있습니다.

이 명령들은 CLASSPATH 변수를 Qshell 환경에 추가하며 "`.:/myclasses.zip:/Product/classes`" 값으로 설정합니다.

- 다음 명령은 Qshell 환경에서 CLASSPATH 변수를 설정합니다.

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

- 이 명령은 CLASSPATH 변수를 명령 행에서 설정합니다.

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

J2SDK는 JDK 1.1.x와 달리 클래스를 탐색합니다. JDK 1.1.x는 시스템 classpath를 먼저 탐색하고 후에 모든 사용자 지정 classpaths를 탐색합니다. J2SDK는 부트스트랩 classpath를 먼저 탐색하고 다시 확장 디렉토리를 탐색한 후 classpath를 탐색합니다.

따라서 앞의 코드 예를 사용하는 JDK 1.1.x에 대한 탐색 순서는 다음과 같습니다.

1. 디폴트 시스템 classpath
2. 현재 작업 중인 디렉토리
3. "루트"(/) 파일 시스템에 위치한 myclasses.zip 파일
4. "루트"(/) 파일 시스템의 제품 디렉토리에 있는 클래스 디렉토리

앞의 코드 예를 사용하는 J2SDK에 대한 탐색 순서는 다음과 같습니다.

1. sun.boot.class.path 등록 정보 내의 부트스트랩 classpath
2. java.ext.dirs 등록 정보 내의 확장 디렉토리
3. 현재 작업 중인 디렉토리
4. "루트"(/) 파일 시스템에 위치한 myclasses.zip 파일
5. "루트"(/) 파일 시스템의 제품 디렉토리에 있는 클래스 디렉토리

Qshell 환경에 입력될 때 CLASSPATH 변수는 환경 변수로 설정됩니다. classpath 매개변수는 경로명 리스트를 지정합니다. 그것은 CLASSPATH 환경 변수와 같은 구문을 갖습니다. classpath 매개변수는 다음과 같은 툴 및 명령에서 사용할 수 있습니다.

- Qshell의 java 명령
- javac 툴
- javah 툴
- javap 툴
- javadoc 툴
- rmic 툴
- RUNJVA(Java 실행) 명령

명령에 대한 자세한 정보는 IBM Developer Kit for Java용 명령 및 툴을 참조하십시오. 이 명령 또는 툴 중 하나와 함께 classpath 매개변수를 사용하는 경우 CLASSPATH 환경 변수는 무시됩니다.

CLASSPATH 환경 변수를 java.class.path 등록 정보를 사용하여 대체할 수 있습니다. java.class.path 등록 정보 뿐만 아니라 다른 등록 정보도 SystemDefault.properties 파일을 사용해 변경할 수 있습니다. SystemDefault.properties 파일 내의 값이 CLASSPATH 환경 변수를 대체합니다. SystemDefault.properties 파일에 대한 자세한 정보는 SystemDefault.properties 파일을 참조하십시오.

JDK 1.1.x에서 os400.class.path.system 등록 정보 또한 클래스를 찾을 때 탐색되는 사항에 영향을 미칩니다. 이 등록 정보는 PRE, POST 또는 NONE의 세 값 중 하나를 가질 수 있습니다. 이 등록 정보를 PRE로 설정하면 디폴트 시스템 classpath가 해당 경로보다 먼저 탐색되는 것이 디폴트입니다. 이 경로는 CLASSPATH 환경 변수 또는 classpath 매개변수에 의해 지정됩니다. os400.class.path.system 등록 정보를 POST로 설정하는 경우 디폴트 시스템 classpath는 모든 사용자 지정 classpath 다음에 탐색됩니다. NONE 값이 사용되는 경우 디폴트 classpath는 전혀 탐색하지 않으며 사용자 지정 classpath만 탐색합니다.

J2SDK에서 -Xbootclasspath 옵션을 동일한 효과로 사용할 수 있습니다. -Xbootclasspath/a:*path*는 기본 부트스트랩 클래스 경로에 *path*를 추가하며 /p:*path*는 부트스트랩 클래스 경로에 *path*를 추가하고 /r:*path*는 부트스트랩 클래스 경로를 *path*로 대체합니다.

주: 시스템 클래스가 발견되지 않거나 사용자 정의 클래스에 의해 올바르지 않게 대체되는 경우 예측할 수 없는 결과가 발생할 수 있으므로 NONE 또는 POST를 지정할 때 주의하십시오. 따라서 모든 사용자 지정 classpath 이전에 시스템 디폴트 classpath를 탐색할 수 있게 해야 합니다.

Java 프로그램 실행 환경을 판별하는 방법에 대한 내용은 Java 시스템 등록 정보를 참조하십시오.

자세한 내용은 프로그램 및 CL Command API 또는 통합 파일 시스템을 참조하십시오.

Java 시스템 등록 정보

Java(TM) 시스템 등록 정보에서는 Java 프로그램이 실행할 환경을 판별합니다. 이것은 OS/400(R)에 있는 시스템 값이나 환경 변수와 비슷합니다. JVM(Java Virtual Machine)을 시작할 때 여러 개의 등록 정보가 설정됩니다.

이 릴리스가 지원하는 시스템 등록 정보를 보려면 상세한 정보를 위해 선택한 버전에 링크하십시오.

- JDK(Java Development Kit) 1.1.8
- Java 2 Software Development Kit(J2SDK), 표준판

버전 1.2, 1.3 및 1.4.

SystemDefault.properties 파일

SystemDefault.properties 파일은 표준 Java(TM) 등록 정보 파일입니다. SystemDefault.properties 파일에 디폴트 등록 정보를 지정할 수 있습니다. 홈 디렉토리에 있는 SystemDefault.properties 파일은 /QIBM/UserData/Java400 디렉토리에 있는 SystemDefault.properties 파일에 대해 우선순위를 갖습니다. 홈 디렉토리에 있는 SystemDefault.properties 파일에 설정된 등록 정보는 시작한 JVM이나 등록 정보 user.home = /YourUserHome/을 지정한 사용자를 위해서만 사용됩니다.

임의의 Java 등록 정보 파일에서 수행하는 것처럼 SystemDefault.properties 파일에 등록 정보 값을 지정할 수 있습니다.

예: SystemDefault.properties 파일

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

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

JDK(Java Development Kit) 1.1.8의 Java 시스템 등록 정보

Java(TM) 시스템 등록 정보에서는 Java 프로그램이 실행할 환경을 판별합니다. 이것은 OS/400(R)에 있는 시스템 값이나 환경 변수와 비슷합니다. JVM(Java Virtual Machine)을 시작할 때 여러 개의 등록 정보가 설정됩니다.

시스템 등록 정보가 JDK 1.1.8에 있는 다음 시스템 디폴트 값으로 설정됩니다.

시스템 등록 정보	시스템 디폴트 값
awt.toolkit	com.ibm.rawt.client.CToolkit
file.encoding.pkg	sun.io

시스템 등록 정보	시스템 디폴트 값
file.separator	/(슬래시)
java.class.version	45.3
java.home	자세한 정보는 복수 JDK(Java Development Kit) 지원을 참조하십시오.
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
line.separator	\n
os.arch	PowerPC
os.name	OS/400
os400.class.path.rawt	0
os400.class.path.security.check	20 유효한 값: 0—보안 체크 없음 10—RUNJVA CHKPATH(*IGNORE)에 해당 20—RUNJVA CHKPATH(*WARN)에 해당 30—RUNJVA CHKPATH(*SECURE)에 해당
os400.class.path.tools	0
os400.create.type	direct 유효한 값: interpret—RUNJVA OPTIMIZE(*INTERPRET) 및 INTERPRET(*OPTIMIZE), 또는 INTERPRET(*YES)에 해당 direct—기타
os400.defineClass.optLevel	20
os400.enbpfrcol	0 유효한 값: 0—CRTJVAPGM ENBPFRCOL(*NONE)에 해당 1—CRTJVAPGM ENBPFRCOL(*ENTRYEXIT)에 해당 7—CRTJVAPGM ENBPFRCOL(*FULL)에 해당
os400.interpret	0 유효한 값: 0—CRTJVAPGM INTERPRET(*NO)에 해당 1—CRTJVAPGM INTERPRET(*YES)에 해당
os400.optimization	10 유효한 값: 0—CRTJVAPGM OPTIMIZE(*INTERPRET)에 해당 10—CRTJVAPGM OPTIMIZE(10)에 해당 20—CRTJVAPGM OPTIMIZE(20)에 해당 30—CRTJVAPGM OPTIMIZE(30)에 해당 40—CRTJVAPGM OPTIMIZE(40)에 해당
os400.run.mode	program_create_type 유효한 값: interpret—RUNJVA OPTIMIZE(*INTERPRET) 및 INTERPRET(*OPTIMIZE), 또는 INTERPRET(*YES)에 해당 program_create_type—Otherwise
os400.stdin.allowed	0

시스템 등록 정보	시스템 디폴트 값
os400.verify.checks.disable	65535 os400.class. 이 시스템 등록 정보값은 하나 이상의 숫자 값의 합을 표시하는 스트링입니다. 값 리스트는 os400.verify.checks.disable 숫자 값을 참조하십시오.
path.separator	:(콜론)

이 시스템 등록 정보 세트는 추가 시스템 정보에 근거하여 설정됩니다.

시스템 등록 정보	설명
file.encoding	OS/400 작업 CCSID를 해당 ISO ASCII CCSID로 맵합니다. 또한 file.encoding 값을 ISO ASCII CCSID(코드화 문자 세트 ID)를 나타내는 Java 값으로 설정합니다. 가능한 file.encoding 값과 가장 가깝게 일치하는 iSeries 코드화 문자 세트 ID(CCSID) 사이의 관계를 표시하는 표는 file.encoding 값과 iSeries CCSID를 참조하십시오.
java.class.path	클래스를 찾는 데 사용되는 경로. 사용자 지정 classpath가 뒤따르는 디폴트 시스템 classpath로 디폴트 설정됩니다. os400.class.path.system 시스템 등록 정보를 사용하여 java.class.path 시스템 등록 정보를 변경할 수 있습니다.
java.compiler	코드가 JIT(Just-In-Time) 컴파일러(jitc)로 컴파일되는지 아니면 JIT 컴파일러와 직접 실행(jitc_de) 둘다에 의해 컴파일되는지 여부를 지정합니다.
java.version	실행할 JDK(Java Development Kit)을 판별합니다. 설치되지 않은 JDK를 지정하는 경우 오류 메세지가 나타납니다. JDK를 지정하지 않으면 가장 최신 JDK가 디폴트입니다. JDK가 하나만 설치되어 있으면 그것이 디폴트가 됩니다. 버전 세부사항은 복수 JDK에 대한 지원을 참조하십시오.
os.version	OS/400 릴리스 레벨은 제품 정보 검색 API(어플리케이션 프로그램 인터페이스)로부터 얻습니다.
os400.CertificateContainer	시작한 Java 프로그램과 지정한 등록 정보에 대해 지정된 인증 컨테이너를 사용하도록 SSL(Java 보안 소켓층) 지원에 지정합니다. os400.secureApplication 시스템 등록 정보를 지정할 경우 이 시스템 등록 정보는 무시됩니다. 예를 들면 `-Dos400.certificateContainer= /home/username/mykeyfile.kdb`를 입력하거나 통합 파일 시스템에 있는 임의의 다른 키파일을 입력하십시오.
os400.CertificateLabel	os400.CertificateContainer 시스템 등록 정보와 함께 이 시스템 등록 정보를 지정할 수 있습니다. 이 등록 정보를 사용하면 지정된 컨테이너에서 SSL(Secure Sockets Layer)이 사용하기 원하는 인증을 선택할 수 있습니다. 예를 들어, `-Dos400.certificateLabel=myCert`를 입력하십시오. 여기에서 myCert는 인증을 작성하거나 가져올 때 DCM(Digital Certificate Manager)을 통해 인증에 할당하는 레이블명입니다.

시스템 등록 정보	설명
os400.child.stdio.convert	Java의 stdin, stdout 및 stderr에 대한 자료 변환 제어를 허용합니다. 자료 변환은 ASCII 자료와 EBCDIC 사이의 변환을 위해 JVM(Java Virtual Machine)에서 디폴트에 의해 발생합니다. runtime.exec() 메소드를 사용하여 이 프로세스에서 시작한 모든 하위 프로세스에 유효한 이 등록 정보를 사용하여 자료 변환을 끄거나 끌 수 있습니다. 디폴트 값을 참조하십시오.
os400.class.path.system	PRE(이 값은 os400 디폴트 시스템 classpath가 classpath를 java.class.path 시스템 등록 정보를 구성할 때 사용자 지정 부분에 사전 첨부됩니다). 다른 값은 POST(시스템 디폴트 클래스가 classpath의 사용자 지정 부분에 첨부됩니다.) 및 NONE(사용자 지정 classpath만이 사용됩니다.)입니다. 디폴트는 PRE입니다. 등록 정보는 대소문자를 구분하지 않습니다. 예를 들면, NONE, none, noNe을 지정할 수 있습니다. 그러나 등록 정보명은 대소문자를 구분합니다. 예를 들면, OS400.CLASS.PATH.SYSTEM을 지정할 수 없습니다. 잠재적인 문제를 피하려면 이 시스템 등록 정보를 변경하지 않아야 합니다.
 os400.file.create.auth, os400.dir.create.auth	이러한 등록 정보는 파일과 디렉토리에 할당된 권한을 지정합니다. 값을 사용하지 않거나 지원되는 값을 사용하여 등록 정보를 지정하면 공용 권한이 *NONE이 됩니다. os400.file.create.auth=RWX 또는 os400.dir.create.auth=RWX를 지정할 수 있습니다. 여기서 R=read, W=write 및 X=execute입니다. 이러한 권한을 결합해도 유효합니다.
os400.file.io.mode	디폴트가 BINARY가 아닌 TEXT를 지정할 때 file.encoding이 아니면 파일의 CCID를 변환합니다.
 os400.jit.mmi.threshold	JIT 컴파일로 컴파일하기 전에 메소드가 실행할 횟수를 설정합니다. 
 os400.pool.size	스레드 로컬 힙에서 각 힙 풀에 사용 가능한 공간을 정의합니다(킬로바이트 단위). 
os400.runtime.exec	• EXEC(1.3 이상의 경우 디폴트) - EXEC 인터페이스를 사용하여 runtime.exec()를 통해 함수를 호출합니다. 이것은 다른 플랫폼과 가장 호환이 잘 됩니다. • QSHELL(1.2 이상의 경우 디폴트) - QSHELL 인터프리터를 사용하여 runtime.exec()를 통해 함수를 호출합니다. 이것은 변수 대체의 사용과 내장 함수의 호출을 허용합니다.
os400.secureApplication	레지스터된 보안 어플리케이션 이름과 이 시스템 등록 정보(os400.secureApplication)를 사용할 때 시작하는 Java 프로그램을 연관시킵니다. DCM(Digital Certificate Manager)을 사용하여 레지스터된 보안 어플리케이션명을 볼 수 있습니다.

시스템 등록 정보	설명
os400.stderr	파일이나 소켓으로 stderr의 맵핑을 허용합니다. 디폴트 값을 참조하십시오.
os400.stdin	파일이나 소켓으로 stdin 맵핑을 허용합니다. 디폴트 값을 참조하십시오.
os400.stdio.convert	Java의 stdin, stdout 및 stderr에 대한 자료 변환 제어를 허용합니다. 자료 변환은 ASCII 자료와 EBCDIC 사이의 변환을 위해 JVM(Java Virtual Machine)에서 디폴트에 의해 발생합니다. 현재 Java 프로그램에 유효한 이 등록 정보를 사용하여 자료 변환을 켜거나 끌 수 있습니다. 디폴트 값을 참조하십시오.
os400.stdout	파일이나 소켓으로 stdout의 맵핑을 허용합니다. 디폴트 값을 참조하십시오.
user.dir	getcwd API를 사용하는 현재 작업 디렉토리.
user.home	Get API(getpwnam)를 사용하여 초기 작업 디렉토리를 검색합니다. SystemDefault.properties 파일을 user.home 경로에 배치하여 /QIBM/UserData/Java400/SystemDefault.properties에서 디폴트 등록 정보를 대체할 수 있습니다. iSeries 서버를 사용자 정의하여 사용자 자신의 디폴트 등록 정보 값 세트를 지정할 수 있습니다.
user.language	JVM(Java Virtual Machine)이 이 시스템 등록 정보를 사용하여 작업 LANGID 값을 읽고 이 값을 사용하여 대응하는 언어를 찾습니다.
user.name	JVM(Java Virtual Machine)이 이 시스템 등록 정보를 사용하여 TCB(Trusted Computing Base)의 보안 섹션 (Security.UserName)에서 유효 사용자 프로파일명을 검색합니다.
user.region	JVM(Java Virtual Machine)이 이 시스템 등록 정보를 사용하여 작업 CNTRYID 값을 읽고 이 값을 사용하여 사용자 영역을 판별합니다.
user.timezone	JVM(Java Virtual Machine)이 QlgRetrieveLocalInformation API를 사용하여 시간대 이름을 확보하기 위해 이 시스템 등록 정보를 사용합니다. 시간대 정보를 사용할 수 없는 경우 user.timezone이 만국 표준시(UTC)로 설정됩니다.

os400.stdio.convert 및 os400.child.stdio.convert 시스템 등록 정보 값

다음 표는 os400.stdin, os400.stdout 및 os400.stderr 시스템 등록 정보에 대한 시스템 값입니다.

값	설명
N(디폴트)	읽기 또는 쓰기 작업 동안 수행되는 stdio 변환이 없습니다.
Y	읽기 또는 쓰기 작업 동안 모든 stdio가 file.encoding 값에서 CCSID 작업으로 변환 또는 역변환됩니다.
1	읽기 작업 동안 stdin 자료만이 CCSID 작업에서 file.encoding으로 변환됩니다.
2	쓰기 작업 동안 stdout 자료만이 file.encoding에서 CCSID 작업으로 변환됩니다.
3	stdin 변환과 stdout 변환이 모두 실행됩니다.
4	쓰기 작업 동안 stderr 자료만이 file.encoding에서 CCSID 작업으로 변환됩니다.
5	stdin 변환과 stderr 변환이 모두 실행됩니다.
6	stdout 변환과 stderr 변환이 모두 실행됩니다.

값	설명
7	모든 stdio 변환이 수행됩니다.

os400.stdin, os400.stdout 및 os400.stderr 시스템 등록 정보 값

다음 표는 os400.stdin, os400.stdout 및 os400.stderr 시스템 등록 정보에 대한 시스템 값입니다.

값	이름 예	설명	예
파일	SomeFileName	SomeFileName은 현재 디렉토리에 대한 절대 또는 상대 경로입니다.	file:/QIBM/UserData/Java400/Output.file
포트	HostName	포트 주소	port:myhost:2000
포트	TCPAddress	포트 주소	port:1.1.11.111:2000

os400.verify.checks.disable 숫자 값

os400.verify.checks.disable 시스템 등록 정보 값은 다음 리스트에 나오는 하나 이상의 숫자 값의 합을 표시하는 스트링입니다.

값	설명
1	**로컬 클래스에 대한 액세스 검사 바이패스**는 로컬 파일 시스템에서 로드된 클래스의 경우 JVM(Java(TM) virtual machine)이 개인 및 보호 필드와 메소드에서 액세스 검사를 바이패스하려 함을 나타냅니다. 이것은 둘러싸고 있는 클래스의 개인 및 보호 메소드와 필드를 참조하는 내부 클래스가 포함된 어플리케이션을 전송할 때 도움이 됩니다.
2	**초기 로드 중에 NoClassDefFoundError 억제:** JVM(Java Virtual Machine)이 타입캐스팅(typecasting) 및 필드 또는 메소드 액세스에 대한 초기 확인 체크 중에 발생하는 NoClassDefFoundErrors를 무시하기 원함을 표시합니다.
4	**LocalVariableTable 체크가 바이패스될 수 있음:** 클래스의 LocalVariableTable에서 오류가 발생하는 경우 클래스가 LocalVariableTable이 존재하지 않는 것처럼 동작하도록 표시합니다. 그렇지 않으면 LocaleVariableTable의 오류가 ClassFormatError를 유발합니다.
7	런타임에 사용되는 값.

십진, 16진 또는 8진 형식으로 값을 지정할 수 있습니다. 0(영)보다 작은 값은 무시됩니다. 예를 들어, 리스트에서 처음 두 개의 값을 선택하려면 이 iSeries 명령 구문을 사용하십시오.

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

J2SDK(Java system properties for Java 2 Software Development Kit), 표준판

Java(TM) 시스템 등록 정보에서는 Java 프로그램이 실행할 환경을 판별합니다. 이 등록 정보는 OS/400의 시스템 값 또는 환경 변수와 같습니다. JVM(Java Virtual Machine)을 시작할 때 여러 개의 등록 정보가 설정됩니다.

시스템 등록 정보는 J2SDK(Java 2 Software Development Kit), 표준판, 버전 1.4에서 다음의 시스템 디폴트 값으로 설정됩니다. JNI(Java Native Interface) 호출 API를 사용하는 경우보다 JAVA 또는 RUNJVA CL 명령을 수행하는 경우에 많은 시스템 등록 정보는 다른 디폴트 값을 가집니다. 다음의 표는 API의 사용을 반

영한 것입니다.

시스템 등록 정보	시스템 디폴트 값
awt.toolkit	JDK 1.1.x의 경우 디폴트 값은 com.ibm.rawt.client.CToolkit입니다. J2SDK의 경우 디폴트 값은 com.ibm.rawt2.ahost.java.awt.AHToolkit입니다.
file.encoding.pkg	sun.io
file.separator	/(슬래시)
java.class.version	 48.0 
java.ext.dirs	 /QIBM/ProdData/Java400/jdk14/lib/ext:/QIBM/UserData/Java400/ext 
java.home	자세한 정보는 복수 JDK(Java Development Kit) 지원을 참조하십시오.
java.library.path	OS/400 라이브러리 리스트
java.policy	 /QIBM/ProdData/Java400/jdk14/lib/security/java.policy 
java.specification.name	Java 언어 지정
java.specification.vendor	Sun Microsystems, Inc.
java.specification.version	 1.4 
 sun.boot.class.path 	Class_Path_Sys
java.use.policy	true
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com

시스템 등록 정보	시스템 디폴트 값
java.vm.name	 Classic VM 
java.vm.specification.name	JVM(Java Virtual Machine) 지정
java.vm.specification.vendor	Sun Microsystems, Inc.
java.vm.specification.version	 1.0 
java.vm.vendor	IBM Corporation
java.vm.version	 1.4 
line.separator	\n
os.arch	PowerPC
os.name	OS/400
os400.class.path.rawt	0
os400.class.path.security.check	20 유효한 값: 0—보안 체크 없음 10—RUNJVA CHKPATH(*IGNORE)에 해당 20—RUNJVA CHKPATH(*WARN)에 해당 30—RUNJVA CHKPATH(*SECURE)에 해당
os400.class.path.tools	0
os400.create.type	 해석  유효한 값: interpret—RUNJVA OPTIMIZE(*INTERPRET) 및 INTERPRET(*OPTIMIZE), 또는 INTERPRET(*YES)에 해당 direct—기타
os400.defineClass.optLevel	20
os400.enbpfrcol	0 유효한 값: 0—CRTJVAPGM ENBPFRCOL(*NONE)에 해당 1—CRTJVAPGM ENBPFRCOL(*ENTRYEXIT)에 해당 7—CRTJVAPGM ENBPFRCOL(*FULL)에 해당 0이 아닌 값의 경우 JIT는 *JVAENTRY, *JVAEXIT, *JVAPRECALL 및 *JVAPOSTCALL 이벤트를 생성합니다.

시스템 등록 정보	시스템 디폴트 값
os400.interpret	0 유효한 값: 0—CRTJVAPGM INTERPRET(*NO)에 해당 1—CRTJVAPGM INTERPRET(*YES)에 해당
os400.optimization	 0  유효한 값: 0—CRTJVAPGM OPTIMIZE(*INTERPRET)에 해당 10—CRTJVAPGM OPTIMIZE(10)에 해당 20—CRTJVAPGM OPTIMIZE(20)에 해당 30—CRTJVAPGM OPTIMIZE(30)에 해당 40—CRTJVAPGM OPTIMIZE(40)에 해당
os400.run.mode	 jitc_de  유효한 값: interpret—RUNJVA OPTIMIZE(*INTERPRET) 및 INTERPRET(*OPTIMIZE), 또는 INTERPRET(*YES)에 해당 program_create_type jitc_de—Otherwise
os400.stdin.allowed	0
os400.verify.checks.disable	65535 이 시스템 등록 정보값은 하나 이상의 숫자 값의 합을 표시하는 스트링입니다. 값 리스트는 os400.verify.checks.disable 숫자 값을 참조하십시오.
path.separator	:(콜론)

이 시스템 등록 정보 세트는 추가 시스템 정보에 근거하여 설정됩니다.

시스템 등록 정보	설명
file.encoding	OS/400 작업 CCSID를 해당 ISO ASCII CCSID로 맵합니다. 또한 file.encoding 값을 ISO ASCII CCSID(코드화 문자 세트 ID)를 나타내는 Java 값으로 설정합니다. 가능한 file.encoding 값과 가장 가깝게 일치하는 iSeries 코드화 문자 세트 ID(CCSID) 사이의 관계를 표시하는 표는 file.encoding 값과 iSeries CCSID를 참조하십시오.
java.class.path	클래스를 찾는데 사용되는 경로. 사용자 지정 classpath로 디폴트 설정됩니다.
java.compiler	코드가 JIT(Just-In-Time) 컴파일러(jitc)로 컴파일되는지 아니면 JIT 컴파일러와 직접 실행(jitc_de) 둘다에 의해 컴파일되는지 여부를 지정합니다.

시스템 등록 정보	설명
java.version	실행할 JDK(Java Development Kit)를 판별합니다. 설치되지 않은 JDK를 지정하는 경우 오류 메세지가 나타납니다. JDK를 지정하지 않으면 가장 최신 JDK가 디폴트입니다. JDK가 하나만 설치되어 있으면 그것이 디폴트가 됩니다. 버전 세부사항은 복수 JDK에 대한 지원을 참조하십시오.
os.version	OS/400 릴리스 레벨은 제품 정보 검색 API(어플리케이션 프로그램 인터페이스)로부터 얻습니다.
os400.CertificateContainer	시작한 Java 프로그램과 지정한 등록 정보에 대해 지정된 인증 컨테이너를 사용하도록 SSL(Java 보안 소켓층) 지원에 지정합니다. os400.secureApplication 시스템 등록 정보를 지정할 경우 이 시스템 등록 정보는 무시됩니다. 예를 들면 `-Dos400.certificateContainer=/home/username/mykeyfile.kdb` 또는 통합 파일 시스템에 있는 다른 키파일을 입력하십시오.
os400.CertificateLabel	os400.CertificateContainer 시스템 등록 정보와 함께 이 시스템 등록 정보를 지정할 수 있습니다. 이 등록 정보를 사용하면 지정된 컨테이너에서 SSL(Secure Sockets Layer)이 사용하기 원하는 인증을 선택할 수 있습니다. 예를 들어, `-Dos400.certificateLabel=myCert`를 입력하십시오. 여기에서 myCert는 인증을 작성하거나 가져올 때 DCM(Digital Certificate Manager)을 통해 인증에 할당하는 레이블명입니다.
os400.child.stdio.convert	Java의 stdin, stdout 및 stderr에 대한 자료 변환 제어를 허용합니다. 자료 변환은 ASCII 자료와 EBCDIC 사이의 변환을 위해 JVM(Java Virtual Machine)에서 디폴트에 의해 발생합니다. runtime.exec() 메소드를 사용하여 이 프로세스에서 시작한 모든 하위 프로세스에 유효한 이 등록 정보를 사용하여 자료 변환을 끄거나 끌 수 있습니다. 디폴트 값을 참조하십시오.
os400.class.path.system	 이 시스템 등록 정보는 J2SDK의 경우에 무시합니다.
 os400.define.class.cache.file	이 등록 정보는 JAR 또는 ZIP 파일의 이름을 지정합니다. 디폴트는 널(null)입니다. 436 페이지의 『클래스 로더 캐시』를 참조하십시오. 
 os400.define.class.cache.hours	등록 정보는 십진 값입니다. 디폴트 값은 168이며 최대 십진 값은 9999입니다. 436 페이지의 『클래스 로더 캐시』를 참조하십시오. 
 os400.define.class.cache.maxpgms	등록 정보는 십진 값입니다. 디폴트 값은 5000이며 최대 십진 값은 40000입니다. 436 페이지의 『클래스 로더 캐시』를 참조하십시오. 

시스템 등록 정보	설명
os400.exception.trace	이 등록 정보를 지정하면 JVM이 나갈 때 가장 최근의 예외가 표준 출력으로 송신됩니다. 이 등록 정보에 지정된 값은 현재 무시되지만 나중에 변경될 것입니다. 이 등록 정보는 순수하게 디버깅으로만 사용됩니다.
 os400.file.create.auth, os400.dir.create.auth	이러한 등록 정보는 파일과 디렉토리에 할당된 권한을 지정합니다. 값을 사용하지 않거나 지원되는 값을 사용하여 등록 정보를 지정하면 공용 권한이 *NONE이 됩니다. os400.file.create.auth=RWX 또는 os400.dir.create.auth=RWX를 지정할 수 있습니다. 여기서 R=read, W=write 및 X=execute입니다. 이러한 권한을 결합해도 유효합니다. 
os400.file.io.mode	디폴트가 BINARY가 아닌 TEXT를 지정할 때 file.encoding이 아니면 파일의 CCID를 변환합니다.
 os400.jit.mmi.threshold	JIT 컴파일로 컴파일하기 전에 메소드가 실행할 횟수를 설정합니다. 
 os400.pool.size	스레드 로컬 힙에서 각 힙 풀에 사용 가능한 공간을 정의합니다 (킬로바이트 단위). 
os400.runtime.exec	• EXEC(1.3 이상의 경우 디폴트) - EXEC 인터페이스를 사용하여 runtime.exec()를 통해 함수를 호출합니다. 이것은 다른 플랫폼과 가장 호환이 잘 됩니다. • QSHELL(1.2 이상의 경우 디폴트) - QSHELL 인터프리터를 사용하여 runtime.exec()를 통해 함수를 호출합니다. 이것은 변수 대체의 사용과 내장 함수의 호출을 허용합니다.
os400.secureApplication	레지스터된 보안 어플리케이션 이름과 이 시스템 등록 정보 (os400.secureApplication)를 사용할 때 시작하는 Java 프로그램을 연관시킵니다. DCM(Digital Certificate Manager)을 사용하여 레지스터된 보안 어플리케이션명을 볼 수 있습니다.
 os.400.security.properties	사용할 java.security 파일 전체를 제어할 수 있도록 합니다. 이 등록 정보를 지정할 경우, J2SDK는 디폴트인 J2SDK 고유 java.security를 포함하여 다른 java.security 파일을 사용하지 않습니다. 
os400.stderr	파일이나 소켓으로 stderr의 맵핑을 허용합니다. 디폴트 값을 참조하십시오.
os400.stdin	파일이나 소켓으로 stdin 맵핑을 허용합니다. 디폴트 값을 참조하십시오.
os400.stdin.allowed	stdin이 허용되는지(1) 아니면 허용되지 않는지(0) 여부를 지정합니다. 호출자가 일괄처리 작업을 실행 중인 경우 stdin이 허용되지 않습니다. 디폴트 값은 0입니다.

시스템 등록 정보	설명
os400.stdio.convert	Java의 stdin, stdout 및 stderr에 대한 자료 변환 제어를 허용합니다. 자료 변환은 ASCII 자료와 EBCDIC 사이의 변환을 위해 JVM(Java Virtual Machine)에서 디폴트에 의해 발생합니다. 현재 Java 프로그램에 유효한 이 등록 정보를 사용하여 자료 변환을 켜거나 끌 수 있습니다. 디폴트 값을 참조하십시오.
os400.stdout	파일이나 소켓으로 stdout의 맵핑을 허용합니다. 디폴트 값을 참조하십시오.
user.dir	getcwd API를 사용하는 현재 작업 디렉토리.
user.home	Get API(getpwnam)를 사용하여 초기 작업 디렉토리를 검색합니다. SystemDefault.properties 파일을 user.home 경로에 배치하여 /QIBM/UserData/Java400/SystemDefault.properties에서 디폴트 등록 정보를 대체할 수 있습니다. iSeries 서버를 사용자 정의하여 사용자 자신의 디폴트 등록 정보 값을 지정할 수 있습니다.
user.language	JVM(Java Virtual Machine)이 이 시스템 등록 정보를 사용하여 작업 LANGID 값을 읽고 이 값을 사용하여 대응하는 언어를 찾습니다.
user.name	JVM(Java Virtual Machine)이 이 시스템 등록 정보를 사용하여 TCB(Trusted Computing Base)의 보안 섹션 (Security.UserName)에서 유효 사용자 프로파일명을 검색합니다.
user.region	JVM(Java Virtual Machine)이 이 시스템 등록 정보를 사용하여 작업 CNTRYID 값을 읽고 이 값을 사용하여 사용자 영역을 판별합니다.
user.timezone	JVM(Java Virtual Machine)이 QlgRetrieveLocalInformation API를 사용하여 시간대 이름을 확보하기 위해 이 시스템 등록 정보를 사용합니다. 시간대 정보를 사용할 수 없는 경우 user.timezone이 만국 표준시(UTC)로 설정됩니다.

국제화된 Java 프로그램 작성

특정 지역에 맞도록 Java(TM) 프로그램을 사용자 정의해야 하는 경우, Java 로케일로 국제화된 Java 프로그램을 작성할 수 있습니다.

국제화된 Java 프로그램을 작성하려면 다음의 단계를 수행하십시오.

1. 로케일(locale) 관련 코드 및 자료 분리. 예를 들면, 프로그램 내의 스트링, 날짜 및 숫자들
2. Locale 클래스를 사용하여 로케일 설정 또는 확보
3. 디폴트 로케일을 사용하지 않을 때 로케일을 지정하기 위한 날짜 및 숫자 형식화
4. 스트링 및 기타 로케일 관련 자료를 처리하기 위한 자원 번들 작성

Java 프로그램에서 이러한 타스크를 수행하려면 다음 예를 참조하십시오.

- java.util.DateFormat 클래스를 사용하여 날짜 국제화
- java.util.NumberFormat 클래스를 사용하여 숫자 표시 국제화
- java.util.ResourceBundle 클래스를 사용하여 로케일 특정 자료 국제화

국제화에 대한 자세한 내용은 다음 링크를 클릭하십시오.

- OS/400 국제화
- Sun Microsystems, Inc.의 국제화

iSeries 서버에서 시간대 환경 변수

J2SDK(Java(TM) 2 SDK), Stanard Edition, 버전 1.4를 사용하면 원시 메소드 getSystemTimeZoneID를 사용하여 JVM의 시간대를 설정할 수 있습니다. iSeries 서버는 *ENV 오브젝트의 일부인 *LOCALE 오브젝트를 사용합니다. *LOCALE 오브젝트의 tname 필드를 적합한 시스템 값으로 설정하십시오. 그러면, 이 값이 getSystemTimeZoneID()에서 연관된 Java 스트링 오브젝트로 리턴됩니다.

시간대 구성: JVC는 현지 시각을 올바로 판별하기 위해 LOCALE에 현재 작업에 대한 QUTCOFFSET 시스템 값 및 시각 정보를 모두 설정할 것을 요구합니다. QUTCOFFSET는 현지 시각과 만국 표준시(UTC) 사이의 시간차를 지정하는 시스템 값입니다. 중앙 표준시(CST)의 경우 이것은 -6:00입니다. 중앙 주간시(CDT)의 경우 올바른 값은 -5:00입니다. QUTCOFFSET 값은 JVM이 UTC에 대한 올바른 값을 판별할 수 있게 합니다.

작업에 대한 LOCALE 정보는 시각 정보가 포함된 *LOCALE 오브젝트를 작성하고 작업에 대한 사용자 프로파일에서 QLOCALE 시스템 값이나 LOCALE 키워드를 사용하여 작업에 대한 *LOCALE을 지정함으로 설정됩니다. LOCALE 작성 및 사용에 대한 자세한 정보는 OS/400 국제화 책에서 찾을 수 있습니다.

*LOCALE 정보를 올바로 설정하면 JVM은 디폴트로 user.timezone 등록 정보를 올바른 시간대로 설정할 수 있습니다. *LOCALE 오브젝트에 의해 제공되는 디폴트 설정을 대체하려면 user.timezone 등록 정보를 명령행에서 수동으로 설정할 수 있습니다.

다음은 Java에 대한 올바른 시간대를 구성하기 위해 *LOCALE 오브젝트에 포함시켜야 하는 LC_TOD 정보입니다.

```
LC_TOD

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

END LC_TOD
```

다음 표에서는 시스템 값 및 연관된 Java 스트링 오브젝트를 나타냅니다.

주: 시스템 값 ″Hong Kong″은 중국(홍콩 S.A.R.)을 말합니다.

시스템 값	**Java** 스트링 오브젝트
Africa/Abidjan	Africa/Abidjan
Africa/Accra	Africa/Accra
Africa/Addis_Ababa	Africa/Addis_Ababa
Africa/Algiers	Africa/Algiers
Africa/Asmera	Africa/Asmera
Africa/Bamako	GMT
Africa/Bangui	Africa/Bangui
Africa/Banjul	Africa/Banjul
Africa/Bissau	Africa/Bissau
Africa/Blantyre	Africa/Blantyre
Africa/Brazzaville	Africa/Luanda
Africa/Bujumbura	Africa/Bujumbura
Africa/Cairo	Africa/Cairo
Africa/Casablanca	Africa/Casablanca
Africa/Ceuta	Europe/Paris
Africa/Conakry	Africa/Conakry
Africa/Dakar	Africa/Dakar
Africa/Dar_es_Salaam	Africa/Dar_es_Salaam
Africa/Djibouti	Africa/Djibouti
Africa/Douala	Africa/Douala
Africa/El_Aaiun	Africa/Casablanca
Africa/Freetown	Africa/Freetown
Africa/Gaborone	Africa/Gaborone
Africa/Harare	Africa/Harare
Africa/Johannesburg	Africa/Johannesburg
Africa/Kampala	Africa/Kampala
Africa/Khartoum	Africa/Khartoum
Africa/Kigali	Africa/Kigali
Africa/Kinshasa	Africa/Kinshasa
Africa/Lagos	Africa/Lagos
Africa/Libreville	Africa/Libreville
Africa/Lome	Africa/Lome
Africa/Luanda	Africa/Luanda
Africa/Lubumbashi	Africa/Lubumbashi
Africa/Lusaka	Africa/Lusaka
Africa/Malabo	Africa/Malabo
Africa/Maputo	Africa/Maputo
Africa/Maseru	Africa/Maseru
Africa/Mbabane	Africa/Mbabane
Africa/Mogadishu	Africa/Mogadishu
Africa/Monrovia	Africa/Monrovia

시스템 값	**Java** 스트링 오브젝트
Africa/Nairobi	Africa/Nairobi
Africa/Ndjamena	Africa/Ndjamena
Africa/Niamey	Africa/Niamey
Africa/Nouakchott	Africa/Nouakchott
Africa/Ouagadougou	Africa/Ouagadougou
Africa/Porto-Novo	Africa/Porto-Novo
Africa/Sao_Tome	Africa/Sao_Tome
Africa/Timbuktu	Africa/Timbuktu
Africa/Tripoli	Africa/Tripoli
Africa/Tunis	Africa/Tunis
Africa/Windhoek	Africa/Windhoek
America/Adak	America/Adak
America/Anchorage	America/Anchorage
America/Anguilla	America/Anguilla
America/Antigua	America/Antigua
America/Araguaina	America/Sao_Paulo
America/Aruba	America/Aruba
America/Asuncion	America/Asuncion
America/Atka	America/Adak
America/Barbados	America/Barbados
America/Belize	America/Belize
America/Bogota	America/Bogota
America/Boise	America/Denver
America/Buenos_Aires	America/Buenos_Aires
America/Cancun	America/Chicago
America/Caracas	America/Caracas
America/Cayenne	America/Cayenne
America/Cayman	America/Cayman
America/Chicago	America/Chicago
America/Chihuahua	America/Denver
America/Costa_Rica	America/Costa_Rica
America/Cuiaba	America/Cuiaba
America/Curacao	America/Curacao
America/Dawson	America/Los_Angeles
America/Dawson_Creek	America/Dawson_Creek
America/Denver	America/Denver
America/Detroit	America/New_York
America/Dominica	America/Dominica
America/Edmonton	America/Edmonton
America/El_Salvador	America/El_Salvador
America/Ensenada	America/Los_Angeles

시스템 값	**Java** 스트링 오브젝트
America/Fort_Wayne	America/Indianapolis
America/Fortaleza	America/Fortaleza
America/Glace_Bay	America/Halifax
America/Godthab	America/Godthab
America/Goose_Bay	America/Thule
America/Grand_Turk	America/Grand_Turk
America/Grenada	America/Grenada
America/Guadeloupe	America/Guadeloupe
America/Guatemala	America/Guatemala
America/Guayaquil	America/Guayaquil
America/Guyana	America/Guyana
America/Halifax	America/Halifax
America/Havana	America/Havana
America/Indiana/Indianapolis	America/Indianapolis
America/Indianapolis	America/Indianapolis
America/Inuvik	America/Denver
America/Iqaluit	America/New_York
America/Jamaica	America/Jamaica
America/Juneau	America/Anchorage
America/La_Paz	America/La_Paz
America/Lima	America/Lima
America/Los_Angeles	America/Los_Angeles
America/Louisville	America/New_York
America/Managua	America/Managua
America/Manaus	America/Manaus
America/Martinique	America/Martinique
America/Mazatlan	America/Mazatlan
America/Menominee	America/Winnipeg
America/Mexico_City	America/Mexico_City
America/Miquelon	America/Miquelon
America/Montevideo	America/Montevideo
America/Montreal	America/Montreal
America/Montserrat	America/Montserrat
America/Nassau	America/Nassau
America/New_York	America/New_York
America/Nipigon	America/New_York
America/Nome	America/Anchorage
America/Noronha	America/Noronha
America/Panama	America/Panama
America/Pangnirtung	America/Thule
America/Paramaribo	America/Paramaribo

시스템 값	**Java** 스트링 오브젝트
America/Phoenix	America/Phoenix
America/Port-au-Prince	America/Port-au-Prince
America/Port_of_Spain	America/Port_of_Spain
America/Porto_Acre	America/Porto_Acre
America/Puerto_Rico	America/Puerto_Rico
America/Rainy_River	America/Chicago
America/Rankin_Inlet	America/Chicago
America/Regina	America/Regina
America/Santiago	America/Santiago
America/Santo_Domingo	America/Santo_Domingo
America/Sao_Paulo	America/Sao_Paulo
America/Scoresbysund	America/Scoresbysund
America/Shiprock	America/Denver
America/St_Johns	America/St_Johns
America/St_Kitts	America/St_Kitts
America/St_Lucia	America/St_Lucia
America/St_Thomas	America/St_Thomas
America/St_Vincent	America/St_Vincent
America/Tegucigalpa	America/Tegucigalpa
America/Thule	America/Thule
America/Thunder_Bay	America/New_York
America/Tijuana	America/Tijuana
America/Tortola	America/Tortola
America/Vancouver	America/Vancouver
America/Virgin	America/St_Thomas
America/Whitehorse	America/Los_Angeles
America/Winnipeg	America/Winnipeg
America/Yakutat	America/Anchorage
America/Yellowknife	America/Denver
Antarctica/Casey	Antarctica/Casey
Antarctica/DumontDUrville	Antarctica/DumontDUrville
Antarctica/Mawson	Antarctica/Mawson
Antarctica/McMurdo	Antarctica/McMurdo
Antarctica/Palmer	Antarctica/Palmer
Antarctica/South_Pole	Antarctica/McMurdo
Arctic/Longyearbyen	Europe/Oslo
Asia/Aden	Asia/Aden
Asia/Almaty	Asia/Almaty
Asia/Amman	Asia/Amman
Asia/Anadyr	Asia/Anadyr
Asia/Aqtau	Asia/Aqtau

시스템 값	**Java** 스트링 오브젝트
Asia/Aqtobe	Asia/Aqtobe
Asia/Ashkhabad	Asia/Ashkhabad
Asia/Baghdad	Asia/Baghdad
Asia/Bahrain	Asia/Bahrain
Asia/Baku	Asia/Baku
Asia/Bangkok	Asia/Bangkok
Asia/Beirut	Asia/Beirut
Asia/Bishkek	Asia/Bishkek
Asia/Brunei	Asia/Brunei
Asia/Calcutta	Asia/Calcutta
Asia/Chungking	Asia/Shanghai
Asia/Colombo	Asia/Colombo
Asia/Dacca	Asia/Dacca
Asia/Damascus	Asia/Damascus
Asia/Dubai	Asia/Dubai
Asia/Dushanbe	Asia/Dushanbe
Asia/Gaza	Asia/Amman
Asia/Harbin	Asia/Shanghai
Asia/Hong_Kong	Asia/Hong_Kong
Asia/Irkutsk	Asia/Irkutsk
Asia/Istanbul	Europe/Istanbul
Asia/Jakarta	Asia/Jakarta
Asia/Jayapura	Asia/Jayapura
Asia/Jerusalem	Asia/Jerusalem
Asia/Kabul	Asia/Kabul
Asia/Kamchatka	Asia/Kamchatka
Asia/Karachi	Asia/Karachi
Asia/Kashgar	Asia/Shanghai
Asia/Katmandu	Asia/Katmandu
Asia/Krasnoyarsk	Asia/Krasnoyarsk
Asia/Kuala_Lumpur	Asia/Kuala_Lumpur
Asia/Kuwait	Asia/Kuwait
Asia/Macao	Asia/Macao
Asia/Magadan	Asia/Magadan
Asia/Manila	Asia/Manila
Asia/Muscat	Asia/Muscat
Asia/Nicosia	Asia/Nicosia
Asia/Novosibirsk	Asia/Novosibirsk
Asia/Omsk	Asia/Novosibirsk
Asia/Phnom_Penh	Asia/Phnom_Penh
Asia/Pyongyang	Asia/Pyongyang

시스템 값	**Java** 스트링 오브젝트
Asia/Qatar	Asia/Qatar
Asia/Rangoon	Asia/Rangoon
Asia/Riyadh	Asia/Riyadh
Asia/Saigon	Asia/Saigon
Asia/Seoul	Asia/Seoul
Asia/Shanghai	Asia/Shanghai
Asia/Singapore	Asia/Singapore
Asia/Taipei	Asia/Taipei
Asia/Tashkent	Asia/Tashkent
Asia/Tbilisi	Asia/Tbilisi
Asia/Tehran	Asia/Tehran
Asia/Tel_Aviv	Asia/Jerusalem
Asia/Thimbu	Asia/Thimbu
Asia/Tokyo	Asia/Tokyo
Asia/Ujung_Pandang	Asia/Ujung_Pandang
Asia/Ulan_Bator	Asia/Ulan_Bator
Asia/Urumqi	Asia/Shanghai
Asia/Vientiane	Asia/Vientiane
Asia/Vladivostok	Asia/Vladivostok
Asia/Yakutsk	Asia/Yakutsk
Asia/Yekaterinburg	Asia/Yekaterinburg
Asia/Yerevan	Asia/Yerevan
Atlantic/Azores	Atlantic/Azores
Atlantic/Bermuda	Atlantic/Bermuda
Atlantic/Canary	Atlantic/Canary
Atlantic/Cape_Verde	Atlantic/Cape_Verde
Atlantic/Faeroe	Atlantic/Faeroe
Atlantic/Jan_Mayen	Atlantic/Jan_Mayen
Atlantic/Madeira	Europe/London
Atlantic/Reykjavik	Atlantic/Reykjavik
Atlantic/South_Georgia	Atlantic/South_Georgia
Atlantic/St_Helena	Atlantic/St_Helena
Atlantic/Stanley	Atlantic/Stanley
Australia/ACT	Australia/Sydney
Australia/Adelaide	Australia/Adelaide
Australia/Brisbane	Australia/Brisbane
Australia/Broken_Hill	Australia/Broken_Hill
Australia/Canberra	Australia/Sydney
Australia/Darwin	Australia/Darwin
Australia/Hobart	Australia/Hobart
Australia/LHI	Australia/Lord_Howe

시스템 값	**Java** 스트링 오브젝트
Australia/Lord_Howe	Australia/Lord_Howe
Australia/Melbourne	Australia/Sydney
Australia/NSW	Australia/Sydney
Australia/North	Australia/Darwin
Australia/Perth	Australia/Perth
Australia/Queensland	Australia/Brisbane
Australia/South	Australia/Adelaide
Australia/Sydney	Australia/Sydney
Australia/Tasmania	Australia/Hobart
Australia/Victoria	Australia/Sydney
Australia/West	Australia/Perth
Australia/Yancowinna	Australia/Broken_Hill
Brazil/Acre	America/Porto_Acre
Brazil/DeNoronha	America/Noronha
Brazil/East	America/Sao_Paulo
Brazil/West	America/Manaus
CET	Europe/Paris
CST	America/Chicago
CST6CDT	America/Chicago
Canada/Atlantic	America/Halifax
Canada/Central	America/Winnipeg
Canada/East-Saskatchewan	America/Regina
Canada/Eastern	America/Montreal
Canada/Mountain	America/Edmonton
Canada/Newfoundland	America/St_Johns
Canada/Pacific	America/Vancouver
Canada/Saskatchewan	America/Regina
Canada/Yukon	America/Los_Angeles
Chile/Continental	America/Santiago
Chile/EasterIsland	Pacific/Easter
Cuba	America/Havana
EET	America/Indianapolis
EST5EDT	America/New_York
Egypt	Africa/Cairo
Eire	Europe/Dublin
Etc/GMT	GMT
Etc/GMT0	GMT
Etc/Greenwich	GMT
Etc/UCT	UTC
Etc/UTC	UTC
Etc/Universal	UTC

시스템 값	**Java** 스트링 오브젝트
Etc/Zulu	UTC
Europe/Amsterdam	Europe/Amsterdam
Europe/Andorra	Europe/Andorra
Europe/Athens	Europe/Athens
Europe/Belfast	Europe/London
Europe/Belgrade	Europe/Belgrade
Europe/Berlin	Europe/Berlin
Europe/Bratislava	Europe/Prague
Europe/Brussels	Europe/Brussels
Europe/Bucharest	Europe/Bucharest
Europe/Budapest	Europe/Budapest
Europe/Chisinau	Europe/Chisinau
Europe/Copenhagen	Europe/Copenhagen
Europe/Dublin	Europe/Dublin
Europe/Gibraltar	Europe/Gibraltar
Europe/Helsinki	Europe/Helsinki
Europe/Istanbul	Europe/Istanbul
Europe/Kaliningrad	Europe/Kaliningrad
Europe/Kiev	Europe/Kiev
Europe/Lisbon	Europe/Lisbon
Europe/Ljubljana	Europe/Belgrade
Europe/London	Europe/London
Europe/Luxembourg	Europe/Luxembourg
Europe/Madrid	Europe/Madrid
Europe/Malta	Europe/Malta
Europe/Minsk	Europe/Minsk
Europe/Monaco	Europe/Monaco
Europe/Moscow	Europe/Moscow
Europe/Oslo	Europe/Oslo
Europe/Paris	Europe/Paris
Europe/Prague	Europe/Prague
Europe/Riga	Europe/Riga
Europe/Rome	Europe/Rome
Europe/Samara	Europe/Samara
Europe/San_Marino	Europe/Rome
Europe/Sarajevo	Europe/Belgrade
Europe/Simferopol	Europe/Simferopol
Europe/Skopje	Europe/Belgrade
Europe/Sofia	Europe/Sofia
Europe/Stockholm	Europe/Stockholm
Europe/Tallinn	Europe/Tallinn

시스템 값	**Java** 스트링 오브젝트
Europe/Tirane	Europe/Tirane
Europe/Vaduz	Europe/Vaduz
Europe/Vatican	Europe/Rome
Europe/Vienna	Europe/Vienna
Europe/Vilnius	Europe/Vilnius
Europe/Warsaw	Europe/Warsaw
Europe/Zagreb	Europe/Belgrade
Europe/Zurich	Europe/Zurich
Factory	GMT
GB	Europe/London
GB-Eire	Europe/London
GMT	GMT
GMT0	GMT
Greenwich	GMT
HST	Pacific/Honolulu
Hongkong	Asia/Hong_Kong
Iceland	Atlantic/Reykjavik
Indian/Antananarivo	Indian/Antananarivo
Indian/Chagos	Indian/Chagos
Indian/Christmas	Indian/Christmas
Indian/Cocos	Indian/Cocos
Indian/Comoro	Indian/Comoro
Indian/Kerguelen	Indian/Kerguelen
Indian/Mahe	Indian/Mahe
Indian/Maldives	Indian/Maldives
Indian/Mauritius	Indian/Mauritius
Indian/Mayotte	Indian/Mayotte
Indian/Reunion	Indian/Reunion
Iran	Asia/Tehran
Israel	Asia/Jerusalem
Jamaica	America/Jamaica
Japan	Asia/Tokyo
Libya	Africa/Tripoli
MET	Europe/Paris
MST	America/Phoenix
MST7MDT	America/Denver
Mexico/BajaNorte	America/Tijuana
Mexico/BajaSur	America/Mazatlan
Mexico/General	America/Mexico_City
NZ	Pacific/Auckland
NZ-CHAT	Pacific/Chatham

시스템 값	**Java** 스트링 오브젝트
Navajo	America/Denver
PRC	Asia/Shanghai
PST	America/Los_Angeles
PST8PDT	America/Los_Angeles
Pacific/Apia	Pacific/Apia
Pacific/Auckland	Pacific/Auckland
Pacific/Chatham	Pacific/Chatham
Pacific/Easter	Pacific/Easter
Pacific/Efate	Pacific/Efate
Pacific/Enderbury	Pacific/Enderbury
Pacific/Fakaofo	Pacific/Fakaofo
Pacific/Fiji	Pacific/Fiji
Pacific/Funafuti	Pacific/Funafuti
Pacific/Galapagos	Pacific/Galapagos
Pacific/Gambier	Pacific/Gambier
Pacific/Guadalcanal	Pacific/Guadalcanal
Pacific/Guam	Pacific/Guam
Pacific/Honolulu	Pacific/Honolulu
Pacific/Kiritimati	Pacific/Kiritimati
Pacific/Kosrae	Pacific/Kosrae
Pacific/Majuro	Pacific/Majuro
Pacific/Marquesas	Pacific/Marquesas
Pacific/Nauru	Pacific/Nauru
Pacific/Niue	Pacific/Niue
Pacific/Norfolk	Pacific/Norfolk
Pacific/Noumea	Pacific/Noumea
Pacific/Pago_Pago	Pacific/Pago_Pago
Pacific/Palau	Pacific/Palau
Pacific/Pitcairn	Pacific/Pitcairn
Pacific/Ponape	Pacific/Ponape
Pacific/Port_Moresby	Pacific/Port_Moresby
Pacific/Rarotonga	Pacific/Rarotonga
Pacific/Saipan	Pacific/Saipan
Pacific/Samoa	Pacific/Pago_Pago
Pacific/Tahiti	Pacific/Tahiti
Pacific/Tarawa	Pacific/Tarawa
Pacific/Tongatapu	Pacific/Tongatapu
Pacific/Truk	Pacific/Truk
Pacific/Wake	Pacific/Wake
Pacific/Wallis	Pacific/Wallis
Poland	Europe/Warsaw

시스템 값	**Java** 스트링 오브젝트
Portugal	Europe/Lisbon
ROC	Asia/Taipei
ROK	Asia/Seoul
Singapore	Asia/Singapore
SystemV/AST4ADT	America/Thule
SystemV/CST6CDT	America/Chicago
SystemV/EST5EDT	America/New_York
SystemV/MST7MDT	America/Denver
SystemV/PST8PDT	America/Los_Angeles
SystemV/YST9YDT	America/Anchorage
Turkey	Europe/Istanbul
UCT	UTC
US/Alaska	America/Anchorage
US/Aleutian	America/Adak
US/Arizona	America/Phoenix
US/Central	America/Chicago
US/East-Indiana	America/Indianapolis
US/Eastern	America/New_York
US/Hawaii	Pacific/Honolulu
US/Michigan	America/New_York
US/Mountain	America/Denver
US/Pacific	America/Los_Angeles
US/Pacific-New	America/Los_Angeles
US/Samoa	Pacific/Pago_Pago
UTC	UTC
Universal	UTC
W-SU	Europe/Moscow
WET	Europe/London
Zulu	UTC

Java 로케일

로케일은 동일한 언어와 관습을 공유하는 지리적 또는 정치적 지역입니다. Java(TM)에서 로케일 클래스는 지역을 나타냅니다.

지원되는 Java 로케일

IBM Developer Kit for Java는 다음의 로케일을 지원합니다. iSeries 작업 및 CNTRYID는 디폴트 로케일을 판별합니다. 자세한 내용은 Java 시스템 등록 정보를 참조하십시오.

JDK 1.1.6의 로케일명	**ISO** 로케일명	언어/국가 또는 지역
ar	ar_EG	아랍어/이집트

JDK 1.1.6의 로케일명	ISO 로케일명	언어/국가 또는 지역
be	be_BY	벨라루시어/벨라루시
bg	bg_BG	불가리아어/불가리아
ca	ca_ES	카탈루나어/스페인
cs	cs_CZ	체코어/체코
da	da_DK	덴마크어/덴마크
de	de_DE	독일어/독일
de_AT	de_AT	독일어/오스트리아
de_CH	de_CH	독일어/스위스
el	el_GR	그리스어/그리스
en	en_US	영어/미국
en_AU	en_AU	영어/호주
en_CA	en_CA	영어/캐나다
en_GB	en_GB	영어/영국
en_IE	en_IE	영어/아일랜드
en_NZ	en_NZ	영어/뉴질랜드
en_ZA	en_ZA	영어/남아프리카
es	es_ES	스페인어/스페인
es_AR	es_AR	스페인어/아르헨티나
es_BO	es_BO	스페인어/볼리비아
es_CL	es_CL	스페인어/칠레
es_CR	es_CR	스페인어/코스타리카
es_DO	es_DO	스페인어/도미니크 공화국
es_EC	es_EC	스페인어/에콰도르
es_GT	es_GT	스페인어/과테말라
es_HN	es_HN	스페인어/온두라스
es_MX	es_MX	스페인어/멕시코
es_NI	es_NI	스페인어/니카라과
es_PA	es_PA	스페인어/파나마
es_PE	es_PE	스페인어/페루
es_PR	es_PR	스페인어/푸에르토리코
es_PY	es_PY	스페인어/파라과이
es_SV	es_SV	스페인어/엘살바도르
es_UY	es_UY	스페인어/우루과이
es_VE	es_VE	스페인어/베네주엘라
et	et_EE	에스토니아어/에스토니아
fi	fi_FI	핀란드어/핀란드
fr	fr_FR	프랑스어/프랑스
fr_BE	fr_BE	불어/벨기에
fr_CA	fr_CA	불어/캐나다
fr_CH	fr_CH	불어/스위스
hr	hr_HR	크로아티아어/크로아티아

JDK 1.1.6의 로케일명	ISO 로케일명	언어/국가 또는 지역
hu	hu_HU	헝가리어/헝가리
is	is_IS	아이슬란드어/아이슬란드
it	it_IT	이태리어/이태리
it_CH	it_CH	이태리어/스위스
iw	iw_IL	히브리어/이스라엘
ja	ja_JP	일본어/일본
ko	ko_KR	한국어/한국
lt	lt_LT	리투아니아어/리투아니아
lv	lv_LV	라트비아어/라트비아
mk	mk_MK	마케도니아어/마케도니아
nl	nl_NL	네덜란드어/네덜란드
nl_BE	nl_BE	네덜란드/벨기에
no	no_NO_B	노르웨이어/노르웨이
no_NO_NY	no_NO_NY	노르웨이어/노르웨이, NY
pl	pl_PL	폴란드어/폴란드
pt	pt_PT	포르투칼어/포르투칼
ro	ro_RO	루마니아어/루마니아
ru	ru_RU	러시아어/러시아
sh	sh_SP	세르-크로아티아어/세르비아
sk	sk_SK	슬로바키아어/슬로바키아
sl	sl_SI	슬로베니아어/슬로베니아
sq	sq_AL	알바니아어/알바니아
sr	sr_SP	세르비아어/세르비아
sv	sv_SE	스웨덴어/스웨덴
tr	tr_TR	터키어/터키
uk	uk_UA	우크라이나어/우크라이나
zh	zh_CN	간체 한자
zh_TW	zh_TW	정체 한자

예: java.util.DateFormat 클래스를 사용하여 날짜 국제화: 다음 예는 날짜를 형식화하기 위해 로케일을 사용하는 방법을 보여줍니다.

예 1: 날짜 국제화를 위한 java.util.DateFormat 클래스 사용의 예

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

자세한 정보는 국제화된 Java(TM) 프로그램 작성을 참조하십시오.

예: java.util.NumberFormat 클래스를 사용하여 숫자 표시 국제화: 다음 예는 숫자를 형식화하기 위해 로케일을 사용하는 방법을 보여줍니다.

예 1: 숫자 표시 국제화를 위한 java.util.NumberFormat 클래스 사용의 예

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

자세한 정보는 국제화된 Java(TM) 프로그램 작성을 참조하십시오.

예: java.util.ResourceBundle 클래스를 사용하여 로케일 특정 자료 국제화: 다음 예는 프로그램 스트링을 국제화하기 위해 자원 번들과 함께 로케일을 사용할 수 있는 방법을 보여줍니다.

이 등록 정보 파일은 다음과 같은 작업을 위해 ResourceBundleExample 프로그램에 필요합니다.

RBExample.properties의 내용

Hello.text=Hello

RBExample_de.properties의 내용

Hello.text=Guten Tag

RBExample_fr_FR.properties의 내용

Hello.text=Bonjour

예 1: 로케일 특정 자료 국제화를 위한 java.util.ResourceBundle 클래스 사용의 예

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

자세한 정보는 국제화된 Java(TM) 프로그램 작성을 참조하십시오.

Java 문자 코드화

내부적으로, JVM(Java(TM) virtual machine)은 항상 유니코드 자료로 작성합니다. 그러나 JVM의 안팎으로 전송되는 모든 자료의 형식은 file.encoding 등록 정보와 일치합니다. JVM 안으로 읽히는 자료는 file.encoding에서 유니코드로 변환되며 JVM으로부터 전송되는 자료는 유니코드에서 file.encoding으로 변환됩니다.

Java 프로그램을 위한 자료 파일은 통합 파일 시스템에 저장됩니다. 통합 파일 시스템의 파일에는 파일에 포함된 자료의 문자 코드화를 식별하는 코드화 문자 세트 ID(CCSID)가 태그로 표시됩니다. file.encoding과 iSeries 서버에 있는 CCSID의 상호 연관성에 관해서는 File.encoding 값 및 iSeries CCSID 표를 참조하십시오.

자료가 Java 프로그램에 의해 읽히는 경우, file.encoding과 일치하는 문자 코드화로 이루어지는 것으로 예상됩니다. 자료가 Java 프로그램에 의해 파일에 기록되는 경우 file.encoding과 일치하는 문자 코드화로 기록됩니다. 이 점은 javac 명령에 의해 처리되는 Java 소스 코드 파일(.java 파일)과 .net 패키지를 사용하여 TCP/IP를 통해 송신 및 수신되는 자료에도 적용됩니다.

System.in, System.out 및 System.err에서 읽거나 여기에 기록되는 자료는 stdin, stdout 및 stderr에 지정될 때 다른 소스에서 읽거나 여기에 기록되는 자료와 다르게 처리됩니다. stdin, stdout 및 stderr은 일반적으로 iSeries 서버의 EBCDIC 장치에 접속되므로 file.encoding의 일반 문자 코드화에서 iSeries 작업 CCSID와 일치하는 CCSID로 변환시키기 위해 자료에 대해 JVM이 변환을 수행합니다. System.in, System.out 또는 System.err의 방향이 파일이나 소켓으로 재지정되고 stdin, stdout 또는 stderr로 방향이 지정되지 않은 경우, 이와 같은 추가 자료 변환이 수행되지 않으며 file.encoding과 일치하는 문자 코드화로 자료가 남습니다.

자료를 Java 프로그램과의 사이에서 file.encoding과 다른 문자코드화로 읽거나 기록해야 하는 경우 프로그램이 Java IO 클래스 java.io.InputStreamReader, java.io.FileReader, java.io.OutputStreamReader 및 java.io.FileWriter를 사용할 수 있습니다. 이러한 Java 클래스들은 현재 JVM이 사용 중인 디폴트 file.encoding 등록 정보에 우선하여 file.encoding 값을 지정할 수 있게 해줍니다.

JDBC API를 통한 DB2/400 데이터베이스로 들어가는 자료나 이 데이터베이스로부터 나오는 자료는 iSeries 데이터베이스의 CCSID로 또는 그 CCSID로부터 변환됩니다.

Java 고유 인터페이스를 통해 다른 프로그램으로 또는 다른 프로그램으로부터 전송되는 자료는 변환되지 않습니다.

국제화에 대한 자세한 정보는 OS/400 국제화를 참조하십시오.

또는 Sun Microsystems, Inc.의 국제화를 참조하십시오.

File.encoding 값 및 iSeries CCSID: 다음 표는 가능한 file.encoding 값과 iSeries 코드화 문자세트 ID(CCSID)에 대응하는 근사값 사이의 관계를 나타냅니다.

file.encoding	CCSID	설명
Big5	950	8비트 ASCII 정체 한자 BIG-5
CNS11643	964	정체 한자를 위한 한자 문자 세트
Cp037	037	IBM EBCDIC US, 캐나다, 네덜란드,...
Cp273	273	IBM EBCDIC 독일, 오스트리아
Cp277	277	IBM EBCDIC 덴마크, 노르웨이
Cp278	278	IBM EBCDIC 핀란드, 스웨덴
Cp280	280	IBM EBCDIC 이탈리아
Cp284	284	IBM EBCDIC 스페인, 라틴 아메리카
Cp285	285	IBM EBCDIC 영국
Cp297	297	IBM EBCDIC 프랑스
Cp420	420	IBM EBCDIC 아랍어
Cp424	424	IBM EBCDIC 헤브리어
Cp437	437	8비트 ASCII 미국 PC
Cp500	500	IBM EBCDIC 국제
Cp737	737	8비트 ASCII 그리스 MS-DOS
Cp775	775	8비트 ASCII 발트어 MS-DOS
Cp838	838	IBM EBCDIC 태국
Cp850	850	8비트 ASCII 라틴-1 다국적
Cp852	852	8비트 ASCII 라틴-2
Cp855	855	8비트 ASCII 키릴 문자
Cp856	856	8비트 ASCII 헤브리어
Cp857	857	8비트 ASCII 라틴-5
Cp860	860	8비트 ASCII 포르투칼
Cp861	861	8비트 ASCII 아이슬랜드
Cp862	862	8비트 ASCII 히브리어
Cp863	863	8비트 ASCII 캐나다
Cp864	864	8비트 ASCII 아랍어
Cp865	865	8비트 ASCII 덴마크, 노르웨이
Cp866	866	8비트 ASCII 키릴 문자
Cp868	868	8비트 ASCII 우르두말(Urdu)
Cp869	869	8비트 ASCII 그리스
Cp870	870	IBM EBCDIC 라틴-2
Cp871	871	IBM EBCDIC 아이슬란드
Cp874	874	8비트 ASCII 태국
Cp875	875	IBM EBCDIC 그리스
Cp918	918	IBM EBCDIC 우르두말(Urdu)
Cp921	921	8비트 ASCII 발트어
Cp922	922	8비트 ASCII 에스토니아
Cp930	930	IBM EBCDIC 일본 확장 가다가나(katakana)
Cp933	933	IBM EBCDIC 한국어
Cp935	935	IBM EBCDIC 간체 한자

file.encoding	CCSID	설명
Cp937	937	IBM EBCDIC 정체 한자
Cp939	939	IBM EBCDIC 일본 확장 라틴
Cp942	942	8비트 ASCII 일본어
Cp943	943	개방 환경을 위해 혼합된 일본 PC 자료
Cp943C	943	개방 환경을 위해 혼합된 일본 PC 자료
Cp948	948	8비트 ASCII IBM 정체 한자
Cp949	949	8비트 ASCII 한국어 KSC5601
Cp950	950	8비트 ASCII 정체 한자 BIG-5
Cp964	964	EUC 정체 한자
Cp970	970	EUC 한국어
Cp1006	1006	ISO 8비트 우르두말(Urdu)
Cp1025	1025	IBM EBCDIC 키릴 문자
Cp1026	1026	IBM EBCDIC 터키어
Cp1046	1046	8비트 ASCII 아랍어
Cp1097	1097	IBM EBCDIC Farsi
Cp1098	1098	8비트 ASCII Farsi
Cp1112	1112	IBM EBCDIC 발트어
Cp1122	1122	IBM EBCDIC 에스토니아
Cp1123	1123	IBM EBCDIC 우크라이나
Cp1124	1124	ISO 8비트 우크라이나
Cp1250	1250	MS-Win 라틴-2
Cp1251	1251	MS-Win 키릴 문자
Cp1252	1252	MS-Win 라틴-1
Cp1253	1253	MS-Win 그리스
Cp1254	1254	MS-Win 터키어
Cp1255	1255	MS-Win 헤브리어
Cp1256	1256	MS-Win 아랍어
Cp1257	1257	MS-Win 발트어
Cp1258	1251	MS-Win 러시아어
Cp1381	1381	8비트 ASCII 간체 한자 GB
Cp1383	1383	EUC 간체 한자
Cp33722	33722	EUC 일본어
EUC_CN	1383	간체 한자를 위한 EUC
EUC_JP	33722	일본어를 위한 EUC
EUC_KR	970	한국어를 위한 EUC
EUC_TW	964	정체 한자를 위한 EUC
GB2312	1381	8비트 ASCII 간체 한자 GB
GBK	1386	새로운 간체 한자 8비트 ASCII 9
ISO2022CN_CNS	사용 가능한 것이 없음	정체 한자를 위한 7비트 ASCII
ISO2022CN_GB	사용 가능한 것이 없음	간체 한자를 위한 7비트 ASCII
ISO2022JP	5054	일본어를 위한 7비트 ASCII

file.encoding	CCSID	설명
ISO2022KR	25546	한국어를 위한 7비트 ASCII
ISO8859_1	819	ISO 8859-1 ISO 라틴-1
ISO8859_2	912	ISO 8859-2 ISO 라틴-2
ISO8859_3	913	ISO 8859-3 ISO 라틴-3
ISO8859_4	914	ISO 8859-4 ISO 라틴-4
ISO8859_5	915	ISO 8859-5 ISO 라틴-5
ISO8859_6	1089	ISO 8859-6 ISO 라틴-6(아랍어)
ISO8859_7	813	ISO 8859-7 ISO 라틴-7(그리스/라틴)
ISO8859_8	916	ISO 8859-8 ISO 라틴-8(헤브리어)
ISO8859_9	920	ISO 8859-9 ISO 라틴-9(ECMA-128, 터어키)
JIS0201	897	일본 산업 표준 X0201
JIS0208	952	일본 산업 표준 X0208
JIS0212	953	일본 산업 표준 X0212
Johab	사용 가능한 것이 없음	한국 조합형 한글 암호화(전체)
K018_R		키릴 문자
KSC5601	949	8비트 ASCII 한국어
MS874	874	MS-Win 태국
SJIS	932	8비트 ASCII 일본어
TIS620	874	타이 산업 표준 620
UTF8	1208	UTF-8(iSeries에서 아직 사용할 수 없는 IBM CCSID 1208)
유니코드	13488	UNICODE, UCS-2
UnicodeBig	13488	유니코드와 동일
UnicodeBigUnmarked		바이트 순서 표시를 갖지 않는 유니코드
UnicodeLittle		little-endian 바이트 순서를 갖는 유니코드
UnicodeLittleUnmarked		바이트 순서 표시를 갖지 않는 UnicodeLittle

디폴트 값에 대해서는 file.encoding 디폴트 값을 참조하십시오.

디폴트 file.encoding 값: 이 표는 JVM(Java(TM) virtual machine)이 시작할 때 iSeries 코드화 문자 세트 ID(CCSID)를 기반으로 file.encoding 값이 어떻게 설정되는 지 표시합니다.

iSeries CCSID	디폴트 file.encoding	설명
37	ISO8859_1	미국, 캐나다, 뉴질랜드 및 호주의 영어; 포르투칼 및 브라질의 포루투칼어; 네덜란드의 독일어
256	ISO8859_1	인터내셔널 #1
273	ISO8859_1	독일어/독일, 독일어/오스트리아
277	ISO8859_1	덴마크어/덴마크, 노르웨이어/노르웨이, 노르웨이어/노르웨이, NY
278	ISO8859_1	핀란드어/핀란드
280	ISO8859_1	이태리어/이태리
284	ISO8859_1	카탈로니아어/스페인, 스페인어/스페인

iSeries CCSID	디폴트 file.encoding	설명
285	ISO8859_1	영어/영국, 영어/아일랜드
290	Cp943C	일본 EBCDIC 혼합의 SBCS 부분 (CCSID 5026)
297	ISO8859_1	프랑스어/프랑스
420	Cp1046	아랍어/이집트
423	ISO8859_7	그리스
424	ISO8859_8	히브리어/이스라엘
500	ISO8859_1	독일어/스위스, 프랑스어/벨기에, 프랑스어/캐나다, 프랑스어/스위스
833	Cp970	한국 EBCDIC 혼합의 SBCS 부분 (CCSID 933)
836	Cp1383	간체 한자 EBCDIC 혼합의 SBCS 부분 (CCSID 935)
838	TIS620	태국어
870	ISO8859_2	체코어/체코 공화국, 크로아티아어/크로와티아, 헝가리어/헝가리 폴란드어/폴란드
871	ISO8859_1	아이슬란드어/아이슬란드
875	ISO8859_7	그리스어/그리스
880	ISO8859_5	불가리아(ISO 8859_5)
905	ISO8859_9	확장 터키
918	Cp868	우르두말(Urdu)
930	Cp943C	일본 EBCDIC 혼합(CCSID 5026과 유사함)
933	Cp970	한국어/한국
935	Cp1383	간체 한자
937	Cp950	정체 한자
939	Cp943C	일본 EBCDIC 혼합(CCSID 5035와 유사함)
1025	ISO8859_5	벨로루스어/벨라루스, 불가리아어/불가리아, 마케도니아어/마케도니아, 러시아어/러시아
1026	ISO8859_9	터키어/터키
1027	Cp943C	일본 EBCDIC 혼합의 SBCS 부분 (CCSID 5035)
1097	Cp1098	Farsi
1112	Cp921	리투아니아어/리투아니아, 라트비아어/라트비아, 발트어
1388	GBK	간체 한자 EBCDIC 혼합(GBK가 포함됨)
5026	Cp943C	일본 EBCDIC 혼합 CCSID(확장 가다가나)
5035	Cp943C	일본 EBCDIC 혼합 CCSID(확장 라틴)
8612	Cp1046	아랍어(기본 양식만)(또는 ASCII 420 및 8859_6)
9030	Cp874	태국어(호스트 확장 SBCS)

iSeries CCSID	디폴트 file.encoding	설명
13124	GBK	간체 한자 EBCDIC 혼합의 SBCS 부분 (GBK가 포함됨)
28709	Cp948	정체 한자 EBCDIC 혼합의 SBCS 부분 (CCSID 937)

릴리스 간 호환성

Java(TM) 클래스 파일은 Sun이 제거했거나 지원을 변경한(Sun 문서 참조) 몇 개의 피처를 사용하지 않는 한 상향 호환이 가능합니다.

(JDK 1.1.x -> 1.2.x -> 1.3.x -> 1.4.x)

클래스 파일은 이전 JDK 레벨에서 사용 가능한 Java 피처만을 사용하는 한 하향 호환도 가능합니다.

(1.4.x -> 1.3.x -> 1.2.x -> 1.1.x)

이 점은 iSeries 서버에도 해당되는데, 그것은 그대로 구현되기 때문입니다. 릴리스 간 가용성에 대한 정보는 The Source for Java Technology java.sun.com 을 참조하십시오.

iSeries 서버의 Java 프로그램이 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하여 최적화된 경우 JVAPGM이 클래스 파일에 첨부됩니다. 이러한 JVAPGM의 내부 구조가 V4R4에서 변경되었습니다. 이것은 V4R4 이전에 작성된 JVAPGM이 V4R4 이후의 릴리스에서 유효하지 않음을 의미합니다. 따라서 다시 작성해야 합니다. 아무 조치도 취하지 않으면 시스템은 이전과 동일한 최적화 레벨로 JVAPGM을 자동 작성합니다. 특히 JAR 또는 ZIP 파일의 경우에는 CRTJVAPGM을 수동으로 수행하는 것이 좋습니다. 그러면, 프로그램 크기가 가장 작은 최적화 상태가 됩니다.

최적화 레벨 40에서 최적의 성능을 위해 각 OS/400 릴리스에서 CRTJVAPGM을 수행하는 것이 좋으며 그렇지 않을 경우 JDK 버전이 변경됩니다. 이 점은 JDKVER 기능이 CRTJVAPGM에서 사용되는 경우에 특히 해당되며 Sun JDK의 메소드가 JVAPGM으로 인라인되기 때문입니다. 이것은 성능을 현저히 향상시킬 수 있습니다. 그러나 이러한 인라인을 무효화하는 후속 릴리스의 JDK에서 변경이 이루어진 경우 프로그램은 실제로 더 낮은 최적화 레벨보다 느리게 실행될 수 있습니다. 이것은 적합한 작업을 구하기 위해 특별 케이스 코드를 실행해야 하기 때문입니다.

성능 정보에 대한 자세한 내용은 Java 런타임 성능을 참조하십시오.

IBM Developer Kit for Java를 사용한 데이터베이스 액세스

IBM Developer Kit for Java(TM)를 사용할 경우, Java 프로그램은 3가지 방식으로 사용자의 데이터베이스 파일에 액세스할 수 있습니다.

- JDBC 드라이버는 IBM Developer Kit for Java JDBC 드라이버를 사용하여 Java 프로그램이 데이터베이스 파일에 액세스하는 방법을 설명합니다.
- SQLJ 지원은 IBM Developer Kit for Java를 사용하여 Java 어플리케이션의 삽입 SQL문을 사용하는 방법을 설명합니다.
- 

  Java SQL 루틴에서는 Java 저장 프로시듀어 및 Java 사용자 정의 기능을 사용하여 Java 프로그램에 액세스하는 방법에 대해 설명합니다.

IBM Developer Kit for Java JDBC 드라이버로 iSeries 데이터베이스에 액세스

"원시" 드라이버라고도 하는 IBM Developer Kit for Java(TM) JDBC 드라이버는 iSeries 데이터베이스 파일에 대한 프로그래밍 액세스를 제공합니다. Java 언어로 기록된 어플리케이션인 JDBC(Java Database Connectivity) API를 사용하여 내장 SQL(구조화 조회 언어)로 JDBC 데이터베이스 기능에 액세스하고, SQL 문을 실행하고, 결과를 검색하고, 변경사항을 다시 데이터베이스에 전달할 수 있습니다.

JDBC API를 사용하면 분배된 이종 환경에서 여러 자료 소스와 대화할 수도 있습니다.

JDBC API의 기초가 되는 SQL99 CLI(Command Language Interface)는 ODBC의 기초입니다. JDBC는 Java 프로그래밍 언어에서 SQL 표준에 정의된 추상 및 개념으로의 자연스럽고 사용하기 쉬운 맵핑을 제공합니다.

JDBC 드라이버를 사용하려면 다음을 참조하십시오.

JDBC 시작하기
JDBC 프로그램 작성 및 iSeries 서버에서의 실행에 대한 학습을 따르면 됩니다.

연결
어플리케이션 프로그램에는 한 번에 여러 연결이 있을 수 있습니다. Connection 오브젝트를 사용하여 JDBC에서 자료 소스에 대한 연결을 나타낼 수 있습니다. 데이터베이스에 대해 SQL문을 처리하기 위해 Statement 오브젝트는 연결 오브젝트를 통해 작성됩니다.

DatabaseMetaData
DatabaseMetaData 인터페이스는 자료 소스와 대화하는 방법을 판별하기 위해 어플리케이션 서버와 툴이 사용합니다. 어플리케이션이 특정 자료 소스에 대한 정보를 얻기 위해 DatabaseMetaData 메소드를 사용합니다.

예외

Java 언어는 예외를 사용하여 프로그램에 대한 오류 처리 기능을 제공합니다. 예외는 명령의 정상적인 흐름을 방해하는 프로그램을 실행하면 발생하는 이벤트입니다.

트랜잭션

트랜잭션은 논리적 작업 단위입니다. 트랜잭션은 자료 무결성을 제공하고 어플리케이션 의미론을 정정하고 동시 액세스 중에 자료에 대한 일관적인 관점을 제공하기 위해 사용됩니다. 모든 JDBC 준수 드라이버는 트랜잭션을 지원해야 합니다.

명령문 유형

Statement 인터페이스와 해당 PreparedStatement 및 CallableStatement 서브클래스를 사용하여 데이터베이스에 대해 SQL 명령을 처리합니다. SQL문으로 인해 ResultSet 오브젝트가 생성됩니다.

ResultSets

ResultSet 인터페이스는 조회를 실행하여 생성된 결과에 대한 액세스를 제공합니다. ResultSet의 자료는 일정한 수의 열과 일정한 수의 행이 있는 표로 간주할 수 있습니다. 디폴트로 표의 행이 순차적으로 검색됩니다. 한 행 내에서 열 값은 임의 순서로 액세스할 수 있습니다.

JDBC 오브젝트 풀

Connection, Statement 및 ResultSet 오브젝트와 같이 JDBC에서 사용하는 많은 오브젝트는 작성하는 비용이 많이 들기 때문에 JDBC 오브젝트 풀을 사용하면 상당한 성능 이점이 있을 수 있습니다. 오브젝트 풀을 사용하면 이러한 오브젝트를 필요할 때마다 작성하는 대신 재사용할 수 있습니다.

일괄처리 갱신

일괄처리 갱신 지원을 사용하면 사용자 프로그램과 데이터베이스 사이에서 데이터베이스에 대한 여러 갱신사항을 단일 트랜잭션으로 전달할 수 있습니다. 일괄처리 갱신은 한 번에 많은 갱신을 수행해야 할 때 성능을 상당히 향상시킬 수 있습니다.

확장 자료 유형

iSeries 데이터베이스에 제공되는 SQL3 자료 유형이라고 하는 몇 가지 새로운 자료 유형이 있습니다. SQL3 자료 유형은 엄청난 유연성을 제공합니다. 일련화된 Java 오브젝트, XML(Extensible Markup Language) 문서 및 노래, 제품 그림, 직원 사진 및 영화 클립과 같은 멀티미디어 자료 저장에 이상적입니다. SQL3 자료 유형에는 다음의 내용이 들어 있습니다.

- 고유 유형
- 큰 2진 오브젝트, 큰 문자 오브젝트 및 2바이트 큰 문자 오브젝트와 같은 큰 오브젝트
- Datalink

RowSet

RowSet 스펙은 실제 구현보다 구조에 충실하기 위한 것입니다. RowSet 인터페이스는 모든 RowSet가 가지고 있는 핵심 기능 세트를 정의합니다.

분배 트랜잭션
JTA(Java Transaction API)에는 복잡한 트랜잭션에 대한 지원이 있습니다. Connection 오브젝트로부터 커플링을 해제하는 지원도 제공합니다. JTA 및 JDBC는 함께 작업하여 Connection 오브젝트로부터 트랜잭션의 커플링을 해제하고 동시에 여러 트랜잭션에 대한 단일 연결 작업을 수행할 수 있게 합니다. 반대로 단일 트랜잭션에서 여러 연결 작업을 수행할 수 있습니다.

성능 추가 정보
다음의 성능 추가 정보를 사용하면 JDBC 어플리케이션에서 가능한한 가장 적합한 성능을 확보할 수 있습니다.

JDBC에 대한 자세한 정보는 Sun Microsystems, Inc.의 JDBC 문서를 참조하십시오.

iSeries 원시 JDBC 드라이버에 대한 자세한 정보는 IBM Developer Kit for Java JDBC 웹 페이지를 참조하십시오.

JDBC 시작하기

Developer Kit와 함께 제공된 JDBC(Java(TM) Database Connectivity) 드라이버를 Developer Kit for Java JDBC 드라이버라고 합니다. 이 드라이버를 흔히 원시 JDBC 드라이버라고도 합니다.

사용자의 필요에 적합한 JDBC 드라이버를 선택하려면 다음의 제안사항을 고려하십시오.

- 데이터베이스가 있는 서버에서 직접 실행하는 프로그램은 성능을 위해 원시 JDBC 드라이버를 사용해야 합니다. 여기에는 대부분의 서브릿 및 JSP(JavaServer Pages)와 iSeries 서버에서 로컬로 실행하도록 기록된 어플리케이션이 들어 있습니다.
- 리모트 iSeries 서버에 연결해야 하는 프로그램은 Toolbox JDBC 드라이버를 사용합니다. Toolbox는 JDBC를 확실하게 구현한 것이며 Toolbox for Java의 일부로 제공됩니다. 순수한 Java가 되기 위해 Toolbox JDBC 드라이버는 클라이언트를 설정하기 위한 것일 뿐이며 서버 설정이 거의 필요하지 않습니다.
- iSeries 서버에서 실행하며 리모트인 비 iSeries 데이터베이스에 연결해야 하는 프로그램은 원시 JDBC 드라이버를 사용하고 이 리모트 서버에 대해 DRDA(Distributed Relational Database Architecture) 연결을 설정합니다.

JDBC를 시작하려면 다음을 참조하십시오.

JDBC 드라이버의 유형
이 주제는 JDBC 드라이버 유형을 정의합니다. 드라이버 유형을 사용하여 데이터베이스에 연결하는데 사용하는 기술을 분류합니다.

요구사항

이 주제는 다음에 액세스하는 데 필요한 요구사항을 나타냅니다.

- 핵심 JDBC
- JDBC 2.0 선택적 패키지
- JTA(Java Transaction API)

JDBC 학습

JDBC 프로그램을 작성하고 원시 JDBC 드라이버를 사용하여 iSeries 서버에서 이 프로그램을 실행하기 위한 중요한 첫 번째 단계입니다.

JDBC 드라이버의 유형:

이 주제에서는 JDBC(Java(TM) Database Connectivity) 드라이버 유형을 정의합니다. 드라이버 유형을 사용하여 데이터베이스에 연결하는데 사용하는 기술을 분류합니다. JDBC 드라이버 벤더는 이러한 유형을 사용하여 제품의 작동 방법을 설명합니다. 일부 JDBC 드라이버 유형은 다른 어플리케이션보다 일부 어플리케이션에 적합합니다.

유형 1: 유형 1 드라이버는 "브릿지" 드라이버입니다. ODBC(Open Database Connectivity)와 같은 다른 기술을 사용하여 데이터베이스와 통신합니다. ODBC 드라이버는 많은 RDBMS(Relational Database Management System) 플랫폼을 위해 존재하기 때문에 이 점은 장점이라고 할 수 있습니다. JNI(Java Native Interface)를 사용하여 JDBC 드라이버로부터 ODBC 함수를 호출합니다.

JDBC를 함께 사용할 수 있으려면 먼저 유형 1 드라이버는 브릿지 드라이버를 설치하고 구성해야 합니다. 이것은 기간계 어플리케이션에 대한 심각한 장애가 될 수 있습니다. 애플릿은 원시 코드를 로드할 수 없기 때문에 유형 1 드라이버를 애플릿에서 사용할 수 없습니다.

유형 2: 유형 2 드라이버는 원시 API를 사용하여 데이터베이스 시스템과 통신합니다. Java 원시 메소드를 사용하여 데이터베이스 조작을 수행하는 API 함수를 호출합니다. 유형 2 드라이버는 일반적으로 유형 1 드라이버보다 속도가 빠릅니다.

유형 2 드라이버를 작동시키려면 원시 2진 코드를 설치하고 구성해야 합니다. 유형 2 드라이버도 JNI를 사용합니다. 애플릿은 원시 코드를 로드할 수 없기 때문에 애플릿에서 유형 2 드라이버를 사용할 수 없습니다. 유형 2 JDBC 드라이버를 사용하려면 일부 DBMS(Database Management System) 네트워킹 소프트웨어를 설치해야 합니다.

Developer Kit for Java JDBC 드라이버는 유형 2 JDBC 드라이버입니다.

유형 3: 이러한 드라이버는 네트워킹 프로토콜과 미들웨어를 사용하여 서버와 통신합니다. 그러면 서버는 프로토콜을 DBMS에 고유한 DBMS 기능 호출로 변환합니다.

유형 3 JDBC 드라이버는 클라이언트에 원시 2진 코드를 설치할 것을 요구하지 않기 때문에 가장 유연한 JDBC 솔루션입니다. 유형 3 드라이버에는 클라이언트 설치가 필요하지 않습니다.

유형 4: 유형 4 드라이버는 Java를 사용하여 DBMS 벤더 네트워킹 프로토콜을 구현하지 않습니다. 프로토콜은 대개 독점적이므로 DBMS 벤더는 일반적으로 유형 4 JDBC 드라이버를 제공하는 유일한 회사입니다.

유형 4 드라이버는 모두 Java 드라이버입니다. 이는 클라이언트 설치나 구성이 없음을 의미합니다. 그러나 기초 프로토콜이 보안 및 네트워크 연결과 같은 문제를 올바로 처리하지 못하면 유형 4 드라이버가 일부 어플리케이션에 적합하지 않을 수 있습니다.

Toolbox JDBC 드라이버는 유형 4 JDBC 드라이버이며 API가 순수한 Java 네트워킹 프로토콜 드라이버임을 나타냅니다.

JDBC 요구사항:

JDBC 어플리케이션을 작성하고 배치하기 전에 다음을 설치해야 합니다.

- 『핵심 JDBC』
- 『JDBC 2.0 선택적 패키지』
- 62 페이지의 『Java 트랜잭션 API』

핵심 JDBC: 로컬 데이터베이스로의 핵심 JDBC(Java(TM) Database Connectivity) 액세스의 경우, 요구사항이 없습니다. 모든 지원이 내장되었으며 이미 설치되었고 구성되었습니다.

주:

- 과거에는 JDK 1.2 이상의 경우 기호 링크를 추가해야 했습니다. 이 요구사항은 V4R5 PTF SF65439 및 V5R1 PTF SI00959를 사용하여 제거했습니다.
- 과거에는 시스템에 연결하기 위해 관계 데이터베이스 디렉토리에 최소한 한 항목(일반적으로 로컬 데이터베이스에 연결하기 위한 *LOCAL)이 있는지 확인해야 했습니다. 이 요구사항도 제거했습니다. 로컬 데이터베이스에 대한 항목이 없는 경우에 로컬 시스템에 액세스할 때 시스템의 이름을 사용하여 항목을 작성합니다. 관계 데이터베이스 디렉토리 항목은 원시 JDBC 드라이버를 사용하여 연결하려는 리모트 시스템에 맞게 계속 구성해야 합니다.

JDBC 2.0 선택적 패키지: JDBC 2.0 선택적 패키지의 클래스를 사용해야 하는 경우에는 클래스 경로에 jdbc2_0-stdext.jar 파일을 포함시켜야 합니다. 이 JAR(Java ARchive) 파일에는 JDBC 2.0 선택적 패키지를 사용하기 위해 어플리케이션을 기록하는 데 필요한 모든 표준 인터페이스가 들어 있습니다. 확장 클래스 경로에 JAR 파일을 추가하려면 UserData 확장 디렉토리에서 JAT 파일의 위치로 기호 링크를 작성하십시오. 이 작업은 한 번만 수행해야 합니다. JDBC 2.0 선택적 패키지 JAT 파일은 런타임 시에 어플리케이션이 항상 사용할 수 있습니다. 다음의 명령을 사용하여 확장 클래스 경로에 선택적 패키지를 추가하십시오.

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

주: 이 요구사항은 JDK 1.2와 1.3만을 위한 것입니다. JDK 1.4는 JDBC 3.0 지원이 제공되는 첫 번째 릴리스이므로 모든 JDBC(즉, 핵심 JDBC와 선택적 패키지)는 프로그램이 항상 찾는 기본 JDK 런타임 JAR 파일로 이동합니다.

Java 트랜잭션 API: 어플리케이션에서 JTA(Java Transaction API)를 사용해야 하는 경우에는 클래스 경로에 jta-spec1_0_1.jar 파일을 포함시켜야 합니다. 이 JAR 파일에는 JTA를 사용하기 위해 어플리케이션을 기록하는 데 필요한 모든 표준 인터페이스가 들어 있습니다. 확장 클래스 경로에 JAR 파일을 추가하려면 UserData 확장 디렉토리에 JAR 파일의 위치로 기호 링크를 작성하십시오. 이 작업은 한 번만 가능하며 일단 완료했으면 어플리케이션이 런타임 시에 JTA JAR 파일을 항상 사용할 수 있습니다. 다음의 명령을 사용하여 확장 클래스 경로에 JTA를 추가하십시오.

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

JDBC 준수: 원시 JDBC 드라이버는 모든 관련 JDBC 스펙을 준수합니다. JDBC 드라이버의 준수 레벨은 OS/400 릴리스가 아닌 사용하는 JDK 릴리스에 따라 좌우됩니다. 다양한 JDK에 대한 원시 JDBC 드라이버의 준수 레벨은 다음과 같습니다.

JDK 릴리스	**JDBC 드라이버의 준수 레벨**
JDK 1.1	이 JDK는 JDBC 1.0을 준수합니다.
JDK 1.2	이 JDK는 JDBC 2.0을 준수하며 JDBC 2.1 선택적 패키지를 지원합니다.
JDK 1.3	이 JDK는 JDBC 2.0을 준수하며 JDBC 2.1 선택적 패키지를 지원합니다(JDK 1.3에서는 JDBC 관련 변경이 없었습니다).
JDK 1.4	이 JDK는 JDBC 3.0을 준수하지만 JDBC 선택적 패키지가 더 이상 없습니다(이에 대한 지원은 현재 핵심 JDK의 일부입니다).

JDBC 학습:

다음은 JDBC(Java(TM) Database Connectivity) 프로그램을 작성하고 이를 원시 JDBC 드라이버와 함께 iSeries 서버에서 실행하기 위한 학습서입니다. 사용자의 프로그램이 JDBC를 실행하는 데 필요한 기본 단계를 보여주기 위한 것입니다.

프로그램 예는 표를 작성하고 이 표를 일부 자료로 채웁니다. 프로그램은 조회를 처리하여 데이터베이스로부터 이 자료를 얻고 화면에 표시합니다.

예 프로그램 실행: 예 프로그램을 실행하려면 다음 단계를 수행하십시오.

1. 워크스테이션으로 프로그램을 복사하십시오.
   a. 프로그램 예를 복사하여 워크스테이션의 파일에 붙여넣으십시오.

b. 제공된 공용 클래스와 동일한 이름과 .java 확장자를 사용하여 파일을 저장하십시오. 이 경우에는 로컬 워크스테이션에서 파일의 이름을 BasicJDBC.java로 지정해야 합니다.

2. 워크스테이션에서 iSeries 서버로 파일을 전송하십시오.명령 프롬트에서 다음 명령을 입력하십시오.

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

이러한 명령을 실행하려면 파일을 넣을 디렉토리가 있어야 합니다. 예에서 /home/cujo는 위치이지만 원하는 위치를 사용할 수 있습니다.

주: 앞에서 언급한 FTP 명령은 서버 설정 방법에 따라 다를 수는 있지만 비슷합니다. 파일을 통합 파일 시스템에 전송하는 한 파일을 iSeries 서버에 전송하는 방법은 문제가 되지 않습니다. VisualAge for Java와 같은 툴은 이러한 프로세스를 완전히 자동화할 수 있습니다.

3. 파일을 넣는 디렉토리로 클래스 경로를 설정하여 Java 명령을 실행하면 Java 명령이 이 파일을 찾을 수 있도록 하십시오. CL 명령에서 WRKENVVAR을 사용하여 사용자 프로파일에 대해 설정된 환경 변수를 확인할 수 있습니다.
   - 이름이 CLASSPATH인 환경 변수가 표시되었으면 .java 파일을 넣는 위치가 여기에 나열된 디렉토리의 스트링에 있는지 확인하거나 위치가 지정되지 않았으면 추가해야 합니다.
   - CLASSPATH 환경 변수가 없으면 추가해야 합니다. 다음의 명령을 사용하여 추가할 수 있습니다.

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

주: CL 명령에서 Java 코드를 컴파일하려면 tools.jar 파일을 포함시켜야 합니다. 이 JAR 파일은 javac 명령을 포함합니다.

4. Java 파일을 클래스 파일로 컴파일하십시오.
   CL 명령행에서 다음 명령을 입력하십시오.

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

QSH에서 Java 파일을 컴파일할 수도 있습니다.

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

QSH는 tools.jar 파일을 찾을 수 있는지 자동으로 확인합니다. 따라서 클래스 경로에 추가할 필요가 없습니다. 현재 디렉토리는 classpath에 있습니다. 디렉토리 변경(cd) 명령을 발행하여 BasicJDBC.java 파일도 찾을 수 있습니다.

주: 워크스테이션에서 파일을 컴파일하고 FTP를 사용하여 2진 모드로 iSeries 서버에 클래스 파일을 보낼 수 있습니다. 다음은 어느 플랫폼에서나 실행할 수 있는 Java의 능력을 나타낸 예입니다. CL 명령행이나 QSH에서 다음의 명령을 사용하여 프로그램을 실행하십시오.

```
java BasicJDBC
```

출력은 다음과 같습니다.

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

참조: Java 및 JDBC에 대한 자세한 정보는 다음의 자원을 문의하십시오.

- Native JDBC driver external 웹 사이트
- Toolbox JDBC driver external 웹 사이트
- Sun의 JDBC 페이지
- iSeries 및 iSeries 사용자를 위한 Java/JDBC 포럼
- IBM JDBC 전자 우편 주소

«

예: JDBC:

»

다음은 BasicJDBC 프로그램을 사용하는 방법을 보여주는 예입니다.

예: BasicJDBC

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
////////////////////////////////////////////////////////////////////////////
//
// 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 the
        // way it should. 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();
        }
    }
}
```

예로 JNDI 사용:

DataSources는 JNDI(Java(TM) Naming and Directory Interface)와 함께 작업합니다. JNDI는 JDBC(Java Database Connectivity)가 데이터베이스에 대한 추상 계층인 것처럼 디렉토리 서비스에 대한 Java 추상 계층입니다. JNDI는 LDAP(Lightweight Directory Access Protocol)과 함께 가장 자주 사용하지만 COS(CORBA Object Services), Java RMI(Java Remote Method Invocation) 레지스트리 또는 기초 파일 시스템과 함께 사용할 수도 있습니다. 이러한 다양한 사용은 공통 JNDI 요구를 특정 디렉토리 서비스 요구로 변환하는 다양한 디렉토리 서비스 제공자를 통해 수행합니다.

DataSource 샘플은 JNDI 파일 시스템 서비스 제공자를 사용하여 설계되었습니다. 제공된 예를 실행하려면 JNDI 서비스 제공자가 제 위치에 있어야 합니다.

다음의 지시에 따라 파일 시스템 서비스 제공자를 위한 환경을 설정하십시오.

1. JNDI 사이트  로부터 파일 시스템 JNDI 지원을 다운로드하십시오.
2. **계속**을 클릭하여 JNDI 1.2.1을 다운로드하십시오. 사용권 계약이 표시됩니다.
3. **수용**을 클릭한 다음 **FS 문맥**을 클릭하여 JNDI 문맥 지원에 대한 다운로드 옵션을 표시하십시오.
4. fscontext.zip을 다운로드하고 워크스테이션에서 파일의 압축을 푸십시오.
5. FTP를 사용하여 fscontext.jar 및 providerutil.jar를 시스템으로 전송하고 /QIBM/UserData/Java400/ext에 넣으십시오. 이것은 확장 디렉토리이며 여기에 넣는 JAR 파일은 어플리케이션을 실행하면 자동으로 찾을 수 있습니다(즉, 클래스 경로에 필요하지 않음).

JNDI에 대한 서비스 제공자를 위한 지원이 있으면 어플리케이션에 대한 문맥 정보를 설정해야 합니다. 이러한 설정은 SystemDefault.properties 파일에 필수 정보를 넣어야 가능합니다. 시스템에는 디폴트 등록 정보를 지정할 수 있는 위치가 많지만 가장 좋은 방법은 홈 디렉토리(즉, /home/)에서 SystemDefault.properties라고 하는 텍스트 파일을 작성하는 것입니다.

파일을 작성하려면 다음의 행을 사용하거나 기존 파일에 이 행들을 추가하십시오.

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

이러한 행들은 파일 시스템 제공자가 JNDI 요구를 처리하며 /DataSources/jdbc가 JNDI를 사용하는 타스크의 루트임을 지정합니다. 이 위치를 변경할 수 있지만 지정한 디렉토리는 존재하는 것이어야 합니다. 지정한 위치는 DataSource 예를 바인드하고 배치하는 위치입니다.

연결

연결 오브젝트는 Java(TM) Database Connectivity(JDBC) 자료로의 연결을 나타냅니다. 데이터베이스에 대해 SQL문을 처리하기 위해 Statement 오브젝트는 Connection 오브젝트를 통해 작성됩니다. 어플리케이션 프로그램에는 한 번에 여러 연결이 있을 수 있습니다. 이러한 Connection 오브젝트를 모두 동일한 데이터베이스에 연결하거나 각각 다른 데이터베이스에 연결할 수 있습니다.

JDBC에서 두 가지 방법으로 연결을 확보할 수 있습니다.

- DriverManager 클래스를 통해.
- DataSource를 사용을 통해.

DataSource를 사용하여 연결을 확보하면 어플리케이션 이식성과 유지보수성이 향상되기 때문에 선호합니다. 또한 어플리케이션이 연결 및 명령문 풀과 분배된 트랜잭션을 투명하게 사용할 수 있습니다.

연결 확보에 대한 자세한 내용은 다음의 섹션을 참조하십시오.

DriverManager
DriverManager는 어플리케이션이 사용할 수 있도록 사용 가능한 JDBC 드라이버 세트를 관리하는 정적 클래스입니다.

연결 등록 정보
표에는 유효한 JDBC 드라이버 연결 등록 정보, 각각의 값 및 설명이 들어 있습니다.

UDBDataSource와 함께 DataSource 사용
특정 등록 정보를 갖도록 설정하고 JNDI(Java Naming and Directory Interface)를 사용하여 일부 디렉토리 서비스에 바인드하여 UDBDataSource 클래스로 DataSource를 배치할 수 있습니다.

DataSource 등록 정보
표에는 유효한 DataSource 등록 정보, 각각의 값 및 설명이 들어 있습니다.

기타 DataSource 구현
원시 JDBC 드라이버와 함께 DataSource 인터페이스의 다른 구현이 제공됩니다. UDBDataSource 및 관련 함수가 채택될 때까지 브릿지로서만 존재합니다.

연결을 확보했으면 이를 사용하여 다음의 JDBC 타스크를 수행할 수 있습니다.

- 데이터베이스와 대화하기 위해 138 페이지의 『명령문 작성』.

- 데이터베이스에 대해 트랜잭션 제어.
- 데이터베이스에 대한 메타데이터 검색.

DriverManager:

DriverManager는 Java(TM) Developer Kit(JDK)에서 제공하는 정적 클래스입니다. 어플리케이션이 사용할 수 있도록 JDBC(Java Database Connectivity) 드라이버 세트를 관리할 책임이 있습니다. 어플리케이션은 필요하면 여러 JDBC 드라이버를 동시에 사용할 수 있습니다. 어플리케이션 프로그램이 JDBC 드라이버를 지정하는 방법은 URL(Uniform Resource Locator)을 사용하는 것입니다. 특정 JDBC 드라이브에 대한 URL을 DriverManager에 전달하여 어플리케이션은 어플리케이션으로 리턴되어야 하는 JDBC 연결 유형에 대해 DriverManager에 알립니다.

이를 수행하기 전에 DriverManager는 연결을 분배할 수 있도록 사용 가능한 JDBC 드라이버를 인식해야 합니다. Class.forName 메소드 호출을 통해 메소드에 전달된 스트링명에 기초하여 실행 중인 JVM(Java Virtual Machine)으로 클래스를 로드합니다. 다음은 원시 JDBC 드라이버를 로드하는 데 사용되는 class.forName 메소드의 예입니다.

예: 원시 JDBC 드라이버 로드

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

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

JDBC 드라이버는 드라이버 구현 클래스가 로드되면 자동으로 DriverManager에 이 드라이버에 대해 알려주도록 설계되어 있습니다. 앞에서 언급한 코드 행이 처리된 후에 작업할 때 사용하는 DriverManager가 원시 JDBC 드라이버를 사용할 수 있습니다. 다음의 코드 행은 원시 JDBC URL을 사용하여 Connection 오브젝트를 요구합니다.

예: Connection 오브젝트 요구

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

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

JDBC URL의 가장 간단한 형태는 콜론으로 구분되어 있는 세 개의 값으로 구성된 리스트입니다. 리스트에서 첫 번째 값은 JDBC URL이 항상 jdbc인 프로토콜을 나타냅니다. 두 번째 값은 하위 프로토콜이며 원시 JDBC 드라이버를 지정하기 위해 db2를 사용합니다. 세 번째 값은 특정 시스템에 대한 연결을 설정하기 위한 시스템

명입니다. 로컬 데이터베이스에 연결할 때 사용할 수 있는 두 개의 특별한 값이 있습니다. *LOCAL 및 localhost 입니다(둘 다 대소문자를 구분하지 않음). 다음과 같이 특정 시스템명을 제공할 수 있습니다.

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

그러면 rchasmop 시스템에 대한 연결이 작성됩니다. 연결하려고 하는 시스템이 리모트 시스템인 경우(예를 들어, DRDA(Distributed Relational Database Architecture)를 통해) 관계형 데이터베이스 디렉토리의 시스템 명을 사용해야 합니다.

주: 달리 지정하지 않았으면 사인 인하기 위해 현재 사용하는 사용자 ID와 암호를 데이터베이스에 대한 연결을 설정할 때에도 사용합니다.

등록 정보: DriverManager.getConnection 메소드는 앞에 표시된 단일 스트링 URL을 사용하며 DriverManager에서 Connection 오브젝트를 확보하기 위한 메소드 중 하나일 뿐입니다. 사용자 ID와 암호를 사용하는 DriverManager.getConnection 메소드의 다른 버전도 있습니다. 이 버전의 예를 보여줍니다.

예: 사용자 ID와 암호를 사용하는 DriverManager.getConnection 메소드

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

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

이 코드 행은 어플리케이션을 실행하고 있는 사용자가 누구이든지 상관없이 암호 newtiger를 사용하는 사용자 cujo로 로컬 데이터베이스에 연결하려고 시도합니다. 계속적인 사용자 정의를 위해 java.util.Properties 오브젝트를 사용하는 DriverManager.getConnection 메소드의 버전도 있습니다. 다음은 예를 보여줍니다.

예: java.util.Properties 오브젝트를 사용하는 DriverManager.getConnection 메소드

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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);
```

이 코드는 매개변수로 사용자 ID와 암호를 전달한 앞에서 언급된 버전과 기능상 동등합니다.

원시 JDBC 드라이버에 대한 연결 등록 정보의 전체 리스트는 Connection 등록 정보를 참조하십시오.

URL 등록 정보: 등록 정보를 지정하는 또 다른 방법은 URL 오브젝트 스스로 리스트에 등록 정보를 넣는 것입니다. 리스트의 각 등록 정보는 세미콜론으로 구분되며 리스트의 형태는 `property name=property value` 이어야 합니다. 이 형태는 단축 형태일 뿐이고 처리를 수행하는 방법을 많이 변경하지 않습니다. 다음은 그 예입니다.

예: URL 등록 정보 지정

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

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

이 코드는 역시 앞에서 언급한 예와 기능상 동등합니다.

등록 정보 값이 등록 정보 오브젝트와 URL 오브젝트에 모두 지정되었으면 URL 버전이 등록 정보 오브젝트의 버전보다 우선적입니다. 다음은 예를 보여줍니다.

예: URL 등록 정보

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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);
```

예에서는 Properties 오브젝트의 버전 대신 URL 스트링의 사용자 ID와 암호를 사용합니다. 그러면 앞에서 언급한 코드와 기능적으로 동등하게 됩니다.

자세한 정보는 다음의 예를 참조하십시오.

- 원시 JDBC 및 Toolbox JDBC 동시 사용
- 액세스 등록 정보
- 유효하지 않은 사용자 ID 및 암호

예: 원시 JDBC 및 Toolbox JDBC 동시 사용:

프로그램에서 원시 JDBC 연결 및 Toolbox JDBC 연결하는 방법의 예입니다.

예: 원시 JDBC 및 Toolbox JDBC 동시 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
//////////////////////////////////////////////////////////////////////////////
//
// 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 Toolbox
// JDBC connection.
//
// This technique is convenient because it allows you to use different
```

```
// JDBC drivers for different tasks concurrently. For example, the
// Toolbox 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 Toolbox JDBC driver.");
     else
            System.out.println("There is something wrong with conn2.");

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

예: 등록 정보 액세스:

다음은 등록 정보 액세스를 사용하는 방법을 보여주는 예입니다.

예: 등록 정보 액세스

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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.
        }
    }
}
```

예: 유효하지 않은 사용자 ID 및 암호:

다음은 SQL 명명 모드에서 Connection 등록 정보를 사용하는 방법을 보여주는 예입니다.

예: 유효하지 않은 사용자 ID 및 암호

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
//////////////////////////////////////////////////////////////////////////////
//
// 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());
        }

    }
}
```

연결 등록 정보: 이 표에는 유효한 JDBC 드라이버 연결 등록 정보, 각각의 값 및 설명이 들어 있습니다.

등록 정보	값	의미
″access″	″all″, ″read call″, ″read only″	이 값은 특정 연결을 사용하여 수행할 수 있는 작업의 유형을 제한하는 데 사용할 수 있습니다. 디폴트 값은 ″all″이며 기본적으로 연결이 JDBC API에 대해 완전한 액세스 권한을 가지고 있음을 의미합니다. ″read call″ 값은 연결이 조회만을 수행하고 저장 프로시듀어를 호출할 수 있게 합니다. SQL문을 통해 데이터베이스를 갱신하려는 시도는 중단됩니다. ″read only″ 값은 연결을 조회로만 제한하는 데 사용할 수 있습니다. 저장 프로시듀어 호출 및 update 명령문은 중단됩니다.
 ″batch style″	″2.0″, ″2.1″	JDBC 2.1 스펙은 일괄처리 갱신의 예외를 처리하는 두 번째 방법을 정의합니다. 드라이버는 이 중 하나를 따를 수 있습니다. 디폴트는 JDBC 2.0 스펙에 정의된 대로 작업합니다. 

등록 정보	값	의미
″block size″	″0″, ″8″, ″16″, ″32″, ″64″, ″128″, ″256″, ″512″	결과 세트에 대한 시간에 페치되는 행수입니다. 결과 세트의 일반적인 이송 전용 처리의 경우 이 크기의 블록이 확보됩니다. 그런 다음, 데이터베이스는 각행이 사용자 어플리케이션에 의해 처리됨에 따라 액세스가 불가능해집니다. 블록의 끝에 도달했을 때에만 데이터베이스는 다른 자료 블록을 요구합니다. 이 값은 ″blocking enabled″ 등록 정보가 true로 설정된 경우에만 사용할 수 있습니다. 블록 크기 등록 정보를 ″0″으로 설정하는 것은 ″blocking enabled″ 등록 정보를 ″false″로 설정하는 것과 같습니다. 디폴트는 블록 크기 ″32″로 블록을 사용하는 것입니다. 현재 매우 임의적인 결정이며 나중에 디폴트가 변경될 수 있습니다. 현재, 블록은 화면이동 가능한 결과 세트에서 사용되지 않습니다.
″blocking enabled″	″true″, ″false″	이 값은 연결이 결과 세트 행 검색에 블록을 사용해야 하는지의 여부를 판별하는 데 사용됩니다. 블록은 결과 세트의 처리 성능을 상당히 개선할 수 있습니다. 디폴트로 이 등록 정보는 true로 설정됩니다.
 ″cursor hold″	″true″, ″false″	이 값은 트랜잭션이 확약될 때까지 결과 세트를 열린 상태로 두어야 하는지의 여부를 지정합니다. true 값은 확약이 호출된 후에 어플리케이션이 열린 결과 세트에 액세스할 수 있음을 의미합니다. false 값은 연결된 상태에서 확약이 열린 커서를 닫음을 의미합니다. 디폴트로 이 등록 정보는 true로 설정됩니다. 이 값 등록 정보는 연결에 대해 작성된 모든 결과 세트의 디폴트 값으로 사용합니다. JDBC 3.0에 커서 보류성 지원이 추가되었기 때문에 어플리케이션이 나중에 다른 보류성을 지정하면 이 디폴트를 간단하게 대체됩니다. 

등록 정보	값	의미
 ″data truncation″	″true″, ″false″	이 값은 문자 자료의 절단으로 인해 경고 및 예외가 생성되는지(true) 아니면 자료를 자동으로 절단하기만 하면 되는지(false)를 지정합니다. 디폴트가 true이면 문자 필드의 자료 절단을 따라야 합니다. 
 ″date format″	″julian″, ″mdy″, ″dmy″, ″ymd″, ″usa″, ″iso″, ″eur″, ″jis″	이 등록 정보는 날짜의 형식화 방식을 변경할 수 있게 합니다. 
 ″date separator″	″/″, ″-″, ″.″, ″,″, ″b″	이 등록 정보는 날짜 분리자를 변경할 수 있게 합니다. 일부 dateFormat 값과 함께 사용할 경우에만 유효합니다(시스템 규칙 준수). 
 ″decimal separator″	″.″, ″,″	이 등록 정보는 십진 분리자를 변경할 수 있게 합니다. 

등록 정보	값	의미
″do escape processing″	″true″, ″false″	이 등록 정보는 연결 아래의 명령문이 이탈 처리를 수행해야 하는지의 여부에 대해 플래그를 설정합니다. 이탈 처리를 사용하는 것은 SQL문이 모든 플랫폼에서 일반적이고 유사한 것이 되도록 코드화하는 방법이지만 데이터베이스는 이탈 섹션을 읽고 사용자에게 적합한 시스템 특정 버전을 대체합니다. 이것은 시스템이 추가 작업을 부담하는 것을 제외하면 장점으로 볼 수 있습니다. 사용자가 이미 유효한 iSeries SQL 구문이 포함된 SQL문만을 사용할 것이라는 것을 알고 있으면 이 값을 ″false″로 설정하여 성능을 증가시키는 것이 좋습니다. 이 등록 정보의 디폴트 값은 참이며 JDBC 스펙을 준수해야 하기 때문입니다(즉 이탈 처리는 디폴트로 활동 상태입니다). 이 값은 JDBC 스펙의 부족으로 인해 추가되는 것입니다. 명령문 클래스에서만 이탈 처리를 마침표 Off로 설정할 수 있습니다. 간단한 명령문을 처리하는 경우에는 이렇게 설정해도 상관이 없습니다. 명령문을 작성하고 이탈 처리를 Off로 설정하고 명령문 처리를 시작하십시오. 그러나 준비된 명령문과 호출 가능한 명령문의 경우에 이 체계는 제대로 이루어지지 않습니다. 준비되었거나 호출 가능한 명령문이 구성되었으며 이후로 변경되지 않았을 때 SQL문을 제공합니다. 따라서, 명령문을 미리 준비하고 나중에 이탈 처리를 변경하는 것은 좋지 않습니다. 이 연결 등록 정보가 있으면 추가 오버헤드를 방지할 수 있습니다.
 ″errors″	″basic″, ″full″	이 등록 정보를 사용하면 전체 시스템 2차 레벨 오류 텍스트를 SQLException 오브젝트 메세지에 리턴할 수 있습니다. 디폴트는 표준 메세지 텍스트만을 리턴하는 basic입니다. 

등록 정보	값	의미
 ″libraries″	공백으로 구분된 라이브러리 리스트(라이브러리 리스트는 콜론이나 쉼표로 구분할 수도 있습니다).	이 등록 정보를 사용하면 라이브러리 리스트를 서버 작업의 라이브러리 리스트에 넣거나 특정 디폴트 라이브러리를 설정할 수 있습니다. ″naming″ 등록 정보는 이 등록 정보의 작업 방식에 영향을 줍니다. ″naming″이 ″sql″로 설정되는 디폴트의 경우 JDBC는 ODBC와 같이 작동합니다. 라이브러리 리스트는 연결의 처리 방식에는 영향을 주지 않습니다. 규정되지 않은 모든 표에 대한 디폴트 라이브러리가 있습니다. 디폴트로 이 라이브러리의 이름은 연결된 사용자 프로파일과 동일합니다. 라이브러리 등록 정보를 지정하면 리스트에서 첫 번째 라이브러리가 디폴트 라이브러리가 됩니다. 디폴트 라이브러리를 연결 URL에 지정한 경우(″jdbc:db2:*local/mylibrary″에서와 마찬가지) 이 등록 정보의 값을 대체합니다. ″naming″이 ″system″으로 설정된 경우에 이 등록 정보에 대해 지정된 각 라이브러리가 사용자의 라이브러리 리스트에 추가되며 라이브러리 리스트를 탐색하여 규정되지 않은 표 참조를 분석합니다. 
 ″lob threshold″	500000 미만의 값이면 모두 가능	이 등록 정보는 lob 열이 임계값 크기보다 작으면 lob 열에 대한 로케이터 대신 결과 세트 기억장치에 실제 값을 넣도록 드라이버에 지시합니다. 이 등록 정보는 lob 자료 크기 자체가 아닌 열 크기에 대해 작용합니다. 예를 들어, lob 열이 각 lob에 대해 최대 1MB를 보유하도록 정의되었지만 모든 열 값이 500KB 미만인 경우에 로케이터를 계속 사용합니다. 최대 할당 크기인 16MB 보다 크게 자료 블록이 항상 커지지 않는다는 위험 없이 자료 블록을 페치하도록 크기 제한이 설정된다는 점을 참고하십시오. 큰 결과 세트를 사용하면 계속해서 이 제한을 초과하기 쉬우며 페치가 실패합니다. 블록 크기 등록 정보와 이 등록 정보가 자료 블록의 크기와 상호 작용하는 방법에 있어서 주의를 요합니다. 디폴트는 0입니다. 로케이터는 항상 lob 자료에 사용합니다. 

등록 정보	값	의미
″naming″	″sql″, ″system″	이 등록 정보는 종래의 iSeries 명명 구문이나 표준 sql 명명 구문 중 하나를 사용할 수 있게 합니다. ″system″ 명명은 콜렉션과 표 값을 분리하는 데 ″/″ 문자를 사용해야 함을 의미하며 ″sql″ 명명은 값들을 분리하는 데 ″.″ 문자를 사용해야 함을 의미합니다. 이 값의 설정에는 디폴트 라이브러리가 무엇인지를 가리키는 분기도 있습니다. 자세한 정보는 라이브러리 등록 정보(84 페이지 참조)를 참조하십시오. 디폴트는 ″sql″ 명명을 사용하는 것입니다.
″password″	anything	이 등록 정보는 연결에 암호를 지정할 수 있도록 합니다. 이 등록 정보는 ″user″ 등록 정보와 함께 지정하지 않으면 올바로 작동되지 않습니다. 이러한 등록 정보들은 iSeries 작업을 실행하고 있는 사용자가 아닌 사용자로 데이터베이스에 연결할 수 있도록 합니다. ″user″ 및 ″password″ 등록 정보를 지정하는 것은 연결 메소드를 서명 getConnection(스트링 url, 스트링 userId, 스트링 password)과 함께 사용하는 것과 같습니다.
 ″prefetch″	″true″, ″false″	이 등록 정보는 결과 세트에 대한 첫 번째 자료를 처리 후 즉시 페치해야 하는지 아니면 자료를 요구할 때까지 기다려야 하는지를 지정합니다. 디폴트가 true이면 자료를 사전 페치해야 합니다. 원시 JDBC 드라이버를 사용하는 어플리케이션의 경우에는 이러한 상태가 드물게 발생합니다. 이 등록 정보는 주로 데이터베이스 엔진이 사용자가 요구하기 전에 사용자를 대신하여 결과 세트에서 자료를 페치하지 않는 것이 중요한 Java 저장 프로시듀어 및 사용자 정의 기능과 함께 내부용으로 존재합니다. 
 ″reuse objects″	″true″, ″false″	이 등록 정보는 사용자가 닫은 일부 오브젝트 유형을 드라이버가 다시 사용하려고 시도해야 하는지의 여부를 지정합니다. 이는 성능 향상입니다. 디폴트는 true입니다.

등록 정보	값	의미
 ″server trace″	정수의 스트링 표현	이 등록 정보는 JDBC 서버 작업의 추적을 가능하게 합니다. 서버 추적이 가능한 경우에 클라이언트가 서버에 연결되면 추적을 시작하고 연결이 단절되면 종료합니다. 추적 자료는 서버의 스풀 파일에 수집합니다. 상수를 더하고 이 합계를 set 메소드에 전달하여 여러 레벨의 서버 추적을 함께 작동시킬 수 있습니다. **주:** 이 등록 정보는 일반적으로 지원 담당자가 사용하며 그 값을 더 이상 설명하지 않습니다.
 ″time format″	″hms″, ″usa″, ″iso″, ″eur″, ″jis″	이 등록 정보를 사용하면 시간 값의 형식화 방식을 변경할 수 있습니다. 
 ″time separator″	″:″, ″.″, ″,″, ″b″	이 등록 정보를 사용하면 시간 분리자를 변경할 수 있습니다. 일부 timeFormat 값과 함께 사용할 경우에만 유효합니다(시스템 규칙 준수). 
″trace″	″true″, ″false″	이 등록 정보는 연결의 추적을 on으로 설정할 수 있게 합니다. 이것을 간단한 디버깅 지원으로 사용할 수 있습니다. 현재 이 기능을 향상시키는 것을 고려 중입니다. D2. The JDBC driver threw an exception. What do I do?를 참조하십시오. 디폴트 값은 ″false″로 추적을 사용하지 않습니다.
″transaction isolation″	″none″, ″read committed″, ″read uncommitted″, ″repeatable read″, ″serializable″	이 등록 정보는 연결에 대한 트랜잭션 분리 레벨을 설정할 수 있게 합니다. 이 등록 정보를 특정 레벨로 설정하는 것과 연결 인터페이스에서 setTransactionIsolation 메소드의 레벨을 지정하는 것에는 차이가 없습니다. 이 등록 정보의 디폴트 값은 ″none″이며 JDBC가 자동확약 모드를 디폴트로 사용하기 때문입니다.

등록 정보	값	의미
″translate binary″	″true″, ″false″	이 등록 정보는 JDBC 드라이버가 2진 및 varbinary 자료 값들을 문자 및 varchar 자료 값인 것처럼 취급하도록 강제하는데 사용할 수 있습니다. 이 등록 정보의 디폴트는 ″false″이며 2진 자료가 문자자료와 동일하게 취급되지 않습니다.
 ″use block insert″	″true″, ″false″	이 등록 정보는 원시 JDBC 드라이버가 데이터베이스에 자료 블록을 삽입하기 위해 블록 삽입 모드로 들어가도록 합니다. 이것은 일괄처리 갱신의 최적화된 버전입니다. 이 최적화된 모드는 특정 시스템 제한사항을 위반하지 않을 것을 보장하는 어플리케이션에서만 사용할 수 있으며 그렇지 않으면 자료 삽입 실패 및 잠재적인 자료 손상이 발생합니다. 이 등록 정보를 On으로 설정한 어플리케이션은 일괄처리된 갱신을 수행하려고 시도할 때 로컬 시스템에만 연결해야 합니다. 블록화 삽입은 DRDA를 통해 관리할 수 없기 때문에 리모트 연결을 설정할 때 DRDA를 사용해서는 안됩니다. 어플리케이션은 SQL insert 명령문 및 values 섹션이 있는 PreparedStatement가 모든 insert values 매개변수를 작성하는지도 확인해야 합니다. values 리스트에서는 상수가 허용되지 않습니다. 이 점은 시스템의 블록화 삽입 엔진의 요구사항입니다. 디폴트는 거짓입니다.
″user″	anything	이 등록 정보는 연결에 대해 사용자 ID를 지정할 수 있게 합니다. 이 등록 정보는 ″user″ 등록 정보와 함께 지정하지 않으면 올바로 작동되지 않습니다. 이러한 등록 정보들은 iSeries 작업을 실행하고 있는 사용자가 아닌 사용자로 데이터베이스에 연결할 수 있도록 합니다. ″user″ 및 ″password″ 등록 정보를 지정하는 것은 연결 메소드를 서명 getConnection(스트링 url, 스트링 userId, 스트링 password)과 함께 사용하는 것과 같습니다.

예: JNDI로 UDBDataSource 작성 및 바인드:

다음은 JNDI로 UDBDataSource 작성 및 바인드 하는 방법을 보여주는 예입니다.

예: JNDI로 UDBDataSource 작성 및 바인드

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

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

예: UDBDataSourceBind 작성 및 DataSource 등록정보 설정:

다음은 UDBDataSource 작성 및 DataSource 등록 정보로서 사용자 ID와 암호를 설정하는 방법을 보여주는 예입니다.

예: UDBDataSourceBind 작성 및 DataSource 등록정보 설정

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

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

예: UDBDataSource를 바인드하기 전에 초기 문맥 확보:

다음의 예는 UDBDataSource를 바인드하기 전에 초기 문맥을 확보합니다. 그런 다음 이 문맥에서 lookup 메소드를 사용하여 어플리케이션이 사용할 유형 DataSource의 오브젝트를 리턴합니다.

예: UDBDataSource를 바인드하기 전에 초기 문맥 확보

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// 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();
    }
}
```

예: UDBDataSource 작성 및 사용자 ID와 암호 확보:

다음은 UDBDataSource를 작성하고 getConnection 메소드를 사용하여 런타임 시에 사용자 ID와 암호를 확보하는 방법을 보여주는 예입니다.

예: UDBDataSource 작성 및 사용자 ID와 암호 확보

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
/// 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();
    }
}
```

UDBDataSource와 함께 DataSources 사용:

DataSource 인터페이스는 JDBC(Java(TM) Database Connectivity) 드라이버 사용시 보다 많은 유연성을 허용하기 위해 설계되었습니다. DataSource의 사용은 두 단계로 분리할 수 있습니다.

- **배치**
  배치는 JDBC 어플리케이션을 실제로 실행하기 전에 발생하는 설정 단계입니다. 배치에는 대개 특정 등록 정보를 갖도록 DataSource를 설정한 다음 JNDI(Java Naming and Directory Interface)를 사용하여 디렉토리 서비스로 바인드하는 작업이 필요합니다. 디렉토리 서비스는 대개 LDAP(Lightweight Directory Access Protocol)이지만 CORBA(Common Object Request Broker Architecture) 오브젝트 서비스, Java RMI(Remote Method Invocation) 또는 기초 파일 시스템과 같은 다른 것일 수도 있습니다.
- **사용**
  DataSource의 런타임 사용과 배치를 분리하면 많은 어플리케이션들이 DataSource를 재사용할 수 있습니다. 배치의 일부 측면을 변경하면 이 DataSource를 사용하는 모든 어플리케이션이 변경사항을 자동으로 채택합니다.

DataSource의 장점은 어플리케이션 개발 프로세스에 직접 영향을 미치지 않고 JDBC 드라이버가 어플리케이션을 대신하여 작업을 수행할 수 있다는 점입니다. 자세한 정보는 연결 풀, 명령문 풀 및 분배 트랜잭션을 참조하십시오.

UDBDataSourceBind: UDBDataSourceBind 프로그램은 UDBDataSource를 작성하고 JNDI와 바인드하는 예입니다. 이 프로그램은 요구된 모든 기본 타스크를 수행합니다. 즉, UDBDataSource 오브젝트를 인스턴스화하고 이 오브젝트에 대한 등록 정보를 설정하고 JNDI 문맥을 검색하고 오브젝트를 JNDI 문맥 내의 이름에 바인드합니다.

배치 시간 코드는 벤더마다 다릅니다. 어플리케이션은 작업하려는 특정 DataSource 구현을 가져와야 합니다. 가져오기 리스트에서 패키지 규정 UDBDataSource 클래스를 가져옵니다. 이 어플리케이션에서 가장 익숙하지 않은 부분은 JNDI에 대해 수행한 작업입니다(예를 들어, Context 오브젝트의 검색과 바인드 호출). 추가 정보는 Sun Microsystems, Inc.의 JNDI  를 참조하십시오.

이 프로그램을 실행하고 정상적으로 완료했으면 JNDI 디렉토리 서비스에 SimpleDS라고 하는 새로운 항목이 있습니다. 이 항목은 JNDI 문맥이 지정한 위치에 있습니다. DataSource 구현을 이제 배치합니다. 어플리케이션 프로그램은 이 DataSource를 활용하여 데이터베이스 연결 및 JDBC 관련 작업을 검색할 수 있습니다.

UDBDataSourceUse: UDBDataSourceUse 프로그램은 이미 배치된 어플리케이션을 사용하는 JDBC 어플리케이션의 예입니다.

JDBC 어플리케이션은 앞의 예에서 UDBDataSource를 바인드하기 전에 수행했던 작업과 같이 초기 문맥을 확보합니다. 그런 다음 lookup 메소드를 사용하여 어플리케이션이 사용할 유형 DataSource의 오브젝트를 리턴합니다.

주: 런타임 어플리케이션은 DataSource 인터페이스의 메소드에만 관심이 있으므로 어플리케이션은 구현 클래스를 알아야 할 필요가 없습니다. 따라서 어플리케이션의 이식이 가능합니다.

UDBDataSourceUse가 사용자가 속한 조직 내에서 큰 조작을 실행하는 복잡한 어플리케이션이라고 가정하십시오. 조직 내에 12개 이상의 유사한 큰 어플리케이션이 있습니다. 네트워크에서 시스템 중 하나의 이름을 변경해야 합니다. 배치 툴을 실행하고 단일 UDBDataSource 등록 정보를 변경하면 코드를 변경하지 않고 모든 어플리케이션에서 이 새로운 작동을 수행할 수 있습니다. DataSource의 한 가지 장점은 시스템 설정 정보를 강화할 수 있게 한다는 점입니다. 또 다른 주요 장점은 연결 풀, 명령문 풀 및 분배 트랜잭션에 대한 지원과 같이 드라이버가 어플리케이션이 모르게 기능을 구현할 수 있게 한다는 점입니다.

UDBDataSourceBind 및 UDBDataSourceUse를 자세하게 분석한 후에 DataSource 오브젝트가 수행할 작업을 어떻게 알았는지 궁금했을 것입니다. 이들 프로그램에는 시스템, 사용자 ID 또는 암호를 지정하기 위한 코드가 없습니다. UDBDataSource 클래스에는 모든 등록 정보에 대한 디폴트 값이 있습니다. 디폴트로 실행 중인 어플리케이션의 사용자 프로파일과 암호를 사용하여 로컬 iSeries 서버에 연결합니다. 대신 사용자 프로파일 cujo를 사용하여 연결을 작성했는지 확인하려는 경우에 다음의 두 방법으로 이를 수행할 수 있었습니다.

- DataSource 등록 정보로서 사용자 ID 및 암호을 설정하십시오. 이 기술을 사용하는 방법에 대해서는 예: UDBDataSourceBind 작성 및 DataSource 등록 정보 설정을 참조하십시오.

- 런타임 시에 사용자 ID와 암호를 사용하는 DataSource getConnection 메소드를 사용하십시오. 이 기술을 사용하는 방법에 대해서는
  예: UDBDataSource 작성 및 사용자 ID와 암호 확보를 참조하십시오.

DriverManager를 사용하여 작성한 연결에 대해 지정할 수 있는 등록 정보가 있는 것처럼 UDBDataSource에 대해 지정할 수 있는 등록 정보가 많습니다. 원시 JDBC 드라이버에 대해 지원되는 등록 정보의 리스트는 DataSource 등록 정보를 참조하십시오.

이러한 리스트들은 유사하지만 향후 릴리스에서 유사할 것인지는 확실하지 않습니다. DataSource 인터페이스로의 코딩을 시작하도록 권장합니다.

주: 원시 JDBC 드라이버에는 두 개의 다른 DataSource 구현이 있지만 직접적인 사용은 권장하지 않습니다.

- DB2DataSource
- DB2StdDataSource

DataSource 등록 정보:

이 표에는 유효한 자료 소스 등록 정보, 각각의 값 및 설명이 들어 있습니다.

Set 메소드(자료 유형)	**값**	**설명**
setAccess (String)	″all″, ″read call″, ″read only″	이 등록 정보는 특정 연결을 사용하여 수행할 수 있는 작업의 유형을 제한하는 데 사용할 수 있습니다. 디폴트 값은 ″all″이며, 기본적으로 연결이 Java(TM) Database Connectivity(JDBC) API로의 완전 액세스 권한을 가지고 있음을 의미합니다. ″read call″ 값은 연결이 조회만을 수행하고 저장 프로시듀어를 호출할 수 있게 합니다. SQL문을 통해 데이터베이스를 갱신하려는 시도는 SQLException의 원인이 됩니다. ″read only″ 값은 연결을 조회로만 제한합니다. 저장 프로시듀어 호출 또는 update 명령문을 처리하려는 시도로 인해 SQLException이 발생했습니다.
setBatchStyle (String)	″2.0″, ″2.1″	JDBC 2.1 스펙은 일괄처리 갱신의 예외를 처리하는 두 번째 방법을 정의합니다. 드라이버는 이 중 하나를 따를 수 있습니다. 디폴트는 JDBC 2.0 스펙에 정의된 대로 작업합니다.

Set 메소드(자료 유형)	**값**	**설명**
setUseBlocking (boolean)	″true″, ″false″	이 등록 정보는 연결이 결과 세트 행 검색에 블록을 사용해야 하는지의 여부를 판별하는 데 사용됩니다. 블록은 결과 세트의 처리 성능을 상당히 개선할 수 있습니다. 디폴트로 이 등록 정보는 true로 설정됩니다.
setBlockSize (int)	″0″, ″8″, ″16″, ″32″, ″64″, ″128″, ″256″, ″512″	등록 정보는 결과 세트에 대한 시간에 페치되는 행수를 표시합니다. 결과 세트의 일반적인 이송 전용 처리의 경우 데이터베이스가 가진 조회를 만족하는 행이 많을때 이크기의 블록이 확보됩니다. JDBC 드라이버의 내부 기억장치에서 블록의 끝에 도달했을 때에만 자료 블록에 대한 다른 요구를 데이터베이스에 보냅니다. 이 값은 useBlocking 등록 정보가 true로 설정된 경우에만 사용할 수 있습니다. 자세한 정보는 setUseBlocking(94 페이지 참조)을 참조하십시오. 블록 크기 등록정보를 ″0″으로 설정하는 것은 setUseBlocking(false)을 호출하는 것과 같은 영향을 미칩니다. 디폴트는 블록 크기 ″32″로 블록을 사용하는 것입니다. 이것은 매우 임의적인 결정이며 나중 릴리스에 디폴트가 변경될 수 있습니다. 현재, 블록은 화면이동 가능한 결과 세트에서 사용되지 않습니다. 블록화 기능을 사용하면 사용자 어플리케이션이 가지고 있는 커서 민감도의 정도에 영향을 줍니다. 민감한 커서는 다른 SQL문에 의해 변경된 사항을 감지합니다. 그러나 자료 캐싱으로 인해 데이터베이스에서 자료를 페치해야 할 때에만 변경사항을 감지합니다.

Set 메소드(자료 유형)	**값**	**설명**
setCursorHold (boolean)	″true″, ″false″	이 등록 정보는 트랜잭션을 확약할 때 결과 세트를 열어 두어야 하는지의 여부를 지정합니다. true 값은 확약이 호출된 후에 어플리케이션이 열린 결과 세트에 액세스할 수 있음을 의미합니다. false 값은 연결된 상태에서 확약이 열린 커서를 닫음을 의미합니다. 디폴트로 이 등록 정보는 true로 설정됩니다. 이 등록 정보는 연결에 대해 작성된 모든 결과 세트의 디폴트 값으로 사용합니다. JDBC 3.0에 커서 지원이 추가되었기 때문에(자세한 내용은 ResultSet 특성 섹션참조) 어플리케이션이 나중에 다른 커서 지원을 지정하면 이 디폴트는 간단하게 대체됩니다.
setDataTruncation (boolean)	″true″, ″false″	등록 정보는 다음과 같이 지정합니다. • 문자 자료의 절단으로 인해 경고 및 예외가 생성됩니다(true). • 자료를 자동으로 절단해야 합니다(false). 추가적인 자세한 정보는 DataTruncation을 참조하십시오.
setDatabaseName (String)	임의의 이름	이 등록 정보는 DataSource가 연결하려고 시도하는 데이터베이스를 지정합니다. 디폴트는 *LOCAL입니다. 데이터베이스 이름은 어플리케이션을 실행하는 시스템의 관계형 데이터베이스 디렉토리에 존재하거나 로컬 시스템을 지정하기 위해 특수 값 *LOCAL 또는 localhost이어야 합니다.
setDataSourceName (String)	임의의 이름	이 등록 정보를 사용하면 ConnectionPoolDataSource JNDI(Java Naming and Directory Interface) 이름을 전달하여 연결 풀을 지원할 수 있습니다.
setDateFormat (String)	″julian″, ″mdy″, ″dmy″, ″ymd″, ″usa″, ″iso″, ″eur″, ″jis″	이 등록 정보를 사용하면 날짜의 형식화 방식을 변경할 수 있습니다.
setDateSeparator (String)	″/″, ″-″, ″.″, ″,″, ″b″	이 등록 정보는 날짜 분리자를 변경할 수 있게 합니다. 일부 dateFormat 값과 함께 사용할 경우에만 유효합니다(시스템 규칙 준수).
setDecimalSeparator (String)	″.″, ″,″	이 등록 정보를 사용하면 십진 분리자를 변경할 수 있습니다.
setDescription (String)	임의의 이름	이 등록 정보를 사용하면 이 DataSource 오브젝트의 텍스트 설명을 설정할 수 있습니다.
setDoEscapeProcessing (boolean)	″true″, ″false″	이 등록 정보는 SQL문이 처리 완료를 피했는지의 여부를 지정합니다. 이 등록 정보의 디폴트 값은 true입니다.

Set 메소드(자료 유형)	**값**	**설명**
setFullErrors (boolean)	″true″, ″false″	이 등록 정보를 사용하면 SQLException 오브젝트 메세지에 전체 시스템의 2차 레벨 오류 텍스트를 리턴할 수 있습니다. 디폴트는 false입니다.
setLibraries (String)	공백으로 구분된 라이브러리 리스트	이 등록 정보를 사용하면 라이브러리 리스트를 서버 작업의 라이브러리 리스트에 넣을 수 있습니다. 이 등록 정보는 setSystemNaming(true)을 사용할 때에만 사용합니다.
setLobThreshold (int)	500000 미만의 값이면 모두 가능	이 등록 정보는 LOB 열이 임계값 크기보다 작으면 로케이터 오브젝트(LOB) 로케이터 대신 실제 값을 넣도록 드라이버에 지시합니다.
setLoginTimeout (int)	임의의 값	이 등록 정보는 현재 무시되며 추후에 사용할 계획입니다.
setNetworkProtocol (int)	임의의 값	이 등록 정보는 현재 무시되며 추후에 사용할 계획입니다.
setPassword (String)	임의의 스트링	이 등록 정보는 연결에 암호를 지정할 수 있도록 합니다. 사용자 ID를 설정하지 않으면 이 등록 정보는 무시됩니다.
setPortNumber (int)	임의의 값	이 등록 정보는 현재 무시되며 추후에 사용할 계획입니다.
setPrefetch (boolean)	″true″, ″false″	이 등록 정보는 드라이버가 결과 세트에 대한 첫 번째 자료를 처리 후 즉시 페치해야 하는지 아니면 자료를 요구할 때까지 기다려야 하는지를 지정합니다. 디폴트는 참입니다.
setReuseObjects (boolean)	″true″, ″false″	이 등록 정보는 사용자가 닫은 일부 오브젝트 유형을 드라이버가 다시 사용하려고 시도해야 하는지의 여부를 지정합니다. 이것은 성능 향상입니다. 디폴트는 참입니다.
setServerName (String)	임의의 이름	이 등록 정보는 현재 무시되며 추후에 사용할 계획입니다.
setServerTraceCategories (int)	정수의 스트링 표현	이 등록 정보는 JDBC 서버 작업의 추적을 가능하게 합니다. 서버 추적이 가능한 경우에 클라이언트가 서버에 연결되면 추적을 시작하고 연결이 단절되면 종료합니다. 추적 자료는 서버의 스풀 파일에 수집합니다. 상수를 더하고 이 합계를 set 메소드에 전달하여 여러 레벨의 서버 추적을 함께 작동시킬 수 있습니다. **주:** 이 등록 정보는 보통 지원 담당자가 사용하므로, 그 값에 대해서는 더 이상 설명하지 않습니다.

Set 메소드(자료 유형)	**값**	**설명**
setSystemNaming (boolean)	″true″, ″false″	이 등록 정보를 사용하면 콜렉션 및 표를 마침표(SQL 명명) 또는 슬래시(시스템 명명)로 구분해야 하는지를 지정할 수 있습니다. 이 등록 정보는 디폴트 라이브러리를 사용하는지(SQL 명명) 또는 라이브러리 리스트를 사용하는지(시스템 명명)도 판별합니다. 디폴트는 SQL 명명입니다.
setTimeFormat (String)	″hms″, ″usa″, ″iso″, ″eur″, ″jis″	이 등록 정보를 사용하면 시간 값의 형식화 방식을 변경할 수 있습니다.
setTimeSeparator (String)	″:″, ″.″, ″,″, ″b″	이 등록 정보를 사용하면 시간 분리자를 변경할 수 있습니다. 이 등록 정보는 일부 timeFormat 값과 함께 사용할 경우에만 유효합니다(시스템 규칙 준수).
setTrace (boolean)	″true″, ″false″	이 등록 정보는 간단한 추적을 가능하게 할 수 있습니다. 디폴트 값은 false입니다.
setTransactionIsolationLevel (String)	″none″, ″read committed″, ″read uncommitted″, ″repeatable read″, ″serializable″	이 등록 정보를 사용하면 트랜잭션 분리 레벨을 지정할 수 있습니다. 이 등록 정보의 디폴트 값은 ″none″이며 JDBC가 자동확약 모드를 디폴트로 사용하기 때문입니다.
setTranslateBinary (Boolean)	″true″, ″false″	이 등록 정보는 JDBC 드라이버가 2진 및 varbinary 자료 값들을 문자 및 varchar 자료 값인 것처럼 취급하도록 강제하는데 사용할 수 있습니다. 이 등록 정보의 디폴트는 false입니다.
setUseBlockInsert (boolean)	″true″, ″false″	이 등록 정보는 원시 JDBC 드라이버가 데이터베이스에 자료 블록을 삽입하기 위해 블록 삽입 모드로 들어가도록 합니다. 이것은 일괄처리 갱신의 최적화된 버전입니다. 이 최적화된 모드는 특정 시스템 제한사항을 위반하지 않을 것을 보장하는 어플리케이션에서만 사용할 수 있으며 그렇지 않으면 자료 삽입 실패 및 잠재적인 자료 손상이 발생합니다. 이 등록 정보를 On으로 설정한 어플리케이션은 일괄처리된 갱신을 수행하려고 시도할 때 로컬 시스템에만 연결해야 합니다. 블록화 삽입은 DRDA를 통해 관리할 수 없기 때문에 리모트 연결을 설정할 때 DRDA를 사용해서는 안됩니다. 어플리케이션은 SQL insert 명령문 및 values 섹션이 있는 PreparedStatement가 모든 insert values 매개변수를 작성하는지도 확인해야 합니다. values 리스트에서는 상수가 허용되지 않습니다. 이 점은 시스템의 블록화 삽입 엔진의 요구사항입니다. 디폴트는 false입니다.

Set 메소드(자료 유형)	값	설명
setUser (String)	anything	이 등록 정보를 사용하면 연결을 확보하기 위해 사용자 ID를 설정할 수 있습니다. 이 등록 정보는 암호 등록 정보도 설정할 것을 요구합니다.

기타 DataSource 구현:

원시 JDBC 드라이버와 함께 포함된 두 가지 DataSource 인터페이스 구현이 있습니다. 이러한 DataSource 구현은 제거된 것으로 간주해야 합니다. 계속해서 사용할 수는 있지만 추후의 개선에서는 향상되지 않습니다. 예를 들어, 확실한 연결 및 명령문 풀은 이러한 구현에 추가되지 않습니다. 이러한 구현은 UDBDataSource 인터페이스 및 관련 기능을 채택할 때까지 존재합니다.

DB2DataSource: DB2DataSource는 DataSource 인터페이스의 초기 구현이었으며 전체 스펙을 준수하지는 않습니다(즉, 스펙보다 앞선 것입니다). DB2DataSource는 오늘날 WebSphere(R) 사용자들이 현재 릴리스로 마이그레이트하기 위해서만 존재하므로 그렇지 않은 경우에는 사용하지 말아야 합니다.

DB2StdDataSource: DB2StdDataSource는 JDBC 선택적 패키지 스펙이 최종 스펙이 되면 스펙을 준수하는 DB2DataSource 구현의 개정판입니다. 새로운 버전은 이미 DB2DataSource 버전에서 기록된 코드를 분리하지 않도록 제공되었습니다.

이러한 DataSource 구현들을 활용하는 어플리케이션을 기록한 경우에 이전의 모든 등록 정보가 지원되면 UDBDataSource로의 마이그레이트는 사소한 타스크입니다. UDBDataSource로 마이그레이트하여 새로운 UDBDataSource 클래스의 기능을 확보하는 것이 바람직합니다.

IBM Developer Kit for Java용 DatabaseMetaData 인터페이스

DatabaseMetaData 인터페이스는 기초 자료 소스에 대한 정보를 제공하기 위해 IBM Developer Kit for Java(TM) JDBC 드라이버에 의해 구현됩니다. 주로 자료 소스와 대화하는 방법을 판별하기 위해 어플리케이션 서버와 툴이 사용합니다. 어플리케이션이 자료 소스에 대한 정보를 얻기 위해 DatabaseMetaData 메소드를 사용하기도 하지만 일반적인 것은 아닙니다.

DatabaseMetaData 인터페이스에는 제공하는 다음의 정보 유형에 따라 분류할 수 있는 150개 이상의 메소드가 포함됩니다.

- 자료 소스에 대한 99 페이지의 『일반 정보 검색』
- 자료 소스는 99 페이지의 『피처 지원 판별』
- 100 페이지의 『자료 소스 한계』
- 100 페이지의 『SQL 오브젝트 및 해당 속성』

- 자료 소스가 제공한 100 페이지의『트랜잭션 지원』

DatabaseMetaData 인터페이스에는 다양한 DatabaseMetaData 메소드의 리턴값으로 사용되는 상수인 40개 이상의 필드도 포함됩니다.

DatabaseMetaData 인터페이스에서 메소드에 대해 변경된 사항에 관한 정보는 100 페이지의『JDBC 3.0 변경사항』을 참조하십시오.

DatabaseMetaData 오브젝트 작성: DatabaseMetaData 오브젝트는 연결 메소드 getMetaData를 사용하여 작성됩니다. 오브젝트가 작성되면 기초 자료 소스에 대한 정보를 동적으로 발견하는데 사용할 수 있습니다. 다음의 예는 DatabaseMetaData 오브젝트를 작성하며 이를 사용하여 표 이름에 허용되는 최대 문자 수를 판별합니다.

예: DatabaseMetaData 오브젝트 작성

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

일반 정보 검색: 일부 DatabaseMetaData 메소드는 자료 소스에 대한 일반 정보와 각각의 구현에 대한 세부사항을 동적으로 찾는데 사용됩니다. 일부 메소드는 다음을 포함합니다.

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

피처 지원 판별: 큰 DatabaseMetaData 메소드 그룹은 주어진 피처나 피처 세트가 드라이버나 기초 자료 소스에 의해 지원되는지의 여부를 판별하는 데 사용할 수 있습니다. 또한 메소드중 일부는 제공 되는 지원 레벨을 설명합니다. 개별 피처에 대한 지원을 설명하는 메소드 중 일부는 다음을 포함합니다.

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

피처 지원 레벨을 설명하기 위한 메소드는 다음과 같습니다.

- supportsANSI92EntryLevelSQL
- supportsCoreSQLGrammar

자료 소스 한계: 다른 메소드 그룹은 주어진 자료 소스에 의해 부과된 한계를 제공합니다. 이 범주의 메소드 중 일부는 다음을 포함합니다.

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

이 그룹의 메소드는 정수로 한계 값을 리턴합니다. 리턴값 0은 한계가 없거나 한계를 알 수 없음을 의미합니다.

SQL 오브젝트 및 해당 속성: 일부 DatabaseMetaData 메소드는 주어진 자료 소스를 채우는 SQL 오브젝트에 대한 정보를 제공합니다. 이러한 메소드는 SQL 오브젝트의 속성을 판별할 수 있습니다. 이 메소드는 각 행이 특정 오브젝트를 설명하는 ResultSet 오브젝트를 리턴합니다. 예를 들어, 메소드 getUDTs는 자료 소스에 정의된 각 사용자 정의 표(UDT)에 대한 행이 있는 ResultSet 오브젝트를 리턴합니다. 이 범주의 예는 다음을 포함합니다.

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

트랜잭션 지원: 작은 메소드 그룹은 자료 소스가 지원하는 트랜잭션 의미론에 대한 정보를 제공합니다. 이 범주의 예는 다음을 포함합니다.

- supportsMultipleTransactions
- getDefaultTransactionIsolation

DatabaseMetaData 인터페이스를 사용하는 방법의 예는 예: IBM Developer Kit for Java의 DatabaseMetaData 인터페이스를 참조하십시오.

JDBC 3.0 변경사항:

JDBC 3.0의 일부 메소드에 대한 리턴값이 변경되었습니다. 다음의 메소드가 리턴한 ResultSet에 필드를 추가하여 JDBC 3.0에서 이러한 메소드가 갱신되었습니다.

- getTables
- getColumns
- getUDTs
- getSchemas

주: JDK(Java Development Kit) 1.4를 사용하여 어플리케이션을 개발하고 있는 경우에는 테스트 시에 일정한 수의 열이 리턴되는 것을 인식할 수 있습니다. 어플리케이션을 작성하고 이러한 모든 열에 액세스할 것으로 예상합니다. 그러나 어플리케이션을 JDK의 이전 릴리스에서도 실행하도록 설계하고 있으면 어플리케이션은 이전 JDK 릴리스에 없는 이러한 필드에 액세스하려고 시도할 때 SQLException을 받습니다. SafeGetUDTs는 어플리케이션이 JDK 1.4, JDK 1.3 및 이전 JDK 릴리스에 대해 작업하기 위해 작성하는 방법의 한 예입니다.

예: IBM Developer Kit for Java용 DatabaseMetaData 인터페이스:

다음 예는 표 리스트를 리턴하는 방법을 보여줍니다.

예 1: 표 리스트 리턴

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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():
```

자세한 정보는, IBM Developer Kit for Java(TM)용 DatabaseMetaData 인터페이스의 내용을 참조하십시오.

예: 둘 이상의 열이 있는 메타데이터 ResultSet 사용:

다음은 둘 이상의 열이 있는 메타데이터 ResultSet를 사용하는 방법의 예입니다.

예: 둘 이상의 열이 있는 메타데이터 ResultSet 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
//////////////////////////////////////////////////////////////////////////////
//
// SafeGetUDTs 예 프로그램은 이전 릴리스보다 JDK 1.4에 더 많은
// 열이 있는 메타데이터 ResultSets을 사용하는 방법을 설명합니다.
//
// 명령 구문:
//    java SafeGetUDTs
//
//////////////////////////////////////////////////////////////////////////////
//
// 이 소스는 IBM Developer for Java JDBC 드라이버의 예입니다.
// IBM 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수
// 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권
// 사용권을 부여합니다.
//
// 이 샘플 코드는 IBM에 의해 예시 목적으로만 제공됩니다.
// 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다.
// 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에
// 대해 보증할 수 없습니다.
//
// 여기에 포함된 모든 프로그램은 어떠한 보증없이 "현상태대로"
// 제공됩니다. 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증은
// 명시적으로 거부됩니다.
//
// 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.
                }
            }
        }
    }
}
```

«

예외

Java(TM) 언어에서는 예외를 사용하여 해당 프로그램에 대해 오류 처리 기능을 제공합니다. 예외는 명령의 정상적인 흐름을 방해하는 프로그램을 실행하면 발생하는 이벤트입니다.

Java 런타임 시스템 및 Java 패키지의 많은 클래스는 일부 상황에서 throw문을 사용하여 예외 상태를 유발합니다. 동일한 메카니즘을 사용하여 Java 프로그램에서 예외 상태를 유발할 수 있습니다.

예외에 대한 자세한 내용은 다음의 섹션을 참조하십시오.

SQLException
SQLException 클래스와 그 하위 유형은 자료 소스에 액세스하는 동안 발생하는 오류와 경고에 대한 정보를 제공합니다.

SQLWarning
메소드는 데이터베이스 액세스 경고를 발생시키는 경우에 SQLWarning 오브젝트를 생성합니다. 다음 인터페이스의 메소드는 SQLWarning을 생성할 수 있습니다.

- Connection
- 명령문 및 그 하위 유형, PreparedStatement 및 CallableStatement
- ResultSet

DataTruncation
DataTruncation은 SQLWarning의 서브클래스입니다. SQLWarning이 발생하지 않으면 다른 SQLWarning 오브젝트와 마찬가지로 DataTruncation 오브젝트가 때때로 발생하고 접속됩니다.

110 페이지의『자동 절단』
setMaxFieldSize 명령문 메소드를 사용하면 모든 열에 대해 최대 필드 *크기*를 지정할 수 있습니다. *크기*가 최대 필드 *크기* 값을 초과해서 자료가 잘린 경우에 경고 또는 예외를 보고하지 않습니다.

SQLException:

SQLException 클래스와 그 하위 유형은 자료 소스에 액세스하는 동안 발생하는 오류와 경고에 대한 정보를 제공합니다.

인터페이스가 정의한 대부분의 JDBC와 달리 클래스에서는 예외 지원을 제공합니다. JDBC 어플리케이션을 실행하는 동안 발생하는 예외에 대한 기본 클래스는 SQLException입니다. JDBC API의 모든 메소드는

SQLException을 유발할 수 있는 것으로 선언합니다. SQLException은 java.lang.Exception을 확장한 것이며 데이터베이스 문맥에서 발생하는 실패와 관련된 추가 정보를 제공합니다. 특히 SQLException에서는 다음의 정보를 사용할 수 있습니다.

- 텍스트 설명
- SQLState
- 오류 코드
- 발생한 다른 예외에 대한 참조

ExceptionExample은 SQLException 포착(이 경우에 예상됨) 및 이것이 제공하는 모든 정보 덤프를 올바로 처리하는 프로그램입니다.

주: JDBC는 예외들을 함께 체인으로 연결할 수 있는 메카니즘을 제공합니다. 이 메카니즘을 사용하면 드라이버 또는 데이터베이스가 한 번의 요구로 여러 오류를 보고할 수 있습니다. 현재 원시 JDBC 드라이버가 이를 수행하는 예는 없습니다. 이 정보는 참조용으로만 제공되며 드라이버가 향후에 이 작업을 수행하지 않을 것임을 정확히 표현하는 것은 아닙니다.

명시한 대로 오류가 발생하면 SQLException 오브젝트가 작성됩니다. 이 내용은 올바르지만 완전한 그림은 아닙니다. 실제로는 원시 JDBC 드라이버가 실제 SQLException을 거의 유발하지 않습니다. 자체의 SQLException 서브클래스에 대한 인스턴스를 유발합니다. 따라서 아래에 설명된 대로 실제로 실패한 것에 대한 자세한 정보를 판별할 수 있습니다.

DB2Exception.java: DB2Exception 오브젝트는 직접 작성되지 않습니다. 이 기본 클래스는 모든 JDBC 예외에 공통적인 기능을 보유하기 위해 사용합니다. 이 클래스에는 JDBC가 유발하는 표준 예외인 두 개의 서브클래스가 있습니다. 이러한 서브클래스는 DB2DBException.java와 DB2JDBCException.java입니다. DB2DBException은 사용자에게 보고된 데이터베이스로부터 직접 온 예외입니다. DB2JDBCException은 JDBC 드라이버가 자체적으로 문제를 발견하면 발생합니다. 이러한 방식으로 예외 클래스 계층을 분할하면 두 가지 예외 유형을 다르게 처리할 수 있습니다.

DB2DBException.java: 언급한 대로 DB2DBException은 데이터베이스에서 직접 온 예외입니다. 이 예외는 JDBC 드라이버가 CLI를 호출하고 다시 SQLERROR 리턴 코드를 받으면 발견됩니다. 이러한 경우에 메세지 텍스트, SQLState 및 벤더 코드를 알기 위해 CLI 함수 SQLError를 호출합니다. SQLMessage의 대체 텍스트도 검색하며 사용자에게 리턴합니다. DatabaseException 클래스는 데이터베이스가 인식하고 JDBC 드라이버에 보고하여 예외 오브젝트를 빌드하도록 하는 오류를 발생시킵니다.

DB2JDBCException.java: DB2JDBCException은 JDBC 드라이버 자체로부터 온 오류 상태에 대해 생성됩니다. 이 예외 클래스의 기능은 근본적으로 다릅니다. JDBC 드라이버는 스스로 예외 및 오퍼레이팅 시스템과 데이터베이스가 데이터베이스에서 비롯된 예외에 대해 처리하는 다른 문제의 메세지 언어 변환을 처리합니다. 가능하면 JDBC 드라이버는 데이터베이스의 SQLState를 충실하게 따릅니다. JDBC 드라이버가 유발하는 예외에 대한 벤더 코드는 항상 -99999입니다. CLI 계층이 자주 인식하고 리턴하는 DB2DBException에는 -99999 오류 코드가 있습니다. JDBCException 클래스는 JDBC 드라이버가 인식하는 오류를 발생시키며 스스로 예외를 빌드합니다. 릴리스의 개발 중에 실행했을 때 다음의 출력이 작성되었습니다. 스택의 맨 위에 DB2JDBCException이 있음을 알 수 있습니다. 이것은 데이터베이스에 대한 요구를 작성하기 전에 JDBC 드

라이버로부터 오류가 보고되었음을 나타냅니다.

예: SQLException:

다음은 SQLException을 포착하고 이것이 제공하는 모든 정보를 덤프하는 예입니다.

예: SQLException

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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:

다음 인터페이스의 메소드는 데이터베이스 액세스 경고를 발생시키는 경우에 SQLWarning 오브젝트를 생성합니다.

- Connection
- 명령문 및 그 하위 유형, PreparedStatement 및 CallableStatement
- ResultSet

메소드가 SQLWarning 오브젝트를 생성한 경우 호출자에게 자료 액세스 경고가 발생했음을 알리지 않습니다. getWarnings 메소드는 SQLWarning 오브젝트를 검색하기 위해 적합한 오브젝트에서 호출해야 합니다. 그러나 SQLWarning의 DataTruncation 서브클래스는 일부 상황에서 버릴 수 있습니다. 원시 JDBC 드라이버가 효율성의 향상을 위해 일부 데이터베이스 생성 경고를 무시하도록 선택한다는 점을 유의해야 합니다. 예를 들어, ResultSet.next 메소드를 통해 ResultSet의 끝을 넘어서 자료를 검색하려고 시도하면 시스템에 의해 경고가 생성됩니다. 이러한 경우 다음 메소드는 사용자에게 오류를 알리기 위해 true 대신 false를 리턴하도록 정의됩니다. 다시 언급하기 위해 오브젝트를 작성할 필요가 없으므로 경고는 간단히 무시됩니다.

여러 개의 자료 액세스 경고가 발생하면 첫 번째 경고로 체인 처리되며 SQLWarning.getNextWarning 메소드를 호출하여 검색할 수 있습니다. 체인에 더 이상 경고가 없으면 getNextWarning은 널(null)을 리턴합니다.

후속 SQLWarning 오브젝트는 다음 명령문을 처리할 때까지 아니면 ResultSet 오브젝트의 경우에는 커서의 위치를 다시 지정할 때 체인에 계속 추가됩니다. 결과적으로 모든 체인의 모든 SQLWarning 오브젝트를 제거합니다.

Connection, Statement 및 ResultSet 오브젝트를 사용하면 SQLWarning이 생성됩니다. SQLWarning은 정보용 메세지로서 특정 조작이 정상적으로 완료되었지만 사용자가 주의해야 하는 다른 정보가 있을 수 있음을 나타냅니다. SQLWarning은 SQLException 클래스의 확장이지만 유발되지 않습니다. 그 대신 생성시키는 오브젝트에 접속됩니다. SQLWarning이 생성되면 경고가 생성되었음을 어플리케이션에 알리는 수단이 없습니다. 어플리케이션은 경고 정보를 적극적으로 요구해야 합니다.

SQLException과 마찬가지로 SQLWarning도 서로 체인으로 연결할 수 있습니다. Connection, Statement 또는 ResultSet 오브젝트에서 clearWarnings 메소드를 호출하여 해당 오브젝트에 대한 경고를 해제할 수 있습니다.

주: clearWarnings 메소드를 호출하면 모든 경고가 해제되는 것은 아닙니다. 특정 오브젝트와 연관된 경고만을 해제합니다.

JDBC 드라이버는 사용자가 수동으로 해제하지 않은 경우에 특정 시간에 SQLWarning 오브젝트를 해제합니다. 다음의 조치를 취하면 SQLWarning 오브젝트가 해제됩니다.

- Connection 인터페이스의 경우에 새로운 Statement, PreparedStatement 또는 CallableStatement 오브젝트를 작성하면 경고가 해제됩니다.

- Statement 인터페이스의 경우에 다음 명령문을 처리할 때(또는 PreparedStatement and CallableStatement에 대해 명령문을 다시 처리할 때) 경고가 해제됩니다.
- ResultSet 인터페이스의 경우에는 커서의 위치를 다시 지정할 때 경고가 해제됩니다.

DataTruncation:

DataTruncation은 SQLWarning의 서브클래스입니다. SQLWarning이 발생하지 않으면 다른 SQLWarning 오브젝트와 마찬가지로 DataTruncation 오브젝트가 때때로 발생하고 접속됩니다. DataTruncation 오브젝트는 SQLWarning에 의해 리턴되는 것 이상의 추가 정보를 제공합니다. 사용할 수 있는 정보는 다음을 포함합니다.

- 전송되어야 하는 자료의 바이트 수.
- 절단된 열 또는 매개변수 색인.
- 색인이 매개변수에 대한 것인지 아니면 ResultSet 열에 대한 것인지의 여부.
- 데이터베이스에서 읽거나 데이터베이스에 쓸 때 절단이 발생했는지의 여부.
- 실제로 전송된 자료의 양.

일부 경우에는 정보를 해독할 수 있지만 완전히 직관할 수는 없는 상황이 발생합니다. 예를 들어, PreparedStatement의 setFloat 메소드를 사용하여 정수값이 있는 열에 값을 삽입하면 부동 수가 열이 보유할 수 있는 최대값보다 클 수 있기 때문에 DataTruncation이 발생합니다. 이러한 경우에 절단할 바이트 계수가 이치에 맞지 않지만 드라이버가 절단 정보를 제공하는 데 중요합니다.

set() 및 update() 메소드 보고: JDBC 드라이버들 사이에는 미묘한 차이점이 있습니다. 원시 및 Toolbox JDBC 드라이버와 같은 일부 드라이버는 매개변수 설정 시에 자료 절단 문제를 포착하여 보고합니다. 이러한 작업은 PreparedStatement set 메소드 또는 ResultSet update 메소드에 대해 수행합니다. 다른 드라이버는 명령문 처리 시에 이 문제를 보고하고 execute, executeQuery 또는 updateRow 메소드에 의해 수행됩니다.

처리를 더 이상 계속할 수 없을 때가 아닌 사용자가 올바르지 않은 자료를 제공했을 경우 문제를 보고하지 못하면, 두 가지 이점이 있습니다.

- 처리 시에 문제를 언급하지 않고 문제가 있을 때 어플리케이션에서 실패를 언급할 수 있습니다.
- 매개변수를 설정하는 시기를 검사하여 JDBC 드라이버는 명령문 처리 시에 데이터베이스에 전달된 값이 유효한지 확인할 수 있습니다. 그러면 데이터베이스가 작업을 최적화할 수 있으며 처리를 보다 빠르게 완료할 수 있습니다.

ResultSet.update() 메소드가 DataTruncation 예외 발생 시킴: 이전의 일부 릴리스에서는 절단 상태가 존재하면 ResultSet.update() 메소드가 경고를 공고했습니다. 이러한 경우는 자료 값이 데이터베이스에 삽입될 예정일 때 발생합니다. 스펙은 JDBC 드라이버가 이러한 경우에 예외를 발생시키도록 지시합니다. 결과적으로 JDBC 드라이버는 이러한 방식으로 작업합니다.

자료 절단 오류를 받은 ResultSet 갱신 함수 처리와 오류를 받은 update 또는 insert 명령문의 준비된 명령문 매개변수 세트 처리 사이에는 주목할 만한 차이점이 없습니다. 두 경우 모두에 문제는 동일합니다. 원하는 위치에 맞지 않는 자료를 제공했을 것입니다.

NUMERIC 및 DECIMAL은 소수점의 오른쪽 부분을 자동으로 자릅니다. 이것은 UDB NT용 JDBC가 작업하는 방식과 iSeries 서버에서 대화식 SQL이 작업하는 방식입니다.

주: 자료가 잘릴 때에는 값을 반올림하지 않습니다. NUMERIC 또는 DECIMAL 열에 맞지 않는 매개변수의 분수 부분은 경고 없이 간단하게 제거됩니다.

다음은 values 섹션의 값을 실제로 준비된 명령문에 설정되는 매개변수로 가정한 예입니다.

```
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.
```

자료 절단 경고와 자료 절단 예외의 차이점

스펙은 데이터베이스에 기록할 값에 대한 자료 절단은 예외 상태를 발생시킨다는 점을 나타냅니다. 자료 절단이 데이터베이스에 기록되는 값에 대해 수행되지 않으면 경고가 생성됩니다. 이는 자료 절단 상태가 발생한 시점을 식별했음을 의미하며 자료 절단이 처리되는 명령문 유형도 알고 있어야 합니다. 이 점을 요구사항으로 하여 다음은 몇 가지 SQL문 유형의 작동을 나타낸 것입니다.

- SELECT문에서 조회 매개변수는 데이터베이스 내용을 전혀 손상하지 않습니다. 따라서 자료 절단 상태는 항상 경고를 게재하여 처리합니다.
- VALUES INTO 및 SET 명령문에서 입력 값은 출력 값을 생성하는 데에만 사용합니다. 결과로 경고가 발행됩니다.
- CALL문에서 JDBC 드라이버는 저장 프로시듀어가 매개변수를 사용하여 수행하는 작업을 판별할 수 없습니다. 저장 프로시듀어 매개변수가 잘리면 예외 상태가 항상 발생합니다.
- 다른 모든 명령문 유형은 경고를 게재하지 않고 예외 상태를 발생시킵니다.

Connection 및 DataSource에 대한 자료 절단 등록 정보: 그동안 여러 릴리스에서 자료 절단 등록 정보를 사용할 수 있었습니다. 이 등록 정보의 디폴트는 true이며 자료 절단 문제를 검사하고 경고를 게재하거나 예외 상태를 유발함을 의미합니다. 값이 데이터베이스 열에 맞지 않아도 게의치 않는 경우에 이 등록 정보를 편의 및 성능 상의 이유로 제공합니다. 드라이버가 가능한한 많은 값을 열에 넣도록 할 수 있습니다.

문자 및 2진 기반 자료 유형에만 영향을 주는 자료 절단 등록 정보: 바로 이전의 두 릴리스에서는 자료 절단 등록 정보가 자료 절단 예외 상태가 발생할 수 있는지의 여부를 판별했습니다. JDBC 어플리케이션이 절단을 중요하게 여기지 않을 때 값이 잘리는 것을 염려하지 않도록 자료 절단 등록 정보를 제 위치에 넣었습니다. 어플리케이션이 DECIMAL(2,0)에 100을 삽입하려고 시도할 때 데이터베이스에 저장된 값 00 또는 10을 원하는 경우는 거의 드뭅니다. 따라서 JDBC 드라이버의 자료 절단 등록 정보는 매개변수가 CHAR, VARCHAR, CHAR FOR BIT DATA 및 VARCHAR FOR BIT DATA와 같은 문자 기반 유형에 대한 것일 경우만을 존중하도록 변경되었습니다.

매개변수에만 적용되는 자료 절단 등록 정보: 자료 절단 등록 정보는 데이터베이스가 아닌 JDBC 드라이버의 설정입니다. 결과적으로 명령문 리터럴에는 영향을 주지 않습니다. 예를 들어, 자료 절단 플래그를 false로 설정하면 데이터베이스의 CHAR(8) 열에 값을 삽입하도록 처리되는 다음의 명령문이 계속 실패합니다(연결이 다른 위치에서 할당된 java.sql.Connection 오브젝트인 것으로 가정).

```
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.
```

중요하지 않은 자료 절단에 대해 예외 상태를 유발하는 원시 JDBC 드라이버: 원시 JDBC 드라이버는 사용자가 매개변수로 제공하는 자료를 주목하지 않습니다. 이렇게 하면 처리 속도만 늦어집니다. 그러나 값이 잘려도 문제가 되지 않으며 자료 절단 연결 등록 정보를 false로 설정하지 않은 상황이 있을 수 있습니다.

예를 들어, 전달된 char(10)인 'dettinger'는 값에 대해 중요한 모든 사항이 맞더라도 예외 상태를 유발합니다. 이는 UDB NT용 JDBC가 작업하는 방법에 영향을 주지만 SQL문의 리터럴로 값을 전달한 경우에는 이러한 작동이 발생하지 않습니다. 이 경우에 데이터베이스 엔진은 추가 공간을 자동으로 버립니다.

예외 상태를 유발하지 않는 JDBC 드라이버의 문제점은 다음과 같습니다.

- 성능 오버헤드가 필요 여부에 상관 없이 모든 set 메소드에 널리 퍼져 있습니다. 이점이 없는 대부분의 경우에 함수에는 setString()만큼 상당한 성능 오버헤드가 있을 수 있습니다.
- 사용자의 조치는 예를 들어, 전달된 스트링 값에 대한 trim 함수 호출과 같이 사소한 것입니다.
- 데이터베이스 열에 대해 고려해야 하는 문제가 있습니다. CCSID 37의 공간은 CCSID 65535 또는 13488의 공간이 전혀 아닙니다.

자동 절단: setMaxFieldSize 명령문 메소드를 사용하면 모든 열에 대해 최대 필드 *크기*를 지정할 수 있습니다. *크기*가 최대 필드 *크기* 값을 초과해서 자료가 잘린 경우에 경고 또는 예외를 보고하지 않습니다. 앞에서 언급한 자료 절단 등록 정보와 같이 이 메소드는 CHAR, VARCHAR, CHAR FOR BIT DATA 및 VARCHAR FOR BIT DATA와 같은 문자 기반 유형에만 영향을 줍니다.

트랜잭션

트랜잭션은 논리적 작업 단위입니다. 논리적 작업 단위를 완료하려면 데이터베이스에 대해 몇 가지 조치를 취해야 합니다. 트랜잭션 지원을 통해 어플리케이션은 다음을 확인할 수 있습니다.

- 논리적 작업 단위를 완료하기 위한 모든 단계를 수행했는지의 여부.
- 작업 단위 파일에 대한 단계 중 하나가 실패했을 때 논리적 작업 단위의 일부로 수행한 모든 작업을 복원할 수 있으며 트랜잭션이 시작되기 전에 데이터베이스가 이전의 상태로 되돌아갈 수 있는지의 여부.

트랜잭션은 자료 무결성을 제공하고 어플리케이션 의미론을 정정하고 동시 액세스 중에 자료에 대한 일관적인 관점을 제공하기 위해 사용됩니다. 모든 JDBC(Java(TM) Database Connectivity) 호환 드라이버는 트랜잭션을 지원해야 합니다.

주: 이 절에서는 로컬 트랜잭션 및 트랜잭션의 표준 JDBC 개념만을 설명합니다. Java 및 원시 JDBC 드라이버는 JTA(Java Transaction API), 분배 트랜잭션 및 2단계 확약 프로토콜(2PC)을 지원합니다.

모든 트랜잭션 작업은 Connection 오브젝트 레벨에서 처리합니다. 트랜잭션에 대한 작업이 완료되면 commit 메소드를 호출하여 마무리지을 수 있습니다. 어플리케이션이 트랜잭션을 중단하면 rollback 메소드를 호출합니다.

연결 하의 모든 Statement 오브젝트는 트랜잭션의 일부입니다. 이는 어플리케이션이 세 개의 Statement 오브젝트를 작성하고 각 오브젝트를 사용하여 데이터베이스를 변경한 경우에 확약 또는 롤백 호출이 발생하면 세 개의 모든 명령문에 대한 작업이 영구적인 작업이 되거나 이 작업을 버릴 것임을 의미합니다.

확약 및 롤백 SQL문을 사용하여 전적으로 SQL에 대해 작업할 때 트랜잭션을 완료합니다. 이러한 SQL문은 동적으로 준비할 수 없으며 트랜잭션을 완료하기 위해 JDBC 어플리케이션에서 이를 사용하려고 시도해서는 안됩니다.

어플리케이션에서 트랜잭션을 사용하려면 다음을 참조하십시오.

자동 확약 모드
JDBC는 데이터베이스에 대한 모든 갱신이 즉시 영구적인 내용이 되는 자동 확약 모드를 사용합니다.

트랜잭션 분리 레벨
트랜잭션 분리 레벨은 트랜잭션 내에서 명령문이 볼 수 있는 자료를 지정하고 동시 액세스의 레벨에 직접 영향을 줍니다.

savepoint
savepoint는 전체 트랜잭션을 버리지 않고 어플리케이션이 롤백할 수 있는 체크 포인트입니다. savepoint에 대해 다음의 정보를 찾으십시오.

- savepoint로의 설정 및 롤백
- savepoint 해제

자동 확약 모드:

디폴트로 JDBC는 자동 확약이라고 하는 조작 모드를 사용합니다. 이는 데이터베이스에 대한 모든 갱신이 즉시 영구적인 내용이 되는 것을 의미합니다. 논리적 작업 단위에 데이터베이스에 대한 둘 이상의 갱신이 필요한

상황은 자동 확약 모드에서 안전하게 수행할 수 없습니다. 한 번 갱신한 후와 다른 갱신을 수행하기 전에 어플리케이션이나 시스템에 어떤 상태가 발생하면 자동 확약 모드에서 실행할 때 첫 번째 변경을 복원할 수 없습니다.

자동 확약 모드에서는 변경이 즉시 영구적인 내용이 되므로 어플리케이션은 commit 메소드나 rollback 메소드를 호출할 필요가 없습니다. 따라서 어플리케이션의 쓰기가 쉬워집니다.

연결이 유지되는 동안 자동 확약 모드를 동적으로 작동하거나 작동 불가능하게 할 수 있습니다. 자료 소스가 이미 있다고 가정하면 다음과 같은 방법으로 자동 확약을 작동할 수 있습니다.

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

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

자동 확약 설정이 트랜잭션 중간에 변경된 경우에는 지연 중인 작업이 자동으로 확약됩니다. 분배 트랜잭션의 일부인 연결에 대해 자동 확약이 작동되면 SQLException이 생성됩니다.

트랜잭션 분리 레벨:

트랜잭션 분리 레벨은 트랜잭션내에서 명령문이 볼 수 있는 자료를 지정합니다. 이 레벨은 동일한 목표 자료 소스에 대해 트랜잭션들 사이에 가능한 대화를 정의하여 동시 액세스 레벨에 직접 영향을 줍니다.

데이터베이스 이상: 데이터베이스 이상은 단일 트랜잭션의 범위에서 확인했을 때에는 잘못된 것으로 보이지만 모든 트랜잭션의 범위에서 확인했을 때에는 올바른 생성된 결과입니다. 여러 가지 유형의 데이터베이스 이상에 대한 설명은 다음과 같습니다.

- **Dirty** 읽기는 다음과 같은 경우에 발생합니다.
  1. 트랜잭션 A가 표에 한 행을 삽입한 경우.
  2. 트랜잭션 B가 새 행을 읽는 경우.
  3. 트랜잭션 A가 롤 백하는 경우.

  트랜잭션 B는 트랜잭션 A에 의해 삽입된 행에 기초하여 시스템에 대한 작업을 완료했을 수 있지만 이 행은 결코 데이터베이스의 영구적인 부분이 될 수 없습니다.
- **Nonrepeatable** 읽기는 다음과 같은 경우에 발생합니다.
  1. 트랜잭션 A가 한 행을 읽는 경우
  2. 트랜잭션 B가 행을 변경하는 경우
  3. 트랜잭션 A가 동일한 행을 두 번 읽고 새로운 결과를 얻는 경우
- **Phantom** 읽기는 다음과 같은 경우에 발생합니다.
  1. 트랜잭션 A가 SQL 조회의 WHERE절을 만족시키는 모든 행을 읽는 경우
  2. 트랜잭션 B가 WHERE절을 만족시키는 추가 행을 삽입하는 경우

3. 트랜잭션 A가 WHERE 조건을 다시 평가하고 추가 행을 선택하는 경우

주: DB2/400은 잠금 전략으로 인해 지시된 레벨에서 허용 가능한 데이터베이스 이상에 어플리케이션을 항상 노출시키지 않습니다.

JDBC 트랜잭션 분리 레벨: IBM Developer Kit for Java JDBC API에는 5가지 레벨의 트랜잭션 분리 레벨이 있습니다. 제한이 약한 것에서부터 강한 것까지 나열하면 다음과 같습니다.

JDBC_TRANSACTION_NONE
JDBC 드라이버가 트랜잭션을 지원하지 않음을 나타내는 특별한 상수입니다.

JDBC_TRANSACTION_READ_UNCOMMITTED
이 레벨은 트랜잭션이 자료에 대한 미확약 변경을 볼 수 있도록 허용합니다. 모든 데이터베이스 이상이 이 레벨에서 가능합니다.

JDBC_TRANSACTION_READ_COMMITTED
이 레벨은 트랜잭션 내에서 변경된 사항을 트랜잭션이 확약될 때까지 트랜잭션 외부에서 볼 수 없음을 의미합니다. dirty 읽기를 방지합니다.

JDBC_TRANSACTION_REPEATABLE_READ
이 레벨은 트랜잭션이 완료되지 않았을 때 다른 트랜잭션이 변경할 수 없도록 읽은 행의 잠금을 유지함을 의미합니다. dirty 읽기와 nonrepeatable 읽기를 허용하지 않습니다. phantom 읽기는 여전히 가능합니다.

JDBC_TRANSACTION_SERIALIZABLE
값을 추가하거나 표에서 값을 제거하는 다른 트랜잭션이 WHERE 조건을 변경할 수 없도록 트랜잭션에 대해 표를 잠급니다. 그러면 모든 유형의 데이터베이스 이상이 방지됩니다.

setTransactionIsolation 메소드를 사용하여 연결에 대한 트랜잭션 분리 레벨을 변경할 수 있습니다.

고려사항: JDBC 스펙이 앞에서 언급된 다섯 개의 트랜잭션 레벨을 정의하는 것은 흔한 오해입니다. TRANSACTION_NONE 값이 확약 제어 없이 실행하는 개념을 나타낸다고 흔히들 생각합니다. JDBC 스펙은 동일한 방식으로 TRANSACTION_NONE을 정의하지 않습니다. TRANSACTION_NONE은 드라이버가 트랜잭션을 지원하지 않고 JDBC 준수 드라이버가 아닌 레벨로 JDBC 스펙에 정의합니다.
getTransactionIsolation 메소드를 호출할 때 NONE 레벨은 보고하지 않습니다.

이 문제는 최소한 JDBC 드라이버의 디폴트 트랜잭션 분리 레벨이 구현에 의해 정의된다는 사실과 혼동합니다. 원시 JDBC 드라이버 디폴트 트랜잭션 분리 레벨에 대한 트랜잭션 분리의 디폴트 레벨은 NONE입니다. 이 레벨에서 드라이버는 저널이 없는 파일에 대해 작업할 수 있으며 QGPL 라이브러리의 파일과 같은 스펙을 작성할 필요가 없습니다.

원시 JDBC 드라이버는 setTransactionIsolation 메소드에 JDBC_TRANSACTION_NONE을 전달하거나 연결 등록 정보로 none을 지정할 수 있게 합니다. 그러나 getTransactionIsolation 메소드는 값이 none일 때 항상 JDBC_TRANSACTION_READ_UNCOMMITTED를 보고합니다. 어플리케이션에서 요구사항인 경우에 실행하고 있는 레벨을 추적하는 것은 어플리케이션의 책임입니다.

이전의 릴리스에서는 시스템에 true 자동 확약 모드의 개념이 없었기 때문에 트랜잭션 분리 레벨을 변경하여 JDBC 드라이버가 자동 확약에 대한 true 지정을 처리했습니다. 이것은 기능에 매우 가깝지만 모든 시나리오에 대한 올바른 결과를 제공하지 않았습니다. 이러한 작업은 더 이상 수행하지 않으며 데이터베이스는 트랜잭션 분리 레벨의 개념과 자동 확약의 개념을 분리합니다. 따라서 자동 확약을 작동시켜서 JDBC_TRANSACTION_SERIALIZABLE 레벨에서 실행하는 것이 전적으로 유효합니다. 자동 확약 모드가 아닌 JDBC_TRANSACTION_NONE 레벨에서 실행하는 경우만 유효하지 않은 시나리오입니다. 시스템이 트랜잭션 분리 레벨을 사용하여 실행하고 있지 않을 때 어플리케이션은 확약 경계 이상으로 제어를 이어받을 수 없습니다.

JDBC 스펙과 iSeries 플랫폼 간의 트랜잭션 분리 레벨: iSeries 플랫폼은 트랜잭션 분리 레벨에 대해 JDBC 스펙이 제공하는 이름과 일치하지 않는 공통 이름을 가지고 있습니다. 다음의 표는 iSeries 플랫폼이 사용하지만 JDBC 스펙이 사용하는 것과 동등하지 않은 이름들을 일치시킵니다.

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

* 이 표에서는 확실하게 하기 위해 JDBC_TRANSACTION_NONE 값을 iSeries 레벨 *NONE 및 *NC에 맞추어 정렬합니다. 이것은 직접적인 스펙 대 iSeries 레벨 대응이 아닙니다.

savepoint:

savepoint는 트랜잭션에서 "스테이징 포인트"의 설정을 가능하게 합니다. savepoint는 전체 트랜잭션을 버리지 않고 어플리케이션이 롤백할 수 있는 체크 포인트입니다. Savepoint는 JDBC 3.0에서 새 기능으로서, 어플리케이션을 사용하려면 JDK(Java(TM) Development Kit) 1.4에서 실행되어야 합니다. 또한 savepoint는 Developer Kit for Java에서도 새로운 기능이므로, Developer Kit for Java의 이전 릴리스와 함께 JDK 1.4를 사용하지 않는 경우에는 savepoint가 지원되지 않습니다.

주: 시스템은 savepoint에 대한 작업을 위해 SQL문을 제공합니다. JDBC 어플리케이션은 어플리케이션에서 직접 이러한 명령문을 사용하지 않는 것이 좋습니다. 사용해도 무방하지만 이를 수행했을 때 JDBC 드라이버가 savepoint를 추적할 수 없게 됩니다. 최소한 두 모델의 혼합 사용(즉, 사용자 자신의 savepoint SQL 명령문 사용과 JDBC API 사용)은 피해야 합니다.

savepoint로의 설정 및 롤백: savepoint는 트랜잭션의 작업 전체에서 설정할 수 있습니다. 그러면 어플리케이션은 뭔가 잘못되고 있을 때 이러한 savepoint로 롤백하고 이 포인트에서 처리를 계속할 수 있습니다. 다음의 예에서 어플리케이션은 값 FIRST를 데이터베이스 표에 삽입합니다. 그 후에 savepoint를 설정하고 다른

값 SECOND를 데이터베이스에 삽입합니다. savepoint로의 롤백을 실행하고 SECOND 삽입 작업을 복원하지만 FIRST는 지연 중인 트랜잭션의 일부분으로 남습니다. 마지막으로 값 THIRD를 삽입하고 트랜잭션을 확약합니다. 데이터베이스 표에는 값 FIRST와 THIRD가 들어 있습니다.

예: savepoint로의 설정 및 롤백

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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();
```

자동 확약 모드에 있는 동안 savepoint 설정에 문제가 되지는 않겠지만 트랜잭션이 종료하면 더 이상 사용하지 않기 때문에 롤백할 수 없습니다.

savepoint 해제: savepoint는 어플리케이션이 Connection 오브젝트에서 releaseSavepoint 메소드를 사용하여 해제할 수 있습니다. savepoint가 해제된 후에 롤백하려고 시도하면 예외가 발생합니다. 트랜잭션이 확약하거나 롤백하면 모든 savepoint가 자동으로 해제됩니다. savepoint를 롤백하면 그 뒤의 다른 savepoint도 해제됩니다.

분배 트랜잭션

보통 Java(TM) Database Connectivity(JDBC)에서 트랜잭션은 로컬입니다. 이는 단일 연결이 트랜잭션의 모든 작업을 수행하며 연결은 한 번에 하나의 트랜잭션에 대해서만 작업할 수 있음을 의미합니다. 이 트랜잭션에 대한 모든 작업이 완료되었거나 실패했으면 확약 또는 롤백을 호출하여 작업을 영구적으로 만들고 새로운 트랜잭션을 시작할 수 있습니다.

로컬 트랜잭션 이상의 기능을 제공하는 트랜잭션에 대한 확장 지원을 Java에서 사용할 수 있습니다. 이 지원은 JTA(Java Transaction API) 1.0.1 specification  을 사용하여 완전히 지정합니다.

JTA(Java Transaction API)에는 복잡한 트랜잭션에 대한 지원이 있습니다. Connection 오브젝트로부터 커플링을 해제하는 지원도 제공합니다. JDBC는 ODBC(Object Database Connectivity) 및 X/Open CLI(Call Level Interface) 스펙 이후에 모델화되므로 JTA는 X/Open XA(Extended Architecture) 스펙 이후에 모델화됩니다. JTA 및 JDBC는 함께 작업하여 Connection 오브젝트로부터 트랜잭션의 커플링을 해제합니다. Connection 오브젝트에서 트랜잭션의 커플링을 해제하면 여러 트랜잭션에서 동시에 한 연결 작업을 수행할 수 있습니다. 반대로 한 트랜잭션에서 여러 연결 작업을 수행할 수 있습니다.

주: JTA에 대해 작업하려는 경우에 확장 클래스 경로에서 필수 JAR(Java Archive) 파일에 대한 자세한 정보는 JDBC 시작하기를 참조하십시오. JDBC 2.0 선택적 패키지와 JTA JAR 파일을 모두 갖고자 할 수 있습니다(이 파일들은 JDK 1.4를 실행하고 있으면 JDK가 자동으로 찾습니다). 디폴트로 찾을 수 없습니다.

JTA를 사용한 트랜잭션: JTA와 JDBC를 함께 사용하는 경우에 이들 사이에는 트랜잭션 작업을 수행하는 일련의 단계가 있습니다. XA에 대한 지원은 XADataSource 클래스를 통해 제공됩니다. 이 클래스에는 ConnectionPoolDataSource 수퍼클래스와 정확히 동일한 방식으로 연결 풀을 설정하는 지원이 있습니다.

XADataSource 인스턴스를 사용하면 XAConnection 오브젝트를 검색할 수 있습니다. XAConnection 오브젝트는 JDBC Connection 오브젝트와 XAResource 오브젝트 모두에 대해 컨테이너의 역할을 수행합니다. XAResource 오브젝트는 XA 트랜잭션 지원을 처리하도록 설계되었습니다. XAResource는 트랜잭션 ID(XID)라고 하는 오브젝트를 통해 트랜잭션을 처리합니다.

XID는 사용자가 구현해야 하는 인터페이스입니다. X/Open 트랜잭션 ID의 XID 구조에 대한 Java 맵핑을 나타냅니다. 오브젝트는 세부분을 포함하고 있습니다.

- 글로벌 트랜잭션의 형식 ID
- 글로벌 트랜잭션 ID
- 분기 규정자

이 인터페이스에 대한 완전한 세부사항은 JTA 스펙을 참조하십시오.

예: 트랜잭션을 처리하기 위한 JTA 사용은 어플리케이션에서 트랜잭션을 처리하기 위해 JTA를 사용하는 방법을 나타냅니다.

풀 및 분배 트랜잭션에 대한 UDBXADataSource 지원 사용: Java 트랜잭션 API 지원은 연결 풀에 대한 직접 지원을 제공합니다. UDBXADataSource는 ConnectionPoolDataSource의 확장으로서 풀된 XAConnection 오브젝트에 대한 어플리케이션 액세스를 허용합니다. UDBXADataSource는 ConnectionPoolDataSource이므로 UDBXADataSource의 구성과 사용은 오브젝트 풀에 대한 DataSource 지원 사용에 설명된 것과 동일합니다.

XADataSource 등록 정보: ConnectionPoolDataSource에서 제공하는 등록 정보 이외에 XADataSource 인터페이스는 다음의 등록 정보를 제공합니다.

Set 메소드 **(자료 유형)**	**값**	**설명**
setLockTimeout (int)	0 또는 양의 값	양의 값은 트랜잭션 레벨에서 유효한 잠금 시간종료 값(초)입니다. 잠금 시간종료 값 0은 다른 레벨(작업 또는 표)에서는 잠금 시간종료 값이 실행되었더라도 트랜잭션 레벨에서는 잠금 시간종료 값이 실행되지 않았음을 의미합니다. 디폴트 값는 0입니다.

Set 메소드 **(**자료 유형**)**	값	설명
setTransactionTimeout (int)	0 또는 양의 값	양의 값은 유효한 트랜잭션 시간종료 값(초)입니다. 트랜잭션 시간종료 값 0은 트랜잭션 시간종료 값이 실행되지 않았음을 의미합니다. 트랜잭션이 시간종료 값보다 오래 활성 상태였으면 롤백만 수행하도록 표시되며 그 아래에서 작업을 수행하려는 후속 시도는 예외를 발생시킵니다. 디폴트 값는 0입니다.

ResultSet 및 트랜잭션: 앞의 예에 표시된 대로 트랜잭션의 시작과 종료를 분명하게 구별할 뿐만 아니라 잠시 트랜잭션을 일시중단했다가 나중에 재개할 수 있습니다. 트랜잭션 중에 작성된 ResultSet 자원에 대한 여러 가지 시나리오를 제공합니다.

간단한 트랜잭션 종료: 트랜잭션을 종료할 때 이 트랜잭션 아래에서 작성된 모든 열린 ResultSet는 자동으로 닫힙니다. 최대 병렬 처리를 위해 ResultSet 사용을 완료했으면 명시적으로 닫는 것이 바람직합니다. 그러나 트랜잭션 중에 열린 ResultSet에 XAResource.end 호출 이후에 액세스하면 예외가 발생합니다.

이 작동을 보여주는 예: 트랜잭션 종료를 참조하십시오.

일시중단 및 재개: 트랜잭션이 일시중단된 동안 트랜잭션이 활성 상태였을 때 작성된 ResultSet에 대한 액세스는 허용되지 않으며 예외가 발생합니다. 그러나 트랜잭션을 재개하면 ResultSet를 다시 사용할 수 있으며 트랜잭션을 일시중단하기 전에 있었던 상태와 동일한 상태가 됩니다.

이 작동을 보여주는 예: 트랜잭션 일시중단 및 재개를 참조하십시오.

일시중단된 ResultSet 실행: 트랜잭션이 일시중단되어 있는 동안에는 ResultSet에 액세스할 수 없습니다. 그러나 작업을 수행하기 위해 다른 트랜잭션 하에 Statement 오브젝트를 다시 처리할 수 있습니다. JDBC Statement 오브젝트에는 한 번에 하나의 ResultSet만이 있을 수 있으므로(저장 프로시듀어 호출의 여러 동시 ResultSet에 대한 JDBC 3.0 지원 제외) 새로운 트랜잭션의 요구를 충족시키기 위해 일시중단된 트랜잭션에 대한 ResultSet를 닫아야 합니다. 다음은 정확히 발생하는 상황을 보여줍니다.

예: 일시중단된 ResultSets는 이 작동을 보여줍니다.

주: JDBC 3.0을 사용하면 Statement는 저장 프로시듀어 호출에 대해 여러 ResultSet를 동시에 열 수 있지만 한 단위로 취급되며 Statement가 새로운 트랜잭션 하에 다시 처리되면 모두 닫힙니다. 현재 단일 명령문에 대해 두 트랜잭션의 ResultSet를 동시에 활성화하는 것은 불가능합니다.

멀티플렉싱: JTA API는 JDBC 연결에서 트랜잭션의 커플링을 해제하기 위한 것입니다. 이 API를 사용하면 한 트랜잭션에서 여러 연결 작업을 수행하거나 동시에 여러 트랜잭션에서 한 연결 작업을 수행할 수 있습니다. 이것을 **멀티플렉싱**이라고 하며 JDBC만을 사용해서는 수행할 수 없는 많은 복잡한 타스크들을 수행할 수 있습니다.

이 예는 단일 트랜잭션에 대한 여러 연결 작업을 보여줍니다.

이 예는 한 번에 발생하는 여러 트랜잭션과의 단일 연결을 보여줍니다.

JTA 사용에 대한 자세한 정보는 JTA 스펙을 참조하십시오. JDBC 3.0 스펙에는 두 기술을 함께 이용하여 분배 트랜잭션을 지원하는 방법에 대한 정보도 들어 있습니다.

2단계 확약 및 트랜잭션 기록: JTA API는 어플리케이션에 완전히 분배된 2단계 확약 프로토콜의 책임을 구체화합니다. 예에서와 같이, JTA 및 JDBC를 사용하여 JTA 트랜잭션 하에서 데이터베이스에 액세스할 때 어플리케이션은 XAResource.prepare() 및 XAResource.commit() 메소드를 사용하거나 XAResource.commit() 메소드만을 사용하여 변경사항을 확약합니다.

또한 단일 트랜잭션을 사용하여 여러 고유 데이터베이스에 액세스할 때 2단계 확약 프로토콜 및 이러한 데이터베이스에서 트랜잭션 atomicity에 필요한 연관된 기록이 수행되는지 확인하는 것은 어플리케이션의 책임입니다. 일반적으로 어플리케이션 서버 또는 트랜잭션 모니터의 제어 하에 여러 데이터베이스(즉, XAResource)에서의 2단계 확약 처리 및 그 기록이 수행되므로 어플리케이션 자체는 실제로 이 문제를 염려하지 않아도 됩니다.

예를 들어, 어플리케이션은 일부 commit() 메소드를 호출하거나 오류 없이 처리로 부터 복귀할 수 있습니다. 그러면 기초 어플리케이션 서버 또는 트랜잭션 모니터는 단일 분배 트랜잭션에 참여하는 각 데이터베이스(XAResource)에 대한 처리를 시작합니다.

어플리케이션 서버는 2단계 확약 처리 중 광범위한 기록을 사용합니다. 참여하는 각 데이터베이스(XAResource)에 대해 차례대로 XAResource.prepare() 메소드를 호출한 후에 참여하는 각 데이터베이스(XAResource)에 대해 XAResource.commit() 메소드를 호출합니다.

이 처리 중 실패가 발생한 경우 어플리케이션 서버의 트랜잭션 모니터 기록부를 사용하면 어플리케이션 서버 자체가 그 뒤에 JTA API를 사용하여 분배 트랜잭션을 회복할 수 있습니다. 어플리케이션 서버 또는 트랜잭션 모니터의 제어 하에 이루어지는 이 회복을 통해 어플리케이션 서버는 참여하는 각 데이터베이스(XAResource)에서 인식된 상태에 트랜잭션을 놓습니다. 그러면 모든 참여 데이터베이스에서 분배된 전체 트랜잭션은 잘 알려진 상태에 놓입니다.

예: 트랜잭션을 처리하기 위한 JTA 사용:

이것은 JTA(Java(TM) Transaction API)를 사용하여 어플리케이션의 트랜잭션을 처리하는 방법을 보여주는 예입니다.

예: 트랜잭션을 처리하기 위한 JTA 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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 116 페이지의 『JTA를 사용한 트랜잭션』
         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();
             }
         }
     }
}
```

«

예: 트랜잭션에서 작용하는 복수 접속:

»

단일 트랜잭션에서 작용하는 복수 접속을 사용하는 방법 예입니다.

예: 트랜잭션에서 작용하는 복수 접속

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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());
        }
      }
   }
}
```

예: 복수 트랜잭션으로 접속 사용:

다음은 복수 접속으로 하나의 접속을 사용하는 방법을 보여주는 예입니다.

예: 복수 트랜잭션으로 접속 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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();
           }
        }
    }
}
```

예: 일시중단된 ResultSets:

다음은 작업을 수행하기 위해 다른 트랜잭션에서 Statement 오브젝트를 다시 처리하는 방법을 나타낸 예입니다.

예: 일시중단된 ResultSet

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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 116 페이지의 『JTA를 사용한 트랜잭션』
         // 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();
       }
```

```
        }
    }
}
```

예: 트랜잭션 종료:

어플리케이션에서 트랜잭션 종료의 예입니다.

예: 트랜잭션 종료

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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 116 페이지의 『JTA를 사용한 트랜잭션』 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();
             }
        }
    }
}
```

예: 트랜잭션 일시중단 및 재개:

트랜잭션의 일시중단 및 재개의 예입니다.

예: 트랜잭션 일시중단 및 재개

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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. 116 페이지의 『JTA를 사용한 트랜잭션』 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();
```

```
            }
        }
    }
}
```

명령문 유형

Statement 인터페이스와 해당 PreparedStatement 및 CallableStatement 서브클래스를 사용하여 데이터베이스에 대해 SQL(구조화 조회 언어) 명령을 처리합니다. SQL문으로 인해 ResultSet 오브젝트가 생성됩니다.

Statement 인터페이스의 서브클래스는 Connection 인터페이스에 대해 많은 메소드를 사용하여 작성합니다. 단일 Connection 오브젝트에는 여러 개의 Statement 오브젝트가 동시에 작성될 수 있습니다. 이전 릴리스에서는 작성할 수 있는 Statement 오브젝트의 정확한 수를 제공했습니다. 이 릴리스에서는 여러 가지 유형의 Statement 오브젝트가 데이터베이스 엔진 내에서 서로 다른 수의 ″핸들″을 사용하기 때문에 이러한 작업이 불가능합니다. 따라서 사용하고 있는 Statement 오브젝트의 유형은 한 번에 하나의 연결 하에 활동 상태가 될 수 있는 명령문의 수에 영향을 줍니다.

어플리케이션은 메소드 Statement.close를 호출하여 어플리케이션이 명령문 처리를 완료했음을 나타냅니다. 모든 Statement 오브젝트는 이를 작성한 연결이 닫히면 닫힙니다. 그러나 Statement 오브젝트를 닫기 위해 이 작동에 완전히 의존해서는 안됩니다. 예를 들어, 연결을 명시적으로 닫는 대신 연결 풀을 사용하도록 어플리케이션이 변경되면 연결이 전혀 닫히지 않기 때문에 어플리케이션에는 명령문 핸들이 “부족”합니다. Statement 오브젝트가 더 이상 필요하지 않게 되었을 때 즉시 이를 닫으면 명령문이 사용하고 있는 외부 데이터베이스 자원이 즉시 해제됩니다.

원시 JDBC 드라이버는 명령문 손실을 감지하려고 시도하며 사용자 대신 이를 처리합니다. 그러나 이 지원에 의존하면 성능이 감소됩니다.

명령문 및 서브클래스의 사용은 다음을 참조하십시오.

명령문
Statement 오브젝트는 정적 SQL문 처리와 이로 인해 작성된 결과를 확보하는 데 사용합니다.

PreparedStatements
PreparedStatement는 Statement 인터페이스의 서브클래스이며 SQL문에 매개변수를 추가하는 지원을 제공합니다.

CallableStatement
CallableStatement는 PreparedStatement 인터페이스를 확장한 것이며 PreparedStatement가 제공하는 입력 매개변수 지원 이외에 출력 및 입/출력 매개변수에 대한 지원을 제공합니다.

Statement를 확장하는 PreparedStatement를 CallableStatement가 확장하는 상속 계층으로 인해 각 인터페이스의 피처를 인터페이스를 확장하는 클래스에서 사용할 수 있습니다. 예를 들어, Statement 클래스의 피처는 PreparedStatement 및 CallableStatement 클래스에서도 지원합니다. Statement 클래스에서의 executeQuery, executeUpdate 및 execute 메소드는 예외입니다. 이러한 메소드들을 PreparedStatement 또는 CallableStatement 오브젝트와 함께 사용하려고 하면 동적으로 처리할 SQL문을 보고 예외를 유발합니다.

명령문:

Statement 오브젝트는 정적 SQL문의 처리와 이로 인해 작성된 결과를 확보하는 데 사용합니다. 각 Statement 오브젝트에 대해 한 번에 하나의 ResultSet만을 열 수 있습니다. SQL문을 처리하는 모든 명령문 메소드는 명령문의 현재 ResultSet가 열려 있으면 내재적으로 닫습니다.

명령문 작성: createStatement 메소드를 사용하여 Connection 오브젝트에서 Statement 오브젝트를 작성합니다. 예를 들어, 이름이 conn인 Connection 오브젝트가 있다고 가정하면 다음의 코드 행은 SQL문을 데이터베이스에 전달하기 위해 Statement 오브젝트를 작성합니다.

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

ResultSet 특성 지정: ResultSet의 특성은 결과적으로 ResultSet를 작성하는 명령문과 연관되어 있습니다. Connection.createStatement 메소드를 사용하면 이러한 ResultSet 특성을 지정할 수 있습니다. 다음은 createStatement 메소드에 대해 유효한 호출의 일부 예입니다.

예: createStatement 메소드

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// 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);
```

이 특성에 대한 자세한 정보는 ResultSets를 참조하십시오.

명령문 처리: Statement 오브젝트를 사용한 SQL문 처리는 executeQuery(), executeUpdate() 및 execute() 메소드를 통해 수행됩니다.

SQL 조회에서 결과가 리턴됩니다.: ResultSet 오브젝트를 리턴하는 SQL 조회 명령문을 처리할 경우 executeQuery() 메소드를 사용해야 합니다. Statement 오브젝트의 executeQuery 메소드를 사용하여 ResultSet를 확보하는 프로그램 예를 참조할 수 있습니다.

주: executeQuery를 사용하여 처리한 SQL문이 ResultSet를 리턴하지 않으면 SQLException이 유발됩니다.

SQL문에 대한 갱신 계수 리턴: SQL이 DDL(data definition language) 명령문이나 갱신 계수를 리턴하는 DML(data manipulation language) 명령문이 되는 것으로 알려진 경우 executeUpdate() 메소드를 사용해야 합니다. StatementExample 프로그램은 Statement 오브젝트의 executeUpdate 메소드를 사용합니다.

예상 리턴을 알 수 없는 SQL문 처리: SQL문의 유형이 알려지지 않는 경우 execute 메소드를 사용해야 합니다. 이 메소드를 처리한 후 JDBC 드라이버는 SQL문이 API 호출을 통해 생성한 결과의 유형을 어플리케이션에 알릴 수 있습니다. execute 메소드는 결과가 최소한 하나의 ResultSet이면 true를 리턴하고 리턴 값이 갱신 계수이면 false를 리턴합니다. 이 정보가 제공된 상태에서 어플리케이션은 명령문 메소드의 getUpdateCount 또는 getResultSet를 사용하여 SQL문 처리에서 리턴 값을 검색할 수 있습니다. StatementExecute 프로그램은 Statement 오브젝트에서 execute 메소드를 사용합니다. 이 프로그램은 SQL문인 매개변수를 전달할 것으로 예상합니다. 사용자가 제공한 SQL의 텍스트를 보지 않고 프로그램은 명령문을 처리하고 처리된 것에 대한 정보를 판별합니다.

주: 결과가 ResultSet일 때 getUpdateCount 메소드를 호출하면 -1을 리턴합니다. 결과가 갱신 계수일 때 getResultSet 메소드를 호출하면 널(null)을 리턴합니다.

cancel 메소드: 원시 JDBC 드라이버의 메소드들을 동기화하여 동일한 오브젝트에 대해 실행 중인 두 스레드가 오브젝트를 손상시키지 못하도록 방지할 수 있습니다. 예외는 cancel 메소드입니다. cancel 메소드는 하나의 스레드가 장기 수행 중인 SQL문을 동일한 오브젝트에 대한 다른 스레드에서 중단하는 데 사용할 수 있습니다. 원시 JDBC 드라이버는 스레드에 작업을 중단하도록 강요할 수 없습니다. 수행하고 있던 타스크에 상관 없이 중단하도록 요구할 수만 있습니다. 이와 같은 이유로 인해 취소된 명령문을 중단하려면 여전히 시간이 많이 소요됩니다. cancel 메소드를 사용하여 시스템에서 이탈 SQL 조회를 정지할 수 있습니다.

예: Statement 오브젝트의 executeUpdate 메소드 사용:

다음은 Statement 오브젝트의 executeUpdate 메소드를 사용하는 방법을 보여주는 예입니다.

예: Statement 오브젝트의 executeUpdate 메소드 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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:

PreparedStatement는 Statement 인터페이스를 확장하며 SQL문에 매개변수를 추가하기 위한 지원을 제공합니다.

데이터베이스에 전달되는 SQL문은 사용자에게 결과를 리턴하기 위해 2단계 프로세스를 수행합니다. 먼저 준비한 다음 처리합니다. Statement 오브젝트를 사용하면 이 두 단계는 어플리케이션에게 한 단계로 보입니다. PreparedStatement는 이 두 단계를 따로 분리할 수 있게 합니다. 오브젝트를 작성하면 준비 단계가 이루어지고 PreparedStatement 오브젝트에 대해 executeQuery, executeUpdate 또는 execute 메소드를 호출하면 처리 단계가 이루어집니다.

SQL 처리를 분리된 단계로 구분할 수 있는 것은 매개변수 마커의 추가 없이는 의미가 없습니다. 매개변수 마커는 어플리케이션에 있기 때문에 준비 시에는 특정 값이 없지만 처리 시간 이전에 하나의 값을 제공함을 데이터베이스에 알릴 수 있습니다. 매개변수 마커는 SQL문에서 의문 부호로 표시합니다.

매개변수 마커는 특정 요구에 사용하는 일반 SQL문을 작성할 수 있게 합니다. 예를 들어, 다음의 SQL 조회 명령문을 사용하십시오.

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

이것은 하나의 값 즉, 이름이 Dettinger인 직원에 대한 정보만을 리턴하는 특정 SQL문입니다. 매개변수 마커를 추가하여 명령문이 보다 유연해질 수 있습니다.

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

간단하게 매개변수 마커를 값으로 설정하면 표에서 직원에 대한 정보를 확보할 수 있습니다.

앞의 Statement 예는 준비 단계를 한 번만 수행하고 매개변수에 대해 다른 값을 사용하여 반복적으로 처리할 수 있기 때문에 여러 Statement에서 상당한 성능 개선을 제공합니다.

주: PreparedStatement 사용은 원시 JDBC 드라이버의 명령문 풀을 지원하기 위한 요구사항입니다.

PreparedStatement 작성: prepareStatement 메소드는 새 PreparedStatement 오브젝트를 작성할 때 사용됩니다. createStatement 메소드와 달리 PreparedStatement 오브젝트가 작성되면 SQL문이 제공됩니다. 이 때 SQL문이 사전컴파일됩니다. 예를 들어, 이름이 conn인 Connection 오브젝트가 이미 있다고 가정하면 다음의 예는 PreparedStatement 오브젝트를 작성하고 데이터베이스 내에서 처리할 수 있도록 SQL문을 준비합니다.

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

ResultSet 특성 및 자동 생성 키 지원 지정: createStatement 메소드와 마찬가지로 prepareStatement 메소드는 ResultSet 특성을 지정하기 위한 지원을 제공할 부담이 큽니다. prepareStatement 메소드에도 자동 생성 키에 대한 다양한 작업이 있습니다. 다음은 prepareStatement 메소드에 대한 유효한 호출의 일부 예입니다.

예: prepareStatement 메소드

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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);
```

매개변수 처리: PreparedStatement 오브젝트를 실행하기 전에 각 매개변수 마커를 일정한 값으로 설정해야 합니다. PreparedStatement 오브젝트는 매개변수 설정을 위해 여러 가지 메소드를 제공합니다. 모든 메소드는 형식 set<Type>이며 여기서 <Type>은 Java 자료 유형입니다. 이 메소드의 예로는 setInt, setLong, setString, setTimestamp, setNull 및 setBlob가 있습니다. 이러한 거의 모든 메소드는 두 개의 매개변수를 사용합니다.

- 첫 번째 매개변수는 명령문 내에 있는 매개변수의 색인입니다. 매개변수 마커는 1부터 시작하여 번호가 매겨집니다.
- 두 번째 매개변수는 매개변수를 설정하는 값입니다. setBinaryStream에서의 길이 매개변수와 같이 추가 매개변수가 있는 두 개의 set<Type> 메소드가 있습니다.

자세한 정보는 java.sql 패키지에 대한 Javadoc를 참조하십시오. ps 오브젝트에 대한 앞의 예에서 준비된 SQL문이 제공된 상태에서 다음의 코드는 처리하기 전에 매개변수 값을 지정하는 방법을 보여줍니다.

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

설정되지 않은 매개변수 마커를 사용하여 PreparedStatement를 처리하려고 시도하면 SQLException이 유발됩니다.

주: 매개변수 마커를 설정하면 다음의 상태가 발생하지 않는 한 프로세스 간에 동일한 값을 유지합니다.

- set 메소드에 대한 다른 호출로 인해 값이 변경된 경우.
- clearParameters 메소드를 호출했을 때 값이 제거된 경우.

clearParameters 메소드는 모든 매개변수가 설정되지 않은 것으로 플래그를 붙입니다. clearParameters를 호출한 후에 모든 매개변수는 다음 프로세스 이전에 set 메소드를 다시 호출해야 합니다.

ParameterMetaData 지원: 새로운 ParameterMetaData 인터페이스를 통해 매개변수에 대한 정보를 검색할 수 있습니다. 이 지원은 ResultSetMetaData를 충족시키는 것이며 이와 유사합니다. 정밀도, 스케일, 자료 유형, 자료 유형 이름과 같은 정보와 매개변수가 널값을 허용하는지의 여부를 모두 제공합니다.

어플리케이션 프로그램에서 이 새로운 지원을 사용하는 방법에 대해서는 예: ParameterMetaData를 참조하십시오.

PreparedStatement 프로세스:

PreparedStatement 오브젝트를 사용한 SQL문 처리는 Statement 오브젝트의 처리와 같이 executeQuery, executeUpdate 및 execute 메소드를 사용하여 수행합니다. Statement 버전과 달리 오브젝트를 작성했을 때 SQL문이 이미 제공되었기 때문에 이들 메소드에 대한 매개변수가 전달되지 않습니다. PreparedStatement는 Statement를 확장하기 때문에 어플리케이션은 SQL문을 사용하는 executeQuery, executeUpdate 및 execute 메소드의 버전을 호출하려고 시도할 수 있습니다. 이렇게 하면 SQLException이 유발됩니다.

SQL 조회에서 결과 리턴: ResultSet 오브젝트를 리턴하는 SQL 조회 명령문을 실행할 경우, executeQuery 메소드를 사용해야 합니다. PreparedStatementExample 프로그램은 PreparedStatement 오브젝트의 executeQuery 메소드를 사용하여 ResultSet를 확보합니다.

주: executeQuery 메소드를 사용하여 처리한 SQL문이 ResultSet를 리턴하지 않으면 SQLException이 유발됩니다.

SQL문에 대한 갱신 계수 리턴: SQL이 DDL(data definition language) 명령문이나 갱신 계수를 리턴하는 DML(data manipulation language) 명령문이 되는 것으로 알려진 경우, executeUpdate 메소드를 사용해야 합니다. PreparedStatementExample 샘플 프로그램은 PreparedStatement 오브젝트의 executeUpdate 메소드를 사용합니다.

예상 리턴을 알 수 없는 SQL문 처리: SQL문의 유형이 알려지지 않는 경우, execute 메소드를 사용해야 합니다. 이 메소드를 처리한 후에 JDBC 드라이버는 SQL문이 API 호출을 통해 생성한 결과의 유형을 어플리케이션에 알려줄 수 있습니다. execute 메소드는 결과가 최소한 하나의 ResultSet이면 true를 리턴하고 리턴

값이 갱신 계수이면 false를 리턴합니다. 이 정보가 제공된 상태에서 어플리케이션은 getUpdateCount 또는 getResultSet 명령문 메소드를 사용하여 SQL문 처리에서 리턴 값을 검색할 수 있습니다.

주: 결과가 ResultSet일 때 getUpdateCount 메소드를 호출하면 -1을 리턴합니다. 결과가 갱신 계수일 때 getResultSet 메소드를 호출하면 널(null)을 리턴합니다.

예: ResultSet를 얻기 위해 PreparedStatement 사용:

PreparedStatement 오브젝트의 executeQuery 메소드를 사용하여 ResultSet를 얻는 예입니다.

예: ResultSet를 얻기 위해 PreparedStatement 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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.");
            }

        }
    }
}
```

예: ParameterMetaData:

ParameterMetaData 인터페이스를 사용하여 매개변수에 대한 정보를 검색하는 예입니다.

예: ParameterMetaData

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

CallableStatements:

CallableStatement 인터페이스는 PreparedStatement를 확장하며 출력 및 입/출력 매개변수에 대한 지원을 제공합니다. CallableStatement 인터페이스에는 PreparedStatement 인터페이스에서 제공하는 입력 매개변수에 대한 지원도 있습니다.

CallableStatement 인터페이스를 사용하면 저장 프로시듀어를 호출할 때 SQL문을 사용할 수 있습니다. 저장 프로시듀어는 데이터베이스 인터페이스가 있는 프로그램입니다. 이러한 프로그램은 다음을 소유합니다.

- 입력 및 출력 매개변수 또는 입력 및 출력에 모두 해당하는 매개변수가 있을 수 있습니다.
- 리턴값이 있을 수 있습니다.
- 여러 ResultSet를 리턴하는 기능이 있습니다.

개념적으로는 JDBC에서 저장 프로시듀어가 데이터베이스에 대한 단일 호출이지만 저장 프로시듀어와 연관된 프로그램은 수백 개의 데이터베이스 요구를 처리할 수 있습니다. 저장 프로시듀어 프로그램은 일반적으로 SQL 문을 사용하여 수행되지 않는 수많은 다른 프로그램 상 필요한 타스크도 수행할 수 있습니다.

CallableStatements는 준비 및 처리 단계의 결합을 분리하는 PreparedStatement 모델을 따르기 때문에 최적화된 재사용의 가능성이 잠재되어 있습니다(자세한 내용은 PreparedStatement 참조). 저장 프로시듀어의 SQL 문이 프로그램에 바인드되므로 정적 SQL로 처리되며 이러한 방식으로 추가 수행 이점을 얻을 수 있습니다. 많은 데이터베이스 작업을 하나의 재사용 가능한 데이터베이스 호출에 캡슐화하는 것은 저장 프로시듀어를 최적으로 사용하는 한 예입니다. 이 호출만이 네트워크를 통해 다른 시스템으로 이동되지만 요구는 리모트 시스템에서 많은 작업을 수행할 수 있습니다.

CallableStatement 작성: prepareCall 메소드는 새로운 CallableStatement 오브젝트를 작성할 때 사용합니다. prepareStatement에서와 마찬가지로 CallableStatement 오브젝트 작성시 SQL문이 제공됩니다. 이때 SQL 문이 사전컴파일됩니다. 예를 들어, 이름이 conn인 Connection 오브젝트가 이미 있다고 가정하면 다음은 CallableStatement 오브젝트를 작성하고 데이터베이스 내에서 SQL문을 처리하도록 준비하는 준비 단계를 완료합니다.

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

ADDEMPLOYEE 저장 프로시듀어는 새로운 직원명, 해당 사회 보장 번호 및 소속된 관리자의 사용자 ID에 대한 입력 매개변수를 사용합니다. 이 정보를 통해 많은 회사 데이터베이스 표를 시작일, 과, 부서 등과 같은 직원에 대한 정보로 갱신할 수 있습니다. 또한 저장 프로시듀어는 해당 직원에 대한 표준 사용자 ID와 전자

우편 주소를 생성할 수 있는 프로그램입니다. 저장 프로시듀어는 이니셜 사용자명 및 암호를 사용하여 고용주에게 전자 우편을 보낼 수도 있으며 고용주는 직원에게 해당 정보를 제공할 수 있습니다.

ADDEMPLOYEE 저장 프로시듀어는 리턴값을 갖기 위해 설정합니다. 리턴 코드가 실패하면 호출 프로그램이 사용할 수 있는 성공 또는 실패 코드일 수 있습니다. 리턴 값은 새로운 직원의 회사 ID 번호로 정의할 수도 있습니다. 마지막으로 저장 프로시듀어가 내부적으로 조회를 처리했을 수 있으며 이러한 조회의 ResultSet를 호출 프로그램이 사용할 수 있도록 계속 열어 두었을 것입니다. 새로운 모든 직원의 정보를 조회하고 리턴된 ResultSet를 통해 호출자가 이 정보를 사용할 수 있게 하는 것이 합리적입니다.

이러한 각 타스크 유형을 수행하는 방법은 다음 섹션에서 다룹니다.

ResultSet 특성 및 자동 생성 키 지원 지정: createStatement 및 prepareStatement와 마찬가지로 ResultSet 특성을 지정하기 위한 지원을 제공하는 여러 가지 prepareCall 버전이 있습니다. prepareStatement와 달리 prepareCall 메소드는 CallableStatements에서 자동 생성된 키에 대한 여러 가지 작업을 제공하지 않습니다 (JDBC 3.0은 이 개념을 지원하지 않습니다). 다음은 prepareCall 메소드에 대한 유효한 호출의 일부 예입니다.

예: prepareCall 메소드

```
// 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);
```

매개변수 처리: 언급한 바와 같이 CallableStatement 오브젝트는 세 가지 유형의 매개변수를 사용합니다.

- **IN**
  IN 매개변수는 PreparedStatements와 동일한 방식으로 처리합니다. 상속된 PreparedStatement 클래스의 다양한 set 메소드를 사용하여 매개변수를 설정합니다.
- **OUT**
  OUT 매개변수는 registerOutParameter 메소드를 사용하여 처리합니다. registerOutParameter의 가장 일반적인 양식은 첫 번째 매개변수로 색인 매개변수를 사용하며 두 번째 매개변수로 SQL 유형을 사용하는 것입니다. 그러면, 명령문을 처리할 때 매개변수에서 예상 자료를 JDBC 드라이버에 알려줍니다. java.sql 패키지 Javadoc에서 두 가지 종류의 registerOutParameter 메소드를 찾을 수 있습니다.
- **INOUT**
  INOUT 매개변수에서는 IN 매개변수와 OUT 매개변수 둘 다에 대한 작업을 수행해야 합니다. 각 INOUT 매개변수에 대해 명령문을 처리하기 전에 먼저 set 메소드와 registerOutParameter 메소드를 호출해야 합니다. 매개변수를 설정하거나 등록하지 못하면 명령문이 처리될 때 SQLException이 발생합니다.

자세한 정보는 예: 입력 및 출력 매개변수를 사용한 프로시듀어 작성을 참조하십시오.

PreparedStatements와 마찬가지로 set 메소드를 다시 호출하지 않는 한 프로세스들 사이에서 CallableStatement 매개변수 값을 동일하게 유지합니다. clearParameters 메소드는 출력하도록 등록된 매개변수에 영향을 주지 않습니다. clearParameters를 호출한 후에 모든 IN 매개변수를 다시 값으로 설정해야 하지만 모든 OUT 매개변수는 다시 등록할 필요가 없습니다.

주: 매개변수의 개념을 매개변수의 색인과 혼동하지 마십시오. 저장 프로시듀어 호출은 일정한 수의 매개변수가 전달될 것으로 예상합니다. 특정 SQL문에 ? 문자(매개변수 마커)가 포함되어 있으며 런타임에 제공될 값들을 나타냅니다. 두 개념간의 차이를 알려면 다음의 예를 고려하십시오.

```
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
```

이름을 사용한 저장 프로시듀어 매개변수 액세스: 저장 프로시듀어에 대한 매개변수에는 연관된 이름이 있으며 다음의 저장 프로시듀어 선언을 보십시오.

예: 저장 프로시듀어 매개변수

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

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

이름이 PARM1인 하나의 정수 매개변수가 있습니다. JDBC 3.0에는 이름과 색인을 사용하여 저장 프로시듀어 매개변수를 지정하는 지원이 있습니다. 이 프로시듀어에 대한 CallableStatement를 설정하는 코드는 다음과 같습니다.

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

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

자세한 정보는 CallableStatements 프로세스를 참조하십시오.

CallableStatements 프로세스:

CallableStatement 오브젝트를 사용한 SQL 저장 프로시듀어 호출은 PreparedStatement 오브젝트와 함께 사용하는 동일한 메소드를 사용하여 수행합니다.

저장 프로시듀어 결과 리턴: SQL query 명령문이 저장 프로시듀어 내에서 처리되는 경우에 저장 프로시듀어를 호출하는 프로그램이 조회 결과를 사용할 수 있습니다. 저장 프로시듀어 내에서 여러 조회를 호출할 수 있으며 호출 프로그램은 사용 가능한 모든 ResultSet를 처리할 수 있습니다.

자세한 정보는 예: 복수 ResultSet로 프로시듀어 작성을 참조하십시오.

주: 저장 프로시듀어가 executeQuery를 사용하여 처리되며 ResultSet를 리턴하지 않으면 SQLException이 발생합니다.

ResultSet에 동시 액세스: 『저장 프로시듀어 결과 리턴』은 ResultSets 및 저장 프로시듀어에 대해 다루며, 모든 JDK(Java(TM) Development Kit) 릴리스에 대해 작업하는 예를 제공합니다. 이 예에서 저장 프로시듀어가 연 첫 번째 ResultSet부터 마지막 열린 ResultSet까지의 순서로 ResultSet가 처리됩니다. 다음 ResultSet를 사용하기 전에 하나의 ResultSet를 닫습니다.

JDK 1.4에는 저장 프로시듀어에서 동시에 여러 ResultSet에 대해 작업하는 지원이 있습니다.

주: 이 피처는 V5R2에서 명령행 인터페이스(CLI)를 통해 기초 시스템 지원에 추가되었습니다. 결과적으로 V5R2 이전에 시스템에서 실행 중인 JDK 1.4는 이 지원을 사용할 수 없습니다.

저장 프로시듀어에 대한 갱신 계수 리턴: 저장 프로시듀어에 대한 갱신 계수 리턴은 JDBC 스펙에 설명된 피처이지만 현재 iSeries 플랫폼에서 지원되지 않습니다. 저장 프로시듀어 호출에서 여러 갱신 계수를 리턴하는 방법은 없습니다. 저장 프로시듀어 내에서 처리된 SQL문의 갱신 계수가 필요한 경우에 값을 리턴하는 방법은 두 가지입니다.

- 출력 매개변수로서 값 리턴.
- 매개변수에서 리턴값으로 값을 다시 전달. 이것은 출력 매개변수의 특별한 케이스입니다. 자세한 정보는 『리턴값이 있는 저장 프로시듀어 처리』를 참조하십시오.

예상 리턴을 알 수 없는 저장 프로시듀어 처리: 저장 프로시듀어의 결과를 알 수 없는 경우에 실행 메소드를 사용해야 합니다. 이 메소드를 처리한 후에 JDBC 드라이버는 저장 프로시듀어가 API 호출을 통해 생성한 결과의 유형을 어플리케이션에 알려줄 수 있습니다. 결과가 하나 이상의 ResultSet이면 실행 메소드는 참을 리턴합니다. 갱신 계수는 저장 프로시듀어 호출에서 비롯된 것이 아닙니다.

리턴값이 있는 저장 프로시듀어 처리: iSeries 플랫폼은 함수의 리턴 값과 유사한 리턴 값이 있는 저장 프로시듀어를 지원합니다. 저장 프로시듀어의 리턴 값은 다른 매개변수 마크와 비슷하게 레이블을 붙이며 저장 프로시듀어 호출에 의해 지정되는 방법과 같이 레이블을 붙입니다. 이 예는 다음과 같습니다.

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

저장 프로시듀어 호출의 리턴 값은 항상 정수 유형이며 다른 출력 매개변수처럼 등록해야 합니다.

자세한 정보는 예: 리턴값으로 프로시듀어 작성을 참조하십시오.

«

예: IBM Developer Kit for Java용 CallableStatement 인터페이스:

다음은 CallableStatement 인터페이스 사용법의 예입니다.

예: CallableStatement 인터페이스

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

자세한 정보는 CallableStatements를 참조하십시오.

예: 복수 ResultSet로 프로시듀어 작성:

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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();
       }
   }
}
```

예: 입력 및 출력 매개변수로 프로시듀어 작성:

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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();
        }
    }
}
```

예: 리턴 값으로 프로시듀어 작성:

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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.  Note that return value support
            // is new in V4R5.
            String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX3 " +
                         " LANGUAGE SQL SPECIFIC MYLIBRARY.SQLSPEX3 " +
                         " EX3: BEGIN " +
                         "      RETURN 1976; " +
                         " END EX3 ";

           try {
                s.executeUpdate(sql);
            } catch (SQLException e) {
                // 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();
        }
    }
}
```

ResultSets

ResultSet 인터페이스는 조회를 실행하여 생성된 결과에 대한 액세스를 제공합니다. 개념적으로 ResultSet의 자료는 일정한 수의 열과 일정한 수의 행이 있는 표로 간주할 수 있습니다. 디폴트로 표의 행이 순차적으로 검색됩니다. 한 행 내에서 열 값은 임의 순서로 액세스할 수 있습니다.

ResultSet 오브젝트를 사용하려면 다음을 참조하십시오.

ResultSet 특성
이 절에서는 다음과 같은 ResultSet 특성을 설명합니다.

- ResultSet 유형
- 동시성
- Connection 오브젝트를 확약하여 ResultSet를 닫는 능력
- ResultSet 특성 지정

커서 이동
iSeries JDBC(Java(TM) Database Connectivity) 드라이버는 화면이동이 가능한 ResultSet를 지원합니다. 화면이동이 가능한 ResultSet를 사용하면 여러 가지 커서 배치 메소드를 사용하여 순서에 상관 없이 자료 행을 처리할 수 있습니다.

ResultSet 자료 검색
ResultSet 오브젝트가 행에 대한 열 자료를 확보하는 메소드를 제공하는 방법을 배우십시오.

ResultSet 변경
iSeries JDBC 드라이버를 사용하면 다음의 타스크를 수행하여 ResultSet를 변경할 수 있습니다.

- 행 갱신
- 행 삭제
- 행 삽입
- 찾은 갱신사항 변경

ResultSet 작성
Statement, PreparedStatement 또는 CallableStatement 인터페이스가 제공하는 executeQuery 메소드를

사용하여 ResultSet 오브젝트를 작성할 수 있습니다. 이 절에서는 ResultSet 오브젝트가 어플리케이션에서 더 이상 필요없게 되었을 때 이를 닫는 방법도 설명합니다.

ResultSet 특성:

디폴트로 작성된 모든 ResultSet의 유형은 전달 전용이며 동시성을 읽기 전용이고 커서는 확약 경계 이상으로 보유합니다. 이에 대한 예외는 확약하면 커서를 내부적으로 닫도록 WebSphere가 현재 커서 보유 디폴트를 변경하는 경우입니다. 이러한 특성은 Statement, PreparedStatement 및 CallableStatement 오브젝트에서 액세스할 수 있는 메소드를 통해 구성 가능합니다.

ResultSet 유형: ResultSet 유형은 ResultSet에 대해 다음을 지정합니다.

- ResultSet가 화면이동 가능한지의 여부.
- ResultSet 인터페이스에서 상수가 정의한 JDBC(Java(TM) Database Connectivity) ResultSet의 유형.

이 ResultSet 유형의 정의는 다음과 같습니다:

TYPE_FORWARD_ONLY
ResultSet의 처음부터 끝까지 처리하는 데에만 사용할 수 있는 커서. 이것이 디폴트 유형입니다.

TYPE_SCROLL_INSENSITIVE
ResultSet를 통해 다양한 방법으로 화면이동하는 데 사용할 수 있는 커서. 이 커서 유형은 열려 있는 동안 데이터베이스에 대해 변경된 사항에는 민감하지 않습니다. 여기에는 조회를 처리할 때나 자료를 페치할 때 조회를 충족시키는 행이 들어 있습니다.

TYPE_SCROLL_SENSITIVE
ResultSet를 통해 다양한 방법으로 화면이동하는 데 사용할 수 있는 커서. 이 커서 유형은 열려 있는 동안 데이터베이스에 대해 변경된 사항에 민감합니다. 데이터베이스에 대해 변경된 사항은 ResultSet 자료에 직접적인 영향을 미칩니다.

JDBC 1.0 ResultSet는 항상 전달 전용입니다. JDBC 2.0에서는 화면이동 커서를 추가했습니다.

주: 블록화 가능 및 블록 크기 연결 등록 정보는 TYPE_SCROLL_SENSITIVE 커서의 민감성 정도에 영향을 줍니다. 블록화는 JDBC 드라이버 계층 자체에서 자료를 캐싱하여 성능을 향상시킵니다.

표에 행을 삽입할 때 민감한 ResultSet와 민감하지 않은 ResultSet 사이의 차이점을 나타낸 예: 민감한 ResultSet와 민감하지 않은 ResultSet를 참조하십시오.

ResultSet의 민감성에 기초하여 SQL문의 변경이 where 절에 미치는 영향을 보여주는 예: ResultSet 민감성을 참조하십시오.

동시성: 동시성은 ResultSet의 갱신 가능 여부를 판별합니다. 유형은 역시 ResultSet 인터페이스의 상수가 정의합니다. 사용할 수 있는 동시성 설정은 다음과 같습니다:

CONCUR_READ_ONLY

데이터베이스로부터 자료를 읽는 데에만 사용할 수 있는 ResultSet. 이것이 디폴트 설정입니다.

CONCUR_UPDATEABLE

변경할 수 있는 ResultSet. 이러한 변경사항은 기초 데이터베이스에 넣을 수 있습니다. 자세한 정보는 ResultSet 변경을 참조하십시오.

JDBC 1.0 ResultSet는 항상 전달 전용입니다. JDBC 2.0에서는 갱신 가능한 ResultSet를 추가했습니다.

주: JDBC 스펙에 따르면 여러 값들을 함께 사용할 수 없는 경우에 JDBC 드라이버는 ResultSet 동시성 설정의 ResultSet 유형을 변경할 수 있습니다. 이러한 경우에 JDBC 드라이버는 Connection 오브젝트에 경고를 보냅니다.

어플리케이션이 TYPE_SCROLL_INSENSITIVE, CONCUR_UPDATEABLE ResultSet를 지정하는 경우가 있습니다. 자료의 사본을 작성하여 데이터베이스 엔진에서 민감하지 않도록 구현합니다. 그러면 이 사본을 통해 기초 데이터베이스를 갱신할 수 없게 됩니다. 이 조합을 지정하면 드라이버는 민감성을 TYPE_SCROLL_SENSITIVE로 변경하고 요구가 변경되었음을 나타내는 경고를 작성합니다.

보유성: 보유성 특성은 Connection 오브젝트에 대한 호출 확약이 ResultSet를 닫는지의 여부를 판별합니다. 보유성 특성에 대한 작업을 위한 JDBC API는 버전 3.0에서 새로운 기능입니다. 그러나 원시 JDBC 드라이버는 연결 하에 작성된 모든 ResultSet에 대해 이 디폴트를 지정할 수 있게 하는 여러 릴리스에 연결 등록 정보를 제공했습니다(Connection 등록 정보 및 DataSource 등록 정보 참조). API 지원은 연결 등록 정보에 대한 설정을 대체합니다. 보유성 특성에 대한 값은 ResultSet 상수가 정의하며 다음과 같습니다.

HOLD_CURSOR_OVER_COMMIT

commit 절을 호출하면 열려 있는 모든 커서가 계속 열려 있습니다. 이것이 원시 JDBC 디폴트 값입니다.

CLOSE_CURSORS_ON_COMMIT

commit 절을 호출하면 열려 있는 모든 커서가 닫힙니다.

주: 연결에 대해 롤백을 호출하면 열려 있는 모든 커서를 항상 닫습니다. 이는 약간 알려져 있는 사실이지만 데이터베이스가 커서를 처리하는 일반적인 방법입니다.

JDBC 스펙에 따르면 커서 보유성의 디폴트는 구현에서 정의합니다. 일부 플랫폼은 CLOSE_CURSORS_ON_COMMIT를 디폴트로 사용하도록 선택합니다. 이 점은 일반적으로 대부분의 어플리케이션에서 문제가 되지 않지만 확약 경계를 넘어서 커서에 대해 작업하고 있는 경우에 작업하고 있는 드라이버가 수행하는 작업을 알고 있어야 합니다. Toolbox JDBC 드라이버도 HOLD_CURSORS_ON_COMMIT 디폴트를 사용하지만 Windows(R) NT용 UDB의 JDBC 드라이버에서 디폴트는 CLOSE_CURSORS_ON_COMMIT입니다.

ResultSet 특성 지정: ResultSet 오브젝트를 작성한 후에 ResultSet의 특성은 변경되지 않습니다. 따라서 오브젝트를 작성하기 전에 특성을 지정했습니다. createStatement, prepareStatement 및 prepareCall 메소드를 다양하게 변경하여 이러한 특성을 지정할 수 있습니다.

ResultSet 특성은 다음의 주제를 참조하십시오.

- Statement에 대한 138 페이지의『ResultSet 특성 지정』
- PreparedStatement에 대한 142 페이지의『ResultSet 특성 및 자동 생성 키 지원 지정』
- CallableStatement에 대한 149 페이지의『ResultSet 특성 및 자동 생성 키 지원 지정』

주: ResultSet 유형 및 ResultSet 동시성을 확보하기 위한 ResultSet 메소드가 있지만 ResultSet의 보유성을 확보하기 위한 메소드는 없습니다.

예: 민감한 ResultSet와 민감하지 않은 ResultSet:

다음의 예는 표에 행을 삽입할 때 민감한 ResultSet와 민감하지 않은 ResultSet를 보여줍니다.

예: 민감함 ResultSet와 민감하지 않은 ResultSet

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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();
        }
    }
}
```

예: ResultSet 민감성:

다음의 예는 ResultSet의 민감성에 기초하여 변경이 SQL문의 where 절에 미치는 영향을 보여줍니다.

예: ResultSet 민감성

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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();
        }
    }
}
```

커서 이동:

ResultSet.next 메소드는 ResultSet에서 한 번에 한 행씩 이동하는 데 사용됩니다. JDBC(Java(TM) Database Connectivity) 2.0로 iSeries JDBC 드라이버는 화면이동이 가능한 ResultSet를 지원합니다. 화면이동이 가능한 ResultSet를 사용하면 previous, absolute, relative, first 및 last 메소드를 사용하여 순서에 상관 없이 자료 행을 처리할 수 있습니다.

디폴트로 JDBC ResultSet는 항상 전달 전용이며 호출할 수 있는 유효한 커서 배치 메소드는 next()뿐임을 의미합니다. 화면이동이 가능한 ResultSet를 명시적으로 요구해야 합니다. 자세한 정보는 158 페이지의 『ResultSet 유형』을 참조하십시오.

화면이동이 가능한 ResultSet를 사용하면 다음의 커서 배치 메소드를 사용할 수 있습니다.

메소드	설명
Next	이 메소드는 ResultSet에서 커서를 한 행 앞으로 이동합니다. 커서가 유효한 행에 있으면 이 메소드는 true를 리턴하고 그렇지 않으면 false를 리턴합니다.
Previous	이 메소드는 ResultSet에서 커서를 한 행 뒤로 이동합니다. 커서가 유효한 행에 있으면 이 메소드는 true를 리턴하고 그렇지 않으면 false를 리턴합니다.

메소드	설명
First	이 메소드는 ResultSet의 첫 번째 행으로 커서를 이동합니다. 커서가 첫 번째 행에 있으면 이 메소드는 true를 리턴하고 ResultSet가 비어 있으면 false를 리턴합니다.
Last	이 메소드는 ResultSet의 마지막 행으로 커서를 이동합니다. 커서가 마지막 행에 있으면 이 메소드는 true를 리턴하고 ResultSet가 비어 있으면 false를 리턴합니다.
BeforeFirst	이 메소드는 ResultSet에서 첫 번째 행의 바로 앞으로 커서를 이동합니다. ResultSet가 비어 있으면 이 메소드는 아무 영향을 주지 않습니다. 이 메소드의 리턴 값은 없습니다.
AfterLast	이 메소드는 ResultSet에서 마지막 행의 바로 뒤로 커서를 이동합니다. ResultSet가 비어 있으면 이 메소드는 아무 영향을 주지 않습니다. 이 메소드의 리턴 값은 없습니다.
Relative(int rows)	이 메소드는 현재 위치에 따라 커서를 이동합니다. • 행이 0이면 이 메소드는 아무 영향을 주지 않습니다. • 행이 양수이면 커서는 이 수만큼의 행을 건너뛰어 이동합니다. 현재 위치와 ResultSet의 끝 사이에 입력 매개변수가 지정한 수보다 적은 행이 있으면 이 메소드는 afterLast와 비슷하게 작동합니다. • 행이 음수이면 커서는 이 수만큼의 행 이전으로 이동합니다. 현재 위치와 ResultSet의 끝 사이에 입력 매개변수가 지정한 수보다 적은 행이 있으면 이 메소드는 beforeFirst와 비슷하게 작동합니다. 커서가 유효한 행에 있으면 이 메소드는 true를 리턴하고 그렇지 않으면 false를 리턴합니다.
Absolute(int row)	이 메소드는 행 값이 지정한 행으로 커서를 이동합니다. 행 값이 양수이면 커서는 ResultSet의 처음부터 이 행 수만큼 건너뛴 곳에 위치합니다. 첫 번째 행은 1이고 두 번째 행은 2 등으로 번호를 매깁니다. ResultSet에 행 값이 지정한 수보다 적은 행이 있으면 이 메소드는 afterLast와 동일한 방식으로 작동합니다. 행 값이 음수이면 커서는 ResultSet의 끝에서부터 이 행 수보다 이전에 위치합니다. 마지막 행은 -1이고 마지막에서 두 번째 행은 -2 등으로 번호를 매깁니다. ResultSet에 행 값이 지정한 수보다 적은 행이 있으면 이 메소드는 beforeFirst와 동일한 방식으로 작동합니다. 행 값이 0이면 이 메소드는 beforeFirst와 동일한 방식으로 작동합니다. 커서가 유효한 행에 있으면 이 메소드는 true를 리턴하고 그렇지 않으면 false를 리턴합니다.

ResultSet 자료 검색:

ResultSet 오브젝트는 행에 대한 열 자료를 구하기 위한 여러가지 메소드를 제공합니다. 이들은 모두 get<*Type*> 형식입니다. 여기서 <*Type*>은 Java(TM) 자료 유형입니다. 이런 메소드의 일부 예로는 getInt, getLong, getString, getTimestamp 및 getBlob가 있습니다. 이러한 거의 모든 메소드는 ResultSet 내에 있는 열의 색인이거나 열 이름인 단일 매개변수를 사용합니다.

ResultSet 열은 1부터 시작하여 번호가 매겨집니다. 열 이름이 사용되며 ResultSet에 동일한 이름의 열이 둘 이상 있는 경우 첫 번째 것이 리턴됩니다. getTime, getDate 및 getTimestamp에 전달할 수 있는 선택적 Calendar 오브젝트와 같은 추가 매개변수가 있는 일부 get<Type> 메소드가 있습니다. 완전한 세부사항을 보려면 java.sql 패키지에 대한 Javadoc를 참조하십시오.

오브젝트를 리턴하는 get 메소드의 경우에 ResultSet의 열이 널(null)이면 리턴 값은 널입니다. 기본 유형의 경우 널을 리턴할 수 없습니다. 이와 같은 경우에 값은 0이거나 false입니다. 어플리케이션이 널과 0 또는 false를 구별해야 하는 경우 호출 이후에 즉시 wasNull 메소드를 사용할 수 있습니다. 그러면 이 메소드는 값이 실제 0 또는 false 값인지 아니면 ResultSet 값이 실제로 널이기 때문에 이 값이 리턴되었는지를 판별할 수 있습니다.

ResultSet 인터페이스를 사용하는 방법의 예는 예: IBM Developer Kit for Java의 ResultSet 인터페이스를 참조하십시오.

ResultSetMetaData 지원: getMetaData 메소드가 ResultSet 오브젝트에서 호출되면 이 ResultSet 오브젝트의 열을 설명하는 ResultSetMetaData 오브젝트를 리턴합니다. 프로세스되고 있는 SQL문이 런타임까지 알려지지 않는 경우 ResultSetMetaData를 사용하여 자료를 검색하는 데 사용해야 하는 메소드를 판별할 수 있습니다. 다음의 코드 예는 ResultSetMetaData를 사용하여 결과 세트의 각 열 유형을 판별합니다.

예: ResultSetMetaData를 사용한 결과 세트의 각 열 유형 판별

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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);
```

ResultSetMetaData 인터페이스를 사용하는 방법의 예는 예: IBM Developer Kit for Java의 ResultSetMetaData 인터페이스를 참조하십시오.

ResultSet 변경:

ResultSet의 디폴트 설정은 읽기 전용입니다. 그러나 JDBC(Java(TM) Database Connectivity) 2.0을 사용하여 iSeries JDBC 드라이버는 갱신 가능한 ResultSet에 대한 완전한 지원을 제공합니다. ResultSet의 갱신 방법에 대해서는 ResultSet 159 페이지의 『동시성』을 참조하십시오.

행 갱신: ResultSet 인터페이스를 통해 데이터베이스 표에서 행을 갱신할 수 있습니다. 이 프로세스에 포함된 단계는 다음과 같습니다.

1. 다양한 update<*Type*> 메소드(여기서 <*Type*>은 Java 자료 유형)를 사용하여 특정 행에 대한 값을 변경하십시오. 이러한 update<*Type*> 메소드는 값을 검색하는 데 사용할 수 있는 get<*Type*> 메소드에 해당합니다.
2. 기초 데이터베이스에 행 적용

데이터베이스 자체는 두 번째 단계까지 갱신하지 않습니다. updateRow 메소드를 호출하지 않고 ResultSet에서 열을 갱신하면 데이터베이스가 변경되지 않습니다.

행에 대한 계획된 갱신은 cancelUpdates 메소드를 사용하여 버릴 수 있습니다. updateRow 메소드를 호출했으면 데이터베이스에 대한 변경은 최종적이고 복원할 수 없습니다.

주: 데이터베이스가 갱신된 행을 지정하는 방법을 가지고 있지 않으면 rowUpdated 메소드는 항상 false를 리턴합니다. 이에 따라 updatesAreDetected 메소드가 false를 리턴합니다.

행 삭제: ResultSet 인터페이스를 통해 데이터베이스 표에서 행을 삭제할 수 있습니다. deleteRow 메소드가 제공되며 현재 행을 삭제합니다.

행 삽입: ResultSet 인터페이스를 통해 데이터베이스 표에 행을 삽입할 수 있습니다. 이 프로세스는 어플리케이션이 특별하게 커서를 이동시키고 데이터베이스에 삽입하려는 값을 빌드하는 ″행 삽입″을 활용합니다. 이 프로세스에 포함된 단계는 다음과 같습니다.

1. 삽입 행에 커서 위치.
2. 새로운 행에서 열에 대한 각 값 설정.
3. 데이터베이스에 행 삽입 및 선택적으로 커서를 ResultSet 내의 현재 행으로 다시 이동.

주: 커서가 놓인 표로 새로운 행이 삽입되지 않습니다. 일반적으로 표 자료 공간의 끝에 추가됩니다. 디폴트로 관계형 데이터베이스는 위치 의존적이지 않습니다. 예를 들어, 커서를 세 번째 행으로 이동시키고 후속 사용자가 자료를 페치하면 네 번째 행 앞에 나타나는 것을 삽입할 것으로 예상해서는 안됩니다.

찾은 갱신사항에 대한 지원: ResultSet를 통한 데이터베이스 갱신 메소드 이외에 SQL문을 사용하여 찾을 갱신사항을 실행할 수 있습니다. 이 지원은 명명된 커서 사용에 의존합니다. JDBC는 Statement에서 setCursorName 메소드를 제공하고 ResultSet에서 getCursorName 메소드를 제공하여 이러한 값에 대한 액세스 권한을 제공합니다.

원시 JDBC 드라이버와 함께 이 피처가 지원되면 두 개의 DatabaseMetaData 메소드인 supportsPositionedUpdated와 supportsPositionedDelete는 모두 true를 리턴합니다.

자세한 정보는 예: 다른 명령문의 커서를 통해 명령문을 사용하여 값 변경을 참조하십시오.

자세한 정보는 예: 다른 명령문의 커서를 통해 표에서 값 제거를 참조하십시오.

예: 다른 명령문의 커서를 통해 표에서 값 제거:

다음은 다른 명령문의 커서를 통해 표에서 값을 제거하는 방법의 예입니다.

예: 다른 명령문의 커서를 통해 표에서 값 제거

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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();
        }
    }
}
```

예: 다른 명령문의 커서를 통해 명령문을 사용하여 값 변경:

다음은 다른 명령문의 커서를 통해 명령문을 사용하여 값을 변경하는 방법의 예입니다.

예: 다른 명령문의 커서를 통해 명령문을 사용하여 값 변경

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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();
        }
    }
}
```

ResultSets 작성:

ResultSet 오브젝트를 작성하려면 Statement, PreparedStatement 또는 CallableStatement 인터페이스에서 executeQuery 메소드를 사용하십시오. 그러나 다른 메소드도 사용할 수 있습니다. 예를 들어 몇 개의 DatabaseMetaData 메소드는 getColumns, getTables, getUDTs, getPrimaryKeys 등과 같은 ResultSet를 리턴합니다. 단일 SQL문이 처리할 여러 ResultSet를 리턴할 수도 있습니다. 또한 Statement, PreparedStatement 또는 CallableStatement 인터페이스가 제공하는 execute 메소드를 호출한 후에 getResultSet 메소드를 사용하여 ResultSet 오브젝트를 검색할 수 있습니다.

자세한 정보는 예: 복수 ResultSet로 프로시듀어 작성을 참조하십시오.

ResultSet 닫기: 연관된 Statement 오브젝트가 닫히면 ResultSet 오브젝트가 자동으로 닫히지만 사용을 완료했을 때 ResultSet 오브젝트를 닫는 것이 바람직합니다. 이와 같이 하면 내부 데이터베이스 자원이 즉시 해제되어 어플리케이션 처리량이 증가합니다.

DatabaseMetaData 호출로 생성된 ResultSet를 닫는 것도 중요합니다. 이러한 ResultSet를 작성하기 위해 사용한 Statement 오브젝트에 직접 액세스할 수 없기 때문에 Statement 오브젝트에 대한 닫기를 직접 호출할 수 없습니다. 외부 ResultSet 오브젝트를 닫으면 JDBC 드라이버가 내부 Statement 오브젝트를 닫는 방식으로 이러한 오브젝트들은 서로 링크되어 있습니다. 이러한 오브젝트를 수동으로 닫지 않으면 시스템을 계속 작동하지만 필요한 것보다 많은 자원을 사용합니다.

주: ResultSet의 보유성 특성도 사용자를 대신하여 ResultSet를 자동으로 닫을 수 있습니다. ResultSet 오브젝트에서 닫기를 여러 번 호출할 수 있습니다.

예: IBM Developer Kit for Java용 ResultSet 인터페이스: 다음은 ResultSet 인터페이스를 사용하는 방법을 보여주는 예입니다.

예 1: ResultSet 인터페이스

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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");
                }
            }
        }
    }
}
```

예: IBM Developer Kit for Java용 ResultSetMetaData 인터페이스:

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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");
```

```
                }
            }
        }
    }
}
```

JDBC 오브젝트 풀

오브젝트 풀링은 JDBC(Java(TM) Database Connectivity) 및 성능을 논의할 때 가장 자주 언급되는 주제입니다. Connection, Statement 및 ResultSet 오브젝트와 같이 JDBC에서 사용하는 많은 오브젝트는 작성하는 비용이 많이 들기 때문에 이러한 오브젝트가 필요할 때마다 작성하지 않고 재사용하면 상당한 성능 이점이 있을 수 있습니다.

많은 어플리케이션은 이미 사용자를 대신하여 오브젝트 풀을 처리합니다. 예를 들어, WebSphere에는 JDBC 오브젝트의 풀을 위한 확장 지원이 있으며 풀을 관리하는 방법을 제어할 수 있습니다. 이로 인해 사용자 자신의 풀 메카니즘에 대해 염려하지 않고 원하는 기능을 얻을 수 있습니다. 그러나 이 지원이 제공되지 않으면 사소한 어플리케이션을 제외한 모든 어플리케이션에 대한 솔루션을 찾아야 합니다.

JDBC 프로그램에서 오브젝트 풀을 사용하려면 다음을 참조하십시오.

오브젝트 풀에 대한 DataSource 지원 사용
DataSource를 사용하면 여러 어플리케이션들이 데이터베이스에 액세스하기 위해 공통적인 구성을 공유할 수 있습니다. 이 작업은 각 어플리케이션이 동일한 DataSource 이름을 참조하도록 하면 가능합니다.

ConnectionPoolDataSource 등록 정보
ConnectionPoolDataSource 인터페이스가 제공하는 등록 정보 세트를 사용하여 이 인터페이스를 구성할 수 있습니다.

DataSource 기반 명령문 풀
연결 풀 내에서 명령문 풀을 사용할 수 있습니다. UDBConnectionPoolDataSource 인터페이스의 maxStatements 등록 정보를 사용하면 DataSource는 연결 하에 풀이 가능한 명령문 수를 지정할 수 있습니다.

사용자 자신의 풀 솔루션 빌드
DataSource에 대한 지원을 요구하거나 다른 제품에 의존하지 않고 사용자 자신의 연결 및 명령문 풀을 개발할 수 있습니다.

오브젝트 풀에 대한 DataSource 사용:

DataSource를 사용하면 여러 어플리케이션이 데이터베이스에 액세스하기 위해 공통 구성을 공유할 수 있습니다. 이 작업은 각 어플리케이션이 동일한 DataSource 이름을 참조하도록 하면 가능합니다.

DataSource를 사용하면 중앙 위치에서 많은 어플리케이션을 변경할 수 있습니다. 예를 들어, 모든 어플리케이션이 사용하는 디폴트 라이브러리의 이름을 변경하고 하나의 DataSource를 사용하여 모두에 대한 연결을 확보했으면 이 DataSource에서 콜렉션의 이름을 갱신할 수 있습니다. 그러면 모든 어플리케이션은 새로운 디폴트 라이브러리를 사용하기 시작합니다.

DataSource를 사용하여 어플리케이션에 대한 연결을 확보할 때 연결 풀에 대한 원시 JDBC 드라이버의 내장 지원을 사용할 수 있습니다. 이 지원은 ConnectionPoolDataSource 인터페이스의 구현으로 제공됩니다.

풀은 실제 Connection 오브젝트 대신 ″논리″ Connection 오브젝트를 전달하여 수행합니다. **논리 Connection 오브젝트**는 풀된 Connection 오브젝트가 리턴하는 연결 오브젝트입니다. 각 논리 연결 오브젝트는 풀된 연결 오브젝트가 나타내는 실제 연결에 대한 임시 핸들의 역할을 수행합니다. 어플리케이션에 Connection 오브젝트가 리턴되면 이들 둘 사이엔 특별한 차이점이 없습니다. Connection 오브젝트에서 close 메소드를 호출할 때 미묘한 차이점이 있습니다. 이 호출은 논리 연결을 무효화하고 다른 어플리케이션이 실제 연결을 사용할 수 있는 풀에 실제 연결을 리턴합니다. 이 기술을 통해 많은 논리 연결 오브젝트가 단일 실제 연결을 재사용할 수 있습니다.

연결 풀 설정: 연결 풀은 ConnectionPoolDataSource 오브젝트를 참조하는 DataSource 오브젝트를 작성하여 수행합니다. ConnectionPoolDataSource 등록 정보에는 풀 유지보수의 다양한 측면들을 처리하기 위해 설정할 수 있는 등록 정보가 있습니다.

자세한 정보는 UDBDataSource 및 UDBConnectionPoolDataSource를 사용하여 연결 풀 설정 방법에 대한 예를 참조하십시오. 이 예에서 JNDI의 역할에 대한 자세한 내용은 JNDI(Java 명명 및 디렉토리 인터페이스)를 참조하십시오.

예에서 두 DataSource 오브젝트를 함께 바인드하는 링크는 dataSourceName입니다. 이 링크는 풀을 자동으로 관리하는 ConnectionPoolDataSource 오브젝트에 대한 연결 설정을 미루도록 DataSource 오브젝트에 지시합니다.

어플리케이션 풀 및 풀 해제: Connection 풀을 사용하는 어플리케이션과 사용하지 않는 어플리케이션 사이에는 차이점이 없습니다. 따라서 어플리케이션 코드가 완료된 후에 어플리케이션 코드를 변경하지 않고 풀 지원을 추가할 수 있습니다.

자세한 내용은 예: 연결 풀 성능 테스트를 참조하십시오.

다음은 개발 중 앞의 프로그램을 로컬로 실행한 결과입니다.

```
Start timing the non-pooling DataSource version...
Time spent: 6410

Start timing the pooling version...
Time spent: 282
```

Java program completed.

디폴트로 UDBConnectionPoolDataSource는 단일 연결을 풀합니다. 어플리케이션이 한 연결을 여러 번 요구하고 한 번에 한 연결만을 요구하는 경우에 UDBConnectionPoolDataSource가 완벽한 솔루션입니다. 많은 동시 연결이 필요하면 사용자의 요구 및 자원과 일치하도록 ConnectionPoolDataSource를 구성해야 합니다.

«

예: UDBDataSource 및 UDBConnectionPoolDataSource를 사용하여 연결 풀 설정:

»

다음은 UDBDataSource 및 UDBConnectionPoolDataSource를 사용하여 연결 풀을 사용하는 방법을 보여주는 예입니다.

예: UDBDataSource 및 UDBConnectionPoolDataSource를 사용하여 연결 풀 설정

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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);
  }
}
```

«

예: 연결 풀 성능 테스트:

»

다음은 풀을 사용하지 않는 예의 성능과 비교하여 풀을 사용하는 예의 성능을 테스트하는 방법을 보여주는 예입니다.

예: 연결 풀 성능 테스트

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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 등록 정보:

ConnectionPoolDataSource 인터페이스는 구성을 위해 등록 정보 세트를 제공합니다. 이러한 등록 정보에 대한 설명은 다음의 표에 제공되어 있습니다.

등록 정보	설명
initialPoolSize	풀을 처음으로 인스턴스화할 때 이 등록 정보는 풀에 넣는 연결 수를 판별합니다. minPoolSize와 maxPoolSize 범위를 벗어나 이 값을 지정하면 minPoolSize 또는 maxPoolSize를 작성할 초기 연결 수로 사용합니다.
maxPoolSize	풀을 사용할 때 풀이 가지고 있는 것보다 많은 연결을 요구할 수 있습니다. 이 등록 정보는 풀에서 작성할 수 있는 최대 연결 수를 지정합니다. 어플리케이션은 풀이 최대 *크기*이며 모든 연결을 사용 중인 경우에 연결을 풀에 리턴하는 것을 "방해"하지 않고 기다립니다. 그 대신 JDBC 드라이버는 DataSource 등록 정보에 기초하여 새로운 연결을 구성하고 연결을 리턴합니다. maxPoolSize를 0으로 지정하면 시스템에 분배할 수 있는 자원이 있는 한 풀이 제한 없이 증가할 수 있습니다.
minPoolSize	풀 사용의 장애로 인해 포함된 연결 수가 증가할 수 있습니다. 활동 레벨이 일부 Connection이 풀에서 제거되지 않는 지점까지 감소되면 특별한 이유 없이 자원이 증가합니다. 이러한 경우에 JDBC 드라이버는 누적시킨 연결의 일부를 해제하는 능력을 가지고 있습니다. 이 등록 정보를 사용하면 JDBC에 연결을 해제하도록 지시하여 일정한 후의 연결을 사용할 수 있도록 항상 유지할 수 있습니다. minPoolSize를 0으로 지정하면 풀이 모든 연결을 해제하고 어플리케이션이 각 연결 요구에 대한 연결 시간을 실제로 되돌릴 수 있습니다.
maxIdleTime	연결은 사용하지 않고 쉬고 있는 기간을 기록합니다. 이 등록 정보는 어플리케이션이 연결이 해제되기 전에 사용하지 않을 수 있는 기간을 지정합니다(즉, 필요한 양보다 많은 연결이 있음). 이 등록 정보는 초 단위 시간이며 실제 닫기가 발생한 시기를 지정하지 않습니다. 연결을 해제해야 하는 충분한 시간이 경과했을 때 지정됩니다.
propertyCycle	이 등록 정보는 이러한 규칙들의 시행 사이에 경과할 수 있는 시간(초)을 나타냅니다.

주: maxIdleTime 또는 propertyCycle 시간을 0으로 설정하는 것은 JDBC 드라이버가 스스로 풀에서 제거할 연결을 검사하지 않음을 의미합니다. initial, min 및 max *크기*에 대해 지정된 규칙이 계속 시행됩니다.

maxIdleTime 및 propertyCycle이 0이 아니면 관리 스레드를 사용하여 풀을 계속해서 주시해야 합니다. 스레드는 propertyCycle 초마다 활동하고 풀의 모든 연결을 검사하여 maxIdleTime 초 이상 사용하지 않고 있는 연결을 확인합니다. minPoolSize에 도달할 때까지 이 기준에 적합한 연결을 풀에서 제거합니다.

DataSource 기반 명령문 풀:

UDBConnectionPoolDataSource 인터페이스에서 사용할 수 있는 다른 등록 정보는 maxStatements입니다. 이 등록 정보는 연결 풀 내에서 명령문 풀을 허용합니다. 명령문 풀은 PreparedStatement와 CallableStatement에만 영향을 줍니다. Statement 오브젝트는 풀하지 않습니다.

명령문 풀의 구현은 연결 풀의 구현과 유사합니다. 어플리케이션이 Connection.prepareStatement(″select * from tablex″)를 호출하면 풀 모듈은 Statement 오브젝트가 연결 하에 이미 준비되었는지 검사합니다. 이미 준비되었으면 실제 오브젝트 대신 논리 PreparedStatement 오브젝트가 사용자에게 전달됩니다. 닫기를 호출하면 Connection 오브젝트가 풀로 리턴되고 논리 Connection 오브젝트를 버리며 Statement 오브젝트를 재사용할 수 있습니다.

maxStatements 등록 정보를 사용하면 DataSource는 연결 하에 풀이 가능한 명령문 수를 지정할 수 있습니다. 값 0은 명령문 풀을 사용하지 말 것을 나타냅니다. 명령문 풀이 가득 차면 최소한 최근에 사용한 알고리즘을 적용하여 버릴 명령문을 판별합니다.

예: 두 개의 DataSource 성능 테스트는 연결 풀만을 사용하는 하나의 DataSource와 명령문 및 연결 풀을 사용하는 다른 DataSource를 테스트합니다.

다음의 예는 개발 중 이 프로그램을 로컬로 실행한 결과입니다.

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

예: 두 DataSource의 성능 테스트:

다음은 연결 풀만을 사용하는 하나의 DataSource와 명령문 및 연결 풀을 사용하는 다른 DataSource를 테스트하는 예입니다.

예: 두 DataSource의 성능 테스트

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
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);

    }
}
```

사용자 자신의 연결 풀 빌드:

DataSource에 대한 지원을 요구하거나 다른 제품에 의존하지 않고 사용자 자신의 연결 및 명령문 풀을 개발할 수 있습니다.

풀 기술은 작은 Java 어플리케이션에서 시연할 수 있지만 서브릿 또는 큰 n층 어플리케이션에 동등하게 적용할 수 있습니다. 이 예를 사용하여 성능 문제를 설명합니다.

시연 어플리케이션에는 두 가지 기능이 있습니다.

- 데이터베이스 표에 새로운 색인과 이름을 삽입하는 기능.
- 표에서 제공된 색인에 대한 이름을 읽는 기능.

어플리케이션의 완벽한 코드는 IBM의 Developer Kit for Java JDBC  웹 페이지에서 다운로드 할 수 있습니다.

어플리케이션 예가 올바로 수행되지 않습니다. 이 코드를 통해 getValue 메소드에 대해 100회의 호출을 실행하고 putValue 메소드에 대해 100회의 호출을 실행하면 표준 워크스테이션에서 평균 31.86초가 걸렸습니다.

문제는 모든 요구에 대해 너무 많은 데이터베이스 작업이 있다는 점입니다. 즉, 연결을 확보하고 명령문을 확보하고 명령문을 처리하고 명령문을 닫고 연결을 닫습니다. 각 요구 이후에 모두 버리는 대신 이 프로세스의 부분들을 다시 사용하는 방법이 있습니다. **연결** 풀은 연결 작성 코드를 풀에서 연결을 확보하기 위한 코드로 대체한 다음 연결 닫기 코드를 사용할 풀에 연결을 리턴하기 위한 코드로 대체합니다.

연결 풀의 구성자는 연결을 작성하고 풀에 넣습니다. 풀 클래스에는 사용할 연결을 찾고 연결에 대한 작업을 완료했을 때 풀에 연결을 리턴하기 위한 take 및 put 메소드가 있습니다. 풀 오브젝트는 공유 자원이기 때문에 이러한 메소드들을 동기화하지만 대개는 여러 스레드가 풀 자원을 동시에 조작하는 것을 원하지 않습니다.

getValue 메소드에 대한 호출 코드가 변경되었습니다. putValue 메소드는 표시되지 않지만 이 메소드는 정확히 변경되었으며 IBM의 Developer Kit for Java JDBC  웹 페이지에서 사용할 수 있습니다. 연결 풀 오브젝트의 설치도 표시되지 않습니다. 구성자를 호출하고 풀에서 원하는 연결 오브젝트의 수를 전달할 수 있습니다. 이 단계는 어플리케이션을 시작할 때 수행해야 합니다.

이러한 변경을 통한 이전 어플리케이션 실행(즉, 100개의 getValue 메소드와 100개의 putValue 메소드 요구 사용)은 연결 풀 코드를 제 위치에 넣고 평균 13.43초가 걸렸습니다. 작업부하의 처리 시간은 연결 풀 없이 원래 처리 시간보다 절반 이상이나 줄었습니다.

사용자 자신의 명령문 풀 빌드: 연결 풀을 사용할 때 각 명령문 처리 시에 명령문을 작성하고 닫는데 시간을 낭비했습니다. 다음은 재사용할 수 있는 오브젝트를 낭비하는 또 다른 예입니다.

오브젝트를 재사용하려면 준비된 명령문 클래스를 사용하면 됩니다. 대부분의 어플리케이션에서 동일한 SQL문을 약간 변경하여 재사용합니다. 예를 들어, 어플리케이션을 통한 한 번의 반복은 다음의 조회를 생성할 수 있습니다.

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

그 뒤의 반복은 다음의 조회를 생성할 수 있습니다.

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

이것은 동일한 조회이지만 다른 매개변수를 사용합니다. 두 조회 모두 다음의 조회를 통해 수행할 수 있습니다.

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

그런 다음 첫 번째 조회를 처리할 때 매개변수 마커(의문 부호로 표시)를 100000으로 설정하고 두 번째 조회를 처리할 때 50000으로 설정할 수 있습니다. 그러면 세 가지 이유로 연결 풀이 제공할 수 있는 것 이상으로 성능이 향상됩니다.

- 보다 적은 오브젝트가 작성됩니다. 모든 요구에 대해 Statement 오브젝트를 작성하는 대신 PreparedStatement 오브젝트를 작성하고 재사용합니다. 따라서 보다 적은 구성자를 실행합니다.
- SQL문을 설정하기 위한 데이터베이스 작업(**준비**)을 재사용할 수 있습니다. SQL문 준비는 SQL문 텍스트의 내용과 시스템이 요구된 타스크를 수행하는 방법을 결정하는 작업이 필요하므로 상당히 비용이 많이 듭니다.
- 추가 오브젝트 작성을 제거할 때 종종 간과하는 이점이 있습니다. 작성되지 않은 것을 폐기할 필요가 없습니다. 이 모델은 Java 가비지 콜렉터에서 보다 쉬우며 많은 사용자에게 오랜 시간에 걸쳐 성능 이점을 가져다 줍니다.

Connection 대신 PreparedStatement 오브젝트를 풀하도록 시연 프로그램을 변경할 수 있습니다. 프로그램을 변경하면 보다 많은 오브젝트를 재사용하고 성능을 개선할 수 있습니다. 풀할 오브젝트가 포함된 클래스를 기록하여 시작할 수 있습니다. 이 클래스는 사용할 다양한 자원을 캡슐화해야 합니다. 연결 풀을 예로 들면 Connection이 유일하게 풀된 자원이므로 캡슐화 클래스가 필요하지 않았습니다. 풀된 각 오브젝트에는 하나의 Connection과 두 개의 PreparedStatement가 있어야 합니다. 그런 다음 연결 대신 데이터베이스 액세스 오브젝트가 있는 풀 클래스를 작성할 수 있습니다.

마지막으로 데이터베이스 액세스 오브젝트를 확보하고 사용하려는 오브젝트의 자원을 지정하도록 어플리케이션을 변경해야 합니다. 특정 자원을 지정하는 것을 제외하고는 어플리케이션은 동일합니다.

이러한 변경을 통해 현재 실행하는 동일한 테스트는 평균 0.83초가 걸립니다. 이 시간은 원래의 프로그램 버전보다 약 38초가 빠릅니다.

고려사항: 성능은 복제를 통해 향상됩니다. 항목을 재사용하지 않으면 풀하는 데 자원을 낭비합니다.

대부분의 어플리케이션에는 중요한 코드 섹션이 있습니다. 일반적으로 어플리케이션은 코드의 10 - 20%에 대해서만 처리 시간의 80 - 90%를 사용합니다. 어플리케이션에서 10,000개의 SQL문을 잠재적으로 사용한 경우에 모두 풀되는 것은 아닙니다. 어플리케이션의 중요한 코드 섹션에서 사용한 SQL문을 식별하고 풀하는 것이 목적입니다.

Java 구현에서 오브젝트를 작성하려면 상당한 비용이 들 수 있습니다. 풀 솔루션을 유리하게 사용할 수 있습니다. 프로세스에서 사용하는 오브젝트는 다른 사용자가 시스템을 사용하려고 시도하기 전에 처음부터 작성됩니다. 이러한 오브젝트는 필요할 때마다 자주 재사용합니다. 성능은 뛰어나고 계속해서 어플리케이션을 세부 조정하여 보다 많은 수의 사용자가 사용할 수 있습니다. 결과적으로 보다 많은 오브젝트가 풀됩니다. 또한 보다 큰 처리량을 확보하기 위해 어플리케이션의 데이터베이스 액세스에 대한 보다 효율적인 멀티스레드를 허용합니다.

Java(JDBC 사용)는 동적 SQL에 기초하며 느려지는 경향이 있습니다. 풀은 이 문제를 최소화할 수 있습니다. 시작 시에 명령문을 준비하면 데이터베이스에 대한 액세스가 정적이 될 수 있습니다. 명령문을 준비한 후에 동적 SQL과 정적 SQL 사이에 성능 차이는 거의 없습니다.

Java에서 데이터베이스 액세스의 성능은 효율적일 수 있으며 오브젝트 지향 설계 또는 코드 유지보수성을 희생하지 않고 수행할 수 있습니다. 명령문 및 연결 풀을 빌드하기 위한 코드를 기록하는 것은 어렵지 않습니다. 또한 코드를 변경하고 향상시켜서 여러 어플리케이션 및 어플리케이션 유형(웹 기반, 클라이언트/서버) 등을 지원할 수 있습니다.

일괄처리 갱신

JDBC 2.0의 새로운 피처는 일괄처리 갱신 지원입니다. 이 피처를 사용하면 데이터베이스에 대한 갱신사항을 사용자 프로그램과 데이터베이스 사이에 하나의 트랜잭션으로 전달할 수 있습니다. 이 프로시듀어는 한 번에 많은 갱신을 수행해야할 때 성능을 상당히 향상시킬 수 있습니다. 예를 들어, 대기업이 새로 고용한 직원들이 월요일에 업무를 시작할 것을 요구하면 이 요구사항에 따라 한 번에 직원 데이터베이스에 대한 여러 갱신사항(이 경우에는 삽입)을 처리해야 합니다. 갱신의 일괄처리를 작성하고 한 단위로 데이터베이스에 제출하면 처리 시간을 절약할 수 있습니다.

다음은 일괄처리 갱신의 두 가지 유형입니다.

- 일괄처리 갱신은 Statement 오브젝트를 사용합니다.
- 일괄처리 갱신은 PreparedStatement 오브젝트를 사용합니다.

일괄처리 갱신 지원을 사용하려면 다음을 참조하십시오.

Statement 일괄처리 갱신

Statement 일괄처리 갱신을 수행하기 전에 자동 확약을 Off로 설정했는지 확인해야 합니다. 자동 확약을 Off로 설정하면 표준 Statement 오브젝트를 작성할 수 있습니다. 그런 다음 addBatch 메소드를 사용하여 일괄처리에 명령문을 추가할 수 있습니다. 일괄처리에 원하는 모든 명령문을 추가했으면 executeBatch 메소드를 사용하여 모두 처리하거나 언제든지 clearBatch 메소드를 사용하여 일괄처리를 비울 수 있습니다.

PreparedStatement 일괄처리 갱신

preparedStatement 일괄처리는 Statement 일괄처리와 비슷합니다. 그러나 preparedStatement 일괄처리는 항상 동일한 준비된 명령문을 제거하고 매개변수를 이 명령문으로 변경하기만 합니다.

BatchUpdateException

executeBatch 메소드에 대한 호출이 실패하면 BatchUpdateException이 발생합니다. BatchUpdateException을 사용하면 메세지, SQLState 및 벤더 코드를 받기 위해 항상 호출했던 메소드와 동일한 메소드를 모두 호출할 수 있습니다. BatchUpdateException은 정수 배열을 리턴하는 getUpdateCounts 메소드도 제공합니다. 정수 배열에는 일괄처리에서 처리된 모든 명령문에서부터 실패가 발생한 시점까지의 갱신 계수가 들어 있습니다.

블록화 삽입 지원

데이터베이스 표에 한 번에 여러 행을 삽입하는 iSeries 조작인 블록화 삽입을 사용할 수 있습니다.

Statement 일괄처리 갱신:

Statement 일괄처리 갱신을 수행하려면 자동 확약을 Off로 설정해야 합니다. Java(TM) Database Connectivity(JDBC)에서, 자동 확약은 디폴트로 On입니다. 자동 확약은 데이터베이스에 대한 갱신을 각 SQL 문이 처리된 후에 확약함을 의미합니다. 데이터베이스에 전달되는 명령문 그룹을 하나의 기능적 그룹으로 취급하려면 데이터베이스가 각 명령문을 개별적으로 확약하지 않도록 해야 합니다. 자동 확약을 Off로 설정하지 않은 상태에서 일괄처리 중간에 명령문이 실패하면 전체 일괄처리를 롤백할 수 없으며 명령문 중 절반이 종료되었기 때문에 다시 시도하십시오. 또한 일괄처리에서 각 명령문을 확약하는 추가 작업은 많은 오버헤드를 작성합니다. 자세한 내용은 트랜잭션을 참조하십시오.

자동 확약을 끈 후에 표준 Statement 오브젝트를 작성할 수 있습니다. executeUpdate와 같은 메소드를 사용하여 명령문을 처리하는 대신 addBatch 메소드를 사용하여 일괄처리에 명령문을 추가하십시오. 일괄처리에 원하는 모든 명령문을 추가했으면 executeBatch 메소드를 사용하여 모두 처리할 수 있습니다. 언제든지 clearBatch 메소드를 사용하여 일괄처리를 비울 수 있습니다.

다음 예는 이런 메소드를 사용할 수 있는 방법을 보여줍니다.

예: Statement 일괄처리 갱신

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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();
```

이 예에서 정수 배열이 executeBatch 메소드로부터 리턴됩니다. 이 배열에는 일괄처리에서 처리되는 각 명령문에 대해 하나의 정수 값이 있습니다. 값이 데이터베이스에 삽입되는 경우에 각 명령문에 대한 값은 1입니다 (즉, 처리가 성공함을 가정). 그러나 일부 명령문은 여러 행에 영향을 주는 update 명령문일 수 있습니다. INSERT, UPDATE 또는 DELETE 이외에 다른 명령문을 일괄처리에 삽입하면 예외 상태가 발생합니다.

PreparedStatement 일괄처리 갱신:

preparedStatement 일괄처리는 Statement 일괄처리와 비슷하지만 preparedStatement 일괄처리는 항상 동일한 "준비된" 명령문을 제거하며 매개변수를 이 명령문으로 변경하기만 합니다. 다음은 preparedStatement 일괄처리를 사용하는 예입니다.

예: PreparedStatement 일괄처리 갱신

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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:

일괄처리 갱신 시에 중요한 고려사항은 executeBatch 메소드에 대한 호출이 실패하면 취할 조치입니다. 이 경우에 BatchUpdateException이라고 하는 새로운 예외 상태가 발생합니다. BatchUpdateException은 SQLException의 서브클래스이며 메세지, SQLState 및 벤더 코드를 받기 위해 항상 호출했던 메소드와 동일한 메소드를 모두 호출할 수 있도록 합니다. BatchUpdateException은 정수 배열을 리턴하는 getUpdateCounts 메소드도 제공합니다. 정수 배열에는 일괄처리에서 처리된 모든 명령문에서부터 실패가 발생한 시점까지의 갱신 계수가 들어 있습니다. 배열 길이는 일괄처리에서 실패한 명령문을 알려줍니다. 예를 들어, 예외 상태로 리턴된 배열의 길이가 3이면 일괄처리에서 네 번째 명령문이 실패한 것입니다. 따라서 리턴된 하나의 BatchUpdateException 오브젝트에서 성공된 모든 명령문에 대한 갱신 계수, 실패한 명령문 및 실패에 대한 모든 정보를 판별할 수 있습니다.

현재 일괄처리 갱신을 처리하는 표준 수행은 각 명령문을 독립적으로 처리하는 성능과 동등합니다. 일괄처리 갱신의 최적화된 지원에 대한 자세한 정보를 보려면 블록화 삽입 지원을 참조하십시오. 코드화 및 향후 성능 최적화의 이점을 활용할 때 새로운 모델을 계속 사용해야 합니다.

주: JDBC 2.1 스펙에서는 일괄처리 갱신의 예외 상태 처리 방식에 대한 다른 옵션이 제공됩니다. JDBC 2.1은 일괄처리 항목이 실패한 후에 처리 일괄처리가 계속되는 모델을 도입했습니다. 실패한 각 항목에 대해 리턴된 갱신 계수 정수의 배열에 특수 갱신 계수가 있습니다. 이 계수는 항목 중 하나가 실패해도 큰 일괄처리가 처리를 계속할 수 있게 합니다. 이러한 두 조작 모드에 대한 자세한 내용은 JDBC 2.1 또는 JDBC 3.0 스펙을 참조하십시오. 디폴트로 원시 JDBC 드라이버는 JDBC 2.0 정의를 사용합니다. 드라이버는 DriverManager를 사용하여 연결을 설정할 때 사용하는 Connection 등록 정보를 제공합니다. 드라이버는 DataSource를 사용하여 연결을 설정할 때 사용하는 DataSource 등록 정보도 제공합니다. 이러한 등록 정보를 사용하면 어플리케이션은 일괄처리 조작의 실패 처리 방법을 선택할 수 있습니다.

블록화 삽입 지원:

블록화 삽입은 데이터베이스 표에 한 번에 여러 행을 삽입할 수 있도록 매우 최적화된 방법을 제공하는 iSeries 서버에 대한 특별한 유형의 조작입니다. 블록화 삽입은 일괄처리 갱신의 서브세트로 간주할 수 있습니다. 일괄처리 갱신은 갱신 요구의 한 형태로 볼 수 있지만 블록화 삽입은 고유합니다. 그러나 일괄처리 갱신의 블록화 삽입 유형은 공통적입니다. 이 기능의 이점을 활용하기 위해 원시 JDBC 드라이버가 변경되었습니다.

블록화 삽입 지원 사용 시 시스템 제한사항으로 인해 원시 JDBC 드라이버에 대한 디폴트 설정은 블록화 삽입을 작동 불가능하게 합니다. Connection 등록 정보 또는 DataSource 등록 정보를 통해 작동할 수 있습니다. 블록화 삽입 사용 시 대부분의 제한사항은 사용자 대신 확인하고 처리할 수 있지만 몇 가지 제한사항은 이러한 처리가 불가능합니다. 따라서 디폴트로 블록화 삽입 지원이 작동 중지되는 이유가 됩니다. 제한사항의 리스트는 다음과 같습니다:

- 사용된 SQL문은 VALUES 섹션이 있는 INSERT문이어야 하는데, 이는 SUBSELECT가 있는 INSERT 문이 아님을 의미합니다. JDBC 드라이버는 이 제한사항을 인식하며 적합한 조치를 취합니다.

- PreparedStatement를 사용해야 하는데, 이는 Statement 오브젝트에 대한 최적화된 지원이 없음을 의미합니다. JDBC 드라이버는 이 제한사항을 인식하며 적합한 조치를 취합니다.
- SQL문은 표의 모든 열에 대한 매개변수 마커를 지정해야 합니다. 이는 열에 대한 상수값을 사용하거나 데이터베이스가 열에 대한 디폴트 값을 삽입하도록 할 수 없음을 의미합니다. JDBC 드라이버에는 SQL문에서 특정 매개변수 마커에 대한 테스트를 처리하는 메카니즘이 없습니다. 최적화된 블록화 삽입을 수행하도록 등록 정보를 설정하고 SQL문에서 디폴트나 상수의 사용을 방지하지 않으면 데이터베이스 표에서 끝나는 값이 올바르지 않습니다.
- 로컬 시스템에 연결해야 합니다. 이는 DRDA가 블록화 삽입 조작을 지원하지 않기 때문에 리모트 시스템에 액세스하기 위해 DRDA를 사용하여 연결할 수 없음을 의미합니다. JDBC 드라이버에는 로컬 시스템에 대한 연결을 테스트하는 메카니즘이 없습니다. 최적화된 블록화 삽입을 수행하도록 등록 정보를 설정하고 리모트 시스템에 대한 연결을 시도하면 일괄처리 갱신의 처리가 실패합니다.

다음의 코드 예는 블록화 삽입 처리에 대한 지원을 가능하게 하는 방법을 나타냅니다. 이 코드와 블록화 삽입 지원을 사용하지 않는 버전은 연결 URL에 `use block insert=true`가 추가된다는 점만이 다릅니다.

예: 블록화 삽입 처리

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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...
```

비슷한 테스트의 경우에 블록화 삽입은 블록화 삽입을 사용하지 않을 때 동일한 조작을 수행하는 것보다 몇 배가 빠릅니다. 예를 들어, 이전 코드에서 수행된 테스트는 블록화 삽입 사용 시보다 9배가 빨랐습니다. 오브젝트 대신 기본 유형만을 사용하는 경우는 최대 16배가 빠를 수 있습니다. 상당한 양의 작업이 진행되고 있는 어플리케이션에서는 예상값을 적절하게 변경해야 합니다.

확장 자료 유형

V4R4 e-PACK과 함께 iSeries 데이터베이스에 제공되는 SQL3 자료 유형이라고 하는 몇 가지 새로운 자료 유형이 있습니다. Java(TM) Database Connectivity(JDBC) 2.0 이상에서는 SQL99 표준의 한 파트인 이들 자료 유형에 대해 작업할 수 있도록 지원합니다.

SQL3 자료 유형은 엄청난 유연성을 제공합니다. 일련화된 Java 오브젝트, XML(Extensible Markup Language) 문서 및 노래, 제품 그림, 직원 사진 및 영화 클립과 같은 멀티미디어 자료 저장에 이상적입니다.

고유 유형: 고유 유형은 표준 데이터베이스 유형에 기초하는 사용자 정의 유형입니다. 예를 들어, CHAR(9)인 사회 보장 번호(SSN) 유형을 내부적으로 정의할 수 있습니다. 다음의 SQL문이 이러한 DISTINCT 유형을 작성합니다.

```
CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)
```

고유 유형은 항상 내장 자료 유형에 맵핑합니다. SQL의 문맥에서 고유 유형을 사용하는 방법과 시기에 대한 자세한 정보는 SQL 참조서를 참조하십시오.

JDBC에서 고유 유형을 사용하려면 기본 유형에 액세스할 때와 동일한 방식으로 액세스하십시오. getUDTs 메소드는 시스템에서 사용할 수 있는 고유 유형을 조회할 수 있도록 하는 새로운 메소드입니다. 이 예 프로그램은 다음을 보여줍니다.

- 고유 유형의 작성.
- 이 유형을 사용하는 표 작성.
- PreparedStatement를 사용한 고유 유형 매개변수 설정.
- ResultSet를 사용한 고유 유형 리턴.
- getUDTs에 대한 메타데이터 어플리케이션 프로그래밍 인터페이스(API) 호출을 사용하여 고유 유형에 대해 배우기.

큰 오브젝트: 세 가지 유형의 큰 오브젝트(LOB)가 있습니다.

- 큰 2진 오브젝트(BLOB)
- 큰 문자 오브젝트(CLOB)
- 2바이트 큰 문자 오브젝트(DBCLOB)

DBCLOB는 문자 자료의 내부 기억장치 표시를 제외하고는 CLOB와 유사합니다. Java 및 JDBC는 유니코드와 같은 모든 문자 자료를 객관화하기 때문에 CLOB에 대한 JDBC에서만 지원됩니다. DBCLOB은 JDBC 측면에서 CLOB 지원과 상호 교환적으로 작업합니다.

큰 2진 오브젝트: 여러 가지 면에서 큰 2진 오브젝트(BLOB) 열은 크게 작성할 수 있는 CHAR FOR BIT DATA 열과 유사합니다. 변환되지 않는 바이트의 스트림으로 표현할 수 있는 모든 것을 이 열에 저장할 수 있습니다. 종종 BLOB 열은 일련화된 Java 오브젝트, 그림, 노래 및 기타 2진 자료를 저장하는데 사용합니다.

다른 표준 데이터베이스 유형을 사용하는 방법과 동일한 방법으로 BLOB을 사용할 수 있습니다. 저장 프로시듀어로 전달하고 준비된 명령문에 사용하고 결과 세트에서 갱신할 수 있습니다. PreparedStatement 클래스에는 BLOB을 데이터베이스에 전달하기 위한 setBlob 메소드가 있으며 ResultSet 클래스는 데이터베이스에서 검색할 수 있도록 getBlob 클래스를 추가합니다. BLOB은 JDBC 인터페이스인 BLOB 오브젝트로 Java 프로그램에서 표시합니다.

BLOB을 사용하는 방법에 대한 자세한 정보는 BLOB을 사용하는 코드 작성을 참조하십시오.

큰 문자 오브젝트: 큰 문자 오브젝트(CLOB)는 BLOB을 보완하는 문자 자료입니다. 변환하지 않고 데이터베이스에 자료를 저장하는 대신 자료를 텍스트로 데이터베이스에 저장하고 CHAR 열과 동일한 방법으로 처리합니다. BLOB와 마찬가지로 JDBC 2.0은 CLOB을 직접 다루기 위한 기능을 제공합니다. PreparedStatement 인터페이스에는 setClob 메소드가 들어 있으며 ResultSet 인터페이스에는 getClob 메소드가 들어 있습니다.

CLOB을 사용하는 방법에 대한 자세한 정보는 CLOB을 사용하는 코드 작성을 참조하십시오.

BLOB 및 CLOB 열은 CHAR FOR BIT DATA 및 CHAR 열과 같은 방식으로 작업하지만 외부 사용자의 관점에서 작업하는 방식을 개념적으로 나타낸 것입니다. 내부적으로는 다릅니다. 큰 오브젝트(LOB) 열의 엄청난 잠재적 크기 때문에 일반적으로 자료에 대해 간접적으로 작업합니다. 예를 들어, 행의 블록이 데이터베이스에서 페치되면 LOB 블록을 ResultSet로 이동시키지 않습니다. 대신 LOB 로케이터(즉, 4바이트 정수)라고 하는 포인터를 ResultSet로 이동시킵니다. 그러나 JDBC에서 LOB에 대해 작업할 때 로케이터에 대해 알 필요는 없습니다.

Datalink: **Datalink**는 데이터베이스에서 데이터베이스 외부에 저장된 파일로의 논리 참조가 들어 있는 캡슐화된 값입니다. Datalink는 JDBC 2.0 이하와 JDBC 3.0 이상 중 어느 쪽을 사용하느냐에 따라 JDBC 관점에서 두 가지 다른 방식으로 표현되고 사용됩니다.

Datalink를 사용하는 방법에 대한 자세한 정보는 Datalink를 사용하는 코드 작성을 참조하십시오.

지원되지 않는 SQL3 자료 유형: 정의된 다른 SQL3 자료 유형이 있으며 이 자료 유형에 대해 JDBC API는 지원을 제공합니다. 이러한 자료 유형으로는 ARRAY, REF 및 STRUCT가 있습니다. 현재 iSeries 서버는 이러한 유형을 지원하지 않습니다. 따라서 JDBC 드라이버는 이에 대해 어떠한 형태의 지원도 제공하지 않습니다.

BLOB을 사용하여 코드 기록:

JDBC(Java(TM) Database Connectivity) API를 통해 데이터베이스 큰 2진 오브젝트(BLOB) 열에 대해 수행할 수 있는 타스크가 매우 많습니다. 다음의 주제는 이러한 타스크를 간략하게 설명하며 타스크 수행 방법의 예를 제공합니다.

데이터베이스에서 BLOB을 읽은 후 데이터베이스에 BLOB 삽입: JDBC API를 사용하면 여러 가지 방법으로 데이터베이스에서 BLOB을 제거하고 데이터베이스에 BLOB을 삽입할 수 있습니다. 그러나 BLOB 오브젝트를 작성하는 표준화된 방법은 없습니다. 데이터베이스가 이미 BLOB으로 가득 찼으면 이 점은 문제가 되지 않지만 JDBC를 통해 스크래치에서 BLOB에 대해 작업하려는 경우에 문제가 될 수 있습니다. JDBC API의 BLOB 및 CLOB 인터페이스에 대한 구성자를 정의하는 대신 데이터베이스에 BLOB을 삽입하고 다른 유형으로 데이터베이스에서 직접 제거하는 지원을 제공합니다. 예를 들어, setBinaryStream 메소드는 유형이 BLOB인 데이터베이스 열에 대해 작업할 수 있습니다. 이 예는 BLOB을 데이터베이스에 삽입하거나 데이터베이스에서 검색하는 공통적인 방법의 일부를 보여줍니다.

Blob 오브젝트 API를 사용하여 작업: BLOB는 다양한 드라이버가 구현을 제공하는 인터페이스로서 JDBC에 정의됩니다. 이 인터페이스에는 BLOB 오브젝트와 대화하는 데 사용할 수 있는 일련의 메소드가 있습니다. 이 예는 이 API를 사용하여 수행할 수 있는 공통적인 타스크의 일부를 보여줍니다. BLOB 오브젝트에서 사용할 수 있는 메소드의 완전한 리스트는 JDBC Javadoc를 참조하십시오.

JDBC 3.0을 사용하여 BLOB을 갱신하도록 지원: JDBC 3.0에는 LOB 오브젝트의 변경을 가능하게 하는 지원이 있습니다. 이러한 변경사항은 데이터베이스의 BLOB 열에 저장할 수 있습니다. 이 예는 JDBC 3.0에서 BLOB을 사용하여 수행할 수 있는 타스크의 일부를 보여줍니다.

«

예: BLOB:

»

다음은 데이터베이스에 BLOB을 넣거나 데이터베이스에서 검색하는 방법의 예입니다.

예: BLOB

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
//////////////////////////////////////
// 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.
    }
}
```

예: BLOB 갱신:

다음은 어플리케이션에서 BLOB을 갱신하는 방법의 예입니다.

예: BLOB 갱신

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
//////////////////////////////////////
// UpdateBlobs는 Blob 오브젝트 변경을
// 지원하고, 이 변경을 데이터베이스에
// 적용하는 API의 일부를 나타내는
// 예 프로그램입니다.
//
// 이 프로그램은 PutGetClobs 프로그램
// 완료 후 실행되어야 합니다.
//////////////////////////////////////
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.
   }
}
```

예: BLOB 사용:

다음은 어플리케이션에서 BLOB을 사용하는 방법의 예입니다.

예: BLOB 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
//////////////////////////////////////
// UseBlobs는 Blob 오브젝트와 관련된 API의
// 일부를 나타내는 예 어플리케이션입니다.
//
// 이 프로그램은 PutGetClobs 프로그램
// 완료 후 실행되어야 합니다.
//////////////////////////////////////
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.
    }
}
```

CLOB을 사용하여 코드 기록:

JDBC(Java(TM) Database Connectivity) API를 통해 데이터베이스 CLOB와 DBCLOB 열에 대해 수행할 수 있는 타스크가 매우 많습니다. 다음의 주제는 이러한 타스크를 간략하게 설명하며 타스크 수행 방법의 예를 제공합니다.

데이터베이스에서 CLOB를 읽은 후 데이터베이스에 CLOB 삽입: JDBC API를 사용하면 여러 가지 방법으로 데이터베이스에서 CLOB을 제거하고 데이터베이스에 CLOB을 삽입할 수 있습니다. 그러나 CLOB 오브젝트를 작성하는 표준화된 방법은 없습니다. 데이터베이스가 이미 CLOB으로 가득 찼으면 이 점은 문제가 되지 않지만 JDBC를 통해 스크래치에서 CLOB에 대해 작업하려는 경우에 문제가 될 수 있습니다. JDBC API의 BLOB 및 CLOB 인터페이스에 대한 구성자를 정의하는 대신 데이터베이스에 CLOB을 삽입하고 다른 유형으로 데이터베이스에서 직접 제거하는 지원을 제공합니다. 예를 들어, setCharacterStream 메소드는 유형이 CLOB인 데이터베이스 열에 대해 작업할 수 있습니다. 이 예는 CLOB을 데이터베이스에 삽입하거나 데이터베이스에서 검색하는 공통적인 방법의 일부를 보여줍니다.

Clob 오브젝트 API를 사용하여 작업: CLOB은 다양한 드라이버가 구현을 제공하는 인터페이스로서 JDBC에 정의됩니다. 이 인터페이스에는 CLOB 오브젝트와 대화하는 데 사용할 수 있는 일련의 메소드가 있습니다. 이 예는 이 API를 사용하여 수행할 수 있는 공통적인 타스크의 일부를 보여줍니다. CLOB 오브젝트에서 사용할 수 있는 메소드의 완전한 리스트는 JDBC Javadoc을 참조하십시오.

JDBC 3.0을 사용하여 CLOB을 갱신하도록 지원: JDBC 3.0에는 LOB 오브젝트의 변경을 가능하게 하는 지원이 있습니다. 이러한 변경사항은 데이터베이스의 CLOB 열에 저장할 수 있습니다. 이 예는 JDBC 3.0에서 CLOB 지원을 사용하여 수행할 수 있는 타스크의 일부를 보여줍니다.

«

예: CLOB:

»

다음은 데이터베이스에 CLOB을 넣거나 데이터베이스에서 검색하는 방법의 예입니다.

예: CLOB

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
////////////////////////////////////////
// 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.
   }
}
```

예: CLOB 갱신:

다음은 어플리케이션에서 CLOB을 갱신하는 방법의 예입니다.

예: CLOB 갱신

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
//////////////////////////////////////
// UpdateClobs는 Clob 오브젝트 변경을
// 지원하고, 이 변경을 데이터베이스에
// 적용하는 API의 일부를 나타내는
// 예 프로그램입니다.
//
// 이 프로그램은 PutGetClobs 프로그램
// 완료 후 실행되어야 합니다.
//////////////////////////////////////
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.
   }
}
```

예: CLOB 사용:

다음은 어플리케이션에서 CLOB을 갱신하는 방법의 예입니다.

예: CLOB 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
////////////////////////////////////////
// UpdateClobs는 Clob 오브젝트 변경을
// 지원하고, 이 변경을 데이터베이스에
// 적용하는 API의 일부를 나타내는
// 예 프로그램입니다.
```

```
//
// 이 프로그램은 PutGetClobs 프로그램
// 완료 후 실행되어야 합니다.
/////////////////////////////////////
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.
   }
}
```

Datalink를 사용하여 코드 기록:

Datalink에 대해 작업하는 방법은 작업하고 있는 릴리스에 따라 다릅니다. JDBC 3.0에는 getURL 및 putURL 메소드를 사용하여 Datalink 열에 대해 직접 작업하는 지원이 있습니다. 이전 JDBC 버전에서는 스트링 열인 것처럼 Datalink 열에 대해 작업해야 했습니다. 이제는 데이터베이스가 Datalink와 문자 자료 유형 사이의 자동 변환을 지원하지 않습니다. 결과적으로 SQL문에 일부 유형 캐스트를 수행해야 합니다.

이 예는 Datalink 열에 대해 작업하기 위한 기본 타스크의 일부를 보여줍니다.

예: Datalink:

어플리케이션에서 자료 링크를 사용하는 방법 예입니다.

예: Datalink

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
////////////////////////////////////////
// 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.
    }
}
```

«

예: 고유한 유형:

»

고유한 유형를 사용하는 방법 예입니다.

예: 고유한 유형

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
//////////////////////////////////////////
// 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.
    }
}
```

RowSets

RowSet는 원래 JDBC(Java(TM) Database Connectivity) 2.0 선택적 패키지에 추가된 것입니다. JDBC 스펙의 잘 알려진 일부 인터페이스와 달리 RowSet 스펙은 실제 구현보다 구조에 충실하게 설계되었습니다. RowSet 인터페이스는 모든 RowSet가 가지고 있는 핵심 기능 세트를 정의합니다. RowSet 구현 제공자는 특정 문제 공간에서 요구를 충족시키기 위해 필요한 기능을 자유롭게 정의할 수 있습니다.

원시 JDBC 드라이버를 사용하여 Rowset를 구현하려면 다음을 참조하십시오.

RowSet 특성
RowSet가 충족시킬 특정 등록 정보를 요구할 수 있습니다. 공통 등록 정보에는 결과 RowSet가 지원할 인터페이스 세트가 들어 있습니다.

DB2JdbcRowSet
DB2JdbcRowSet는 DB2ResultSet에서 랩퍼의 역할을 수행하며 이벤트 처리 지원을 제공하는 연결된 RowSet입니다.

DB2CachedRowSet
DB2CachedRowSet는 DB2ResultSet 자료를 오브젝트 내에 저장할 수 있도록 하는 단절된 RowSet입니다. 자료가 오브젝트 내에 있으면 기초 DB2Connection 오브젝트를 닫고 DB2CachedRowSet를 계속 사용할 수 있습니다. DB2CachedRowSet와 관련된 다음의 정보를 찾으십시오.

- DB2CachedRowSets 사용
- DB2CachedRowSet 작성 및 채우기
- DB2CachedRowSet 자료 액세스 및 커서 조작
- DB2CachedRowSet 자료 변경 및 자료 소스에 변경사항 다시 반영
- 기타 DB2CachedRowSet 피처

RowSet 특성:

RowSet가 충족시킬 특정 등록 정보를 요구할 수 있습니다. 공통 등록 정보에는 결과 RowSet가 지원할 인터페이스 세트가 들어 있습니다.

RowSet는 ResultSet: RowSet 인터페이스는 ResultSet 인터페이스를 확장한 것이며 ResultSet가 수행할 수 있는 모든 기능을 RowSet가 수행할 수 있음을 의미합니다. 예를 들어, RowSet는 화면이동 및 갱신이 가능합니다.

데이터베이스에서 단절할 수 있는 RowSet: RowSet의 두가지 범주가 있습니다:

- **연결됨**
  연결된 RowSet가 자료로 채워져도 항상 기초 데이터베이스에 대한 내부 연결을 열고 ResultSet 구현 주변에서 랩퍼의 역할을 수행하게 합니다.
- **단절됨**
  단절된 RowSet는 항상 자료 소스에 대한 연결을 유지할 필요가 없습니다. 단절된 RowSet는 데이터베이스에서 분리시키고 다양한 방법으로 사용한 다음 데이터베이스에 다시 연결시켜서 변경된 사항을 반영할 수 있습니다.

RowSet는 JavaBeans의 구성요소: RowSet에는 JavaBeans 이벤트 처리 모델에 기초한 이벤트 처리를 지원합니다. 또한 설정이 가능한 등록 정보도 있습니다. RowSet는 이러한 등록 정보를 사용하여 다음을 수행할 수 있습니다.

- 데이터베이스에 연결을 설정할 수 있습니다.
- SQL 명령문을 처리할 수 있습니다.
- RowSet가 나타내는 자료의 피처를 판별하고 RowSet 오브젝트의 다른 내부 피처를 처리할 수 있습니다.

일련화 가능한 RowSet: RowSet를 일련화하고 일련화를 취소하면 네트워크 연결을 통해 흐르고 플랫 파일(즉, 워드 프로세싱 또는 기타 구조 문자가 없는 텍스트 문서)에 기록하는 등의 작업이 가능합니다.

DB2CachedRowSet:

DB2CachedRowSet 오브젝트는 단절된 RowSet이며 데이터베이스에 연결하지 않고 사용할 수 있음을 의미합니다. 구현은 CachedRowSet의 설명을 충실하게 따릅니다.

DB2CachedRowSet는 ResultSet의 자료 행에 대한 컨테이너입니다. DB2CachedRowSet는 자체의 모든 자료를 보유하므로 자료를 읽거나 데이터베이스에 자료를 쓰는 동안 명확하지 않게 데이터베이스에 대한 연결을 유지보수할 필요가 없습니다.

DB2CachedRowSet 사용
DB2CachedRowSet에서 제공하는 메소드를 사용하면 여러 사람이 동일한 자료를 사용할 수 있도록 허용하여 데이터베이스의 성능을 향상시킬 수 있습니다. 변경되지 않는 표 자료의 사본을 작성하여 여러 클라이언트에 공통 ResultSet를 분배할 수도 있습니다.

DB2CachedRowSet 작성 및 채우기

다음의 타스크에 따라 자료를 작성하여 DB2CachedRowSet에 넣는 방법에 대해 알아보십시오.

- populate 메소드 사용
- DB2CachedRowSet 등록 정보 및 DataSources 사용
- DB2CachedRowSet 등록 정보 및 JDBC URL 사용
- 기존 데이터베이스 연결을 사용하기 위해 setConnection(Connection) 메소드 사용
- 기존 데이터베이스 연결을 사용하기 위해 execute(Connection) 메소드 사용
- 데이터베이스 요구를 그룹화하기 위해 execute(int) 메소드 사용

DB2CachedRowSet 자료 액세스 및 커서 조작

RowSet는 ResultSet 메소드에 따라 다릅니다. DB2CachedRowSet 자료 액세스 및 커서 이동과 같은 많은 조작에서 ResultSet를 사용하는 경우와 RowSet를 사용하는 경우에 어플리케이션 레벨에 차이가 없습니다.

DB2CachedRowSet 자료 변경 및 자료 소스에 변경사항 다시 반영

DB2CachedRowSet는 RowSet 오브젝트의 자료를 변경하기 위해 표준 ResultSet 인터페이스와 동일한 메소드를 사용합니다. DB2CachedRowSet는 자료가 원래 있었던 데이터베이스에 RowSet에 대한 변경사항을 다시 반영하는 데 사용되는 acceptChanges 메소드를 제공합니다.

기타 DB2CachedRowSet 피처

DB2CachedRowSet 클래스에는 보다 융통성 있게 사용할 수 있도록 하는 추가 기능이 있습니다. DB2CachedRowSet에서 제공하는 메소드를 사용하면 다음의 타스크를 수행할 수 있습니다.

- DB2CachedRowSet에서 콜렉션 확보
- RowSet의 사본 작성
- RowSet에 대한 공유 작성

DB2CachedRowSet 사용:

DB2CachedRowSet 오브젝트는 단절되어 일련화될 수 있으므로, JDBC 드라이버 전체를 실행하는 것이 늘 실용적이지 않은 환경에서 유용합니다(예를 들어, PDA(Personal Digital Assistants) 및 Java(TM)가 가능한 셀룰러 폰).

DB2CachedRowSet 오브젝트가 메모리에 들어 있으며 자료를 항상 알 수 있기 때문에 어플리케이션에 대해 화면이동 가능한 ResultSet의 매우 최적화된 한 형태일 수 있습니다. 화면이동 가능한 DB2ResultSet의 임의적인 이동은 JDBC 드라이버의 자료 행을 캐시하는 기능을 방해할 수 있기 때문에 일반적으로 성능 결함을 발생시킬 수 있지만 RowSet에는 이러한 문제가 없습니다.

DB2CachedRowSet에는 새로운 RowSet를 작성하는 두 가지 메소드가 제공됩니다.

- createCopy 메소드는 복사된 것과 동일한 새로운 RowSet를 작성합니다.
- createShared 메소드는 원본과 동일한 기초 자료를 공유하는 새로운 RowSet를 작성합니다.

createCopy 메소드를 사용하면 여러 클라이언트에 공통 ResultSet를 분배할 수 있습니다. 표 자료가 변경되지 않는 경우에는 RowSet의 사본을 작성하고 이를 각 클라이언트에 전달하는 것이 매번 데이터베이스에 대해 조회를 실행하는 것보다 효율적입니다.

createShared 메소드를 사용하면 여러 사람이 동일한 자료를 사용할 수 있도록 허용하여 데이터베이스의 성능을 향상시킬 수 있습니다. 예를 들어, 고객이 연결하면 홈 페이지에서 가장 잘 팔리는 상위 20개의 제품을 보여주는 웹 사이트를 가지고 있다고 가정합니다. 기본 페이지에 대한 정보를 정기적으로 갱신하려고 하지만 고객이 기본 페이지에 방문할 때마다 가장 자주 구입하는 품목을 알기 위해 조회를 실행하는 것은 실용적이지 못합니다. createShared 메소드를 사용하면 조회를 다시 처리하거나 메모리에 상당한 양의 정보를 저장하지 않아도 각 고객에 대한 ″커서″를 효과적으로 작성할 수 있습니다. 적합한 경우에 가장 자주 구입하는 제품을 찾기 위한 조회를 다시 실행할 수 있습니다. 공유 커서를 작성하는데 사용되며 서브릿이 이를 사용할 수 있도록 RowSet를 새로운 자료로 채울 수 있습니다.

DB2CachedRowSet는 지연 처리 피처를 제공합니다. 이 피처를 사용하면 여러 조회 요청을 함께 그룹화하고 데이터베이스에 대해 하나의 요청으로 처리할 수 있습니다. 데이터베이스가 놓일 수 있는 일부 전산적 억압을 제거하는 212 페이지의 『데이터베이스 요구를 그룹화하기 위해 execute(int) 메소드 사용』 예입니다.

RowSet는 변경된 사항을 데이터베이스에 다시 반영할 수 있도록 주의하여 계속 추적해야 하기 때문에 변경사항을 복원하거나 변경된 모든 사항을 볼 수 있는 기능에 대한 지원이 있습니다. 예를 들어, RowSet에 사용자가 삭제된 행을 페치할 수 있도록 지시하는 데 사용할 수 있는 showDeleted 메소드가 있습니다. 또한 행 삽입 및 삭제가 이루어진 후에 복원할 수 있도록 cancelRowInsert 및 cancelRowDelete 메소드도 있습니다.

DB2CachedRowSet 오브젝트는 이벤트 처리 지원 및 RowSet나 그 일부분을 Java 콜렉션으로 변환할 수 있게 하는 toCollection 메소드로 인해 다른 Java API와의 보다 나은 상호운영성을 제공합니다.

RowSet에 대한 변경이 이루어질 때 변경사항에 대한 정보를 기록하기 위해 화면을 제어하거나 RowSet가 아닌 다른 소스의 변경사항에 대한 정보를 찾을 수 있도록 DB2CachedRowSet의 이벤트 처리 지원을 그래픽 사용자 인터페이스(GUI) 어플리케이션에서 사용할 수 있습니다. 자세한 내용은 예: DB2JdbcRowSet 이벤트를 참조하십시오.

DB2CachedRowSet에 대한 작업의 특정 세부사항은 다음의 주제를 참조하십시오.

- DB2CachedRowSet 작성 및 채우기
- DB2CachedRowSet 자료 액세스 및 커서 조작
- DB2CachedRowSet 자료 변경 및 자료 소스에 변경사항 다시 반영
- 기타 DB2CachedRowSet 피처

이벤트 모델 및 이벤트 처리에 대한 정보는 DB2JdbcRowSet를 참조하십시오. 이 지원은 두 가지 유형의 RowSet에 대해 동일하게 작업합니다.

DB2CachedRowSet 작성 및 채우기:

여러 가지 방법으로 DB2CachedRowSet에 자료를 넣을 수 있습니다.

- 『populate 메소드 사용』
- 『DB2CachedRowSet 등록 정보 및 DataSource 사용』
- 210 페이지의 『DB2CachedRowSet 등록 정보 및 JDBC URL 사용』
- 211 페이지의 『기존 데이터베이스 연결을 사용하기 위해 setConnection(Connection) 메소드 사용』
- 211 페이지의 『기존 데이터베이스 연결을 사용하기 위해 execute(Connection) 메소드 사용』
- 212 페이지의 『데이터베이스 요구를 그룹화하기 위해 execute(int) 메소드 사용』

populate 메소드 사용: DB2CachedRowSet에는 DB2ResultSet 오브젝트에서 RowSet로 자료를 넣는 데 사용할 수 있는 populate 메소드가 있습니다. 다음은 접근방식의 예를 보여줍니다.

예: populate 메소드 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

DB2CachedRowSet 등록 정보 및 DataSource 사용: DB2CachedRowSet에는 DB2CachedRowSet가 SQL 조회 및 DataSource 이름을 수용할 수 있도록 하는 등록 정보가 있습니다. 그런 다음 SQL 조회 및

DataSource 이름을 사용하여 자체에 대한 자료를 작성합니다. 다음은 접근방식의 예를 보여줍니다. 이름이 BaseDataSource인 DataSource에 대한 참조는 이전에 설정된 유효한 DataSource인 것으로 가정합니다.

예: DB2CachedRowSet 등록 정보 및 DataSource 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

DB2CachedRowSet 등록 정보 및 JDBC URL 사용: DB2CachedRowSet에는 DB2CachedRowSet가 SQL 조회 및 JDBC URL을 수용하도록 하는 등록 정보가 있습니다. 그런 다음 조회 및 JDBC URL을 사용하여 자체에 대한 자료를 작성합니다. 다음은 접근방식의 예를 보여줍니다.

예: DB2CachedRowSet 등록 정보 및 JDBC URL 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

기존 데이터베이스 연결을 사용하기 위해 setConnection(Connection) 메소드 사용: JDBC Connection 오브젝트의 재사용을 권장하기 위해 DB2CachedRowSet는 설정된 Connection 오브젝트를 RowSet를 채우는 데 사용하는 DB2CachedRowSet에 전달하는 메카니즘을 제공합니다. 사용자가 제공한 Connection 오브젝트가 전달되면 DB2CachedRowSet는 자체적으로 채운 후에 단절하지 않습니다.

예: 기존 데이터베이스 연결을 사용하기 위해 setConnection(Connection) 메소드 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

기존 데이터베이스 연결을 사용하기 위해 execute(Connection) 메소드 사용: JDBC Connection 오브젝트의 재사용을 권장하기 위해 DB2CachedRowSet는 실행 메소드 호출 시에 DB2CachedRowSet에 설정된 Connection 오브젝트를 전달하는 메카니즘을 제공합니다. 사용자가 제공한 Connection 오브젝트가 전달되면 DB2CachedRowSet는 자체적으로 채운 후에 단절하지 않습니다.

예: 기존 데이터베이스 연결을 사용하기 위해 execute(Connection) 메소드 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

데이터베이스 요구를 그룹화하기 위해 execute(int) 메소드 사용: 데이터베이스의 작업부하를 줄이기 위해 DB2CachedRowSet는 여러 스레드를 데이터베이스에 대해 처리 중인 하나의 요구로 SQL문을 그룹화하는 메카니즘을 제공합니다.

예: 데이터베이스 요구를 그룹화하기 위해 execute(int) 메소드 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

DB2CachedRowSet 자료 액세스 및 커서 조작:

RowSet는 ResultSet 메소드에 따라 다릅니다. 『DB2CachedRowSet 자료에 액세스』 및 215 페이지의 『커서 조작』과 같은 많은 조작에서 ResultSet를 사용하는 경우와 RowSet를 사용하는 경우에 어플리케이션 레벨에는 차이가 없습니다.

DB2CachedRowSet 자료에 액세스: RowSet 및 ResultSet는 동일한 방식으로 자료에 액세스합니다. 다음의 예에서 프로그램은 표를 작성하고 JDBC를 사용하여 다양한 자료 유형으로 채웁니다. 표가 준비되었으면 DB2CachedRowSet가 작성되고 표의 정보로 채워집니다. 이 예는 RowSet 클래스의 다양한 get 메소드도 사용합니다.

예: DB2CachedRowSet 자료 액세스

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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();
    }
  }
}
```

커서 조작: RowSet는 화면이동이 가능하며 화면이동 가능한 ResultSet와 정확히 동일한 작업을 수행합니다. 다음의 예에서 프로그램은 표를 작성하고 JDBC를 사용하여 자료로 표를 채웁니다. 표가 준비되었으면 DB2CachedRowSet 오브젝트가 작성되고 표의 정보로 채워집니다. 이 예에서는 다양한 커서 조작 기능도 사용합니다.

예: 커서 조작

```
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());
  }
 }
}
```

DB2CachedRowSet 자료 변경 및 자료 소스에 변경사항 다시 반영:

DB2CachedRowSet는 RowSet 오브젝트의 자료를 변경하기 위해 표준 ResultSet 인터페이스와 동일한 메소드를 사용합니다. 『DB2CachedRowSet의 행 삭제, 삽입 및 갱신』과 ResultSet의 자료 변경 사이에는 어플리케이션 레벨에 차이가 없습니다. DB2CachedRowSet는 자료가 원래 있었던 220 페이지의 『DB2CachedRowSet의 변경사항을 기초 데이터베이스에 적용』하는 데 사용되는 acceptChanges 메소드를 제공합니다.

DB2CachedRowSet의 행 삭제, 삽입 및 갱신: DB2CachedRowSet를 갱신할 수 있습니다. 다음의 예에서 프로그램은 표를 작성하고 JDBC를 사용하여 자료로 표를 채웁니다. 표가 준비되었으면 DB2CachedRowSet가 작성되고 표의 정보로 채워집니다. 이 예는 RowSet를 갱신하는 데 사용할 수 있는 다양한 메소드도 사용하며 행이 삭제된 후에도 어플리케이션이 이러한 행을 페치할 수 있도록 하는 showDeleted 등록 정보를 사용하는 방법을 보여줍니다. 또한 행 삽입 또는 삭제를 복원하도록 cancelRowInsert 및 cancelRowDelete 메소드를 예에서 사용합니다.

예: DB2CachedRowSet에서 행 삭제, 삽입 및 갱신

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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());
    }
  }
}
```

DB2CachedRowSet의 변경사항을 기초 데이터베이스에 적용: DB2CachedRowSet를 변경한 후에는 RowSet 오브젝트가 존재하는 동안에만 변경사항이 존재합니다. 즉, 단절된 RowSet를 변경해도 데이터베이스에 영향을 주지 않습니다. 기초 데이터베이스에서 RowSet의 변경사항을 반영하기 위해 acceptChanges 메소드를 사용합니다. 이 메소드는 단절된 RowSet에 데이터베이스에 대한 연결을 다시 설정하고 RowSet에 대해 변경한 사항을 기초 데이터베이스에 적용하도록 지시합니다. RowSet가 작성된 후에 다른 데이터베이스 변경사항과의 충돌로 인해 데이터베이스를 안전하게 변경할 수 없으면 예외 상태가 발생하고 트랜잭션이 롤백됩니다.

예: DB2CachedRowSet에 대한 변경사항을 기초 데이터베이스에 다시 반영

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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());
    }
  }
}
```

기타 DB2CachedRowSet 피처:

여러 예에서 확인한 바와 같이 ResultSet와 같이 작업할 뿐만 아니라 DB2CachedRowSet 클래스에는 보다 융통성 있게 사용할 수 있도록 하는 추가 기능이 있습니다. Java(TM) Database Connectivity(JDBC) RowSet 전체나 그 일부를 Java 콜렉션으로 변환하기 위한 메소드가 제공됩니다. 또한 단절된 관계로 인해 DB2CachedRowSet는 ResultSet와 엄격한 일대일 관계를 가지고 있지 않습니다.

DB2CachedRowSet에서 제공하는 메소드를 사용하면 다음의 타스크를 수행할 수 있습니다.

- 『DB2CachedRowSet에서 콜렉션 확보』
- 224 페이지의 『RowSet의 사본 작성』
- 226 페이지의 『RowSet에 대한 공유 작성』

DB2CachedRowSet에서 콜렉션 확보: DB2CachedRowSet 오브젝트에서 콜렉션의 형태를 리턴하는 세 가지 메소드가 있습니다. 다음과 같습니다.

- **toCollection**은 벡터(즉, 각 열에 대해 한 항목)의 ArrayList(즉, 각 행에 대해 한 항목)를 리턴합니다.
- **toCollection(int columnIndex)**은 제공된 열에서 각 행에 대한 값이 포함된 벡터를 리턴합니다.
- **getColumn(int columnIndex)**은 제공된 열에 대해 각 열의 값이 포함된 배열을 리턴합니다.

toCollection(int columnIndex)과 getColumn(int columnIndex)의 주요 차이점은 getColumn 메소드가 기본 유형의 배열을 리턴할 수 있다는 점입니다. 따라서 columnIndex가 정수 자료가 있는 열을 나타내면 java.lang.Integer 오브젝트가 포함된 배열이 아닌 정수 배열이 리턴됩니다.

다음 예는 이런 메소드를 사용할 수 있는 방법을 보여줍니다.

예: DB2CachedRowSet에서 콜렉션 확보

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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());
    }
  }
}
```

RowSet의 사본 작성: createCopy 메소드는 DB2CachedRowSet의 사본을 작성합니다. RowSet와 연관된 모든 자료는 모든 제어 구조, 등록 정보 및 상태 플래그와 함께 복제됩니다.

다음 예는 이 메소드를 사용할 수 있는 방법을 보여줍니다.

예: RowSet의 사본 작성

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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();
    }

  }

}
```

RowSet에 대한 공유 작성: createShared 메소드는 상위 레벨 상태 정보를 사용하여 새로운 RowSet 오브젝트를 작성하고 두 RowSet 오브젝트가 동일한 기초 실제 자료를 공유할 수 있도록 합니다.

다음 예는 이 메소드를 사용할 수 있는 방법을 보여줍니다.

예: RowSet의 공유 작성

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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:

DB2JdbcRowSet는 연결된 RowSet이며 기초 Connection 오브젝트, PreparedStatement 오브젝트 또는 ResultSet 오브젝트의 지원을 통해서만 사용할 수 있음을 의미합니다. 구현은 JdbcRowSet의 설명을 충실하게 따릅니다.

DB2JdbcRowSet 사용: DB2JdbcRowSet 오브젝트는 모든 RowSets에 대한 Java(TM) Database Connectivity(JDBC) 3.0 스펙에서 설명한 이벤트를 지원하므로, 데이터베이스 자료 변경사항을 통지받아야 하는 다른 오브젝트와 로컬 데이터베이스 사이에서 중간 오브젝트의 역할을 수행할 수 있습니다.

한 예로 기본 데이터베이스와 여기에 연결하기 위해 무선 프로토콜을 사용하는 여러 PDA(Personal Digital Assistant)가 있는 환경에서 작업하고 있다고 가정합니다. DB2JdbcRowSet 오브젝트를 사용하여 한 행으로 이동하고 서버에서 실행 중인 마스터 어플리케이션을 사용하여 해당 행을 갱신할 수 있습니다. 행을 갱신하면 RowSet 구성요소에 의해 이벤트가 생성됩니다. PDA에 갱신사항을 보낼 책임이 있는 서비스를 실행하고 있으면 RowSet의 "리스너"로 자체 등록할 수 있습니다. RowSet 이벤트를 받을 때마다 적합한 갱신을 생성하고 무선 장치에 보낼 수 있습니다.

자세한 정보는 예: DB2JdbcRowSet 이벤트를 참조하십시오.

JDBCRowSet 작성: DB2JDBCRowSet 오브젝트를 작성할 수 있도록 몇 가지 메소드가 제공됩니다. 다음은 각 메소드를 요약한 것입니다.

DB2JdbcRowSet 등록 정보 및 DataSources 사용: DB2JdbcRowSet에는 SQL 조회 및 DataSource 이름을 수용하는 등록 정보가 있습니다. 그러면 DB2JdbcRowSet를 사용할 준비가 됩니다. 다음은 접근방식의 예를 보여줍니다. 이름이 BaseDataSource인 DataSource에 대한 참조는 이전에 설정된 유효한 DataSource인 것으로 가정합니다.

예: DB2JdbcRowSet 등록 정보 및 DataSources 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

DB2JdbcRowSet 등록 정보 및 JDBC URL 사용: DB2JdbcRowSet에는 SQL 조회 및 JDBC URL을 수용하는 등록 정보가 있습니다. 그러면 DB2JdbcRowSet를 사용할 준비가 됩니다. 다음은 접근방식의 예를 보여줍니다.

예: DB2JdbcRowSet 등록 정보 및 JDBC URL 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

기존 데이터베이스 연결을 사용하기 위해 setConnection(Connection) 메소드 사용: JDBC Connection 오브젝트의 재사용을 권장하기 위해 DB2JdbcRowSet는 DB2JdbcRowSet에 대해 설정된 연결을 전달할 수 있게 합니다. DB2JdbcRowSet는 이 연결을 사용하여 execute 메소드가 호출되면 자체적으로 사용을 준비합니다.

예: setConnection 메소드 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
// 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();
```

자료 액세스 및 커서 이동: DB2JdbcRowSet를 통한 커서 위치 조작과 데이터베이스 자료에 대한 액세스는 기초 ResultSet 오브젝트가 처리합니다. ResultSet 오브젝트를 통해 수행할 수 있는 타스크는 DB2JdbcRowSet 오브젝트에도 적용됩니다.

자료 변경 및 변경사항을 기초 데이터베이스에 반영: DB2JdbcRowSet를 통한 데이터베이스 갱신 지원은 기초 ResultSet 오브젝트가 모두 처리합니다. ResultSet 오브젝트를 통해 수행할 수 있는 타스크는 DB2JdbcRowSet 오브젝트에도 적용됩니다.

DB2JdbcRowSet 이벤트:

모든 RowSet 구현은 다른 구성요소와 관련이 있는 상황에 대한 이벤트 처리를 지원합니다. 이 지원을 사용하면 어플리케이션 구성요소들은 이벤트 발생 시에 서로 ″대화″할 수 있습니다. 예를 들어, RowSet를 통해 데이터베이스 행을 갱신하면 그래픽 사용자 인터페이스(GUI) 표가 표시되어 자체적으로 갱신됩니다.

다음의 예에서 기본 메소드는 RowSet를 갱신하며 핵심 어플리케이션입니다. 리스너는 이 필드에서 단절된 클라이언트가 사용하는 무선 서버의 일부입니다. 두 프로세스의 코드를 뒤섞지 않고 이들 두 비즈니스 측면을 함께 묶을 수 있습니다. RowSet의 이벤트 지원은 주로 데이터베이스 자료를 사용하여 GUI를 갱신하도록 설계되었지만 이러한 유형의 어플리케이션 문제에도 매우 적합합니다.

예: DB2JdbcRowSet 이벤트

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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 드라이버의 성능 추가 정보

IBM Developer Kit for Java(TM) JDBC 드라이버는 데이터베이스와의 작업시 고성능 Java 인터페이스를 위해 설계된 것입니다. 그러나 최적의 가능한 성능을 확보하려면 JDBC 드라이버가 제공해야 하는 능력의 장점을 활용하는 방법으로 어플리케이션을 빌드해야 합니다. 다음의 추가 정보는 훌륭한 JDBC 프로그래밍 연습이

될 수 있습니다. 대부분 원시 JDBC 드라이버에만 해당되는 것은 아닙니다. 따라서, 이러한 지침에 따라 기록된 어플리케이션은 원시 JDBC 드라이버 이외의 JDBC 드라이버와 함께 사용할 경우에도 잘 수행됩니다.

- SELECT * SQL 조회 방지(233 페이지 참조)
- getXXX(String) 대신 getXXX(int)사용(233 페이지 참조)
- Java 기본 유형에 대한 getObject 호출 방지(233 페이지 참조)
- Statement에 대해 PreparedStatement 사용(234 페이지 참조)
- 비용이 많이 드는 DatabaseMetaData 호출 방지(234 페이지 참조)
- 어플리케이션에 대한 올바른 확약 레벨 사용(234 페이지 참조)
- 유니코드로 자료 저장 고려(234 페이지 참조)
- 저장 프로시듀어 사용(234 페이지 참조)
- 숫자/십진수 대신 BigInt 사용(235 페이지 참조)
- JDBC 자원에 대한 작업이 완료되면 외부적으로 닫기(235 페이지 참조)
- 연결 풀 사용(235 페이지 참조)
- PreparedStatement 풀 사용 고려(235 페이지 참조)
- 효율적인 SQL 사용(235 페이지 참조)

SELECT * SQL 조회 방지

SELECT * FROM...은 SQL에서 조회를 명시하기 위한 공통적인 방법입니다. 그러나 때때로 모든 필드를 조회할 필요가 없습니다. 리턴될 각 열에 대해 JDBC 드라이버는 행을 바인드하고 리턴하는 추가 작업을 수행해야 합니다. 어플리케이션이 특정 열을 전혀 사용하지 않더라도 JDBC 드라이버는 이 열을 인식해야 하며 그 사용을 위해 공간을 예약해야 합니다. 표에 사용되지 않는 열이 거의 없는 경우 이 점은 중요한 오버헤드가 아닙니다. 그러나 사용되지 않는 열의 수가 많은 경우 오버헤드가 심각할 수 있습니다. 더 좋은 해결책은 다음과 같이 어플리케이션이 개별적으로 관심이 있는 열을 나열하는 것입니다.

```
SELECT COL1, COL2, COL3 FROM...
```

getXXX(String) 대신 getXXX(int) 사용

열 이름을 사용하는 버전 대신 숫자 값을 사용하는 ResultSet getXXX 메소드를 사용하십시오. 숫자 상수 대신 지유롭게 열 이름을 사용하는 것이 이점이 있는 것으로 보이지만 데이터베이스 자체는 열 색인을 취급하는 방법만 알고 있습니다. 따라서 호출하는 열 이름이 있는 각 getXXX 메소드는 데이터베이스에 전달하기 전에 JDBC 드라이버가 분석해야 합니다. getXXX 메소드는 일반적으로 수백만 번 실행할 수 있는 루프 내에서 호출되기 때문에 이 작은 오버헤드가 빠르게 누적될 수 있습니다.

Java 기본 유형에 대한 getObject 호출 방지

기본 유형(ints, longs, floats, 등)의 데이터베이스에서 값을 취할 때 getObject를 사용하는 것보다 기본 유형에만 해당되는 get 메소드(getInt, getLong, getFloat)를 사용하는 것이 훨씬 빠릅니다. getObject 호출은 기본 유형에 대해 get 작업을 수행한 다음, 사용자에게 리턴할 오브젝트를 작성합니다. 이 작업은 일반적으로 루

프에서 수행되며 잠재적으로 수명이 짧은 수백만 개의 오브젝트를 작성합니다. 기본 명령에 대해 getObject를 사용하면 가비지 콜렉터를 자주 활성화하여 결국 성능을 저하시키는 결점이 추가됩니다.

Statement에 대해 PreparedStatement 사용

한 번 이상 사용될 SQL문을 기록하는 경우 Statement보다는 PreparedStatement로서 수행하는 것이 낫습니다. 명령문을 실행할 때마다 명령문 준비 및 명령문 프로세스의 2단계 프로세스를 통과하게 됩니다. 준비된 명령문을 사용할 때 명령문은 실행될 때마다가 아니라 구성되었을 때에만 준비됩니다. PreparedStatement가 Statement 보다 빠르게 수행되는 것으로 인식되지만 이 장점은 때때로 프로그래머에 의해 무시됩니다. PreparedStatement가 제공하는 성능 장점으로 인해 가능하다면 어디에나 어플리케이션의 설계에 이를 사용하는 것이 유리합니다(PreparedStatement 풀(235 페이지 참조) 참조).

DatabaseMetaData 호출 방지

DatabaseMetaData 호출의 일부는 비용이 많이 들 수 있다는 점을 유의하십시오. 특히 getBestRowIdentifier, getCrossReference, getExportedKeys 및 getImportedKeys 메소드가 비용이 많이 들 수 있습니다. 일부 DataBaseMetaData 호출에는 시스템 레벨표에 대해 복잡한 결합 조건이 관련되어 있습니다. 편의를 위한 것일 뿐이라면 정보가 필요한 경우에만 사용하십시오.

어플리케이션에 대한 올바른 확약 레벨 사용

JDBC는 여러 트랜잭션이 시스템에 대해 서로 어떠한 영향을 미치는지 판별하는 몇 가지 확약 레벨을 제공합니다(자세한 내용은 트랜잭션 참조). 디폴트는 최저 확약 레벨을 사용하는 것입니다. 이것은 트랜잭션이 확약 경계를 통해 서로 작업의 일부를 볼 수 있음을 의미합니다. 따라서, 특정 데이터베이스 이상의 가능성이 있습니다. 일부 프로그래머는 이러한 이상 발생에 대해 염려할 필요가 없도록 확약 레벨을 증가시킵니다. 확약 레벨이 높을수록 데이터베이스에 보다 과정이 복잡한 잠금이 설정된다는 점을 유의하십시오. 이것은 시스템이 가질 수 있는 동시성의 양을 제한하여 일부 어플리케이션의 성능이 심각하게 저하될 수 있습니다. 때때로, 처음에 어플리케이션의 설계로 인해 이상 상태가 발생할 수 있습니다. 수행하고자 하는 것이 무엇인지 이해하는데 충분한 시간을 할애하고 트랜잭션 분리 레벨을 안전하게 사용할 수 있는 최저 레벨로 제한하십시오.

유니코드로 자료 저장 고려

Java는 작업하는 모든 문자 자료(스트링)를 유니코드로 작성할 것을 요구합니다. 따라서, 유니코드 자료가 없는 표는 자료를 데이터베이스에 넣고 데이터베이스에서 검색하기 때문에 자료를 앞뒤로 변환하는데 JDBC 드라이버를 필요로 합니다. 표가 이미 유니코드로 작성된 경우 JDBC 드라이버는 자료를 변환할 필요가 없으므로 데이터베이스에서 자료를 보다 빠르게 배치할 수 있습니다. 유니코드로 작성된 자료는 Java가 아닌 어플리케이션에 대해 작업할 수 없기 때문에 유니코드를 취급하는 방법을 알지 못하므로 이러한 자료를 이해하는데 주의하십시오. 또한 비문자 자료는 변환이 전혀 이루어지지 않으므로 보다 빠르게 수행되지 않는다는 점을 기억하십시오. 또 다른 사항은 유니코드로 저장된 자료는 1바이트 자료보다 두 배 많은 공간을 차지한다는 점입니다. 그러나 여러 번 읽히는 문자 열이 많은 경우 자료를 유니코드로 저장하여 얻어지는 성능 장점은 상당히 높을 수 있습니다.

저장 프로시듀어 사용

저장 프로시듀어의 사용이 Java에서 지원됩니다. 저장프로시듀어는 JDBC 드라이버가 동적 SQL 대신 정적 SQL을 실행하도록 하여 보다 빠르게 수행할 수 있습니다. 프로그램에서 실행하는 각각의 개별 SQL문에 대해 저장 프로시듀어를 작성하지 마십시오. 그러나 가능하면 SQL문 그룹을 실행하는 저장 프로시듀어를 작성하십시오.

숫자 또는 십잔수 대신 BigInt 사용

스케일이 0인 숫자나 십진수를 사용하는 대신 BigInt 자료 유형을 사용하십시오. 숫자나 십진수 자료 유형은 String 또는 BigDecimal 오브젝트로 변환되는 반면, BigInt는 Java 기본 유형 Long으로 직접 변환됩니다. Java 기본 유형에 대한 getObject 호출 무시(233 페이지 참조)에 언급된 바와 같이 기본 자료 유형을 사용하는 것이 오브젝트 작성이 요구되는 유형을 사용하는 것보다 바람직합니다.

JDBC 자원에 대한 작업이 완료되면 외부적으로 닫기

ResultSet, Statement 및 Connection이 더 이상 필요하지 않으면 어플리케이션이 외부적으로 닫아야 합니다. 그러면, 가능한한 가장 효율적인 방법으로 자료를 정리할 수 있으며 성능이 증가됩니다. 또한 외부적으로 닫지 않은 데이터베이스 자원은 자원 누출을 초래하며 데이터베이스 잠금이 필요한 것보다 길게 유지됩니다. 이것은 어플리케이션 실패로 이어지거나 어플리케이션들의 동시성이 줄어들 수 있습니다.

연결 풀 사용

연결 풀은 여러 사용자들이 각각 자신의 연결 오브젝트를 작성하는 각 사용자 요구 대신 여러 사용자에 대해 JDBC 연결 오브젝트를 재사용하도록 하는 전략입니다. 연결 오브젝트는 작성하는 데 비용이 많이 듭니다. 각 사용자가 새로 작성하는 대신 이들의 풀을 성능이 중요한 어플리케이션에서 공유해야 합니다. 많은 제품들(예 : WebSphere)은 사용자 측의 추가적인 노력을 거의 필요로 하지 않고 사용할 수 있는 연결 풀 지원을 제공합니다. 연결 풀 지원과 함께 제품을 사용하지 않으려는 경우 또는 풀 작업과 수행 방법을 보다 잘 제어하기 위해 사용자 자신의 것을 빌드하려는 경우 이렇게 하는 것이 합리적으로 쉽습니다.

PreparedStatement 풀 사용 고려

명령문 풀은 연결 풀과 유사한 방식으로 작업합니다. 단지 연결을 풀에 넣는 대신 연결 및 PreparedStatement가 들어 있는 오브젝트를 풀에 넣으십시오. 그런 다음, 이 오브젝트를 검색하고 사용하려는 특정 명령문에 액세스하십시오. 그러면, 성능이 상당히 증가할 수 있습니다.

효율적인 SQL 사용

SQL에 위해 JDBC가 빌드되었기 때문에 효율적인 SQL에 기여하는 거의 모든 것은 효율적인 JDBC에 기여합니다. 따라서, JDBC는 최적화된 조회, 현명하게 선택한 색인 및 좋은 SQL 설계의 다른 측면과 같은 장점을 이어받습니다.

IBM Developer Kit for Java DB2 SQLJ 지원을 사용한 데이터베이스 액세스

DB2 SQLJ(Structured Query Language for Java(TM)) 지원은 SQLJ ANSI 표준을 기반으로 합니다. DB2 SQLJ 지원은 IBM Developer Kit for Java에 포함되어 있습니다. DB2 SQLJ 지원을 사용하여 삽입 Java용 SQL 어플리케이션을 작성, 빌드 및 실행할 수 있습니다.

IBM Developer Kit for Java에서 제공하는 SQLJ 지원에는 SQLJ 런타임 클래스가 포함되며 /QIBM/ProdData/Java400/ext/runtime.zip에서 사용할 수 있습니다. SQLJ 런타임 클래스에 대한 자세한 정보는 www.sqlj.org 에서 실행시 제공되는 Runtime API 문서를 참조하십시오.

SQLJ 툴

다음의 툴도 IBM Developer Kit for Java에서 제공하는 SQLJ 지원에 포함됩니다.

- SQLJ 변환 프로그램인 `sqlj`는 Java 소스 명령문으로 SQLJ 프로그램에 있는 삽입 SQL문을 대체하고 SQLJ 프로그램에서 발견되는 SQLJ 조작에 대한 정보가 들어 있는 일련의 프로파일을 생성합니다.
- DB2 SQLJ 프로파일 조정 프로그램인 `db2profc`는 생성된 프로파일에 저장되는 SQL문을 사전컴파일하고 DB2 데이터베이스에 패키지를 생성합니다.
- DB2 SQLJ 프로파일 프린터인 `db2profp`는 보통 텍스트로 DB2 사용자 정의 프로파일의 내용을 인쇄합니다.
- SQLJ 프로파일 작성자 설치 프로그램인 `profdb`는 2진 프로파일의 기존 세트에 디버깅 클래스 작성자를 설치하고 설치 제거합니다.
- SQLJ 프로파일 변환 툴인 `profconv`는 일련의 프로파일 인스턴스를 Java 클래스 형식으로 변환합니다.

주: 이들 툴은 Qshell 인터프리터에서 실행되어야 합니다.

DB2 SQLJ 제한사항

SQLJ로 DB2 어플리케이션을 작성할 경우 다음 제한사항을 참조하십시오.

- DB2 SQLJ 지원은 SQL문 발행에 대한 표준 DB2 Universal Database 제한사항을 준수합니다.
- DB2 SQLJ 프로파일 조정자는 로컬 데이터베이스에 대한 연결과 연관되는 프로파일만 실행시켜야 합니다.
- SQLJ 참조 구현에는 JDK 1.1 이상이 필요합니다. JDK의 복수 버전 실행에 대한 자세한 정보는 복수 JDK(Java Development Kit)에 대한 지원을 참조하십시오.

Java 어플리케이션에서의 SQL 사용에 대한 정보는 Java 어플리케이션에 SQL문 삽입 및 SQLJ 프로그램 컴파일 및 실행을 참조하십시오.

SQLJ(Structured Query Language for Java) 프로파일

프로파일은 SQLJ 소스 파일을 변환할 때 SQLJ 변환 프로그램인 sqlj에 의해 생성됩니다. 프로파일은 일련의 2진 파일입니다. 이것이 이 파일들이 .ser 확장자를 갖는 이유입니다. 이들 파일에는 연관된 SQLJ 소스 파일의 SQL문이 들어 있습니다.

SQLJ 소스 코드에서 프로파일을 생성하려면, .sqlj 파일에서 SQLJ 변환 프로그램인 sqlj를 실행하십시오.

자세한 정보는 SQLJ 프로그램 컴파일 및 실행을 참조하십시오.

SQLJ(structured query language for Java) 변환 프로그램(sqlj)

SQLJ 변환 프로그램인 sqlj는 SQLJ 프로그램에서 발견되는 SQL 조작에 대한 정보가 들어 있는 일련의 프로파일을 생성합니다. SQLJ 변환 프로그램은 /QIBM/ProdData/Java400/ext/translator.zip 파일을 사용합니다.

sqlj 명령행 옵션에 대한 자세한 정보는 www.sqlj.org 의 구현에서 제공하는 SQLJ User's Guide and Reference를 참조하십시오.

DB2 SQLJ 프로파일 조정자 db2profc를 사용하여 프로파일의 SQL문 사전컴파일

DB2 SQLJ 프로파일 조정자 db2profc를 사용하여 Java(TM) 어플리케이션이 보다 효율적으로 데이터베이스에 대해 작업하도록 할 수 있습니다.

DB2 SQLJ 프로파일 조정자는 다음을 수행합니다.

- 프로파일에 저장되는 SQL문을 사전컴파일하고 DB2 데이터베이스에 패키지를 생성합니다.
- SQL문을 작성된 패키지에 있는 연관된 명령문에 대한 참조로 대체하여 SQLJ 프로파일을 사용자 정의합니다.

프로파일에 있는 SQL문을 사전컴파일하려면 Qshell 명령 프롬트에 다음을 입력하십시오.

```
db2profc MyClass_SJProfile0.ser
```

여기서 *MyClass_SJProfile0.ser*은 사전컴파일하려는 프로파일의 이름입니다.

DB2 SQLJ 프로파일 조정자 사용 및 구문

db2profc[*options*] <*SQLJ_profile_name*>

여기서 *SQLJ_profile_name*은 인쇄될 프로파일의 이름이며 *options*는 원하는 옵션 리스트입니다.

db2profp에 사용할 수 있는 옵션은 다음과 같습니다.

- `-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>`

이 옵션에 대한 설명은 다음과 같습니다:

-URL=<*JDBC_URL*>

여기서 *JDBC URL*은 JDBC 연결의 URL입니다. URL에 대한 구문은 다음과 같습니다.

```
"jdbc:db2:systemName"
```

자세한 정보는 IBM Developer Kit for Java JDBC 드라이버를 사용하여 iSeries 데이터베이스에 액세스를 참조하십시오.

-user=<*username*>

Where *username*은 사용자 이름입니다. 디폴트 값은 로컬 연결을 위해 사인 온되는 현재 사용자의 사용자 ID입니다.

-password=<*password*>

여기서 *password*는 사용자 암호입니다. 디폴트 값은 로컬 연결을 위해 사인 온되는 현재 사용자의 암호입니다.

-package=<*library name/package name*>

여기서 *library name*은 패키지가 배치될 라이브러리이며, *package name*은 생성될 패키지의 이름입니다. 디폴트 라이브러리명은 QUSRSYS입니다. 디폴트 패키지명은 프로파일명으로부터 생성됩니다. 패키지명의 최대 길이는 10자입니다. SQLJ 프로파일명이 항상 10자보다 길기 때문에, 디폴트 패키지명은 프로파일명과 다를 것입니다. 디폴트 패키지명은 프로파일명의 첫 번째 문자를 프로파일 키 번호와 연결시켜서 구성됩니다. 프로파일키 번호가 10자보다 긴 경우 프로파일 키 번호의 마지막 10자가 디폴트 패키지명에 사용됩니다. 예를 들어, 다음 도표는 몇몇 프로파일명과 그들의 디폴트 패키지명을 보여줍니다.

프로파일 이름	디폴트 패키지 이름
App_SJProfile0	App_SJPro0
App_SJProfile01234	App_S01234
App_SJProfile012345678	A012345678
App_SJProfile01234567891	1234567891

-commitctrl=<*commitment_control*>

여기서 *commitment_control*은 사용자가 원하는 확약 제어 레벨입니다. 확약 제어는 다음 문자 값 중 하나를 가질 수 있습니다.

값	정의
C	*CHG. Dirty 읽기, 반복 불가능 읽기 및 가상(phantom) 읽기가 가능합니다.
S	*CS. Dirty 읽기는 불가능하지만, 반복 불가능 읽기 및 가상(phantom) 읽기가 가능합니다.
A	*ALL. Dirty 읽기와 반복 불가능 읽기는 불가능하지만 가상(phantom) 읽기가 가능합니다.
N	*NONE. Dirty 읽기, 반복 불가능 읽기 및 가상(phantom) 읽기가 불가능합니다. 이것이 디폴트입니다.

-datefmt=<*date_format*>

여기서 *date_format*는 사용자가 원하는 날짜 형식의 유형입니다. 날짜 형식은 다음 값 중 하나를 가질 수 있습니다.

값	정의
USA	IBM USA 표준(mm.dd.yyyy,hh:mm a.m., hh:mm p.m.)
ISO	국제 표준화 기구(yyyy-mm-dd, hh.mm.ss). 이것이 디폴트입니다.
EUR	IBM 유럽 표준(dd.mm.yyyy, hh.mm.ss)
JIS	일본 산업 표준 서기력(yyyy-mm-dd, hh:mm:ss)
MDY	월/일 년(mm/d/yy)
DMY	일/월/년(dd/mm/yy)
YMD	년/월/일(yy/mm/dd)
JUL	율리우스력(yy/ddd)

날짜 형식은 날짜 결과 열에 액세스할 때 사용됩니다. 모든 출력 자료 필드가 지정된 형식으로 리턴됩니다. 입력 날짜 스트링의 경우 지정된 값을 사용하여 날짜를 유효한 형식으로 지정했는지의 여부를 판별합니다. 디폴트 값은 ISO입니다.

-datesep=<*date_separator*>

*date_separator*는 사용자가 사용하려는 분리자 유형입니다. 날짜 분리자는 날짜 결과 열에 액세스할 때 사용됩니다. 날짜 분리자는 다음 값 중 하나일 수 있습니다.

값	정의
/	슬래시(/)가 사용됩니다.
.	마침표가 사용됩니다.
,	쉼표가 사용됩니다.
-	대시(-)가 사용됩니다. 이것이 디폴트입니다.
공백	공백이 사용됩니다.

-timefmt=<*time_format*>

*time_format*는 시간 필드를 표시하는데 사용하는 형식입니다. 시간 형식은 시간 결과 열에 액세스할 때 사용됩니다. 입력 시간 스트링의 경우 지정된 값이 시간이 유효한 형식으로 지정되는지 여부를 판별하는데 사용됩니다. 시간 형식은 다음 값 중 하나를 가질 수 있습니다.

값	정의
USA	IBM USA 표준(mm.dd.yyyy,hh:mm a.m., hh:mm p.m.)
ISO	국제 표준화 기구(yyyy-mm-dd, hh.mm.ss). 이것이 디폴트입니다.
EUR	IBM 유럽 표준(dd.mm.yyyy, hh.mm.ss)
JIS	일본 산업 표준 서기력(yyyy-mm-dd, hh:mm:ss)
HMS	시간/분/초(hh:mm:ss)

-timesep=<*time_separator*>

여기서 *time_separator*는 시간 결과 열에 액세스하는 데 사용하는 문자입니다. 시간 분리자는 다음 값 중 하나를 가질 수 있습니다.

값	정의
:	콜론이 사용됩니다.
.	마침표가 사용됩니다. 이것이 디폴트입니다.
,	쉼표가 사용됩니다.
공백	공백이 사용됩니다.

-decimalpt=<*decimal_point*>

여기서 *decimal_point*는 사용하는 소수점입니다. 소수점은 SQL문에서 숫자 상수에 사용됩니다. 소수점은 다음 값 중 하나일 수 있습니다.

값	정의
.	마침표가 사용됩니다. 이것이 디폴트입니다.
,	쉼표가 사용됩니다.

-stmtCCSID=<*CCSID*>

*CCSID*는 패키지로 준비되는 SQL문에 대한 코드화 문자 세트 ID입니다. 사용자 정의 시간 중 작업의 값이 디폴트 값입니다.

-sorttbl=<*library_name/sort_sequence_table_name*>

여기서 *library_name/sort_sequence_table_name*은 사용하려는 정렬 순서표의 위치와 이름입니다. 정렬 순서표는 SQL문에 있는 스트링 비교에 사용됩니다. 라이브러리명과 정렬 순서표 이름은 각각 10자로 제한됩니다. 디폴트 값은 사용자 정의 시간 중에 작업으로부터 나옵니다.

-langID=<*language_identifier*>

여기서 *language_identifier*는 사용하려는 언어 ID입니다. 언어 ID에 대한 디폴트 값은 사용자 정의 시간 중에 현재 작업으로부터 취해집니다. 언어 ID는 정렬 순서표와 결합되어 사용됩니다.

위의 필드에 대한 자세한 정보는 DB2 for iSeries 400 SQL Programming Concepts, SC41-5611 을 참조하십시오.

DB2 SQLJ 프로파일(db2profp 및 profp) 내용 인쇄

DB2 SQLJ 프로파일 프린터인 db2profp는 DB2 사용자 정의 프로파일의 내용을 보통 텍스트로 인쇄합니다. 프로파일 프린터인 profp는 SQLJ 변환 프로그램에 의해 생성되는 프로파일의 내용을 보통 텍스트로 인쇄합니다.

SQLJ 변환 프로그램에 의해 생성되는 프로파일의 내용을 보통 텍스트로 인쇄하려면 다음과 같이 profp 유틸리티를 사용하십시오.

```
profp MyClass_SJProfile0.ser
```

여기서 *MyClass_SJProfile0.ser*은 인쇄하려는 프로파일의 이름입니다.

프로파일의 DB2 사용자 정의 버전의 내용을 보통 텍스트로 인쇄하려면 다음과 같이 db2profp 유틸리티를 사용하십시오.

```
db2profp MyClass_SJProfile0.ser
```

여기서 *MyClass_SJProfile0.ser*은 인쇄하려는 프로파일의 이름입니다.

주: 사용자 정의되지 않은 프로파일에서 db2profp를 실행하면 프로파일이 사용자 정의되지 않았다고 통지합니다. 사용자 정의된 프로파일에서 profp를 실행하면 사용자 정의없이 프로파일의 내용을 표시합니다.

DB2 SQLJ 프로파일 프린터 사용 및 구문:

```
db2profp [options] <SQLJ_profile_name>
```

여기서 *SQLJ_profile_name*은 인쇄될 프로파일의 이름이며 *options*는 원하는 옵션 리스트입니다.

db2profp에 사용할 수 있는 옵션은 다음과 같습니다.

-URL=<*JDBC_URL*>

여기서 *JDBC_URL*은 연결하려는 URL입니다. 자세한 정보는 IBM Developer Kit for Java JDBC 드라이버를 사용하여 iSeries 데이터베이스에 액세스를 참조하십시오.

-user=<*username*>

여기서 *username*은 사용자 프로파일의 사용자명입니다.

-password=<*password*>

여기서 *password*는 사용자 프로파일의 암호입니다.

SQLJ 프로파일 감사 설치 프로그램(profdb)

SQLJ 프로파일 감사 설치 프로그램(profdb)은 디버깅 클래스 감사를 설치하고 설치제거합니다. 디버깅 클래스 작성자는 2진 프로파일의 기존 세트에 설치됩니다. 일단 디버깅 클래스 작성자가 설치되면 어플리케이션 실행 시에 만들어진 모든 RTStatement 및 RTResultSet 호출이 기록됩니다. 이들 호출은 파일이나 표준 출력에 기록될 수 있습니다. 그런 다음 어플리케이션의 작동과 추적 오류를 확인하기 위해 기록부를 검사할 수 있습니다. 실행 시간에 기저의 RTStatement 및 RTResultSetcall 인터페이스에 수행된 호출만 감사됨을 주의하십시오.

디버깅 클래스 작성자를 설치하려면 Qshell 명령 프롬트에 다음을 입력하십시오.

```
profdb MyClass_SJProfile0.ser
```

여기서 *MyClass_SJProfile0.ser*은 SQLJ 변환 프로그램에 의해 생성된 프로파일의 이름입니다.

디버깅 클래스 작성자를 설치 취소하려면 Qshell 명령 프롬트에 다음을 입력하십시오.

```
profdb -Cuninstall MyClass_SJProfile.ser
```

여기서 *MyClass_SJProfile0.ser*은 SQLJ 변환 프로그램에 의해 생성된 프로파일의 이름입니다.

profdb 명령행 옵션의 자세한 정보는 www.sqlj.org 를 참조하고, 구현 범주를 선택한 다음 Runtime API 문서에 있는 sqlj.runtime.profile.util.AuditorInstaller 클래스로 이동하십시오.

SQLJ 프로파일 변환 툴(profconv)을 사용하여 Java 클래스 형식으로 일련화된 프로파일 인스턴스 변환

SQLJ 프로파일 변환 툴(profconv)은 일련화된 프로파일 인스턴스를 Java(TM) 클래스 형식으로 변환합니다. profconv 툴은 일부 브라우저가 애플릿과 연관되는 자원 파일에서의 일련화된 오브젝트의 로드를 지원하지 않기 때문에 필요합니다. 변환을 수행하려면 profconv 유틸리티를 실행하십시오.

profconv 유틸리티를 실행하려면 Qshell 명령행에 다음을 입력하십시오.

```
profconv MyApp_SJProfile0.ser
```

여기서 *MyApp_SJProfile0.ser*은 변환하려는 프로파일 인스턴스의 이름입니다.

`profconv` 툴은 `sqlj -ser2class`를 호출합니다. 명령행 옵션에 대해서는 sqlj를 참조하십시오.

Java 어플리케이션에 SQL문 삽입

SQLJ에 있는 정적 SQL문은 SQLJ 절에 있습니다. SQLJ 절은 `#sql`로 시작하고 세미콜론(;) 문자로 끝납니다.

Java(TM) 어플리케이션에서 SQLJ 절을 작성하기 전에 다음 패키지를 가져오십시오.

- import java.sql.*;
- import sqlj.runtime.*;
- import sqlj.runtime.ref.*;

가장 간단한 SQLJ 절은 대괄호로 묶인 SQL문이 뒤따르는 토큰 `#sql`로 구성되고 프로세스됩니다. 예를 들어, 다음 SQLJ 절은 Java 명령문이 합법적으로 나타날 수 있는 모든 곳에 나타날 수 있습니다.

```
#sql { DELETE FROM TAB };
```

이전 예는 TAB이라는 표에서 모든 행을 삭제합니다.

주: SQLJ 어플리케이션 컴파일 및 실행에 대한 정보는 SQLJ 프로그램 컴파일 및 실행을 참조하십시오.

SQLJ 프로세스 절에서, 대괄호 안에 나타나는 토큰은 SQL 토큰 또는 호스트 변수입니다. 모든 호스트 변수는 콜론(:) 문자에 의해 구분됩니다. SQL 토큰은 SQLJ 프로세스 절의 대괄호 밖에는 절대로 발생하지 않습니다. 예를 들어, 다음 Java 메소드는 인수 표에 인수를 삽입합니다.

```
public void insertIntoTAB1 (int x, String y, float z) throws SQLException
{
    #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

메소드 본문은 호스트 변수 x, y 및 z가 들어 있는 SQLJ 프로세스절로 구성됩니다. 호스트 변수에 대한 자세한 정보는 SQLJ의 호스트 변수를 참조하십시오.

일반적으로, SQL 토큰은 대소문자를 구별하지 않으며(큰 따옴표로 구분되는 ID의 경우는 예외), 대문자, 소문자 또는 혼합문자로 쓸 수 있습니다. 그러나 *Java* 토큰은 대소문자를 구분합니다. 이해를 돕기 위해 예에서는

대소문자를 구분하지 않는 SQL 토큰에는 대문자, Java 토큰에는 소문자나 혼합문자를 사용했습니다. 이 주제 전체에서, 소문자 null은 Java "널(null)" 값을 표시하고 대문자 NULL은 SQL "널(null)" 값을 표시하는데 사용됩니다.

다음 유형의 SQL 구조가 SQLJ 프로그램에 나타날 수 있습니다.

- 조회
  예를 들면, SELECT문과 표현식.
- SQL 자료 변경 명령문(DML)
  예를 들어, INSERT, UPDATE, DELETE.
- 자료 명령문
  예를 들어, FETCH, SELECT..INTO.
- 변환 제어 명령문
  예를 들어, COMMIT, ROLLBACK 등.
- 자료 정의 언어(DDL, 스키마 조작 언어라고도 부름) 명령문
  예를 들어, CREATE, DROP, ALTER.
- 저장 프로시듀어에 대한 호출
  예를 들어, CALL MYPROC(:x, :y, :z)
- 저장된 기능의 호출
  예를 들어, VALUES(MYFUN(:x))

삽입 SQLJ의 예는 예: Java 어플리케이션의 내장 SQL문을 참조하십시오.

SQLJ(Structured Query Language for Java)의 호스트 변수: 삽입 SQL문에 대한 인수는 호스트 변수를 통해 전달됩니다. 호스트 변수는 호스트 언어의 변수이며, SQL문에 나타날 수 있습니다. 호스트 변수는 다음과 같이 최고 3개의 부분을 갖습니다.

- 콜론(:) 접두부.
- 매개변수, 변수 또는 필드에 대한 Java ID인 Java(TM) 호스트 변수.
- 선택적 매개변수 모드 ID.

  이 모드 ID는 다음 중 하나일 수 있습니다.
  IN, OUT 또는 INOUT.

Java ID의 평가는 Java 프로그램에서 부가 효과를 갖지 않으므로, SQLJ 절을 대체하기 위해 생성되는 Java 코드에 여러 번 나타날 수 있습니다.

호스트 변수를 포함하는 다음 조회는 조회를 포함하는 범위에서 가시적인 Java 변수, 필드 또는 매개변수 x입니다.

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

예: Java 어플리케이션에 삽입된 SQL문: 다음의 SQLJ 어플리케이션 예인 App.sqlj는 정적 SQL을 사용하여 DB2 샘플 데이터베이스의 EMPLOYEE 표에서 자료를 검색하고 갱신합니다.

예: Java(TM) 어플리케이션의 삽입된 SQL문

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ; // 1(247 페이지 참조)
#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(247 페이지 참조)
{
  str1 = cursor1.empno(); // 4(247 페이지 참조)
  str2 = cursor1.firstnme();

  System.out.print (" empno= " + str1);
  System.out.print (" firstname= " + str2);
```

```
        System.out.println("");
    }
    cursor1.close(); // 9(247 페이지 참조)

    // 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(247 페이지 참조)
      if (cursor2.endFetch()) break; // 8(247 페이지 참조)

      System.out.print (" empno= " + str1);
      System.out.print (" firstname= " + str2);
           System.out.println("");
    }
    cursor2.close(); // 9(247 페이지 참조)

    // rollback the update
    System.out.println("Rollback the update.");
    #sql { ROLLBACK work };
    System.out.println("Rollback done.");
  }
```

```
    catch( Exception e )
    {
                e.printStackTrace();
    }
  }
}
```

1. **iterator를 선언하십시오.** 이 섹션은 다음 두 유형의 iterator를 선언합니다.

   App_Cursor1

   열 자료 유형과 이름을 선언하고, 열명에 따라서 열의 값을 리턴합니다(열에 대한 명명된 바인딩).

   App_Cursor2

   열 자료 유형을 선언하고, 열 위치에 따라서 열의 값을 리턴합니다(열에 대한 위치 바인딩).
2. **iterator를 초기화하십시오.** iterator 오브젝트 cursor1은 조회 결과를 사용하여 초기화됩니다. 조회가 결과를 cursor1에 저장합니다.
3. **iterator를 다음 행으로 전진시키십시오.** cursor1.next() 메소드는 검색할 더 이상의 행이 없는 경우 부울값 `false`를 리턴합니다.
4. **자료를 이동하십시오.** 명명된 액세서 메소드 empno()가 현재 행의 empno라는 열의 값을 리턴합니다. 명명된 액세스 메소드 firstnme()는 현재 행의 firstnme라는 열의 값을 리턴합니다.
5. **자료를 호스트 변수로 SELECT하십시오.** SELECT문이 표에 있는 행 수를 호스트 변수 count1으로 전달합니다.
6. **iterator를 초기화하십시오.** iterator 오브젝트 cursor2가 조회 결과를 사용하여 초기화됩니다. 조회가 결과를 cursor2에 저장합니다.
7. **자료를 검색하십시오.** FETCH문이 결과표의 ByPos 커서에서 선언되는 첫 번째 열의 현재 값을 호스트 변수 str2로 리턴합니다.
8. **FETCH..INTO문의 성공을 체크하십시오.** endFetch() 메소드는 iterator가 행에 위치되지 않는 경우 즉 행을 페치하려는 최종 시도가 실패한 경우 부울값 true를 리턴합니다. endFetch() 메소드는 행을 페치하려는 최종 시도가 성공한 경우 false를 리턴합니다. DB2는 next() 메소드가 호출될 때 행을 페치하려고 시도합니다. FETCH...INTO문은 내재적으로 next() 메소드를 호출합니다.
9. **iterator를 닫으십시오.** close() 메소드가 iterator에 의해 보류되는 모든 자원을 해제합니다. 시스템 자원이 적시에 해제되도록 보장하기 위해 iterator를 명시적으로 닫아야 합니다.

이 예에 대한 백그라운드 정보는 Java 어플리케이션의 삽입 SQL문을 참조하십시오.

SQLJ 프로그램 컴파일 및 실행

Java(TM) 프로그램에 내장 SQLJ문이 있는 경우, 특수 프로시듀어에 따라 컴파일하고 실행해야 합니다.

주:시작하기 전에, 다음이 포함되도록 CLASSPATH를 설정하십시오.

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

- /QIBM/ProdData/Os400/Java400/ext/runtime.zip

Java용 SQL(SQLJ) 프로그램을 컴파일하고 실행하려면 다음 단계를 따르십시오.

1. 내장 SQL이 있는 Java 소스 코드에서 SQLJ 변환 프로그램 sqlj를 사용하여 Java 소스 코드 및 연관 프로파일을 생성하십시오. 각 연결에 대해 하나의 프로파일이 생성됩니다.

   예를 들어, 다음의 명령을 입력하십시오.

   ```
   sqlj MyClass.sqlj
   ```

   여기서 *MyClass.sqlj*는 SQLJ 파일의 이름입니다.

   이 예에서 SQLJ 변환 프로그램은 MyClass.java 소스 코드 파일 및 연관된 프로파일을 생성합니다. 연관된 프로파일은 MyClass_SJProfile0.ser, MyClass_SJProfile1.ser, MyClass_SJProfile2.ser 등으로 명명합니다.

   주: SQLJ 변환 프로그램은 -compile=false 절을 사용하여 컴파일 옵션을 확실하게 끄지 않으면 변환된 Java 소스 코드를 클래스 파일로 자동 컴파일합니다.
2. SQLJ 프로파일 조정자 툴인 db2profc를 사용하여 생성된 프로파일에 DB2 SQLJ 조정자를 설치하고 로컬 시스템에서 DB2 패키지를 작성하십시오.

   예를 들어, 다음 명령을 입력하십시오.

   ```
   db2profc MyClass_SJProfile0.ser
   ```

   여기서 *MyClass_SJProfile0.ser*은 DB2 SQLJ 조정자를 실행하는 프로파일의 이름입니다.

   주: 이 단계는 선택적이지만 런타임 성능을 향상시키기 위해 권장됩니다.
3. 다른 Java 클래스 파일에서와 같이 Java 클래스 파일을 실행하십시오.

   예를 들어, 다음 명령을 입력하십시오.

   ```
   java MyClass
   ```

   여기서 *MyClass*는 Java 클래스 파일의 이름입니다.

Java SQL 루틴

iSeries 서버에서는 SQL문과 프로그램에서 Java(TM) 프로그램에 액세스할 수 있습니다. Java 저장 프로시듀어와 Java UDF(user-defined function)를 사용하여 액세스할 수 있습니다. iSeries 서버는 Java 저장 프로시듀어 및 Java UDF를 호출하기 위해 DB2 및 SQLJ 규약을 모두 지원합니다. Java 저장 프로시듀어와 Java UDF는 모두 JAR 파일에 저장된 Java 클래스를 사용할 수 있습니다. iSeries 서버는 *SQLJ 파트 1* 표준에 정의된 저장 프로시듀어를 사용하여 데이터베이스로 JAR 파일을 등록합니다.

SQL문 및 프로그램에서 Java 어플리케이션에 액세스하려면 다음을 참조하십시오.

Java SQL 루틴 사용

다음의 단계에 따라 Java SQL 루틴을 사용하십시오.

- 루틴에서 Java 메소드를 작성하십시오.
- Java 클래스를 컴파일하십시오.
- 데이터베이스에서 사용하는 JVM(Java Virtual Machine)이 컴파일된 클래스에 액세스할 수 있도록 하십시오.
- 데이터베이스를 사용하여 루틴을 등록하십시오.
- Java SQL 프로시듀어를 사용하십시오.

Java 저장 프로시듀어

Java를 사용하여 저장 프로시듀어를 작성할 때 다음의 가능한 매개변수 전달 스타일을 사용할 수 있습니다.

- JAVA 매개변수 스타일
- DB2GENERAL 매개변수 스타일

Java 사용자 정의 스칼라 함수

Java 스칼라 함수는 Java 프로그램에서 하나의 값을 데이터베이스에 리턴합니다. Java 저장 프로시듀어와 같이 Java 스칼라 함수는 두 개의 매개변수 스타일 JAVA 및 DB2GENERAL 중 하나를 사용합니다.

Java 사용자 정의 표 함수

DB2는 기능에 표를 리턴할 수 있는 능력을 제공합니다. 데이터베이스 외부에서 표 형식으로 데이터베이스에 정보를 발표할 때 유용합니다.

JAR 파일을 조작하는 SQLJ 프로시듀어

Java 저장 프로시듀어와 Java UDF는 모두 Java JAR 파일에 저장된 Java 클래스를 사용할 수 있습니다. JAR 파일을 조작하는 SQLJ 프로시듀어에 대해 다음의 정보를 찾으십시오.

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

Java 저장 프로시듀어 및 UDF에 대한 매개변수 전달 규약

이 섹션에서는 Java 저장 프로시듀어 및 UDF에서 SQL 자료 유형을 표시하는 방법을 설명합니다.

Java SQL 루틴 사용

SQL문과 프로그램에서 Java(TM) 프로그램에 액세스할 수 있습니다. Java 저장 프로시듀어와 Java UDF(user-defined function)를 사용하여 액세스할 수 있습니다.

Java SQL 루틴을 사용하려면 다음의 단계를 따라야 합니다.

1. 루틴에서 Java 메소드를 작성하십시오.

   Java SQL 루틴은 SQL에서 Java 메소드를 처리합니다. 이 메소드는 DB2(R) 또는 SQLJ 매개변수 전달 규약을 사용하여 작성되어야 합니다. Java SQL 루틴이 사용하는 메소드의 코딩에 대한 자세한 정보는 Java 저장 프로시듀어, Java 사용자 정의 기능 및 Java 사용자 정의 테이블 기능을 참조하십시오.

2. Java 클래스를 컴파일하십시오.

   Java 매개변수 스타일을 사용하여 작성한 Java SQL 루틴은 추가 설정 없이 컴파일할 수 있습니다. 그러나 DB2GENERAL 매개변수 스타일을 사용하는 Java SQL 루틴은 com.ibm.db2.app.UDF 클래스나 com.ibm.db2.app.StoredProc 클래스를 확장해야 합니다. 이러한 클래스는 JAR 파일 /QIBM/ProdData/Java400/ext/db2routines_classes.jar에 들어 있습니다. javac를 사용하여 이러한 루틴을 컴파일할 때 이 JAR 파일이 CLASSPATH에 있어야 합니다. 예를 들어, 다음의 명령은 DB2GENERAL 매개변수 스타일을 사용하는 루틴이 포함된 Java 소스 파일을 컴파일합니다.

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

3. 데이터베이스가 사용하는 JVM이 컴파일된 클래스에 액세스할 수 있도록 하십시오.

   데이터베이스 JVM이 사용하는 사용자 정의 클래스는 /QIBM/UserData/OS400/SQLLib/Function 디렉토리나 데이터베이스에 등록된 JAR 파일에 있을 수 있습니다.

   /QIBM/UserData/OS400/SQLLib/Function은 DB2 UDB가 다른 플랫폼에서 Java 저장 프로시듀어 및 Java UDF를 저장하는 디렉토리인 iSeries의 /sqllib/function과 동등합니다. 클래스가 Java 패키지의 일부인 경우에는 적합한 서브디렉토리에 있습니다. 예를 들어, runit 클래스를 foo.bar 패키지의 일부로 작성한 경우에는 파일 runnit.class가 통합 파일 시스템 디렉토리 /QIBM/ProdData/OS400/SQLLib/Function/foo/bar에 있어야 합니다.

   클래스 파일은 데이터베이스에 등록된 JAR 파일에도 있을 수 있습니다. JAR 파일은 SQLJ.INSTALL_JAR 저장 프로시듀어를 사용하여 등록합니다. 이 저장 프로시듀어는 JAR 파일에 JAR ID를 할당할 때 사용합니다. 이 JAR ID를 사용하여 클래스 파일이 상주하는 JAR 파일을 식별합니다. SQLJ.INSTALL_JAR 및 JAR 파일을 조작하기 위한 다른 저장 프로시듀어에 대한 자세한 정보는 JAR 파일을 조작하는 SQLJ 프로시듀어를 참조하십시오.

4. 데이터베이스를 사용하여 루틴을 등록하십시오.

   Java SQL 루틴은 CREATE PROCEDURE 및 CREATE FUNCTION SQL문을 사용하여 데이터베이스로 등록합니다. 이러한 명령문에는 다음의 요소가 들어 있습니다.

CREATE 키워드

Java SQL을 작성하기 위한 SQL문은 CREATE PROCEDURE 또는 CREATE STATEMENT로 시작합니다.

루틴 이름

SQL문은 데이터베이스에 알려진 루틴의 이름을 식별합니다. 이 이름은 SQL에서 Java 루틴에 액세스하는 데 사용하는 이름입니다.

매개변수 및 리턴값

SQL문은 적용 가능한 경우에 Java 루틴에 대한 매개변수와 리턴값을 식별합니다.

LANGUAGE JAVA

SQL문은 키워드 LANGUAGE JAVA를 사용하여 루틴이 Java로 기록되었음을 나타냅니다.

PARAMETER STYLE KEYWORDS

SQL문은 키워드 PARAMETER STYLE JAVA 또는 PARAMETER STYLE DB2GENERAL을 사용하여 매개변수 스타일을 식별합니다.

외부 이름

SQL문은 Java SQL 루틴으로 처리할 Java 메소드를 식별합니다. 외부 이름의 형식은 다음의 두 형식 중 하나입니다.

- 메소드가 /QIBM/UserData/OS400/SQLLib/Function 디렉토리 하에 위치한 클래스 파일에 있으면 형식 *classname.methodname*을 사용하여 메소드를 식별합니다. 여기서 *classname*은 클래스의 완전한 규정명이고 *methodname*은 메소드의 이름입니다.
- 메소드가 데이터베이스에 등록된 JAR 파일에 있으면 형식 *jarid*:*classname.methodname*을 사용하여 메소드를 식별합니다. 여기서 *jarid*는 등록된 JAR 파일의 JAR ID이며 *classname*은 클래스의 이름이고 *methodname*은 메소드의 이름입니다.

iSeries Navigator를 사용하면 Java 매개변수 스타일을 사용하는 저장 프로시듀어 또는 사용자 정의 기능을 작성할 수 있습니다.

5. Java 프로시듀어를 사용하십시오.

   Java 저장 프로시듀어는 SQL CALL문을 사용하여 호출합니다. Java UDF는 다른 SQL문의 일부로 호출하는 기능입니다.

Java 저장 프로시듀어

Java(TM)를 사용하여 저장 프로시듀어를 사용할 경우, 다음 2개의 매개변수 전달 스타일을 사용할 수 있습니다. 권장하는 스타일은 JAVA 매개변수 스타일이며 SQLj: SQL 루틴 표준에 지정된 매개변수 스타일과 일치합니다. 두 번째 스타일인 DB2GENERAL은 DB2(R) UDB가 정의한 매개변수 스타일입니다. 매개변수 스타일은 Java 저장 프로시듀어를 코딩할 때 사용해야 하는 규약을 판별하기도 합니다.

또한 Java 저장 프로시듀어에 있는 일부 제한사항도 알아야 합니다.

JAVA 매개변수 스타일:

Java 매개변수 스타일을 사용하는 Java(TM) 저장 프로시듀어를 코딩할 경우, 다음 규약을 사용해야 합니다.

- Java 메소드는 public void static(instance가 아님) 메소드이어야 합니다.
- Java 메소드의 매개변수는 SQL 호환 가능 유형이어야 합니다.
- Java 메소드는 매개변수가 널(null) 허용 유형(String과 같음)일 때 SQL NULL 값에 대해 테스트할 수 있습니다.
- 단일 요소 배열을 사용하여 출력 매개변수를 리턴합니다.
- Java 메소드는 getConnection 메소드를 사용하여 현재 데이터베이스에 액세스할 수 있습니다.

JAVA 매개변수 스타일을 사용하는 Java 저장 프로시듀어는 public static 메소드입니다. 클래스 내에서 저장 프로시듀어는 메소드명과 서명으로 식별합니다. 저장 프로시듀어를 호출하면 CREATE PROCEDURE문에 의해 정의된 변수 유형에 기초하여 서명이 동적으로 생성됩니다.

널값을 허용하는 Java 유형에 매개변수를 전달하면 Java 메소드는 이 매개변수를 널과 비교하여 입력 매개변수가 SQL NULL인지 판별할 수 있습니다.

다음의 Java 유형은 널값을 지원하지 않습니다.

- short
- int
- long
- float
- double

널값을 지원하지 않는 Java 유형에 널값을 전달하면 SQL 예외가 오류 코드 -20205와 함께 리턴됩니다.

출력 매개변수는 하나의 요소가 있는 배열로 전달됩니다. Java 저장 프로시듀어는 배열의 첫 번째 요소를 설정하여 출력 매개변수를 설정할 수 있습니다.

다음의 JDBC(Java Database Connectivity) 호출을 사용하여 내장 어플리케이션 문맥에 대한 연결에 액세스합니다.

```
connection=DriverManager.getConnection("jdbc:default:connection");
```

그러면 이 연결은 JDBC API를 사용하여 SQL문을 실행합니다.

다음은 한 번의 입력과 두 번의 출력을 사용하는 작은 저장 프로시듀어입니다. 제공된 SQL 조회를 실행하고 결과 및 SQLSTATE에 모두 행 수를 리턴합니다.

예: 한 번의 입력과 두 번의 출력을 사용하는 저장 프로시듀어

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
   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();
      }
     }
    }
```

SQLj 표준에서 JAVA 매개변수 스타일을 사용하는 루틴으로 결과 세트를 리턴하려면 결과 세트를 명시적으로 설정해야 합니다. 결과 세트를 리턴하는 프로시듀어를 작성할 때 매개변수 리스트의 끝에 추가 결과 세트 매개변수가 추가됩니다. 예를 들어, 다음의 명령문은

```
CREATE PROCEDURE RETURNTWO()
DYNAMIC RESULT SETS 2
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'javaClass!returnTwoResultSets'
```

`public static void returnTwoResultSets(ResultSet[] rs1, ResultSet[] rs2)` 서명을 사용하여 Java 메소드를 호출합니다.

다음의 예에 표시된 대로 결과 세트의 출력 매개변수를 명시적으로 설정해야 합니다. DB2GENERAL 스타일에서와 마찬가지로 결과 세트와 해당 명령문을 닫지 말아야 합니다.

예: 두 개의 결과 세트를 리턴하는 저장 프로시듀어

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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);
  }
}
```

서버에서는 결과 세트의 순서를 판별하기 위해 추가 결과 세트 매개변수를 조사하지 않습니다. 서버의 결과 세트는 열린 순서대로 리턴됩니다. SQLj 표준과의 호환성을 보장하기 위해 앞에 표시된 바와 같이 열린 순서대로 결과를 지정해야 합니다.

DB2GENERAL 매개변수 스타일:

DB2GENERAL 매개변수 양식을 사용하는 Java(TM) 저장 프로시듀어를 코딩할 때, 다음 규약을 사용해야 합니다.

- Java 저장 프로시듀어를 정의하는 클래스는 Java com.ibm.db2.app.StoredProc 클래스를 *확장*하는 클래스이거나 이 클래스의 서브클래스이어야 합니다.
- Java 메소드는 public void instance 메소드이어야 합니다.
- Java 메소드의 매개변수는 SQL 호환 가능 유형이어야 합니다.
- Java 메소드는 isNull 메소드를 사용하여 SQL NULL 값에 대해 테스트할 수 있습니다.
- Java 메소드는 set 메소드를 사용하여 리턴 매개변수를 명시적으로 설정해야 합니다.
- Java 메소드는 getConnection 메소드를 사용하여 현재 데이터베이스에 액세스할 수 있습니다.

Java 저장 프로시듀어가 포함된 클래스는 클래스 com.ibm.db2.app.StoredProc을 확장해야 합니다. Java 저장 프로시듀어는 public instance 메소드입니다. 클래스 내에서 저장 프로시듀어는 메소드명과 서명으로 식별합니다. 저장 프로시듀어를 호출하면 CREATE PROCEDURE문에 의해 정의된 변수 유형에 기초하여 서명이 동적으로 생성됩니다.

com.ibm.db2.app.StoredProc 클래스는 Java 메소드가 입력 매개변수가 SQL NULL인지 판별할 수 있게 하는 isNull 메소드를 제공합니다. com.ibm.db2.app.StoredProc 클래스는 출력 매개변수를 설정하는 set...( ) 메소드도 제공합니다. 이러한 메소드를 사용하여 출력 매개변수를 설정해야 합니다. 출력 매개변수를 설정하지 않으면 출력 매개변수는 SQL NULL 값을 리턴합니다.

com.ibm.db2.app.StoredProc 클래스는 내장 어플리케이션 문맥에 대한 JDBC 연결을 페치할 수 있도록 다음의 루틴을 제공합니다. 내장 어플리케이션 문맥에 대한 연결에는 다음의 JDBC 호출을 사용하여 액세스합니다.

```
public Java.sql.Connection getConnection( )
```

그러면 이 연결은 JDBC API를 사용하여 SQL문을 실행합니다.

다음은 한 번의 입력과 두 번의 출력을 사용하는 작은 저장 프로시듀어입니다. 제공된 SQL 조회를 처리하고 결과 및 SQLSTATE에 모두 행 수를 리턴합니다.

예: 한 번의 입력과 두 번의 출력을 사용하는 저장 프로시듀어

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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());
    }
  }
}
```

DB2GENERAL 매개변수 스타일을 사용하는 프로시듀어에서 결과 세트를 리턴하려면 프로시듀어가 종료되어도 결과 세트 및 응답 명령문은 계속 열려 있어야 합니다. 리턴되는 결과 세트는 클라이언트 어플리케이션이 닫아야 합니다. 여러 결과 세트가 리턴되는 경우에는 열린 순서대로 리턴됩니다. 예를 들어, 다음의 저장 프로시듀어는 두 개의 결과 세트를 리턴합니다.

예: 두 개의 결과 세트를 리턴하는 저장 프로시듀어

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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 저장 프로시듀어 제한사항:

다음 제한사항은 Java(TM) 저장 프로시듀어에 적용됩니다.

- Java 저장 프로시듀어는 추가 스레드를 작성하지 말아야 합니다. 작업이 복수 스레드를 허용하는 경우에만 작업에서 추가 스레드를 작성할 수 있습니다. SQL 저장 프로시듀어를 호출하는 작업이 복수 스레드를 허용한다는 보장이 없기 때문에 Java 저장 프로시듀어가 추가 스레드를 작성해서는 안됩니다.
- Java 클래스 파일에 액세스하기 위해 허용된 권한을 사용할 수 없습니다.
- Java 저장 프로시듀어는 항상 시스템에 설치된 Java Development의 최신 버전을 사용합니다.
- Blob 및 Clob 클래스는 java.sql 및 com.ibm.db2.app 패키지에 모두 상주하기 때문에 두 클래스를 동일한 프로그램에서 사용할 경우 프로그래머는 이들 클래스의 전체 이름을 사용해야 합니다. 프로그램은 com.ibm.db2.app에서 Blob 및 Clob 클래스를 저장 프로시듀어에 전달되는 매개변수로 사용하는지 확인해야 합니다.
- Java 저장 프로시듀어를 작성할 때 시스템은 라이브러리에 서비스 프로그램을 생성합니다. 이 서비스 프로그램은 프로시듀어 정의를 저장하는데 사용합니다. 서비스 프로그램에는 시스템이 생성한 이름이 있습니다. 이 이름은 저장 프로시듀어를 작성한 작업의 작업 기록부를 조사하여 확보할 수 있습니다. 서비스 프로그램 오브젝트를 저장한 후에 복원하면 프로시듀어 정의가 복원됩니다. Java 저장 프로시듀어를 한 시스템에서 다른 시스템으로 이동시킬 경우에 사용자는 Java 클래스가 있는 통합 파일 시스템과 함께 프로시듀어 정의가 있는 프로그램을 이동시킬 책임이 있습니다.
- Java 저장 프로시듀어는 데이터베이스에 연결하는데 사용하는 JDBC 연결의 등록 정보(예: 시스템 명명)를 설정할 수 없습니다. 프리페치를 사용할 수 없는 경우를 제외하고는 디폴트 JDBC 연결 등록 정보가 항상 사용됩니다.

Java 사용자 정의 스칼라 함수

Java(TM) 스칼라 함수는 Java 프로그램에서 데이터베이스로 값을 리턴합니다. 예를 들어, 두 수의 합계를 리턴하는 스칼라 함수를 작성할 수 있습니다. Java 저장 프로시듀어와 마찬가지로 Java 스칼라 함수는 두 개의 매개변수 스타일『매개변수 스타일 Java』및 258 페이지의『매개변수 스타일 DB2GENERAL』중 하나를 사용합니다. Java 사용자 정의 기능(UDF)을 코딩할 때에는 Java 스칼라 함수 작성에 대한 제한사항을 알고 있어야 합니다.

매개변수 스타일 Java: Java 매개변수 스타일은 *SQLJ 파트 1: SQL 루틴* 표준이 지정한 스타일입니다. Java UDF를 코딩할 때 다음의 규약을 사용하십시오.

- Java 메소드는 public static 메소드이어야 합니다.
- Java 메소드는 SQL 호환가능한 유형을 리턴해야 합니다. 리턴 값은 메소드의 결과입니다.
- Java 메소드의 매개변수는 SQL 호환가능한 유형이어야 합니다.
- Java 메소드는 널값을 허용하는 Java 유형에 대한 SQL NULL을 테스트할 수 있습니다.

예를 들어, INTEGER를 리턴하고 유형이 CHAR(5), BLOB(10K) 및 DATE인 인수를 사용하는 sample!test3이라고 하는 UDF가 제공되었으면 DB2는 UDF의 Java 구현에 다음의 서명이 있는 것으로 예상합니다.

```
import com.ibm.db2.app.*;
public class sample {
   public static int test3(String arg1, Blob arg2, Date arg3) { ... }
}
```

Java 메소드의 매개변수는 SQL 호환가능한 유형이어야 합니다. 예를 들어, SQL 유형 t1, t2 및 t3인 인수를 사용하고 유형 t4를 리턴하여 UDF를 선언한 경우에 예상 Java 서명을 사용하여 Java 메소드로 호출합니다.

```
public static T4 name (T1 a, T2 b, T3 c)  { .....}
```

여기서,

- *name*은 메소드명입니다.
- T1부터 T4까지는 SQL 유형 t1부터 t4까지에 해당하는 Java 유형입니다.
- *a*, *b* 및 *c*는 입력 인수에 대한 임의의 변수명입니다.

SQL 유형과 Java 유형 사이의 상관 관계는 저장 프로시듀어 및 UDF에 대한 매개변수 전달 규약에서 찾을 수 있습니다.

SQL NULL 값은 초기화되지 않은 Java 변수로 표시합니다. 이러한 변수에는 오브젝트 유형이 아닌 경우에 Java 널값이 있습니다. int와 같은 Java 스칼라 자료 유형에 SQL NULL이 전달된 경우 예외 상태가 발생합니다.

JAVA 매개변수 스타일을 사용할 때 Java UDF에서 결과를 리턴하려면 간단하게 메소드에서 결과를 리턴하십시오.

```
{ ....
  return value;
}
```

UDF 및 저장 프로시듀어에서 사용한 C 모듈과 같이 Java UDF에서 Java 표준 I/O 스트림(System.in, System.out 및 System.err)을 사용할 수 없습니다.

매개변수 스타일 DB2GENERAL: 매개변수 스타일 DB2GENERAL은 Java UDF가 사용합니다. 이 매개변수 스타일에서 리턴 값은 기능의 마지막 매개변수로 전달되며 com.ibm.db2.app.UDF 클래스의 *set* 메소드를 사용하여 설정해야 합니다.

Java UDF를 코딩할 때에는 다음의 규약을 준수해야 합니다.

- Java UDF가 포함된 클래스는 Java com.ibm.db2.app.UDF 클래스를 *확장*하는 것이거나 이 클래스의 서브클래스이어야 합니다.
- DB2GENERAL 매개변수 스타일의 경우 Java 메소드는 public void instance 메소드이어야 합니다.
- Java 메소드의 매개변수는 SQL 호환 가능 유형이어야 합니다.
- isNull 메소드를 사용하여 SQL NULL 값에 대해 Java 메소드를 테스트할 수 있습니다.
- DB2GENERAL 매개변수 스타일의 경우 Java 메소드는 set() 메소드를 사용하여 리턴 매개변수를 명시적으로 설정해야 합니다.

Java UDF가 포함된 클래스는 Java 클래스 com.ibm.db2.app.UDF를 확장해야 합니다. DB2GENERAL 매개변수 스타일을 사용하는 Java UDF는 Java 클래스의 void instance 메소드이어야 합니다. 예를 들어, INTEGER를 리턴하고 유형이 CHAR(5), BLOB(10K) 및 DATE인 인수를 사용하는 sample!test3이라고 하는 UDF의 경우에 DB2는 UDF의 Java 구현에 다음의 서명이 있는 것으로 예상합니다.

```
import com.ibm.db2.app.*;
public class sample extends UDF {
   public void test3(String arg1, Blob arg2, String arg3, int result) { ... }
}
```

Java 메소드의 매개변수는 SQL 유형이어야 합니다. 예를 들어, SQL 유형 t1, t2 및 t3인 인수를 사용하고 유형 t4를 리턴하여 UDF를 선언한 경우에 예상 Java 서명을 사용하여 Java 메소드로 호출합니다.

```
public void name (T1 a, T2 b, T3 c, T4 d)  { .....}
```

여기서,

- *name*은 메소드명입니다.
- T1부터 T4까지는 SQL 유형 t1부터 t4까지에 해당하는 Java 유형입니다.
- *a*, *b* 및 *c*는 입력 인수에 대한 임의의 변수명입니다.
- *d*는 계산된 UDF 결과를 나타내는 임의의 변수명입니다.

SQL 유형과 Java 유형 사이의 상관 관계는 저장 프로시듀어 및 UDF에 대한 매개변수 전달 규약 섹션에 있습니다.

SQL NULL 값은 초기화되지 않은 Java 변수로 표시합니다. Java 규칙에 따르면 이러한 변수가 기본 유형인 경우에는 값이 0이고 오브젝트 유형인 경우에는 값이 Java 널(null)입니다. 일반 0이 아닌 SQL NULL을 알리기 위해 입력 인수에 대해 isNull 메소드를 호출할 수 있습니다.

```
   { ....
if (isNull(1)) { /* argument #1 was a SQL NULL */ }
else           { /* not NULL */ }
   }
```

앞의 예에서 인수는 1부터 시작합니다. 계속되는 다른 기능과 마찬가지로 isNull() 기능은 com.ibm.db2.app.UDF 클래스에서 상속된 것입니다. DB2GENERAL 매개변수 스타일을 사용할 때 Java UDF에서 결과를 리턴하려면 다음과 같이 UDF에서 set() 메소드를 사용하십시오.

```
   { ....
set(2, value);
   }
```

여기서 2는 출력 인수의 지수이고 *value*는 호환가능한 유형의 리터럴 또는 변수입니다. 인수는 선택된 출력의 인수 리스트에 있는 지수입니다. 이 절의 첫 번째 예에서 int 결과 변수에는 지수 4가 있습니다. UDF가 리턴하기 전에 설정되지 않은 출력 인수의 값은 NULL입니다.

UDF 및 저장 프로시듀어에서 사용한 C 모듈과 같이 Java UDF에서 Java 표준 I/O 스트림(System.in, System.out 및 System.err)을 사용할 수 없습니다.

일반적으로 DB2는 조회에 있는 각 입력 또는 결과 세트 행에 대해 한 번씩 UDF를 여러 번 호출합니다. UDF의 CREATE FUNCTION문에 SCRATCHPAD를 지정한 경우 DB2는 UDF에 대한 연속 호출 사이에 일종의 "연결성"이 필요한 것으로 인식하기 때문에 DB2GENERAL 매개변수 스타일 기능의 경우 Java 클래스 구현은 각 호출에 대해 인스턴스화되지 않지만 일반적으로 명령문당 각 UDF 참조에 대해 한 번씩 인스턴스화됩니다. 그러나 UDF에 대해 NO SCRATCHPAD를 지정한 경우 클래스 구성자를 호출하여 UDF에 대한 각 호출마다 명확한 인스턴스를 인스턴스화하지 않습니다.

UDF에 대한 여러 호출을 통해 정보를 저장하는데 스크래치패드가 유용할 수 있습니다. Java UDF는 인스턴스 변수를 사용하거나 스크래치패드를 설정하여 호출들 간의 연속성을 설정할 수 있습니다. Java UDF는 com.ibm.db2.app.UDF에서 사용할 수 있는 getScratchPad 및 setScratchPad 메소드를 사용하여 스크래치패드에 액세스합니다. 조회 끝에 CREATE FUNCTION문에 FINAL CALL 옵션을 지정하면 오브젝트의 public void close() 메소드를 호출합니다(DB2GENERAL 매개변수 스타일 기능의 경우). 이 메소드를 정의하지 않으면 스터브 기능이 이를 인계하며 이벤트는 무시합니다. com.ibm.db2.app.UDF 클래스에는 DB2GENERAL 매개변수 UDF 내에서 사용할 수 있는 유용한 변수와 메소드가 들어 있습니다. 이러한 변수와 메소드는 다음의 표에서 설명합니다.

변수 및 메소드	설명
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;** «	스칼라 UDF의 경우 호출이 첫 번째 호출인지 아니면 정상 호출인지를 판별하기 위한 상수입니다. 표 UDF의 경우 호출이 첫 번째 호출, 열기 호출, 페치 호출, 닫기 호출 또는 최종 호출인지를 판별하기 위한 상수입니다.
public Connection getConnection();	메소드는 이 저장 프로시듀어 호출에 대한 JDBC 연결 핸들을 확보하고, 호출되는 어플리케이션의 데이터베이스 연결을 나타내는 JDBC 오브젝트를 리턴합니다. C 저장 프로시듀어에서의 널(null)SQLConnect() 호출 결과와 유사합니다.
public void close();	FINAL CALL 옵션을 사용하여 UDF를 작성한 경우에 UDF 평가 끝에 데이터베이스가 이 메소드를 호출합니다. C UDF에 대한 최종 호출과 유사합니다. Java UDF 클래스가 이 메소드를 구현하지 않으면 이 이벤트는 무시합니다.
public boolean isNull(int i)	이 메소드는 제공된 지수가 있는 입력 인수가 SQL NULL인지의 여부를 테스트합니다.
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);**	이러한 메소드는 출력 인수를 제공된 값으로 설정합니다. 다음을 포함하여 뭔가 잘못된 경우에 예외가 발생합니다. • UDF 호출이 진행되고 있지 않습니다. • 지수가 유효한 출력 인수를 참조하지 않습니다. • 자료 유형이 일치하지 않습니다. • 자료 길이가 일치하지 않습니다. • 코드 페이지 변환 오류가 발생했습니다.
public void setSQLstate(String string);	이 호출에서 리턴하도록 SQLSTATE를 설정하기 위해 이 메소드를 UDF에서 호출할 수 있습니다. 스트링을 SQLSTATE로 허용할 수 없는 경우 예외가 발생합니다. 사용자는 기능에서 오류 또는 경고를 리턴하도록 외부 프로그램에서 SQLSTATE를 설정할 수 있습니다. 이 경우에 SQLSTATE에는 다음 중 하나가 포함되어야 합니다. • 성공을 나타내는 '00000' • 경고를 나타내는 '01Hxx'(여기서 xx는 두 자리 숫자이거나 대문자 글자) • 오류를 나타내는 '38yxx'(여기서 y는 'I'와 'Z' 사이의 대문자 글자이고 xx는 두 자리 숫자이거나 대문자 글자)
public void setSQLmessage(String string);	이 메소드는 setSQLstate 메소드와 유사합니다. SQL 메세지 결과를 설정합니다. 스트링을 허용할 수 없는 경우(예를 들어, 70자보다 긴 스트링) 예외가 발생합니다.
public String getFunctionName();	이 메소드는 처리 중인 UDF의 이름을 리턴합니다.
public String getSpecificName();	이 메소드는 처리 중인 UDF의 특정 이름을 리턴합니다.

변수 및 메소드	설명
public byte[] getDBinfo();	이 메소드는 처리 중인 UDF에 대해 처리되지 않은 원래의 DBINFO 구조를 바이트 배열로 리턴합니다. DBINFO 옵션을 사용하여 UDF를 등록해야 합니다(CREATE FUNCTION 사용).
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();**	이러한 메소드는 처리 중인 UDF의 DBINFO 구조에서 적합한 필드의 값을 리턴합니다. DBINFO 옵션을 사용하여 UDF를 등록해야 합니다(CREATE FUNCTION 사용). UPDATE문에서 SET 절의 오른쪽에 사용자 정의 기능을 지정한 경우에만 getDBtbschema(), getDBtbname() 및 getDBcolname() 메소드는 의미있는 정보를 리턴합니다.
public int getCCSID();	이 메소드는 작업의 CCSID를 리턴합니다.
public byte[] getScratchpad();	이 메소드는 현재 처리 중인 UDF의 스크래치패드 사본을 리턴합니다. 먼저 SCRATCHPAD 옵션을 사용하여 UDF를 선언해야 합니다.
public void setScratchpad(byte ab[]);	이 메소드는 현재 처리 중인 UDF의 스크래치패드를 제공된 바이트 배열의 내용으로 겹쳐씁니다. 먼저 SCRATCHPAD 옵션을 사용하여 UDF를 선언해야 합니다. 바이트 배열의 *크기*는 getScratchpad()가 리턴하는 *크기*와 동일해야 합니다.
public int getCallType();	이 메소드는 현재 작성 중인 호출의 유형을 리턴합니다. 이러한 값은 sqludf.h에 정의된 C 값과 일치합니다. 가능한 리턴 값은 다음과 같습니다. • SQLUDF_FIRST_CALL • SQLUDF_NORMAL_CALL •  SQLUDF_TF_FIRST • SQLUDF_TF_OPEN • SQLUDF_TF_FETCH • SQLUDF_TF_CLOSE • SQLUDF_TF_FINAL 

Java 사용자 정의 기능 제한사항:

다음 제한사항은 Java(TM) 사용자 정의 기능(UDF)에 적용됩니다.

- Java UDF는 추가 스레드를 작성하지 말아야 합니다. 작업이 복수 스레드를 허용하는 경우에만 작업에서 추가 스레드를 작성할 수 있습니다. SQL 저장 프로시듀어를 호출하는 작업이 복수 스레드를 허용한다는 보장이 없기 때문에 Java 저장 프로시듀어가 추가 스레드를 작성해서는 안됩니다.
- 데이터베이스에 정의된 Java 저장 프로시듀어의 전체 이름은 279자로 제한됩니다. 이 한계는 최대 너비가 279자인 EXTERNAL_NAME 열 때문입니다.
- 허용된 권한을 Java 클래스 파일에 액세스하는 데 사용할 수 없습니다.
- Java UDF는 항상 시스템에 설치된 JDK의 최신 버전을 사용합니다.
- Blob 및 Clob 클래스는 java.sql 및 com.ibm.db2.app 패키지에 모두 상주하기 때문에 두 클래스를 동일한 프로그램에서 사용할 경우 프로그래머는 이들 클래스의 전체 이름을 사용해야 합니다. 프로그램은 com.ibm.db2.app에서 Blob 및 Clob 클래스를 저장 프로시듀어에 전달되는 매개변수로 사용하는지 확인해야 합니다.
- 소스가 있는 기능과 같이 Java UDF가 작성되면 라이브러리의 서비스 프로그램을 사용하여 기능 정의를 저장합니다. 서비스 프로그램의 이름은 시스템에 의해 생성되며 기능을 작성한 작업의 작업 기록부에서 찾을 수 있습니다. 이 오브젝트를 저장한 후에 다른 시스템으로 복원하면 기능 정의가 복원됩니다. Java UDF를 한 시스템에서 다른 시스템으로 이동시킬 경우에 사용자는 Java 클래스가 있는 통합 파일 시스템과 함께 기능 정의가 있는 서비스 프로그램을 이동시킬 책임이 있습니다.
- Java UDF는 데이터베이스에 연결하는데 사용하는 JDBC 연결의 등록 정보(예: 시스템 명명)를 설정할 수 없습니다. 디폴트 JDBC 연결 등록 정보가 항상 사용되며, 프리페치가 불가능할 경우는 예외입니다.

Java 사용자 정의 표 기능:

DB2는 기능에 표를 리턴할 수 있는 능력을 제공합니다. 데이터베이스 외부에서 표 형식으로 데이터베이스에 정보를 발표할 때 유용합니다. 예를 들어, Java 저장 프로시듀어 및 Java UDF(표와 스칼라 모두)에 사용되는 JVM(Java(TM) virtual machine)에 설정한 등록 정보를 나타내는 표를 작성할 수 있습니다.

SQLJ 파트 1: SQL 루틴 표준은 표 기능을 지원하지 않습니다. 결과적으로 매개변수 스타일 DB2GENERAL을 사용해야만 표 기능을 사용할 수 있습니다.

표 기능에 대해 다섯 가지 유형의 호출이 작성됩니다. 다음의 표는 이러한 호출을 설명한 것입니다. 여기에서는 create function SQL문에 스크래치패드가 지정된 것으로 가정합니다.

적시 스캔 시간	**NO FINAL CALL LANGUAGE JAVA SCRATCHPAD**	**FINAL CALL LANGUAGE JAVA SCRATCHPAD**
표 기능의 첫 번째 OPEN 이전	호출 없음	클래스 구성자를 호출합니다(새로운 스크래치패드를 의미). FIRST 호출과 함께 UDF 메소드를 호출합니다.

적시 스캔 시간	**NO FINAL CALL LANGUAGE JAVA SCRATCHPAD**	**FINAL CALL LANGUAGE JAVA SCRATCHPAD**
표 기능을 OPEN할 때마다	클래스 구성자를 호출합니다(새로운 스크래치패드를 의미). OPEN 호출과 함께 UDF 메소드를 호출합니다.	OPEN 호출과 함께 UDF 메소드를 호출합니다.
새로운 표 기능 자료 행을 FETCH할 때마다	FETCH 호출과 함께 UDF 메소드를 호출합니다.	FETCH 호출과 함께 UDF 메소드를 호출합니다.
표 기능을 CLOSE할 때마다	CLOSE 호출과 함께 UDF 메소드를 호출합니다. close() 메소드가 있으면 이 메소드도 호출합니다.	CLOSE 호출과 함께 UDF 메소드를 호출합니다.
표 기능의 마지막 CLOSE 이후	호출 없음	FINAL 호출과 함께 UDF 메소드를 호출합니다. close() 메소드가 있으면 이 메소드도 호출합니다.

예: Java 표 기능: 다음은 Java 사용자 정의 표 기능을 실행하기 위해 사용하는 JVM에서 설정된 등록 정보를 판별하는 Java 표 기능의 예입니다.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
     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);
        }
      }
    }
```

표 기능을 컴파일하고 클래스 파일을 /QIBM/UserData/OS400/SQLLib/Function에 복사한 후에 다음의 SQL 문을 사용하여 데이터베이스에 기능을 등록할 수 있습니다.

```
    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
```

기능을 등록한 후에 SQL문의 일부로 사용할 수 있습니다. 예를 들어, 다음의 SELECT문은 표 기능으로 생성된 표를 리턴합니다.

```
    SELECT * FROM TABLE(PROPERTIES())
```

JAR 파일을 조작하는 SQLJ 프로시듀어

Java(TM) 저장 프로시듀어와 Java UDF 모두는 Java JAR 파일에 저장된 Java 클래스를 사용할 수 있습니다. JAR 파일을 사용하려면 *jar-id*를 JAR 파일과 연관시켜야 합니다. 시스템은 SQLJ 스키마에 저장 프로시듀어를 제공하므로 *jar-id* 및 JAR 파일을 조작할 수 있습니다. 이러한 프로시듀어를 통해 JAR 파일을 설치, 대체 및 제거할 수 있습니다. 또한 JAR 파일과 연관된 SQL 카탈로그를 사용하고 갱신하는 능력을 제공합니다.

자세한 정보는 다음 주제를 참조하십시오.

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

SQLJ.INSTALL_JAR:

SQLJ.INSTALL_JAR 저장 프로시듀어는 JAR 파일을 데이터베이스 시스템에 설치합니다. 이 JAR 파일은 후속 CREATE FUNCTION 및 CREATE PROCEDURE 명령문에서 사용할 수 있습니다.

권한 부여: CALL문의 권한 부여 ID가 보유한 권한에는 SYSJAROBJECTS 및 SYSJARCONTENTS 카탈로그 표에 대해 최소한 다음 중 하나가 포함되어야 합니다.

- 다음의 시스템 권한:
  - 표에 대한 INSERT 및 SELECT 권한
  - 라이브러리 QSYS2에 대한 시스템 권한 *EXECUTE
- 관리 권한

CALL문의 권한 부여 ID가 보유한 권한에는 다음의 권한도 있어야 합니다.

- 설치되는 *jar-url* 매개변수에 지정된 JAR 파일에 대한 읽기(*R) 액세스.
- JAR 파일을 설치하는 디렉토리에 대한 쓰기, 실행 및 읽기(*RWX) 액세스. 이 디렉토리는 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*이며 여기서 *schema*는 *jar-id*의 스키마입니다.

허용된 권한을 이러한 권한으로 사용할 수 없습니다.

SQL 구문:

```
>>-CALL--SQLJ.INSTALL_JAR-- (--'jar-url'--,--'jar-id'--,--deploy--)-->

>-------------------------------------------------------------><
```

설명:

jar-url 설치하거나 대체할 JAR 파일이 포함된 URL. 지원되는 URL 체계는 'file:' 뿐입니다.

jar-id *jar-url*에 지정된 파일과 연관시킬 데이터베이스의 JAR ID. *jar-id*는 SQL 명명을 사용하며 JAR 파일을 내재적 또는 명시적 규정자에 지정된 스키마나 라이브러리에 설치합니다.

deploy 배치 설명자 파일의 install_action을 설명하는 데 사용하는 값. 이 정수가 0이 아닌 값이면 배치 설명자 파일의 install_actions를 install_jar 프로시듀어의 끝에 실행해야 합니다. iSeries용 DB2 UDB의 현재 버전은 값 0만을 지원합니다.

사용법 주의사항: JAR 파일을 설치할 때 iSeries용 DB2 UDB는 SYSJAROBJECTS 시스템 카탈로그에 JAR 파일을 등록합니다. 또한 JAR 파일에서 Java(TM) 클래스 파일의 이름을 추출하고 SYSJARCONTENTS 시스템 카탈로그에 각 클래스를 등록합니다. iSeries용 DB2 UDB는 JAR 파일을 /QIBM/UserData/OS400/SQLLib/Function 디렉토리의 jar/schema 서브디렉토리에 복사합니다. iSeries용 DB2 UDB는 JAR 파일의 새로운 사본에 *jar-id* 절에 제공된 이름을 부여합니다. iSeries용 DB2 UDB가 /QIBM/UserData/OS400/SQLLib/Function/jar의 서브디렉토리에 설치한 JAR 파일은 변경해서는 안됩니다. 그 대신 CALL SQLJ.REMOVE_JAR 및 CALL SQLJ.REPLACE_JAR SQL 명령을 사용하여 설치한 JAR 파일을 제거하거나 대체해야 합니다.

예: 다음의 명령은 SQL 대화식 세션에서 발행합니다.

```
CALL SQLJ.INSTALL_JAR('file:/home/db2inst/classes/Proc.jar', 'myproc_jar', 0)
```

file:/home/db2inst/classes/ 디렉토리에 있는 Proc.jar 파일은 myproc_jar 이름을 사용하여 iSeries용 DB2 UDB에 설치합니다. Procedure.jar 파일을 사용하는 후속 SQL 명령은 이름 myproc_jar를 사용하여 이 파일을 참조합

니다.

«

SQLJ.REMOVE_JAR:

»

SQLJ.REMOVE_JAR 저장 프로시듀어는 데이터베이스 시스템에서 JAR 파일을 제거합니다.

권한부여: CALL문의 권한 부여 ID가 보유한 권한에는 SYSJARCONTENTS 및 SYSJAROBJECTS 카탈로그 표에 대해 최소한 다음 중 하나가 포함되어야 합니다.

- 다음의 시스템 권한:
  - 표에 대한 SELECT 및 DELETE 권한
  - 라이브러리 QSYS2에 대한 시스템 권한 *EXECUTE
- 관리 권한

CALL문의 권한 부여 ID가 보유한 권한에는 다음의 권한도 있어야 합니다.

- 제거하는 JAR 파일에 대한 *OBJMGT 권한. JAR 파일은 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile로 명명합니다.

허용된 권한을 이러한 권한으로 사용할 수 없습니다.

구문:

```
>>-CALL--SQLJ.REMOVE_JAR--(--'jar-id'--,--undeploy--)----------><
```

설명:

jar-id 데이터베이스에서 제거될 JAR 파일의 JAR ID.

undeploy

배치 설명자 파일의 remove_action을 설명하는 데 사용하는 값. 이 정수가 0이 아닌 값이면 배치 설명자 파일의 remove_actions를 install_jar 프로시듀어의 끝에 실행해야 합니다. iSeries용 DB2 UDB의 현재 버전은 값 0만을 지원합니다.

예: 다음의 명령은 SQL 대화식 세션에서 발행합니다.

```
CALL SQLJ.REMOVE_JAR('myProc_jar', 0)
```

JAR 파일 myProc_jar를 데이터베이스에서 제거합니다.

«

SQLJ.REPLACE_JAR:

»

SQLJ.REPLACE_JAR 저장 프로시듀어는 JAR 파일을 데이터베이스 시스템에 다시 놓습니다.

권한부여: CALL문의 권한 부여 ID가 보유한 권한에는 SYSJAROBJECTS 및 SYSJARCONTENTS 카탈로그 표에 대해 최소한 다음 중 하나가 포함되어야 합니다.

- 다음의 시스템 권한:
  - 표에 대한 SELECT, INSERT 및 DELETE 권한
  - 라이브러리 QSYS2에 대한 시스템 권한 *EXECUTE
- 관리 권한

CALL문의 권한 부여 ID가 보유한 권한에는 다음의 권한도 있어야 합니다.

- 설치되는 *jar-url* 매개변수에 지정된 JAR 파일에 대한 읽기(*R) 액세스.
- 제거하는 JAR 파일에 대한 *OBJMGT 권한. JAR 파일은 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile로 명명합니다.

허용된 권한을 이러한 권한으로 사용할 수 없습니다.

구문:

```
>>-CALL--SQLJ.REPLACE_JAR--(--'jar-url'--,--'jar-id'--)--------><
```

설명:

jar-url 대체할 JAR 파일이 포함된 URL. 지원되는 URL 체계는 'file:' 뿐입니다.

jar-id *jar-url*에 지정된 파일과 연관시킬 데이터베이스의 JAR ID. *jar-id*는 SQL 명명을 사용하며 JAR 파일을 내재적 또는 명시적 규정자에 지정된 스키마나 라이브러리에 설치합니다.

사용법 주의사항: SQLJ.REPLACE_JAR 저장 프로시듀어는 SQLJ.INSTALL_JAR를 사용하여 데이터베이스에 이미 설치한 JAR 파일을 대체합니다.

예: 다음의 명령은 SQL 대화식 세션에서 발행합니다.

```
CALL SQLJ.REPLACE_JAR('file:/home/db2inst/classes/Proc.jar', 'myproc_jar')
```

jar-id myproc_jar에 참조된 현재 JAR 파일은 file:/home/db2inst/classes/ 디렉토리에 있는 Proc.jar 파일로 대체됩니다.

SQLJ.UPDATEJARINFO:

SQLJ.UPDATEJARINFO는 SYSJARCONTENTS 카탈로그 표의 CLASS_SOURCE 열을 갱신합니다. 이 프로시듀어는 SQLJ 표준의 일부가 아니지만 iSeries용 DB2 UDB 저장 프로시듀어 빌더가 사용합니다.

권한부여: CALL문의 권한 부여 ID가 보유한 권한에는 SYSJARCONTENTS 카탈로그 표에 대해 최소한 다음 중 하나가 포함되어야 합니다.

- 다음의 시스템 권한:
  - 표에 대한 SELECT 및 UPDATEINSERT 권한
  - 라이브러리 QSYS2에 대한 시스템 권한 *EXECUTE
- 관리 권한

CALL문을 실행하는 사용자는 다음의 권한도 가지고 있어야 합니다.

- *jar-url* 매개변수에 지정된 JAR 파일에 대한 읽기(*R) 액세스. 설치되는 JAR 파일에 대한 읽기(*R) 액세스.
- JAR 파일을 설치하는 디렉토리에 대한 쓰기, 실행 및 읽기(*RWX) 액세스. 이 디렉토리는 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*이며 여기서 *schema*는 *jar-id*의 스키마입니다.

허용된 권한을 이러한 권한으로 사용할 수 없습니다.

구문:

```
>>-CALL--SQLJ.UPDATEJARINFO--(--'jar-id'--,--'class-id'--,--'jar-url'--)-->

>-------------------------------------------------------------><
```

설명:

jar-id 갱신할 데이터베이스의 JAR ID.

class-id
갱신할 클래스의 패키지 규정 클래스명.

jar-url JAR 파일을 갱신할 클래스 파일이 포함된 URL. 지원되는 URL 체계는 'file:' 뿐입니다.

예: 다음의 명령은 SQL 대화식 세션에서 발행합니다.

```
CALL SQLJ.UPDATEJARINFO('myproc_jar', 'mypackage.myclass',
                        'file:/home/user/mypackage/myclass.class')
```

jar-id myproc_jar와 연관된 JAR 파일은 mypackage.myclass 클래스의 새로운 버전으로 갱신됩니다. 클래스의 새로운 버전은 파일 /home/user/mypackage/myclass.class에서 확보합니다.

SQLJ.RECOVERJAR:

SQLJ.RECOVERJAR 프로시듀어는 SYSJAROBJECTS 카탈로그에 저장된 JAR 파일을 사용하여 /QIBM/UserData/OS400/SQLLib/Function/jar/*jarschema*/*jar_id*.jar 파일로 복원합니다.

권한부여: CALL문의 권한 부여 ID가 보유한 권한에는 SYSJAROBJECTS 카탈로그 표에 대해 최소한 다음 중 하나가 포함되어야 합니다.

- 다음의 시스템 권한:
  - 표에 대한 SELECT 및 UPDATEINSERT 권한
  - 라이브러리 QSYS2에 대한 시스템 권한 *EXECUTE
- 관리 권한

CALL문을 실행하는 사용자는 다음의 권한도 가지고 있어야 합니다.

- JAR 파일을 설치하는 디렉토리에 대한 쓰기, 실행 및 읽기(*RWX) 액세스. 이 디렉토리는 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*이며 여기서 *schema*는 *jar-id*의 스키마입니다.
- 제거하는 JAR 파일에 대한 *OBJMGT 권한. JAR 파일은 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile로 명명합니다.

구문:

```
>>-CALL--SQLJ.RECOVERJAR--(--'jar-id'--)----------------------><
```

설명:

jar-id 회복할 데이터베이스의 JAR ID.

예: 다음의 명령은 SQL 대화식 세션에서 발행합니다.

```
CALL SQLJ.UPDATEJARINFO('myproc_jar')
```

myproc_jar와 연관된 JAR 파일은 SYSJARCONTENT 표의 내용으로 갱신됩니다. 이 파일은 /QIBM/UserData/OS400/SQLLib/Function/jar/jar_schema myproc_jar.jar로 복사합니다.

Java 저장 프로시듀어 및 UDF에 대한 매개변수 전달 규약

다음 표에서는 SQL 자료 유형이 Java(TM) 저장 프로시듀어와 UDF에서 표현되는 방법을 설명합니다.

SQL 자료 유형	Java 매개변수 스타일 JAVA	Java 매개변수 스타일 DB2GENERAL
SMALLINT	short	short
INTEGER	int	int
BIGINT	long	long
DECIMAL(p,s)	BigDecimal	BigDecimal
NUMERIC(p,s)	BigDecimal	BigDecimal
REAL 또는 FLOAT(p)	float	float
DOUBLE PRECISION 또는 FLOAT 또는 FLOAT(p)	double	double
CHARACTER(n)	String	String

SQL 자료 유형	Java 매개변수 스타일 JAVA	Java 매개변수 스타일 DB2GENERAL
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
인디케이터 변수	-	-
CLOB	-	com.ibm.db2.app.Clob
BLOB	-	com.ibm.db2.app.Blob
DBCLOB	-	com.ibm.db2.app.Clob
DataLink	-	-

Java와 기타 프로그래밍 언어

Java(TM)에서는, Java 이외 언어로 작성된 코드를 호출할 수 있는 여러 가지 방법이 있습니다.

Java 원시 인터페이스

다른 언어로 작성된 코드를 호출할 수 있는 방법 중 하나는 선택한 Java 메소드를 '원시 메소드'로서 구현하는 것입니다. 원시 메소드는 다른 언어로 작성되어 Java 메소드의 실제 구현을 제공하는 프로시듀어입니다. 원시 메소드는 Java 원시 인터페이스(JNI)를 사용하여 JVM(Java Virtual Machine)에 액세스할 수 있습니다. 원시 메소드는 커널 스레드인 Java 스레드에서 실행되므로 스레드 안전이어야 합니다. 동일한 프로세스 내에서 여러 개의 스레드로 함수를 동시에 시작할 수 있다면 스레드 안전 함수입니다. 호출되는 모든 함수가 스레드 안전을 지원하면 함수도 스레드 안전을 지원합니다.

원시 메소드는 Java에서 직접 지원되지 않는 시스템 기능에 액세스하거나 기존의 사용자 코드에 인터페이스하는 "브릿지"입니다. 호출 중인 코드가 스레드 안전을 지원하지 않을 수 있으므로 원시 메소드를 사용할 때 주의해야 합니다.

JNI 및 ILE 원시 메소드에 대해서는 원시 메소드에 Java 원시 인터페이스 사용을 참조하십시오.

OS/400 PASE 원시 메소드

iSeries JVM(Java Virtual Machine)은 현재 OS/400(R) PASE 환경에서 실행 중인 원시 메소드 사용을 지원합니다. Java용 OS/400 PASE 원시 메소드를 사용하면 AIX(R)에서 실행하는 Java 어플리케이션이 iSeries 서버로 쉽게 이동될 수 있습니다. 클래스 파일과 AIX 원시 메소드 라이브러리를 iSeries의 통합 파일 시스템으로 복사하고, CL(control language), Qshell 또는 OS/400 PASE 터미널 세션 명령 프롬트 중 하나에서 이를 실행할 수 있습니다.

java.lang.Runtime.exec()

java.lang.Runtime.exec()를 사용하여 Java 프로그램에서 프로그램이나 명령을 호출할 수 있습니다. exec() 메소드는 iSeries 프로그램 또는 명령을 실행할 수 있는 다른 프로세스를 시작합니다. 이 모델에서 프로그램간 통신을 위해 하위 프로세스의 standard in, standard out 및 standard err을 사용할 수 있습니다.

프로세스간 통신

한 가지 옵션은 상위 프로세스와 하위 프로세스간의 통신을 위해 소켓을 사용하는 것입니다.

또한 프로그램 사이의 통신을 위해 스트림파일을 사용할 수도 있습니다. 그리고 다른 프로세스에서 실행 중인 프로그램간 통신시에 사용할 수 있는 옵션에 대해서는 프로그램간 통신 예를 참조하십시오.

다른 언어에서 Java를 호출하는 자세한 정보는 예: C에서 Java 호출 또는 예: RPG에서 Java 호출을 참조하시시오.

IBM Toolbox for Java를 사용하여 iSeries 서버에 있는 기존의 프로그램과 명령을 호출할 수도 있습니다. IBM Toolbox for Java와의 프로세스간 통신에는 보통 자료 대기행렬과 iSeries 메세지가 사용됩니다.

주: Runtime.exec(), IBM Toolbox for Java 또는 JNI를 사용하여 Java 프로그램의 이식성을 절충할 수 있습니다. "순수한" Java 환경에서는 이러한 메소드를 사용하지 마십시오.

Java 호출 API

JNI(Java 원시 인터페이스) 스펙의 일부이기도 한 Java 호출 API를 사용하면 비Java 어플리케이션에서도 JVM(Java Virtual Machine)을 사용할 수 있습니다. 또한 이것은 어플리케이션의 확장으로서 Java 코드를 사용할 수 있게 합니다.

원시 메소드에 대한 Java 원시 인터페이스 사용

순수한 Java(TM)만으로 사용자의 프로그래밍 요구를 만족시킬 수 없는 경우에만 원시 메소드를 사용해야 합니다. 다음 상황에서 원시 메소드를 사용하는 경우에만 원시 메소드의 사용을 제한하십시오.

- 순수 Java로 사용할 수 없는 시스템 함수에 액세스할 때.

- 원시 구현을 통해 이용할 수 있는 성능 측면의 메소드를 구현할 때.
- Java가 다른 API를 호출하게 하는 기존의 API(어플리케이션 프로그램 인터페이스)에 대해 인터페이스할 때.

원시 메소드에 JNI(Java 원시 인터페이스)를 사용하려면 다음 단계를 수행하십시오.

1. 표준 Java 언어 구문으로 원시 메소드가 될 메소드를 지정하여 클래스를 설계하십시오.
2. 원시 메소드 구현이 들어 있는 서비스 프로그램(*SRVPGM)에 대한 라이브러리와 프로그램명을 판별하십시오. 클래스에 대한 정적 초기화 프로그램에서 System.loadLibrary() 메소드 호출을 코드화할 때 서비스 프로그램명을 지정하십시오.
3. `javac` 툴을 사용하여 Java 소스를 클래스 파일로 컴파일하십시오.
4. `javah` 툴을 사용하여 헤더 파일(.h)을 작성하십시오. 이 헤더 파일에는 원시 메소드 구현을 작성하기 위한 정확한 프로토타입이 들어 있습니다. -d 옵션은 헤더 파일을 작성해야 하는 디렉토리를 지정합니다.
5. CPYFRMSTMF(스트림 파일에서 복사) 명령을 사용하여 통합 파일 시스템에서 소스 파일의 멤버로 헤더 파일을 복사하십시오. C 컴파일러가 사용할 수 있도록 헤더 파일을 소스 파일 멤버로 복사해야 합니다. CRTCMOD(바인드 ILE C/400 프로그램 작성) 명령에 대해 새로운 스트림 파일 지원을 사용하여 C 소스 및 C 헤더 파일을 통합 파일 시스템에 남겨 두십시오. CRTCMOD 명령 및 스트림 파일 사용에 대한 자세한 내용은 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  를 참조하십시오.
6. 원시 메소드 코드를 작성하십시오. 원시 메소드에 사용된 언어와 함수에 대한 자세한 내용은 Java 원시 메소드 및 스레드 고려사항을 참조하십시오.
   a. 이전 단계에서 작성된 헤더 파일을 포함하십시오.
   b. 헤더 파일의 원형을 정확하게 일치시키십시오.
   c. 스트링이 JVM(Java Virtual Machine)으로 전달되는 경우, 스트링을 ASCII(미국 표준 정보 교환 코드)로 변환하십시오. 자세한 내용은 Java 문자 코드화를 참조하십시오.
7. 원시 메소드가 JVM(Java Virtual Machine)과 상호작용이 가능해야 하는 경우, JNI가 제공하는 함수를 사용하십시오.
8. CRTCMOD를 사용하여 C 소스 코드를 모듈(*MOD) 오브젝트로 컴파일하십시오.
9. CRTSRVPGM(서비스 프로그램 작성) 명령을 사용하여 하나 이상의 모듈 오브젝트를 한 서비스 프로그램(*SRVPGM)에 바인드하십시오. 이 서비스 프로그램명은 System.load() 또는 System.loadLibrary() 함수 호출에 있는 Java 코드에 제공된 이름과 일치해야 합니다.
10. Java 코드에서 System.loadLibrary() 호출을 사용한 경우, 다음 중 하나를 수행하십시오.

    J2SDK보다 이전의 버전을 사용 중인 경우:
    새로운 서비스 프로그램이 포함된 iSeries 라이브러리를 iSeries 라이브러리 리스트에 추가하십시오. 라이브러리를 추가하려면 ADDLIBLE(라이브러리 리스트 항목 추가) 명령을 사용하십시오. 이렇게 하면 Java 프로그램이 System.loadLibrary() 함수를 처리할 때 해당 서비스 프로그램을 찾을 수 있습니다.

    J2SDK, 버전 1.2 이상을 사용 중인 경우:
    라이브러리 리스트를 변경할 필요가 없습니다. 대신, 다음 중 하나를 수행할 수 있습니다.

- LIBPATH 환경 변수에 필요한 라이브러리 리스트를 포함시킵니다. QShell 및 iSeries 명령행에서 LIBPATH 환경 변수를 변경할 수 있습니다.
  - Qshell 명령 프롬트에서 다음을 입력하십시오.

    ```
    export LIBPATH=/QSYS.LIB/MYLIB.LIB
    ```

    ```
    java -Djava.version=1.4 myclass
    ```

  - 또는, 명령행에서 다음을 입력하십시오.

    ```
    ADDENVVAR LIBPATH '/QSYS.LIB/MYLIB.LIB'
    ```

    ```
    JAVA PROP((java.version 1.4)) myclass
    ```

- 또는, **java.library.path** 등록 정보에 리스트를 제공하십시오. QShell 및 iSeries 명령행에서 java.library.path 등록 정보를 변경할 수 있습니다.
  - Qshell 명령 프롬트에서 다음을 입력하십시오.

    ```
    java -Djava.library.path=/QSYS.LIB/MYLIB.LIB -Djava.version=1.4 myclass
    ```

  - 또는, iSeries 명령행에서 다음을 입력하십시오.

    ```
    JAVA PROP((java.library.path '/QSYS.LIB/MYLIB.LIB') (java.version '1.4')) myclass
    ```

여기서 */QSYS.LIB/MYLIB.LIB*는 System.loadLibrary() 호출을 사용하여 로드하려는 라이브러리이며, *myclass*는 Java 어플리케이션의 이름입니다.

11. System.load(스트링 patches)의 patches 구문은 다음 중 하나입니다.
    - ″path″(서비스 프로그램이 제공하는 라이브러리를 지정하는 통합 파일 시스템 파일명) 이것은 ″/qsys.lib/mylib.lib/myNMsp.srvpgm″과 같은 *SRVPGM에 대한 기호 링크입니다.
    - /qsys.lib/sysNMsp.srvpgm
    - /qsys.lib/mylib.lib/myNMsp.srvpgm
    - J2SDK보다 이전의 버전이 있는 경우: /qsys.lib/%libl%.lib/myNMsp.srvpgm

주: System.loadLibrary(″myNMsp″) 메소드를 사용하는 것과 같습니다.

주: pathname이 스트링 리터럴로 사용되는 경우 인용부호로 묶어야 합니다. 예를 들어, System.load(″/qsys.lib/mylib.lib/myNMsp.srvpgm″)입니다.

12. System.loadLibrary(스트링 libya)의 ″libya″ 구문은 mysp입니다. 시스템이 *LIBL을 사용하여 mysp를 찾습니다. 예를 들어, loadLibrary(″myNMsp″)는 System.load(″/qsys.lib/%libl%.lib/myNMsp.srvpgm″)와 동등합니다. ″pathname″이 스트링 리터럴로 사용되는 경우 libname을 인용부호로 묶어야 합니다.

    주: %libl% 구문은 J2SDK의 경우에 지원되지 않습니다.

JNI에 대한 완전한 설명은 Java Native Interface by Sun Microsystems, Inc.와 The Source for Java Technology java.sun.com 을 참조하십시오.

원시 메소드에 JNI 사용법의 예에 대해서는 예: 원시메소드에 Java 원시 인터페이스를 참조하십시오.

Java 호출 API

JNI(Java(TM) Native Interface)의 한 파트인 호출 API를 사용하면 비Java 코드가 JVM을 작성하고, Java 클래스를 로드 및 사용할 수 있습니다. 이 기능은 멀티스레드 처리 프로그램이 멀티스레드에 있는 단일 JVM(Java Virtual Machine)에서 실행 중인 Java 클래스를 사용할 수 있게 합니다.

어플리케이션은 JVM(Java Virtual Machine)을 제어합니다. 어플리케이션은 JVM(Java Virtual Machine)을 작성하고(어플리케이션이 서브루틴을 호출하는 것과 유사한 방법으로) Java 메소드를 호출하고 JVM(Java Virtual Machine)을 소멸시킬 수 있습니다. 일단 JVM(Java Virtual Machine)을 작성했으면 어플리케이션이 명시적으로 소멸시킬 때까지 프로세스에서 실행 준비 상태로 남아 있습니다. 소멸 중에 JVM(Java Virtual Machine)은 종결자 수행, JVM(Java Virtual Machine) 스레드 종료 및 JVM(Java Virtual Machine) 자원 해제와 같은 지우기(clean-up)를 수행합니다.

실행할 준비가 된 JVM(Java Virtual Machine)이 있으면 C로 작성된 어플리케이션은 이 기능을 수행하기 위해 JVM(Java Virtual Machine)을 호출할 수 있습니다. 이것은 또한 JVM(Java Virtual Machine)에서 C 어플리케이션으로 리턴하여 다시 JVM(Java Virtual Machine)을 호출하는 등의 일을 수행할 수 있습니다. JVM(Java Virtual Machine)은 한 번 작성되면 JVM(Java Virtual Machine)을 호출하기 전에 재작성할 필요가 없습니다.

Java 프로그램을 실행하기 위해 호출 API를 사용할 때 QIBM_USE_DESCRIPTOR_STDIO라는 환경 변수를 사용하여 STDOUT 및 STDERR의 목적지를 제어합니다. 이 환경 변수가 Y 또는 I(예를 들면, QIBM_USE_DESCRIPTOR_STDIO=Y), JVM(Java Virtual Machine)은 STDIN(fd 0), STDOUT(fd 1) 및 STDERR(fd 2)에 대한 파일 설명자를 사용합니다. 이 경우 프로그램은 이 작업에서 이들 파일을 첫 번째 3개의 파일이나 파이프로 열어서 이들 파일 설명자를 유효한 값으로 설정해야 합니다. 작업에서 열린 첫 번째 파일에는 fd 0, 두 번째 파일은 fd 1 그리고 세 번째 파일에는 fd 2가 부여됩니다. 파생 API로 초기설정된 작업의 경우 파일 설명자 맵을 사용하여 이 설명자를 미리 지정할 수 있습니다(파생 API에 대한 문서 참조). 환경 변수 QIBM_USE_DESCRIPTOR_STDIO가 설정되지 않거나 임의의 다른 값으로 설정되면 STDIN, STDOUT 또는 STDERR에 파일 설명자가 사용되지 않습니다. 대신, 현재 작업이 소유한 스풀 파일로 STDOUT과 STDERR이 라우트되고, STDIN 사용은 IO 예외를 유발합니다.

호출 API를 사용하는 예에 대해서는 예: Java 호출 API를 참조하십시오. IBM Developer Kit for Java에서 지원되는 호출 API에 대한 자세한 내용은 호출 API 함수를 참조하십시오.

호출 API 함수: IBM Developer Kit for Java(TM)는 다음 호출 API 함수를 지원합니다.

주: API를 사용하기 전에 멀티스레드가 가능한 작업 상태인지 확인해야 합니다. 멀티스레드가 가능한 작업에 대한 자세한 정보는 멀티스레드 어플리케이션을 참조하십시오.

- **JNI_GetDefaultJavaVMInitArgs**

  주: 이 함수는 JDK(Java Development Kit) 1.1.x에 대해서만 지원됩니다.

  JVM(Java Virtual Machine)을 작성할 때 JNI_CreateJavaVM에 전달되어야 하는 인수의 디폴트 값이 들어 있는 JDK 1.1 구조를 리턴합니다.

  서명:

  ```
  jint JNI_GetDefaultJavaVMInitArgs(void *args_);
  ```

- **JNI_GetCreatedJavaVMs**

  작성된 모든 JVM(Java Virtual Machine)에 대한 정보를 리턴합니다.

  서명:

  ```
  jint JNI_GetCreatedJavaVMs(JavaVM **vmBuf,
                jsize bufLen,
                jsize *nVMs);
  ```

  vmBuf는 포인터 수인 bufLen이 *크기*를 판별하는 출력 영역입니다. 각 JVM(Java Virtual Machine)은 java.h에서 정의되는 연관된 JavaVM 구조를 가집니다. 이 API는 vmBuf가 가득차지 않으면 작성된 각 JVM(Java Virtual Machine)과 연관된 JavaVM 구조에 대한 포인터를 vmBuf에 저장합니다. JavaVM 구조에 대한 포인터는 작성되는 해당 JVM(Java Virtual Machine) 순서로 저장됩니다. nVMs는 현재 작성되는 가상 기계의 수를 리턴합니다. iSeries 서버는 둘 이상의 JVM(Java Virtual Machine) 작성을 지원하므로 1보다 큰 값을 예상할 수 있습니다. vmBuf *크기*와 함께 이 정보는 작성된 각 JVM(Java Virtual Machine)에 대한 JavaVM 구조의 포인터가 리턴되는지 여부를 판별합니다.

- **JNI_CreateJavaVM**

  Java를 작성한 후 어플리케이션에서 JVM(Java Virtual Machine)을 사용할 수 있게 합니다.

  Java Development Kit 1.1.x에 대한 서명:

  ```
  jint JNI_CreateJavaVM(JavaVM **p_vm,
                   JNIEnv **p_env,
                 void *vm_args);
  ```

  J2SDK(Java 2 Software Development Kit)에 대한 서명:

  ```
  jint JNI_CreateJavaVM(JavaVM **p_vm,
                   void **p_env,
                 void *vm_args);
  ```

  p_vm은 새로 작성되는 JVM(Java Virtual Machine)의 JavaVM 포인터 주소입니다. 몇몇 다른 JNI 호출 API는 p_vm을 사용하여 JVM(Java Virtual Machine)을 식별합니다. p_env는 새로 작성된 JVM(Java Virtual

Machine)의 JNI 환경 포인터의 주소입니다. 이것은 해당 함수를 시작하는 JNI 함수 표를 가리킵니다. vm_args는 JVM(Java Virtual Machine) 초기화 매개변수가 있는 구조입니다. JDK 1.1.x를 사용할 때 JNI_GetDefaultJavaVMInitArgs를 호출하여 디폴트 값이 들어 있는 구조를 얻을 수 있습니다. J2SDK에서 이것을 수행하는 방법에 대한 세부사항은 Java 원시 인터페이스  를 참조하십시오.

RUNJVA(Java 실행) 명령이나 JAVA 명령을 시작하고 동등한 명령 매개변수를 갖는 등록 정보를 지정하는 경우 명령 매개변수가 우선합니다. 등록 정보는 무시됩니다. 예를 들어, os400.optimization 매개변수는 이 명령에서 무시됩니다.

```
JAVA CLASS(Hello) PROP((os400.optimization 0))
```

JNI_CreateJavaVM API가 지원하는 0S/400 고유 등록 정보 리스트는 Java 시스템 등록 정보를 참조하십시오.

주: 한 프로세스 내의 복수 JVM(Java Virtual Machine)을 사용하여 모든 JVM(Java Virtual Machine)은 원시 메소드에 할당된 동일한 프로세스 정적 기억장치를 공유합니다. Java 가상 기계 내부 구현이 이미 자료를 Java 가상 기계별로 분할하지만, 원시 메소드 어플리케이션이 있으면 JVM(Java Virtual Machine)이 프로세스 정적 기억장치를 공유한다는 점을 고려해야 합니다. 기타 고려사항은 복수 JVM(Java Virtual Machine) 지원을 참조하십시오.

- **DestroyJavaVM**

  JVM(Java Virtual Machine)을 소멸시킵니다.

  서명:

  ```
  jint DestroyJavaVM(JavaVM *vm)
  ```

  JVM(Java Virtual Machine)이 작성될 때 vm은 리턴되는 JavaVM 포인터입니다.

- **AttachCurrentThread**

  JVM(Java Virtual Machine) 서비스를 사용할 수 있도록 스레드를 JVM(Java Virtual Machine)에 연결합니다.

  JDK(Java Development Kit) 1.1.x에 대한 서명:

  ```
  jint AttachCurrentThread(JavaVM *vm,
                           JNIEnv **p_env,
                           void *thr_args);
  ```

  J2SDK(Java 2 Software Development Kit)에 대한 서명:

  ```
  jint AttachCurrentThread(JavaVM *vm,
                           void **p_env,
                           void *thr_args);
  ```

  JavaVM 포인터인 vm은 스레드가 연결되는 JVM(Java Virtual Machine)을 식별합니다. p_env는 현재 스레드의 JNI 인터페이스 포인터가 있는 곳의 위치 포인터입니다. thr_args는 VM 특유의 스레드 접속 인수를 포함합니다.

- **DetachCurrentThread**

서명:

```
jint DetachCurrentThread(JavaVM *vm);
```

vm은 스레드가 분리되고 있는 JVM(Java Virtual Machine)을 식별합니다.

호출 API 함수에 대한 완전한 설명은 Sun Microsystems, Inc.의 Java 원시 인터페이스 또는 The Source for Java Technology java.sun.com 을 참조하십시오.

복수 JVM(Java Virtual Machine) 지원: Sun Microsystems, Inc. 참조 구현과는 달리 iSeries 서버의 Java(TM)는 단일 작업이나 프로세스 내에서 복수 JVM(Java virtual machine) 작성을 지원합니다. 이것은 작업에서 JNI_CreateJavaVM()을 두 번 이상 호출할 수 있으며 JNI_GetCreatedJavaVMs()가 결과 리스트에 둘 이상의 JVM을 리턴할 수 있음을 의미합니다.

- »

  V5R2 이전에는 JNI_GetCreatedJavaVMs 기능이 JVM 리스트에서 둘 이상의 JVM을 리턴하는 것이 가능했습니다. V5R2에서는 JNI_GetCreatedJavaVMs가 많아야 하나의 JVM만을 리턴합니다.
- V5R2 이전에는 단일 프로세스 내에서 JNI_CreateJavaVM 기능을 반복적으로 호출할 수 있었으며 성공한 각 호출을 사용하여 별도의 고유한 JVM을 작성했습니다. V5R2에서는 JNI_CreateJavaVM 기능이 오류 코드를 리턴합니다(jni.h에 정의된 JNI_ERR -1 오류).

«

단일 작업이나 프로세스에서 사용하기 위해 복수 JVM을 작성하려는 경우, 다음을 주의 깊게 고려해야 합니다.

원시 메소드 정적 기억장치 범위

- 원시 메소드 구현이 들어 있는 서비스 프로그램은 작성되는 JVM의 수와 상관없이 작업당 한 번만 활성화됩니다. 이것은 원시 메소드 정적 기억장치가 특정한 JVM(Java Virtual Machine)이 아닌 작업으로 제한됨을 암시합니다.
- 원시 메소드가 정적 기억장치에 배치하는 값은 원시 메소드를 호출한 JVM과 독립적입니다. 이 값들은 작업에 있는 모든 JVM에 가시적입니다.
- 복수 JVM 시나리오에서 의도적으로 원시 메소드 정적 기억장치를 사용하는 경우, JVM에 고유한 동기화된 메소드 및 모니터의 사용에 추가하여 동기화에 대한 가능한 요구사항을 주의 깊게 고려해야 합니다. 원시 메소드를 동기화된 것으로 규정하는 것은 단일 JVM 내에서의 동시 실행만을 금지하며 복수 JVM으로부터의 동시 실행은 금지하지 않습니다.

JVM(Java Virtual Machine) 종료

- JVM(Java Virtual Machine)이 사용자 호출 `java.lang.System.exit()` 또는 내부 JVM 실패로 인해서 비정상적으로 종료되는 경우, 실패한 JVM과 그의 접속된 모든 스레드가 종료됩니다.
- 프로세스 초기 스레드가 실패한 JVM에 접속된 스레드 중 하나인 경우, 예외가 초기 스레드에 전달됩니다. 초기 스레드가 이 예외를 처리하는 경우, 다른 JVM은 계속 실행할 수 있습니다.

- 프로세스 초기 스레드가 처리되지 않는 예외나 기타 다른 이유로 인해서 종료되는 경우 프로세스의 모든 JVM도 종료됩니다.

C로부터의 비정상 종료

멀티스레드 작업의 임의의 스레드에서 ILE/C `exit()` 또는 `abort()` 루틴을 사용하는 경우, 모든 JVM을 포함하여 전체 작업이 즉시 종료됩니다.

예: Java 호출 API: 다음 예는 표준 호출 API 패러다임을 따릅니다. 예를 들면, 다음을 수행합니다.

- JNI_CreateJavaVM을 사용하여 JVM(Java(TM) virtual machine)을 작성합니다.
- JVM(Java Virtual Machine)을 사용하여 실행하려는 클래스 파일을 찾습니다.
- 클래스의 원시 메소드에 대한 methodID를 찾습니다.
- 클래스의 원시 메소드를 호출합니다.
- 예외가 발생하면 오류를 보고합니다.

다음 프로그램을 컴파일하려면 새로운 JVM(Java Virtual Machine)을 시작하기 위해 함수를 내보내기하는 서비스 프로그램과 함께 바인드해야 합니다. 이것은 다음을 위해 필요한 입력점입니다.

- 작성할 매개변수를 초기설정하는 JNI_GetDefaultJavaVMInitArgs.
- JVM(Java Virtual Machine)을 작성하는 JNI_CreateJavaVM.

프로그램을 컴파일할 때 컴파일 명령으로 명시적으로 수행할 필요는 없습니다. 이 입력점을 내보내기하는 서비스 프로그램이 시스템 바인딩 디렉토리에 있습니다. 서비스 프로그램의 이름은 QJVAJNI입니다.

다음 프로그램을 실행하려면 `SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)`를 사용하십시오. JVM(Java Virtual Machine)을 작성하는 모든 작업은 멀티스레드가 가능해야 합니다. iSeries 서버에서 유일하게 멀티스레드가 가능한 작업은 BCI(일괄처리 즉시) 작업입니다. 프로그램의 출력뿐만 아니라 기본 프로그램의 출력도 QPRINT 스풀 파일에서 종료됩니다. WRKSBMJOB(제출된 작업에 대한 작업) 명령을 사용하여 SBMJOB(작업 제출)로 시작된 작업을 보는 경우 스풀 파일을 볼 수 있습니다.

주: 사용자 프로그램이 프로세스에서 유일한 스레드인 경우를 제외하고, 아래에 사용된 C 런타임 `exit()` 루틴은 권장되지 않습니다. 멀티스레드를 지원할 수 있는 프로세스에서 호출될 때 `exit()`은 프로세스의 모든 스레드를 즉시 종료합니다.

예: JDK 1.1.x와 함께 Java 호출 API 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <jni.h>

int main (int argc, char *argv[])
{
    JDK1_1InitArgs initArgs; /* Virtual Machine (VM) initialization structure.
                              * This is the structure that is passed by reference to
```

```
                              JNI_CreateJavaVM().
                              *See jni.h for details.
                              */

    JavaVM* myJVM;           /* The JavaVM and JNIEnv pointers that you get back. */

    JNIEnv* myEnv;           /* ...from the JNI_CreateJavaVM() call. */

    char*   myClasspath;     /* You need to change the classpath, so you have your own. */

    jclass  myClass;         /* The class you are going to find, called 'NativeHello'. */

    jmethodID mainID;        /* The method ID of the class' "main" routine. */

    jclass  stringClass;     /* Required to create a string array argument for 'main'. */

    jobjectArray args;       /* Because main expects an array of strings, you must pass one. */

    /*  Set the version field of the initialization arguments. */
    initArgs.version = 0x00010001;

    /*  Get the default initialization arguments. */
    JNI_GetDefaultJavaVMInitArgs(&initArgs);

    /* Now, you want to add the directory onto the end of the classpath,
     * so that the findClass finds it correctly.  To do this, you have two options:
     * You can append your classpath entries to the default classpath that is returned
     * by the call to JNI_GetDefaultJavaVMInitArgs, or
     * you can use OS/400 specific functions for the same result.  This is a
     *    three-step solution:
     *       1. Set the CLASSPATH environment variable to its requirements with 'putenv()'
     *       2. Clear the initialization arguments classpath to NULL, which forces
     *          JNI_CreateJavaVM to look at the CLASSPATH value
     *       3. Set the "os400.class.path.system=PRE" property, to force JNI_CreateJavaVM
     *          to prepend the system default classpath to the effective classpath.
     *
     * The first option is used in this example, because it is more platform independent
     *
     * Note: You must specify the directory name in UTF-8 format!  So, you wrap
     *       blocks of code in #pragma convert statements.
     */

#pragma convert(819)
    myClasspath = malloc( strlen(initArgs.classpath) + strlen(":/CrtJvmExample") + 1 );
    strcpy( myClasspath, initArgs.classpath );
    strcat( myClasspath, ":/CrtJvmExample" );
    initArgs.classpath = myClasspath;

#pragma convert(0)

    /*  Create the JVM. */
    if (JNI_CreateJavaVM(&myJVM, &myEnv, &initArgs)) {
        fprintf(stderr, "Failed to create the JVM\n");
        exit(1);
    }

    /*  Use the newly created JVM to find the example class.
     *  Note: Again, you are dealing with UTF-8 here, so you
     *  have to wrap the calls in #pragma convert.
```

```
     */

#pragma convert(819)
    if (! (myClass = (*myEnv)->FindClass(myEnv, "NativeHello"))) {

#pragma convert(0)
        /* Cannot find the class, so write an error message
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to find the class 'NativeHello'\n");
        exit(1);  /* Exit stops the entire process on an iSeries server. */
    }

    /* Now, get the method identifier for the 'main' entry point
     * of the class. Note: The signature of 'main' is always
     * the same for every class, "main" and "([Ljava/lang/String;)V"
     * Again, you are dealing with UTF-8.
     */

#pragma convert(819)
    if (! (mainID = (*myEnv)->GetStaticMethodID(myEnv, myClass,
                                                "main",
                                                "([Ljava/lang/String;)V"))) {
        /*  The 'main' methodID is not found for some reason. */
    if ( (*myEnv)->ExceptionOccurred(myEnv) ) {
            /* a java exception occurred, so print it out */
        (*myEnv)->ExceptionDescribe(myEnv);
            /*  The JVM ends. */
            (*myEnv)->FatalError(myEnv, "Failed to find jmethodID of 'main()'");
            }

#pragma convert(0)
        /* Cannot find the 'main' methodID, so write an error message
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to find the 'main()' method\n");
        exit(1);  /* Exit stops the entire process on an iSeries server. */
    }

#pragma convert(819)
    if (! (stringClass = (*myEnv)->FindClass(myEnv,"java/lang/String"))) {

#pragma convert(0)
        /* Did not find java/lang/String, so write an error message
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to find the java/lang/String");
        exit(1);  /* exit stops the entire process on an iSeries server.*/
    }

    /*  Now, you need to create an empty array of strings,
     *  because ([Ljava/lang/String) is a required part of the signature of
     *  every Java main routine.
     */

    if (! (args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0)))   {
```

```
        /* Empty array was not created, so write an error message
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to create empty array of strings");
        exit(1);  /* Exit stops the entire process on an iSeries server. */
    }

   /* 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) ) {
        fprintf(stderr,"An exception occurred while running 'main'");
        exit(1);
    }

    /* Finally, destroy the JavaVM that you created. */

    if ( (*myJVM)->DestroyJavaVM(myJVM) ) {
        fprintf(stderr, "Failed to destroy the JVM\n");
        exit(1);
    }

    /* All done. */

    return 0;
}
```

예: J2SDK와 함께 Java 호출 API 사용

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
#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. */
    initArgs.version = 0x00010002;

    /* 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";

    initArgs.options=options;  /* Pass in the classpath that has been set up. */
    initArgs.nOptions = 1;     /* Only passing the one classpath option */

    /*  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;
}
```

자세한 내용은 Java 호출 API를 참조하십시오.

Java 원시 메소드 및 스레드 고려사항

원시 메소드를 사용하여 Java(TM)에서 사용할 수 없는 기능에 액세스할 수 있습니다.

원시 메소드를 사용하여 Java를 더 잘 활용하려면 다음 개념을 이해해야 합니다.

- Java와 연결된 원시 스레드 중 어느 것으로 작성되었는지에 관계없이 Java 스레드는 모든 부동 소수점 예외를 사용할 수 없게 합니다. 스레드가 부동 소수점 예외를 다시 사용 가능하게 하는 원시 메소드를 실행하는 경우 Java는 사용 가능해진 예외를 다시 중지시키지 않습니다. Java 코드 실행를 위해 리턴하기 전에 사용자 어플리케이션이 예외를 사용할 수 없게 할 경우 부동 소수점 예외가 발생했다면, Java 코드는 올바르게 작동하지 않을 수 있습니다. 원시 스레드가 JVM(Java Virtual Machine)에서 분리될 때 스레드의 부동 소수점 예외 마스크는 스레드가 연결될 때 효력이 있었던 값으로 복원됩니다.
- 원시 스레드가 JVM(Java Virtual Machine)에 연결되면 JVM(Java Virtual Machine)은 Java가 정의하는 1 - 10의 우선순위 체계에 맞도록 필요에 따라 스레드 우선순위를 변경합니다. 스레드가 분리되면 우선순위가 복원됩니다. 스레드는 연결을 한 다음에 원시 메소드 인터페이스(예, POSIX API)를 사용하여 스레드 우선순위를 변경할 수 있습니다. Java는 JVM(Java Virtual Machine)으로 다시 변환할 때 스레드 우선순위를 변경하지 않습니다.
- JNI(Java Native Interface)의 호출 API 구성요소를 사용하여 어플리케이션에 JVM(Java Virtual Machine)을 삽입할 수 있습니다. 어플리케이션이 JVM을 작성했지만 JVM이 비정상적으로 종료된 경우, 프로세스의 초기 스레드가 JVM 종료 시에 JVM에 접속되었다면 이 스레드에 MCH74A5 ″JVM이 종료됨″ iSeries 예외 신호가 표시됩니다. JVM(Java Virtual Machine)은 다음과 같은 이유로 이상 종료될 수 있습니다.
  - 사용자가 java.lang.System.exit() 메소드를 호출함.
  - JVM(Java Virtual Machine)에 필요한 스레드가 종료함.
  - JVM(Java Virtual Machine)에서 내부 오류가 발생함.

  이 작동은 다른 대부분의 Java 플랫폼과 다릅니다. 다른 대부분의 플랫폼에서는 JVM(Java Virtual Machine)이 종료 즉시 JVM(Java Virtual Machine)을 자동으로 작성하는 프로세스가 갑자기 종료합니다. 어플리케이션이 MCH74A5 예외 신호를 모니터하여 처리할 경우에는 프로세스를 계속 실행할 수 있습니다. 그 외의 경우에는 예외가 미처리 상태로 될 때 프로세스가 종료됩니다. iSeries 특정 MCH74A5 예외를 처리하는 코드를 추가하여 어플리케이션을 다른 플랫폼으로 쉽게 이동하지 못하도록 만들 수 있습니다.

원시 메소드는 항상 멀티스레드 프로세스에서 실행되므로 메소드에 있는 코드는 스레드 안전이어야 합니다. 원시 메소드에 사용되는 언어와 함수에 대해 다음과 같은 제한사항이 적용됩니다.

- 이 언어는 스레드 안전을 지원하지 않으므로 원시 메소드에 대해 ILE CL을 사용하지 않아야 합니다. 스레드 안전 CL 명령을 실행하기 위해 C 언어 system() 함수 또는 java.lang.Runtime.exec() 메소드를 사용할 수 있습니다.
  - C 또는 C++ 원시 메소드로부터 스레드 안전 CL 명령을 실행하려면 system() 함수를 사용하십시오.
  - Java로부터 직접 스레드 안전 CL 명령을 실행하려면 java.lang.Runtime.exec() 메소드를 사용하십시오.
- ILE C, ILE C++, ILE COBOL 및 ILE RPG를 사용하여 원시 메소드를 작성할 수 있지만 원시 메소드 내에서 호출되는 모든 함수가 스레드 안전을 지원하는 것이어야 합니다.

주: 원시 메소드 작성에 대한 컴파일시 지원은 현재 C와 C++ 언어에 대해서만 제공됩니다. 다른 언어로 원시 메소드를 작성하는 것이 가능하나 훨씬 복잡할 수 있습니다.

주의:

모든 표준 C, C++, COBOL 또는 RPG 함수가 스레드 안전을 지원하는 것이 아닙니다.

- C와 C++ exit() 및 abort() 함수를 원시 메소드 내에 사용할 수 없습니다. 이 함수는 JVM(Java Virtual Machine)을 실행하는 전체 프로세스를 중단시킵니다. 스레드가 Java에서 시작되었는지의 여부에 관계없이 프로세스 내의 모든 스레드가 이에 포함됩니다.

  주: 참조되는 exit() 함수는 C와 C++ 기능으로 java.lang.Runtime.exit() 메소드와 동일하지 않습니다.

iSeries 서버의 스레드에 대한 자세한 정보는 멀티스레드 어플리케이션을 참조하십시오.

원시 메소드 및 Java 원시 인터페이스

원시 메소드는 Java 이외 다른 언어로 시작하는 Java(TM) 메소드입니다. 원시 메소드는 Java에서 직접 사용할 수 없는 시스템 고유 기능 및 API에 액세스할 수 있습니다.

원시 메소드에는 시스템 지정 코드가 포함되므로 원시 메소드를 사용할 경우에 어플리케이션의 이식성에 제한을 받습니다. 원시 메소드는 신규 원시 코드 명령문이거나 기존 원시 코드를 호출하는 원시 코드 명령문일 수 있습니다.

원시 메소드가 요구된다고 판단되면 메소드가 실행하는 JVM(Java Virtual Machine)과 상호작용이 가능해야 합니다. JNI(Java 원시 인터페이스)는 플랫폼 중립적으로 이 상호운영성을 용이하게 합니다.

JNI는 이 원시 메소드가 JVM(Java Virtual Machine)과 여러 가지 방법으로 상호 운영되도록 하는 일련의 인터페이스 집합입니다. 예를 들어, JNI에 새로운 오브젝트의 작성, 메소드 호출, 필드 확보 및 필드 설정, 예외 처리, 스트링 및 배열의 조작을 수행하는 인터페이스가 포함됩니다.

JNI에 대한 완전한 설명은 Java Native Interface by Sun Microsystems, Inc. 또는 The Source for Java Technology java.sun.com 을 참조하십시오.

원시 메소드의 스트링

많은 JNI(Java(TM) Native Interface) 기능에서는 C 언어 유형 스트링을 매개변수로서 수용합니다. 예를 들면, FindClass() JNI 함수는 classfile의 완전한 규정명을 지정하는 스트링 매개변수를 허용합니다. classfile이 발견되면 FindClass에 의해 로드된 다음, 해당 참조가 FindClass 호출자에게 리턴됩니다.

모든 JNI 함수의 스트링 매개변수가 UTF-8로 코드화됩니다. UTF-8에 대한 자세한 내용은 JNI 스펙을 참조할 수는 있지만, 대부분의 경우에 7-bit ASCII(미국 표준 정보 교환 코드) 문자가 UTF-8 표시와 동등하다는 것만 알면 충분합니다. 7-bit ASCII 문자는 실제로는 8-bit 문자이지만 첫 번째 비트는 항상 0입니다. 따라서 대부분의 ASCII C 스트링이 실제로 이미 UTF-8에 있습니다.

iSeries 서버의 C 컴파일러는 디폴트로 EBCDIC(확장 2진화 십진변환 코드)에서 작동되므로 UTF-8로 JNI 함수에 스트링을 제공할 수 있습니다. 두 가지 방법으로 이것을 실행합니다. 리터럴 스트링 또는 동적 스트링을

사용할 수 있습니다. 리터럴 스트링은 소스 코드를 컴파일할 때 그 값이 알려지는 스트링입니다. 동적 스트링은 컴파일할 때 값이 알려지지 않는 스트링으로서 런타임에 실제로 계산됩니다.

원시 메소드의 리터럴 스트링: 스트링이 7비트 ASCII(미국 표준 정보 교환 코드) 표시된 문자로 구성된 경우, 리터럴 스트링은 UTF-8으로 코드화되기 쉽습니다. 대부분의 경우처럼 스트링이 ascii로 표시될 수 있는 경우, 컴파일러의 현재 코드 페이지를 수정하는 'pragma'문을 사용하여 스트링을 괄호로 묶을 수 있습니다. 그러면, 컴파일러가 JNI가 요구하는 UTF-8 양식으로 스트링을 내부적으로 저장합니다. 스트링이 ASCII로 표시될 수 없으면 기본 ebcdic(확장 2진화 십진 변환 코드)를 동적 스트링으로 취급하고 코드를 JNI로 패스하기 전에 iconv()를 사용하여 처리하는 것이 용이합니다. 동적 스트링에 대한 자세한 정보는 동적 스트링을 참조하십시오.

예를 들어, 이름이 java/lang/String인 클래스를 찾으려면 코드는 다음과 같아야 합니다.

```
#pragma convert(819)
myClass = (*env)->FindClass(env,"java/lang/String");
#pragma convert(0)
```

숫자 819를 갖는 첫 번째 pragma는 컴파일러에게 ASCII로 된 모든 후속 이중 따옴표 스트링(리터럴 스트링)을 저장하도록 지시합니다. 숫자 0을 갖는 두 번째 pragma는 이중 따옴표 스트링에 대해 컴파일러의 디폴트 코드 페이지로 리턴시키도록 지시하며, 디폴트는 대개 EBCDIC 코드 페이지 37입니다. 따라서, pragmas에 의한 호출을 괄호로 묶어서 스트링 매개변수를 UTF-8로 코드화하는 JNI 요구사항을 만족시킵니다.

주의: 텍스트 대체에 주의하십시오. 예를 들어, 코드가 다음과 같다면,

```
#pragma convert(819)
#define MyString "java/lang/String"
#pragma convert(0)
myClass = (*env)->FindClass(env,MyString);
```

컴파일시 MyString 값이 FindClass 호출로 대체되므로 결과 스트링은 EDCDIC입니다. 대체 시에는, pragma, 번호 819는 효력이 없습니다. 그러므로, 리터럴 스트링은 ASCII로 저장되지 않습니다.

EBCDIC, 유니코드 및 UTF-8 사이의 동적 스트링 변환: 런타임에 계산되는 스트링 변수를 조작하려면 스트링을 EBCDIC, 유니코드 및 UTF-8 사이에서 변환할 필요가 있습니다.

코드 페이지 변환 기능에 제공되는 시스템 API는 iconv()입니다. iconv()를 사용하려면 다음 단계를 따르십시오.

1. QtqIconvOpen()로 변환 설명자를 작성하십시오.
2. 스트링으로 변환하기 위해 설명자를 사용하려면 iconv()를 호출하십시오.
3. iconv_close를 사용하여 설명자를 닫으십시오.

원시 메소드 예에 대해 Java(TM) 원시 인터페이스를 사용한 예 3에서, 루틴은 루틴내에서 iconv 변환 설명자를 작성하고 사용한 다음 제거합니다. 이 스키마는 한 iconv_t 설명자가 멀티스레드를 사용하는 문제점을 방지하지만 성능적 측면이 중요한 코드의 경우 정적 기억장치에서 변환 설명자를 작성하고, 상호 배제(mutex) 또는 다른 동기화 기능을 사용하여 설명자에 대한 멀티액세스를 조정하는 것이 훨씬 좋습니다.

예: 원시 메소드에 대한 Java 원시 인터페이스 사용

이 예 프로그램은 C 원시 메소드가 ″Hello, World″를 표시하는데 사용되는 간단한 JNI(Java(TM) Native Interface) 예입니다. javah 툴을 NativeHello 클래스 파일과 함께 사용하여 NativeHello.h 파일을 생성하십시오. 이 예는 NativeHello C 구현이 NATHELLO라는 서비스 프로그램의 일부라고 가정합니다.

주: NATHELLO 서비스 프로그램이 위치한 라이브러리는 이 예를 실행하기 위한 라이브러리 리스트에 있어야 합니다.

예 1: NativeHello 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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...");
    }
}
```

예 2: 생성된 NativeHello.h 헤더 파일

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
/* 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
```

이 NativeHello.c 예는 C에서 원시 메소드 구현을 보여줍니다. 이 예는 Java를 원시 메소드에 연결하는 방법을 보여줍니다. 그러나 이것은 iSeries 서버가 내부적으로 EBCDIC(확장 2진화 십진 변환 코드) 기계라고 하는 사실 때문에 야기되는 문제점을 지적합니다. 또한 JNI에 사실상의 국제화 요소가 현재 부족하기 때문에 야기되는 문제점도 보여줍니다.

이와 같은 이유가 JNI에서 새로운 것은 아니지만 사용자가 작성하는 C 코드에서 몇 가지 iSeries 서버 특유의 고유한 차이를 유발합니다. stdout 또는 stderr에 기록하거나 stdin에서 읽을 때 자료는 EBCDIC 형식으로 코드화되어야 합니다.

C 코드에서 7비트 문자만을 포함하는 대부분의 리터럴 스트링을 JNI가 요구하는 UTF-8 형식으로 쉽게 변환할 수 있습니다. 이렇게 하려면 코드 페이지 변환 pragmas로 리터럴 스트링을 괄호로 묶으십시오. 그러나 정보를 C 코드에서 직접 stdout 또는 stderr로 기록하려는 경우 일부 리터럴을 EBCDIC로 남겨둘 수 있습니다.

주: #pragma convert(0)문은 문자 자료를 EBCDIC로 변환합니다. #pragma convert(819)문은 문자 자료를 ASCII(미국 표준 정보 교환 코드)로 변환합니다. 이 명령문은 컴파일시에 C 프로그램에 있는 문자 자료를 변환합니다.

예 3: NativeHello Java 클래스의 NativeHello.c 원시 메소드 구현

주: 중요한 법률 사항에 대해서는 코드 예 면책사항을 참조하십시오.

```
#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 euc-kr 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)
```

백그라운드 정보에 대해서는 자세한 정보는 원시 메소드에 Java 원시 인터페이스 사용을 참조하십시오.

Java용 IBM OS/400 PASE 원시 메소드

iSeries JVM(Java(TM) virtual machine)은 현재 OS/400(R) PASE 환경에서 실행되는 원시 메소드 사용을 지원합니다. V5R2 이전에 원시 iSeries JVM에서는 ILE 원시 메소드만을 사용했습니다. OS/400 PASE 원시 메소드에 대한 지원에는 다음이 포함됩니다.

- OS/400 PASE 원시 메소드에서 원시 iSeries JNI(Java Native Interface)를 사용합니다.
- 원시 iSeries JVM에서 OS/400 PASE 원시 메소드를 호출할 수 있습니다.

이 새 지원을 사용하면 AIX(R)에서 실행되는 Java 어플리케이션을 사용자의 iSeries 서버로 쉽게 포팅할 수 있습니다. 클래스 파일과 AIX 원시 메소드 라이브러리를 iSeries의 통합 파일 시스템으로 복사하고 CL, Qshell 또는 OS/400 PASE 터미널 세션 명령 프롬트에서 이들을 실행할 수 있습니다.

Java용 IBM OS/400 PASE 원시 메소드 사용에 대해서는 다음 주제를 참조하십시오.

Java OS/400 PASE 환경 변수
OS/400 PASE 원시 메소드를 사용하기 전에 정의해야 하는 환경 변수에 대해 배웁니다. 이들 환경 변수는 OS/400 PASE 및 JVM 런타임 환경을 관리합니다.

Java OS/400 PASE 오류 코드
OS/400 PASE 원시 메소드의 문제점 해결을 위해서는, OS/400 작업 로그 메세지 및 Java 런타임 예외가 표시하는 오류 조건을 찾으십시오.

원시 메소드 라이브러리 관리
Java 라이브러리 명명 규칙 및 라이브러리 탐색 알고리즘에 대해 찾으십시오. 이 정보는 iSeries 서버에서 여러 버전의 원시 메소드 라이브러리를 관리하기 위해서는 중요합니다.

예: Java용 IBM OS/400 PASE 원시 메소드
Java 스트링의 내용을 인쇄하는 간단한 Java 프로그램을 실행하는 방법을 보여줍니다. Java 코드에서 직접 스트링을 액세스하는 대신, 이 예에서는 JNI를 통해 Java 내로 다시 콜백되는 원시 메소드를 호출하여 스트링 값을 구합니다.

이 정보에서는 사용자가 이미 OS/400 PASE에 능숙한 것으로 가정합니다. 자세한 정보는 다음 주제를 참조하십시오.

OS/400 PASE

Java OS/400 PASE 환경 변수
JVM(Java virtual machine)에서는 다음 변수를 사용하여 OS/400 PASE 환경을 시작합니다. Java용 IBM OS/400 PASE 원시 메소드 예를 실행하려면 QIBM_JAVA_PASE_STARTUP 변수를 설정해야 합니다.

예의 환경 변수 설정에 대해서는 다음 주제를 참조하십시오.

IBM OS/400 PASE 예의 환경 변수.

QIBM_JAVA_PASE_STARTUP

다음 조건에 모두 해당되면 이 환경 변수를 설정해야 합니다.

- OS/400 PASE 원시 메소드를 사용하는 중입니다.
- iSeries 명령 프롬트나 Qshell 명령 프롬트에서 Java를 시작합니다.

JVM은 이 환경 변수를 사용하여 PASE 환경을 시작합니다. 변수값은 OS/400 PASE 시작 프로그램을 나타냅니다. iSeries 서버에는 2개의 OS/400 PASE 시작 프로그램이 들어 있습니다.

- /usr/lib/start32: 32비트 OS/400 PASE 환경을 시작합니다.
- /usr/lib/start64: 64비트 OS/400 PASE 환경을 시작합니다.

OS/400 PASE 환경에서 사용되는 모든 공유 라이브러리 오브젝트의 비트 형식은 OS/400 PASE 환경의 비트 형식과 일치해야 합니다.

OS/400 PASE 단말기 세션에서 Java를 시작할 때 이 변수를 사용할 수 없습니다. OS/400 PASE 단말기 세션은 항상 32비트 OS/400 PASE 환경을 사용합니다. OS/400 PASE 단말기 세션에서 시작된 모든 JVM은 단말기 세션과 같은 유형의 PASE 환경을 사용합니다.

QIBM_JAVA_PASE_CHILD_STARTUP

2차 JVM의 OS/400 PASE 환경 변수가 1차 JVM의 OS/400 PASE 환경과 다를 경우 이 선택적 환경 변수를 설정하십시오. Java에서 Runtime.exec() 호출은 2차(또는 하위) JVM을 시작합니다.

자세한 정보는 QIBM_JAVA_PASE_CHILD_STARTUP 사용을 참조하십시오.

예: IBM OS/400 PASE 환경 변수 예: Java용 IBM OS/400 PASE 원시 메소드 예를 사용하려면 다음 환경 변수를 설정해야 합니다.

PASE_LIBPATH

iSeries 서버는 이 OS/400 PASE 환경 변수를 사용하여 OS/400 PASE 원시 메소드 라이브러리의 위치를 식별합니다. 단일 디렉토리 또는 복수 디렉토리로의 경로를 설정할 수 있습니다. 복수 디렉토리의 경우, 콜론(:)을 사용하여 항목을 분리하십시오. 서버는 또한 LIBPATH 환경 변수를 사용할 수도 있습니다.

이 예에서 Java, 원시 메소드 라이브러리 및 PASE_LIBPATH 사용에 대해서는 다음 주제를 참조하십시오.

Java, OS/400 PASE 및 원시 메소드 라이브러리 사용

PASE_THREAD_ATTACH

이 OS/400 PASE 환경 변수를 Y로 설정하면 OS/400 PASE 프로시듀어를 호출할 때 OS/400 PASE가 시작하지 않았던 ILE 스레드가 자동으로 OS/400 PASE에 첨부됩니다.

OS/400 PASE 환경 변수에 대해서는 다음 주제의 해당 항목을 참조하십시오.

OS/400 PASE 환경 변수에 대한 작업

QIBM_JAVA_PASE_STARTUP

JVM은 이 환경 변수를 사용하여 OS/400 PASE 환경을 시작합니다. 변수 값은 OS/400 PASE 시작 프로그램을 나타냅니다.

자세한 정보는 다음 주제를 참조하십시오.

Java OS/400 PASE 변수

QIBM_JAVA_PASE_CHILD_STARTUP 사용: QIBM_JAVA_PASE_CHILD_STARTUP 환경 변수는 모든 2차 JVM에 대한 OS/400 PASE 시작 프로그램을 지시합니다. 다음 조건 모두가 참일 때 QIBM_JAVA_PASE_CHILD_STARTUP을 사용하십시오.

- 실행하려는 Java 어플리케이션은 Runtime.exec()로의 Java 호출을 통해 JVM(Java virtual machines)을 작성합니다.
- 1차 및 2차 JVM 모두는 OS/400 PASE 원시 메소드를 사용합니다.
- 2차 JVM의 OS/400 PASE 환경은 1차 JVM의 OS/400 PASE 환경과 달라야 합니다.

이전에 나열한 조건 모두가 참일 경우, 다음 조치를 수행하십시오.

- QIBM_JAVA_PASE_CHILD_STARTUP 환경 변수를 2차 JVM의 OS/400 PASE 시작 프로그램으로 설정하십시오.
- iSeries 명령 프롬트나 Qshell 명령 프롬트에서 1차 JVM을 시작할 때, QIBM_JAVA_PASE_STARTUP 환경 변수를 1차 JVM의 OS/400 PASE 시작 프로그램으로 설정하십시오.

  주: OS/400 PASE 단말기 세션에서 1차 JVM을 시작할 때 QIBM_JAVA_PASE_STARTUP을 설정하지 마십시오.

2차 JVM 프로세스는 QIBM_JAVA_PASE_CHILD_STARTUP 환경 변수를 승계합니다. 뿐만 아니라 OS/400은 2차 JVM 프로세스의 QIBM_JAVA_PASE_STARTUP 환경 변수를 상위 프로세스의 QIBM_JAVA_PASE_CHILD_STARTUP 환경 변수값으로 설정합니다.

다음 표는 여러 명령 환경 조합과 QIBM_JAVA_PASE_STARTUP 및 QIBM_JAVA_PASE_CHILD_STARTUP의 정의에 대한 결과 OS/400 PASE 환경을 식별합니다.

시작 환경			결과 동작	
명령 환경	**QIBM_JAVA_PASE_STARTUP**	**QIBM_JAVA_PASE_CHILD_STARTUP**	**1차 JVM OS/400 PASE 시작**	**2차 JVM OS/400 PASE 시작**
CL 또는 QSH	정의된 startX	정의된 startY	startX 사용	startY 사용
CL 또는 QSH	정의된 startX	정의되지 않음	startX 사용	startX 사용
CL 또는 QSH	정의되지 않음	정의된 startY	OS/400 PASE 환경이 아님	startY 사용
CL 또는 QSH	정의되지 않음	정의되지 않음	OS/400 PASE 환경이 아님	OS/400 PASE 환경이 아님
OS/400 PASE 단말기 세션	정의된 startX	정의된 startY	허용되지 않음*	허용되지 않음*
OS/400 PASE 단말기 세션	정의된 startX	정의되지 않음	허용되지 않음*	허용되지 않음*
OS/400 PASE 단말기 세션	정의되지 않음	정의된 startY	OS/400 PASE 단말기 세션 환경 사용	startY 사용
OS/400 PASE 단말기 세션	정의되지 않음	정의되지 않음	OS/400 PASE 단말기 세션 환경 사용	OS/400 PASE 환경이 아님

* '허용되지 않음'으로 표시된 행은 QIBM_JAVA_PASE_STARTUP 환경 변수가 OS/400 PASE 단말기 세션과 충돌할 수 있는 상황을 나타냅니다. 상충 가능성이 있으므로, OS/400 PASE 단말기 세션에서는 QIBM_JAVA_PASE_STARTUP을 사용할 수 없습니다.

원시 메소드 라이브러리 관리

원시 메소드 라이브러리를 사용하려면(특히 iSeries 서버에서 원시 메소드 라이브러리의 여러 버전을 관리하려는 경우), Java 라이브러리 명명 규칙과 라이브러리 탐색 알고리즘 모두를 이해해야 합니다.

OS/400은 JVM(Java Virtual Machine)이 로드하는 라이브러리의 이름에 맞는 첫 번째 원시 메소드 라이브러리를 사용합니다. OS/400이 올바른 원시 메소드를 발견했는지 확인하기 위해서는, 라이브러리명이 손상되거나 JVM이 사용하는 원시 메소드 라이브러리가 혼동되지 않도록 해야 합니다.

OS/400 PASE 및 AIX Java 라이브러리 명명 규칙: Java 코드가 Sample 라이브러리를 로드하면, 해당 실행 파일의 이름은 libSample.a 또는 libSample.so이어야 합니다.

Java 라이브러리 탐색 순서: JVM용 OS/400 PASE 원시 메소드가 사용 가능할 경우, 서버는 3개의 다른 리스트(다음 순서로 된)를 사용하여 하나의 원시 메소드 라이브러리 탐색 경로를 작성합니다.

1. OS/400 라이브러리 리스트
2. LIBPATH 환경 변수
3. PASE_LIBPATH 환경 변수

탐색을 수행하기 위해 OS/400은 라이브러리 리스트를 통합 파일 시스템 형식으로 변환합니다. QSYS 파일 시스템 오브젝트는 통합 파일 시스템에서 동일한 이름을 갖지만, 일부 통합 파일 시스템 오브젝트는 동일한 QSYS 파일 시스템 이름을 갖지 않습니다. 라이브러리 로더가 QSYS 파일 시스템 및 통합 파일 시스템 모두에서 오브젝트를 검색하므로, OS/400은 통합 파일 시스템 형식을 사용하여 원시 메소드 라이브러리를 탐색합니다.

다음 표에서는 OS/400이 라이브러리에 있는 항목을 통합 파일 시스템으로 변환시키는 방법을 보여줍니다.

라이브러리 리스트 항목	통합 파일 시스템 형식
QSYS	/qsys.lib
QSYS2	/qsys.lib/qsys2.lib
QGPL	/qsys.lib/qgpl.lib
QTEMP	/qsys.lib/qtemp.lib

예: Sample2 라이브러리 탐색

다음 예에서, LIBPATH는 /home/user1/lib32:/samples/lib32로 설정되고, PASE_LIBPATH는 /QOpenSys/samples/lib로 설정됩니다.

다음 표는 위에서 아래로 읽어갈 때 전체 탐색 경로를 나타냅니다.

소스	통합 파일 시스템 디렉토리
라이브러리 리스트	/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

주: 대문자와 소문자는 /QOpenSys 경로에서만 유효합니다.

Sample2 라이브러리를 탐색하기 위해, Java 라이브러리 로더는 다음 순서로 파일 후보를 탐색합니다.

1. /qsys.lib/sample2.srvpgm
2. /qsys.lib/libSample2.a
3. /qsys.lib/libSample2.so

1. /qsys.lib/qsys2.lib/sample2.srvpgm
2. /qsys.lib/qsys2.lib/libSample2.a
3. /qsys.lib/qsys2.lib/libSample2.so

1. /qsys.lib/qgpl.lib/sample2.srvpgm
2. /qsys.lib/qgpl.lib/libSample2.a
3. /qsys.lib/qgpl.lib/libSample2.so

1. /qsys.lib/qtemp.lib/sample2.srvpgm
2. /qsys.lib/qtemp.lib/libSample2.a
3. /qsys.lib/qtemp.lib/libSample2.so

1. /home/user1/lib32/sample2.srvpgm
2. /home/user1/lib32/libSample2.a
3. /home/user1/lib32/libSample2.so

1. /samples/lib32/sample2.srvpgm
2. /samples/lib32/libSample2.a
3. /samples/lib32/libSample2.so

1. /QOpenSys/samples/lib/SAMPLE2.srvpgm
2. /QOpenSys/samples/lib/libSample2.a
3. /QOpenSys/samples/lib/libSample2.so

OS/400은 실재하는 리스트의 첫 번째 후보를 원시 메소드 라이브러리로서 JVM에 로드합니다. '/qsys.lib/libSample2.a' 및 '/qsys.lib/libSample2.so' 같은 후보가 검색에서 발견되더라도, /qsys.lib 디렉토리에서 통합 파일 시스템 파일이나 기호 링크를 작성할 수 없습니다. 그러므로 OS/400이 이들 후보 파일들을 검사하더라도, /qsys.lib로 시작하는 통합 파일 시스템 디렉토리에서 이들을 찾지 못합니다.

그러나 기타 통합 파일 시스템 디렉토리에서 QSYS 파일 시스템에 있는 OS/400 오브젝트로의 기호 링크를 임의로 작성할 수 있습니다. 그 결과 유효한 파일 후보에는 /home/user1/lib32/sample2.srvpgm 같은 파일이 포함됩니다.

Java OS/400 PASE 오류 코드

다음 리스트에서는 Java용 OS/400 PASE 원시 메소드를 시작할 때나 실행할 때 발생하는 오류에 대해 설명합니다.

시작 오류: JVAB55C "JVM을 작성할 수 없음" 메세지에 대해서는 3개의 새 오류 코드가 있습니다.

- 19 - OS/400 PASE 환경 시작 오류. 사용자 어플리케이션이나 오퍼레이팅 시스템 문제점을 나타냅니다.

  오류 코드 19에는 영어 전용 텍스트도 포함됩니다. 다음 오류 텍스트를 보게 될 수도 있습니다.

  – `Java OS/400 PASE 오류. OS/400 PASE가 이미 활동 중이고, QIBM_JAVA_PASE_STARTUP 환경 변수가 정의되었습니다.`

    QIBM_JAVA_PASE_STARTUP 환경 변수 정의를 제거하거나, 활동 중인 OS/400 PASE 단말기 세션을 종료하십시오.

  – `Java OS/400 PASE 오류. OS/400 PASE 시작 프로그램 &programName을 실행할 수 없습니다.`

    QIBM_JAVA_PASE_STARTUP 환경 변수에서 식별한 OS/400 PASE 프로그램이 없거나, OS/400 PASE 환경에서 프로그램을 실행할 수 없습니다.

  – `Java OS/400 PASE 내부 오류 번호 &errorCode.`

    다음 내부 오류 번호를 보게 될 수도 있습니다.

    - 106 - OS/400 PASE 또는 지정된 OS/400 PASE 형식이 표시된 JDK 버전에서 지원되지 않습니다.

      OS/400 PASE가 지정된 JDK에 대해 지원되지 않거나, OS/400 PASE 시작 프로그램 비트 형식이 지정된 JDK에서 지원되지 않습니다. V5R2의 경우, 지원되는 조합은 다음과 같습니다.

      - JDK 1.2 및 OS/400 PASE 32비트 형식
      - JDK 1.3 및 OS/400 PASE 32비트 형식
      - JDK 1.3 및 OS/400 PASE 64비트 형식

    다음 오류 코드를 서비스 담당자에게 알리십시오.

    - 101 - 식별된 시작 프로그램이 없습니다.
    - 102 - OS/400 PASE JavaVM 포인터를 검색할 수 없습니다.
    - 103 - Qp2CallPase를 찾을 수 없습니다.
    - 104 - OS/400 PASE 포인터 크기 오류.
    - 105 - OS/400 PASE libjvm.a를 찾을 수 없습니다.

- 20 - OS/400 PASE 피연산자가 유효하지 않습니다. 서비스 담당자에게 알리십시오.
- 21 - OS/400 PASE에 작업을 접속할 수 없습니다. 서비스 담당자에게 알리십시오.

런타임 오류: 시작 오류 뿐만 아니라, PaseInternalError 또는 PaseExit Java 예외가 JVM의 Qshell 출력에서 나타날 수 있습니다.

- PaseInternalError - 내부 시스템 오류를 나타냅니다. 사용권 내부 코드 기록부 항목을 검사하십시오. 자세한 내용은 Qp2CallPase를 참조하십시오.
- PaseExit - OS/400 PASE 어플리케이션이 exit() 함수를 호출했거나 OS/400 PASE 환경이 비정상적으로 종료되었습니다. 추가 정보는 작업 기록부 및 사용권 내부 코드 기록부를 참조하십시오.

예: Java용 IBM OS/400 PASE 원시 메소드

Java용 IBM OS/400 PASE 원시 메소드 예에서는 JNI(Java Native Interface)를 사용하여 Java 코드를 콜백하는 원시 C 메소드의 인스턴스를 호출합니다.

예 소스 파일의 HTML 버전을 보려면, 다음 링크를 사용하십시오.

- PaseExample1.java
- PaseExample1.c

OS/400 PASE 원시 메소드 예를 실행하려면, 다음 타스크를 완료해야 합니다.

1. AIX 워크스테이션으로 예 소스 코드 다운로드
2. 예 소스 코드 준비
3. iSeries 서버 준비

Java용 OS/400 PASE 원시 메소드 예 실행: 이전 타스크를 완료하면 예를 실행할 수 있습니다. 다음 명령을 사용하여 예 프로그램을 실행하십시오.

- iSeries 서버 명령 프롬트에서,

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- Qshell 명령 프롬트나 OS/400 PASE 단말기 세션에서,

```
cd /home/example
java PaseExample1
```

통합 언어 환경과 Java 비교

iSeries 서버의 Java(TM) 환경은 ILE(integrated language environment)와 다릅니다. Java는 ILE 언어가 아니며 iSeries 서버에서 프로그램이나 서비스 프로그램을 작성하기 위해 ILE 오브젝트 모듈에 바인드할 수 없습니다.

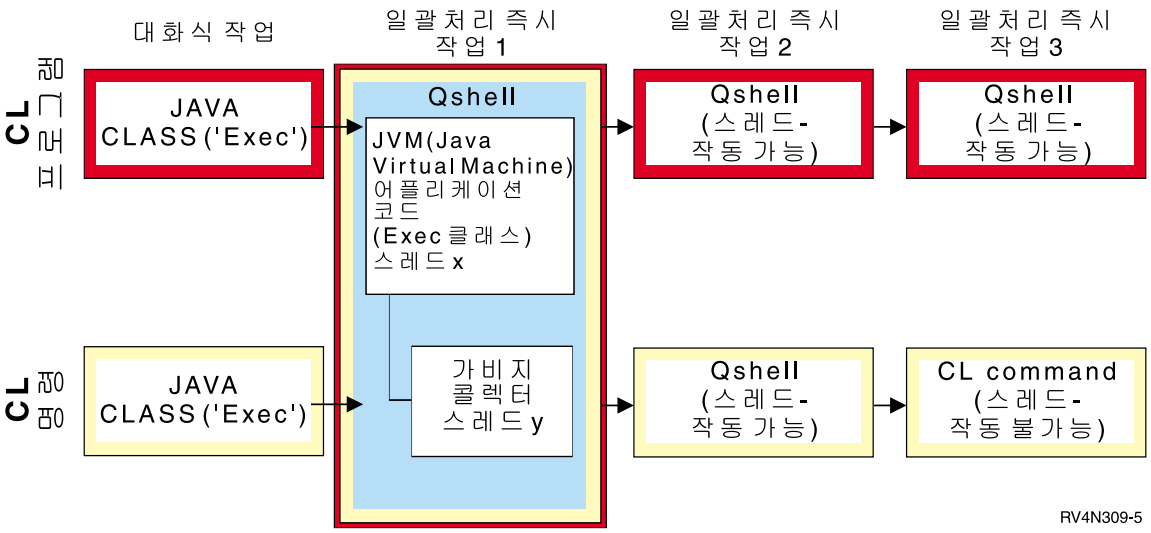
ILE	Java
iSeries 서버 저장 소스 코드 상의 라이브러리 또는 파일 구조의 일부를 구성하는 멤버입니다.	통합 파일 시스템의 스트림 파일이 소스 코드를 포함합니다.
SEU(소스 입력 유틸리티)는 ebcdic(확장 2진화 십진 변환 코드) 소스 파일을 편집합니다.	대개 워크스테이션 편집기를 사용하여 ASCII(미국 표준 정보 교환 코드) 소스 파일을 편집합니다.
소스 파일은 iSeries 서버의 라이브러리에 저장되는 오브젝트 코드 모듈로 컴파일됩니다.	소스 코드는 통합 파일 시스템이 저장하는 클래스 파일로 컴파일됩니다.
오브젝트 모듈은 프로그램 또는 서비스 프로그램에 함께 정적으로 바인드됩니다.	클래스는 런타임에 필요에 따라 동적으로 로드됩니다.
다른 ILE 프로그래밍 언어로 작성된 기능을 직접 호출할 수 있습니다.	Java에서 다른 언어를 호출하려면 Java 고유 인터페이스를 반드시 사용해야 합니다.
ILE 언어가 항상 컴파일되고 기계 명령어로 수행됩니다.	Java 프로그램을 해석하거나 컴파일할 수 있습니다.

java.lang.Runtime.exec() 사용

java.lang.Runtime.exec() 메소드는 Java(TM) 프로그램내의 프로그램이나 명령을 호출합니다. 실제 처리는 exec() 메소드로 전달되는 정보에 기초하여 발생합니다. 모든 경우에, Runtime.exec() 메소드는 스레드 가능한 다른 일괄처리 즉시(BCI) 작업을 작성합니다. BCI 작업은 Runtime.exec() 메소드에서 전달되는 명령 스트링을 처리합니다.

java.lang.Runtime.exec() 메소드를 사용하려면, JDK 1.2 이하를 사용하는 경우 iSeries 서버에 Qshell 인터프리터를 설치해야 합니다. Qshell 인터프리터에 대한 자세한 정보는 Qshell 인터프리터를 참조하십시오.

주: java.lang.Runtime.exec() 메소드는 별도의 프로세스에서 프로그램을 실행합니다. 이것은 동일한 프로세스에서 프로그램을 실행하는 C 시스템 함수와는 다릅니다.



처리 중인 명령이 Qshell 유틸리티인 경우 두 번째 BCI 작업에서 실행하며 세 번째 BCI 작업은 작성되지 않습니다. 처리 중인 명령이 CL 명령인 경우 Qshell을 실행하기 위해 두 번째 BCI 작업이 시작되고 CL 명령을 실행하기 위해 세 번째 BCI 작업이 시작됩니다. Qshell 유틸리티는 QSH를 직접 실행할 수 있는 내장 유틸리

티입니다. Qshell 유틸리티의 예는 Java 프로그램을 컴파일하는 javac 명령입니다. 두 번째(또는 세 번째) BCI 작업의 처리는 JVM(Java Virtual Machine)과 동시에 실행됩니다. 그 작업에서의 나감 또는 종료(shut down) 처리는 원래 JVM(Java Virtual Machine)에 영향을 주지 않습니다.

iSeries 명령이나 프로그램을 호출할 때 호출된 프로그램에 전달되는 매개변수가 해당 프로그램이 예상하는 코드 페이지에 있는지 확인해야 합니다.

java.lang.Runtime.exec() 예는 다른 Java 프로그램 호출, CL 프로그램 호출 또는 CL 명령 호출을 참조하십시오.

예: java.lang.Runtime.exec()로 다른 Java 프로그램 호출

이 예에서는 java.lang.Runtime.exec()로 다른 Java(TM) 프로그램을 호출하는 방법을 보여줍니다. 이 클래스는 IBM Developer Kit for Java의 일부로 제공되는 Hello 프로그램을 호출합니다. Hello 클래스가 System.out에 기록할 때 이 프로그램이 스트림에 대한 처리를 확보하고 읽을 수 있습니다.

주: Qshell 인터프리터를 사용하여 프로그램을 호출합니다.

예 1: CallHelloPgm 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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
```

자세한 백그라운드 정보는 java.lang.Runtime.exec() 사용법을 참조하십시오.

예: java.lang.Runtime.exec()으로 CL 프로그램 호출

이 예는 Java(TM) 프로그램 내에서 CL 프로그램의 실행 방법을 보여줍니다. Java 프로그램 내에서 CL 명령을 호출하는 방법에 대한 예는 CL 명령 호출을 참조하십시오. 이 예에서, Java 클래스 CallCLPgm이 CL 프로그램을 실행합니다. CL 프로그램은 Hello 클래스 파일과 연관된 프로그램을 표시하기 위해 DSPJVAPGM(Java 프로그램 표시) 명령을 사용합니다. 이 예에서는 CL 프로그램이 컴파일되어 JAVSAMPLIB 라이브러리에 있는 것으로 가정합니다. CL 프로그램의 결과는 QSYSPRT 스풀 파일에 있습니다.

주: JAVSAMPLIB는 IBM Developer Kit LP(사용권 프로그램) 번호 5722-JV1 설치 프로세스의 일부로 작성되지 않습니다. 반드시 명시적으로 라이브러리를 작성해야 합니다.

예 1: CallCLPgm 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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
```

예 2: Java CL 프로그램 표시

```
PGM
             DSPJVAPGM  CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
                          OUTPUT(*PRINT)
ENDPGM
```

자세한 내용은 java.lang.Runtime.exec() 사용법을 참조하십시오.

예: java.lang.Runtime.exec()으로 CL 명령 호출

이 예에서는 Java 프로그램 내에서 CL(제어 언어) 명령을 실행하는 방법을 보여줍니다. 이 예에서 Java 프로그램이 CL 명령을 실행합니다. CL 명령은 DSPJVAPGM(Java 프로그램 표시) 명령을 사용하여 Hello 클래스 파일과 연관된 프로그램을 표시합니다. CL 명령의 결과는 QSYSPRT 스풀 파일에 있습니다.

JDK 1.1.8 또는 JDK 1.2를 사용 중인 경우,

`Runtime.getRuntime().exec()` 함수에 전달되는 모든 명령은 인용 부호로 묶이고 Qshell 형식으로 되어 있어야 합니다. 또한 CL 명령을 Qshell에서 실행하기 위해 스트링에 전달해야 합니다.

```
"system \"CL COMMAND\""
```

*CL COMMAND*가 있는 곳이 CL 명령을 실행하는 위치입니다. 따라서 MYCLCOM 명령을 호출하기 위한 행은 다음과 같아야 합니다.

```
Runtime.getRuntime()Exec("system \"MYCLCOM\"");
```

주: JDK 1.3 또는 JDK 1.4를 사용 중인 경우, 슬래쉬와 인용 분리문자(\")는 생략하십시오. 예를 들어 JDK 버전 1.3 이상을 사용 중인 경우, MYCLCOM 명령에 대한 호출은 다음과 같습니다.

```
Runtime.getRuntime()Exec("system MYCLCOM");
```

자세한 정보는, Java 2 Software Development Kit에 대한 Java 시스템 등록 정보, 표준판에 있는 os400.runtime.exec를 참조하십시오.

예 1: CallCLCom 클래스

다음 예에서는 사용자가 JDK 1.1.8 또는 JDK 1.2를 사용 중인 경우, 필요한 Qshell 분리문자를 사용합니다. JDK 버전 1.3 이상을 사용 중인 경우, 분리문자를 생략하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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
```

자세한 내용은 java.lang.Runtime.exec() 사용법을 참조하십시오.

프로세스간 통신

다른 프로세스에서 실행 중인 프로그램과 통신할 때 다음 몇 가지 옵션이 있습니다.

한 가지 옵션은 프로세스간 통신에 대해 소켓을 사용하는 것입니다. 한 프로그램이 서버 프로그램 역할을 하여 소켓 연결시 클라이언트 프로그램의 입력을 청취합니다. 클라이언트 프로그램은 소켓을 사용하여 서버에 연결합니다. 일단 소켓 연결이 설정되면 각 프로그램이 정보를 송수신할 수 있습니다.

다른 옵션은 프로그램 사이의 통신에 대해 스트림 파일을 사용하는 것입니다. 이렇게 하려면 System.in, System.out 및 System.err 클래스를 사용하십시오.

세 번째 옵션은 자료 대기행렬과 iSeries 메세지 오브젝트를 제공하는 IBM Toolbox for Java(TM)를 사용하는 것입니다.

다른 언어에서 Java를 호출할 수 있습니다. 자세한 정보는 예: C에서 Java 호출 및 예: RPG에서 Java 호출을 참조하십시오.

프로그램간 통신을 위해 소켓 사용

소켓 스트림은 개별 프로세스에서 실행 중인 프로그램 사이에서 통신합니다. 프로그램은 개별적으로 시작되거나, 기본 Java(TM) 프로그램에서 java.lang.Runtime.exec() 메소드를 사용하여 시작될 수 있습니다. 프로그램을 Java 이외의 언어로 작성하는 경우 ASCII(미국 표준 정보 교환 코드) 또는 EDCDIC(확장 2진화 십진 변환 코드) 변환이 발생합니다. 자세한 내용은 Java 문자 코드화를 참조하십시오.

소켓을 사용하는 예는 예: 프로그램간 통신을 위해 소켓 사용을 참조하십시오

예 : 프로세스간 통신을 위한 소켓 사용: 이 예에서는 소켓을 사용하여 Java(TM) 프로그램과 C 프로그램 사이에서 통신합니다. 소켓에서 청취하는 C 프로그램을 먼저 시작해야 합니다. 일단 Java 프로그램이 소켓에 연

결되면 C 프로그램이 소켓 연결을 사용하여 Java 프로그램에 스트링을 송신합니다. C 프로그램에서 송신된 스트링은 코드 페이지가 819인 ASCII(미국 표준 정보교환 코드) 스트링입니다.

Qshell 인터프리터 명령 행이나 다른 Java 플랫폼에서 java TalkToC xxxxx nnnn 명령을 사용하여 Java 프로그램을 시작해야 합니다. 또는, iSeries 명령행에 JAVA TALKTOC PARM(xxxxx nnnn)을 입력하여 Java 프로그램을 시작하십시오. xxxxx는 C 프로그램을 실행하고 있는 시스템의 정의역명 또는 인터넷 프로토콜(IP) 주소입니다. nnnn은 C 프로그램이 사용하고 있는 소켓의 포트번호입니다. 또한 이 포트 번호를 C 프로그램을 호출할 때 첫 번째 매개변수로 사용해야 합니다.

예 1: TalkToC 클라이언트 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
import java.net.*;
import java.io.*;

class TalkToC
{
   private String host = null;
   private int port = -999;
   private Socket socket = null;
   private BufferedReader inStream = null;


   public static void main(String[] args)
   {
      TalkToC caller = new TalkToC();
      caller.host = args[0];
      caller.port = new Integer(args[1]).intValue();
      caller.setUp();
      caller.converse();
      caller.cleanUp();

   }  // end main() method

   public void setUp()
   {
      System.out.println("TalkToC.setUp() invoked");

       try
      {
         socket = new Socket(host, port);
         inStream = new BufferedReader(new InputStreamReader(
                                       socket.getInputStream()));
      }
      catch(UnknownHostException e)
      {
         System.err.println("Cannot find host called: " + host);
                e.printStackTrace();
         System.exit(-1);
      }
         catch(IOException e)
      {
         System.err.println("Could not establish connection for " + host);
                e.printStackTrace();
```

```
            System.exit(-1);
        }
    } // end setUp() method

    public void converse()
    {
        System.out.println("TalkToC.converse() invoked");

        if (socket != null && inStream != null)
        {
         try
           {
              System.out.println(inStream.readLine());
           }
           catch(IOException e)
           {
              System.err.println("Conversation error with host " + host);
                  e.printStackTrace();
           }

        } // end if

    }  // end converse() method

    public void cleanUp()
    {
         try
        {

           if(inStream != null)
           {
              inStream.close();
           }
           if(socket != null)
           {
              socket.close();
           }
        } // end try
           catch(IOException e)
        {
           System.err.println("Error in cleanup");
                  e.printStackTrace();
           System.exit(-1);
        }
    } // end cleanUp() method

} // end TalkToC class
```

SockServ.C는 포트 번호에 대한 매개변수에서 전달하여 시작됩니다. 예를 들면, CALL SockServ '2001'입니다.

예 2: SockServ.C 서버 프로그램

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
#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);

}
```

자세한 내용은 프로그램간 통신을 위해 소켓 사용을 참조하십시오.

프로세스간 통신에 입력 및 출력 스트림 사용

입력 및 출력 스트림은 개별 프로세스에서 실행 중인 프로그램 사이에서 통신합니다. java.lang.Runtime.exec() 메소드가 프로그램을 실행합니다. 상위 프로그램은 하위 프로세스 입력 및 출력 스트림에 대하여 처리할 수 있고 해당 스트림에 기록하거나 읽을 수 있습니다. 하위 프로그램이 Java(TM) 이외의 언어로 작성된 경우, ASCII(American Standard Code for Information Interchange) 또는 EBCDIC(extended binary-coded decimal interchange code) 변환이 발생하는지 확인해야 합니다. 자세한 내용은 Java 문자 코드화를 참조하십시오.

입력 및 출력 스트림을 사용하는 예에 대해서는 예: 프로세스간 통신을 위한 입력 및 출력 스트림 사용을 참조하십시오.

예: 프로세스간 통신을 위한 입력 및 출력 스트림 사용: 이 예에서는 Java(TM)에서 C 프로그램을 호출하는 방법과, 프로세스간 통신의 입/출력 스트림을 사용하는 방법을 보여줍니다. 이 예에서 C 프로그램은 표준 출력 스트림에 스트링을 기록하고, Java 프로그램이 이 스트링을 읽고 화면에 표시합니다. 이 예는 JAVSAMPLIB라고 명명된 라이브러리가 작성되었고 CSAMP1 프로그램이 그 라이브러리 안에서 작성되었다고 가정합니다.

주: JAVSAMPLIB는 IBM Developer Kit LP(사용권 프로그램) 번호 5722-JV1 설치 프로세스의 일부로 작성되지 않습니다. 사용자가 그것을 명시적으로 작성해야 합니다.

예 1: CallPgm 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
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
```

예 2: CSAMP1 C 프로그램

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
#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);

}
```

자세한 내용은 프로세스간 통신을 위한 입력 및 출력 스트림 사용을 참조하십시오.

예: C에서 Java 호출

이것은 Java Hello 프로그램을 호출하기 위해 system() 함수를 사용하는 C 프로그램의 예입니다.

예: C에서 Java 호출

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
#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')");
 }
```

예: RPG에서 Java 호출

이것은 QCMDEXC API를 사용하여 Java(TM) Hello 프로그램을 호출하는 RPG 프로그램의 예입니다.

예 1: RPG에서 Java 호출

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
     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 플랫폼

Java(TM) 플랫폼은 Java 애플릿과 어플리케이션을 개발하고 관리하기 위한 환경입니다. Java 언어, Java 패키지 및 JVM(Java Virtual Machine)의 세 가지 주요 구성요소로 구성되어 있습니다. Java 언어와 패키지는 C++ 및 클래스 라이브러리와 유사합니다. Java 패키지에는 해당 Java 구현에서 사용할 수 있는 클래스가 들어 있습니다. API(application programming interface)는 Java를 지원하는 모든 시스템에서 동일해야 합니다.

Java는 컴파일하고 실행하는 방법에 있어서 C++과 같은 종래의 언어와 다릅니다. 종래의 프로그래밍 환경에서는 프로그램의 소스 코드를 특정 하드웨어 및 오퍼레이팅 시스템에 대한 오브젝트 코드로 기록하고 컴파일합니다. 이 오브젝트 코드는 다른 오브젝트 코드 모듈로 바인드하여 실행 프로그램을 작성합니다. 이 코드는 특정 컴퓨터 하드웨어 세트마다 다르며 변경하지 않고 다른 시스템에서 실행할 수 없습니다. 이 그림은 종래의 언어 배치 환경을 보여줍니다.

Java 플랫폼을 효과적으로 사용하려면 다음을 참조하십시오.

Java 애플릿 및 어플리케이션
Java 애플릿을 기록하고 이미지를 포함시키는 방법과 동일한 방법으로 HTML 페이지에 포함시킬 수 있습니다. Java 가능 브라우저를 사용하여 애플릿이 포함된 HTML 페이지를 볼 때 애플릿의 코드를 시스템으로 전송하고 브라우저의 JVM(Java Virtual Machine)이 실행합니다. 웹 브라우저의 사용이 필요하지 않은 Java 어플리케이션도 기록할 수 있습니다.

JVM(Java Virtual Machine)
IBM(R) Operating System/400(R)(OS/400(R))과 같은 오퍼레이팅 시스템이나 웹 브라우저에 JVM을 내장시킬 수 있습니다. JVM은 Java 인터프리터와 Java 런타임 환경으로 구성되어 있습니다. 인터프리터는 클래스 파일을 해석하고 특정 하드웨어 플랫폼에서 Java 명령어를 실행하는 타스크를 수행합니다. JVM은 일단 Java 코드를 기록하고 컴파일한 후에 모든 플랫폼에서 실행할 수 있도록 합니다.

Java JAR 및 클래스 파일
Java 환경은 Java 컴파일러가 하드웨어 특정 명령어 세트에 대한 기계 코드를 생성하지 않는다는 점에서 다른 프로그래밍 환경과 다릅니다. 그 대신 Java 컴파일러는 Java 소스 코드를 Java 클래스 파일이 저장하는 JVM 명령어로 변환합니다. 클래스 파일을 저장하기 위해 JAR 파일을 사용할 수 있습니다. 클래스 파일은 특정 하드웨어 플랫폼을 목표로 지정하지 않고 JVM 구조를 목표로 지정합니다.

Java 스레드
Java는 멀티스레드 프로그래밍 언어이므로 JVM 내에서 한 번에 둘 이상의 스레드를 실행할 수 있습니다. Java 스레드는 Java 프로그램이 동시에 여러 타스크를 수행할 수 있는 방법을 제공합니다.

JDK(Java Development Kit)
JDK(Java Development Kit)는 Sun Microsystems, Inc.에서 Java 개발자를 위해 배포하는 소프트웨어입니다. 여기에는 Java 인터프리터, Java 클래스 및 Java 개발 툴이 들어 있습니다. JDK에 대해 다음의 정보를 찾으십시오.

- Java 패키지
- Java 툴

Java 애플릿 및 어플리케이션

애플릿은 HTML 웹 문서에 포함되도록 지정된 Java(TM) 프로그램입니다. HTML 문서에는 Java 애플릿의 이름과 URL(Uniform Resource Locator)을 지정하는 태그가 들어 있습니다. URL은 인터넷에서 애플릿 바이트코드가 상주하는 위치입니다. Java 애플릿 태그가 포함된 HTML 문서가 표시되면 Java 가능 웹 브라우저가 인터넷에서 Java 바이트 코드를 다운로드하고 JVM(Java Virtual Machine)을 사용하여 웹 문서 내에서 코드를 처리합니다. 이러한 Java 애플릿은 에니메이션 그래픽 또는 대화식 내용을 웹 페이지에 포함시킬 수 있는 것입니다.

자세한 정보는, Writting Applets  , Sun Microsystems의 Java 애플릿 학습을 참조하십시오. 여기에는 애플릿에 대한 개요, 애플릿 작성을 위한 지시사항 및 일부 공통 애플릿 문제점이 들어 있습니다.

어플리케이션은 브라우저의 사용이 필요하지 않은 독립형 프로그램입니다. 명령행에서 Java 인터프리터를 시작하고 컴파일된 어플리케이션이 포함된 파일을 지정하여 Java 어플리케이션을 실행합니다. 어플리케이션은 대개 배치된 시스템에 상주합니다. 어플리케이션은 시스템의 자원에 액세스하며 Java 보안 모델에 의해 제한됩니다.

JVM(Java Virtual Machine)

JVM(Java(TM) virtual machine)은 IBM Operating System/400(OS/400)과 같이, 사용자가 웹 브라우저나 다른 오퍼레이팅 시스템에 추가할 수 있는 런타임 환경입니다. JVM은 Java 컴파일러가 생성하는 명령을 실행합니다. 원래 시작된 플랫폼과 상관 없이 어느 플랫폼에서나 Java 클래스 파일(314 페이지 참조)을 실행할 수 있게 하는 바이트코드 인터프리터와 런타임으로 구성되어 있습니다.

Java 런타임의 일부인 클래스 로더 및 보안 관리자는 다른 플랫폼에서 온 코드를 분리합니다. 또한 로드된 각 클래스가 액세스하는 시스템 자원을 제한할 수 있습니다.

주: Java 어플리케이션은 제한하지 않습니다. 애플릿만 제한합니다. 어플리케이션은 시스템 자원에 자유롭게 액세스하여 원시 메소드를 사용할 수 있습니다. 대부분의 IBM Developer Kit for Java 프로그램은 어플리케이션입니다.

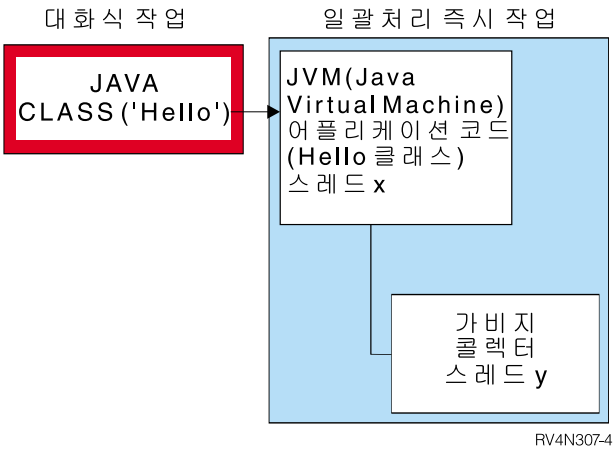
CRTJVAPGM(Java 프로그램 작성) 명령을 사용하여 Java 런타임이 바이트 코드를 확인하기 위해 요구하는 안전 사항을 코드가 충족시키는지 확인할 수 있습니다. 여기에는 유형 제한사항 실행, 자료 변환 검사, 매개변수 스택 넘침 또는 미달이 발생하지 않았는지 확인, 액세스 위반 검사가 포함됩니다. 그러나 바이트 코드를 명시적으로 확인할 필요는 없습니다. 미리 CRTJVAPGM 명령을 사용하지 않으면 클래스를 처음 사용할 때 검사가 이루어집니다. 바이트 코드를 확인했으면 인터프리터는 이 바이트 코드를 해독하고 원하는 조작을 수행하기 위해 필요한 기계 명령어를 실행합니다.

주: 313 페이지의 『Java 인터프리터』는 OPTIMIZE(*INTERPRET) 또는 INTERPRET(*YES)를 지정한 경우에만 사용합니다.

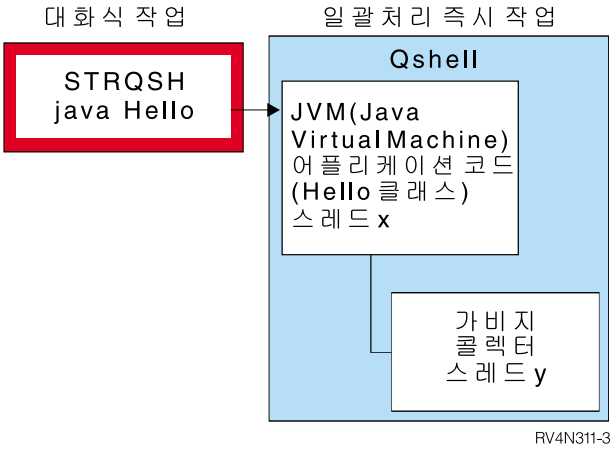
바이트코드의 로드 및 실행 이외에 JVM(Java Virtual Machine)에는 메모리를 관리하는 가비지 콜렉터가 있습니다. 가비지 콜렉션은 바이트코드의 로드 및 해석과 동시에 실행합니다.

Java 런타임 환경

Java 런타임 환경은 iSeries 명령행에 RUNJVA(Java 실행) 명령이나 JAVA 명령을 입력할 때마다 시작됩니다. Java 환경은 멀티스레드 환경이므로 일괄처리 즉시(BCI) 작업과 같은 스레드를 지원하는 작업에서 JVM(Java Virtual Machine)을 실행해야 합니다. 일단 JVM(Java Virtual Machine)이 시작되면 가비지 콜렉터가 실행하는 추가 스레드가 시작됩니다. 일반적인 Java 환경은 다음과 같습니다.



또한 Qshell 인터프리터로부터 Qshell의 `java` 명령을 사용하여 Java 런타임 환경을 시작할 수도 있습니다. 이 환경에서 Qshell 인터프리터는 대화식 작업과 연관되는 BCI 작업에서 실행됩니다. Java 런타임 환경은 Qshell 인터프리터를 수행하는 작업에서 시작됩니다.



Java 런타임 환경이 대화식 작업에서 시작할 때 Java Shell 화면이 표시됩니다. 이 화면은 System.in 스트림에 자료를 입력하고 System.out 및 System.err 스트림에 기록된 자료를 표시하기 위한 입력 행을 제공합니다.

Java 인터프리터

Java 인터프리터는 특정 하드웨어 플랫폼에 대해 Java 클래스 파일을 해석하는 JVM(Java Virtual Machine)의 일부입니다. Java 인터프리터는 각 바이트 코드를 해독하고 이 바이트코드에 대한 일련의 기계 명령어를 실행합니다.

Java JAR 및 클래스 파일

Java(TM) ARchive(JAR) 파일은 많은 파일들을 하나로 결합시키는 파일 형식입니다. JAR을 일반 아카이브 툴로 사용할 수 있으며 애플릿을 비롯한 모든 유형의 Java 프로그램을 분배하는 데에도 사용할 수 있습니다.

Java 애플릿은 각 조각에 대해 새로운 연결을 열지 않고 하나의 HTTP(Hypertext Transfer Protocol) 트랜잭션에서 브라우저로 다운로드합니다. 이러한 다운로드 방법은 애플릿이 웹 페이지를 로드하고 기능을 시작하는 속도를 향상시킵니다.

JAR은 플랫폼들 사이에서 유일한 아카이브 형식입니다. JAR은 오디오 파일 및 이미지 파일을 클래스 파일과 함께 처리하는 유일한 형식입니다. JAR은 Java로 기록된 완전 확장 가능한 형식의 개방적인 표준입니다.

JAR 형식은 파일의 크기를 줄이고 다운로드 시간을 감소시키는 압축을 지원합니다. 또한, 애플릿 작성자는 JAR 파일의 개별 항목에 디지털로 서명하여 기점을 인증할 수 있습니다.

JAR 파일의 클래스를 갱신하려면 Java jar 툴을 참조하십시오.

Java 클래스 파일은 Java 컴파일러가 소스 파일을 컴파일할 때 작성되는 스트림 파일입니다. 클래스 파일에는 각 필드와 클래스의 메소드를 설명하는 표가 들어 있습니다. 파일에는 각 메소드에 대한 바이트코드, 정적 자료 및 Java 오브젝트를 나타내기 위해 사용하는 설명도 들어 있습니다.

Java 스레드

스레드는 프로그램 내에서 실행하는 하나의 독립적인 스트림입니다. Java(TM)는 멀티스레드 프로그래밍 언어이므로, 한번에 하나 이상의 스레드가 JVM내에서 실행될 수 있습니다. Java 스레드는 Java 프로그램이 동시에 여러 타스크를 수행할 수 있는 방법을 제공합니다. 스레드는 원래 프로그램에서의 제어 흐름입니다.

스레드는 현재 프로그램을 지원하고 어플리케이션의 성능과 확장성을 향상시키기 위해 사용하는 현대적인 프로그래밍 구조입니다. 대부분의 프로그래밍 언어는 추가 기능 프로그래밍 라이브러리를 사용하여 스레드를 지원합니다. Java는 내장 API(어플리케이션 프로그램 인터페이스)로 스레드를 지원합니다.

주: 스레드의 사용은 대화성을 증가시키기 위한 지원을 제공하며 보다 많은 타스크를 동시에 실행하기 때문에 키보드에서 기다리는 시간이 줄어들었음을 의미합니다. 그러나 프로그램에 단지 스레드가 있기 때문에 프로그램의 대화성이 증가하는 것은 아닙니다.

스레드는 계속해서 프로그램이 다른 작업을 처리하도록 허용하는 동시에 장기 수행 중인 대화를 기다리는 메카니즘입니다. 스레드는 동일한 코드 스트림을 통해 여러 흐름을 지원하는 능력을 가지고 있습니다. 때로는 **간단한 프로세스**라고도 합니다. Java 언어에는 스레드에 대한 직접 지원이 포함됩니다. 그러나 설계 상 인터럽트를 사용하거나 여러 번 대기하는 비동기 비블록화 입력 및 출력을 지원하지 않습니다.

스레드를 사용하면 기계에 프로세서가 많은 환경에서 확장 및 축소가 가능한 병렬 프로그램을 개발할 수 있습니다. 적절하게 구성하면 여러 트랜잭션 및 사용자를 처리하기 위한 모델도 제공합니다.

여러 가지 상황에 대해 Java 프로그램에서 스레드를 사용할 수 있습니다. 일부 프로그램은 여러 활동에 참여할 수 있어야 하며 계속 사용자의 추가 입력에 응답할 수 있어야 합니다. 예를 들어, 웹 브라우저는 음향을 재생하면서 사용자 입력에 응답할 수 있어야 합니다.

스레드는 비동기 메소드도 사용할 수 있습니다. 두 번째 메소드를 호출하면 두 번째 메소드가 자체의 활동을 계속하기 전에 첫 번째 메소드가 완료될 때까지 기다릴 필요가 없습니다.

스레드를 사용하지 않는 여러 가지 이유가 있습니다. 프로그램이 본래부터 순차 논리를 사용하는 경우에 한 스레드가 전체 순서를 수행할 수 있습니다. 이러한 경우에 여러 스레드를 사용하면 이득은 없이 프로그램만 복잡해집니다. 스레드를 작성하고 시작하는 데 상당한 작업이 필요합니다. 조작에 단지 몇 개의 명령문만 필요한 경우에는 한 스레드에서 처리하는 속도가 보다 빨라집니다. 이 점은 조작이 개념적으로 비동기인 경우에도 해당합니다. 여러 스레드가 오브젝트를 공유하는 경우에는 오브젝트들을 동기화하여 스레드 액세스를 조정하고 일관성을 유지해야 합니다. 동기화하면 프로그램이 복잡해지고 최적의 성능을 위한 조정이 어려워지고 프로그래밍 오류로 이어질 수 있습니다.

스레드에 대한 자세한 정보는 멀티 스레드 어플리케이션 개발을 참조하십시오.

Sun Microsystems, Inc.JDK(Java Development Kit)

JDK(Java(TM) Development Kit)는 Java 개발자를 위해 Sun Microsystems, Inc.가 배포한 소프트웨어입니다. 여기에는 Java 인터프리터, Java 클래스 및 Java 개발 툴(컴파일러, 디버거, 디어셈블러, 애플릿 표시기, 스터브 파일 생성기 및 문서 생성기)이 들어 있습니다.

JDK는 일단 개발되고 JVM(Java Virtual Machine)의 어느 곳에서나 실행하는 어플리케이션을 기록할 수 있게 합니다. 한 시스템에서 JDK를 사용하여 개발된 Java 어플리케이션은 코드를 변경하거나 다시 컴파일하지 않고 다른 시스템에서 사용할 수 있습니다. Java 클래스 파일은 표준 JVM으로 이식할 수 있습니다.

현재 JDK에 대한 자세한 정보를 찾으려면 iSeries 서버에서 IBM Developer Kit for Java의 버전을 확인하십시오.

다음의 명령 중 하나를 입력하여 iSeries 서버에서 디폴트 IBM Developer Kit for Java JVM의 버전을 확인할 수 있습니다.

- Qshell 명령 프롬트에서는 `java -version`.
- CL 명령행에서는 `RUNJVA CLASS(*VERSION)`.

그런 다음 특정 문서에 대해서는 The Source for Java Technology java.sun.com 에서 Sun Microsystems, Inc. JDK의 동일한 버전을 찾으십시오. IBM Developer Kit for Java는 Sun Microsystems, Inc. Java Technology와 호환이 가능한 것으로서 사용자가 JDK(Java Development Kit) 문서에 익숙해야 합니다.

자세한 정보는 다음의 주제를 참조하십시오:

- 복수 JDK(Java Development Kit)에 대한 지원은 여러 가지 JVM 사용에 대한 정보를 제공합니다.
- 원시 메소드 및 JNI(Java Native Interface)는 원시 메소드와 수행 내용을 정의합니다. 이 주제에서도 JNI를 간략하게 설명합니다.

Java 패키지

Java 패키지는 관련된 패키지와 인터페이스를 Java로 그룹화하는 방법입니다. Java 패키지는 다른 언어로 사용할 수 있는 클래스 라이브러리와 유사합니다.

Java API를 제공하는 Java 패키지는 Sun Microsystems, Inc. JDK(Java Development Kit)의 일부로 제공됩니다.

패키지	내용
java.applet	애플릿 클래스
java.awt	그래픽, 창 및 그래픽 사용자 인터페이스(GUI) 클래스
java.awt.datatransfer	자료 전송 클래스
java.awt.event	이벤트 처리 클래스 및 인터페이스
java.awt.image	이미지 처리 클래스
java.awt.peer	플랫폼 독립성을 위한 GUI 인터페이스
java.beans	JavaBeans 구성요소 모델 API
java.io	입력 및 출력 클래스
java.lang	핵심 언어 클래스
java.lang.reflect	반영 API 클래스
java.math	임의의 정밀도 산술
java.net	네트워크 클래스
java.rmi	리모트 메소드 호출 클래스
java.rmi.dgc	RMI 관련 클래스
java.rmi.registry	RMI 관련 클래스
java.rmi.server	RMI 관련 클래스
java.security	보안 클래스
java.security.acl	보안 관련 클래스
java.security.interfaces	보안 관련 클래스
java.sql	데이터베이스 클래스에 대한 JDBC SQL API
java.text	국제화 클래스
java.util	자료 유형
java.util.zip	압축 및 압축해제 클래스

Sun Microsystems, Inc.의 Java API에 대한 자세한 정보는 Sun Microsystems, Inc. API User's Guide를 참조하십시오.

Java 툴

Sun Microsystems, Inc. JDK(Java Development Kit) 제공하는 툴의 전체 리스트는 Sun Microsystems, Inc.의 툴 참조서를 참조하십시오. IBM Developer Kit for Java가 지원하는 각각의 개별 툴에 대한 자세한 정보는 IBM Developer Kit for Java가 지원하는 Java 툴을 참조하십시오.

확장 주제

다음은 IBM Developer Kit for Java(TM)의 확장 주제입니다.

클래스, 패키지 및 디렉토리
각 Java 클래스는 패키지의 일부입니다. 패키지명은 클래스가 있는 디렉토리 구조와 관련됩니다.

통합 파일 시스템의 파일
통합 파일 시스템은 Java 관련 클래스, 소스, ZIP 및 JAR 파일을 계층적 파일 구조로 저장합니다.

파일 권한
Java 프로그램, 클래스, JAR 또는 ZIP 파일을 실행 또는 디버그하려면 읽기 권한이 있어야 합니다. 몇몇 CL 명령에 필요한 파일 권한에 대한 자세한 정보를 찾으십시오.

일괄처리 작업
SBMJOB(작업 제출) 명령을 사용하여 일괄처리 작업으로 Java 프로그램을 실행할 수 있습니다. SBMJOB 명령과 일괄처리 작업이 둘 이상의 작업을 실행할 수 있는지 확인하는 방법에 대한 자세한 정보를 찾으십시오.

Java 클래스, 패키지 및 디렉토리

각 Java(TM) 클래스는 패키지의 파트입니다. Java 소스 파일에 있는 첫 번째 명령문이 패키지에서의 특정 클래스 위치를 나타냅니다. 소스 파일에 패키지 명령문이 없는 경우 클래스는 명명되지 않은 디폴트 패키지의 일부입니다.

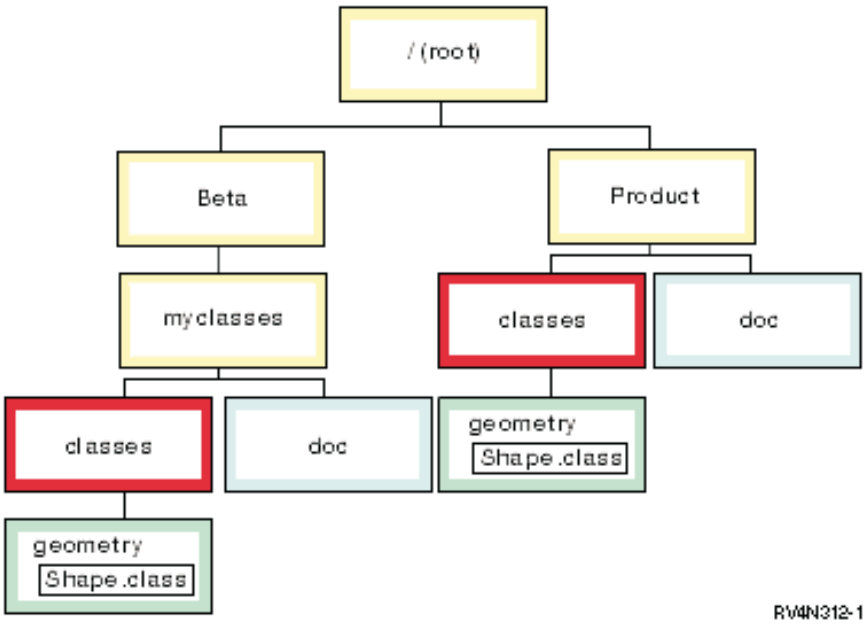
패키지명은 클래스가 있는 디렉토리 구조와 관련됩니다. 통합 파일 시스템은 대부분의 PC 및 UNIX 시스템에 있는 유사한 계층적 파일 구조의 Java 클래스를 지원합니다. 해당 클래스에 대한 패키지명과 대응하는 상대 디렉토리 경로를 갖는 디렉토리에 Java 클래스를 보관해야 합니다. 다음 Java 클래스를 예를 참조하십시오.

```
package classes.geometry;
import java.awt.Dimension;

public class Shape {

Dimension metrics;

   // The implementation for the Shape class would be coded here ...

}
```

앞의 코드에서 패키지 명령문은 Shape 클래스가 classes.geometry 패키지의 일부임을 나타냅니다. Java 런타임을 위한 Shape 클래스를 찾으려면 상대 디렉토리 구조 classes/geometry에 Shape 클래스를 저장하십시오.

주: 패키지명은 클래스가 저장된 상대 디렉토리명에 해당합니다. JVM(Java Virtual Machine) 클래스 로더는 classpath에 지정된 각 디렉토리에 상대 경로명을 첨부하여 해당 클래스를 찾습니다. JVM(Java Virtual Machine) 클래스 로더는 또한 classpath에 지정된 ZIP 또는 JAR 파일을 탐색하여 클래스를 찾을 수 있습니다.

예를 들어, Shape 클래스가 "루트"(/) 파일 시스템의 /Product/classes/geometry 디렉토리에 저장된 경우 /Product를 classpath에 지정해야 합니다.



주: Shape 클래스의 여러 버전이 디렉토리 구조에 있을 수 있습니다. Shape 클래스의 "베타" 버전을 사용하려면 Shape 클래스가 포함된 다른 모든 디렉토리나 ZIP 파일보다 먼저 CLASSPATH에 /Beta/myclasses를 배치하십시오.

Java 컴파일러는 Java 소스 코드를 컴파일할 때 패키지와 클래스를 찾기 위해 Java classpath, 패키지명 및 디렉토리 구조를 사용합니다.

통합 파일 시스템의 파일

통합 파일 시스템은 Java 관련 클래스 파일, 소스 파일, ZIP 파일 및 JAR 파일을 계층적 파일 구조로 저장합니다. 이 통합 파일 시스템에 소스 파일을 저장할 수 있습니다. 아니면 다음과 같은 통합 파일 시스템에 파일을 저장할 수 있습니다.

- "루트"(/) 파일 시스템
- 개방 시스템 파일 시스템(QOpenSys)
- 사용자 정의 파일 시스템
- 라이브러리 파일 시스템(QSYS.LIB)
- OS/2 Warp Server for iSeries 파일 시스템(QLANSrv)
- 광 파일 시스템(QOPT)

주: 다른 통합 파일 시스템은 스레드 안전성을 제공하지 않으므로 지원되지 않습니다.

통합 파일 시스템의 Java 파일 권한

Java(TM) 프로그램, 클래스 파일, JAR 파일 또는 ZIP 파일을 실행하거나 디버그하려면 읽기 권한(*R)이 필요합니다. 모든 디렉토리는 읽기 및 수행 권한(*RX)이 필요합니다.

프로그램을 최적화하기 위해 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하려면 클래스 파일, JAR 파일 또는 ZIP 파일은 읽기 권한(*R)이 있어야 하고, 디렉토리는 수행 권한(*X)이 있어야 합니다. 클래스 파일명에서 패턴을 사용하는 경우, 디렉토리는 읽기 및 수행 권한(*RX)이 있어야 합니다.

DLTJVAPGM(Java 프로그램 삭제) 명령을 사용하여 Java 프로그램을 삭제하려면 클래스 파일에 대한 읽기 및 쓰기 권한(*RW)이 있어야 하며 디렉토리는 실행 권한(*X)이 있어야 합니다. 클래스 파일명에서 패턴을 사용하는 경우 디렉토리는 읽기 및 수행 권한(*RX)이 있어야 합니다.

DSPJVAPGM(Java 프로그램 표시) 명령을 사용하여 Java 프로그램을 표시하려면 클래스 파일에 대한 읽기 권한(*R)이 있어야 하며 디렉토리에는 실행 권한(*X)이 있어야 합니다.

주: 실행 권한(*X)이 없는 파일과 디렉토리는 항상 QSECOFR 권한을 갖는 사용자에 대한 실행 권한(*X)으로 표시됩니다. 다른 사용자가 동일한 파일에 동일하게 액세스하더라도 특정 상황에서 다른 결과를 얻을 수 있습니다. Qshell 인터프리터 또는 java.Runtime.exec()를 사용하여 쉘 스크립트를 실행하는 경우 이것을 알고 있는 것이 중요합니다.

예를 들어, 한 사용자가 쉘 스크립트를 호출하기 위해 java.Runtime.exec()를 사용하는 Java 프로그램을 작성했으면 QSECOFR 권한을 가진 사용자 ID를 사용하여 Java 프로그램을 테스트해야 합니다. 쉘 스크립트의 파일 모드가 읽기 및 쓰기 권한(*RW)을 가지는 경우 통합 파일 시스템은 QSECOFR 권한을 가진 사용자 ID는 파일을 실행할 수 있습니다. 그러나 비QSECOFR 권한 사용자는 동일한 Java 프로그램을 실행하고자 하지만, 통합파일 시스템에 *X가 없기 때문에 쉘 스크립트를 실행할 수 없는 java.Runtime.exec() 코드라고 알립니다. 이 경우 java.Runtime.exec()은 입/출력 예외입니다.

통합 파일 시스템에서 Java 프로그램에 의해 작성된 새로운 파일에 권한을 지정할 수도 있습니다. 파일로 os400.file.create.auth 시스템 등록 정보와 디렉토리로 os400.dir.create.auth를 사용하면 읽기, 쓰기 및 실행 권한을 임의로 결합하여 사용할 수 있습니다.

자세한 내용은 Program and CL Command APIs 또는 통합 파일 시스템을 참조하십시오.

일괄처리 작업으로 Java 실행

Java(TM) 프로그램은 SBMJOB(작업 제출) 명령을 사용하여 일괄처리 작업으로 실행됩니다. 이 모드에서 Java Qshell 입력 화면은 System.in, System.out 및 System.err 스트림을 처리할 수 없습니다.

이러한 스트림을 다른 파일로 그 경로를 재지정할 수 있습니다. 디폴트 처리는 System.out 및 System.err 스트림을 스풀 파일로 송신합니다. System.in으로부터의 읽기 요구를 제외한 입력 및 출력하는 일괄처리 작업에 해당 스풀 파일이 있습니다. Java 프로그램 내에서 System.in, System.out 및 System.err 경로를 재지정할 수 있습니다. 또한 os400.stdin, os400.stdout 및 os400.stderr 시스템 등록 정보를 사용하여 System.in, System.out 및 System.err 경로를 재지정할 수도 있습니다.

주: SBMJOB는 현재 작업 디렉토리(CWD)를 사용자 프로파일에 지정되는 HOME 디렉토리로 설정합니다.

예: 일괄처리 작업으로 Java 실행

```
SBMJOB CMD(JAVA Hello OPTION(*VERBOSE)) CPYENVVAR(*YES)
```

이전 예에서 JAVA 명령을 실행하면 두 번째 작업이 파생됩니다. 그러므로, 일괄처리 작업이 실행하는 서브시스템은 둘 이상의 작업을 실행할 수 있어야 합니다.

다음 단계를 따라서 일괄처리 작업이 둘 이상의 작업을 실행할 수 있는지 확인할 수 있습니다.

1. CL 명령행에서, DSPSBSD(MYSBSD)를 입력하십시오. 여기서 MYSBSD는 일괄처리 작업의 서브시스템 설명입니다.
2. 옵션 6(작업 대기행렬 항목)을 선택하십시오.
3. 작업 대기행렬에 대한 최대 활동 필드를 찾으십시오.

   최대 활동 필드가 1 이하이고 *NOMAX가 아니면, CL 명령행에 다음을 입력하십시오.

   ```
   CHGJOBQE SBSD(MYSBSD) JOBQ(MYJOBQ) MAXACT(*NOMAX)
   ```

   여기서,

   - MYSBSD는 서브시스템 설명입니다.
   - MYJOBQ는 작업 대기행렬입니다.

그래픽 사용자 인터페이스가 없는 호스트에서 Java 어플리케이션 실행

iSeries 서버와 같이 그래픽 사용자 인터페이스(GUI)가 없는 호스트에서 Java(TM) 어플리케이션을 실행하려는 경우, Remote AWT(Remote Abstract Window Toolkit), CBJ(Class Broker for Java) 또는 NAWT(Native Abstract Windowing Toolkit)를 사용할 수 있습니다.

서버 어플리케이션의 설치 및 관리 인터페이스와 함께 Remote AWT를 사용합니다. 이 인터페이스는 일반적으로 복잡한 그래픽을 최소화한 것으로서 대화식으로 처리되는 내용을 가지고 있습니다. Remote AWT는 iSeries 서버와 워크스테이션간에 AWT 처리를 분산시킵니다. 따라서, 그래픽 중심의 대화식 조작에 따른 응답성은 로컬로 연결된 그래픽 단말기가 있는 플랫폼에서 AWT를 구현하는 것만큼 빠르지 않습니다. Remote AWT를 사용하려면 Remote AWT 설정을 참조하십시오.

고성능 GUI 서비스를 위해 CBJ를 사용할 수 있습니다. 복잡한 그래픽이나 대화가 많은 조작에 대해서는 Remote AWT가 권장되지 않으므로, 이러한 환경을 위해 설계된 CBJ를 대신 사용할 수 있습니다. CBJ를 사용하려면 CBJ 설정을 참조하십시오.

X Windows System을 사용하여 iSeries 서버에서 Java 그래픽 계산에 전적으로 NAWT를 사용할 수 있습니다. **X Window System**은 그래픽을 표시하기 위한 클라이언트/서버 기반을 제공하는 그래픽 시스템입니다. X Window 그래픽 서버는 이식가능성이 매우 높기 때문에 다양한 언어 및 오퍼레이팅 시스템에 대한 지원이 가능합니다. NAWT는 Java 어플리케이션과 서브릿에 JDK(Java Development Kit)의 AWT 그래픽 기능을 사용하는 능력을 제공합니다.

IBM Developer Kit for Java Remote Abstract Window Toolkit

Remote Abstract Window Toolkit은 AWT(Abstract Window Toolkit)를 구현한 것입니다. Java(TM) 어플리케이션은 그래픽 사용자 인터페이스(GUI)가 없는 호스트에서 변경없이 실행될 수 있습니다. iSeries 서버는 로컬로 연결되어 있는 그래픽 단말기를 지원하지 않으므로 그래픽 Java 어플리케이션을 iSeries 서버에서 실행하기 위해서는 Remote AWT가 필요합니다.

서버 어플리케이션의 설치 및 관리 인터페이스와 함께 Remote AWT를 사용합니다. 이 인터페이스는 일반적으로 복잡한 그래픽을 최소화한 것으로서 대화식으로 처리되는 내용을 가지고 있습니다. Remote AWT는 iSeries 서버와 워크스테이션간에 AWT 처리를 분산시킵니다. 그래픽 중심의 대화식 조작에 따른 응답성은 로컬로 연결된 그래픽 단말기를 장착한 플랫폼에서 AWT를 구현하는 것만큼 빠르지 않을 수 있습니다.

Java Remote AWT용 IBM Developer Kit가 복잡한 그래픽이나 고급 대화식 연산에는 권장되지 않으므로 이와 같은 환경에 맞추어 설계된 Class Broker for Java를 사용할 수 있습니다.

Remote AWT 설정 방법에 대한 정보는 리모트 화면에서 Remote Abstract Window Toolkit for Java 설정을 참조하십시오.

리모트 화면에서 Remote Abstract Window Toolkit for Java 설정

Remote Abstract Window Toolkit(AWT)를 사용하여, 프로그램 소스를 변경하지 않고 Java(TM) AWT 그래픽 프로그램을 실행하고 그래픽을 리모트로 표시할 수 있습니다. Remote AWT를 사용하려면 TCP/IP를 설정하고 Sun Microsystems, Inc., JDK(Java Development Kit) .1.1.8 또는 Java 2 SDK(J2SDK), Standard Edition을 iSeries 서버 및 리모트 표시장치에 설치해야 합니다.

다음 요구사항을 만족하는 경우 Remote AWT를 위한 리모트 화면으로서 IBM 네트워크 장치를 포함한 모든 그래픽이 가능한 하드웨어를 사용할 수 있습니다.

- Windows(R) 95, Windows NT 4.0, IBM Operating System/2(R) (OS/2(R)), Sun Solaris 또는 AIX(R)를 실행하는 그래픽 가능 하드웨어
- TCP/IP로 iSeries 서버에 액세스하도록 구성된 하드웨어
- Java Development Kit 1.1.8 또는 J2SDK

Remote AWT를 설정하려면 다음 타스크를 완료하십시오.

1. 파일을 리모트 화면에 복사하거나 리모트 화면의 네트워크 드라이브에 경로를 맵핑하여 리모트화면에 Remote AWT 클래스 파일 액세스 허용을 수행합니다.
2. 리모트 화면의 CLASSPATH에 RAWTGui.zip 또는 RAWTGui.jar 추가. JDK 1.1.8의 경우 CLASSPATH 환경 변수를 설정하거나 `java` 명령의 `-classpath` 매개변수를 사용하여 리모트 표시장치의 CLASSPATH에 `RAWTGui.zip` 파일을 추가하십시오. J2SDK의 경우 java 명령의 `-jar` 매개변수를 사용할 때 `RAWTGui.jar` 파일이 CLASSPATH에 자동으로 추가됩니다.
3. 리모트 화면에서 Remote AWT 시작을 수행하십시오.

Remote AWT 사용에 대한 세부사항과 힌트를 알려면 다음 주제를 참조하십시오.

- Remote AWT를 사용하여 Java 프로그램 실행은 여러 JDK 및 Netscape을 사용하여 iSeries 서버에서 Java 프로그램을 실행하는 방법에 대한 지침을 제공합니다.
- Remote Abstract Window Toolkit으로 인쇄에서는 표준 Java AWT 인쇄와 동일한 인쇄 방법을 설명합니다. 또한 iSeries 서버에 인쇄하는 방법도 보여줍니다.
- Remote Abstract Window Toolkit 등록 정보는 os400.class.path.rawt 등록 정보를 사용하여 Remote AWT 어플리케이션을 실행하는 방법을 보여줍니다.
- Remote Abstract Window Toolkit SecurityManager 제한사항은 SecurityManager의 제어 아래서 Remote AWT를 사용하여 Java 어플리케이션을 실행할 때 적용되는 제한사항이 설명됩니다.

TCP/IP 설정에 대한 자세한 정보는 TCP/IP 구성 및 참조, SA30-0227 책에서 ″TCP/IP 설정 방법″을 참조하십시오.

Remote AWT를 설정하는 방법의 예는 예: Windows 리모트 화면에서 Remote Abstract Window Toolkit for Java 설정을 참조하십시오.

AWT에 대한 자세한 정보는 ″javaapi/awt/index.html″>Sun Microsystems, Inc.의 AWT(Abstract Window Toolkit)를 참조하십시오.

리모트 화면에 Remote Abstract Window Toolkit for Java 클래스 파일 액세스 가능화

리모트 화면에 Remote AWT(Abstract Window Toolkit) 클래스 파일을 액세스 가능하게 하려면, JDK(Java(TM) Development Kit) 1.1. *x* 또는 J2SDK(Java 2 SDK), 표준판, 1.2 중 하나에 대해 다음 단계를 따르십시오. 그러나 Remote AWT가 제 기능을 수행하려면 그래픽이 가능한 리모트 표시장치에서 사용하는 RAWTGui.jar 파일의 버전이 호스트에서 사용하는 JDK나 J2SDK 버전과 일치해야 한다는 점을 유의해야 합니다.

JDK .1.1.8을 사용하고 있는 경우, 다음 중 하나를 수행할 수 있습니다.

- 리모트 화면에 Remote AWT 클래스 파일을 복사하십시오.

  Remote AWT 파일은 다음 두 개의 ZIP 파일에 IBM Developer Kit for Java와 함께 설치됩니다.

  – /QIBM/ProdData/Java400/jdk118/RAWTApplHost.zip
  – /QIBM/ProdData/Java400/jdk118/RAWTGui.zip

RAWTApplHost.zip 파일에는 iSeries 서버에 대한 Remote AWT 클래스가 들어 있습니다. RAWTGui.zip 파일에는 리모트 화면에 대한 Remote AWT 클래스가 있습니다.

/QIBM/ProdData/Java400/jdk118에서 리모트 표시장치로 RAWTGui.zip을 복사하십시오.

- 리모트 표시장치의 네트워크 드라이브로 경로 /QIBM/ProdData/Java400/jdk118/RAWTGui.zip을 맵핑시키십시오.

J2SDK, 버전 1.2 이상을 사용하고 있는 경우, 다음 중 하나를 수행할 수 있습니다.

- 리모트 화면에 Remote AWT 클래스 파일을 복사하십시오.

  Remote AWT 파일의 다음 두 개의 JAR 파일에 IBM Developer Kit for Java와 함께 설치됩니다.

  – /QIBM/ProdData/Java400/jdk12/RAWTAHost.jar
  – /QIBM/ProdData/Java400/jdk12/RAWTGui.jar

  1.2가 아닌 다른 J2SDK 버전을 사용하고 있는 경우 이 버전 번호를 이 섹션의 모든 경로 인스턴스로 대체하십시오.

  RAWTAHost.jar 파일에는 iSeries 서버에 대한 Remote AWT 클래스가 들어 있습니다. RAWTGui.jar 파일에 Remote AWT 클래스가 있습니다.

  /QIBM/ProdData/Java400/jdk12에서 리모트 표시장치의 네트워크 드라이브로 RAWTGui.jar을 복사하십시오.

- 리모트 표시장치의 네트워크 드라이브에 경로 /QIBM/ProdData/Java400/jdk12/RAWTGui.jar을 맵핑시키십시오.

리모트 화면의 CLASSPATH에 RAWTGui.zip 또는 RAWTGui.jar 추가

CLASSPATH를 설정하면 리모트 화면의 Java(TM) 가상 기계에서 Remote Abstract Window Toolkit(AWT) 클래스를 발견할 수 있습니다. 이 단계는 JDK 1.1.x에만 필요합니다. 이 단계는 Java 2 SDK(J2SDK)에는 필요하지 않습니다. 리모트 화면의 CLASSPATH에 RAWTGui.zip 파일을 추가하려면 다음 단계 중 하나를 수행하십시오.

- CLASSPATH 환경 변수를 설정하십시오. 세부사항에 대하여 리모트 화면의 JDK(Java Development Kit) 정보를 참조하십시오.

  RAWTGui.zip 파일이 위치한 경로를 CLASSPATH 환경 변수에 추가하십시오.

- `java` 명령의 -classpath 매개변수를 사용하십시오.

  `java` 명령을 사용하여 Remote AWT를 시작할 때 -classpath 매개변수를 사용하여 CLASSPATH를 지정할 수 있습니다. CLASSPATH에는 RAWTGui.zip 파일이 위치한 경로가 들어 있습니다.

  예를 들어 Windows(R)에서, CLASSPATH 매개변수는 다음과 유사하게 표시될 수 있습니다.

  ```
  -classpath c:\jdk1.1.7\lib\classes.zip;c:\rawt\RAWTGui.zip
  ```

J2SDK, 버전 1.2 이상 JAR 지원은 CLASSPATH를 설정하므로 CLASSPATH 매개변수를 외부적으로 설정할 필요가 없습니다. 리모트 표시 화면에서 classpath를 설정하고 Remote AWT를 시작하기 위해 다음 명령을 입력하십시오.

```
java -jar <PATH>RAWTGui.jar
```

<PATH>는 완전히 규정된 드라이브이며 RAWTGui.jar 파일이 위치한 디렉토리입니다. 한 예로 `java -jar c:\rawt2\RAWTGui.jar`가 있습니다.

리모트 화면에서 Remote Abstract Window Toolkit for Java 시작

리모트 화면에서 한 번 서버 디먼을 시작해야 하며, 시작된 서버는 사용자가 종료할 때까지 계속 작동합니다. iSeries 서버에서 나가는 Java(TM) 프로그램은 서버 디먼을 종료하지 않습니다.

주: 서버 디먼을 시작할 때 Welcome 대화 상자 화면이 계속 활동합니다. Welcome 대화 화면이 닫힐 때 서버 디먼이 종료됩니다. 서버 디먼을 사용하는 동안 Welcome 대화 화면을 최소화시켜 화면을 사용하여 서버 디먼을 종료할 수 있습니다.

JDK 1.1.*x*의 Remote AWT(Abstract Window Toolkit) 서버 디먼을 시작하려면 명령행에 다음과 같이 입력하십시오.

```
java -classpath <PATH>RAWTGui.zip;C:\jdk1.1.8\lib\classes.zip
com.ibm.rawt.server.RAWTPCServer
```

<*PATH*>는 완전히 규정된 드라이브이며 RAWTGui.jar 파일이 위치한 디렉토리입니다. 예를 들어, `java -jar c:\rawt2\RAWTGui.jar`이 있습니다.

J2SDK, 버전 1.3에 대한 Remote AWT 서버 디먼을 시작하려면 명령행에 다음과 같이 입력하십시오.

```
java -jar <PATH>RAWTGui.jar
```

<*PATH*>는 완전히 규정된 드라이브이며 RAWTGui.jar 파일이 위치한 디렉토리입니다. 예를 들어, `java -jar c:\rawt2\RAWTGui.jar`이 있습니다.

Java 어플리케이션이 Remote AWT를 사용하여 연결할 때 2000 이상의 첫 번째 사용 가능 포트를 선택합니다. Java 어플리케이션은 종료시까지 이 포트를 사용합니다. 추가 Java 어플리케이션은 2000 이상의 후속 사용 가능 포트에 연결됩니다. 포트의 사용할 수 있는 범위는 9999까지입니다.

TCP/IP 설정에 대한 자세한 정보는 TCP/IP 구성 및 참조, SA30-0227 책에서 ″TCP/IP 설정 방법″을 참조하십시오.

Remote Abstract Window Toolkit을 사용한 Java 프로그램 실행

Remote Abstract Window Toolkit(AWT)을 사용하여 Java(TM)를 실행하려면 다음과 같이 하십시오.

1. 리모트 화면에서 Remote AWT 시작을 수행하십시오.
2. iSeries 서버에서 Java 프로그램을 시작하십시오.

a. 명령 행에 RUNJVA(Java 실행) 명령을 입력하십시오.
   주: Java 프로그램에 Java classpath를 정의해야 합니다.
b. F4(프롬트) 키를 누르십시오.
c. 클래스 매개변수 행에 Java 프로그램 클래스명을 입력하십시오.
d. F10(추가 매개변수) 키를 누르십시오.
e. 뒷장 키를 누르십시오.
f. 다음 등록 정보명 매개변수 행에 RmtAwtServer를 입력하십시오.
g. 다음 등록 정보 값 매개변수 행에 리모트 화면의 TCP/IP 주소를 입력하십시오(예를 들면, 1.1.11.11).
h. 등록 정보명 매개변수 행에 os400.class.path.rawt를 입력하십시오.
i. 등록 정보 값 매개변수 행에 1을 입력하십시오.
j. 추가 등록 정보를 보려면 +를 입력하십시오.
k. 등록 정보명 매개변수 행에 java.version을 입력하십시오.
l. 등록 정보 값 매개변수 행에 1.3을 입력하십시오. 이 버전은 리모트 표시장치에서 실행되고 있는 RAWTGui.jar 디먼의 버전과 일치해야 합니다.

   명령행의 패턴은 다음과 같으며 모두 한 행에 입력해야 합니다.

```
java class(classname) prop(('RmtAwtServer' '1.1.11.11')
     ('os400.class.path.rawt' '1')('java.version' '1.3'))
```

m. Enter 키를 누르십시오.

Netscape와 함께 Remote AWT를 사용하여 Java 프로그램을 실행할 수도 있습니다.

Netscape와 함께 Remote Abstract Window Toolkit을 사용하여 Java 프로그램 실행: Netscape와 함께 Java(TM) 어플리케이션을 실행할 때 다음 두 방법 중 하나로 실행할 수 있습니다.

한 가지 옵션은 com.ibm.rawt.server.StartRAWT.class가 들어 있는 HTML 파일을 열어서 Netscape JVM(Java Virtual Machine) 내에서 Remote Abstract Window Toolkit(AWT) 서버를 시작하는 것입니다. 예를 들어, 아래 RAWT.html 파일을 참조하십시오. 일단 시작되면 iSeries 서버에서 Java 어플리케이션을 시작할 수 있습니다.

또는, com.ibm.rawt.server.StartRAWT400.class와 IBM Toolbox for Java 클래스가 들어 있는 HTML 파일을 열어서 Netscape JVM(Java Virtual Machine) 내에서 Remote AWT 서버를 시작할 수 있습니다. 예를 들어, 아래에 나온 RAWT400.html 파일을 참조하십시오. 일단 시작되면 Java 어플리케이션이 상주하는 iSeries 서버에 사인 온하여 어플리케이션을 시작할 수 있습니다.

Netscape JVM(Java Virtual Machine) 내에서 Remote AWT 서버 실행

Netscape JVM(Java Virtual Machine) 내에서 Remote AWT 서버를 실행하려면 다음 단계를 수행하십시오.

1. 다음 예에서 RAWTGui.zip의 특정 설치 정보를 위한 .html 파일을 편집합니다. 파일 RAWT.html은 Netscape JVM(Java Virtual Machine) 내에서 Remote AWT를 시작합니다.

   예: Netscape JVM(Java Virtual Machine) 내에서 Remote AWT 시작

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
<HTML>
<BODY TEXT="#000000" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<APPLET CODE="com.ibm.rawt.server.StartRAWT.class"
codebase="file://C|remote_awt\jdk1.1.7\lib\RAWTGui.zip"
WIDTH=600 HEIGHT=50>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

2. Netscape 4.05 이상을 사용하여 RAWT.html 페이지를 탐색합니다. 요구된 모든 권한을 부여한 다음에 Netscape는 Remote AWT 서버를 시작하여 JVM(Java Virtual Machine) 내에서 실행합니다.
3. Remote AWT를 사용하여 iSeries 서버에서 Java 어플리케이션을 시작하십시오.
4. 어플리케이션을 나간 다음 Remote AWT 서버를 다시 시작하려면 시프트(shift) 키를 누른 상태에서 재로드(Reload) 버튼을 클릭하십시오.

Netscape JVM 내에서 Remote AWT 서버 실행 및 iSeries 서버에 사인 온:

Netscape JVM에서 Remote AWT 서버를 실행하고 iSeries 서버에 사인 온하려면 다음의 단계를 따르십시오.

1. 다음 예에서 jt400.zip 및 RAWTGui.zip의 특정 설치 정보를 위한 .html 파일을 편집하십시오. 이 파일 RAWT400.html은 Remote AWT를 시작하며 IBM Toolbox for Java를 사용하여 iSeries 서버에 사인 온하십시오.

   예: iSeries 서버에 사인 온하기 위해 리모트 AWT 시작 및 Java용 Toolbox 사용

   주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
<HTML>
<BODY TEXT="#000000" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<APPLET ARCHIVE="file://C\jt400\lib\jt400.zip"
code="com.ibm.rawt.server.StartRAWT400.class"
codebase="file://C|remote_awt\jdk1.1.1\lib\RAWTGui.zip"
WIDTH=600 HEIGHT=50>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

2. Netscape 4.05를 사용하여 RAWT400.html 페이지를 탐색하십시오. 요구된 모든 권한을 부여한 다음, Netscape는 다음과 같은 옵션을 수행할 수 있는 패널을 표시하는 Remote AWT 애플릿을 시작합니다.
   - iSeries 서버에 액세스하기 위해 IBM Toolbox for Java를 사용하여 Remote AWT가 있는 iSeries 서버에 사인 온하십시오.
   - Remote AWT 등록 정보와 함께 Java 어플리케이션명과 인수를 입력하십시오.
   - 어플리케이션 시작 버튼을 눌러 Remote AWT와 함께 지정된 Java 어플리케이션을 시작하십시오.

Remote Abstract Window Toolkit으로 인쇄

Remote AWT(Remote Abstract Window Toolkit)로 인쇄하는 것은 표준 Java(TM) AWT 인쇄와 동일합니다. Remote AWT 리모트 화면이 인쇄 출력을 처리하고 리모트 화면 오퍼레이팅 시스템에 알려진 임의의 프린터로 출력을 지정합니다. 이것은 리모트 화면에 직접 연결된 프린터 또는 리모트 화면 오퍼레이팅 시스템에 알려진 네트워크 프린터일 수 있습니다.

리모트 표시장치에 인쇄하거나 iSeries 서버에 인쇄하도록 선택할 수 있습니다. 어플리케이션이 인쇄를 요구할 때 새로운 인쇄 대화가 표시됩니다. 인쇄 요구는 사용자가 리모트 화면 프린터 또는 OS/400 프린터 중에서 선택할 수 있게 합니다. OS/400 프린터를 선택하는 경우 시작(Sign On) 대화 화면이 나타납니다. 일단 시작(Sign On)하면 인쇄 대화 화면이 나타납니다. OS/400 인쇄 대기행렬, 인쇄 파일, 파일 및 배너 페이지 타이틀을 지정할 수 있습니다. 또한 용지 크기, 인쇄 방향 및 복사 수를 선택할 수도 있습니다.

리모트 인쇄를 사용하려면 IBM Toolbox for Java(5763-JC1)를 설치하고 iSeries 서버의 classpath에 다음을 추가해야 합니다.

```
QIBM/ProdData/HTTP/Public/jt400/lib/jt400.zip
```

classpath 환경 변수를 추가하거나 classpath 매개변수를 사용하여 classpath를 갱신할 수 있습니다.

주: iSeries 서버에 인쇄하는 동안 이 메세지가 나타나면, IBM Toolbox for Java가 설치되지 않았거나 IBM Toolbox for Java 클래스가 classpath에 없는 것입니다.

```
클래스 파일 로드 실패: com/ibm/as400/access/PrintObjectList.class
이벤트 발송 중에 예외가 발생했습니다:
java.lang.NoClassDefFoundError: com/ibm/as400/access/PrintObjectList
```

Remote Abstract Window Toolkit 등록 정보

iSeries 서버에서 Java(TM) Remote AWT 어플리케이션을 실행할 때 os400.class.path.rawt 등록 정보를 1 값으로 사용해야 합니다. Remote AWT를 사용하는 경우에는 다수의 디폴트 등록 정보가 필요합니다. 이 디폴트 등록 정보는 os400.class.path.rawt 등록 정보를 사용할 때 Remote AWT의 적절한 버전과 CLASSPATH와 함께 설정됩니다. Remote AWT의 버전은 JDK 버전에 따라서 설정되며 지정하지 않을 경우 이것이 디폴트 버전이나 java.version 등록 정보로 지정하는 버전입니다.

다음은 Remote AWT에 필수인 디폴트 등록 정보입니다.

JDK 1.1.*x*의 경우:

- awt.toolkit=com.ibm.rawt.CToolkit

J2SDK의 경우:

- awt.toolkit=com.ibm.rawt2.ahost.java.awt.AHToolkit
- java.awt.graphicsenv=com.ibm.rawt2.ahost.java.awt.AHGraphicsEnvironment
- java.awt.printerjob=com.ibm.rawt2.ahost.java.awt.print.AHPrinterjob

리모트 화면에 대한 Remote AWT 등록 정보

서버 디먼 또는 Java 어플리케이션이 다음 메세지로 이상 종료하는 경우 리모트 화면의 Java 버전을 체크하십시오.

```
Applic tion-host/User-station에 있는 JDK 버전이 Remote AWT
버전과 호환되지 않습니다.
```

버전 레벨을 확인하려면 명령 행에 `java -version`을 입력하십시오. JDK 버전에 문제점이 있는 경우 리모트 화면에서 이 새로운 등록 정보를 사용할 수 있습니다. 이 등록 정보는 iSeries 서버에는 해당되지 않습니다. 버전이 레벨 1.1.x가 아닌 경우 적합한 버전을 설치해야 합니다. 버전 레벨이 1.1.x인 경우 Java 버전을 표시하는 이 등록 정보로 Remote AWT 서버 또는 Java 어플리케이션을 실행할 수 있습니다.

```
-DJdkVersion=1.1.x
```

Remote Abstract Window Toolkit SecurityManager 제한사항

Java(TM) SecurityManager는 일반적으로 사용되지 않습니다. 그러나 SecurityManager가 설치된 경우 RAWT가 작동할 수 있도록 나열된 호출들을 성공시켜야 합니다.

- SecurityManager.checkAccess(..)
- SecurityManager.checkMemberAccess(..)
- SecurityManager.checkExit(..)
- SecurityManager.checkRead(String file)
- SecurityManager.checkConnect(...)
- SecurityManager.checkListen(...)
- SecurityManager.checkAccept(...)
- SecurityManager.checkPropertiesAccess(..)

예: Windows 리모트 화면에서 Remote Abstract Window Toolkit for Java(TM) 설정

이 예는 Windows(R) 리모트 화면에서 Remote AWT를 설정하는 한 가지 방법을 보여줍니다. 사용자 설정에 따라 다양한 방법으로 설정을 수행할 수 있습니다. 다른 리모트 화면 오퍼레이팅 시스템에서도 이와 같이 처리할 수 있습니다. 설정 및 시작 프로세스는 Windows .bat 파일이나 리모트 화면 오퍼레이팅 시스템이 제공하는 다른 프로그래밍 기능으로 자동화됩니다.

Windows 리모트 화면에서 Remote Abstract Window Toolkit(AWT) for Java(TM)를 설정하려면 다음 타스크를 수행하십시오.

- Remote AWT 클래스 파일을 리모트 화면에 액세스할 수 있게 하십시오.

  리모트 화면에 Remote AWT 클래스 파일을 복사하십시오. 다음 타스크 중 하나를 수행하십시오.

  – /QIBM/ProdData/Java400/jdk118/RAWTGui.zip을 c:\rawt\RAWTGui.zip으로 복사합니다.

  – /QIBM/ProdData/Java400/jdk1*x*/RAWTGui.jar을 c:\rawt2\RAWTGui.jar로 복사합니다. 여기서 *x*는 J2SDK(Java 2 Software Development Kit), 표준판 버전(2,3 또는 4)입니다.

- 명령 행에 다음을 입력하여 리모트 화면에서 Remote AWT를 시작하십시오.

```
     java -classpath c:\jdk1.1.8\lib\classes.zip;c:\rawt\RAWTGui.zip
java com.ibm.rawt.server.RAWTPCServer
```

또는

```
java -jar c:\rawt2\RAWTGui.jar
```

자세한 내용은 리모트 화면에서 Remote Abstract Window Toolkit for Java 설정을 참조하십시오.

Class Broker for Java

Class Broker for Java(TM)(CBJ)는 클라이언트/서버 어플리케이션을 Java로 쓰기 위한 범용 구조입니다. 일반적으로 클라이언트/서버 어플리케이션은 클라이언트 오브젝트와 서버 오브젝트로 구성됩니다. 서버와 클라이언트는 두 오브젝트 간의 모든 통신을 처리합니다. 통신은 RMI(Remote Method Invocation) 또는 소켓 연결로 실행됩니다. RMI는 사용하는 것이 쉽지 않으며 소켓을 효율적으로 사용하기 위해서는 많이 익혀야 필요합니다.

CBJ는 사용이 쉬우며 융통성이 있고 소켓 연결시의 복잡성이 없습니다. 어플리케이션 초기화를 위한 몇 번의 CBJ 클래스 호출을 제외하고 중간 클라이언트/서버 어플리케이션은 로컬 어플리케이션처럼 보입니다. CBJ는 클라이언트와 서버 사이의 모든 통신과 자원 로딩을 처리합니다. 프로그램 오브젝트 중 어떤 오브젝트가 클라이언트에서 실행하고 있고 어떤 오브젝트가 서버에서 실행하고 있는지는 대부분 투명하게 드러납니다. CBJ는 CBJ 런타임을 사용하여 클라이언트와 서버 프록시를 작성합니다. 브로커가 일단 프록시 오브젝트를 작성하면 클라이언트가 서버 프록시에서 메소드를 호출함으로써 리모트 서버와 통신합니다. 마찬가지로, 서버 오브젝트가 클라이언트 프록시에서 메소드를 호출함으로써 클라이언트와 통신합니다. 이와 같이, 어플리케이션의 클라이언트와 서버쪽 모두 로컬 오브젝트 메소드를 호출하는 것처럼 보입니다.

Class Broker for Java 사용에 관한 정보는 Class Broker for Java 설정을 참조하십시오.

리모트 화면에서 Class Broker for Java 설정

Class Broker for Java(TM)(CBJ)를 사용하여, iSeries 서버에서 CBJ(Class Broker for Java) 가능 Java 그래픽 프로그램을 실행하고 그래픽을 리모트로 표시할 수 있습니다.

사용자가 CBJ를 설치하거나 시스템 관리자가 설치하도록 만들 수 있습니다. 시스템 관리자가 제품을 설치하는 경우 모든 프로그래머가 동일한 Java 코드를 공유할 수 있습니다.

Windows(R) 95/98/NT, UNIX(R) 또는 iSeries 서버에서 CBJ를 설치할 수 있습니다. 대부분의 경우에, 클라이언트 기계와 서버 기계에 CBJ를 설치해야 합니다.

주: 클라이언트 기계가 서버 기계에 있는 웹 서버를 통해 액세스되는 클라이언트 애플릿을 실행 중인 경우 클라이언트 기계에 CBJ를 설치할 필요가 없습니다.

iSeries 서버에 CBJ를 설치하려면 iSeries 서버에 CBJ(Class Broker for Java) 설치를 참조하십시오.

워크스테이션에 CBJ를 설치하려면 Windows 또는 UNIX에 CBJ(Class Broker for Java)를 참조하십시오.

iSeries 서버에 Class Broker for Java 설치

iSeries 서버에 CBJ(Class Broker for Java(TM))를 설치하려면, 다음 단계를 수행하십시오.

1. IBM Developer Kit for Java가 제대로 설치되었는지 확인하십시오. 설치를 테스트하려면 IBM Developer Kit for Java 설치를 참조하십시오.
2. 통합 파일 시스템의 디렉토리를 선택하여 cbj_1.1.jar이라는 CBJ 패키지를 저장하십시오. 예를 들면, /usr/local입니다.
3. QSH 명령을 입력하여 Qshell 인터프리터를 시작하고 통합 파일 시스템에서 선택한 디렉토리로 찾아 가십시오. 예를 들면, /usr/local입니다. IBM Developer Kit for Java를 설치하면 cbj_1.1.jar 패키지가 QIBM/ProdData/Java400/ext에 설치됩니다.
4. 다음 명령을 Qshell에 입력하십시오.

   ```
   jar xvf "PATH"cbj_1.1.jar
   ```

   "PATH"는 cbj_1.1.jar 패키지가 위치한 디렉토리입니다. 예를 들면, QIBM/ProdData/Java400/ext입니다.

   CBJ 파일이 /usr/local/JCBroker라는 서브디렉토리에 추출됩니다. 자세한 정보는 cbj_.1.1.jar의 패키지 내용을 참조하십시오.
5. 다음의 명령을 입력하여 iSeries 서버에 jcb.jar에 대한 Java 프로그램을 작성하십시오.

   ```
   CRTJVAPGM CLSF('/usr/local/JCBroker/lib/jcb.jar')
   ```
6. CBJ 클래스를 (디버그 모드에서가 아니라) 사용하려는 경우, Java 명령행에 JCBroker\lib 및 JCBroker\lib\jcb.jar을 추가해야 합니다. CLASSPATH 환경 변수에는 JCBroker\lib 및 JCBroker\lib\jcb.jar을 추가하지 마십시오. ARCHIVE 태그에 applet_jcb.jar을 설정하는 애플릿을 실행할 때 클래스 로드와 충돌할 수 있기 때문입니다.
7. 디버그 모드에서 CBJ를 실행하려는 경우, classpath에서 jcb.jar을 jcbd.jar로 대체하거나 Java 명령행의 classpath 값을 대체하십시오. 다음 명령을 사용할 수도 있습니다.

   ```
   CRTJVAPGM CLSF('/usr/local/JCBroker/lib/jcbd.jar')
   ```
8. CBJ API, 데모 프로그램 실행, 등록 정보 편집, CBJ 어플리케이션 설계 및 작성 및 기타 주제에 대한 자세한 정보는 cpj_1.1.jar 패키지에 있는 JCBroker/index.html 파일을 참조하십시오.

Windows 또는 UNIX에 Class Broker for Java 설치

Windows(R)에 CBJ(Class Broker for Java(TM))를 설치하려면 다음 단계를 수행하십시오.

1. JDK/JRE1.1 또는 JDK/JRE1.2가 적절하게 설치되었는지 확인하십시오.
2. cbj_1.1.jar이라는 CBJ 패키지를 저장할 디렉토리를 선택하십시오(예를 들면, C:\).
3. 해당 디렉토리(C:\)로 변경한 후 다음 명령을 입력하십시오.

   ```
   C:\ > jar xvf cbj_1.1.jar
   ```

   CBJ 파일이 추출되고 디렉토리에 복사됩니다. 자세한 정보는 cbj_.1.1.jar의 패키지 내용을 참조하십시오.
4. CBJ 클래스를 사용하려면(디버그 모드에서가 아님) Java 명령행의 classpath 옵션에 C:\JCBroker\lib 및 C:\JCBroker\lib\jcb.jar을 추가하십시오.

   주: 시스템 CLASSPATH 환경 변수에는 추가하지 마십시오. ARCHIVE 태그에서 applet_jcb.jar을 설정하는 애플릿을 실행할 때 클래스 로드와 충돌할 수 있기 때문입니다.

5. 디버그 모드에서 CBJ를 실행하려면 classpath에서 jcb.jar을 jcbd.jar로 대체하십시오.

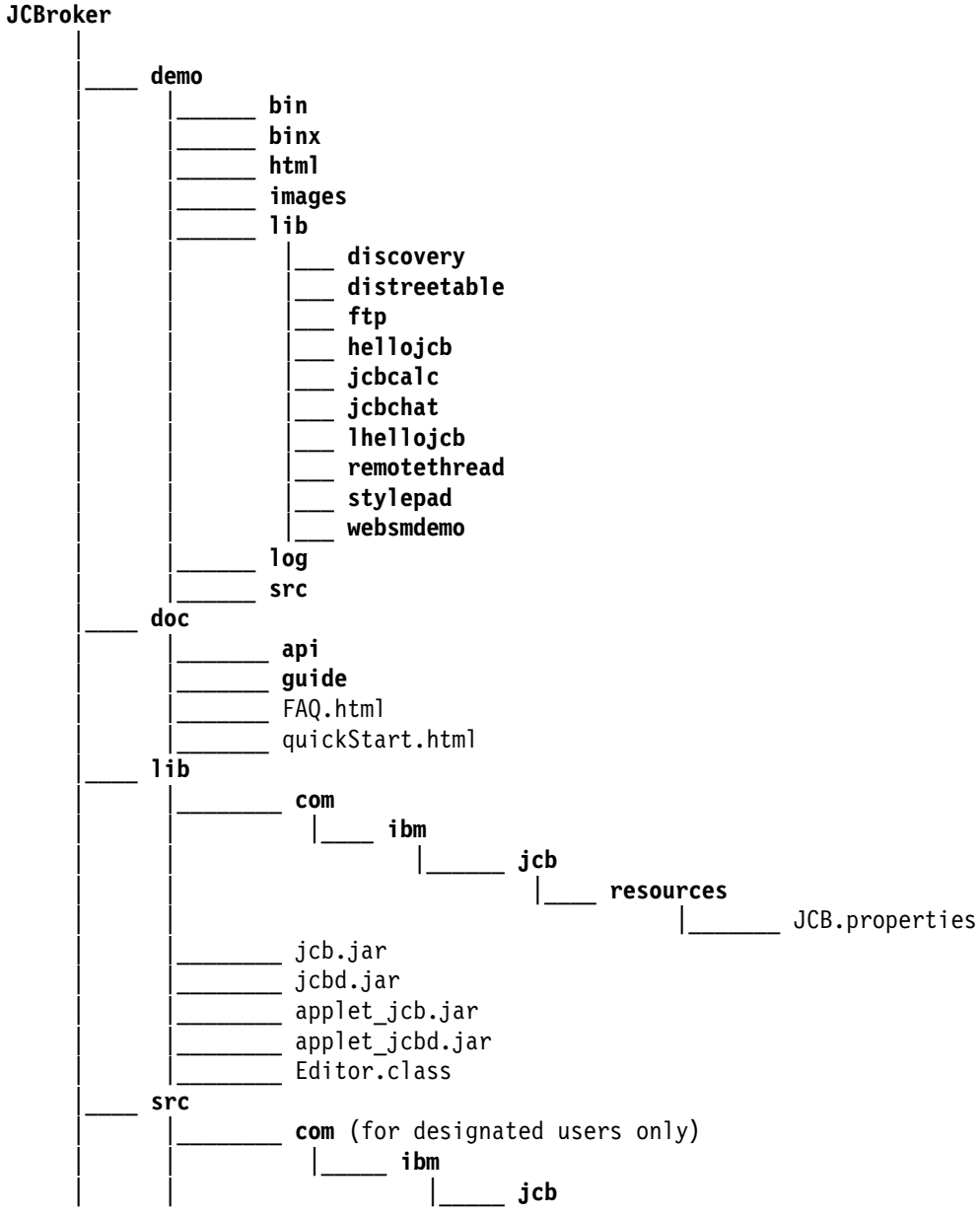
   UNIX(R)에 CBJ를 설치하려면, 다음 시스템에 따른 변경사항을 제외하고 Windows의 경우와 같은 단계를 수행하십시오.

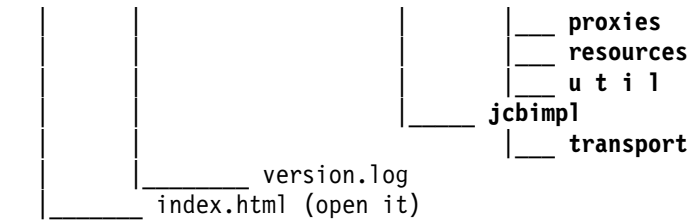
   - Windows 파일 분리자 "\"를 UNIX 파일 분리자 "/"로 대체하십시오.
   - Windows classpath 분리자 ";"을 UNIX classpath 분리자 ":"으로 대체하십시오.
   - Windows 시스템 환경 변수 "%XXX%"를 UNIX 시스템 환경 변수 "$XXX"로 대체하십시오.

6. CBJ API, 데모 프로그램 실행, 등록 정보 편집, CBJ 어플리케이션 설계 및 작성 및 기타 주제에 대한 자세한 정보는 cpj_1.1.jar 패키지에 있는 JCBroker/index.html 파일을 참조하십시오.

cbj_1.1.jar 패키지 내용

다음은 CBJ(Class Broker for Java(TM)) 패키지(cbj_1.1.jar)입니다. 굵은체의 항목은 서브디렉토리의 기본 디렉토리를 나타냅니다. 굵은체가 아닌 항목은 파일을 표시합니다.





JCBroker 디렉토리 아래 그 다음 레벨의 디렉토리는 다음 자료로 구성됩니다.

demo

이 디렉토리는 bin, binx, html, images, lib, log 및 src의 서브디렉토리를 갖습니다. 이 디렉토리에는 Windows(R) .exe 파일, UNIX(R) .exe, HTML 페이지, GIF 파일, 클래스 파일 및 여러 데모 어플리케이션의 소스 코드(예를 들어)가 들어 있습니다. 데모 지침에 따라 위의 데모 예를 실행할 수 있습니다. log 디렉토리는 디버그 기록부 파일의 위치 보유자(placeholder)입니다.

doc

이 디렉토리는 api 및 guide의 두 서브디렉토리를 갖습니다. CBJ에 대한 CBJ API 안내서 및 사용자 안내서가 여기에 저장됩니다. API 문서는 J2SDK, javadoc 툴을 사용하여 생성됩니다. API 문서나 사용자 안내서를 보려면 대응 디렉토리에서 index.html 파일을 여십시오. 사용자 안내서는 안내 수준의 문서인 반면에 QuickStart 안내서는 신속하게 찾아볼 수 있는 항목별 문서입니다.

이 디렉토리에는 또한 CBJ의 설치와 사용을 설명하고 데모 예의 실행을 보여주는 FAQ.html 파일과 quickStart.html 파일이 포함되어 있습니다.

lib

이 디렉토리는 새로운 CBJ 기반의 어플리케이션을 개발, 실행 및 전개하는 데 필요한 CBJ 클래스와 자원을 포함합니다. 클래스는 jcb.jar 파일에 패키지되어 있으며, 디버그 모드에 대해서는 jcbd.jar 파일에 패키지되어 있습니다. 애플릿의 ARCHIVE 태그에 포함된 클래스는 applet_jcb.jar 파일에 패키지되어 있습니다. 디버그 모드의 경우 클래스가 applet_ jcbd.jar 파일에 패키지되어 있습니다. JCB.properties라는 이름의 Java Class Broke(JCB) 등록 정보 파일은 com/ibm/jcb/resource 서브디렉토리에 있습니다. 시작할 때 CBJ 런타임이 읽습니다. 이 파일을 변경해서 기본설정에 반영할 수 있습니다. 설치와 설정 섹션에서 등록 정보를 설명합니다. 애플릿 등록 정보는 applet_JCB.properties 파일에만 포함되고 applet_ jcb.jar 및 applet_ jcbd.jar의 한 부분이 됩니다. 이 파일을 변경해서 애플릿 모드에서 기본설정에 반영할 수 있습니다. 이 파일을 변경하려면 이 디렉토리에 상주하는 Java 어플리케이션 편집기를 실행하십시오.

src

이 디렉토리는 CBJ 소스 코드와 version.log라는 초기 버전 기록부 파일을 포함합니다. 디렉토리는 선택된 패키지를 제외하고 비어 있습니다.

index.html

이것은 사용자에게 이 문서의 각 내용을 알려주는 시작 페이지입니다.

NAWT(Native Abstract Windowing Toolkit)

NAWT(Native Abstract Windowing Toolkit)은 JDK(Java Development Kit)의 AWT(Abstract Windowing Toolkit) 그래픽 기능을 사용할 수 있는 기능을 갖춘 Java(TM) 어플리케이션과 서브릿을 제공합니다.

직접 사용자 대화가 필요 없는 Java 그래픽 어플리케이션에서 NAWT("노티"로 발음됨)를 사용할 수 있습니다. 이러한 어플리케이션의 예로는 JPEG 또는 GIF 인코드 파일 또는 출력 스트림과 같은 이미지를 생성하는 코드가 있습니다. 이 릴리스의 경우, NAWT는 JDK 버전 1.2와 1.3을 지원합니다.

NAWT를 비대화식 어플리케이션에 대한 RAWT(Remote Abstract Windowing Toolkit)의 대안으로 사용할 수도 있습니다. RAWT에서는 접속된 PC 서버가 그래픽 계산을 수행합니다. RAWT를 보다 일반적인 그래픽 기반 어플리케이션 클래스에 사용할 수 있지만 리모트 PC가 필요하며 대개 iSeries 서버와 PC GUI 서버 사이의 네트워크 지연으로 인해 약간의 성능 오버헤드가 발생합니다.

NAWT에서는 기초 그래픽 엔진으로 X Window System을 사용하여 iSeries 서버에서 그래픽 작업을 완벽하게 수행합니다. X Window System은 그래픽 표시를 위한 클라이언트/서버 기본을 제공하는 그래픽 시스템입니다.

NAWT 설치 및 어플리케이션에서의 사용에 대한 자세한 정보는 다음 주제를 참조하십시오.

NAWT 설치
권장되는 X Window 그래픽 서버, OS/400 PASE, 필수 소프트웨어 수정 프로그램(PTF), iSeries Tools for Developers PRPQ 등에 대한 정보를 포함하여, iSeries 서버에 NAWT 설치의 내용을 읽으십시오.

VNC 사용 추가 정보
CL 프로그램에서 BNC 표시장치 서버를 시작하는 방법과 VNC 표시장치 서버를 종료하는 방법을 알 수 있습니다.

NAWT가 사용하는 VNC 서버에 대해서는 AT&T Research Virtual Network Computing  웹 페이지를 참조하십시오.

NAWT 설치

NAWT는 그래픽 엔진을 위해 X Window System에 의존합니다. 이 릴리스에 대해 권장되는 X Window 그래픽 서버는 VNC(Virtual Network Computing)이며 OS/400 PASE(Portable Application Solutions Environment) 하에서 실행합니다.

PASE는 IBM의 AIX 오퍼레이팅 시스템용으로 컴파일된 대부분의 2진 실행 파일들을 실행할 수 있는 UNIX와 유사한 환경입니다. VNC는 AT&T 연구소의 제품이며, 사용자가 직접 그래픽 가능 표시장치에 접속할 필요가 없는 X 서버인 *가상* X Window 서버를 제공합니다.

NAWT(Native Abstract Windowing Toolkit)를 설치하고 실행하려면, 다음 타스크를 수행하십시오.

1. 334 페이지의『OS/400 PASE 설치』
2. NAWT PTF 설치(334 페이지의『NAWT PTF 설치』페이지 참조)

3. 『iSeries Tools for Developers PRPQ 설치』
4. 『VNC 암호 파일 작성』
5. 335 페이지의 『Java 시스템 등록 정보 구성』
6. 335 페이지의 『VNC 서버 시작』
7. 335 페이지의 『환경 변수 설정』
8. 설치 프로시듀어 확인(336 페이지 참조)

OS/400 PASE 설치: OS/400 PASE(Portable Application Solutions Environment), 5722SS1, 옵션 33을 주문하여 설치하십시오. 자세한 정보는 OS/400 PASE를 참조하십시오.

NAWT PTF 설치: 소프트웨어 수정 프로그램(PTF)을 설치하기 전에, 다음 단계를 수행함으로써 사용하고자 하는 JDK 버전에 해당되는 사용권 프로그램 5722JV1 옵션이 있는지 확인하십시오.

1. 명령 행에 GO LICPGM(사용권 프로그램으로 이동) 명령을 입력하십시오.
2. 옵션 10(설치된 사용권 프로그램 표시)을 선택하고, 사용하려는 JDK 버전에 해당하는 사용권 프로그램 5722JV1 옵션이 설치되어 있는지 확인하십시오.

   옵션은 다음과 같습니다.

JDK 버전	옵션
JDK 1.2	57SSJV1 옵션 3
JDK 1.3	57SSJV1 옵션 5

최신 Java 그룹 소프트웨어 수정 프로그램을 적용하여 최신 NAWT 수정 프로그램을 선택하십시오. 소프트웨어 수정 프로그램에 대해서는 소프트웨어 수정 프로그램 사용을 참조하십시오.

iSeries Tools for Developers PRPQ 설치: iSeries Tools for Developers PRPQ(5799PTL)를 설치하십시오. PRPQ가 없으면, 주문해야 합니다.

PRPQ 새 버전에는 VNC(Virtual Network Computing)의 사전컴파일된 OS/400 PASE 가능 버전이 들어 있습니다. 이번 버전에는 VNC가 없습니다. PRPQ 설치 방법은 가지고 있는 버전에 따라 다릅니다.

- 2002년 6월 14일 이후 주문된 PRPQ 버전의 경우, Application Factory - iSeries Tools for Development 웹 사이트에서 사용가능한 설치 지침을 사용하여 이 타스크를 완료하십시오.

  주: PRPQ에서 사용가능한 VNC 지원을 설치하려면, 어플리케이션 팩토리 웹 사이트에 있는 설치 지침만을 따르십시오. 설정 지침을 수행할 필요는 없습니다.
- 2002년 6월 14일 이전 주문된 PRPQ 버전의 경우, 이 타스크를 완료하려면 iSeries Tools for Developers PRPQ 이전 버전 설치를 참조하십시오.

VNC 암호 파일 작성: 디폴트로 VNC에는 권한이 없는 사용자 액세스로부터 VNC 화면을 보호하는 암호 파일이 필요합니다. 암호화된 암호를 작성하는 방법은 사용 중인 PRPQ 버전에 따라 달라집니다.

- 2002년 6월 14일 이후 주문된 PRPQ 버전의 경우, iSeries 명령 프롬트에서 다음 명령을 사용하십시오.

  ```
  MKDIR DIR('/home/your_profile_name/.vnc')
  QAPTL/VNCPASSWD USEHOME(*NO) PWDFILE('/home/your_profile_name/.vnc/passwd')
  ```

- 2002년 6월 14일 이전 주문된 PRPQ 버전의 경우, iSeries 명령 프롬트에서 다음 명령을 사용하십시오.

  ```
  MKDIR DIR('/home/your_profile_name/.vnc')
  VNCSAVF/VNCPASSWD USEHOME(*NO) PWDFILE('/home/your_profile_name/.vnc/passwd')
  ```

VNC 서버를 시작한 사용자에게만 암호 파일이 필요합니다.

Java 시스템 등록 정보 구성

Java 시스템 등록 정보를 설정하십시오. 첫 번째 행에서는 원하는 JDK 버전(1.2나 1.3)용 Java를 구성하고, 두 번째 행에서는 NAWT가 가능하도록 합니다.

```
     java.version=version
os400.class.path.rawt=2
```

여기서 *version*은 사용하려는 JDK 버전에 따라 **1.2** 또는 **1.3**입니다.

Java 시스템 등록 정보 설정 방법에 대해서는, IBM Developer Kit for Java에 대한 iSeries 서버 사용자 정의를 참조하십시오.

VNC 서버 시작

VNC 서버를 시작하면 다음 단계를 수행하십시오.

1. PASE 쉘 시작:

   ```
   CALL QP2TERM
   ```

2. PASE 쉘에서 VNC 서버의 NAWT 구성 버전을 시작하십시오.

   ```
   /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
   ```

   서버가 시작되면, ″New ‘X’desktop is *systemname:1*″과 비슷한 메세지가 표시됩니다. 다음 단계를 위해 이 데스크탑명을 기억해 두십시오.

   주: 다른 VNC 서버를 시작하면 화면 번호(즉, 콜론의 오른쪽에 있는 번호)가 다를 수 있습니다. 동시에 실행 중인 각 VNC 서버에는 고유의 화면 번호가 필요합니다. vncserver_java로의 호출에서 표시장치 번호를 지정하지 않으면, vncserver_java 프로그램이 사용가능한 표시장치를 찾습니다. 다음 명령을 사용하여 vncserver_java 프로그램을 시작함으로써 특정 표시장치를 요청합니다.

   ```
   /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java :n
   ```

   여기서 *n*은 사용하려는 화면 번호입니다.

환경 변수 설정

NAWT로 Java를 실행하는 세션에서는, X 서버 위치와 X 권한 파일 위치를 Java에 알려야 합니다. 환경 변수를 설정하여 이 정보를 Java에 알려줍니다.

Java 프로그램을 실행하려는 세션에서는, DISPLAY 환경 변수를 사용자 시스템명과 표시장치 번호(즉, vncserver_java를 실행할 때 인쇄된 값)로 설정하십시오.

또한, XAUTHORITY 환경 변수를 /home/*your_profile_name*/.Xauthority로 설정하십시오.

예를 들어 iSeries 명령 프롬트에서 다음 명령을 입력하십시오.

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:1')
ADDENVVAR ENVVAR(XAUTHORITY) VALUE('/home/your_profile_name/.Xauthority')
```

주:

- VNC 서버가 시작되면 .Xauthority 파일이 작성되거나 수정됩니다. X 서버 권한지정에서는 X 서버 연결에서 보안 프로토콜을 강제실행하므로, 다른 사용자의 어플리케이션이 사용자의 X 서버 요청을 가로채지 않도록 합니다.
- 실제로 JVM을 시작하는 사용자만이 이 환경 변수를 설정해야 합니다. 예를 들어, 서브릿 환경에서는 서브릿 엔진을 시작하는 사용자만이 환경 변수를 설정해야 합니다.

설치 프로시듀어 확인

다음 명령을 사용하여 NAWT 테스트 Java 어플리케이션을 실행함으로써, 위 단계를 성공적으로 완료했는지 확인하십시오.

```
JAVA CLASS(NAWTtest) CLASSPATH('/QIBM/ProdData/Java400/')
```

테스트 어플리케이션은 통합 파일 시스템에서 /tmp/NAWTtest.jpg라고 하는 JPEG 인코드 이미지를 작성합니다. 어플리케이션이 Java 예외를 발생시키지 않고 파일을 작성했는지 확인하십시오. 이미지를 표시하려는 경우, 2진 모드를 사용하여 이미지 파일을 그래픽 가능 시스템으로 업로드할 수 있습니다.

iSeries Tools for Developer의 이전 버전 설치

2002년 6월 14일 이전에 주문한 iSeries Tools for Developers PRPQ(5799PTL)의 경우, PRPQ에는 VNC(Virtual Network Computing)의 사전 컴파일된 OS/400 PASE 가능한 버전이 들어 있지 않습니다.

다음 지침을 사용하여 향상된 PRPQ를 보유하고 있는지 여부를 판별하고, 이전 버전의 PRPQ를 보유하고 있는 경우에는 VNC를 설치하십시오.

향상된 PRPW가 있는지 판별: PRPQ 5799-PTL은 있지만 VNC가 들어있는 향상된 버전인지 확실하지 않으면, 다음 파일이 존재하는지 검사하십시오.

```
/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
```

향상된 버전 PRPQ에는 vncserver_java 파일이 있지만, 이전 버전에는 없습니다. vncserver_java가 iSeries 서버에 없으면, 최신 버전의 PRPW를 주문 설치하거나, 다음 지침에 따라 VNC 설치를 완료할 수 있습니다.

VNC 설치: iSeries Tools for Developers PRPQ의 이전 버전에서 VNC를 설치하려면, 다음 단계를 완료하십시오.

1. 다음 명령을 실행하여 iSeries 서버의 저장 파일을 작성하십시오.

```
crtlib  vncsavf
crtsavf vncsavf/vncpasswd
crtsavf vncsavf/vnc
crtsavf vncsavf/fonts
crtsavf vncsavf/icewm
```

2. 다음 리스트에 있는 링크를 클릭하여 Application Factory - iSeries Tools for Development 웹 사이트에서 사용자의 워크스테이션으로 저장 파일을 다운로드하십시오.
   - vnc.savf
   - vncpasswd.savf
   - fonts.savf
   - icewm.savf
3. 워크스테이션에서 다음 명령을 실행하여 FTP를 사용해서 사용자 워크스테이션에서 iSeries 서버로 저장 파일을 전송하십시오.

```
     ftp youriseriesserver
bin
cd /qsys.lib/vncsavf.lib
put vnc.savf
put vncpasswd.savf
put fonts.savf
put icewm.savf
quit
```

4. iSeries 서버에서 다음 명령을 실행하여 저장 파일을 복원하십시오.

```
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. VNC 암호 파일을 작성하여 NAWT를 계속 설치하십시오.

«

VNC 사용 추가 정보

»

이 페이지에서는 VNC(Virtual Network Computing) 사용에 대한 추가 정보를 설명합니다.

CL 프로그램에서 VNC 표시 서버 시작

다음의 예는 DISPLAY 환경 변수를 설정하고 CL 프로그램에서 VNC를 자동으로 시작하는 한 방법입니다. display :*n*을 아직 실행하지 않는 것으로 가정합니다. 또한 VNCPASSWD 명령을 실행하여 VNC에 필요한 암호 파일을 이미 작성한 것으로 가정합니다.

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
call qp2shell parm('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
```

여기서 *n*은 시작하려는 표시 번호를 나타내는 숫자 값입니다.

VNC 표시 서버 종료

시작한 VNC 서버를 종료하려면 다음과 같이 하십시오.

```
call qp2shell parm('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' '-kill' ':n')
```

여기서 *n*은 종료하려는 표시 번호를 나타내는 숫자 값입니다.

제 2 장 Java 보안

iSeries 서버에서 실행되는 대부분의 Java(TM) 프로그램들은 애플릿이 아닌 어플리케이션이므로, ″sandbox″ 보안 모델이 이들을 제한하지 않습니다. 보안 관점에서 Java 어플리케이션은 iSeries 서버상의 다른 프로그램과 동일한 보안 제한을 받습니다. iSeries 서버에서 Java 프로그램을 실행하려면 통합 파일 시스템에서 클래스 파일에 대한 권한이 있어야 합니다. 일단 프로그램이 시작되면 프로그램은 사용자 권한에서 실행됩니다.

허용 권한을 사용하여 프로그램을 실행 중인 사용자의 권한과 프로그램 소유자의 권한을 갖는 오브젝트에 액세스할 수 있습니다. 허용 권한은 원래 액세스 권한이 없었던 오브젝트에 사용자 권한을 임시로 제공합니다. 두 개의 새로운 허용 권한 매개변수인 USRPRF 및 USEADPAUT에 대한 자세한 내용은 CRTJVAPGM(Java 프로그램 작성) 명령 정보를 참조하십시오.

»

IBM Developer Kit for Java는 Java 어플리케이션을 위해 다음의 보안 피처를 제공합니다.

Java 보안 모델
JVM(Java Virtual Machine) 내의 바이트 코드 로더와 검증 프로그램도 Java 보안 모델을 사용하여 어느 정도의 Java 보안을 제공합니다. 애플릿에서와 같이, 바이트 코드 로더와 검증 프로그램은 바이트 코드가 유효한 지와, 자료 유형이 제대로 사용되는 지를 검사합니다. 또한 레지스터와 메모리가 올바르게 액세스되는지, 그리고 스택이 넘치거나(overflow) 부족(underflow)하지 않은지를 검사합니다. 이러한 체크를 통해 JVM(Java Virtual Machine)이 안전하게 시스템의 통합성을 지키면서 실행할 수 있습니다.

JCE(Java Cryptography Extension)
iSeries 서버에서의 JCE(Java Cryptography Extension) 구현은 Sun Microsystems, Inc.의 구현과 호환됩니다. 이 문서에서는 iSeries 구현의 고유한 특징을 다룹니다. 여기에서는 사용자가 JCE에 대한 일반 문서에 익숙한 것으로 가정합니다.

Java 보안 소켓 확장
JSSE(Java Secure Socket Extension)는 SSL(Secure Sockets Layer) 프로토콜의 Java 구현입니다. JSSE은 SSL 및 TLS(Transport Layer Security) 프로토콜에서는 클라이언트와 서버가 TCP/IP를 통해 보안 통신을 진행시킬 수 있습니다. 이 문서에서는 JSSE의 iSeries 구현에 대한 고유한 내용을 설명합니다. 사용자가 JSSE용 일반 문서에 익숙한 것으로 가정합니다.

JAAS(Java Authentication and Authorization Service)
JAAS(Java Authentication and Authorization Service)는 IBM Developer Kit for Java가 지원하는 또 하나의 보안 항목입니다. 현재, J2SDK(Java 2 Software Development Kit) 표준판은 코드가 시작된 위치와 코드를 서명한 사람에 기초하여 액세스 제어(코드 소스 기본액세스 제어)를 제공합니다. 그러나 코드를 실행하는 사람에 기초하여 추가 액세스 제어를 시행하는 능력은 부족합니다. JAAS는 Java 2 보안 모델에 이 지원을 추가하는 구조를 제공합니다.

JGSS(Java Generic Security Service)
JGSS(Java Generic Security Service)는 IBM Developer Kit for Java가 지원하는 다른 보안 항목입니다. JGSS에서는 어플리케이션 사이에서의 보안 메시징을 위한 일반 인터페이스를 제공합니다. JGSS는 비밀키, 공용키 또는 기타 보안 기법을 기반으로 다양한 보안 메카니즘을 지원합니다.

주: J2SDK, 버전 1.4에서 JAAS, JCE, JGSS 및 JSSE는 기본 JDK의 일부이며, 확장 기능으로 간주되지 않습니다. 이전 JDK 버전에서는 이러한 보안 항목들이 확장 기능입니다.

Java 보안 모델

모든 시스템에서 Java(TM) 애플릿을 다운로드할 수 있으므로, JVM 내에 보안 메카니즘이 존재하여 악성 애플릿에 대해 보호합니다. Java 런타임 시스템은 JVM(Java Virtual Machine)이 바이트 코드를 로드할 때 바이트 코드를 검증합니다. 이것은 바이트코드가 유효하고 코드가 JVM(Java Virtual Machine)이 Java 애플릿에 설정한 제한사항을 위반하지 않게 합니다. Java 애플릿은 수행할 수 있는 조작, 메모리에 액세스하는 방법 및 JVM(Java Virtual Machine)을 사용하는 방법에서 제한됩니다. Java 애플릿이 기본 오퍼레이팅 시스템 또는 시스템상의 자료에 대한 액세스를 얻는 것을 막기 위해 제한사항이 적소에 배치됩니다. Java 애플릿이 자체의 sandbox에서만 ″작동″될 수 있으므로 ″sandbox″ 보안 모델입니다.

″sandbox″ 보안 모델은 클래스 로드 프로그램, 클래스 파일 검증 프로그램 및 java.lang.SecurityManager 클래스의 조합입니다.

보안에 대한 자세한 정보는 Sun Microsystems, Inc.의 보안 문서 및 SSL을 사용하여 보안 어플리케이션을 참조하십시오.

JCE(Java Cryptography Extension)

JCE(Java(TM) Cryptography Extension) 1.2는 J2SDK(Java 2 Software Development Kit), 표준판의 표준 확장입니다. iSeries 서버에서의 JCE 구현은 Sun Microsystems, Inc.의 구현과 호환됩니다. 이 문서에서는 iSeries 구현의 고유한 특징을 다룹니다. 사용자가 JCE 부가 제품에 대한 일반 문서에 익숙한 것으로 가정합니다. 사용자가 이 서비스 및 iSeries 정보와 쉽게 작업할 수 있도록 Sun JCE 문서  로의 링크를 제공합니다.

iSeries 서버에서 암호화 레벨은 암호 액세스 제공업체 제품에 의해 제어됩니다. 이것은 5722-AC2 및 5722-AC3의 두 버전에서 사용할 수 있습니다. 5722-AC3 제품은 모든 암호화 알고리즘을 허용합니다. 5722-AC2 제품은 Triple-DES를 허용하지 않으며 대칭 알고리즘을 56비트로 제한하고 비대칭 알고리즘을 1024비트로 제한합니다.

이미 언급한 5722-AC2에 대한 제한사항을 제외하고 IBM JCE 제공업체는 다음의 알고리즘을 지원합니다.

- DES
- Triple-DES
- RC2

- RC4
- Blowfish
- RSA
- Diffie-Hellman
- DSA
- Mars
- MD2
- MD5
- SHA-1
- Seal

또한 난수 생성기도 제공합니다.

IBM JCE를 Java 1.2와 함께 사용하려면 /QIBM/ProdData/Java400/jdk12/lib/security/java.security 파일을 편집하십시오.

파일의 섹션은 다음과 같은 변경이 필요합니다.

```
#
# To use the IBMJCE security provider, you need to:
# 1) Install an IBM Cryptographic Access Provider Product
# 2) uncomment the second provider entry that follows.
#
# List of providers and their preference orders:
#
security.provider.1=sun.security.provider.Sun
#security.provider.2=com.ibm.crypto.provider.IBMJCE
```

IBM JCE를 Java 1.3과 함께 사용하려면 /QIBM/ProdData/OS400/Java400/jdk/lib/security/java.security 파일을 편집하십시오.

파일의 섹션은 다음과 같은 변경이 필요합니다.

```
#
# 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
```

Java 보안 소켓 확장

JSSE(Java(TM) Secure Socket Extension)은 보안 소켓 계층(SSL) 프로토콜의 Java 구현입니다. JSSE는 SSL 및 TLS(Transport Layer Security) 프로토콜을 사용하여 클라이언트와 서버가 TCP/IP에서 보안 통신을 실행할 수 있도록 합니다.

JSSE는 다음 기능을 제공합니다.

- 자료 암호화
- 리모트 사용자 ID 인증
- 리모트 시스템명 인증
- 클라이언트/서버 인증 수행
- 메세지 무결성 확인

J2SDK(Java 2 Software Development Kit, Standard Edition), 버전 1.4로 통합된 JSSE는 SSL 단독에서 보다 많은 기능을 제공합니다. 자세한 정보는 다음 주제를 참조하십시오.

SSL(JSSE, 버전 1.0.8) 사용
SSL은 서버와 클라이언트 인증 수단을 제공함으로써 기밀 유지 및 자료 무결성을 제공합니다. 모든 SSL 통신은 서버와 클라이언트와의 "핸드쉐이크"로 시작합니다. 핸드쉐이크 중에 SSL은 클라이언트와 서버가 서로 간의 통신을 위해 사용하는 Cipher suite를 조정합니다. Cipher suite는 SSL을 통해 사용할 수 있는 다양한 보안 피처의 조합입니다. J2SDK, 버전 1.4 이전의 지원되는 버전과 함께 SSL을 사용할 수 있습니다.

JSSE, 버전 1.4 사용
JSSE은 SSL과 TLS 모두의 기초 메카니즘을 추상화시키는 구조입니다. 기초 프로토콜의 복잡도와 특수성을 추상화시킴으로써, JSSE는 프로그래머가 안전하고 암호화된 통신을 사용할 수 있도록 하는 반면, 동시에 보안 약점을 가능한 최소화시킵니다. 이 정보는 J2SDK, 버전 1.4를 실행하는 iSeries 서버에서 JSSE를 사용하는 경우에만 적용됩니다.

주: 이 정보는 J2SDK, 버전 1.4와 함께 제공되는 JSSE 버전에 대한 것입니다. 이전 버전의 JSSE에 대해서는 Sun Java 웹 사이트의 Java Secure Socket Extention 을 참조하십시오.

SSL(JSSE, 버전 1.0.8) 사용

SSL(secure sockets layer)의 Java 구현인 JSSE(Java Secure Socket Extension, 버전 1.0.8)를 사용하여 Java(TM) 어플리케이션을 좀 더 안전하게 할 수 있습니다. SSL은 다음과 같이 하여 어플리케이션의 보안을 향상시킵니다.

- 암호화를 통해 통신 자료를 보호합니다.
- 리모트 사용자 ID를 인증합니다.
- 리모트 시스템명을 인증합니다.

주: SSL은 Java 어플리케이션의 소켓 통신을 암호화하기 위해 디지털 인증을 사용합니다. 디지털 인증은 보안 시스템, 사용자 및 어플리케이션을 식별하기 위한 인터넷 표준입니다. IBM DCM(Digital Certificate Manager)을 사용하여 디지털 인증을 제어할 수 있습니다. 자세한 정보는 IBM 디지털 인증 관리자를 참조하십시오.

SSL을 사용하여 Java 어플리케이션을 보안하려면 다음과 같이 하십시오.

- SSL을 지원하도록 iSeries 서버를 준비하십시오.
- 다음과 같이 하여 SSL을 사용할 수 있도록 Java 어플리케이션을 설계하십시오.
  - 소켓 팩토리를 아직 사용하지 않는 경우 소켓 팩토리 사용을 위한 Java 소켓코드 변경.
  - SSL을 사용하도록 Java 코드변경.
- 다음과 같이 하여 Java 어플리케이션을 보안하도록 디지털 인증을 사용하십시오.
  1. 사용할 디지털 인증 유형 선택.
  2. 어플리케이션을 실행할 때 디지털 인증 사용.

또한 QsyRegisterAppForCertUse API를 사용하여 Java 어플리케이션을 보안 어플리케이션으로서 등록할 수 있습니다. 자세한 정보는 QsyRegisterAppForCertUse를 참조하십시오.

SSL의 Java 버전에 대한 자세한 정보는 Sun Microsystems, Inc.의 Java Secure Socket Extension  을 참조하십시오.

보안 소켓층 지원을 위해 iSeries 서버 준비

보안 소켓층(SSL)을 사용할 수 있도록 시스템을 준비하려면 DCM(Digital Certificate Manager) LP를 설치해야 합니다.

- 

  5722-SS1 OS/400 - Digital Certificate Manager

또한 다음의 Cryptographic Access Provider LP 중 하나를 설치해야 합니다.

- 

  5722-AC1 Cryptographic Access Provider 40비트
- 5722-AC2 Cryptographic Access Providers 56비트
- 5722-AC3 Cryptographic Access Provider 128비트

또한 시스템의 디지털 인증에 액세스할 수 있는지 아니면 디지털 인증을 작성할 수 있는지도 확인해야 합니다. iSeries 디지털 인증 관리 및 인터넷에 대한 자세한 정보는 IBM Digital Certificate Manager 시작을 참조하십시오.

Cryptographic Access Providers

Cryptographic Access Providers는 사용할 시스템에 대해 많은 cipher suite를 제공합니다. cipher suite는 다양한 보안 피처의 조합입니다. 다음 리스트는 각 Cryptographic Access Providers가 제공하는 cipher suite입니다.

5722-AC1 Cryptographic Access Provider 40비트

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

5722-AC2 Cryptographic Access Provider 56비트

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_DES_CBC_MD5
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

5722-AC3 Cryptographic Access Provider 128비트

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC2_CBC_128_MD5
SSL_RSA_WITH_DES_CBC_MD5
SSL_RSA_WITH_3DES_EDE_CBC_MD5

국가 또는 영역에 따라서 선택할 수 있는 Cryptographic Access Providers가 제한될 수 있습니다. 일단 Cryptographic Access Providers를 로드하면 그 Access Provider가 제공하는 모든 cipher suite를 사용할 수 있습니다.

소켓 팩토리 사용을 위한 Java 코드 변경

기존의 코드를 사용하여 SSL(보안 소켓층)을 사용하려면 먼저 소켓 팩토리를 사용하도록 코드를 변경해야 합니다.

소켓 팩토리를 사용하려고 코드를 변경하려면 다음 단계를 수행하십시오.

1. 프로그램에 다음 행을 추가하여 SocketFactory 클래스를 가져오십시오.

   ```
   import javax.net.*;
   ```

2. SocketFactory 오브젝트의 인스턴스를 선언하는 행을 추가하십시오. 예를 들면 다음과 같습니다.

   ```
   SocketFactory socketFactory
   ```

3. SocketFactory 인스턴스를 SocketFactory.getDefault() 메소드와 동일하게 설정하여 초기설정하십시오. 예를 들면 다음과 같습니다.

   ```
   socketFactory = SocketFactory.getDefault();
   ```

   SocketFactory의 전체 선언은 다음과 같아야 합니다.

   ```
   SocketFactory socketFactory = SocketFactory.getDefault();
   ```

4. 기존의 소켓을 초기설정하십시오. 선언하는 각 소켓에 대해 소켓 팩토리에 SocketFactory 메소드 createSocket(host,port)을 호출하십시오.

   소켓 선언은 다음과 같아야 합니다.

   ```
   Socket s = socketFactory.createSocket(host,port);
   ```

   여기서,

- *s*는 작성되고 있는 소켓입니다.
- *socketFactory*는 2단계에서 작성된 SocketFactory입니다.
- *host*는 호스트 서버명을 표시하는 스트링 변수입니다.
- *port*는 소켓을 연결하는 포트 번호를 표시하는 정수 변수입니다.

이상의 모든 단계가 완료되면 코드에서 소켓 팩토리가 사용됩니다. 이 외에는 코드를 변경할 필요가 없습니다. 호출하는 모든 메소드와 소켓에 대한 모든 구문은 여전히 작용합니다.

소켓 팩토리를 사용하기 위해 변환되는 클라이언트 프로그램의 예에 대해서는 예: 서버 소켓 팩토리 사용을 위한 Java(TM) 코드 변경을 참조하십시오.

소켓 팩토리를 사용하도록 변환되는 클라이언트 프로그램의 예에 대해서는 예: 클라이언트 소켓 팩토리 사용을 위한 Java 코드 변경을 참조하십시오.

예: 서버 소켓 팩토리 사용을 위한 Java 코드 변경

다음 예는 간단한 소켓 클래스 simpleSocketServer에서 소켓 팩토리를 사용하여 모든 소켓을 작성할 수 있도록 간단한 소켓 클래스를 변경하는 방법을 보여줍니다. 첫 번째 예는 소켓 팩토리가 없는 simpleSocketServer 클래스를 보여줍니다. 두 번째 예는 소켓 팩토리가 있는 simpleSocketServer 클래스를 보여줍니다. 두 번째 예에서 simpleSocketServer의 이름을 factorySocketServer로 재명명합니다.

예 1: 소켓 팩토리가 없는 소켓 서버 프로그램

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
/* 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
```

예 2: 소켓 팩토리가 있는 간단한 소켓 서버 프로그램

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
/* 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();
  }

}
```

백그라운드 정보에 대해서는 소켓 팩토리 사용을 위한 Java(TM) 코드 변경을 참조하십시오.

예: 클라이언트 소켓 팩토리를 사용하도록 Java 코드 변경

다음 예는 간단한 소켓 클래스 simpleSocketClient에서 소켓 팩토리를 사용하여 모든 소켓을 작성할 수 있도록 간단한 소켓 클래스를 변경하는 방법을 보여줍니다. 첫 번째 예는 소켓 팩토리가 없는 simpleSocketClient 클래스를 보여줍니다. 두 번째 예는 소켓 팩토리가 있는 simpleSocketClient 클래스를 보여줍니다. 두 번째 예에서 simpleSocketClient의 이름을 factorySocketClient로 재명명합니다.

예 1: 소켓 팩토리가 없는 소켓 클라이언트 프로그램

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
/* 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.
```

예 2: 소켓 팩토리가 있는 간단한 소켓 클라이언트 프로그램

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
/* 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.
```

백그라운드 정보에 대해서는 소켓 팩토리를 사용하기 위한 Java(TM) 코드 변경을 참조하십시오.

보안 소켓 계층 사용을 위한 Java 코드 변경

소켓을 작성하도록 코드에 이미 소켓 팩토리를 사용한 경우 보안 소켓층(SSL) 지원을 프로그램에 추가할 수 있습니다. 코드에서 아직 소켓 팩토리를 사용하지 않는 경우, 소켓 팩토리 사용을 위한 Java(TM) 코드 변경을 참조하십시오.

SSL을 사용하려고 코드를 변경하려면 다음 단계를 수행하십시오.

1. SSL 지원 추가를 위한 javax.net.ssl.* 가져오기:

   ```
   import javax.net.ssl.*;
   ```

2. 초기설정을 위해 SSLSocketFactory을 사용하여 SocketFactory 선언:

   ```
   SocketFactory newSF = SSLSocketFactory.getDefault();
   ```

3. 이전에 SocketFactory를 사용했던 것과 동일한 방법으로 소켓을 초기설정하도록 새로운 SocketFactory 사용.

   ```
   Socket s = newSF.createSocket(args[0], serverPort);
   ```

이제 코드에 SSL 지원이 사용됩니다. 이 외에는 코드를 변경할 필요가 없습니다.

코드 예에 대해서는 예: 보안소켓층 사용을 위한 Java 클라이언트 변경과 예: 보안 소켓층 사용을 위한 Java 서버 변경을 참조하십시오.

예: 보안 소켓 계층 사용을 위한 Java 서버 변경

다음 예는 SSL(보안 소켓층)을 사용하기 위해 factorySocketServer라는 클래스를 변경하는 방법을 보여줍니다.

첫 번째 예는 SSL을 사용하지 않는 factorySocketServer 클래스를 보여줍니다. 두 번째 예는 SSL을 사용하여 factorySSLSocketClient로 재명명된 동일한 클래스를 보여줍니다.

예 1: SSL 지원이 없는 간단한 factorySocketServer 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

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

예 2: SSL 지원이 없는 간단한 factorySocketServer 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

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

백그라운드 정보에 대해서는 보안 소켓 계층 사용을 위한 Java(TM) 코드 변경을 참조하십시오.

예: 보안 소켓 계층을 사용하기 위한 Java 클라이언트 변경

다음 예는 SSL(보안 소켓층)을 사용하기 위해 factorySocketClient라는 클래스를 변경하는 방법을 보여줍니다.

첫 번째 예는 SSL을 사용하지 않는 factorySocketClient 클래스를 보여줍니다. 두 번째 예는 SSL을 사용하는 factorySSLSocketClient를 보여줍니다.

예 1: SSL 지원이 없는 간단한 factorySocketClient 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
/* 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.
```

예 2: SSL 지원이 없는 간단한 factorySocketClient 클래스

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// 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.
```

백그라운드 정보에 대해서는 보안 소켓 계층을 사용하기 위한 Java(TM) 코드 변경을 참조하십시오.

사용할 디지털 인증 선택

사용할 디지털 인증을 결정할 때 여러 가지 요소를 고려해야 합니다. 시스템의 디폴트 인증을 사용하거나 사용할 다른 인증을 지정할 수 있습니다.

다음과 같은 경우에 시스템의 디폴트 인증을 사용하는 것이 좋습니다.

- Java(TM) 어플리케이션에는 특정 보안 요구사항이 없습니다.
- Java 어플리케이션에 필요한 보안의 종류를 모르는 경우.
- 시스템의 디폴트 인증이 Java 어플리케이션의 보안 요구사항을 충족시키는 경우.

주: 시스템의 디폴트 인증을 사용하기로 결정한 경우 시스템 관리자와 체크하여 디폴트 시스템 인증이 작성되어 있는지 확인하십시오. 디지털 인증 관리에 대한 자세한 정보는 IBM Digital Certificate Manager 시작을 참조하십시오.

시스템의 디폴트 인증을 사용하지 않으려면 다른 인증을 선택해야 합니다. 다음 두 가지 유형의 인증에서 선택할 수 있습니다.

- **사용자 인증**: 어플리케이션의 사용자를 식별하는 인증.
- **시스템 인증**: 어플리케이션을 실행 중인 시스템을 식별하는 인증.

다음과 같은 경우에 사용자 인증을 사용해야 합니다.

- 어플리케이션을 클라이언트 어플리케이션으로 실행하는 경우.
- 인증이 어플리케이션에 대해 작업하고 있는 사용자를 식별하기를 원하는 경우.

다음과 같은 경우에 시스템 인증을 사용해야 합니다.

- 어플리케이션을 서버 어플리케이션으로 실행하는 경우.
- 인증이 어플리케이션을 실행 중인 시스템을 식별하기를 원하는 경우.

필요한 인증의 종류를 알면 액세스할 수 있는 인증 컨테이너에서 원하는 디지털 인증을 선택할 수 있습니다.

Java 어플리케이션 실행시 디지털 인증 사용

SSL(secure sockets layer)을 사용하려면, 디지털 인증을 사용하여 Java 어플리케이션을 실행해야 합니다.

사용할 디지털 인증을 지정하려면 다음 등록 정보를 사용하십시오.

- os400.certificateContainer
- os400.certificateLabel

예를 들어, 디지털 인증 MYCERTIFICATE를 사용하여 Java 어플리케이션 MyClass.class를 실행하고 MYCERTIFICATE가 디지털 인증 컨테이너 YOURDCC에 있으면 `java` 명령은 다음과 비슷합니다.

```
java -Dos400.certificateContainer=YOURDCC
     -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

아직 사용할 디지털 인증을 결정하지 못한 경우 사용할 디지털 인증 선택을 참조하십시오. 시스템의 디폴트 인증 컨테이너에 저장되는 시스템의 디폴트 인증을 사용할 것을 결정할 수도 있습니다.

시스템의 디폴트 디지털 인증을 사용하면 인증이나 인증 컨테이너를 지정할 필요가 없습니다. Java 어플리케이션이 자동으로 시스템의 디폴트 디지털 인증을 사용합니다.

iSeries 디지털 인증 관리 및 인터넷에 대한 자세한 정보는 IBM Digital Certificate Manager 시작을 참조하십시오.

디지털 인증 및 -os400.certificateLabel 등록 정보: 디지털 인증은 보안 시스템, 사용자 및 어플리케이션을 식별하기 위한 인터넷 표준입니다. 디지털 인증은 디지털 인증 컨테이너에 저장됩니다. 디지털 인증 컨테이너의 디폴트 인증을 사용하려는 경우 인증 레이블을 지정할 필요가 없습니다. 특정 디지털 인증을 사용하려면 다음 등록 정보를 사용하여 java 명령에 해당 인증의 레이블을 지정해야 합니다.

```
os400.certificateLabel=
```

예를 들어, 사용하려는 인증의 이름이 MYCERTIFICATE인 경우 입력하는 java 명령은 다음과 비슷합니다.

```
java -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

이 예에서, Java 어플리케이션 MyClass는 인증 MYCERTIFICATE를 사용합니다. MYCERTIFICATE는 MyClass에서 사용할 수 있도록 시스템의 디폴트 인증 컨테이너에 있어야 합니다.

디지털 인증 컨테이너 및 -os400.certificateContainer 등록 정보: 디지털 인증 컨테이너는 디지털 인증을 저장합니다. iSeries 시스템 디폴트 인증 컨테이너를 사용하려는 경우 인증 컨테이너를 지정할 필요가 없습니다. 특정 디지털 인증 컨테이너를 사용하려면 다음 등록 정보를 사용하여 java 명령에 해당 디지털 인증 컨테이너를 지정해야 합니다.

```
os400.certificateContainer=
```

예를 들어, 사용할 디지털 인증이 있는 인증 컨테이너의 이름이 들어 있는 MYDCC일 경우 입력하는 java 명령은 다음과 같습니다.

```
java -Dos400.certificateContainer=MYDCC MyClass
```

이 예에서는 이름이 MyClass.class인 Java 어플리케이션이 디폴트 디지털 인증 컨테이너 MYDCC에 있는 디지털 인증을 사용하여 시스템에서 실행됩니다. 어플리케이션에 작성하는 모든 소켓은 MYDCC에 있는 디폴트 인증을 사용하여 소켓을 식별하고 소켓의 모든 통신을 보안합니다.

디지털 인증 컨테이너에 있는 디지털 인증 MYCERTIFICATE를 사용하려는 경우 입력하는 java 명령은 다음과 비슷합니다.

```
java -Dos400.certificateContainer=MYDCC
     -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

Java 보안 소켓 확장, 버전 1.4 사용

JSSE(Java Secure Socket Extension)에서는 SSL(Secure Sockets Layer) 프로토콜과 TLS(Transport Layer Security) 프로토콜 모두를 사용하여 클라이언트와 서버 사이에서 안전하고 암호화된 통신을 제공합니다.

JSSE의 IBM 구현을 IBM JSSE라고 합니다. IBM JSSE에는 원시 iSeries JSSE 제공자와 순수 Java JSSE 제공자가 포함됩니다.

JSSE를 지원하도록 iSeries 서버를 구성하는 정보에 대해서는 다음 링크를 사용하십시오.

JSSE를 지원하도록 서버 구성
IBM JSSE를 사용하도록 iSeries 서버를 구성하는 방법을 알 수 있습니다. 정보에는 소프트웨어 요구사항, JSSE 제공자 변경 방법, 그리고 필요한 보안 등록 정보와 시스템 등록 정보가 들어 있습니다.

원시 iSeries JSSE 제공자 사용
JSSE KeyStore 클래스와 SSLConfiguration 클래스의 원시 iSeries 구현 사용에 대한 내용입니다.

JSSE 예
예 프로그램을 사용하여 어플리케이션에서 JSSE를 사용하는 방법을 알 수 있습니다. 예 Java 소스 코드에서는 클라이언트와 서버가 클라이언트와 서버 모두에서 SSLContext 오브젝트를 사용하여 보안 통신 환경을 작성하는 방법을 보여줍니다.

*JSSE*를 지원하도록 iSeries 서버 구성

iSeries 서버에서 J2SDK(Java 2 Software Development Kit), 버전 1.4를 사용할 경우, JSSE는 이미 구성되어 있습니다. 디폴트 구성에서는 원시 iSeries JSSE 제공자를 사용합니다.

소프트웨어 요구사항: J2SDK, 버전 1.4와 함께 JSSE를 사용하려면, iSeries 서버에 IBM 암호 액세스 제공자 128비트(5722-AC3)가 설치되어 있어야 합니다. 자세한 내용은 암호 액세스 제공자를 참조하십시오.

JSSE 제공자 변경: 원시 iSeries JSSE 제공자 대신 순수 Java JSSE 제공자를 사용하도록 JSSE를 구성할 수 있습니다. 일부 특정 JSSE 보안 등록 정보 및 Java 시스템 등록 정보를 변경함으로써 두 제공자 간의 전환을 수행할 수 있습니다. 자세한 정보는 다음 주제를 참조하십시오.

- JSSE 제공자
- JSSE 보안 등록 정보
- Java 시스템 등록 정보

보안 관리자: Java 보안 관리자가 사용 가능한 상태로 JGSS 어플리케이션을 실행하는 경우, 어플리케이션 네트워크 권한을 설정해야 합니다. 자세한 내용은 Java 2 SDK의 권한 에 있는 SSLPermission을 참조하십시오.

JSSE 제공자

IBM JSSE에는 원시 iSeries JSSE 제공자와 순수 Java JSSE 제공자가 들어 있습니다. 선택하는 제공자는 어플리케이션의 요구사항에 따라 차이가 있습니다.

두 JSSE 제공자 모두 JSSE 인터페이스 스펙을 맞습니다. 두 제공자는 서로와 통신할 수 있으며, 다른 SSL 또는 TLS 구현(비Java 구현에서 조차도)과 통신할 수 있습니다.

순수 Java JSSE 제공자: 순수 Java JSSE 제공자는 다음 피처를 제공합니다.

- 디지털 인증(예를 들면 JKS, PKCS12 등등)을 제어하고 구성하는 모든 유형의 KeyStore 오브젝트에 대해 작업합니다.
- 복수의 구현들에게서 JSSE 구성요소 조합을 사용할 수 있습니다(예를 들어, 원시 iSeries JSSE 제공자의 X509TrustManager를 사용하여 이 순수 Java JSSE 제공자의 SSLContext를 초기화할 수 있습니다).

IBMJSSE는 순수 Java 구현의 제공자 이름입니다. 여러 JSSE 클래스의 java.security.Security.getProvider() 메소드나 여러 getInstance() 메소드로 이 제공자의 이름을 대소문자 구분하여 전달해야 합니다.

원시 iSeries JSSE 제공자: 원시 iSeries JSSE 제공자는 다음 피처를 제공합니다.

- 원시 iSeries SSL 지원 사용
- 디지털 인증 관리자를 사용하여 디지털 인증 구성 및 제어
- 최적의 성능 제공

주: 원시 iSeries JSSE 제공자에게는 고유 iSeries 유형의 KeyStore가 필요합니다. JSSE 원시 iSeries JSSE 제공자는 다른 구현의 구성요소가 자신의 구현으로 접속되지 못하도록 합니다.

IbmISeriesSslProvider는 원시 iSeries 구현의 이름입니다. 여러 JSSE 클래스의 java.security.Security.getProvider() 메소드나 여러 getInstance() 메소드로 이 제공자의 이름을 대소문자로 구분하여 전달해야 합니다.

디폴트 JSSE 제공자 변경: 보안 등록 정보를 적절히 변경하여 디폴트 JSSE 제공자를 변경할 수 있습니다. 자세한 정보는 다음 주제를 참조하십시오.

- JSSE 보안 등록 정보

JSSE 제공자를 변경한 뒤, 시스템 등록 정보가 새 제공자에 필요한 디지털 인증 정보(keystore)의 구성을 제대로 지정했는지 확인하십시오. 자세한 정보는 다음 주제를 참조하십시오.

- Java 시스템 등록 정보

«

»

JSSE 보안 등록 정보

JVM(Java Virtual Machine)은 Java 마스터 보안 등록 정보 파일을 편집하여 설정한 많은 중요 보안 등록 정보를 사용합니다. 이 java.security 파일은 보통 iSeries 서버의 /QIBM/ProdData/Java400/jdk14/lib/security 디렉토리에 상주합니다.

다음 리스트에서는 JSSE 사용에 대한 여러 관련 보안 등록 정보에 대해 설명합니다. java.security 파일을 편집하기 위한 안내서로서 설명을 사용하십시오.

security.provider.<integer>

사용하려는 JSSE 제공자. 또한 암호 제공자 클래스를 고정으로 등록합니다. 다음 예에서처럼 다른 JSSE 제공자를 지정하십시오.

```
security.provider.5=com.ibm.as400.ibmonly.net.ssl.Provider
security.provider.6=com.ibm.jsse.IBMJSSEProvider
```

ssl.KeyManagerFactory.algorithm

디폴트 KeyManagerFactory 알고리즘을 지정합니다. 원시 iSeries JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

순수 Java JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

자세한 내용은 javax.net.ssl.KeyManagerFactory의 javadoc을 참조하십시오.

ssl.TrustManagerFactory.algorithm

디폴트 TrustManagerFactory 알고리즘을 지정합니다. 원시 iSeries JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

순수 Java JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

자세한 내용은 javax.net.ssl.TrustManagerFactory의 javadoc을 참조하십시오.

ssl.SocketFactory.provider

디폴트 SSL 소켓 팩토리를 지정합니다. 원시 iSeries JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.SocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLSocketFactoryImpl
```

순수 Java JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
```

자세한 내용은 javax.net.ssl.SSLSocketFactory의 javadoc을 참조하십시오.

ssl.ServerSocketFactory.provider

디폴트 SSL 서버 소켓 팩토리를 지정합니다. 원시 iSeries JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.ServerSocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLServerSocketFactoryImpl
```

순수 Java JSSE 제공자의 경우, 다음을 사용하십시오.

```
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

자세한 내용은 javax.net.ssl.SSLServerSocketFactory의 javadoc을 참조하십시오.

JSSE Java 시스템 등록 정보

어플리케이션에서 JSSE를 사용하려면, 구성 확인을 제공하기 위해 디폴트 SSLContext 오브젝트에 필요한 여러 등록 정보를 지정해야 합니다. 등록 정보 일부는 두 제공자 모두에 적용되는 반면, 다른 등록 정보는 원시 iSeries 제공자에만 적용됩니다.

원시 iSeries JSSE 제공자를 사용할 때 등록 정보를 하나도 지정하지 않은 경우, os400.certificateContainer의 디폴트 값은 *SYSTEM이며, 이는 JSSE가 시스템 인증 저장에서 디폴트 항목을 사용함을 의미합니다.

두 제공자 모두에 대해 작업하는 등록 정보: 다음 등록 정보는 두 JSSE 제공자 모두에 적용됩니다. 각 설명에는 적용 가능한 디폴트 등록 정보가 들어 있습니다.

javax.net.ssl.trustStore

디폴트 TrustManager가 사용하려는 KeyStore 오브젝트가 들어있는 파일 이름. 디폴트 값은 jssecacerts 또는 (jssecacerets가 없는 경우) cacerts입니다.

javax.net.ssl.trustStoreType

디폴트 TrustManager가 사용하려는 KeyStore 오브젝트의 유형. 디폴트 값은 KeyStore.getDefaultType 메소드가 리턴한 값입니다.

javax.net.ssl.trustStorePassword

디폴트 TrustManager가 사용하려는 KeyStore 오브젝트의 암호.

javax.net.ssl.keyStore

디폴트 KeyManager가 사용하려는 KeyStore 오브젝트가 들어있는 파일 이름.

javax.net.ssl.keyStoreType

디폴트 TrustManager가 사용하려는 KeyStore 오브젝트의 유형. 디폴트 값은 KeyStore.getDefaultType 메소드가 리턴한 값입니다.

javax.net.ssl.keyStorePassword

디폴트 KeyManager가 사용하려는 KeyStore 오브젝트의 암호.

iSeries 원시 JSSE 제공자와만 작업하는 등록 정보: 다음 등록 정보는 원시 iSeries JSSE 제공자에게만 적용됩니다.

os400.secureApplication

어플리케이션 ID. JSSE는 다음 등록 정보를 지정하지 않은 경우에만 이 등록 정보를 사용합니다.

- javax.net.ssl.keyStore
- javax.net.ssl.keyStoreType
- and javax.net.ssl.keyStorePassword

os400.certificateContainer

사용하려는 키링의 이름. JSSE는 다음 등록 정보를 지정하지 않은 경우에만 이 등록 정보를 사용합니다.

- javax.net.ssl.keyStore
- javax.net.ssl.keyStoreType
- javax.net.ssl.keyStorePassword
- os400.secureApplication

os400.certificateLabel

사용하려는 키링의 레이블. JSSE는 os400.certificateContainer 등록 정보를 설정하고 사용할 경우에만 이 등록 정보를 사용합니다.

추가 정보: 시스템 등록 정보에 대해서는 다음 주제를 참조하십시오.

- iSeries 서버의 J2SDK, 버전 1.4용 Java 시스템 등록 정보
- Sun Java 웹 사이트의 System Properties

원시 iSeries JSSE 제공자 사용

원시 iSeries JSSE 제공자는 JSSE 클래스 및 인터페이스의 전체 슈트를 제공합니다. 원시 iSeries 제공자를 효과적으로 사용하려면, 다음 정보를 참조하십시오.

- 361 페이지의『SSLContext.getInstance 메소드의 프로토콜 값』
- 361 페이지의『원시 iSeries KeyStore 구현』

- 『원시 iSeries 제공자 사용시 제한사항』
- SSLConfiguration의 javadoc 정보

SSLContext.getInstance 메소드의 프로토콜 값: 다음 표에서는 원시 iSeries JSSE 제공자의 SSLContext.getInstance 메소드에 대한 프로토콜 값을 식별하고 설명합니다.

프로토콜 값	지원되는 SSL 프로토콜
SSL	SSL 버전 2, SSL 버전 3 및 TLS 버전 1
SSLv2	SSL 버전 2
SSLv3	SSL 버전 3
TLS	SSL 버전 2, SSL 버전 3 및 TLS 버전 1
TLSv1	TLS 버전 1
SSL_TLS	SSL 버전 2, SSL 버전 3 및 TLS 버전 1

원시 iSeries KeyStore 구현: 원시 iSeries 제공자는 IbmISeriesKeyStore 유형의 KeyStore 클래스 구현을 제공합니다. 이 keystore 구현에서는 디지털 인증 관리자 지원에 대한 랩퍼를 제공합니다. keystore의 내용은 특별한 어플리케이션 ID나 키링 파일, 암호 및 레이블을 기본으로 합니다. JSSE는 디지털 인증 관리자의 keystore 항목을 로드합니다. 항목을 로드하기 위해, JSSE는 어플리케이션이 처음 keystore 항목이나 keystore 정보에 액세스하려 할 때의 해당 어플리케이션 ID나 키링 정보를 사용합니다. keystore를 수정할 수 없으므로, 디지털 인증 관리자를 사용함으로써 수행해야 하는 모든 구성 변경사항을 수행해야 합니다.

디지털 인증 관리자 사용에 대해서는 다음 주제를 참조하십시오.

디지털 인증 관리자

원시 iSeries 제공자 사용시 제한사항: 원시 iSeries JSSE 제공자가 작동하려면, JSSE 어플리케이션이 원시 구현의 구성요소만을 사용해야 합니다. 예를 들어 원시 iSeries JSSE 가능 어플리케이션은 순수 Java JSSE 제공자를 사용하여 작성한 X509KeyManager 오브젝트를 사용하여, 원시 iSeries JSSE 제공자를 사용하여 작성한 SSLContext 오브젝트를 성공적으로 초기화할 수 없습니다.

뿐만 아니라, IbmISeriesKeyStore 오브젝트 또는 com.ibm.as400.SSLConfiguration 오브젝트를 사용함으로써 원시 iSeries 제공자에서 X509KeyManager 및 X509TrustManager의 구현을 시작해야 합니다.

주: 이전에 언급한 제한사항은 이후 릴리스에서 변경될 수 있으므로, 원시 iSeries JSSE 제공자에서는 비원시 구성요소(예를 들면 JKS KeyStore 또는 IbmX509 TrustManagerFactory)와 접속할 수 있습니다.

»

예: IBM JSSE(Java Secure Sockets Extension)

JSSE 예에서는 클라이언트와 서버가 원시 iSeries JSSE 제공자를 사용하여 보안 통신이 가능한 문맥을 작성하는 방법을 보여줍니다.

주: 두 예 모두에서는 java.security 파일이 지정한 등록 정보에 관계없이 원시 iSeries JSSE 제공자를 사용합니다.

예: SSLContext 오브젝트를 사용한 SSL 클라이언트

이 클라이언트 프로그램 예에서는 SSLContext 오브젝트를 이용하며, 이는 ″MY_CLIENT_APP″ 어플리케이션 ID를 사용하기 위해 초기화됩니다. 이 프로그램은 java.security 파일에서 지정된 내용에 관계없이 원시 iSeries 구현을 사용합니다.

예: SSLContext 오브젝트를 사용한 SSL 서버

다음 서버 프로그램에서는 이전에 작성된 keystore 파일로 초기화된 SSLContext 오브젝트를 사용합니다. keystore 파일의 이름은 `/home/keystore.file`이고, keystore 암호는 `password`입니다.

예 프로그램에서 IbmISeriesKeyStore 오브젝트를 작성하려면 keystore 파일이 필요합니다. KeyStore 오브젝트는 MY_SERVER_APP를 어플리케이션 ID로 지정해야 합니다.

keystore 파일을 작성하기 위해 다음 명령을 사용할 수 있습니다.

- Qshell 명령 프롬트에서,

```
java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
  -storepass password -appid MY_SERVER_APP
```

  Qshell에서 Java 명령 사용에 대해서는 다음 주제를 참조하십시오.

  Qshell

- iSeries 명령 프롬트에서,

```
RUNJVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
     '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
```

다음 면책사항은 IBM JSSE 예 모두에 적용됩니다.

코드 예 면책사항

IBM은 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.

모든 샘플 예제는 IBM에 의해 예시 목적으로만 제공됩니다. 이러한 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.

여기에 포함된 모든 프로그램은 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 어떠한 종류의 보증 없이 ″현상태대로″ 제공됩니다.

예: SSLContext 오브젝트를 사용한 SSL 클라이언트

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
////////////////////////////////////////////////////////////////////////////
//
// 이 예 클라이언트 프로그램에서는 SSLContext 오브젝트를 사용하며, 이것은
// "MY_CLIENT_APP" 어플리케이션 ID를 사용하기 위해 초기화됩니다.
//
// 예에서는 java.security 파일에서 지정한 등록 정보에 관계없이 원시 iSeries JSSE 제공자를
// 사용합니다.
//
// Command syntax:
//     java -Djava.version=1.4 SslClient
//
// J2SDK 버전 1.이 디폴트로 사용되도록 구성한 경우 "-Djava.version=1.4"는
// 필요없습니다.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;

/**
 * SSL 클라이언트 프로그램.
 */
public class SslClient {

   /**
    * SslClient 기본 메소드.
    *
    * @param args 명령행 인수(사용되지 않음)
    */
   public static void main(String args[]) {
       /*
        * 예외 발생을 감지하도록 설정됩니다.
        */
          try {
          /*
           * 어플리케이션 ID를 지정하기 위해 SSLConfiguration 오브젝트를 초기화합니다.
           * "MY_CLIENT_APP"는 DCM(Digital Certificate Manager)으로 정확히 등록 및
           * 구성되어야 합니다.
           */
          SSLConfiguration config = new SSLConfiguration();
          config.setApplicationId("MY_CLIENT_APP"
          /*
           * SSLConfiguration 오브젝트로부터 keystore를 가져옵니다.
           */
          Char[] password = "password".toCharArray();
          KeyStore ks = config.getKeyStore(password);
          /*
           * KeyManagerFactory를 할당하고 초기화합니다.
           */
          KeyManagerFactory kmf =
                              KeyManagerFactory.getInstance("IbmISeriesX509");
          Kmf.init(ks, password);
          /*
           * TrustManagerFactory를 할당하고 초기화합니다.
           */
```

```
            TrustManagerFactory tmf =
                            TrustManagerFactory.getInstance("IbmISeriesX509");
            tmf.init(ks);
            /*
             * SSLContext를 할당하고 초기화합니다.
             */
            SSLContext c =
                       SSLContext.getInstance("SSL", "quot;);
            C.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
            /*
             * SSLContext로부터 SSLSocketFactory를 가져옵니다.
             */
            SSLSocketFactory sf = c.getSocketFactory();
            /*
             * SSLSocket을 작성합니다.
             *
             * 하드코드된 IP 주소를 서버의 IP 주소나 호스트 이름으로 변경합니다.
             */
            SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
            /*
             * 보안 세션을 사용하여 서버로 메세지를 전송합니다.
             */
            String sent = "Test of java SSL write";
            OutputStream os = s.getOutputStream();
            os.write(sent.getBytes());
            /*
             * 결과를 화면에 기록합니다.
             */
            System.out.println("Wrote " + sent.length() + " bytes...");
            System.out.println(sent);
            /*
             * 보안 세션을 사용하여 서버에서 메세지를 수신합니다.
             */
            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);
            /*
             * 결과를 화면에 기록합니다.
             */
            System.out.println("Read " + received.length() + " bytes...");
            System.out.println(received);
         } catch (Exception e) {
            System.out.println("Unexpected exception caught: " +
                               e.getMessage());
                 e.printStackTrace();
        }
    }

}
```

예: SSLContext 오브젝트를 사용하는 SSL 서버

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
////////////////////////////////////////////////////////////////////////////
//
// 다음 서버 프로그램은 이전에 작성된 keystore 파일로 초기화한 SSLContext
// 오브젝트를 이용합니다.
//
// keystore 파일에는 다음 이름과 keystore 암호가 있습니다.
//   파일명: /home/keystore.file
//   암호:  password
//
// IbmISeriesKeyStore 오브젝트를 작성하려면 예 프로그램에 keystore 파일이 필요합니다.
// KeyStore 오브젝트는 MY_SERVER_APP를 오브젝트 ID로 지정해야 합니다.
//
// keystore 파일을 작성하기 위해 다음 Qshell 명령을 사용할 수 있습니다.
//
//   java com.ibm.as400.SSLConfiguration  -create  -keystore /home/keystore.file
//     -storepass password  -appid MY_SERVER_APP
//
// 명령 구문:
//     java -Djava.version=1.4 JavaSslServer
//
// J2SDK 버전 1.이 디폴트로 사용되도록 구성될 경우에는 "-Djava.version=1.4"가
// 필요없습니다.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;

/**
* 어플리케이션 ID를 사용한 Java SSL 서버 프로그램.
*/
public class JavaSslServer {

   /**
    * JavaSslServer 기본 메소드.
    *
    * @param은 명령행 인수입니다(사용되지 않음)
    */
   public static void main(String args[]) {
       /*
        * 예외 발생을 감지하도록 설정됩니다.
        */
           try {
           /*
            * keystore 오브젝트를 할당하고 초기화합니다.
            */
           Char[] password = "password".toCharArray();
           KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
           FileInputStream fis = new FileInputStream("/home/keystore.file"
           Ks.load(fis, password);
           /*
            * KeyManagerFactory를 할당하고 초기화합니다.
            */
           KeyManagerFactory kmf =
                             KeyManagerFactory.getInstance("IbmISeriesX509");
```

```
            Kmf.init(ks, password);
            /*
             * TrustManagerFactory를 할당하고 초기화합니다.
             */
            TrustManagerFactory tmf =
                            TrustManagerFactory.getInstance("IbmISeriesX509");
            tmf.init(ks);
            /*
             * SSLContext를 할당하고 초기화합니다.
             */
            SSLContext c =
                       SSLContext.getInstance("SSL", "IbmISeriesSslProvider");
            C.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
            /*
             * SSLContext에서 SSLServerSocketFactory를 가져옵니다.
             */
            SSLServerSocketFactory sf = c.getSSLServerSocketFactory();
            /*
             * SSLServerSocket을 작성합니다.
             */
            SSLServerSocket ss =
                (SSLServerSocket) sf.createServerSocket(13333);
            /*
             * accept()를 수행하여 SSLSocket을 작성합니다.
             */
            SSLSocket s = (SSLSocket) ss.accept();
            /*
             * 보안 세션을 사용하여 클라이언트에게서 메세지를 수신합니다.
             */
            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);
            /*
             * 결과를 화면에 기록합니다.
             */
            System.out.println("Read " + received.length() + " bytes...");
            System.out.println(received);
            /*
             * 보안 세션을 사용하여 메세지를 다시 클라이언트로 에코합니다.
             */
            OutputStream os = s.getOutputStream();
            os.write(received.getBytes());
            /*
             * 결과를 화면에 기록합니다.
             */
            System.out.println("Wrote " + received.length() + " bytes...");
            System.out.println(received);
         } catch (Exception e) {
            System.out.println("Unexpected exception caught: " +
                               e.getMessage());
                 e.printStackTrace();
        }
    }

}
```

제 3 장 JAAS(Java Authentication and Authorization Service)

JAAS(Java(TM) Authentication and Authorization Service)는 표준판인 J2SDK(Java 2 Software Development Kits)의 표준 확장입니다. 현재, J2SDK는 코드가 시작된 위치와 코드를 서명한 사람에 기초하여 액세스 제어(코드 소스 기본 액세스 제어)를 제공합니다. 그러나 코드를 실행하는 사람에 기초하여 추가 액세스 제어를 시행하는 능력은 부족합니다. JAAS는 Java 2 보안 모델에 이 지원을 추가하는 구조를 제공합니다.

JAAS API는 IBM과 Sun Microsystems, Inc.에서 J2SDK

버전 1.2와 1.3

에 대한 부가 제품으로 사용합니다. IBM과 Sun은 특정 사용자나 ID를 현재 Java 스레드에 연관시킬 수 있도록 이 부가 제품을 소개하고 있습니다. 이 작업은 javax.security.auth.Subject 메소드를 사용하고 선택적으로 com.ibm.security.auth.ThreadSubject 메소드를 사용하여 기저의 오퍼레이팅 시스템 스레드를 통해 수행됩니다.

주: J2SDK 버전 1.4의 경우에 JAAS는 더 이상 부가 제품이 아니지만 기본 SDK의 일부입니다.

iSeries 서버에서의 JAAS 구현은 Sun Microsystems, Inc.의 구현과 호환됩니다. 이 문서에서는 iSeries 구현의 고유한 특징을 다룹니다. 사용자가 JAAS 부가 제품에 대한 일반 문서에 익숙한 것으로 가정합니다. 사용자가 이 서비스와 iSeries 정보에 보다 쉽게 작업할 수 있도록 다음의 링크를 제공합니다.

- API Developers Guide는 소프트웨어 개발시 JAAS API를 사용에 대한 정보를 제공합니다.
- Login/Authentication Module Developers Guide는 JAAS의 인증 측면을 중심으로 설명합니다.
- JAAS API Specification에는 JAAS에 대한 Javadoc 정보가 들어 있습니다.

JAAS를 사용하는 방법에 대한 자세한 내용은 다음 주제 중 하나를 선택하십시오.

- JAAS를 위한 iSeries 서버 준비 및 구성
- JAAS 샘플
- iSeries 서버 특정 JAAS Javadoc

JAAS(Java Authentication and Authorization Service)용 iSeries 서버 준비 및 구성

JAAS(Java(TM) Authentication and Authorization Service)를 사용하기 위해서는 소프트웨어 요구사항을 만족시키고 iSeries 서버를 구성해야 합니다.

iSeries 서버에서 JAAS 1.0을 실행하기 위한 소프트웨어 요구사항

다음의 사용권 프로그램을 설치하십시오.

- Java 2 SDK, 버전 1.4(J2SDK)
- IBM Toolbox for Java(mod 4) 사용권 프로그램(5722-JC1)은 OS 스레드 ID를 변경해야 합니다. 여기에는 iSeries OS 스레드 ID 변경과 원시 구현 클래스를 지원하기 위해 필요한 ProfileTokenCredential 클래스가 들어 있습니다.

시스템 구성

JAAS를 사용하기 위해 시스템을 구성하려면 다음의 단계를 따르십시오.

1. JDKs 1.2 및 1.3에서 확장 디렉토리에 jaas13.jar 파일에 대한 기호 링크를 추가하십시오. 확장 클래스 로더가 JAR 파일을 로드해야 합니다. 링크를 추가하기 위해 iSeries 명령행에서 다음의 명령을 실행하십시오(모두 한 행에).

   ```
   ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas13.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jaas13.jar')
   ```

   주: JDK 1.4의 경우에는 확장 디렉토리에 기호 링크를 추가할 필요가 없습니다. JAAS는 이 버전에 대한 기본 SDK의 일부입니다.

2. 디폴트 login.config 파일이 com.ibm.as400.security.auth.login.BasicAuthenticationLoginModule을 호출하는 ${java.home}/lib/security에 제공됩니다. 이 login.config 파일이 일회용 ProfileTokenCredential을 인증된 주제에 접속합니다. 사용자 자신의 login.config 파일을 다른 옵션으로 사용하려면 어플리케이션을 호출할 때 다음의 시스템 등록 정보를 포함시킬 수 있습니다.

   ```
   -Djava.security.auth.login.config=your login.config file
   ```

3. 확장 디렉토리에 jt400Native.jar 파일에 대한 기호 링크를 추가하십시오. 그러면, 확장 클래스 로더는 이 파일을 로드할 수 있습니다. jaas13.jar 파일은 IBM Toolbox for Java의 일부인 증명서 구현 클래스에 대해 이 JAR 파일을 요구합니다. 어플리케이션 클래스 로더는 CLASSPATH에 이를 포함시켜서 이 파일을 로드할 수도 있습니다. 이 파일이 클래스 경로 디렉토리에서 로드된 경우 확장 디렉토리에 기호 링크를 추가하지 마십시오.

jt400Native.jar 파일을 /QIBM/ProdData/Java400/jdk14/lib/ext 디렉토리에 기호 링크로 연결하면 서버의 모든 JDK 1.4 사용자들이 이 jt400Native.jar 버전으로 실행해야 합니다.

이것은 다양한 사용자들이 여러 가지 다른 버전의 IBM Toolbox for Java 클래스를 요구할 경우에 바람직하지 않습니다. 다른 옵션으로는 이전에 설명한 바와 같이 어플리케이션 CLASSPATH에 jt400Native.jar를 넣는 것이 있습니다. 또 하나의 옵션은 사용자 자신의 디렉토리에 기호 링크를 추가한 다음 어플리케이션을 호출할 때 java.ext.dirs 시스템 등록 정보를 지정하여 확장 디렉토리 classpath에 이 디렉토리를 포함시키는 것입니다.

jt400Native.jar 파일을 /QIBM/ProdData/Java400/jdk13/lib/ext 디렉토리에 링크로 연결하려면 iSeries 명령행에서 다음의 명령을 실행하여 링크를 추가하십시오.

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400Native.jar')
```

jt400Native.jar 파일을 /QIBM/ProdData/Java400/jdk14/lib/ext 디렉토리에 링크로 연결하려면 iSeries 명령행에서 다음의 명령을 실행하여 링크를 추가하십시오.

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
   NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')
```

jt400Native.jar 파일을 사용자 자신의 디렉토리에 링크로 연결하려면 다음과 같이 하십시오.

a. iSeries 명령행에서 다음의 명령을 실행하여 링크를 추가하십시오.

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
            NEWLNK('your extension directory/jt400Native.jar')
```

b. Java 프로그램을 호출할 때 다음의 패턴을 사용하십시오.

```
java -Djava.ext.dirs=your extension directory:default
extension directories
```

주: iSeries 증명서 클래스에 대한 정보는 IBM Toolbox for Java를 참조하십시오. **보안 클래스**를 클릭하십시오. **인증 서비스**를 클릭하십시오. **ProfileTokenCredential** 클래스를 클릭하십시오. **패키지**를 클릭하십시오.

4. IBM Toolbox for Java JAR 파일의 실제 위치에 적합한 허가를 부여하기 위해 Java 2 정책 파일을 갱신하십시오. 이러한 파일들이 확장 디렉토리에 기호 링크로 연결되고 이러한 디렉토리들이 ${java.home}/lib/security/java.policy 파일에서 java.security.AllPermission을 부여받더라도 권한 부여는 JAR 파일의 실제 위치에 기초합니다.

   IBM Toolbox for Java에서 증명서 클래스를 성공적으로 사용하려면 어플리케이션의 Java 2 정책 파일에 다음을 추가하십시오.

```
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";
 }
```

또한 IBM Toolbox for Java JAR 파일이 수행하는 조작이 권한 모드에서 실행되지 않으므로 어플리케이션의 codeBase에 대한 허가를 추가해야 합니다.

Java 2 정책 파일에 대한 정보는 API Developers Guide를 참조하십시오.

5. iSeries 호스트 서버가 시작되어 실행되고 있는지 확인하십시오. Toolbox에 상주하는 ProfileTokenCredential 클래스(예: jt400Native.jar)는 인증된 주제에 첨부된 증명서로 사용됩니다. 증명서 클래스는 호스트 서버에 대한 액세스를 요구합니다. iSeries 명령 프롬프트에 다음과 같이 입력하여 서버가 시작되어 실행되고 있는지 확인할 수 있습니다.
   - StrHostSVR *all
   - StrTcpSvr *DDM

   서버가 이미 시작된 경우 이러한 단계는 아무 효과가 없습니다. 서버가 시작되지 않은 경우 시작됩니다.

JAAS(Java Authentication and Authorization Service) 샘플

이 정보에서는 iSeries 서버에서 JAAS(Java(TM) Authentication and Authorization Service)의 몇몇 샘플로의 링크를 제공합니다. 문서에는 두 개의 JAAS 샘플인 HelloWorld와 SampleThreadSubjectLogin이 포함됩니다. 지침과 소스 코드는 다음의 링크를 클릭하십시오.

- HelloWorld
- SampleThreadSubjectLogin

제 4 장 IBM JGSS(Java Generic Security Service)

JGSS(Java Generic Security Service)에서는 인증 및 보안 메시징을 위한 일반 인터페이스를 제공합니다. 이 인터페이스하에서는 비밀키, 공용키 또는 기타 보안 기법을 기반으로 다양한 보안 메카니즘을 접목할 수 있습니다.

기초 보안 메카니즘의 복잡도와 특수성을 표준화된 인터페이스로 추상화시킴으로써, JGSS는 보안 네트워크 어플리케이션 개발 시 다음의 이점을 제공합니다.

- 단일 추상 인터페이스로 어플리케이션을 개발합니다.
- 변경하지 않고 서로 다른 보안 메카니즘과 함께 어플리케이션을 사용할 수 있습니다.

JGSS는 GSS-API(Generic Security Service Application Programming Interface)용 Java 바인딩을 정의합니다. 이것은 IETF(Internet Engineering Task Force)에 의해 표준화되고 X/Open 그룹에 의해 채택된 암호 API입니다.

IBM의 JGSS 구현을 IBM JGSS라고 합니다. IBM JGSS는 Kerberos V5를 디폴트 기초 보안 시스템으로 사용하는 GSS-API 구조를 구현한 것입니다. 또한 Kerberos 증명서를 작성하고 사용하기 위한 JAAS(Java(TM) Authentication and Authorization Service) 로그인 모듈의 기능도 있습니다. 뿐만 아니라, 사용자가 이들 증명서를 사용할 때 JGSS가 JAAS 인증 점검을 수행하도록 할 수 있습니다.

IBM JGSS에는 원시 iSeries JGSS 제공자, Java JGSS 제공자 및 Java 버전의 Kerberos 증명서 관리 툴(kinit, ktab 및 klist)이 포함됩니다.

주: 원시 iSeries JGSS 제공자는 원시 iSeries NAS(Network Authentication Services) 라이브러리를 사용합니다. 원시 제공자를 사용하는 경우, 원시 iSeries Kerberos 유틸리티를 사용해야 합니다. 자세한 내용은 JGSS 제공자를 참조하십시오.

JGSS 사용에 대해서는 다음 주제를 참조하십시오.

JGSS 개념
GSS-API 동작에 대한 상위 레벨 설명과 보안 메카니즘에 대한 간단한 설명을 포함하여, JGSS 개념에 대해 소개합니다.

JGSS를 사용하도록 서버 구성
J2SDK(Java(TM) 2 Software Development Kit, Standard Edition)와 함께 IBM JGSS를 사용하도록 iSeries 서버를 구성하는 방법을 알려줍니다. 정보에는 보안 관리자와 함께 JGSS를 사용하는 데 필요한 권한의 식별과 설정이 포함됩니다.

JGSS 어플리케이션 실행
iSeries 서버에서 JGSS 어플리케이션을 실행하는 내용에 대해 알려줍니다. 문서에는 동작 개념에 대한 설명과, JAAS 사용을 위한 지침이 들어 있습니다.

JGSS 어플리케이션 개발
JGSS를 사용하여 보안 어플리케이션을 개발하는 방법을 볼 수 있습니다. 전송 토큰 생성, JGSS 오브젝트 작성, 문맥 설정 등에 대해 다룹니다.

JGSS javadoc 참조 정보
org.ietf.jgss api 패키지에 있는 클래스와 메소드, Java 버전의 Kerberos 증명서 관리 툴(kinit, ktab 및 klist)에 대한 javadoc 정보를 검토합니다.

JGSS 샘플
샘플 프로그램을 사용하여 사용자의 어플리케이션에서 JGSS를 사용하는 방법을 찾습니다. 샘플 문서에는 Java 소스 코드, 샘플, 구성 및 정책 파일을 실행하기 위한 지침이 들어 있습니다.

Java 보안 및 일반 보안 서비스에 대한 자세한 내용은 다음 문서를 참조하십시오.

- Sun Microsystems, Inc.의 J2SDK Security enhancement
  더 많은 Java GSS-API 정보로의 링크가 들어 있습니다.
- 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
- X/Open Group GSS-API Extensions for DCE

주: 중요 법적 고지사항에 대해서는 코드 면책사항 관련정보를 참조하십시오.

«

»

JGSS 개념

JGSS 동작은 4개의 고유한 스테이지로 구성되며, 이들은 GSS-API(Generic Security Service Application Programming Interface)에 의해 표준화된 상태입니다.

1. 프린시펄용 증명서 수집
2. 프린시펄간 통신간 보안 문맥 작성 및 설정
3. 피어간 보안 메세지 교환
4. 자원 클린업 및 릴리스

이와 함께, JGSS는 Java 암호 구조를 사용하여 서로 다른 보안 메카니즘간의 원활한 연결을 제공합니다.

다음 링크를 사용하여 이들 중요 JGSS 개념에 대한 고급 설명을 읽으십시오.

- 프린시펄 및 증명서

- 문맥 설정
- 메세지 보호 및 교환
- 자원 클린업 및 릴리스
- 보안 메카니즘

프린시펄 및 증명서

어플리케이션이 피어와의 JGSS 보안 통신에 관여하는 ID를 프린시펄이라고 합니다. 프린시펄은 실제 사용자거나 무인 서비스일 수 있습니다. 프린시펄에서는 해당 메카니즘의 ID 증거로서 보안 메카니즘 고유의 증명서를 받습니다. 예를 들어 Kerberos 메카니즘을 사용하는 경우, 프린시펄의 증명서는 Kerberos KDC(Kerberos key distribution center)에서 발행한 TGT(ticket-granting ticket) 양식으로 되어 있습니다. 복수 메카니즘 환경에서 GSS-API 증명서에는 여러 증명서 요소가 들어 있으며, 이들 각각은 기초가 되는 메카니즘 증명서를 나타냅니다.

GSS-API 표준에서는 프린시펄이 증명서를 확보하는 방법에 대해 규정하지 않으며, GSS-API 구현은 보통 증명서 확보를 위한 방법을 제공하지 않습니다. 프린시펄은 GSS-API를 사용하기 전에 증명서를 확보합니다. GSS-API는 단지 프린시펄을 위해 증명서의 보안 메카니즘을 조회만 합니다.

IBM JGSS에는 Java 버전의 Kerberos 증명서 관리 툴 kinit, ktab 및 klist가 들어 있습니다. 이와 함께, IBM JGSS는 JAAS를 사용하는 선택적 Kerberos 로그인 인터페이스를 제공함으로써 표준 GSS-API를 향상시킵니다. 순수한 Java JGSS 제공자는 선택적 로그인 인터페이스를 지원하지만, 원시 iSeries 제공자는 지원하지 않습니다. 자세한 정보는 다음 주제를 참조하십시오.

- Kerberos 증명서 확보
- JGSS 제공자

문맥 설정

보안 증명서를 확보한 두개의 통신 피어들은 각자의 증명서를 사용하여 보안 문맥을 설정합니다. 피어가 하나의 결합 문맥을 설정하더라도, 각 피어는 자신의 문맥 로컬 사본을 유지합니다. 문맥 설정에는 승인 피어에 대해 자신을 인증하는 개시 피어가 포함됩니다. 개시자는 선택적으로 상호 인증을 요청할 수 있으며, 이 경우에 수신자는 개시자에 대해 자신을 인증합니다.

문맥 설정이 완료되면, 설정된 문맥에는 두 피어 사이에서 후속 보안 메세지가 교환될 수 있도록 하는(공유 암호 키같은) 상태 정보가 들어갑니다.

메세지 보호 및 교환

두 피어가 문맥을 설정한 다음에는 보안 메세지를 교환할 준비가 된 것입니다. 메세지 발행자는 자신의 로컬 GSS-API 구현을 호출하여 메세지를 코드화시키며, 메세지 무결성 및 선택적으로 메세지 기밀성을 확인합니다. 그런 다음 어플리케이션이 그 결과 토큰을 피어로 전송합니다.

피어의 로컬 GSS-API 구현에서는 다음 방식으로 설정된 문맥의 정보를 사용합니다.

- 메세지 무결성 확인
- 메세지 해독(메세지가 암호화된 경우)

자원 클린업 및 릴리스

자원을 정리하기 위해, JGSS 어플리케이션은 더 이상 필요없는 문맥을 삭제합니다. JGSS 어플리케이션이 삭제된 문맥에 액세스할 수 있더라도, 메세지 교환을 위해 이를 사용하면 예외가 발생됩니다.

보안 메카니즘

GSS-API는 하나 이상의 기본 보안 메카니즘 위에서의 추상 구조로 구성됩니다. 구조가 기본 보안 메카니즘과 대화하는 방법은 구현에 따라 다릅니다. 이러한 구현은 두개의 일반 범주로 존재합니다.

- 한 쪽 측면에서는 통합 구현이 구조를 단일 메카니즘에 단단히 바인드합니다. 이 종류의 구현에서는 다른 메카니즘을 사용하거나 같은 메카니즘을 다르게 구현할 수 없습니다.
- 다른쪽 측면에서는 고도의 모듈화된 구현이 사용 편의성 및 유연성을 제공합니다. 이 종류의 구현에서는 서로 다른 메카니즘과 이들의 구현을 구조에 무리없이 용이하게 접합시킬 수 있습니다.

IBM JGSS는 후자 범주에 속합니다. 모듈화 구현으로서, IBM JGSS는 JCA(Java Cryptographic Architecture)가 정의한 제공자 구조를 사용하며, 모든 기초 메카니즘을 JCA 제공자로 처리합니다. JGSS 제공자는 JGSS 보안 메카니즘의 구체적 구현을 지원합니다. 어플리케이션은 복수의 메카니즘을 예시화하여 사용할 수 있습니다.

제공자가 복수의 메카니즘을 지원할 수 있으므로, JGSS가 다른 보안 메카니즘을 사용하기 쉽습니다. 그러나 복수 메카니즘을 사용할 수 있는 경우 GSS-API는 두 통신 피어가 한 메카니즘을 선택하는 방법을 제공하지 않습니다. 메카니즘을 선택하는 한가지 방법은 두 피어 사이의 실제 메카니즘을 절충하는 의사 메카니즘인 SPNEGO(Simple And Protected GSS-API Negotiating Mechanism)으로 시작하는 것입니다. IBM JGSS에는 SPNEGO 메카니즘이 없습니다.

SPNEGO에 대해서는, IETF(Internet Engineering Task Force) RFC 2478 SPNEGO(Simple and Protected GSS-API Negotiation Mechanism)  를 참조하십시오.

IBM JGSS를 사용하도록 iSeries 서버 구성

JGSS를 사용하도록 iSeries 서버를 구성하는 방법은 서버에서 실행하는 J2SDK(Java 2 Software Development Kit) 버전에 따라 달라집니다. JGSS를 사용하도록 iSeries 서버를 구성하는 정보에 대해서는 다음 링크를 사용하십시오.

- J2SDK, 버전 1.3과 함께 JGSS 사용
- J2SDK, 버전 1.4와 함께 JGSS 사용
- 원시 iSeries JGSS 제공자를 사용하도록 JGSS 구성

J2SDK, 버전 1.3과 함께 JGSS를 사용하도록 iSeries 서버 구성

iSeries 서버에서 J2SDK(Java 2 Software Development Kit), 버전 1.3을 사용할 경우, JSGG를 사용하도록 사용자 서버를 준비하고 구성해야 합니다. 디폴트 구성에서는 순수한 Java JGSS 제공자를 사용합니다.

소프트웨어 요구사항

J2SDK, 버전 1.3과 함께 JGSS를 사용하려면, 사용자 서버에 JAAS(Java Authentication and Authorization Service) 1.3이 설치되어 있어야 합니다.

JGSS를 사용하도록 서버 구성

J2SDK, 버전 1.3과 함께 JGSS를 사용하도록 서버를 구성하려면, ibmjgssprovider.jar 파일의 확장 디렉토리에 기호 링크를 추가하십시오. ibmjgssprovider.jar 파일에는 JGSS 클래스와 순수한 Java JGSS 제공자가 들어 있습니다. 기호 링크를 추가하면 확장 클래스 로더가 ibmjgssprovider.jar 파일을 로드할 수 있습니다.

기호 링크 추가

기호 링크를 추가하려면, iSeries 명령행에서 (한줄로) 다음 명령을 입력하고 **ENTER**를 누르십시오.

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssprovider.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/ibmjgssprovider.jar')
```

주: iSeries 서버의 디폴트 Java 1.3 정책에서는 JGSS에 대해 적합한 권한을 부여합니다. 자신만의 java.policy 파일을 작성하려는 경우, ibmjgssprovider.jar에 허용된 권한 목록은 JVM 권한을 참조하십시오.

JGSS 제공자 변경

순수한 Java 제공자를 디폴트로 사용하는 JGSS를 사용하도록 서버를 구성한 다음에는, 원시 iSeries JGSS 제공자를 사용하도록 JGSS를 구성할 수 있습니다. 원시 제공자를 사용하도록 JGSS를 구성한 다음에는 두 제공자 사이를 쉽게 전환할 수 있습니다. 자세한 정보는 다음 주제를 참조하십시오.

- JGSS 제공자
- 원시 iSeries JGSS 제공자를 사용하도록 JGSS 구성

보안 관리자

Java 보안 관리자가 사용 가능한 상태로 IBM JGSS 어플리케이션을 실행하는 경우, 보안 관리자 사용을 참조하십시오.

원시 iSeries JGSS 제공자를 사용하도록 JGSS 구성

IBM JGSS는 디폴트로 순수한 Java 제공자를 사용합니다. 원시 iSeries JGSS 제공자를 사용하도록 하는 옵션이 있습니다. 다른 제공자에 대해서는 JGSS 제공자를 참조하십시오.

소프트웨어 요구사항

원시 iSeries JGSS 제공자는 IBM Toolbox for Java내의 클래스에 액세스할 수 있어야 합니다. Toolbox for Java 액세스 방법에 대해서는 IBM Toolbox for Java에 액세스하기 위해 원시 iSeries JGSS 제공자 설정을 참조하십시오.

네트워크 인증 서비스가 구성되어 있는 지 확인하십시오. 자세한 정보는, 네트워크 인증 서비스를 참조하십시오.

원시 iSeries JGSS 제공자 지정

J2SDK, 버전 1.3과 함께 원시 iSeries JGSS 제공자를 사용하기 전에, JGSS를 사용하도록 사용자 서버를 구성했는지 확인하십시오. 자세한 정보는 J2SDK, 버전 1.3과 함께 JGSS를 사용하도록 iSeries 서버 구성을 참조하십시오. J2SDK, 버전 1.4를 사용 중인 경우, JGSS가 이미 구성되어 있습니다.

주: 다음 지침에서, ${java.home}은 서버에서 사용 중인 Java 버전의 위치 경로를 나타냅니다. 예를 들어 J2SDK, 버전 1.4를 사용 중인 경우, ${java.home}은 /QIBM/ProdData/Java400/jdk14입니다. 명령에서 ${java.home}을 Java 홈 디렉토리로의 실제 경로로 대체하십시오.

원시 iSeries JGSS 제공자를 사용하도록 JGSS를 구성하려면, 다음 타스크를 완료하십시오.

- 원시 iSeries 제공자 JAR 파일의 확장 디렉토리에 기호 링크 추가(377 페이지 참조)
- java.security 파일의 보안 제공자 리스트에 원시 iSeries JGSS 제공자 추가(377 페이지 참조)

기호 링크 추가

ibmjgssiseriesprovider.jar 파일의 확장 디렉토리에 기호 링크를 추가하려면, iSeries 명령행에서 (한줄로) 다음 명령을 입력한 다음 **ENTER**를 누르십시오.

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssiseriesprovider.jar')
NEWLNK('${java.home}/lib/ext/ibmjgssiseriesprovider.jar')
```

ibmjgssiseriesprovider.jar 파일의 확장 디렉토리로 기호 링크를 추가한 후에, 확장 클래스 로더가 JAR 파일을 로드합니다.

보안 제공자 리스트에 제공자 추가

java.security 파일에 있는 보안 제공자 리스트에 원시 제공자를 추가합니다.

1. ${java.home}/lib/security/java.security를 편집을 위해 여십시오.
2. 보안 제공자 리스트를 찾으십시오. java.security 파일의 맨 위 근처에 있어야 하며, 다음과 비슷합니다.

   ```
   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. 원시 Java 제공자 이전에 보안 제공자 리스트에 원시 iSeries JGSS 제공자를 추가하십시오. 즉, com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider를 com.ibm.jgss.IBMJGSSProvider보다 낮은 순서의 리스트에 추가한 다음, IBMJGSSProvider의 위치를 갱신하십시오. 예를 들면 다음과 같습니다.

   ```
   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
   ```

   IBMJGSSiSeriesProvider가 리스트에서 네 번째 항목이 되고, IBMJGSSProvider가 다섯 번째 항목이 되었는 지 확인하십시오. 또한, 보안 제공자 리스트의 항목 순서가 순차적인 지, 각 항목이 하나씩 증가하는 지 검사하십시오.
4. java.security 파일을 저장한 뒤 닫으십시오.

J2SDK, 버전 1.4와 함께 *JGSS*를 사용하도록 iSeries 서버 구성

iSeries 서버에서 J2SDK(Java 2 Software Development Kit), 버전 1.4를 사용하는 경우, JSGG가 이미 구성되었습니다. 디폴트 구성에서는 순수한 Java JGSS 제공자를 사용합니다.

JGSS 제공자 변경

순수한 Java JGSS 제공자 대신 원시 iSeries JGSS 제공자를 사용하도록 JGSS를 구성할 수 있습니다. 원시 제공자를 사용하도록 JGSS를 구성한 다음에는 두 제공자 사이를 쉽게 전환할 수 있습니다. 자세한 정보는 다음 주제를 참조하십시오.

- JGSS 제공자
- 원시 iSeries JGSS 제공자를 사용하도록 JGSS 구성

보안 관리자

Java 보안 관리자가 사용 가능한 상태로 JGSS 어플리케이션을 실행하는 경우, 보안 관리자 사용을 참조하십시오.

JGSS 제공자

IBM JGSS에는 원시 iSeries JGSS 제공자와 순수한 Java JGSS 제공자가 들어 있습니다. 사용하도록 선택한 제공자는 사용자의 어플리케이션 필요에 따라 다릅니다.

순수한 Java JGSS 제공자는 다음 피처를 제공합니다.

- 어플리케이션에 대해 최상위 이식성 제공
- 선택적 JAAS Kerberos 로그인 인터페이스에 대한 작업
- Java Kerberos 증명서 관리 툴과의 호환

원시 iSeries JGSS 제공자는 다음 피처를 제공합니다.

- 원시 iSeries Kerberos 라이브러리 사용
- Qshell Kerberos 증명서 관리 툴과의 호환
- JGSS 어플리케이션 빠른 실행

주: 두 JGSS 제공자 모두는 GSS-API 스펙을 유지하므로, 서로 호환됩니다. 즉, 순수한 Java JGSS를 사용하는 어플리케이션은 원시 iSeries JGSS 제공자를 사용하는 어플리케이션과 상호 동작할 수 있습니다.

JGSS 제공자 변경

주: 원시 iSeries JGSS 제공자로 변경하기 전에 J2SDK, 버전 1.3을 실행하는 경우, JGSS를 사용하도록 서버가 구성되었는 지 확인하십시오. 자세한 정보는 다음 주제를 참조하십시오.

- J2SDK, 버전 1.3과 함께 JGSS를 사용하도록 iSeries 서버 구성

- 원시 JGSS 제공자를 사용하도록 JGSS 구성

다음 옵션중 하나를 사용하여 JGSS 제공자를 쉽게 변경할 수 있습니다.

- `${java.home}/lib/security/java.security`에서 보안 제공자 리스트를 편집합니다.

  주: ${java.home}은 서버에서 사용 중인 Java 버전의 위치 경로를 나타냅니다. 예를 들어 J2SDK, 버전 1.3을 사용 중인 경우, ${java.home}은 /QIBM/ProdData/Java400/jdk13입니다.
- GSSManager.addProviderAtFront() 또는 GSSManager.addProviderAtEnd()를 사용하여 JGSS 어플리케이션에서 제공자 이름을 제공합니다. 자세한 정보는 GSSManager javadoc을 참조하십시오.

보안 관리자 사용

Java 보안 관리자가 사용 가능한 상태로 JGSS 어플리케이션을 실행하는 경우, 어플리케이션 및 JGSS에 필요한 권한이 있는 지 확인해야 합니다. JGSS를 사용하는 데 필요한 권한에 대한 자세한 정보는 다음 주제를 참조하십시오.

- JVM 권한
- JAAS 권한 검사

JVM 권한

JGSS가 수행하는 액세스 제어 검사는 물론 JVM(Java Virtual Machine)은 파일, Java 등록 정보, 패키지 및 소켓등을 포함한 여러 자원에 액세스할 때 권한 검사를 수행합니다.

JVM 권한 사용에 대해서는 Java 2 SDK의 권한  을 참조하십시오.

다음 리스트에서는 JGSS의 JAAS 피처를 사용하거나 보안 관리자와 함께 JGSS를 사용할 때 필요한 권한을 나타냅니다.

- 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 권한 검사

IBM JGSS는 JAAS 가능 프로그램이 증명서를 사용하고 서비스에 액세스할 때 런타임 권한 검사를 수행합니다. Java 등록 정보 javax.security.auth.useSubjectCredsOnly를 거짓으로 설정하여 이 선택적 JAAS 피처를 작동 불가능하게 할 수 있습니다. 뿐만 아니라 JGSS는 어플리케이션이 보안 관리자와 실행할 때에만 권한 검사를 수행합니다.

JGSS는 현재 액세스 제어 문맥에 영향을 주는 Java 정책에 대해 권한 검사를 수행합니다. JGSS는 다음 특정 권한 점검을 수행합니다.

- javax.security.auth.kerberos.DelegationPermission
- javax.security.auth.kerberos.ServicePermission

DelegationPermission 검사

DelegationPermission을 사용하여 보안 정책은 Kerberos의 티켓 전송 및 프록싱 피처에 대한 사용을 제어할 수 있습니다. 이들 피처를 사용하면, 클라이언트가 클라이언트를 위해 서비스를 작동시킬 수 있습니다.

DelegationPermission에서는 다음의 순서로 두 개의 인수를 취합니다.

1. 종속 프린시펄은 클라이언트를 위해 클라이언트의 권한으로 작동하는 서비스 프린시펄의 이름입니다.
2. 클라이언트가 종속 프린시펄에게 사용하도록 허용하려는 서비스 이름.

예: DelegationPermission 검사 사용

다음 예에서 superSecureServer는 종속 프린시펄이고, krbtgt/REALM.IBM.COM@REALM.IBM.COM은 superSecureServer에게 클라이언트를 위해 사용하도록 허용하려는 서비스입니다. 이런 경우, 서비스는 클라이언트에 대한 티켓 허가 티켓이고, 이것은 superSecureServer가 클라이언트를 위해 서비스용 티켓을 확보할 수 있음을 의미합니다.

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
```

이전 예에서, DelegationPermission은 클라이언트 권한이 superSecureServer만이 사용할 수 있는 KDC(Key Distribution Center)로부터 새로운 티켓 허용 티켓을 확보할 수 있도록 합니다. 클라이언트가 새 티켓 허용 티켓을 superSecureServer에게 전송한 뒤, superSecureServer는 클라이언트를 위해 작동하는 기능을 갖게 됩니다.

다음 예에서는 클라이언트가 seperSecureServer가 클라이언트를 위해 FTP 서비스에만 액세스하도록 하는 새로운 티켓을 확보할 수 있습니다.

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"ftp/ftp.ibm.com@REALM.IBM.COM\"";
```

자세한 정보는 Sun 웹 사이트의 J2SDK documentation 에 있는 javax.security.auth.kerberos.DelegationPermission 클래스를 참조하십시오.

ServicePermission 검사

ServicePermission 검사에서는 문맥 시작과 승인에서 증명서 사용을 제한합니다. 문맥 개시자는 문맥을 시작할 수 있는 권한이 있어야 합니다. 또한, 문맥 승인자는 문맥을 승인할 권한이 있어야 합니다.

예: ServicePermission 검사 사용

다음 예에서는 클라이언트에 권한을 부여함으로써 클라이언트가 ftp 서비스로 문맥을 시작할 수 있도록 합니다.

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "initiate";
```

다음 예에서는 서버에 권한을 부여함으로써 서버측이 ftp 서비스에 대한 비밀키에 액세스하여 시작할 수 있도록 합니다.

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "accept";
```

자세한 내용은 Sun 웹 사이트의 J2SDK documentation 에 있는 javax.security.auth.kerberos.ServicePermission 클래스를 참조하십시오.

IBM JGSS 어플리케이션 실행

IBM JGSS(Java Generic Security Service) API 1.0은 서로 다른 기초 보안 메카니즘의 복잡도 및 특수성으로부터 보안 어플리케이션을 보호합니다. JGSS에서는 JAAS(Java Authentication and Authorization Service) 및 IBM JCE(Java Cryptography Extension)가 제공하는 피처를 사용합니다.

JGSS 피처에는 다음이 포함됩니다.

- ID 인증
- 메세지 무결성 및 기밀성
- 선택적 JAAS Kerberos 로그인 인터페이스 및 인증 검사

JGSS 어플리케이션 실행에 대해서는 다음 주제를 참조하십시오.

Kerberos 증명서 확보
Kerberos 증명서 확보 방법과 비밀키 작성법을 알려줍니다. JAAS를 사용하여 Kerberos 로그인 및 인증 검사를 수행하는 방법에 대해 배우고, JVM(Java Virtual Machine)에 필요한 JAAS 권한 리스트를 검토합니다.

구성 및 정책 파일
JGSS를 실행하는 데 필요한 서로 다른 종류의 지원 파일에 대해 배웁니다. 여기에는 구성 파일, 정책 파일, Java 마스터 보안 등록 정보 파일 및 증명서 캐시가 포함됩니다.

디버깅
JGSS 디버깅을 사용하여 유용한 디버깅 메세지를 범주화하고 표시하는 내용에 대해 읽습니다.

JGSS 샘플
샘플 프로그램을 사용하여 JGSS 설정을 테스트하고 확인합니다. 샘플 문서에는 Java 소스 코드, 샘플을 실행하기 위한 지침, 구성 및 정책 파일 등등이 포함됩니다.

Kerberos 증명서 확보 및 비밀키 작성

GSS-API는 증명서 확보 방법을 정의하지 않습니다. 이러한 이유로, IBM JGSS Kerberos 메카니즘에서는 사용자가 다음 방법 중 한가지 방법으로 Kerberos 증명서를 확보해야 합니다.

- Kinit 및 Ktab 툴

- 선택적 JAAS Kerberos 로그인 인터페이스

Kinit 및 Ktab 툴

JGSS 제공자의 선택사항에서는 Kerberos 증명서와 비밀키를 확보하기 위해 사용하는 툴을 판별합니다.

순수 Java JGSS 제공자 사용

순수 Java JGSS 제공자를 사용 중인 경우, IBM JGSS Kinit 및 Ktab 툴을 사용하여 증명서와 비밀키를 확보하십시오. Kinit 및 Ktab 툴은 명령행 인터페이스를 사용하며, 다른 버전에서 제공하는 것과 비슷한 옵션들을 제공합니다.

- Kinit 툴을 사용하여 Kerberos 증명서를 확보할 수 있습니다. 이 툴은 KDC(Kerberos Distribution Center)에 연락하여 TGT(ticket-granting ticket)를 확보합니다. TGT를 사용하면 GSS-API를 사용하는 서비스를 포함하여 다른 Kerberos 가능 서비스에 액세스할 수 있습니다.
- 서버는 Ktab 툴을 사용하여 비밀키를 확보할 수 있습니다. JGSS는 서버의 키 표 파일에 비밀키를 저장합니다. 자세한 정보는 Ktab Java 문서를 참조하십시오.

뿐만 아니라, 어플리케이션에서는 JAAS 로그인 인터페이스를 사용하여 TFT 및 비밀키를 확보할 수 있습니다. 자세한 정보는 다음을 참조하십시오.

- Kinit javadoc
- Ktab javadoc
- JAAS 로그인 인터페이스

원시 iSeries JGSS 제공자 사용

원시 iSeries JGSS 제공자를 사용 중인 경우, Qshell kinit 및 klist 유틸리티를 사용하십시오. 자세한 정보는 Kerberos 증명서 및 키표의 유틸리티를 참조하십시오.

JAAS Kerberos 로그인 인터페이스

IBM JGSS는 JAAS(Java Authentication and Authorizaiton Service) Kerberos 로그인 인터페이스의 기능을 제공합니다. Java 등록 정보 javax.security.auth.useSubjectCredsOnly를 거짓으로 설정함으로써 이 피처를 작동 불가능하도록 설정할 수 있습니다.

주: 순수한 Java JGSS 제공자가 로그인 인터페이스를 사용할 수 있더라도, 원시 iSeries JGSS 제공자는 사용할 수 없습니다.

JAAS에 대해서는 JAAS(Java Authentication and Authorization Service)를 참조하십시오.

JAAS 및 JVM 권한

보안 관리자를 사용하는 경우, JGSS와 JGSS가 필요한 JVM 및 JAAS 권한을 가지고 있는 지 확인해야 합니다. 자세한 내용은 보안 관리자 사용을 참조하십시오.

JAAS 구성 파일 옵션

로그인 인터페이스에서는 로그인 모듈로서 사용될 com.ibm.security.auth.module.Krb5LoginModule을 지정하는 JAAS 구성 파일이 필요합니다. 다음 표에서는 Krb5LoginModule이 지원하는 옵션들을 나열합니다. 옵션에서는 대소문자를 구분하지 않습니다.

옵션 이름	값	디폴트	설명
principal	<스트링>	없음; 프롬트 표시	Kerberos 프린시펄 이름
credsType	개시자|승인자|모두	개시자	JGSS 증명서 유형
forwardable	참|거짓	거짓	이송 가능한 TGT(ticket-granting ticket)를 확보했는지 여부
proxiable	참|거짓	거짓	프록시 가능한 TGT(ticket-granting ticket)를 확보했는지 여부
useCcache	<URL>	ccache를 사용하지 마십시오.	지정된 증명서 캐시에서 TGT를 검색합니다.
useKeytab	<URL>	키 표를 사용하지 마십시오.	지정된 키 표에서 비밀키를 검색합니다.
useDefaultCcache	참|거짓	디폴트 ccache를 사용하지 마십시오.	디폴트 증명서 캐시에서 TGT를 검색합니다.
useDefaultKeytab	참|거짓	디폴트 키 표를 사용하지 마십시오.	지정된 키 표에서 비밀키를 검색합니다.

Krb5LoginModule 사용에 대한 간단한 예는 샘플 JAAS 로그인 구성 파일을 참조하십시오.

옵션 비호환성

프린시펄 이름을 제외한 일부 Krb5LoginModule 옵션들은 다른 옵션들과 호환되지 않습니다. 즉, 함께 지정할 수 없습니다. 다음 표는 호환되는 로그인 모듈 옵션과, 호환되지 않는 로그인 모듈 옵션을 나타냅니다.

표에 있는 인디케이터는 두 연관 옵션간의 관계를 설명합니다.

- X = 비호환
- N/A = 적용 불가능 조합
- Blank = 호환

Krb5LoginModule 옵션	**credsType 개시자**	**credsType 승인자**	**credsType 모두**	**이송**	**프록시**	**Ccache 사용**	**Keytab 사용**	**useDefaultCcache**	**useDefaultKeytab**
credsType=개시자		N/A	N/A				X		X
credsType=승인자	N/A		N/A	X	X	X		X	
credsType=모두	N/A	N/A							
이송가능		X				X	X	X	X
프록시 가능		X				X	X	X	X

Krb5LoginModule 옵션	credsType 개시자	credsType 승인자	credsType 모두	이송	프록시	Ccache 사용	Keytab 사용	useDefaultCcache	useDefaultKeytab
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	

프린시펄 이름 옵션

다른 옵션과 조합하여 프린시펄 이름을 지정할 수 있습니다. 프린시펄 이름을 지정하지 않으면, Krb5LoginModule이 사용자에게 프린시펄 이름을 입력하도록 프롬트를 표시합니다. Krb5LoginModule의 사용자에 대한 프롬트 제시 여부는 지정하는 다른 옵션에 따라 차이가 있습니다. 자세한 내용은 『프린시펄 이름 및 암호에 대한 프롬트』를 참조하십시오.

서비스 프린시펄 이름 형식

다음 형식 중 하나를 사용하여 서비스 프린시펄명을 지정해야 합니다.

- <service_name> (예를 들면 superSecureServer)
- <service_name>@<host> (예를 들면 superSecureServer@myhost)

후자의 형식에서, <host>는 서비스가 상주하는 기계의 호스트 이름입니다. 완전히 규정된 호스트 이름을 사용할 수 있습니다(꼭 사용해야 하는 것은 아닙니다).

주: JAAS는 특정 문자를 분리문자로 인식합니다. JAAS 스트링에서 다음 문자를 사용할 경우(예: 프린시펄 이름), 문자를 인용부호로 묶으십시오.

_ (밑줄)
: (콜론) / (슬래시) \ (백슬래쉬)

프린시펄 이름 및 암호에 대한 프롬트

JAAS 구성 파일에서 지정한 옵션들은 Krb5LoginModule 로그인이 대화식인지 아닌지를 결정합니다.

- 비대화식 로그인은 어떤 경우에도 정보에 대해 프롬트를 표시하지 않습니다.
- 대화식 로그인은 프린시펄 이름, 암호 또는 모두에 대해 프롬트를 표시합니다.

비대화식 로그인

증명서 유형을 개시자로 지정하고(`credsType=initiator`) 다음 조치 중 하나를 수행하면 로그인은 비대화식으로 진행됩니다.

- useCcache 옵션 지정
- useDefaultCcache 옵션을 참으로 설정

증명서 유형을 승인자나 모두로 지정하고(`credsType=acceptor` 또는 `credsType=both`) 다음 조치 중 하나를 수행하는 경우에도 로그인이 비대화식으로 진행됩니다.

- useKeytab 옵션 지정

- useDefaultKeytab 옵션을 참으로 설정

대화식 로그인

기타 다른 구성에서는 Kerberos KDC로부터 TGT를 확보할 수 있도록 로그인 모듈에서 프린시펄 이름과 암호에 대해 프롬트를 표시합니다. 로그인 모듈은 사용자가 프린시펄 옵션을 지정할 경우 암호에 대해서만 프롬트를 표시합니다.

대화식 로그인에서는 어플리케이션이 로그인 문맥을 작성할 때 콜백 핸들러로서 com.ibm.security.auth.callback.Krb5CallbackHandler를 지정해야 합니다. 콜백 핸들러는 입력을 위해 프롬트를 표시해야 합니다.

증명서 유형 옵션

증명서 유형이 개시자나 승인자(`credsType=both`)이어야 하는 경우, Krb5LoginModule은 TGT와 비밀키를 확보합니다. 로그인 모듈은 TGT를 사용하여 문맥을 승인할 문맥과 비밀키를 초기화합니다. JAAS 구성 파일에는 로그인 모듈이 두가지 유형의 증명서를 확보하기에 충분한 정보가 들어 있어야 합니다.

증명서 유형이 승인자 및 모두인 경우, 로그인 모듈에서는 서비스 프린시펄을 가정합니다.

구성 및 정책 파일

JGSS와 JAAS는 여러 구성 및 정책 파일에 의존합니다. 사용자의 환경과 어플리케이션에 맞도록 이들 파일을 편집해야 합니다. JGSS와 함께 JAAS를 사용하지 않으려는 경우, JAAS 구성 및 정책 파일을 무시해도 됩니다.

- 『Kerberos 구성 파일』
- 387 페이지의 『JAAS 구성 파일』
- 387 페이지의 『JAAS 정책 파일』
- 387 페이지의 『Java 마스터 보안 등록 정보 파일』
- 388 페이지의 『증명서 캐시 및 서버 키 표』

주: 다음 지침에서, ${java.home}은 서버에서 사용 중인 Java 버전의 위치 경로를 나타냅니다. 예를 들어 J2SDK, 버전 1.4를 사용 중인 경우, ${java.home}은 /QIBM/ProdData/Java400/jdk14입니다. 등록 정보 설정의 ${java.home}을 Java 홈 디렉토리의 실제 경로로 대체하십시오.

Kerberos 구성 파일

IBM JGSS에서는 Kerberos 구성 파일이 필요합니다. Kerberos 구성 파일의 디폴트 이름과 위치는 사용하는 오퍼레이팅 시스템에 따라 다릅니다. JGSS에서는 다음 순서로 디폴트 구성 파일을 탐색합니다.

1. Java 등록 정보 java.security.krb5.conf가 참조하는 파일
2. ${java.home}/lib/security/krb5.conf

3. Microsoft Windows(R) 플랫폼의 c:\winnt\krb5.ini
4. Solaris(TM) 플랫폼의 /etc/krb5/krb5.conf
5. 기타 Unix(R) 플랫폼의 /etc/krb5.conf

JAAS 구성 파일

JAAS 로그인 피처를 사용하려면 JAAS 구성 파일이 필요합니다. 다음 등록 정보 중 하나를 설정하여 JAAS 구성 파일을 지정할 수 있습니다.

- Java 시스템 등록 정보 java.security.auth.login.config
- ${java.home}/lib/security/java.security 파일의 보안 등록 정보 login.config.url.<integer>

자세한 정보는 Sun Java Authentication and Authorization Service(JAAS) 웹 사이트를 참조하십시오.

JAAS 정책 파일

디폴트 정책 구현을 사용하는 경우, JGSS는 정책 파일에 권한을 기록함으로써 엔티티에 JAAS 권한을 부여합니다. 다음 등록 정보 중 하나를 설정하여 JAAS 정책 파일을 지정할 수 있습니다.

- Java 시스템 등록 정보 java.security.policy
- ${java.home}/lib/security/java.security 파일의 보안 등록 정보 policy.url.<integer>

J2SDK, 버전 1.4를 사용 중인 경우, JGSS용의 별도 정책 파일을 지정하는 것은 선택적입니다. J2SDK, 버전 1.4의 디폴트 정책 제공자는 JAAS에 필요한 정책 파일 항목을 지원합니다.

자세한 정보는 Sun Java Authentication and Authorization Service(JAAS) 웹 사이트를 참조하십시오.

Java 마스터 보안 등록 정보 파일

JVM(Java Virtual Machine)은 Java 마스터 보안 등록 정보 파일을 편집하여 설정한 많은 중요 보안 등록 정보를 사용합니다. 이 java.security 파일은 주로 iSeries 서버의 ${java.home}/lib/security 디렉토리에 있습니다.

다음 리스트에서는 JGSS를 사용하기 위한 여러 관련 보안 등록 정보에 대해 설명합니다. java.security 파일을 편집하기 위한 안내서로서 설명을 사용하십시오.

주: 적용 가능한 경우, 설명에는 JGSS 샘플을 실행하는 데 필요한 해당 값들이 들어 있습니다.

security.provider.<integer>: 사용하려는 JGSS 제공자. 또한 암호 제공자 클래스를 고정으로 등록합니다. IBM JGSS는 IBM JCE Provider가 제공하는 암호 및 기타 보안 서비스를 사용합니다. 다음 예와 똑같이 sun.security.provider.Sun 및 com.ibm.crypto.provider.IBMJCE 패키지를 지정하십시오.

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

policy.provider: 시스템 정책 핸들러 클래스. 예를 들면 다음과 같습니다.

```
policy.provider=sun.security.provider.PolicyFile
```

policy.url.<integer>: 정책 파일의 URL. 샘플 정책 파일을 사용하려면, 다음 항목을 포함시키십시오.

```
policy.url.1=file:/home/user/jgss/config/java.policy
```

login.configuration.provider: JAAS 로그인 구성 핸들러 클래스, 예를 들면,

```
login.configuration.provider=com.ibm.security.auth.login.ConfigFile
```

auth.policy.provider: JAAS 프린시펄 기준 액세스 제어 정책 핸들러 클래스, 예를 들면

```
auth.policy.provider=com.ibm.security.auth.PolicyFile
```

login.config.url.<integer>: JAAS 로그인 구성 파일의 URL. 샘플 구성 파일을 사용하려면, 다음 항목을 포함시키십시오.

```
login.config.url.1=file:/home/user/jgss/config/jaas.conf
```

auth.policy.url.<integer>: JAAS 정책 파일의 URL. JAAS 정책 파일에 프린시펄 기반 및 CodeSource 기반 구조 모두를 포함시킬 수 있습니다. 샘플 정책 파일을 사용하려면, 다음과 같은 항목을 포함시키십시오.

```
auth.policy.url.1=file:/home/user/jgss/config/jaas.policy
```

증명서 캐시 및 서버 키 표

사용자 프린시펄은 증명서 캐시에 자신의 Kerberos 증명서를 보유합니다. 서비스 프린시펄은 키 표에 자신의 보안 키를 보유합니다. 런타임시, IBM JGSS는 이들 캐시를 다음 방식으로 찾습니다.

사용자 증명서 캐시

JGSS는 다음 순서를 사용하여 사용자 증명서 캐시를 찾습니다.

1. Java(TM) 등록 정보 KRB5CCNAME이 참조하는 파일
2. 환경 변수 KRB5CCNAME이 참조하는 파일
3. Unix 시스템의 /tmp/krb5cc_<uid>
4. ${user.home}/krb5cc_${user.name}
5. ${user.home}/krb5cc (${user.name}을 확보할 수 없는 경우)

서버 키 표

JGSS는 다음 순서를 사용하여 서버 키 표 파일을 찾습니다.

1. Java(TM) 등록 정보 KRB5_KTNAME 값
2. Kerberos 구성 파일의 libdefaults 스탠자에 있는 default_keytab_name 항목
3. ${user.home}/krb5_keytab

IBM JGSS 어플리케이션 개발

IBM JGSS 어플리케이션 개발에 대해서는 다음 주제를 참조하십시오.

프로그램 단계
전송 토큰 사용, 필요한 JGSS 오브젝트 작성, 문맥 설정 및 삭제, 메세지별 서비스 사용을 포함하여 JGSS 어플리케이션을 개발하는 데 필요한 단계를 제공합니다.

JGSS 어플리케이션으로 JAAS 사용
JGSS의 JAAS Kerberos 로그인 사용 가능 설정에 대한 내용입니다. 정보에는 로그인 피처 사용을 위한 요구사항과 일부 예제 코드가 들어 있습니다.

디버깅
JGSS 디버깅을 사용하여 유용한 디버깅 메세지를 범주화하고 표시하기 위한 내용이 들어 있습니다.

JGSS javadoc 참조 정보
org.ietf.jgss api 패키지에 있는 클래스와 메소드, Java 버전의 Kerberos 증명서 관리 툴(kinit, ktab 및 klist)에 대한 javadoc 정보를 검토합니다.

JGSS 샘플
샘플 프로그램을 사용하여 사용자의 어플리케이션에서 JGSS를 사용하는 방법을 찾습니다. 샘플 문서에는 Java 소스 코드, 샘플, 구성 및 정책 파일을 실행하기 위한 지침이 들어 있습니다.

JGSS 어플리케이션을 개발하려면, 고급 GSS-API 스펙과 Java 바인딩 스펙에 익숙해야 합니다. IBM JGSS 1.0은 기본적으로 이 스펙을 기반으로 하며 이 스펙을 준수합니다. 자세한 정보는 다음 링크를 참조하십시오.

- RFC 2743: Generic Security Service Application Programming Interface Version 2, Update 1
- RFC 2853: Generic Security Service API Version 2: Java Bindings

«

»

IBM JGSS 어플리케이션 프로그래밍 단계

JGSS 어플리케이션의 연산은 GSS-API(Generic Security Service Application Programming Interface) 연산 모델을 따릅니다. JGSS 연산의 중요 개념에 대해서는 JGSS 개념을 참조하십시오.

JGSS 전송 토큰

중요 JGSS 연산의 일부에서는 Java 바이트 배열 양식으로 토큰을 생성합니다. 어플리케이션은 한 JGSS 피어에서 다른 피어로 토큰을 전송해야 합니다. JGSS에는 어플리케이션이 토큰 전송을 위해 사용하는 프로토콜을

어떤 방식으로든 제한하지 않습니다. 어플리케이션은 다른 어플리케이션(즉, 비JGSS) 자료와 함께 JGSS 토큰을 전송할 수 있습니다. 그러나 JGSS 연산에서는 JGSS 고유의 토큰만을 승인하고 사용합니다.

JGSS 어플리케이션 내에서의 연산 순서

JGSS 연산에서는 아래 나열된 순서로 사용해야 하는 일정 프로그래밍 구조가 필요합니다. 각 단계는 개시자와 승인자 모두에 적용됩니다.

주: 정보에는 고급 JGSS API 사용에 대해 설명하고 사용자 어플리케이션이 org.ietf.jgss 패키지를 가져오는 것으로 가정하는 일부 예제 코드가 들어 있습니다. 여러 고급 API가 과부하되어도, 일부는 가장 일반적으로 사용되는 메소드 양식만을 보여줍니다. 물론 사용자의 요구에 가장 적합한 API 메소드를 사용하십시오.

1. GSSManager 작성
   GSSManager 인스턴스는 다른 JGSS 오브젝트 인스턴스를 작성하기 위한 팩토리로 동작합니다.
2. GSSName 작성
   GSSName은 JGSS 프린시펄의 ID를 나타냅니다. 몇몇 JGSS 연산에서는 널인 GSSName을 지정할 때의 디폴트 프린시펄을 찾아 사용할 수 있습니다.
3. GSSCredential 작성
   GSSCredential에는 프린시펄의 메카니즘 고유 증명서가 들어 있습니다.
4. GSSContext 작성
   GSSContext는 문맥 설정 및 후속 메세지 개개 서비스에 사용됩니다.
5. 문맥에서 선택적 서비스 선택
   어플리케이션은 상호 인증과 같은 선택적 서비스를 명시적으로 요청해야 합니다.
6. 문맥 설정
   개시자는 그 자체를 승인자로서 인증합니다. 그러나 상호 인증을 요청할 경우, 이번에는 승인자가 그 자체를 개시자에 대해 인증합니다.
7. 메세지별 서비스 사용
   개시자와 승인자는 설정된 구문을 통해 보안 메세지를 교환합니다.
8. 문맥 삭제
   어플리케이션은 더 이상 필요하지 않은 문맥을 삭제합니다.

GSSManager 작성

GSSManager 추상 클래스는 다음 JGSS 오브젝트를 작성하기 위한 팩토리로서 제공됩니다.

- GSSName
- GSSCredential
- GSSContext

GSSManager에는 지원되는 보안 메카니즘과 이름 유형을 결정하기 위한 메소드 및 JGSS 제공자를 지정하기 위한 메소드가 있습니다. GSSManager getInstance 정적 메소드를 사용하여 디폴트 GSSManager 인스턴스를 작성하십시오.

```
GSSManager manager = GSSManager.getInstance();
```

GSSName 작성

GSSName은 GSS-API 프린시펄의 ID를 나타냅니다. GSSName은 지원되는 각각의 기본 메카니즘에 대해, 프린시펄의 다양한 표현을 포함할 수 있습니다. 하나의 이름 표현만 들어있는 GSSName을 메카니즘 이름(MN)이라고 합니다.

GSSManager에는 스트링이나 연속적인 바이트 배열로 GSSName을 작성하기 위한 여러 과부하 메소드가 있습니다. 메소드는 지정된 이름 유형에 따라 스트링이나 바이트 배열을 해석합니다. 보통 GSSName 바이트-배열 메소드를 사용하여 내보낸 이름을 재구성합니다. 내보낸 이름은 일반적으로 GSSName.NT_EXPORT_NAME 유형의 메카니즘 이름입니다. 일부 메소드는 이름을 작성할 보안 메카니즘을 지정할 수 있습니다.

예: GSSName 사용

다음 기본 코드 일부는 GSSName 사용법을 보여줍니다.

주: Kerberos 서비스 이름 스트링을 <service> 또는 <service@host>로서 지정하십시오. 여기서 <service>는 서비스 이름이고, <host>는 서비스가 실행되는 기계의 호스트 이름입니다. 완전히 규정된 호스트 이름을 사용할 수 있습니다(꼭 사용해야 하는 것은 아닙니다). 스트링의 <host> 부분을 생략하면, GSSName이 로컬 호스트 이름을 사용합니다.

```
// 사용자 foo의 GSSName을 작성합니다.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);

// 사용자 foo의 Kerberos V5 메카니즘 이름을 작성합니다.
Oid krb5Mech = new Oid("1.2.840.113554.1.2.2");
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME, krb5Mech);

// GSSName 표준 메소드를 사용하여 비메카니즘적인 이름에서 메카니즘적인 이름을
// 작성합니다.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);
GSSName fooKrb5Name = fooName.canonicalize(krb5Mech);
```

GSSCredential 작성

GSSCredential에는 프린시펄에 대한 문맥 작성에 필요한 암호 정보가 들어있으며, 복수 메카니즘에 대한 증명서 정보가 들어있을 수 있습니다.

GSSManager에는 3개의 증명서 작성 메소드가 있습니다. 메소드 2개에서는 GSSName, 증명서 유효기간, 증명서를 얻은 하나 이상의 메카니즘 및 증명서 사용 유형을 매개변수로서 사용합니다. 다른 한가지 메소드에서는 사용 유형을, 다른 매개변수에는 디폴트 값을 사용합니다. 널(null) 메카니즘의 지정에서도 디폴트 메카니즘이 사용됩니다. 널(null)배열의 메카니즘을 지정하면 메소드가 디폴트 메카니즘 세트에 대한 증명서를 리턴합니다.

주: IBM JGSS는 Kerberos V5 메카니즘만을 지원하므로, 이것이 디폴트 메카니즘입니다.

어플리케이션은 한 번에 3개 증명서 유형(*시작*, *승인* 또는 *시작 및 승인*) 중 하나만을 작성할 수 있습니다.

- 문맥 개시자가 *시작* 증명서를 작성합니다.
- 승인자가 *승인* 증명서를 작성합니다.
- 개시자로서 작동한 승인자가 *시작 및 승인* 증명서를 작성합니다.

예: 증명서 확보

다음 예에서는 개시자를 위한 디폴트 증명서를 확보합니다.

```
GSSCredentials fooCreds = manager.createCredentials(GSSCredential.INITIATE)
```

다음 예에서는 디폴트 유효 기간이 있는 개시자 foo에 대해 Kerberos V5 증명서를 확보합니다.

```
GSSCredential fooCreds = manager.createCredential(fooName, GSSCredential.DEFAULT_LIFETIME,
                                                  krb5Mech,GSSCredential.INITIATE);
```

다음 예에서는 모든 디폴트 승인자 증명서를 확보합니다.

```
GSSCredential serverCreds = manager.createCredential(null, GSSCredential.DEFAULT_LIFETIME,
                                                     (Oid)null, GSSCredential.ACCEPT);
```

»

GSSContext 작성

IBM JGSS는 문맥을 작성하기 위해 GSSManager가 지원하는 두가지 메소드를 지원합니다.

- 문맥 개시자가 사용하는 메소드
- 승인자가 사용하는 메소드

주: GSSManager에서는 이전에 내보낸 문맥 재작성이 포함된 문맥 작성을 위한 제3의 메소드가 제공됩니다. 그러나, IBM JGSS Kerberos V5 메카니즘은 내보낸 문맥 사용을 지원하지 않으며, IBM JGSS는 이 메소드를 지원하지 않습니다.

어플리케이션에서는 문맥 승인을 위해 개시자 문맥을 사용할 수 없으며, 문맥 초기화를 위해 승인자 문맥을 사용할 수도 없습니다. 문맥 작성을 위해 지원되는 메소드 모두에서는 증명서를 입력해야 합니다. 증명서 값이 널(null)이면, JGSS는 디폴트 증명서를 사용합니다.

예: GSSContext 사용

다음 예에서는 프린시펄(foo)이 호스트(securityCentral)에서 피어(superSecureServer)와의 문맥을 시작할 수 있는 문맥을 작성합니다. 예에서는 피어를 superSecureServer@securityCentral로서 지정합니다. 작성된 문맥은 디폴트 기간동안 유효합니다.

```
GSSName serverName = manager.createName("superSecureServer@securityCentral",
                                        GSSName.NT_HOSTBASED_SERVICE, krb5Mech);
GSSContext fooContext = manager.createContext(serverName, krb5Mech, fooCreds,
                                              GSSCredential.DEFAULT_LIFETIME);
```

다음 예에서는 임의의 피어가 시작한 문맥을 승인하기 위한 superSecureServer 문맥을 작성합니다.

```
GSSContext serverAcceptorContext = manager.createContext(serverCreds);
```

어플리케이션이 두가지 유형의 문맥을 작성하고 동시에 사용할 수 있음을 기억하십시오.

선택적 보안 서비스 요구

어플리케이션은 임의의 선택적 보안 서비스를 요구할 수 있습니다. IBM JGSS는 다음 선택적 서비스를 지원합니다.

- 위임
- 상호 인증
- 재생 감지
- 무작위 감지
- 사용 가능한 메세지 개개 기밀성
- 사용 가능한 메세지 개개 무결성

선택적 서비스를 요구하려면, 어플리케이션이 문맥에서 해당 요구 메소드를 사용함으로써 이를 명시적으로 요청해야 합니다. 개시자만이 이들 선택적 서비스를 요구할 수 있습니다. 개시자는 문맥 설정을 시작하기 전에 요구해야 합니다.

선택적 서비스에 대해서는, IETF(Internet Engineering Task Force) RFC 2743 내의 선택적 지원 Generic Security Services Application Programming Interface Version 2, Update 1  을 참조하십시오.

예: 선택적 서비스 요구

다음 예에서, 문맥(fooContext)은 상호 인증 및 위임 서비스가 가능하도록 요청합니다.

```
fooContext.requestMutualAuth(true);
fooContext.requestCredDeleg(true);
```

문맥 설정

두 개의 통신 피어는 메세지 개개 서비스를 사용할 수 있는 보안 문맥을 설정해야 합니다. 개시자는 문맥에서 initSecContext()를 호출하여, 개시자 어플리케이션에 토큰을 리턴합니다. 개시자 어플리케이션은 문맥 토큰을 승인자 어플리케이션으로 전송합니다. 승인자는 문맥에서 acceptSecContext()를 호출하여, 개시자로부터 받은 문맥 토큰을 지정합니다. 기초 메카니즘과 개시자가 선택한 선택적 서비스에 따라, acceptSecContext()는 승인자 어플리케이션이 개시자 어플리케이션으로 전송해야 하는 토큰을 생성합니다. 개시자 어플리케이션은 수신된 토큰을 사용하여 initSecContext()를 한번 더 호출합니다.

어플리케이션은 GSSContext.initSecContext()와 GSSContext.acceptSecContext()를 여러 번 호출할 수 있습니다. 어플리케이션은 또한 문맥 설정 도중 피어와 여러 토큰을 교환할 수도 있습니다. 따라서, 일반적인 문맥 설정 메소드에서는 어플리케이션이 문맥을 설정할 때까지 루프를 사용하여 GSSContext.initSecContext() 또는 GSSContext.acceptSecContext()를 호출합니다.

예: 문맥 설정

다음 예에서는 개시자(foo) 측에서의 문맥 설정에 대해 설명합니다.

```
byte array[] inToken = null; // 입력 토큰이 첫 번째 호출의 경우 널(null)입니다.
int inTokenLen = 0;

do {
     byte[] outToken = fooContext.initSecContext(inToken, 0, inTokenLen);

     if (outToken != null) {
         send(outToken); // 승인자에게 토큰을 전송합니다.
     }

     if( !fooContext.isEstablished()) {
         inToken = receive(); // 승인자에게서 토큰을 받습니다.
           inTokenLen = inToken.length;
     }
 } while (!fooContext.isEstablished());
```

다음 예에서는 승인자 측에서의 문맥 설정에 대해 설명합니다.

```
// 문맥 설정을 위한 승인자 코드는 다음과 같습니다.
do {
     byte[] inToken = receive(); // 개시자에게서 토큰을 받습니다.
     byte[] outToken =
         serverAcceptorContext.acceptSecContext(inToken, 0, inToken.length);

     if (outToken != null) {
         send(outToken); // 개시자에게 토큰을 전송합니다.
     }
 } while (!serverAcceptorContext.isEstablished());
```

메세지 개개 서비스 사용

보안 문맥을 설정한 뒤, 두 개의 통신 피어는 설정된 문맥에서 보안 메세지를 교환할 수 있습니다. 각 피어는 문맥을 설정할 때 개시자로 설정되었는 지 승인자로 설정되었는 지에 관계없이 보안 메세지를 시작할 수 있습니다. 메세지를 안전하게 하기 위해, IBM JGSS는 메세지에서 암호 메세지 무결성 코드(MIC)를 계산합니다. 선택적으로, IBM JGSS는 개인 보호 정책의 보장에 도움을 주기 위해 Kerberos V5 메카니즘이 메세지를 암호화하도록 할 수 있습니다.

메세지 송신

IBM JGSS는 메세지 보안을 위한 2개의 메소드 세트인 wrap()과 getMIC()를 제공합니다.

wrap() 사용

랩 메소드는 다음 조치를 수행합니다.

- MIC를 계산합니다.
- 메세지를 암호화합니다(선택적).
- 토큰을 리턴합니다.

호출 어플리케이션은 GSSContext와 함께 MessageProp 클래스를 사용하여 메세지를 암호화할 것인 지 여부를 지정합니다.

리턴된 토큰에는 MIC와 메세지 텍스트 모두가 들어 있습니다. 메세지 텍스트는(암호화된 메세지의 경우) 암호 텍스트나(암호화되지 않은 메세지의 경우) 보통 텍스트입니다.

getMIC() 사용

getMIC 메소드는 다음 조치를 수행하지만 메세지를 암호화할 수 없습니다.

- MIC를 계산합니다.
- 토큰을 리턴합니다.

리턴된 토큰에는 계산된 MIC는 들어있지만, 원래 메세지는 없습니다. 그러므로, MIC 토큰을 피어에 전송할 뿐 아니라, 피어는 MIC를 검증할 수 있도록 원래 메세지를 어느 정도 인식하고 있어야 합니다.

예: 메세지 개개 서비스를 사용한 메세지 송신

다음 예에서는 하나의 피어(foo)가 다른 피어(superSecureServer)로 전달하기 위해 메세지를 랩핑하는 방법을 보여줍니다.

```
byte[] message = "Ready to roll!".getBytes();
MessageProp mprop = new MessageProp(true); // foo는 메세지가 암호화되길 원합니다.
byte[] wrappedMessage =
    fooContext.wrap(message, 0, message.length, mprop);
send(wrappedMessage); // 랩된 메세지를 superSecureServer로 전송합니다.

// superSecureServer가 foo로 전달하기 위한 MIC를 확보할 수 있는 방법입니다.
byte[] message = "You bet!".getBytes();
MessageProp mprop = null;  // superSecureServer는 디폴트 품질이
```

```
                                    // 보호인 내용입니다.

    byte[] mic =
        serverAcceptorContext.getMIC(message, 0, message.length, mprop);
    send(mic);
    // MIC를 foo로 전송합니다. foo는 MIC 확인을 위한 원래 메세지가 필요합니다.
```

메세지 수신

랩된 메세지 리시버는 unwrap()을 사용하여 메세지를 해독합니다. unwrap 메소드는 다음 조치를 수행합니다.

- 암호 MIC가 메세지에 들어 있는 지 확인합니다.
- 송신자가 MIC를 계산했던 원래 메세지를 리턴합니다.

송신자가 메세지를 암호화한 경우, unwrap()은 MIC를 확인하기 전에 메세지를 해독하고 원래 단순 메세지를 리턴합니다. MIC 토큰의 리시버는 verifyMIC()를 사용하여 제공된 메세지를 통해 MIC를 확인합니다.

피어 어플리케이션은 자신의 프로토콜을 사용하여 JGSS 문맥과 메세지 토큰을 서로에게 전달합니다. 피어 어플리케이션은 또한 토큰이 MIC인 지 또는 랩된 메세지인 지를 결정하는 프로토콜을 정의해야 합니다. 예를 들어, 해당 프로토콜의 일부는 SASL(Simple Authentication and Security Layer) 어플리케이션에서 사용하는 프로토콜과 같이 간단(및 확실)할 수 있습니다. SASL 프로토콜은 문맥 승인자가 문맥 설정 다음에 메세지 개개 (랩된) 토큰을 전달하는 첫 번째 피어가 되도록 지정합니다.

자세한 정보는 SASL(Simple Authentication and Security Layer)  을 참조하십시오.

예: 메세지 개개 서비스를 사용한 메세지 수신

다음 예에서는 피어(superSecureServer)가 다른 피어(foo)로 부터 받은 랩된 토큰을 랩제거하는 방법을 보여줍니다.

```
MessageProp mprop = new MessageProp(false);

byte[] plaintextFromFoo =
    serverAcceptorContext.unwrap(wrappedTokenFromFoo, 0,
                                 wrappedTokenFromFoo.length, mprop);

// superSecureServer는 mprop를 점검하여 foo가 적용한 메세지 등록 정보
// (메세지가 암호화되었는 지 여부와 같은)를 판별합니다.

// foo가 superSecureServer로부터 받은 MIC를 확인합니다.

MessageProp mprop = new MessageProp(false);
fooContext.verifyMIC(micFromFoo, 0, micFromFoo.length, messageFromFoo, 0,
                     messageFromFoo.length, mprop);

// foo는 이제 mprop를 점검하여
// superSecureServer가 적용한 메세지 등록 정보를 판별합니다. 특히, getMIC는 메세지를
// 암호화하지 않으므로 메세지가 암호화되지 않도록 하십시오.
```

문맥 삭제

피어는 문맥이 더 이상 필요하지 않으면 문맥을 삭제합니다. JGSS 연산에서 각각의 피어는 단독으로 문맥 삭제 시점을 결정하며 해당 피어에게 알릴 필요가 없습니다.

JGSS는 문맥 삭제 토큰을 정의하지 않습니다. 문맥을 삭제하기 위해, 피어는 GSSContext 오브젝트의 배치(dispose) 메소드를 호출하여 문맥에서 사용하던 자원들을 해제시킵니다. 배치된 GSSContext 오브젝트는 어플리케이션이 오브젝트를 널(null)로 설정하지 않는 한 계속 액세스할 수 있습니다. 그러나, 배치된(그러나 계속 액세스할 수 있는) 문맥을 사용하려 하면 예외가 발생합니다.

JGSS 어플리케이션과 함께 JAAS 사용

IBM JGSS에는 어플리케이션이 JAAS를 사용하여 증명서를 확보할 수 있도록 하는 선택적 JAAS 로그인 기능이 있습니다. JAAS 로그인 기능이 프린시펄 증명서와 비밀키를 JAAS 로그인 문맥의 주제 오브젝트에 저장한 다음, JGSS는 그 주제에서 증명서를 검색할 수 있습니다.

JGSS의 디폴트 작동은 주제에서 증명서와 비밀키를 검색하는 것입니다. Java 등록 정보 javax.security.auth.useSubjectCredsOnly를 거짓으로 설정함으로써 이 피처를 작동 불가능하도록 설정할 수 있습니다.

주: 순수한 Java JGSS 제공자가 로그인 인터페이스를 사용할 수 있더라도, 원시 iSeries JGSS 제공자는 사용할 수 없습니다.

JAAS 피처에 대해서는 Kerberos 증명서 및 비밀키 확보를 참조하십시오.

JAAS 로그인 기능을 사용하려면, 어플리케이션이 다음 방식으로 JAAS 프로그래밍 모델을 따라야 합니다.

- JAAS 로그인 문맥 작성
- JAAS JAAS Subject.doAs 구성의 경계내에서 작동

다음 코드 일부는 JAAS Subject.doAs 구성의 경계내에서 작동 개념에 대해 설명합니다.

```
static class JGSSOperations implements PrivilegedExceptionAction {
    public JGSSOperations() {}
    public Object run () throws GSSException {
        // JGSS 어플리케이션 코드가 여기서 실행됩니다.
    }
}

public static void main(String args[]) throws Exception {
    // Kerberos 콜백 핸들러인
    // com.ibm.security.auth.callback.Krb5CallbackHandler를
    // 사용할 로그인 문맥을 작성합니다.

    // "JGSSClient"에 대한 JAAS 구성이어야 합니다.
    LoginContext loginContext =
```

```
            new LoginContext("JGSSClient", new Krb5CallabackHandler());
            loginContext.login();

        // JAAS 권한 모드에서 JGSS 어플리케이션 전체를 실행합니다.
        Subject.doAsPrivileged(loginContext.getSubject(),
                               new JGSSOperations(), null);
}
```

디버깅

JGSS 문제점을 식별하려 할 때, JGSS 디버깅 기능을 사용하여 유용한 범주화 메세지를 생성하십시오. Java 등록 정보 com.ibm.security.jgss.debug에 대해 적합한 값을 설정하여 하나 이상의 범주를 사용 가능하게 할 수 있습니다. 콤마를 사용하여 범주 이름을 구분함으로써 복수의 범주를 활성화할 수 있습니다.

범주 디버깅에는 다음이 포함됩니다.

범주	설명
도움말	디버그 범주를 나열
모두	모든 범주에 대한 디버깅을 온으로 설정
Off	디버깅을 완전히 오프
app	어플리케이션 디버깅(디폴트)
ctx	문맥 연산 디버깅
cred	증명서(이름 포함) 연산
marsh	토큰 정렬
mic	MIC 연산
prov	제공자 연산
qop	QOP 연산
unmarsh	토큰 정렬해제
unwrap	랩제거 연산
wrap	랩 연산

JGSS 디버그 클래스

JGSS 어플리케이션을 프로그램에서 디버그하려면 IBM JGSS 구조의 디버그 클래스를 사용하십시오. 어플리케이션에서는 디버그 클래스를 사용하여 디버그 범주를 온 및 오프로 설정하고, 활성화된 범주의 디버그 정보를 표시할 수 있습니다.

디폴트 디버깅 구성자는 Java 등록 정보 com.ibm.security.jgss.debug를 읽어서 활성화(온 상태로 전환)할 범주를 판별합니다.

예: 어플리케이션 범주 디버깅

다음 예에서는 어플리케이션 범주의 디버그 정보를 요청하는 방법을 보여줍니다.

```
import com.ibm.security.jgss.debug;

Debug debug = new Debug(); // Java 등록 정보에서 범주를 결정합니다.

// 많은 작업에서 someBuffer를 설정해야 합니다. 범주가
// 디버깅이 설정되기 전에 온으로 되었는 지 확인하십시오.

if (debug.on(Debug.OPTS_CAT_APPLICATION)) {
    // 자료로 someBuffer를 채웁니다.
 debug.out(Debug.OPTS_CAT_APPLICATION, someBuffer);
    // someBuffer는 바이트 배열이나 스트링입니다.
```

샘플: IBM JGSS(Java Generic Security Service)

IBM JGSS 샘플 파일에는 클라이언트 및 서버 프로그램, 구성 파일, 정책 파일과 javadoc 참조 정보가 포함됩니다.

HTML 버전의 샘플을 보거나, 샘플 프로그램의 javadoc 정보와 소스 코드를 다운로드할 수 있습니다. 샘플을 다운로드하면 javadoc 참조 정보를 보고, 코드를 점검하며, 구성 및 정책 파일을 편집하고, 샘플 프로그램을 컴파일 및 실행할 수 있습니다.

- HTML 버전의 샘플 보기
- 샘플 javadoc 정보 다운로드 및 보기
- 샘플 프로그램 다운로드 및 실행

샘플 프로그램 설명

JGSS 샘플에는 4개의 프로그램이 포함됩니다.

- 비JAAS 서버
- 비JAAS 클라이언트
- JAAS 작동 가능 서버
- JAAS 작동 가능 클라이언트

JAAS 작동 가능 버전은 해당 비JAAS 상대방과의 완벽한 상호운용이 가능합니다. 그러므로 비JAAS 서버에 대해 JAAS 작동 가능 클라이언트를 실행할 수 있고, JAAS 작동 가능 서버에 대해 비JAAS 클라이언트를 실행할 수 있습니다.

주: 샘플을 실행할 때, 구성 및 정책 파일의 이름, JGSS 디버그 옵션 및 보안 관리자를 포함하여 하나 이상의 선택적 Java 등록 정보를 지정할 수 있습니다. 또한 JAAS 피처를 온이나 오프로 설정할 수도 있습니다.

1 서버 또는 2 서버 구성으로 샘플을 실행할 수 있습니다. 1 서버 구성은 하나의 1차 서버와 통신하는 하나의 클라이언트로 구성됩니다. 2 서버 구성은 1차와 2차 서버로 구성되며, 여기서 1차 서버는 2차 서버에 대해 개시자 또는 클라이언트로서 동작합니다.

2 서버 구성 사용 시, 클라이언트는 먼저 문맥을 시작하고 1차 서버와 보안 메세지를 교환합니다. 그런 다음 클라이언트는 자신의 증명서를 1차 서버에게 위임합니다. 그러면 1차 서버는 클라이언트를 위해 이들 증명서를 사용하여 문맥을 시작하고 2차 서버와 보안 메세지를 교환합니다. 또한 1차 서버가 자신을 위해 클라이언트로서 동작하는 2서버 구성을 사용할 수도 있습니다. 이 경우 1차 서버는 자신의 증명서를 사용하여 문맥을 시작하고 2차 서버와 메세지를 교환합니다.

1차 서버에 대해 동시에 여러 클라이언트를 실행할 수도 있습니다. 2차 서버에 대해 직접 클라이언트를 실행할 수 있는 경우라도, 2차 서버는 위임된 증명서를 사용할 수 없거나 자신의 증명서를 사용하여 개시자로서 실행될 수 없습니다.

IBM JGSS 샘플 보기

IBM Java 일반 보안 서비스(JGSS) 샘플 파일에는 클라이언트 및 서버 프로그램, 구성 파일, 정책 파일과 javadoc 참조 정보가 포함됩니다. 다음 링크를 사용하여 JGSS 샘플의 HTML 버전을 보십시오.

자세한 정보는 다음 주제를 참조하십시오.

- 399 페이지의 『샘플 프로그램 설명』
- 샘플 프로그램 다운로드 및 실행

샘플 프로그램 보기

다음 링크를 사용하여 HTML 버전의 JGSS 샘플 프로그램을 보십시오.

- 샘플 비JAAS 클라이언트 프로그램
- 샘플 비JAAS 서버 프로그램
- 샘플 JAAS 작동 가능 클라이언트 프로그램
- 샘플 JAAS 작동 가능 서버 프로그램

샘플 구성 및 정책 파일 보기

다음 링크를 사용하여 HTML 버전의 JGSS 구성 및 정책 파일을 보십시오.

- Kerberos 구성 파일
- JAAS 구성 파일
- JAAS 정책 파일
- Java 정책 파일

샘플: IBM JGSS 비JAAS 클라이언트 프로그램

샘플 클라이언트 프로그램의 사용에 대해서는 샘플 프로그램 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// IBM JGSS 1.0 샘플 클라이언트 프로그램

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.*;

/**
 * JGSS 샘플 클라이언트:
 * JGSS 샘플 서버와 함께 사용됩니다.
 * 클라이언트는 먼저 서버와의 문맥을 설정하고, MIC 다음에 랩된 메세지를
 * 서버로 전송합니다.
 * MIC는 랩된 보통 텍스트를 통해 계산됩니다.
 * 클라이언트는 서버가 문맥 설정 도중 그 자체를 인증해야 합니다(상호 인증).
 * 또한 증명서를 서버에 위임합니다.
 *
 * JGSS가 JAAS를 통해 증명서를 확보하지 않으므로 JAVA 변수
 * javax.security.auth.useSubjectCredsOnly를 거짓으로
 * 설정합니다.
 *
 * 서버는 입력 매개변수를 사용하고, jgss.ini 파일의 정보로 이를 보충합니다.
 * 명령행에서 지원되지 않는 필수 입력은 jgss.ini 파일에서 얻습니다.
 *
 * 사용법: 클라이언트 [옵션]
 *
 * -? 옵션은 지원되는 옵션을 포함한 도움말 메세지를 생성합니다.
 *
 * 이 샘플 클라이언트는 JAAS를 사용하지 않습니다.
 * 클라이언트는 JAAS 샘플 클라이언트 및 서버에 대해 실행될 수 있습니다.
 * JAAS를 사용하는 샘플 클라이언트에 대해서는 {@link JAASClient JAASClient}를 참조하십시오.
 */

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; 이 메세지를 생성시킵니다. "
    + "\n  -n name\t\t클라이언트의 프린시펄 이름(영역 없음)"
    + "\n  -s serverName\t\t서버의 프린시펄 이름(영역 없음)"
    + "\n  -h serverHost[:port]\t서버의 호스트 이름"
    +         " (및 선택적 포트 번호)"
   + "\n  -p port\t\t서버가 청취할 포트"
    + "\n  -m msg\t\t서버로 전송할 메세지";


// 호출자는 초기화를 호출해야 합니다 (먼저 processArgs를 호출해야 할 수도 있습니다).
public Client (String programName) throws Exception
{
    testUtil = new Util();
    if (programName != null)
    {
        program = programName;
        debugPrefix = programName + ": ";
    }
}

// 호출자는 초기화를 호출해야 합니다(먼저 processArgs를 호출해야 할 수도 있습니다).
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);

        // 등록 정보가 설정되지 않았습니다. 지정된 값으로 설정하십시오.

        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
{
        // 피어 이름
    if (serverNameWithoutRealm != null)
    {
        this.serverName = serverNameWithoutRealm;
    }
 else
    {
        this.serverName = testUtil.getDefaultServicePrincipalWithoutRealm();
    }

        // 피어 호스트
    if (serverHostname != null)
    {
```

```
        this.serviceHostname = serverHostname;
    }
 else
    {
        this.serviceHostname = testUtil.getDefaultServiceHostname();
    }

        // 피어 포트
    if (serverPort > 0)
    {
        this.servicePort = serverPort;
    }
 else
    {
        this.servicePort = testUtil.getDefaultServicePort();
    }

        // 피어 메세지
    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");
            }
            // 나중에 더 추가
        }

        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); // Subject 증명서를 사용하지 마십시오.
        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");
    }
}
```

샘플: IBM JGSS 비JAAS 서버 프로그램

샘플 서버 프로그램 사용에 대해서는 IBM JGSS 샘플 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// 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.*;

/**
 * JGSS 샘플 서버; JGSS 샘플 클라이언트와 결합하여 사용됩니다.
 *
 * 계속해서 클라이언트 연결을 청취하며,
 * 스레드를 파생시켜 수신 연결에 대한 서비스를 제공합니다.
 * 여러 스레드를 동시에 실행할 수 있습니다.
 * 즉, 여러 클라이언트를 동시에 서비스할 수 있습니다.
 *
 * 각 스레드는 먼저 클라이언트와 문맥을 설정한 다음
 * MIC 다음에 오는 랩된 메세지를 기다립니다.
 * 클라이언트가 MIC * 클라이언트가 랩한 단순 텍스트를
 * 계산하는 것으로 가정합니다.
 *
 * 클라이언트가 서버에 증명서를 위임하면, 위임된 증명서는
 * 2차 서버와의 통신에 사용됩니다.
 *
 * 또한, 서버는 서버 뿐만 아니라 클라이언트로서 동작하기 위해
 * 시작될 수 있습니다(-b 옵션을 사용하여). 이런 경우,
 * 서버가 파생시킨 첫 번째 스레드에서는 서버 프린시펄 자신의 증명서를 사용하여
 * 2차 서버와 통신합니다.
 *
 * 2차 서버는 (2차 서버)와의 연락을 시작하는 (1차) 서버보다 먼저 시작되어야 합니다.
 * 2차 서버와의 통신에서, 1차 서버는 JGSS 개시자 (즉, 클라이언트)로서 동작하며, 문맥을
 * 설정하고 2차 서버와의 랩 및 MIC 메세지 개개 교환에 참여합니다.
 *
```

```
 * 서버는 입력 매개변수를 사용하고, jgss.ini 파일의 정보로 이를 보충합니다.
 * 명령행에서 지원되지 않는 필수 입력은 jgss.ini 파일에서 얻습니다.
 * jgss.ini 파일이 없거나 특별한 변수가 ini 파일에서 지정되지 않으면 내장 디폴트가 사용됩니다.
 *
 * 사용법: 서버 [옵션]
 *
 * -? 옵션은 지원되는 옵션을 포함한 도움말 메세지를 생성합니다.
 *
 * 이 샘플 서버는 JAAS를 사용하지 않습니다.
 * JGSS가 JAAS를 통해 증명서를 확보하지 않으므로 JAVA 변수
 * javax.security.auth.useSubjectCredsOnly를 거짓으로
 * 설정합니다.
 * 서버는 JAAS 샘플 클라이언트 및 서버에 대해 실행될 수 있습니다.
 * JAAS를 사용하는 샘플 서버에 대해서는 {@link JAASServer JAASServer}를 참조하십시오.
 */

class Server implements Runnable
{
    /*
     * 주:
     * 이 Server 클래스는 복수의 스레드에서 동시에 실행될 것으로 예상됩니다.
     * 정적 변수는 명령행 초기화 도중 설정된 명령행 인수와 변수
     * (서버 자신의 증명서 gssCred와 같은)의 변수 세트로 구성됩니다.
     * 이들 변수들은 한번 설정되면 변경되지 않으며, 실행되는 모든 스레드 사이에서
     * 공유됩니다.
     *
     * 초기에 설정된 후 유일하게 변경되는 정적 변수는
     * 변수 'beenInitiator'로서, 이것은 서버 자신의 증명서를 사용하여
     * 서버를 개시자로서 실행하도록 하기 위해 첫 번째 스레드가 '참'으로 설정합니다.
     *
     * 변수 'tcp'는 비정적이며, 스레드마다 설정되어 연결된 스레드가 제공되는 클라이언트의
     * 소켓을 나타냅니다.
     */

    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; 이 메세지를 생성시킵니다. "
       + "\n  -# number\t\t1차 서버인 지 2차 서버인 지 여부"
       +         " \n\t\t\t(1 = 1차, 2 = 2차; default = 첫 번째)"
```

```
      + "\n  -n name\t\t서버의 프린시펄 이름(영역 없음)"
      + "\n  -p port\t\t서버가 청취할 포트"
      + "\n  -s serverName\t\t2차 서버의 프린시펄 이름"
      +         " (영역 없음)"
      + "\n  -h serverHost[:port]\t2차 서버의 호스트 이름"
      +         " (및 선택적 포트 번호)"
      + "\n  -P port\t\t2차 서버의 포트 번호"
      + "\n  -m msg\t\t2차 서버로 전송할 메세지"
      + "\n  -b \t\t서버 자신의 증명서를 사용하여"
      +         " 클라이언트와 서버 모두로서 실행됩니다.";
   // 비정적 변수는 각 스레드가 이 클래스의 별도 인스턴스를 실행시키므로
   // 스레드마다 다릅니다.
   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"));

           // 등록 정보가 설정되지 않았습니다. 지정된 값으로 설정하십시오.
```

```
        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;

    // 사용자 포트
    if (port > 0)
    {
        myPort = port;
    }
    else if (primary)
    {
        myPort = testUtil.getDefaultServicePort();
    }
 else
    {
        myPort = testUtil.getDefaultService2Port();
    }

    if (primary)
    {
        ///// 피어 이름
        if (serverNameWithoutRealm != null)
        {
            serviceNameNoRealm = serverNameWithoutRealm;
        }
 else
        {
            serviceNameNoRealm =
                  testUtil.getDefaultService2PrincipalWithoutRealm();
        }
```

```
        // 피어 호스트
        if (serverHostname != null)
        {
            if (serverHostname.equalsIgnoreCase("localHost"))
            {
                serverHostname = InetAddress.getLocalHost().getHostName();
            }

            serviceHost = serverHostname;
        }
    else
        {
            serviceHost = testUtil.getDefaultService2Hostname();
        }

        // 피어 포트
        if (serverPort > 0)
        {
            servicePort = serverPort;
        }
    else
        {
            servicePort = testUtil.getDefaultService2Port();
        }

        // 피어 메세지
        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;

        ////// 기타 서버

        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,  // 1차 또는 2차 서버
```

```
          name,     // 이름
          iport,    // 사용자 포트
          sname,    // 기타 서버 이름
          shost,    // 기타 서버의 호스트 이름
          isport,   // 기타 서버 포트
          smessage, // 기타 서버의 메세지
          clientServer); // 자신의 증명서로 개시자로서 실행되는 지 여부
}

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(); // 서버 소켓을 닫습니다.
            if (server != null)
                server.dispose(); // 클라이언트 소켓을 닫습니다.
        } 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);

      // 개시자로 문맥을 처음 설정합니다.
      if (!establishContext(context))
  return;

      // 그런 다음 개시자로부터의 메세지를 처리합니다.
      // MIC 다음에 랩된 메세지를 수신할 것으로 예상합니다.
      // MIC는 랩된 메세지를 수신했던 단순 텍스트를 대상으로 계산되어야 합니다.
      // 필요하면 위임된 증명서를 사용하십시오.
      // 그런 다음 필요하면 자신의 증명서를 사용하여 개시자로서 실행하십시오.
      // 첫 번째 스레드만이 이를 수행합니다.

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;
            }

            // 먼저 랩된 메세지를 예상합니다.
            if (!unwrapped)
            {
                response = unwrap(context, request);
                unwrapped = true;
                continue; // 다음 요구를 처리합니다.
            }

            // MIC 다음에 옵니다.
            verifyMIC(context, request, response);

            // 증명서를 위임한 경우 개시자 역할을 담당합니다.
            if (context.getCredDelegState())
                useDelegatedCred(context);

            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "clientServer=" + clientServer
                + ", beenInitiator=" + beenInitiator);

            // 필요하면 자신의 증명서를 사용하여 개시자로서 실행합니다.
            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();

          // 각 스레드에 있는 예외로 인해 서버가 다운되지 않도록
          // 스레드 개개의 예외를 처리하십시오.
  return;
      } finally {
          if (context != null)
          {
         try {
                   context.dispose();
               } catch (Exception exc) {}
          }
      }
}

synchronized void runAsInitiatorOnce(Thread thread)
    throws InterruptedException
{
```

```
    if (!beenInitiator)
    {
        // 후속 스레드가 runAsInitiator를 시도하지 않도록
        // 플래그를 참으로 먼저 설정합니다.
        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); // 자신의 증명서를 사용합니다.
    }
 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); // Subject의 증명서를 사용하지 마십시오.
            server.processArgs(args);
            server.initialize();
            server.processRequests();
        } catch (Exception exc) {
            debug.out(Debug.OPTS_CAT_APPLICATION, programName + ": EXCEPTION");
            exc.printStackTrace();
            throw exc;
        }
    }
}
```

샘플: IBM JGSS JAAS 작동 가능 클라이언트 프로그램

샘플 클라이언트 프로그램 사용에 대해서는 IBM JGSS 샘플 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// IBM Java GSS 1.0 샘플 JAAS 작동 가능 클라이언트 프로그램

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;

/**
 * JAAS를 사용하는 Java GSS 샘플 클라이언트.
 *
 * JAAS 로그인하고, JAAS 로그인 문맥내에서 작동합니다.
 *
 * JAVA 변수
 * javax.security.auth.useSubjectCredsOnly를 설정하지 않고
 * 변수를 디폴트 값인 참으로 남겨두면,
 * Java GSS가 (클라이언트가 작성한) 로그인 문맥과 연관된 JAAS 주제의
 * 증명서를 확보할 수 있습니다.
 *
```

```
 * JAASClient는 모든 측면에서 그 수퍼클래스 {@link Client Client}와
 * 동일하며, JAAS 이외 샘플 클라이언트와 서버에 대해 실행될 수 있습니다.
 */

class JAASClient extends Client
{
    JAASClient(String programName) throws Exception
    {
        // useSubjectCredsOnly를 설정하지 마십시오. 프로그램명만을 설정하십시오.
        // 설정되지 않은 경우 useSubjectCredsOnly의 디폴트값은 "참"입니다.
        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 ...");
    }
}
```

샘플: IBM JGSS JAAS 작동 가능 서버 프로그램

샘플 서버 프로그램 사용에 대해서는 IBM JGSS 샘플 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// IBM Java GSS 1.0 샘플 JAAS 작동 가능 서버 프로그램

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;

/**
 * JAAS를 사용하는 Java GSS 샘플 서버
 *
 * JAAS 로그인하고, JAAS 로그인 문맥내에서 작동합니다.
 *
 * JAVA 변수
 * javax.security.auth.useSubjectCredsOnly를 설정하지 않고
 * 변수를 디폴트값인 참으로 남겨두면,
 * Java GSS가 (서버가 작성한) 로그인 문맥과 연관된 JAAS 주제의
 * 증명서를 확보할 수 있습니다.
 *
 * JAASServer는 모든 측면에서 그 수퍼클래스 {@link Server Server}와
 * 동일하며, JAAS 이외 샘플 클라이언트와 서버에 대해 실행될 수 있습니다.
 */

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 {
            // useSubjectCredsOnly를 설정하지 않습니다.
            // 설정되지 않은 경우 useSubjectCredsOnly의 디폴트 값은 "참"입니다.

            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;
        }
    }
}
```

샘플: Kerberos 구성 파일

샘플 구성 파일 사용에 대해서는 IBM JGSS 샘플 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
# --------------------------------------------------------------------------------
# JGSS 샘플 어플리케이션을 실행하기 위한 Kerberos 구성 파일.
# 사용자의 환경에 맞도록 항목을 수정하십시오.
#--------------------------------------------------------------------------------

[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
```

샘플: JAAS 로그인 구성 파일

샘플 구성 파일 사용에 대해서는 IBM JGSS 샘플 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
/**
 * --------------------------------------------------------------------------------
 * JGSS 샘플용 JAAS 로그인 구성
 * --------------------------------------------------------------------------------
 *
 * 코드 예 면책사항 정보
 * IBM은 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록
 * 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.
 * 모든 샘플 코드는 IBM에 의해 예시 목적으로만 제공됩니다.
 * 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다.
 * 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.
 * 여기에 포함된 모든 프로그램은 어떠한 보증없이 "현상태대로" 제공됩니다.
```

```
 * 타인 권리의 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증은
 * 명시적으로 거부됩니다.
 *
 *
 * 지원되는 옵션:
 *     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)
 *
 * Kerberos 구성 파일의 디폴트 영역은 지정된
 * 프린시펄에 영역 구성요소가 없는 경우 사용됩니다.
 */

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;
};
```

샘플: JAAS 정책 파일

샘플 정책 파일에 대해서는 IBM JGSS 샘플 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// ------------------------------------------------------------------------
// JGSS 샘플 어플리케이션을 실행하기 위한 JAAS 정책 파일
// 사용자의 환경에 맞도록 이들 권한을 수정하십시오.
// 위에서 언급했던 이외의 목적으로는 권장하지 않습니다.
// 특히, 이 정책 파일이나 그 내용을 사용하여
// 제품 환경에서 자원을 보호하지 마십시오.
//
// 코드 예 면책사항 정보
// IBM은 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든
// 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.
// 모든 샘플 코드는 IBM에 의해 예시 목적으로만 제공됩니다.
// 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다.
// 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.
// 여기에 포함된 모든 프로그램은 어떠한 보증없이 "현상태대로" 제공됩니다.
// 타인 권리의 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증은
// 명시적으로 거부됩니다.
//
// ------------------------------------------------------------------------
```

```
//--------------------------------------------------------------------------
// 클라이언트만을 위한 권한
//--------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "bob@REALM.IBM.COM"
{
    // foo는 서버와의 문맥을 시작할 수 있어야 합니다.
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "initiate";

    // foo가 자신의 신임을 서버로 위임할 수 있습니다.
    permission javax.security.auth.kerberos.DelegationPermission
        "\"gss_service/myhost.ibm.com@REALM.IBM.COM\" \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
};

//--------------------------------------------------------------------------
// 서버만을 위한 권한
//--------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service/myhost.ibm.com@REALM.IBM.COM"
{
    // 호스트에서 네트워크 연결을 승인하기 위한 서버의 권한
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // JGSS 문맥을 승인하기 위한 서버의 권한
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "accept";

    // 서버는 2차(백업) 서버와 통신할 때 클라이언트로서 작동합니다.
    // 이 권한을 사용하면 서버가 2차 서버와의 문맥을 시작할 수 있습니다.
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "initiate";
};

//--------------------------------------------------------------------------
// 2차 서버를 위한 권한
//--------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service2/myhost.ibm.com@REALM.IBM.COM"
{
    // 호스트에서 네트워크 연결을 승인하기 위한 2차 서버의 권한
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // JGSS 문맥을 승인하기 위한 서버의 권한
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "accept";
};
```

샘플: Java 정책 파일

샘플 정책 파일에 대해서는 IBM JGSS 샘플 다운로드 및 실행을 참조하십시오.

주: 중요한 법률 정보에 대해서는 코드 예 면책 사항을 참조하십시오.

```
// ----------------------------------------------------------------------------
// iSeries 서버에서 JGSS 샘플 어플리케이션을 실행하기 위한
// Java 정책 파일.
// 사용자의 환경에 맞도록 이들 권한을 수정하십시오.
// 위에서 언급했던 이외의 목적으로는 권장하지 않습니다.
// 특히, 제품 환경에서 자원을 보호할 목적으로
// 이 정책 파일이나 내용을 사용하지 마십시오.
//
// 코드 예 면책사항
// IBM은 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든
// 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.
// 모든 샘플 코드는 IBM에 의해 예시 목적으로만 제공됩니다.
// 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다.
// 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.
// 여기에 포함된 모든 프로그램은 어떠한 보증없이 "현상태대로" 제공됩니다.
// 타인 권리의 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증은
// 명시적으로 거부됩니다.
//
//-----------------------------------------------------------------------------

grant CodeBase "file:ibmjgsssample.jar" {
    // Java 1.3의 경우
    permission javax.security.auth.AuthPermission "createLoginContext";

    // Java 1.4의 경우
    permission javax.security.auth.AuthPermission "createLoginContext.JAASClient";
    permission javax.security.auth.AuthPermission "createLoginContext.JAASServer";

    permission javax.security.auth.AuthPermission "doAsPrivileged";

    // KDC의 티켓을 요청하는 권한
    permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/REALM.IBM.COM@REALM.IBM.COM", "initiate";

    // sun.security.action 클래스에 액세스하는 권한
    permission java.lang.RuntimePermission "accessClassInPackage.sun.security.action";

    // Java 등록 정보 전체에 액세스합니다.
    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";

    // Kerberos KDC 호스트와 통신하는 권한
    permission java.net.SocketPermission "kdc.ibm.com", "connect,accept,resolve";

    // localhost에서 샘플을 실행합니다.
    permission java.net.SocketPermission "myhost.ibm.com", "accept,connect,resolve";
```

```
    permission java.net.SocketPermission "localhost", "listen,accept,connect,resolve";

    // 몇몇 가능한 Kerberos config 위치에 액세스합니다.
    // 사용자 환경에 적용 가능하도록 파일 경로를 수정합니다.
    permission java.io.FilePermission "${user.home}/krb5.ini", "read";
    permission java.io.FilePermission "${java.home}/lib/security/krb5.conf", "read";

    // 서버키를 구할 수 있도록 Kerberos 키 표에 액세스합니다.
    permission java.io.FilePermission "/QIBM/UserData/OS400/NetworkAuthentication/keytab
    /krb5.keytab",
    "read";

    // 사용자의 Kerberos 증명서 캐시에 액세스합니다.
    permission java.io.FilePermission "${user.home}/krb5cc_${user.name}", "read";
};
```

샘플: IBM JGSS 샘플의 javadoc 정보 다운로드 및 보기

IBM JGSS 샘플 프로그램의 문서를 다운로드하여 보려면, 다음 단계를 완료하십시오.

1. javadoc 정보를 저장할 기존 디렉토리를 선택하십시오(또는 새 디렉토리를 작성하십시오).
2. javadoc 정보(jgsssampledoc.zip)를 디렉토리로 다운로드하십시오.
3. jgsssampledoc.zip에서 디렉토리로 파일을 추출하십시오.
4. 브라우저를 사용하여 index.htm 파일에 액세스하십시오.

코드 예 면책사항

IBM은 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.

모든 샘플 예제는 IBM에 의해 예시 목적으로만 제공됩니다. 이러한 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.

여기에 포함된 모든 프로그램은 어떠한 보증없이 "현상태대로" 제공됩니다. 타인 권리의 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증은 명시적으로 거부됩니다.

샘플: 샘플 프로그램 다운로드 및 실행

샘플을 수정 또는 실행하기 전에, 399 페이지의 『샘플 프로그램 설명』을 읽으십시오.

샘플 프로그램을 실행하려면, 다음 타스크를 수행하십시오.

1. 샘플 파일을 iSeries 서버로 다운로드

2. 샘플 파일 실행 준비
3. 샘플 프로그램 실행

샘플 실행 방법에 대해서는 예: 비JAAS 샘플 실행을 참조하십시오.

코드 예 면책사항

IBM은 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.

모든 샘플 예제는 IBM에 의해 예시 목적으로만 제공됩니다. 이러한 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.

여기에 포함된 모든 프로그램은 어떠한 보증없이 "현상태대로" 제공됩니다. 타인 권리의 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증은 명시적으로 거부됩니다.

샘플: IBM JGSS 샘플 다운로드

샘플을 수정 또는 실행하기 전에 399 페이지의 『샘플 프로그램 설명』을 읽으십시오.

샘플 파일을 다운로드하고 이들을 iSeries 서버에 저장하려면, 다음 단계를 완료하십시오.

1. iSeries 서버에서, 샘플 프로그램, 구성 파일 및 정책 파일을 저장하려는 기존 디렉토리(또는 새로 작성)를 선택하십시오.
2. 샘플 프로그램(ibmjgsssample.zip)을 다운로드하십시오.
3. 서버 디렉토리에 ibmjgsssample.zip 압축 파일을 푸십시오.

ibmjgsssample.jar 압축 파일이 풀린 후 다음 조치를 수행합니다.

- 샘플 .class 파일이 들어있는 ibmgjsssample.jar를 선택된 디렉토리로 둡니다.
- 구성 및 정책 파일이 있는 서브디렉토리(이름은 config)를 작성합니다.
- 샘플 .java 소스 파일이 들어있는 서브디렉토리(이름은 src)를 작성합니다.

관련 정보

관련 타스크에 대해 읽거나 예를 찾아 볼 수 있습니다.

- 샘플 파일 실행 준비
- 샘플 프로그램 실행
- 예: 비JAAS 샘플 실행

샘플: 샘플 프로그램 실행 준비

샘플을 수정 또는 실행하기 전에 399 페이지의 『샘플 프로그램 설명』을 읽으십시오.

소스 코드를 다운로드한 다음, 샘플 프로그램을 실행하기 전에 다음 타스크를 수행해야 합니다.

- 사용자 환경에 적합하도록 구성 및 정책 파일을 편집하십시오. 자세한 정보는 각 구성 및 정책 파일의 주석을 참조하십시오.
- java.security 파일에 사용자의 iSeries 서버에 맞는 설정이 들어 있는 지 확인하십시오. 자세한 내용은 387 페이지의 『Java 마스터 보안 등록 정보 파일』을 참조하십시오.
- 사용 중인 J2SDK 버전에 맞는 iSeries 서버의 디렉토리로 수정된 Kerberos 구성 파일(krb5.conf)을 놓으십시오.
  - J2SDK 버전 1.3의 경우: /QIBM/ProdData/Java400/jdk13/lib/security
  - J2SDK 버전 1.4의 경우: /QIBM/ProdData/Java400/jdk14/lib/security

관련 정보

관련 타스크에 대한 내용을 읽거나 예를 찾을 수 있습니다.

- 샘플 파일을 iSeries 서버로 다운로드
- 샘플 프로그램 실행
- 예: 비JAAS 샘플 실행

샘플: 샘플 프로그램 실행

샘플을 수정 또는 실행하기 전에 399 페이지의 『샘플 프로그램 설명』을 읽으십시오.

소스 코드를 다운로드하여 수정한 다음, 샘플 중 하나를 실행할 수 있습니다.

샘플을 실행하려면, 서버 프로그램을 먼저 시작해야 합니다. 서버 프로그램은 클라이언트 프로그램을 시작하기 전에 실행 중이어야 하며, 연결을 수신할 준비가 되어야 합니다. 서버가 포트 <server_port>에서 청취 중인 경우, 연결을 수신할 준비가 된 것입니다. <server_port>를 기억하거나 기록해 두십시오. 이것은 클라이언트를 시작할 때 지정해야 하는 포트 번호입니다.

다음 명령을 사용하여 샘플 프로그램을 시작하십시오.

```
java [-Dproperty1=value1 ... -DpropertyN=valueN] com.ibm.security.jgss.test.<program> [options]
```

여기서

- [-DpropertyN=valueN]은 하나 이상의 선택적 Java 등록 정보로서, 구성과 정책 파일의 이름, JGSS 디버그 옵션 및 보안 관리자가 포함됩니다. 자세한 정보는 다음 예와 JGSS 어플리케이션 실행을 참조하십시오.
- <program>은 실행하려는 샘플 프로그램을 지정하는 필수 매개변수입니다(Client, Server, JAASClient 또는 JAASServer).
- [options]는 실행하려는 샘플 프로그램의 선택적 매개변수입니다. 지원되는 옵션 리스트를 표시하려면, 다음 명령을 사용하십시오.

```
java com.ibm.security.jgss.test.<program> -?
```

주: Java 등록 정보 javax.security.auth.useSubjectCredsOnly를 거짓으로 설정하여 JGSS 가능 샘플에서 JAAS 피처를 오프로 끄십시오. 물론, JAAS 가능 샘플의 디폴트 값은 JAAS를 온으로 설정하는 것으로서, 이는 등록 정보 값이 참임을 의미합니다. 비JAAS 클라이언트와 서버 프로그램은 사용자가 등록 정보 값을 명시적으로 설정하지 않는 한, 등록 정보를 거짓으로 설정합니다.

관련 정보

관련 타스크에 대해 읽거나 예를 찾아 볼 수 있습니다.

- 샘플 파일을 iSeries 서버로 다운로드
- 샘플 파일 실행 준비
- 예: 비JAAS 샘플 실행

IBM JGSS javadoc 참조 정보

IBM JGSS의 javadoc 참조 정보에는 org.ietf.jgss api 패키지 및 Java 버전의 일부 Kerberos 증명서 관리 툴에 있는 클래스와 메소드가 포함됩니다.

JGSS에 여러 범용으로 액세스 가능한 패키지(예를 들면 com.ibm.security.jgss 및 com.ibm.security.jgss.spi)가 포함되더라도, 표준화된 org.ietf.jgss 패키지의 API만 사용해야 합니다. 이 패키지만 사용하면 어플리케이션이 GSS-API 스펙에 맞으며, 최적의 상호운영성과 이식성이 보증됩니다.

- org.ietf.jgss
- kinit
- ktab
- klist

제 5 장 IBM Developer Kit for Java를 사용하여 Java 프로그램 성능 조정

iSeries 서버용 Java 어플리케이션을 빌드할 때 여러 측면의 Java(TM) 어플리케이션 성능을 고려해야 합니다. 다음은 더 좋은 성능을 얻을 수 있는 방법에 대한 세부사항과 힌트에 대한 일부 링크입니다.

- 클래스 파일, JAR 파일 또는 ZIP 파일을 실행하기 전에 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하여 Java 코드의 런타임 성능을 향상시키십시오.
- 최상의 정적 컴파일 성능을 달성하도록 최적화 레벨을 변경하십시오.
- 최적의 가비지 콜렉션 성능을 위해 값을 신중하게 설정하십시오.
- 원시 메소드만을 사용하여 비교적 오래 실행되며 Java에서 직접 사용할 수 없는 시스템 기능을 시작하십시오.
- 메소드 인라인 처리를 수행하고 메소드 호출 성능을 크게 개선하려면 컴파일시 `javac -o` 옵션을 사용하십시오.
- 어플리케이션 전반에서 정상적인 흐름이 일어나지 않는 경우 Java 예외를 사용하십시오.

Java 프로그램에서 성능 문제점을 찾으려면 PEX(성능 탐색기)와 함께 다음 툴을 사용하십시오.

- iSeries JVM(Java Virtual Machine)을 사용하여 Java 추적 이벤트를 수집할 수 있습니다.
- 각 Java 메소드에 소요되는 시간을 판별하려면 Java 호출 추적을 사용하십시오.
- Java 프로파일링은 각 Java 메소드와 Java 프로그램이 사용하는 모든 시스템 기능에 소요되는 CPU 시간의 상대적 크기를 찾습니다.
- Java 성능 자료 콜렉터를 사용하여 iSeries 서버에서 실행되는 프로그램에 대한 프로파일 정보를 제공하십시오.

모든 작업 세션이 PEX를 시작하고 종료할 수 있습니다. 일반적으로, 수집되는 자료는 시스템 전체에 걸친 것이며 Java 프로그램을 포함하여 시스템의 모든 작업에 관련됩니다. 때로는 Java 어플리케이션 내부에서 성능 콜렉션을 시작하고 중단하는 것이 필요할 수 있습니다. 이것은 콜렉션 시간을 줄이며 대개 호출 또는 리턴 추적에 의해 생성되는 큰 볼륨의 자료를 줄일 수 있습니다. PEX는 Java 스레드 안에서 수행할 수 없습니다. 콜렉션을 시작하고 중단하려면 대기행렬 또는 공유 메모리로 독립적 작업으로 대화하는 원시 메소드에 기록해야 합니다. 그런 다음, 두 번째 작업이 적절한 시간에 콜렉션을 시작하고 중단합니다.

어플리케이션 레벨 성능 자료 이외에 기존의 iSeries 시스템 레벨 성능 분석 툴을 사용할 수 있습니다. 이러한 툴은 Java 스레드를 기본으로 하는 통계를 보고합니다.

PEX 보고서의 예는 Performance Tools for iSeries, SC41-5340  책을 참조하십시오.

Java 런타임 성능 고려사항

Java(TM) 코드의 런타임 성능을 현저히 향상시키려면 Java 클래스 파일, JAR 파일 또는 ZIP 파일을 실행하기 전에 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하십시오. CRTJVAPGM 명령은 바이트 코드를 사용하여 iSeries 서버에 대해 최적화된 원시 명령어가 들어 있는 Java 프로그램을 작성하고 이 Java 프로그램 오브젝트를 클래스 파일, JAR 파일 또는 ZIP 파일과 연관시킵니다.

Java 클래스 파일, JAR 파일 또는 ZIP 파일을 실행하기 전에 CRTJVAPGM 명령을 사용하지 않을 경우 최적 Java 프로그램이 최적화 레벨 10으로 작성되지 않기 때문에 처음에는 Java 코드가 더 느리게 실행됩니다. 후속적으로 실행하고 나면 Java 프로그램이 저장되고 클래스 파일이나 JAR 파일과의 연관 관계나 유지되므로 실행 속도가 훨씬 빨라집니다. 바이트 코드를 해석적으로 실행하면 어플리케이션을 개발하는 동안 허용될 정도의 성능을 얻을 수 있지만 작성 환경에서는 Java 코드를 실행하기 전에 CRTJVAPGM 명령을 사용하려고 할 수도 있습니다.

JIT 컴파일러는 코드를 컴파일하여 성능을 향상시키며, 코드는 특정한 JVM(Java Virtual Machine) 런타임 환경에 대해 최적화됩니다. 사용자의 필요에 맞는 최적화 기법을 결정하기 위해 CRTJVAPGM을 수행할 필요가 없는이점을 JIT의 사용에 따르는 약간 느린 프로그램 시작 및 런타임 처리와 비교해야 합니다.

프로그램이 느리게 실행되고 있는 경우 DSPJVAPGM(Java 프로그램 표시) 명령을 입력하여 Java 프로그램의 속성을 보십시오.

클래스 로더 캐시

사용자 클래스 로더가 작성한 JVAPGM을 다시 사용하기 위해 캐시하여 사용자 클래스 로더의 성능을 향상시킬 수 있습니다. 다음 중 하나로 Java 등록 정보를 작동시키도록 지정하지 않으면 이 옵션은 작동 불가능합니다.

- /QIBM/UserData/Java400/SystemDefault.properties
- /home//SystemDefault.properties
- RUNJVA CL 명령에서
- JAVA QSH 명령에서

다음의 등록 정보를 사용하여 이 기능을 작동시킵니다.

os400.define.class.cache.file

등록 정보의 값은 유효한 JAR(Java ARchive) 파일의 이름(전체 경로 포함)이어야 합니다. JAR 파일에는 유효한 JAR 디렉토리(`jar` QSH 명령으로 빌드함)가 포함되어야 하지만 `jar` 명령이 기능을 수행하도록 하는 데 필요한 단일 멤버 이외의 다른 내용이 없어야 합니다. 이 JAR 파일은 Java 클래스 경로에 포함시키서는 안됩니다.

예를 들면, 다음 행은 /QIBM/UserData/Java400/SystemDefault.properties로 추가됩니다.

```
os400.define.class.cache.file=/QIBM/ProdData/Java400/QDefineClassCache.jar
```

Developer Kit for Java는 적합한 JAR 파일을 /QIBM/ProdData/Java400/QDefineClassCache.jar로 설치하고 등록 정보 파일 예를 /QIBM/ProdData/Java400/SystemDefaultCacheExample.properties로 설치합니다. SystemDefaultCacheExample.properties 파일을 /QIBM/UserData/Java400/으로 복사하고 SystemDefault.properties로 이름을 변경하여 캐싱 기능을 글로벌로 작동시킬 수 있습니다.

주: 이렇게 변경하기 전에 이 디렉토리에 기존의 SystemDefault.properties 파일이 없는지 확인하십시오. 또는 단일 사용자의 환경에만 영향을 주려면 파일을 /home//으로 복사하고 비슷하게 이름을 변경할 수 있습니다.

캐시하도록 지정된 JAT 파일에서 DSPJVAPGM을 사용하여 캐시할 JVAPGM의 수를 판별할 수 있습니다. DSPJVAPGM 화면의 **Java 프로그램** 필드는 캐시한 JVAPGM의 수를 나타내며 **Java 프로그램 크기** 필드는 캐시한 JVAPGM이 소모한 기억장치의 양을 나타냅니다. DSPJVAPGM이 캐시에 사용된 JAR 파일에 적용되면 화면의 다른 필드는 의미가 없습니다.

캐싱을 가능하게 하는 os400.define.class.cache.file Java 등록 정보 이외에 두 개의 다른 등록 정보를 지정하여 캐싱 특성을 제어할 수 있습니다.

os400.define.class.cache.hours

이 등록 정보를 사용하여 캐시에서 JVAPGM을 지속해야 하는 시간(시)을 지정할 수 있습니다.

os400.define.class.cache.maxpgms

이 등록 정보를 사용하면 캐시할 수 있는 최대 JVAPGM 수를 지정하여 이 한계를 초과했을 때 가장 오래된 JVAPGM을 먼저 삭제하도록 할 수 있습니다. 캐시된 JVAPGM의 지속 시간을 판별하는 데 사용하는 시간 값은 JVAPGM을 참조할 때마다 갱신됩니다.

예:

```
    os400.define.class.cache.hours=48
os400.define.class.cache.maxpgms=10000
```

os400.define.class.cache.hours의 디폴트 값은 168시간(일주일)이며 최대값은 9999입니다. os400.define.class.cache.maxpgms의 디폴트 값은 5000이며 최대값은 40000입니다. 이들 등록 정보 중 하나에 대해 값 0을 지정했거나 값을 유효한 십진수로 분석할 수 없는 경우에 디폴트 값을 사용합니다.

캐싱에 대해 앞에서 언급한 등록 정보에 대한 자세한 정보는 시스템 등록 정보(32 페이지 참조)를 참조하십시오.

«

Java 프로그램을 실행할 때 사용할 모드 선택

Java(TM) 프로그램을 실행할 때, 사용하려는 모드를 선택할 수 있습니다. 모든 모드는 코드를 검증하고 사전 검증된 양식의 프로그램을 유지하기 위해 Java 프로그램 오브젝트를 작성합니다. 다음 모드의 모든 것을 사용할 수 있습니다.

- 해석

- 직접 처리
- JIT(Just-In-Time) 컴파일
- JIT(Just-In-Time) 컴파일 및 직접 처리

선택 모드	세부사항
해석	각 바이트코드는 런타임에 해석됩니다. 해석 모드에서 Java 프로그램 실행에 대한 정보는 RUNJVA(Java 실행) 명령을 참조하십시오.
직접 처리	메소드에 대한 기계 명령어가 해당 메소드에 대한 첫 번째 호출 중에 생성되고 다음에 프로그램이 실행할 때 사용하도록 저장됩니다. 또한 한 사본이 전체 시스템에 대해 공유됩니다. 직접 처리를 사용한 Java 프로그램 실행에 대한 정보는 RUNJVA(Java 실행) 명령을 참조하십시오.

JIT(Just-In-Time) 컴파일	메소드에 대한 기계 명령어는 해당 메소드에 대한 첫 번째 호출 중에 생성되고 JVM(Java Virtual Machine) 실행 기간 동안 저장됩니다.
	
	JIT(Just-In-Time) 컴파일러를 사용하려면 컴파일러 값을 `jitc`로 설정해야 합니다. 환경 변수를 추가하거나 java.compiler 시스템 등록 정보를 설정하여 값을 설정할 수 있습니다. 컴파일러 값을 설정하려면 아래 리스트에서 한 메소드를 선택하십시오.
	• iSeries 서버의 명령행 프롬트에서, ADDENVVAR(Add Environment Variable) 명령을 사용하여 환경 변수를 추가하십시오. 그런 다음, RUNJVA(Java 실행) 명령이나 JAVA 명령을 사용하여 Java 프로그램을 실행하십시오. 예를 들어 다음을 사용하십시오.
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc)`
	`JAVA CLASS(Test)`
	• iSeries 명령행에서 java.compiler 시스템 등록 정보를 설정하십시오. 예를 들면, `JAVA CLASS(Test) PROP((java.compiler jitc))`를 입력하십시오.
	• Qshell 인터프리터 명령행에서 java.compiler 시스템 등록 정보를 설정하십시오. 예를 들어, `java -Djava.compiler=jitc Test`를 입력하십시오.
	
	일단 이 값을 설정하면 JIT 컴파일러가 Java 코드를 실행하기 전에 모든 코드를 최적화합니다.

선택 모드	세부사항

JIT(Just-In-Time) 컴파일 및 직접 처리	JIT(Just-In-Time) 컴파일러를 사용하는 가장 일반적인 방법은 `jit_de` 옵션을 사용하는 것입니다. 이 옵션을 갖고 실행할 때 이미 직접 처리로 최적화된 프로그램이 직접 처리 모드에서 실행합니다. 직접 최적화에 대해 최적화되지 않은 프로그램은 JIT 모드에서 실행됩니다.
	
	JIT와 직접 처리를 함께 사용하려면 컴파일러 값을 `jitc_de`로 설정해야 합니다. 환경 변수를 추가하거나 java.compiler 시스템 등록 정보를 설정하여 값을 설정할 수 있습니다. 컴파일러 값을 설정하려면 리스트에서 한 메소드를 선택하십시오.
	• iSeries 명령행에 ADDENVVAR(환경 변수 추가) 명령을 입력하여 환경 변수를 추가하십시오. 그런 다음, RUNJVA(Java 실행) 명령이나 JAVA 명령을 사용하여 Java 프로그램을 실행하십시오. 예를 들면, 다음과 같이 입력하십시오.
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc_de)`
	`JAVA CLASS(Test)`
	• iSeries 명령행에서 java.compiler 시스템 등록 정보를 설정하십시오. 예를 들면, `JAVA CLASS(Test) PROP((java.compiler jitc_de))`를 입력하십시오.
	• Qshell 인터프리터 명령행에서 java.compiler 시스템 등록 정보를 설정하십시오. 예를 들어, `java -Djava.compiler=jitc_de Test`를 입력하십시오.
	
	일단 이 값이 설정되면 직접 처리로서 작성된 클래스 파일에 대한 Java 프로그램이 사용됩니다. Java 프로그램이 직접 처리로서 작성되지 않은 경우 클래스 파일은 실행하기 전에 JIT에 의해 최적화됩니다. 자세한 정보는 JIT(Just-In-Time) 컴파일러 및 직접실행의 비교를 참조하십시오.

Java 프로그램을 실행할 수 있는 세 가지 방법이 있습니다(CL, QSH, 및 JNI). 각각에는 모드를 지정하기 위한 고유의 방법이 있습니다. 다음의 표는 이러한 방법이 어떻게 이루어지는지를 보여줍니다.

모드	CL 명령	QShell 명령	JNI 호출 API
Interpret	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(디폴트)	INTERPRET(*OPTIMIZE) OPTIMIZE(*JIT)	-Djava.compiler=″jitc_de″	os400.run.mode=″jitc_de″

Java 인터프리터

Java(TM) 인터프리터는 특정 하드웨어 플랫폼에 대한 Java 클래스 파일을 해석하는 JVM(Java Virtual Machine)의 한 파트입니다. Java 인터프리터는 각 바이트 코드를 해독하고 이 바이트코드에 대한 일련의 기계 명령어를 실행합니다.

정적 컴파일

Java(TM) 변환 프로그램은 클래스 파일을 iSeries JVM(Java Virtual Machine)을 사용하여 실행을 준비하도록 사전 처리하는 IBM Operating System/400(OS/400) 구성요소입니다. Java 변환 프로그램은 일관성 있고 클래스 파일과 연관되는 최적화된 프로그램 오브젝트를 작성합니다. 디폴트의 경우 프로그램 오브젝트는 해당 클래스의 컴파일된 64비트 RISC 기계 명령어 버전입니다. Java 인터프리터는 런타임에 최적화 프로그램 오브젝트를 해석하지 않습니다. 대신, 클래스 파일이 로드될 때 직접 실행합니다.

Java 프로그램은 디폴트로 JIT를 사용하여 최적화됩니다. Java 변환 프로그램을 사용하려면 CRTJVAPGM을 수행하거나 RUNJVA 또는 JAVA 명령에 변환 프로그램의 사용을 지정하십시오.

Java 변환 프로그램을 명시적으로 시작하려면 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하십시오. CRTJVAPGM 명령은 명령이 실행하는 동안 클래스 파일이나 JAR 파일을 최적화하므로, 프로그램이 실행하는 동안 필요한 작업이 없습니다. 이것은 프로그램이 처음으로 실행할 때 프로그램의 속도를 향상시킵니다. 디폴트 최적화에 의존하는 대신에 CRTJVAPGM 명령을 사용하는 것이 가능한 최상의 최적화를 보장하고 클래스 파일이나 JAR 파일과 연관되는 Java 프로그램에 대한 공간을 더 잘 활용하는 방법입니다.

클래스 파일, JAR 파일 또는 ZIP 파일에 CRTJVAPGM 명령을 사용하면 파일에 있는 모든 클래스가 최적화되며, 그 결과 Java 프로그램 오브젝트의 일관성이 유지됩니다. 이것은 런타임 성능을 향상시킵니다. CRTJVAPGM 명령이나 CHGJVAPGM(Java 프로그램 변경) 명령을 사용하여 최적화 레벨을 변경하거나 디폴트 10이 아닌 최적화 레벨을 선택할 수도 있습니다. 최적화 레벨 40에서는 JAR 파일내의 클래스 사이에서 클래스간 바인딩이 수행되며 일부 경우에는 클래스가 인라인됩니다. 클래스간 바인딩은 호출 속도를 향상시킵니다. 인라인은 메소드 호출의 오버헤드를 완전히 제거합니다. 일부 경우에는 JAR 파일 또는 ZIP 파일내의 클래스 사이에 메소드를 인라인할 수 있습니다. CRTJVAPGM 명령에 OPTIMIZE(*INTERPRET)를 지정하면 명령에 지정되는 모든 클래스가 검증되고 해석 모드로 실행되도록 준비됩니다.

RUNJVA(Java 실행) 명령도 OPTIMIZE(*INTERPRET)를 지정할 수 있습니다. 이 매개변수는 연관된 프로그램 오브젝트의 최적화 레벨에 관계없이 JVM(Java Virtual Machine)하에서 수행되는 모든 클래스가 해석됨을 지정합니다. 이것은 최적화 레벨 40으로 변환된 클래스를 디버그할 때 유용합니다. 해석을 강제 실행하려면 INTERPRET(*YES)를 사용하십시오.

클래스 로더가 작성한 Java 프로그램의 재사용에 대한 정보는 436 페이지의 『클래스 로더 캐시』를 참조하십시오.

Java 정적 컴파일 성능 고려사항

사용자가 설정하는 최적화 레벨로 변환의 속도를 결정할 수 있습니다. 최적화 레벨 10이 가장 빠른 변환이지만, 결과 프로그램은 일반적으로 상위 최적화 레벨로 설정된 것보다 느립니다. 최적화 레벨 40은 변환에 긴 시간이 걸리지만 대개 빠르게 실행됩니다.

Java(TM) 프로그램 수가 적으면 레벨 40으로 최적화될 수 없습니다. 따라서 레벨 40으로 실행되지 않는 일부 프로그램은 대신 레벨 30으로 실행될 수 있습니다. 사용권 내부 코드 최적화 LICOPT 매개변수 스트링을 사용하여 최적화 레벨 40에서 실행되지 않는 프로그램을 실행할 수 있습니다. 그러나 사용자의 프로그램에 대해 레벨 30의 성능도 충분합니다.

다른 JVM(Java Virtual Machine)에서 작업하는 것 같이 Java 코드를 실행하는 데 문제가 있을 경우 레벨 40 대신에 레벨 30을 사용해보십시오. 레벨 30에서 실행되고 성능이 허용 가능하면 그 외 다른 작업이 요구되지 않습니다. 향상된 성능이 필요한 경우 다양한 형식의 최적화를 작동 가능 또는 불가능하게 하는 방법에 대한 내용은 LICOPT 매개변수 스트링을 참조하십시오. 예를 들어, 먼저 OPTIMIZE(40) LICOPT(NoPreresolveExtRef)를 사용하여 프로그램 작성을 시도할 수 있습니다. 어플리케이션에 사용할 수 없는 클래스에 대한 불능 호출이 들어 있을 경우 이 LICOPT 값은 문제 없이 프로그램을 실행시킵니다.

Java 프로그램이 작성된 최적화 레벨을 판별하기 위해 DSPJVAPGM(Java 프로그램 표시) 명령을 사용할 수 있습니다. Java 프로그램의 최적화 레벨을 변경하려면 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하십시오.

JIT 컴파일러

JIT(Just-In-Time) 컴파일러는 각 메소드에 대한 첫 번째 호출시에 해당 메소드에 대한 기계 명령어를 생성하는 플랫폼 고유 컴파일러입니다. 성능을 해석 이상으로 개선시키기 위해 JIT 컴파일러는 필요할 때 코드를 컴파일합니다.

JIT 컴파일러와 직접 처리 사이의 차이점을 이해하려면 JIT 컴파일러와 직접 처리의 비교를 참조하십시오.

java.compiler 등록 정보에 대해서는 Java(TM) 시스템 등록 정보를 참조하십시오. 실행하고 있는 버전을 선택하십시오.

JIT 컴파일러 및 직접 처리의 비교

JIT(Just-In-Time) 컴파일러나 직접 처리 모드를 사용하여 Java(TM) 프로그램 실행을 결정하려는 경우, 이 표에서는 사용자의 환경에 가장 적합한 최선의 결정에 도움이 되는 추가 정보를 제공합니다.

JIT 컴파일러	직접 처리
필요할 때 모든 메소드의 자동 컴파일을 제공합니다. 이 방법이 직접 처리보다 훨씬 더 빠릅니다.	사용자가 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하여 전체 클래스나 JAR 파일을 컴파일하거나 파일이 런타임에 자동으로 컴파일됩니다.

JIT 컴파일러	직접 처리
코드가 변경되는 동안 최적화 비용을 절감하기 위해 프로그램 개발 중에 사용됩니다. 또한 작거나 상대적으로 이용도가 낮은 어플리케이션의 배치에 사용됩니다. 코드가 런타임에 생성되거나 발견되는 매우 동적인 어플리케이션과 함께 사용할 수도 있습니다.	상대적으로 큰 어플리케이션에 사용됩니다. 최적화 레벨 40에서의 직접 처리 코드는 일반적으로 JIT 보다 더 빠릅니다. 서버 어플리케이션을 전개할 준비가 되었으면 최적화 레벨 40에서 직접 처리를 사용하십시오. 이것이 임의의 주어진 시간에 복수 사용자에 의해 사용 중일 수 있으며 계속 JIT를 사용하는 비용이 너무 큰 오버헤드이기 때문입니다.
최적화는 런타임에 빠르게 수행될 수 있는 것들로 제한되니다.	최적화가 런타임에 수행되지 않으므로 더 복잡한 최적화가 가능합니다.

최적화 레벨

최적화 레벨 필드에 값을 입력하여, 클래스 파일 오브젝트나 JAR 파일 오브젝트에 접속된 Java(TM) 프로그램의 최적화 레벨을 지정합니다. 이 옵션을 사용하여 Java 프로그램의 크기와 성능을 제어할 수 있습니다. 최적화 레벨이 나열되는 경우, 내부 양식에 iSeries 기계 명령어가 포함됩니다. 이러한 기계 명령어는 지정된 최적화 레벨에 따라 최적화됩니다. 프로그램이 실행되면 iSeries 서버가 기계 명령어를 직접 실행합니다.

다음 리스트는 최적화 레벨의 차이점과 각 레벨의 용도를 보여줍니다.

10

Java 프로그램에는 클래스 파일 바이트코드의 변환된 버전이 있지만 최소한의 추가 컴파일러가 최적화됩니다. 디버그 되는 동안 변수를 표시하며 변경할 수 있습니다. V5R1 이상의 경우 최적화 레벨 10은 프로그램을 컴파일하는 데 소요되는 시간을 불필요하게 연장할 수 있습니다. 그 대신 최소한 최적화 레벨 20을 사용하십시오.

20

Java 프로그램에는 클래스 파일 바이트코드의 컴파일 버전이 있으며 추가 컴파일러가 최적화됩니다. 디버그되는 동안 변수를 표시하지만 변경할 수 없습니다.

30

Java 프로그램에는 클래스 파일 바이트코드의 컴파일 버전이 있으며 최적화 레벨 20보다 더 많은 컴파일러를 최적화합니다. 최적화가 코드의 정확한 지점에서 중단하여 프로그램 변수를 표시하는 기능을 감소시키기 때문에 상위 최적화 레벨에서 디버깅하는 것이 훨씬 어렵습니다. 디버그되는 동안 변수를 표시하지만 변경할 수 없습니다. 제시된 값은 변수의 현재 값이 아닐 수도 있습니다.

40

Java 프로그램에는 클래스 파일 바이트코드의 컴파일 버전이 있으며 최적화 레벨 30보다 더 많은 컴파일러를 최적화합니다. 최적화 레벨 40에는 클래스 간 최적화가 포함됩니다. 일부의 경우에는 관련이 없는 클래스(상속 또는 포함과 같은 관련이 없음)에 대해 정적 초기화 프로그램을 실행하는 순서가 정적 초기화 스펙에 요약된 것과 다를 수 있습니다. 또한 최적화 레벨 40에서 최적화는 호출 및 명령어를 추적할 수 없습니다.

주: Java 프로그램이 최적화에 실패하거나 최적화 레벨 40의 예외를 벗어나는 경우, 최적화 레벨 30을 사용하십시오.

Java 가비지 콜렉션

가비지 콜렉션은 프로그램이 더 이상 참조하지 않는 오브젝트가 사용하는 기억장치를 비우는 프로세스입니다. 가비지 콜렉션을 사용하여, 프로그래머는 더 이상 오브젝트를 명시적으로 "해제" 또는 "삭제"하기 위해 오류가 발생할 가능성이 있는 코드를 작성할 필요가 없습니다. 이 코드는 자주 "메모리 유출" 프로그램 오류를 일으킵니다. 가비지 콜렉터는 사용자 프로그램이 더 이상 도달할 수 없는 오브젝트 또는 오브젝트 그룹을 자동으로 검출합니다. 모든 프로그램 구조에서 그 오브젝트에 대한 참조가 없기 때문입니다. 일단 오브젝트가 수집되면 공간을 할당하여 다르게 사용할 수 있습니다.

312 페이지의 『Java 런타임 환경』에는 더 이상 사용하지 않은 메모리를 비우는 가비지 콜렉터가 포함됩니다. 가비지 콜렉터는 필요할 때마다 자동으로 수행됩니다.

가비지 콜렉터는 또한 java.lang.Runtime.gc() 메소드를 사용하는 Java 프로그램의 제어에서 명시적으로 시작됩니다.

IBM Developer Kit for Java 특징에 대해서는 IBM Developer Kit for Java 확장 가비지 콜렉션을 참조하십시오.

IBM Developer Kit for Java 확장 가비지 콜렉션

IBM Developer Kit for Java(TM)는 확장 가비지 콜렉터 알고리즘을 구현합니다. 이 알고리즘을 사용하면 Java 프로그램 조작을 중단시켜야만 가능한 오브젝트도 중단없이 찾아서 콜렉션 처리할 수 있습니다. 동시 처리 콜렉터가 협력하여 단일 스레드 대신 실행 스레드 아래에 있는 오브젝트에 대한 참조를 탐색합니다.

많은 가비지 콜렉터가 "모든 작업을 중지시킵니다". 즉, 콜렉션 주기가 발생하는 곳에서 가비지 콜렉터가 작업을 하는 동안 가비지 콜렉션을 수행하는 스레드를 제외한 모든 스레드가 중단됩니다. 이 상황이 발생하면 Java 프로그램은 실행을 멈추고, 플랫폼의 모든 복수 프로세서 기능은 콜렉터가 실행되는 동안 Java에 사용되지 못합니다. iSeries 알고리즘은 모든 프로그램 스레드를 동시에 중단하지 않습니다. 프로그램 스레드는 가비지 콜렉터가 타스크를 수행하는 동안에도 작업을 계속할 수 있습니다. 따라서 중단을 막고 모든 프로세서는 가비지 콜렉션 중에도 사용됩니다.

가비지 콜렉션은 사용자가 JVM을 시작할 때 지정한 매개변수에 기초하여 자동으로 발생합니다. 또한 java.lang.Runtime.gc() 메소드를 사용하여 Java 프로그램 제어하에서 가비지 콜렉션을 명시적으로 시작할 수도 있습니다.

기본적인 정의에 대해서는 Java 가비지 콜렉션을 참조하십시오.

Java 가비지 콜렉션 성능 고려사항

iSeries JVM(Java(TM) virtual machine)에서 가비지 콜렉션은 연속적인 비동기 모드로 동작합니다. RUNJVA(Java 실행) 명령의 GCHINL(가비지 콜렉션 초기 크기) 매개변수가 어플리케이션 성능에 영향을 줄

수 있습니다. 이 매개변수는 가비지 콜렉션 사이에서 허용되는 새로운 오브젝트 공간의 *크기*를 지정합니다. 작은 값은 너무 많은 가비지 콜렉션 오버헤드를 유발할 수 있습니다. 큰 값은 가비지 콜렉션을 제한하고 메모리 부족 오류를 유발할 수 있습니다. 그러나 대부분의 어플리케이션에 있어서 디폴트 값은 맞습니다.

가비지 콜렉션은 해당 오브젝트에 대해 유효한 참조가 있는지를 평가하여 오브젝트가 더 이상 필요없는지를 판별합니다.

Java 원시 메소드 호출 성능 고려사항

iSeries 서버에서 원시 메소드 호출은 다른 플랫폼의 원시 메소드 호출도 수행하지 않습니다. iSeries 서버에서 Java(TM)는 기계 인터페이스(MI) 하에서 JVM(Java Virtual Machine)을 이동시켜 최적화됩니다. 원시 메소드 호출은 위의 MI 코드에 대한 호출이 필요하며 다시 JNI(Java Native Interface) 호출이 필요할 수 있습니다. 원시 메소드로는 작은 루틴을 수행하지 않아야 하며, 작은 루틴은 Java로 쉽게 작성할 수 있습니다. 원시 메소드만을 사용하여 상대적으로 오래 실행하고 Java에서 직접 사용할 수 없는 시스템 기능을 시작하십시오.

Java 메소드 인라이닝 성능 고려사항

메소드 인라인닝은 메소드 호출 성능을 상당히 개선할 수 있습니다. 모든 최종 메소드는 인라인닝을 위한 잠재적인 후보입니다. 인라인 피처는 컴파일시에 `javac -o` 옵션을 통해 iSeries 서버에서 사용할 수 있습니다. `javac -o` 옵션을 사용하는 경우 클래스 파일과 변환된 Java(TM) 프로그램 *크기*가 증가합니다. `-o` 옵션을 사용할 때 어플리케이션의 공간과 성능 등록 정보를 둘다 고려해야 합니다.

Java 변환 프로그램은 최적화 레벨 30 및 최적화 레벨 40에 대한 인라인닝을 작동할 수 있게 합니다. 최적화 레벨 30은 단일 클래스내에서 최종 메소드의 일부 인라인닝을 작동할 수 있게 합니다. 최적화 레벨 40은 ZIP 파일이나 JAR 파일내에서 최종 메소드의 인라인닝을 작동할 수 있게 합니다. AllowInlining 및 NoAllowInlining LICOPT 매개변수 스트링으로 메소드 인라인닝을 제어할 수 있습니다. iSeries 인터프리터는 메소드 인라이닝을 수행하지 않습니다.

Java 예외 성능 고려사항

iSeries 예외 구조는 변하기 쉬운 인터럽트 및 재시도 기능을 허용합니다. 이것은 혼합 언어 상호작용을 허용합니다. iSeries 서버에서 Java™ 예외를 발생시키면 다른 플랫폼에서 보다 더 많은 비용이 듭니다. 이것은 Java 예외가 정상 어플리케이션 경로에서 일상적으로 사용되지 않는 한 전체 어플리케이션 성능에 영향을 주지 않는 것이어야 합니다.

Java 호출 추적 성능 분석 툴

Java(TM) 메소드 호출 추적은 각 Java 메소드에서 소요된 시간에 대한 중요한 성능 정보를 제공합니다. 다른 JVM(Java Virtual Machine)에서는 `java` 명령에 -prof(프로파일) 옵션을 사용했을 것입니다. iSeries 서버에서 메소드 호출 추적을 작동시키려면 CRTJVAPGM(Java 프로그램 작성) 명령행에 ENBPFRCOL(성능 콜렉션 작동 가능) 명령을 반드시 지정해야 합니다. 이 키워드로 Java 프로그램을 작성한 후 호출/리턴 추적 유형을 포함하는 PEX(성능 탐색기) 정의를 사용하여 메소드 호출 추적 콜렉션을 시작할 수 있습니다.

PRTPEXRPT(성능 탐색기 보고서 인쇄) 명령으로 생성되는 호출/리턴 추적 결과는 추적되는 모든 Java 메소드에 대한 각 호출당 CPU 시간을 보여줍니다. 어떤 경우에는 호출 리턴 추적을 위한 모든 클래스 파일을 작동시키지 않을 수 있습니다. 또한 추적할 수 없는 원시 메소드와 시스템 기능을 호출하는 중일 수 있습니다. 이 경우 이 메소드나 시스템 기능에서 소비되는 모든 CPU 시간이 합산됩니다. 그 후 호출되어 작동이 가능한 마지막 Java 메소드에 시간이 보고됩니다.

Java 이벤트 추적 성능 분석 툴

iSeries JVM(Java(TM) virtual machine)을 사용하면 특정 Java 이벤트를 추적할 수 있습니다. 이러한 이벤트는 Java 코드의 다른 툴 없이 수집할 수 있습니다. 이러한 이벤트에는 가비지 콜렉션, 스레드 작성, 클래스 로드 및 잠금과 같은 활동이 있습니다. RUNJVA(Java 실행) 명령은 이들 이벤트를 지정하지 않습니다. 대신에, PEX(성능 탐색기) 정의를 작성하고 STRPEX(성능 탐색기 시작) 명령을 사용하여 이벤트를 수집합니다. 각 이벤트는 시간소인(time stamp) 및 중앙 처리 장치(CPU) 주기와 같은 유용한 성능 정보를 포함합니다. 동일한 추적 정의를 사용하여 Java 이벤트와 디스크 입력 및 출력과 같은 다른 시스템 활동을 둘다 추적할 수 있습니다.

Java 이벤트의 설명은 Performance Tools for iSeries, SC41-5340 책을 참조하십시오.

Java 프로파일 성능 분석 툴

시스템 범용 CPU 프로파일링에서는 사용자의 Java 프로그램에 사용되는 각 Java(TM) 메소드와 모든 시스템 기능에 소요된 상대적 CPU 시간을 계산합니다. 성능 모니터 카운터 넘침(*PMCO) 실행 주기 이벤트를 추적하는 PEX(성능 탐색기) 정의를 사용하십시오. 일반적으로 1밀리초의 간격으로 지정됩니다. 유효한 추적 프로파일을 수집하기 위해서 2 - 3분의 CPU 시간을 누적할 때까지 Java 어플리케이션을 실행해야 합니다. 이것은 100,000개 이상의 샘플을 생성합니다. PRTPEXRPT(성능 탐색기 보고서 인쇄) 명령이 전체 어플리케이션에 걸쳐서 소비되는 CPU 시간의 히스토그램을 작성합니다. 이것은 모든 Java 메소드와 모든 시스템 수준 활동을 포함합니다. PDC(Performance Data Collector) 툴은 iSeries 서버에서 실행되는 프로그램에 대한 프로파일 정보도 제공합니다.

주: CPU 프로파일링은 해석되는 Java 프로그램의 상대적인 CPU 사용률은 표시하지 않습니다.

JVMPI(Java Virtual Machine Profiler Interface)

»

JVMPI(Java(TM) Virtual Machine Profiler Interface)는 Sun의 Java 2 SDK, 표준판 버전 1.2에서 발표되고 구현된 JVM(Java Virtual Machine)을 프로파일링하기 위한 실험적인 인터페이스입니다.

JVMPI 지원은 JVM 및 JIT(Just-in-time) 컴파일러에 후크를 넣어서 활성화되면 프로파일 에이전트에 이벤트 정보를 제공합니다. 프로파일 에이전트는 DLL(dynamic link library)로 구현됩니다. 프로파일러는 JVMPI 이벤트의 작동 및 작동 불가능을 위해 JVM에 제어 정보를 보냅니다. 예를 들어, 프로파일러는 메소드 Entry 또는 Exit 후크에 관심이 없을 수 있으며 이러한 이벤트 통지를 받지 않을 것임을 JVM에 알릴 수 있습니다.

JVM 및 JIT에는 이벤트를 작동할 수 있는 경우에 프로파일 에이전트에 이벤트 통지를 보내는 JVMPI 이벤트 후크가 내장되어 있습니다. 프로파일러는 관심이 있는 이벤트를 JVM에 알리고 JVM은 이벤트 발생 시에 이벤트에 대한 통지를 프로파일러에 보냅니다.

자세한 정보는 Sun Microsystems, Inc.의 JVMPI  를 참조하십시오.

Java 성능 자료 수집

iSeries 서버에서 Java(TM) 성능 자료를 수집하려면, 다음 단계를 수행하십시오.

1. 다음 사항을 지정하는 PEX(성능 탐색기) 정의를 작성하십시오.
   - 사용자 정의명
   - 자료 콜렉션 유형
   - 작업명
   - 시스템 정보를 수집하려는 시스템 이벤트 시리즈

   주: PEX 정의 *STATS는 `java_g -prof` 유형을 출력하고 Java 프로그램의 특정 작업명을 알고 있는 경우에 *TRACE 정의보다 좋습니다.

   ```
   ADDPEXDFN DFN(YOURDFN) JOB(*ALL/YOURID/QJVACMDSRV)
   DTAORG(*HIER) TEXT('your stats defination')
   ```

   위는 하나의 *STATS 정의 예로서 이것이 모든 Java 이벤트를 실행시키지는 않습니다. 사용자의 Java 세션에 있는 Java 이벤트만 프로파일 처리합니다. 이 작업 모드는 Java 프로그램을 실행하는 데 걸리는 시간을 증가시킬 수 있습니다.

   ```
   ADDPEXDFN DFN(YOURDFN)
   TYPE(*TRACE) JOB(*ALL) TRCTYPE(*SLTEVT) SLTEVT(*YES)
   PGMEVT(*JVAENTRY *JVAEXIT)
   ```

   위는 하나의 *TRACE 정의 예로서 ENBPFRCOL(*ENTRYEXIT)으로 컴파일한 시스템의 Java 프로그램에서 Java 입력 이벤트 및 나감 이벤트를 작성합니다. 이로 인해 사용자의 Java 프로그램 이벤트 수와 PEX 자료 콜렉션 기간에 따라서는 콜렉션 유형의 분석이 *STATS 추적보다 느릴 수 있습니다.
2. PEX 정의의 프로그램 이벤트 범주에 있어서 *JVAENTRY 및 *JVAEXIT를 작동시키면 PEX는 Java 입력과 나감을 인식합니다.

   주: JIT(Just-in-time) 컴파일러를 사용하여 Java 코드를 실행 중이면 직접 처리를 위해 CRTJVAPGM 명령을 사용하던 때처럼 입력 및 나감을 처리할 수 없습니다. 대신, os400.enbprfcol 시스템 등록 정보를 사용할 때 JIT가 enter와 exit 훅을 가진 코드를 생성합니다.
3. Java 프로그램을 준비하여 iSeries 성능 자료 콜렉터에 대한 프로그램 이벤트를 보고하십시오. 성능 자료를 보고하려는 Java 프로그램에서 CRTJVAPGM(Java 프로그램 작성) 명령을 사용하여 수행할 수 있습니다. 반드시 ENBPFRCOL(*ENTRYEXIT) 매개변수를 사용하여 Java 프로그램을 작성해야 합니다.

주: 성능 자료를 수집하려는 모든 Java 프로그램에서 이 단계를 반복해야 합니다. 이 단계를 수행하지 않으면 성능 자료가 PEX에 의해 수집되지 않으며 JPDC(Java Performance Data Converter)툴을 실행하여 생성되지 않습니다.

4. STRPEX(성능 탐색기 시작) 명령을 사용하여 PEX 자료 콜렉션을 시작하십시오.
5. 분석하려는 프로그램을 실행하십시오. 이 프로그램이 실행(production) 환경에 있어서는 안됩니다. 이와 같이 하면 적은 시간에 많은 자료를 생성합니다. 콜렉션 시간은 5분으로 제한하십시오. 이 시간 동안 실행되는 Java 프로그램이 많은 양의 PEX 시스템 자료를 생성합니다. 너무 많은 자료가 수집되면 자료를 처리하는데 필요 이상의 많은 시간이 요구됩니다.
6. ENDPEX(성능 탐색기 종료) 명령을 사용하여 PEX 자료 콜렉션을 종료하십시오.

   주: PEX 자료 콜렉션을 처음 종료하는 것이 아닌 경우 대체 파일을 *YES로 지정하지 않으면 자료를 저장하지 않습니다.
7. JPDC 툴을 실행하십시오.
8. 통합 파일 시스템 디렉토리를 java_g -prof 표시기 또는 Jinsight 표시기와 같은 사용자가 선택한 표시기가 있는 시스템에 연결하십시오. iSeries 서버로부터 이 파일을 복사하여 적합한 프로파일 툴에 입력하여 사용할 수 있습니다.

성능 자료 콜렉터 툴

PDC(Performance Data Collector) 툴은 iSeries 서버에서 실행되는 프로그램에 대한 프로파일 정보를 제공합니다.

많은 JVM(Java(TM) virtual machine)의 업계 표준 프로파일 옵션은 `java_g` 피처의 구현에 따라 다릅니다. 이 버전은 JVM(Java Virtual Machine)의 고유 디버그 버전으로 `-prof` 옵션을 제공합니다. Java 프로그램에 호출할 때 이 옵션을 지정합니다. 이 옵션을 지정할 경우 JVM(Java Virtual Machine)은 Java 프로그램이 설정되는 동안 프로그램이 운영되었던 부분에 대한 정보가 있는 레코드 파일을 생성합니다. JVM(Java Virtual Machine)은 이 정보를 실시간으로 생성합니다.

iSeries 서버에서, PEX(Performance Explorer) 피처는 프로그램 및 레코드 특정 시스템 이벤트를 분석합니다. DB2(R) 데이터베이스는 이 정보를 저장하고, SQL 기능을 사용하여 검색합니다. PEX 정보는 Java 프로파일 자료를 생성하는 특정 프로그램 정보에 대한 저장소입니다. 이 프로파일 자료는 `java_g -prof` 프로그램 프로파일 정보와 호환됩니다. JPDC(Java Performance Data Converter) 툴은 `java_g -prof` 프로그램 출력 및 Jinsight라고 하는 특정 IBM 툴에 대한 프로그램 프로파일 정보를 제공합니다.

Java 성능 자료 수집에 대한 정보는 Java 성능 자료 수집을 참조하십시오.

JPDC(Java Performance Data Converter) 툴

JDPC(Java(TM) Performance Data Converter) 툴에서는 iSeries 서버에서 실행 중인 Java 프로그램에 대한 Java 성능 자료를 작성할 수 있습니다. 이 성능 자료는 Sun Microsystems, Inc.의 JVM(Java Virtual Machine) `java_g -prof` 옵션 및 IBM Jinsight 출력의 성능 자료 출력과 호환됩니다.

주: JDPC 툴은 읽기 가능한 출력을 생성하지 않습니다. 자료를 분석하려면 `java_g -prof` 또는 Jinsight 자료를 허용하는 Java 프로그램 툴을 사용하십시오.

JDPC 툴은 DB2/400(JDBC 사용)이 저장하는 iSeries PEX(성능 탐색기) 자료에 액세스 합니다. 자료를 Jinsight 또는 일반 성능 유형으로 변환합니다. 그런 다음, JDPC는 사용자 특정 위치에서 통합 파일 시스템에 출력 파일을 저장합니다.

주: iSeries 서버에서 지정된 Java 어플리케이션을 실행하는 동안 PEX 자료를 수집하기 위해 적합한 iSeries PEX 자료 수집 절차를 따라야 합니다. 시작 및 나감 프로그램 또는 수집 및 저장 프로시듀어를 정의하는 PEX 정의를 설정해야 합니다. PEX 자료를 수집하고 PEX 정의를 설정하는 방법에 대한 자세한 내용은 Performance Tools for iSeries, SC41-5340 을 참조하십시오.

JPDC 수행 방법에 대한 자세한 내용은 JPDC(Java Performance Data Converter) 실행을 참조하십시오.

Qshell 명령 행 인터페이스 또는 RUNJVA(Java 실행) 명령을 사용하여 JPDC 프로그램을 시작할 수 있습니다.

JPDC(Java Performance Data Converter) 실행

성능 분석 자료 콜렉션을 위한 JPDC(Java(TM) Performance Data Converter)를 실행하려면 다음 단계를 수행하십시오.

1. 첫 번째 입력 인수로 `java_g -prof`에 해당하는 `general` 또는 Jinsight 출력에 해당하는 `jinsight`를 입력하십시오.
2. 두 번째 입력 인수인 자료를 수집하는 데 사용했던 PEX(성능 탐색기) 정의명을 입력하십시오.

   주: 정의명이 연결되어 내부에서 사용되므로 이름을 4 - 5자로 제한해야 합니다.
3. 세 번째 입력 인수인 JPDC 툴이 생성하는 파일명을 입력하십시오. 생성되는 파일은 현재 통합 파일 시스템 디렉토리에 기록됩니다. 통합 파일 시스템 현재 디렉토리를 지정하려면 cd(PF4) 명령을 사용하십시오.
4. iSeries 호스트 관계형 데이터베이스 디렉토리 항목의 이름인 네 번째 입력 인수를 입력하십시오. 항목명이 무엇인지 보려면 WRKRDBDIRE(관계형 데이터베이스 디렉토리 항목에 대한 작업) 명령을 사용하십시오. 이 데이터베이스는 *LOCAL이 표시되어 있는 유일한 관계형 데이터베이스입니다.

이 코드를 작동시키려면 /QIBM/ProdData/Java400/ext/JPDC.jar 파일이 iSeries 서버의 Java classpath에 있어야 합니다. 프로그램이 실행될 때 현재 디렉토리에서 텍스트 출력 파일을 찾을 수 있습니다.

iSeries 명령행이나 Qshell 환경을 사용하여 JPDC를 실행할 수 있습니다. 자세한 내용은 예: JPDC(Java Performance Data Converter) 실행을 참조하십시오.

예: JPDC(Java Performance Data Converter) 실행

iSeries 명령행이나 Qshell 환경을 사용하여 JPDC(Java(TM) Performance Data Converter)를 실행할 수 있습니다.

iSeries 명령행 사용:

1. iSeries 명령행에 RUNJVA(Java 실행) 명령이나 JAVA 명령을 입력하십시오.
2. 클래스 매개변수 행에 `com.ibm.as400.jpdc.JPDC`를 입력하십시오.
3. 매개변수 행에 `general pexdfn mydir/myfile myrdbdire`를 입력하십시오.
4. classpath 매개변수 행에 `'/QIBM/ProdData/Java400/ext/JPDC.jar'`를 입력하십시오.

   주: `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 스트링이 CLASSPATH 환경 변수에 있는 경우 classpath를 생략할 수 있습니다. ADDENVVAR(환경 변수 추가) 명령, CHGENVVAR(환경 변수 변경) 명령 또는 WRKENVVAR(환경 변수에 대한 작업) 명령 중 하나를 사용하여 이 스트링을 CLASSPATH 환경 변수에 추가할 수 있습니다.

Qshell 환경 사용:

1. STRQSH(Qshell 시작) 명령을 입력하여 Qshell 인터프리터를 시작하십시오.
2. 명령 행에 다음을 입력하십시오.

   ```
   java -classpath /QIBM/ProdData/Java400/ext/JPDC.jar com.ibm.as400/jpdc/JPDC
   jinsight pexdfn mydir/myfile myrdbdire
   ```

   주: `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 스트링이 현재 환경에 추가된 경우 classpath를 생략할 수 있습니다. ADDENVVAR 명령, CHGENVVAR 또는 WRKENVVAR 명령을 사용하여 해당 스트링을 사용자의 현재 환경에 추가할 수 있습니다.

자세한 내용은 JPDC(Java Performance Data Converter) 실행을 참조하십시오.

제 6 장 IBM Developer Kit for Java의 명령과 툴

IBM Developer Kit for Java(TM)를 사용 중인 경우, Qshell 인터프리터나 CL 명령과 함께 Java 툴을 사용할 수 있습니다.

Java 프로그래밍 경험이 있다면 툴이 Sun Microsystems, Inc.의 Java Development Kit을 사용하는 툴과 유사하기 때문에 Qshell 인터프리터 Java 툴을 사용하는 것이 훨씬 편리할 수 있습니다. Qshell 환경 사용에 대한 내용은 Qshell 인터프리터를 참조하십시오.

iSeries 프로그래머의 경우 iSeries 서버 환경에 일반적인 Java용 명령을 사용할 수도 있습니다. CL 명령 및 iSeries Navigator 명령 사용에 대한 자세한 내용을 참조하십시오.

IBM Developer Kit for Java에서 다음 명령과 툴을 사용할 수 있습니다.

- Qshell 환경에는 프로그램 개발에 일반적으로 요구되는 Java 개발 툴이 포함됩니다.
- CL 환경에는 Java 프로그램을 최적화하고 관리하기 위한 CL 명령이 포함됩니다.
- iSeries Navigator 명령 또한 최적화된 Java 프로그램을 작성하고 실행합니다.

IBM Developer Kit for Java가 지원하지 않는 Java 툴

IBM Developer Kit for Java(TM)는 다음 툴을 지원합니다.

- 457 페이지의『Java rmic 툴』
- 458 페이지의『Java rmid 툴』
- 458 페이지의『Java rmiregistry 툴』
- 458 페이지의『Java serialver 툴』
- 458 페이지의『Java tnameserv 툴』

몇 가지 예외의 경우 ajar 툴을 제외한 Java 툴은 Sun Microsystems Inc.가 문서화한 구문과 옵션을 지원합니다. Java 툴은 모두 Qshell 인터프리터를 사용하여 실행해야 합니다.

STRQSH 또는 QSH(Qshell 시작) 명령을 사용해서 Qshell 인터프리터를 시작할 수 있습니다. Qshell 인터프리터가 실행 중일 때 QSH 명령 입력 화면이 나타납니다. Qshell에서 실행 중인 Java 툴과 프로그램의 모든 출력과 메세지가 화면에 나타납니다. 또는 모든 Java 프로그램에 대한 입력은 이 화면에서 읽힙니다. 자세한 내용은 Qshell의 Java명령을 참조하십시오.

주: iSeries 명령 입력 기능을 Qshell 내에서 직접 사용할 수 없습니다. 명령을 입력하려면 F21(CL 명령 입력) 키를 누르십시오.

Java 툴

- 『Java ajar 툴』
- 453 페이지의『Java appletviewer 툴』
  - 453 페이지의『Remote Abstract Window Toolkit으로 Java appletviewer 툴 실행』
- 453 페이지의『Java extcheck 툴』
- 453 페이지의『Java idlj 툴』
- 454 페이지의『Java jar 툴』
- 454 페이지의『Java jarsigner 툴』
- 454 페이지의『Java javac 툴』
- 455 페이지의『Java javadoc 툴』

Java ajar 툴

ajar 툴은 Java(TM) ARchive(JAR) 파일을 작성하고 조작하는 데 사용하는 jar 툴의 대체 인터페이스입니다. ajar 툴을 사용하여 JAR 파일과 ZIP 파일을 조작할 수 있습니다.

ZIP 인터페이스나 UNZIP 인터페이스가 필요하면 jar 툴 대신 ajar 툴을 사용하십시오.

ajar 툴은 jar 툴과 마찬가지로 JAR 파일의 내용을 나열하고, JAR 파일을 추출하고, 새로운 JAR 파일을 작성하며, 여러 ZIP 형식을 지원합니다. 또한 ajar 툴은 기존 JAR 파일에 파일을 추가하고 삭제할 수 있습니다.

Qshell 인터프리터를 사용하면 ajar 툴을 사용할 수 있습니다. 자세한 정보는 ajar - 대체 Java 아카이브를 참조하십시오.

Java appletviewer 툴

appletviewer 툴을 사용하면 웹 브라우저가 없어도 애플릿을 실행할 수 있습니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 appletviewer 툴과 호환됩니다.

Qshell 인터프리터를 사용하여 appletviewer 툴을 사용할 수 있습니다. appletviewer 툴을 실행하려면 Remote Abstract Window Toolkit을 사용해야 합니다. appletviewer 툴을 사용하기 위해 Remote AWT를 설정하는 방법에 대한 정보는 『Remote Abstract Window Toolkit으로 Java appletviewer 툴 실행』을 참조하십시오.

appletviewer 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 appletviewer 툴을 참조하십시오.

Remote Abstract Window Toolkit으로 Java appletviewer 툴 실행: Java appletviewer 툴을 사용하기 위해서는, WIndows(R) 리모트 표시장치에서 Remote Abstract Window Toolkit for Java를 설정하거나, Qshell 인터프리터에서 sun.applet.AppletViewer 클래스를 사용하거나 appletviewer 툴을 Remote AWT 등록 정보와 함께 실행해야 합니다.

예를 들어, sun.applet.AppletViewer 클래스를 사용하여 TicTacToe 디렉토리의 example1.html을 실행하는 경우 명령행은 다음과 유사합니다.

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html') CLASSPATH('/TicTacToe')
PROP((RmtAwtServer '1.1.11.11') (os400.class.path.rawt 1)(java version 1.3))
```

Qshell 인터프리터에서 appletviewer 툴을 사용하고 TicTacToe 디렉토리의 example1.html을 실행하는 경우 명령은 다음과 유사합니다.

```
qsh "enter"
cd TicTacToe "enter"
Appletviewer -J-DRmtAwtServer=1.1.11.11 -J-Dos400.class.path.rawt=1 -J-Djava.version=1.3
example1.html
```

주: -J는 Appletviewer에 대한 런타임 플래그입니다. -D는 등록 정보입니다.

Java extcheck 툴

Java 2 SDK(J2SDK), 표준판, 버전 1.2 이상에서는 extcheck 툴이 목표 JAR 파일과 현재 설치된 확장 JAR 파일 사이의 버전 충돌을 감지합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 keytool 툴과 호환됩니다.

Qshell 인터프리터를 사용하면 extcheck 툴을 사용할 수 있습니다.

extcheck 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 extcheck 툴을 참조하십시오.

Java idlj 툴

idlj 툴은 제공된 IDL(Interface Definition Language) 파일에서 Java 바인딩을 생성합니다. idlj 툴은 IDL 대 Java 컴파일러라고도 합니다. 이 툴은 JDK(Java Development Kit) 버전 1.3 및 1.4에서 Sun Microsystems 사가 제공하는 idlj 툴과 호환됩니다.

idlj 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 appletviewer 툴을 참조하십시오.

Java jar 툴

jar 툴은 복수 파일을 단일 JAR 파일로 결합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 jar 툴과 호환됩니다.

Qshell 인터프리터를 사용하여 jar 툴을 사용할 수 있습니다.

JAR 파일을 작성하고 조작하기 위한 jar 툴에 대한 대체 인터페이스는 452 페이지의 『Java ajar 툴』을 참조하십시오.

iSeries 400 파일 시스템에 대한 자세한 정보는 통합 파일 시스템 또는 통합 파일 시스템의 파일을 참조하십시오.

jar 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 jar 툴을 참조하십시오.

Java jarsigner 툴

Java 2 SDK(J2SDK), 표준판, 버전 1.2 이상에서, jarsigner 툴은 JAR 파일을 서명하고 서명된 JAR 파일의 서명을 확인합니다. jarsigner 툴은 JAR 파일을 서명하기 위한 개인용 키를 찾아야 할 때 keytool이 작성하고 관리하는 키저장소에 액세스합니다. J2SDK에서, jarsigner 및 keytool 툴이 javakey 툴을 대체합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 jarsigner 툴과 호환됩니다.

Qshell 인터프리터를 사용하여 jarsigner 툴을 사용할 수 있습니다.

jarsigner 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 jarsigner 툴을 참조하십시오.

Java javac 툴

javac 툴은 Java 프로그램을 컴파일합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 javac 툴과 호환되지만 하나의 예외가 있습니다.

-classpath

디폴트 classpath를 대체하지 마십시오. 대신에, 이 옵션은 시스템 디폴트 classpath에 추가됩니다. -classpath 옵션은 CLASSPATH 환경 변수를 대체합니다.

Qshell 인터프리터를 사용하여 javac 툴을 사용할 수 있습니다.

iSeries 서버에 디폴트로 JDK 1.1.x를 설치했지만 버전 1.2 이상에서 java 명령을 실행해야 하는 경우 다음의 명령을 입력하십시오.

```
javac -djava.version=1.2 <my_dir> MyProgram.java
```

javac 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 javac 툴을 참조하십시오.

Java javadoc 툴

javadoc 툴은 API 문서를 생성합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 javadoc 툴과 호환됩니다.

Qshell 인터프리터를 사용하여 javadoc 툴을 사용할 수 있습니다.

javadoc 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 javadoc 툴을 참조하십시오.

Java 툴

- 『Java javah 툴』
- 456 페이지의 『Java javakey 툴』
- 456 페이지의 『Java javap 툴』
- 457 페이지의 『Java keytool』
- 457 페이지의 『Java native2ascii 툴』
- 457 페이지의 『Java policytool』
- 457 페이지의 『Java rmic 툴』
- 458 페이지의 『Java rmid 툴』
- 458 페이지의 『Java rmiregistry 툴』
- 458 페이지의 『Java serialver 툴』
- 458 페이지의 『Java tnameserv 툴』

Java javah 툴

javah 툴은 Java(TM) 원시 메소드의 구현을 용이하게 합니다. 이 툴은 몇 가지 예외를 제외하고, Sun Microsystems, Inc.가 제공하는 javah 툴과 호환됩니다.

주: 원시 메소드 쓰기는 어플리케이션이 100% 순수한 Java가 아님을 의미합니다. 또한 어플리케이션이 플랫폼 사이에서 직접 이동할 수 없음을 의미합니다. 원시 메소드는 본질적으로 플랫폼이나 시스템에 따라 다릅니다. 원시 메소드를 사용하면 어플리케이션에 대한 개발 및 유지보수 비용이 증가할 수 있습니다.

Qshell 인터프리터를 사용하여 javah 툴을 사용할 수 있습니다. 이 툴은 Java 클래스 파일을 읽고 현재 작업 디렉토리에 C 언어 헤더 파일을 작성합니다. 작성된 헤더 파일은 iSeries 스트림 파일(STMF)입니다. iSeries 서버의 C 프로그램에 포함되기 전에 파일 멤버에 복사해야 합니다.

javah 툴은 Sun Microsystems, Inc.에서 제공하는 툴과 호환되지만 다음의 옵션이 지정된 경우 iSeries 서버는 이를 무시합니다.

-td iSeries 서버의 javah 툴에는 임시 디렉토리가 필요하지 않습니다.

-stubs iSeries 서버의 Java는 원시 메소드의 JNI(Java 원시 인터페이스) 양식만을 지원합니다. 스터브(stub)는 원시 메소드의 이전 JNI 양식에만 필요합니다.

-trace

iSeries 서버의 Java가 지원하지 않는 .c 스터브 파일 출력과 관련됩니다.

-v 지원되지 않습니다.

주: -jni 옵션은 항상 지정해야 합니다. iSeries 서버 시스템은 JNI 이전의 원시 메소드 구현을 지원하지 않습니다.

javah 툴에 대한 자세한 내용은 Sun Microsystems, Inc.의 javah 툴을 참조하십시오.

Java javakey 툴

애플릿에 대한 디지털 서명 생성을 포함하여 암호화 키, 인증 작성 및 관리를 위해 javakey 툴을 사용하십시오. 이 툴은 JDK(Java Development Kit) 버전 1.1.x에서 Sun Microsystems사가 제공하는 javakey 툴과 호환됩니다.

J2SDK(Java 2 Software Development Kit), 표준판, 버전 1.2 이상에서는 javakey 툴이 사용되지 않습니다. JDK 버전 1.1.x의 결함으로 인해 1.1.x javakey 툴을 사용하여 서명된 코드는 J2SDK, 버전 1.2 이상에서 서명되지 않은 것으로 인식됩니다. J2SDK, 버전 1.2 이상을 사용하여 코드에 서명하는 경우 JDK 1.1.x 버전에서 서명되지 않은 것으로 인식됩니다.

주: iSeries SSL(보안 소켓층) 지원은 이 툴에 의해 작성된 키에 액세스할 수 없습니다. 대신, 인증과 iSeries 서버에 통합되고 DCM(Digital Certificate Manager)으로 작성되거나 가져온 키 컨테이너를 사용해야 합니다. 자세한 내용은 보안 소켓층으로 Java 어플리케이션 보안을 참조하십시오.

애플릿 패키지 및 애플릿 서명은 브라우저에 따라 다릅니다. 브라우저 문서를 검사하여 Java JAR 파일 형식 및 javakey 애플릿 서명으로 호환되는지 확인하십시오.

주: javakey 툴로 작성되는 파일은 민감한 정보를 포함합니다. 적절한 통합 파일 시스템 보안 수단이 공용 및 개인 키 파일을 보호합니다.

Qshell 인터프리터를 사용하여 javakey 툴을 사용할 수 있습니다.

iSeries 400 파일 시스템에 대한 자세한 정보는 통합 파일 시스템 또는 통합 파일 시스템의 파일을 참조하십시오.

javakey 툴에 대한 자세한 내용은 javakey tool by Sun Microsystems, Inc.  를 참조하십시오.

Java javap 툴

javap 툴은 컴파일된 Java 파일을 역어셈블하여 Java 프로그램 표현을 인쇄 출력합니다. 이것은 시스템에서 소스 원본 코드를 더 이상 사용할 수 없을 때 유용합니다.

이 툴은 일부 예외를 제외하고 Sun Microsystems, Inc.가 제공하는 javap 툴과 호환됩니다.

-b 이 옵션은 무시됩니다. iSeries 서버에서 JDK(Java는 Java Development Kit) 1.1.4 이상만을 지원하므로 역방향 호환성은 필요 없습니다.

-p iSeries 서버에서 -p는 유효한 옵션이 아닙니다. -private로 입력해야 합니다.

-verify

이 옵션은 무시됩니다. javap 툴은 iSeries 서버에서 검증을 수행하지 않습니다.

Qshell 인터프리터를 사용하여 javap 툴을 사용할 수 있습니다.

주: 클래스를 역어셈블하기 위해 javap 툴을 사용하면 클래스에 대한 사용권 계약을 위반할 수 있습니다. javap 툴을 사용하기 전에 클래스에 대한 사용권 계약에 대하여 문의하십시오.

javap 툴에 대한 자세한 내용은 Sun Microsystems, Inc.의 javap 툴을 참조하십시오.

Java keytool

Java 2 SDK(J2SDK), 표준판, 버전 1.2 이상에서, keytool은 공용 및 개인용 키 쌍과 자체 서명된 인증을 작성하고 키저장소를 관리합니다. J2SDK에서, jarsigner 및 keytool 툴이 javakey 툴을 대체합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 keytool 툴과 호환됩니다.

Qshell 인터프리터를 사용하면 keytool을 사용할 수 있습니다.

keytool에 대한 자세한 정보는 Sun Microsystems, Inc.의 keytool을 참조하십시오.

Java native2ascii 툴

native2ascii 툴은 원시 코드화 문자(비Latin 1 및 비유니코드인 문자)가 있는 파일을 유니코드 코드화 문자의 파일로 변환합니다. 이 툴은 Sun Microsystems Inc.가 제공하는 native2ascii 툴과 호환됩니다.

Qshell 인터프리터를 통해 native2ascii 툴을 사용할 수 있습니다.

native2ascii 툴에 대한 자세한 내용은 Sun Microsystems, Inc.의 native2ascii 툴을 참조하십시오.

Java policytool

Java 2 SDK, 표준판에서, policytool은 사용자 설치의 Java 보안 정책을 정의하는 외부 정책 구성 파일을 작성하고 변경합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 policytool과 호환됩니다.

policytool은 Qshell 인터프리터와 Remote AWT(Remote Abstract Window Toolkit)로 사용할 수 있는 그래픽 사용자 인터페이스(GUI) 툴입니다. 자세한 정보는 IBM Developer Kit for Java Remote AWT(Abstract Window Toolkit)를 참조하십시오.

policytool에 대한 자세한 정보는 Sun Microsystems, Inc.의 policytool을 참조하십시오.

Java rmic 툴

rmic 툴은 Java 오브젝트용 스터브(stub) 파일과 클래스 파일을 생성합니다. 이 툴은 Sun Microsystems Inc.가 제공하는 rmic 툴과 호환됩니다.

Qshell 인터프리터를 통해 rmic 툴을 사용할 수 있습니다.

rmic 툴에 대한 자세한 내용은 Sun Microsystems Inc.의 rmic 툴을 참조하십시오.

Java rmid 툴

Java 2 SDK(J2SDK), 표준판에서, rmid 툴은 활성화 시스템 디먼을 시작하므로 JVM(Java Virtual Machine)에서 오브젝트를 등록하고 활성화할 수 있습니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 rmid 툴과 호환됩니다.

Qshell 인터프리터를 통해 rmid 툴을 사용할 수 있습니다.

rmid 툴에 대한 자세한 정보는 Sun Microsystems, Inc.의 rmid 툴을 참조하십시오.

Java rmiregistry 툴

rmiregistry 툴은 지정된 포트에서 리모트 오브젝트 등록을 시작합니다. 이 툴은 Sun Microsystems Inc.가 제공하는 rmiregistry 툴과 호환됩니다.

Qshell 인터프리터를 통해 rmiregistry 툴을 사용할 수 있습니다.

rmiregistry 툴에 대한 자세한 내용은 Sun Microsystems Inc.의 rmiregistry 툴을 참조하십시오.

Java serialver 툴

serialver 툴은 하나 이상의 클래스에 대하여 버전 번호나 고유 일련 ID를 리턴합니다. 이 툴은 Sun Microsystems Inc.가 제공하는 serialver 툴과 호환됩니다.

Qshell 인터프리터를 통하여 serialver 툴을 사용할 수 있습니다.

serialver 툴에 대한 자세한 내용은 Sun Microsystems Inc.의 serialver 툴을 참조하십시오.

Java tnameserv 툴

Java 2 SDK(J2SDK), 표준판, 버전 1.3에서

또는 높은

tnameserv(Transient Naming Service) 툴은 명명 서비스에 대한 액세스를 제공합니다. 이 툴은 Sun Microsystems, Inc.가 제공하는 tnameserv 툴과 호환됩니다.

tnameserv 툴은 Qshell 인터프리터를 통해 사용할 수 있습니다.

Qshell의 Java 명령

Qshell에서 java 명령은 Java(TM) 프로그램을 실행합니다. 일부를 제외하고는 Sun Microsystems, Inc.가 제공하는 java 툴과 호환됩니다.

IBM Developer Kit for Java는 Qshell에 있는 java 명령의 다음 옵션을 무시합니다.

옵션	설명
-cs	이 옵션은 지원 되지 않습니다.
-checksource	이 옵션은 지원 되지 않습니다.
-debug	이 옵션은 iSeries 내부 디버거에 의해 지원됩니다.
-noasyncgc	가비지 콜렉션은 IBM Developer Kit for Java와 함께 항상 실행됩니다.
-noclassgc	가비지 콜렉션은 IBM Developer Kit for Java와 함께 항상 실행됩니다.
-prof	iSeries 서버에는 자체의 성능 분석 툴이 있습니다.
-ss	이 옵션은 iSeries 서버에는 해당되지 않습니다.
-oss	이 옵션은 iSeries 서버에는 해당되지 않습니다.
-t	iSeries 서버는 자체의 추적 기능을 사용합니다.
-verify	항상 iSeries 서버에서 확인합니다.
-verifyremote	항상 iSeries 서버에서 확인합니다.
-noverify	항상 iSeries 서버에서 확인합니다.

iSeries 서버에서, `-classpath` 옵션은 디폴트 classpath를 대체하지 않습니다. 대신에, 이 옵션은 시스템 디폴트 classpath에 추가됩니다. `-classpath` 옵션은 CLASSPATH 환경 변수를 대체합니다.

Qshell의 `java` 명령은 iSeries 서버의 새로운 옵션을 지원합니다. 지원되는 새로운 옵션은 다음과 같습니다.

옵션	설명
 -chkpath 	이 옵션은 CLASSPATH에 있는 디렉토리에 대한 공용 쓰기 액세스를 체크합니다.
-opt	이 옵션은 최적화 레벨을 지정합니다.
 -showversion	이 옵션은 JDK 버전을 지정합니다. 이 옵션은 JDK 1.3 및 1.4의 경우에 존재합니다.
 -verbose[:class|gc|jni] 	각 가비지 콜렉션 비우기에 대한 메세지가 표시됩니다.
 -Xrun[:]	JVM 시작 중에 JVM_OnLoad 함수에 대한 서비스 프로그램 및 선택적 매개변수 스트링을 나타내는 메세지가 표시됩니다. 

CL 명령 참조 정보의 RUNJVA(Java 실행) 명령에서 이상의 새로운 옵션에 대해 자세히 설명합니다. CRTJAPGM(Java 프로그램 실행), DLTJAPGM(Java 프로그램 삭제) 명령 및 DSPJVAPGM(Java 프로그램 표시) 명령에 대한 CL 명령 참조 문서에 Java 프로그램 관리에 대한 정보가 있습니다.

Qshell 인터프리터를 통하여 Qshell의 `java` 명령을 사용할 수 있습니다.

Qshell의 `java` 명령에 대한 자세한 내용은 Sun Microsystems Inc.의 Java 툴을 참조하십시오.

Java가 지원하는 CL 명령

IBM Developer Kit for Java(TM)는 다음 CL 명령을 지원합니다.

•

ANZJVM(JVM 분석) 명령은 정보를 검색하고 JVM(Java Virtual Machine)으로 설정합니다. 이 명령은 활동 클래스에 대한 정보를 리턴하여 Java 프로그램의 디버그를 돕습니다.

- CHGJVAPGM(Java 프로그램 변경) 명령은 Java 프로그램의 속성을 변경합니다.
- CRTJVAPGM(Java 프로그램 작성) 명령은 Java 클래스 파일, ZIP 파일 또는 JAR 파일에서 iSeries 서버의 Java 프로그램을 작성합니다.
- DLTJVAPGM(Java 프로그램 삭제) 명령은 Java 클래스 파일, ZIP 파일 또는 JAR 파일과 연관된 iSeries Java 프로그램을 삭제합니다.
- DSPJVAPGM(Java 프로그램 표시) 명령은 iSeries의 Java 프로그램 정보를 표시합니다.
- Java Virtual Machine 덤프(DMPJVM) 명령은 스풀된 프린터 파일에 지정된 작업을 위해 JVM에 대한 정보를 덤프합니다.
- JAVA 명령 및 Java 실행(RUNJVA) 명령은 iSeries Java 프로그램을 실행합니다.

자세한 정보는 프로그램 및 CL 명령 API를 참조하십시오.

ANZJVM(JVM 분석) 명령

Java(TM) Virtual Machine 분석(ANZJVM) 명령은 정보를 검색하고 이를 JVM(Java Virtual Machine)으로 설정합니다. 활동 클래스에 대한 정보를 리턴하여 Java 프로그램의 디버그를 돕기 위한 것입니다.

ANZJVM 명령을 실행할 때 461 페이지의 『가비지 콜렉션 주기 강제 실행』을 지정하기 위한 매개변수가 있으며 강제 실행해야 하면 각 전달 이전에 가비지 콜렉션 주기의 강제 실행을 시도합니다. 분석되는 JVM에 대한 가비지 콜렉션 주기가 실행되지 않았으면 주기를 강제 실행할 수 없습니다. 정보를 저장하는 방법과 전달 사이의 간격에 대한 매개변수도 있습니다.

ANZJVM 명령이 완료된 후에 스풀 출력 파일이 작성됩니다. 자세한 정보는 예: ANZJVM 명령 및 출력 파일을 참조하십시오.

자세한 정보는 ANZJVM 명령 및 ANZJVM 구문 다이어그램을 참조하십시오.

자세한 정보는 ANZJVM 명령에 대한 고려사항을 참조하십시오.

ANZJVM 명령 실행

Java(TM) Virtual Machine 분석(ANZJVM) 명령을 사용하여 지정된 작업에 대한 JVM(Java Virtual Machine) 정보를 수집할 수 있습니다. JVM을 복사하고 나중에 자료를 복사한 사본과 비교하면 자료를 분석하여 오브젝트 손실을 찾을 수 있습니다. 간격 매개변수는 힙(heap) 전달 사이의 시간을 지정하는 데 사용합니다. 간격을 0으로 설정하면 두 가지 힙 전달이 있으며 두 번째 전달은 첫 번째 전달이 완료된 후에 바로 시작됩니다. 그런 다음 두 전달에 대한 정보가 모두 리턴됩니다.

다음의 정보는 힙의 각 클래스에 대해 리턴됩니다.

1. 클래스 이름
2. 가비지 콜렉션 힙(heap) 정보
   a. 전달 1
   b. 전달 2
   c. 가비지 콜렉션 힙의 오브젝트 수 변경
3. 사용된 오브젝트 공간
   a. 전달 1
   b. 전달 2
   c. 오브젝트 크기 변경
4. 오브젝트 표에 대해 나열된 것과 동일한 글로벌 레지스트리에 대한 정보인 글로벌 레지스트리 정보.
5. 로더 이름

가비지 콜렉션 주기 강제 실행: 힙의 클리너 보기를 확보하려면 가비지 콜렉션 주기 이후에 가능하면 빨리 확인하는 것이 바람직합니다. ANZJVM에는 가비지 콜렉션을 강제 실행하도록 지정하는 FRCGC 매개변수가 있습니다. 가능한 옵션은 다음과 같습니다.

- *YES
  힙의 각 ANZJVM을 정리하기 전에 가비지 콜렉션을 강제 실행합니다.
- *NO
  가비지 콜렉션은 ANZJVM에 의해 강제 실행되지 않습니다.

ANZJVM 명령 고려사항

ANZJVM을 실행할 수 있는 시간으로 인해 ANZJVM을 완료하기 전에 JVM이 종료될 가능성이 매우 높습니다. JVM이 종료된 경우, ANZJVM은 확보할 수 있었던 자료와 함께 JVAB606 메세지(즉, ANZJVM을 처리하는 동안 JVM이 종료됨)를 리턴합니다.

JVM이 처리할 수 있는 클래스의 수에는 상한이 없습니다. 처리할 수 있는 클래스 수보다 많은 클래스가 있으면 ANZJVM은 보고되지 않은 추가 정보가 있음을 알게 해주는 메세지와 함께 처리할 수 있는 자료를 리턴합니다. 자료를 절단해야 하는 경우 ANZJVM은 가능한한 많은 정보를 리턴합니다.

내부 매개변수의 길이는 3600초(1시간)로 제한됩니다. ANZJVM이 정보를 리턴할 수 있는 클래스의 수는 시스템에 있는 기억장치의 양으로 제한됩니다.

예: ANZJVM 명령

다음의 ANZJVM 명령 예는 사용자 이름이 JOHN이고 작업 번호가 099112인 QJVACMDSRC라고 하는 작업에 대해 60초 간격으로 JVM에 대한 두 개의 사본을 수집합니다. 사본의 자료는 QSYSPRT 인쇄 장치 파일에 저장됩니다.

```
ANZJVM JOB(099112/JOHN/QJVACMDSRV)
```

스풀 출력 파일의 예는 다음과 같습니다.

ANZJVM 명령의 스풀 출력 파일: 다음은 ANZJVM 명령이 실행된 후에 스풀 출력 파일에 포함된 자료의 예입니다.

```
Mon Feb 26 15:39:12 CST 2002
Job: 099112/JOHN/QJVACMDSRV
Interval: 10 seconds
Total garbage collection cycles prior to running: 29
Total garbage collection cycles after running: 31
GC forced: NO
.....................................................................
. Class loader information                                          .
.....................................................................
0 Default class loader
.....................................................................
. GC heap information                                               .
.....................................................................
Loader
|   Number of pass one objects in the GC heap
|   |         Number of pass two objects in the GC heap
|   |         |         Change in the number of objects in the GC heap
|   |         |         |         Pass one object size (K)
|   |         |         |         |       Pass two object size (K)
|   |         |         |         |       |       Change in object size (K)
|   |         |         |         |       |       |       In global registry
|   |         |         |         |       |       |       |   Class name
0   431359    491363    60004     18979   21619   2640    NO  java/lang/
                                                                  String
```

```
0   8         8         0         0         0         0         NO  sun/misc/
                                                                    URLClass
                                                                    Path$
                                                                    JarLoader
0   4         4         0         0         0         0         NO  java/lang/
                                                                    Object
0   7         7         0         0         0         0         NO  java/util/
                                                                    zip/
                                                                    Inflater
0   2         2         0         0         0         0         NO  java/lang/
                                                                    Thread
0   2         2         0         0         0         0         NO  [Ljava/lang/
                                                                    Class;
0   13        13        0         0         0         0         NO  java/io/
                                                                    File
                                                                    Descriptor
0   2         2         0         0         0         0         NO  java/io/
                                                                    Buffered
                                                                    Writer
0   4         4         0         0         0         0         NO  [Ljava/lang/
                                                                    String;
0   1         1         0         0         0         0         NO  [Ljava/io/
                                                                    ObjectStream
                                                                    Class$
                                                                    ObjectStream
                                                                    ClassEntry;
0   404       404       0         24        24        0         NO  [Ljava/util/
                                                                    HashMap$
                                                                     Entry;
0   423       423       0         15        15        0         NO  java/util/
                                                                    jar/
                                                                    Attributes$
                                                                    Name
0   1         1         0         0         0         0         NO  java/io/
                                                                    Buffered
                                                                    InputStream
0   2         2         0         0         0         0         NO  java/io/
                                                                    Buffered
                                                                    Output
                                                                    Stream
0   1         1         0         0         0         0         NO  java/security/
                                                                    Protection
                                                                    Domain
0   1         1         0         0         0         0         NO  sun/security/
                                                                    provider/Sun
0   1         1         0         0         0         0         NO  java/io/
                                                                    File
                                                                    Permission
0   128       128       0         6         6         0         NO  java/util/
                                                                     Hashtable$
                                                                    Entry
0   2         2         0         0         0         0         NO  java/net/
                                                                    URLClass
                                                                    Loader$
                                                                    ClassFinder
0   1         1         0         0         0         0         NO  java/lang/
                                                                    Runtime
0   1         1         0         0         0         0         NO  java/util/
```

```
                                                                BitSet
0   7         7         0         0         0         0       NO  java/util/
                                                                  jar/
                                                                  JarVerifier
0   2         2         0         0         0         0       NO  java/lang/
                                                                  ThreadGroup
0   22        22        0         1         1         0       NO  java/util/
                                                                  Locale
0   8         8         0         0         0         0       YES java/io/
                                                                  RandomAccess
                                                                  File
0   37        37        0         125       125       0       YES [B
0   1         1         0         0         0         0       NO  sun/misc/
                                                                  Launcher
0   871       871       0         117       117       0       YES [C
0   1         1         0         0         0         0       NO  sun/misc/
                                                                  Launcher$
                                                                  Factory
0   435       435       0         10        10        0       YES java/lang/
                                                                  Class
0   1         1         0         0         0         0       NO  java/util/
                                                                  Collections$
                                                                  EmptyList
0   1         1         0         0         0         0       NO  java/util/
                                                                  Collections$
                                                                  EmptyMap
0   1         1         0         0         0         0       NO  java/lang/
                                                                  String$
                                                                  Case
                                                                  Insensitive
                                                                  Comparator
0   1         1         0         2         2         0       NO  [I
0   3         3         0         0         0         0       NO  java/lang/
                                                                  OutOf
                                                                  Memory
                                                                  Error
0   1         1         0         0         0         0       NO  [J
0   800       800       0         41        41        0       NO  java/util/
                                                                  HashMap$
                                                                  Entry
0   1         1         0         0         0         0       NO  java/util/
                                                                  Random
0   5         5         0         0         0         0       NO  java/security/
                                                                  Access
                                                                  Control
                                                                  Context
0   1         1         0         0         0         0       YES java/lang/
                                                                  ref/
                                                                  Reference$
                                                                  Reference
                                                                  Handler
0   1         1         0         8         8         0       NO  [S
0   1         1         0         0         0         0       NO  [Ljava/lang/
                                                                  ref/Soft
                                                                  Reference;
0   1         1         0         0         0         0       NO  java/io/
                                                                  Object
                                                                  Stream
```

```
                                                                Class$
                                                                Compare
                                                                Member
                                                                ByName
0   7         7         0         0         0         0       NO  java/util/
                                                                jar/
                                                                JarFile$
                                                                JarFileEntry
0   1         1         0         0         0         0       NO  java/util/
                                                                Collections$
                                                                EmptySet
0   1         1         0         0         0         0       NO  [Ljava/
                                                                security/
                                                                cert/
                                                                Certificate;
0   1         1         0         0         0         0       NO  java/lang/
                                                                ref/
                                                                Reference
                                                                Queue
0   1         1         0         0         0         0       NO  java/util/
                                                                 Hashtable$
                                                                Empty
                                                                Enumerator
0   1         1         0         0         0         0       YES sun/misc/
                                                                Launcher$
                                                                AppClass
                                                                Loader
0   1         1         0         0         0         0       NO  java/lang/
                                                                Shutdown$
                                                                Lock
0   17        17        0         0         0         0       NO  java/util/
                                                                Vector
0   3         3         0         0         0         0       NO  java/util/
                                                                Stack
0   17        17        0         1         1         0       NO  java/net/URL
0   21        21        0         1         1         0       NO  java/lang/
                                                                ref/
                                                                Finalizer
0   1         1         0         0         0         0       NO  java/io/
                                                                Os400
                                                                FileSystem
0   1         1         0         0         0         0       NO  java/lang/
                                                                Runtime
                                                                Permission
0   1         1         0         0         0         0       NO  [Ljava/io/
                                                                File;
0   2         2         0         0         0         0       NO  sun/io/
                                                                CharToByte
                                                                ISO8859_1
0   1         1         0         0         0         0       NO  sun/misc/
                                                                Launcher$
                                                                ExtClass
                                                                Loader
0   1         1         0         0         0         0       NO  java/lang/
                                                                ref/
                                                                Soft
                                                                Reference
0   1         1         0         0         0         0       NO  sun/security/
```

```
                                                                provider/
                                                                PolicyFile
0   1         1         0         0         0         0         NO  java/io/
                                                                  Object
                                                                  Stream
                                                                  Class$
                                                                  Compare
                                                                  Class
                                                                  ByName
0   1         1         0         0         0         0         NO  sun/net/www/
                                                                  protocol/
                                                                  file/
                                                                  Handler
0   8         8         0         0         0         0         NO  java/util/
                                                                  jar/
                                                                  JarFile
0   30        30        0         2         2         0         NO  java/util/
                                                                  Hashtable
0   1         1         0         0         0         0         NO  java/lang/
                                                                  ref/
                                                                  Reference$
                                                                  Lock
0   2         2         0         0         0         0         NO  java/io/
                                                                  PrintStream
0   1         1         0         0         0         0         NO  java/util/
                                                                   Hashtable$
                                                                  Empty
                                                                  Iterator
0   4         4         0         0         0         0         NO  java/io/File
0   391       391       0         12        12        0         NO  java/util/
                                                                  jar/
                                                                  Attributes
0   2         2         0         0         0         0         NO  sun/misc/
                                                                  URLClassPath
0   2         2         0         0         0         0         NO  java/io/
                                                                  FileInput
                                                                  Stream
0   2         2         0         0         0         0         NO  java/io/
                                                                  Output
                                                                  Stream
                                                                  Writer
0   11        11        0         0         0         0         NO  java/util/
                                                                  ArrayList
0   1         1         0         0         0         0         NO  java/net/
                                                                  Unknown
                                                                  Content
                                                                  Handler
0   3         3         0         0         0         0         NO  java/lang/
                                                                  ref/
                                                                  Reference
                                                                  Queue$Lock
0   2         2         0         0         0         0         NO  java/io/
                                                                  FileOutput
                                                                  Stream
0   1         1         0         0         0         0         NO  sun/misc/
                                                                  URLClass
                                                                  Path$
                                                                  FileLoader
```

```
0   31        31        0         5246    5246    0       NO  [Ljava/lang/
                                                              Object;
0   1         1         0         0       0       0       NO  java/lang/
                                                              Class
                                                              Loader$
                                                              Native
                                                              Library
0   2         2         0         0       0       0       NO  [Ljava/lang/
                                                              Thread;
0   404       404       0         35      35      0       NO  java/util/
                                                              HashMap
0   2         2         0         0       0       0       NO  java/lang/
                                                              Boolean
0   1         1         0         0       0       0       YES java/lang/ref/
                                                              Finalizer$
                                                              Finalizer
                                                              Thread
0   1         1         0         0       0       0       NO  sun/security/
                                                              provider/
                                                              Policy
                                                              Permissions
0   7         7         0         0       0       0       NO  java/util/jar/
                                                              Manifest
0   2         2         0         0       0       0       NO  sun/net/www/
                                                              protocol/
                                                              jar/
                                                              Handler
0   1         1         0         0       0       0       NO  com/sun/
                                                              rsajca/
                                                              Provider
0   1         1         0         0       0       0       NO  java/util/
                                                              Collections
                                                              $Reverse
                                                              Comparator
0   2         2         0         0       0       0       NO  [Ljava/io/
                                                              ObjectStream
                                                              Field;
0   1         1         0         0       0       0       NO  java/security/
                                                              CodeSource
0   34        34        0         5       5       0       NO  [Ljava/util/
                                                               Hashtable$
                                                               Entry;
0   2         2         0         0       0       0       NO  java/lang/ref/
                                                              Reference
                                                              Queue$Null
0   2         2         0         0       0       0       NO  java/util/
                                                              Properties
0   2         2         0         0       0       0       NO  java/util/
                                                              HashSet
0   1         1         0         0       0       0       NO  [Ljava/lang/
                                                              ThreadGroup;
0   1         1         0         0       0       0       NO  java/util/
                                                              HashMap$
                                                              EmptyHash
                                                              Iterator
0   7         7         0         0       0       0       NO  java/io/
                                                              ByteArray
                                                              OutputStream
```

```
.................................................................
. Global registry information                                   .
.................................................................
Loader
|   Number of pass one objects in the GC heap
|   |         Number of pass two objects in the GC heap
|   |         |         Change in the number of objects in the GC heap
|   |         |         |         Pass one object size (K)
|   |         |         |         |       Pass two object size (K)
|   |         |         |         |       |       Change in object size (K)
|   |         |         |         |       |       |       Class name
0   8         8         0         0       0       0       java/io/
                                                              RandomAccessFile
0   8         8         0         64      64      0       [B
0   10        10        0         1       1       0       [C
0   439       439       0         10      10      0       java/lang/Class
0   1         1         0         0       0       0       java/lang/ref/
                                                              Reference$
                                                              ReferenceHandler
0   1         1         0         0       0       0       sun/misc/
                                                              Launcher$
                                                              AppClassLoader
0   1         1         0         0       0       0       java/lang/ref/
                                                              Finalizer$
                                                              FinalizerThread
```

예: Java 프로그램 변경(CHGJVAPGM) 명령

클래스 파일과 연관된 Java(TM) 프로그램을 변경하려면, CHGJVAOGM(Java 프로그램 변경) 명령을 사용하십시오. Java 프로그램 클래스 파일 바이트 코드가 해석됩니다. 프로그램을 시작하려면 RUNJVA(Java 실행) 명령을 사용하십시오. Java 프로그램은 지정된 속성이 현재 프로그램의 속성과 다른 경우에만 재작성됩니다.

예 1: 해석된 Java 프로그램 변경

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
CHGJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class')
   OPTIMIZE(*INTERPRET)
```

다음의 예는 프로그램이 최적화된다는 것을 제외하면 예 1과 같습니다. OPTIMIZE(10)으로 변경되므로 프로그램에는 Java 프로그램 시작시에 실행되는 컴파일 기계 명령어가 포함되어 있습니다.

예 2: JAR 파일에서 최적화된 Java 프로그램 변경

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
CHGJVAPGM CLSF('/projectB/myJavaappfile.jar')
   OPTIMIZE(10)
```

구문 다이어그램과 매개변수에 대해서는 CHGJVAPGM(Java 프로그램 변경) 명령을 참조하십시오.

사용권 내부 코드 옵션 매개변수 스트링

이 표는 사용권 내부 코드 옵션(LICOPT) 매개변수가 인식하는 스트링을 보여줍니다. 이 스트링은 대소문자를 구분하지 않지만 읽기 쉽도록 대소문자가 혼합되어 표시됩니다.

LICOPT 매개변수 스트링

스트링	설명
AllFieldsVolatile	설정되는 경우 모든 필드를 일시적으로 취급합니다.
NoAllFieldsVolatile	설정되는 경우 필드를 일시적으로 취급하지 않습니다.
AllowBindingToLoadedClasses	실행 중인 JVM(Java(TM) virtual machine)내의 defineClass 호출의 결과로서 작성된 임시 클래스 표시가 동일한 JVM내의 다른 클래스 표시에 밀접하게 바인드될 수 있음을 나타냅니다.
NoAllowBindingToLoadedClasses	실행 중인 JVM(Java Virtual Machine)내의 defineClass 호출의 결과로서 작성된 임시 클래스 표시가 동일한 JVM내의 다른 클래스 표시에 밀접하게 바인드될 수 없슴을 표시합니다.
AllowClassCloning	Jar 파일에 대해 여러 Java 프로그램이 생성된 경우 한 프로그램으로부터의 클래스 사본을 다른 프로그램에 대해 생성된 코드에 포함시킬 수 있도록 합니다. 공격적인 인라인을 용이하게 합니다.
NoAllowClassCloning	한 프로그램으로부터의 클래스 사본을 다른 프로그램에 대해 생성된 코드에 포함시키지 못하도록 합니다.
AllowInterJarBinding	클래스나 컴파일되고 있는 JAR 파일 외부의 클래스에 대한 밀접한 바인드를 허용합니다. 공격적인 최적화를 용이하게 합니다.
NoAllowInterJarBinding	컴파일되고 있는 클래스나 JAR 파일 외부의 클래스에 대한 밀접한 바인드를 허용하지 않습니다. 이것은 CRTJVAPGM에 존재하는 CLASSPATH 및 JDKVER 매개변수를 대체합니다.
 AllowMultiThreadedCreate	하나의 스레드만을 사용하여 CRTJVAPGM이 보통 때와 같이 수행함을 나타냅니다. 
 NoAllowMultiThreadedCreate	CRTJVAPGM은 작성 중에 사용할 수 있는 여러 스레드를 사용합니다. 
AnalyzeObjectLifetimes	수명이 짧은 오브젝트를 판별하기 위해 시각적 클래스를 사용하여 분석을 수행합니다. 수명이 짧은 오브젝트는 할당 시에 사용된 메소드보다 수명이 길지 않으며 보다 공격적인 최적화에 달려 있을 수 있습니다.
NoAnalyzeObjectLifetimes	수명이 짧은 오브젝트의 분석을 수행하지 않습니다.
AllowBindingWithinJar	ZIP 파일이나 JAR 파일내의 클래스 표시가 동일한 ZIP 파일이나 JAR 파일내의 다른 클래스 표시에 밀접하게 바인드될 수 있음을 표시합니다.
NoAllowBindingWithinJar	ZIP 파일이나 JAR 파일내의 클래스 표시가 동일한 ZIP 파일이나 JAR 파일내의 다른 클래스 표시에 밀접하게 바인드될 수 없음을 표시합니다.
AllowInlining	변환 프로그램은 로컬 메소드 인라인이 사용됨을 알립니다. 디폴트 최적화 레벨은 30 및 40입니다.

스트링	설명
NoAllowInlining	변환 프로그램은 로컬 메소드 인라인이 허용됨을 알리지 않습니다.
AssumeUnknownFieldsNonvolatile	외부 클래스에 있는 필드의 속성을 판별할 수 없을 때 이 매개변수는 필드가 비휘발성이라고 가정하여 코드를 생성합니다.
NoAssumeUnknownFieldsNonvolatile	외부 클래스에 있는 필드의 속성을 판별할 수 없을 때 이 매개변수는 필드가 휘발성이라고 가정하여 코드를 생성합니다.
BindErrorHandling	AssumeUnknownFieldsNonvolatile, PreresolveExtRef 또는 PreLoadExtRef 사용권 내부 코드 옵션 수신의 결과로서 JVM(Java Virtual Machine) 클래스 로더가 클래스 표시에 메소드 표시가 들어 있고 현재 문맥에서 해당 메소드 표시를 사용할 수 없음을 감지하는 경우에 취해야 하는 조치를 지정합니다.
BindInit	로컬 init 메소드에 대해 바인드 호출을 사용하십시오.
NoBindInit	로컬 init 메소드에 대해 바인드 호출을 사용하지 마십시오.
BindSpecial	로컬 고유 메소드에 대해 바인드 호출을 사용하십시오.
NoBindSpecial	로컬 고유 메소드에 대해 바인드 호출을 사용하지 마십시오.
BindStatic	로컬 정적 메소드에 대해 바인드 호출을 사용하십시오.
NoBindStatic	로컬 정적 메소드에 대해 바인드 호출을 사용하지 마십시오.
BindTrivialFields	프로그램 작성시 간단한 필드 참조를 바인드하십시오.
NoBindTrivialFields	처음에 필드 참조를 해제하십시오.
BindVirtual	로컬 최종 가상 메소드에 대해 바인드 호출을 사용하십시오.
NoBindVirtual	로컬 최종 가상 메소드에 대해 바인드 호출을 사용하지 마십시오.
DeferResolveOnClass	클래스명으로 가정되는 스트링 매개변수를 선택하십시오(예를 들어, java.lang.Integer). PreresolveExtRef를 최적화 레벨 40으로 설정하면 DeferResolveOnClass로 지정된 클래스는 사전 해제 작업이 아닙니다. 이것은 코드에서 사용되지 않는 경로의 일부 클래스가 CLASSPATH에 없는 경우에 유용합니다. 누락된 각 클래스에 대해 ″DeferResolveOnClass=‘somepath.someclass’″를 지정하여 이와 관계없이 최적화 레벨 40을 사용할 수 있습니다. 여러 개의 DeferResolveOnClass 항목들이 허용됩니다.
DevirtualizeFinalJDK	CRTJVAPGM은 표준 JDK 지식을 사용하여 파일 클래스의 마지막 메소드 또는 멤버로 알려진 JDK에 대한 호출을 가상화 해제할 수 있습니다. 디폴트는 최적화 레벨 30 및 40입니다.
NoDevirtualizeFinalJDK	CRTJVAPGM은 표준 JDK의 지식을 사용하여 파일 클래스의 마지막 메소드 또는 멤버로 알려진 JDK에 대한 호출을 가상화 해제할 수 없습니다.
DevirtualizeRecursive	일부 순환 메소드의 경우에 특별한 코드가 생성되도록 하고 순환 메소드 호출의 많은 오버헤드를 제거합니다. 그러나 순환 메소드에 대한 초기 항목에서 추가 점검 논리가 생성되므로, 좁은 순환의 경우에 성능이 향상될 수 없습니다.
NoDevirtualizeRecursive	일부 순환 메소드의 경우에 특별한 코드가 생성되도록 하지 않습니다.

스트링	설명
DisableIntCse	특정 유형의 정수 표현식에 대한 코드를 생성할 때 특정한 공통 부속표현식 최적화가 작동 불가능하게 만듭니다. 이것은 최적화 변환 프로그램에 다른 최적화 기회를 노출시켜서 전체 최적화를 향상시킬 수 있습니다.
NoDisableIntCse	특정 유형의 정수 표현식에 대한 코드를 생성할 때 특정한 공통 부속표현식 최적화가 작동 불가능하게 만들지 않습니다. 이것은 일반적으로 더 낮은 최적화 레벨에서 더 좋은 수행 코드를 가져옵니다.
DoExtBlockCSE	확장 기본 블록 공통 부속표현식 제거를 수행하십시오.
NoDoExtBlockCSE	확장 기본 블록 공통 부속표현식 제거를 수행하지 마십시오.
DoLocalCSE	로컬 공통 부속표현식 제거를 수행하십시오.
NoDoLocalCSE	로컬 공통 부속표현식 제거를 수행하지 마십시오.
EnableCseForCastCheck	설정되는 경우 초기 인스턴스로 대그(DAG)할 수 있는 castcheck 코드를 생성하십시오.
NoEnableCseForCastCheck	설정되지 않은 경우 초기 인스턴스로 대그(DAG)할 수 있는 castcheck 코드를 생성하지 않습니다.
ErrorReporting	런타임 오류 보고 필드**: 확인 또는 클래스 형식 오류가 발생할 때 컴파일이 실패하게 하는 옵션을 제공합니다. 0=모든 오류를 즉시 보고; 0=모든 오류를 즉시 보고; 1=바이트코드 검증 오류 보고 연기; 2=바이트코드 검증 오류 및 클래스 형식 오류에서 런타임 보고 연기.
 HideInternalMethods	복제된 클래스의 메소드를 내부 메소드로 만들어 이에 대한 참조가 없거나 모든 참조가 인라인화된 경우에 이러한 메소드를 생략할 수 있습니다. 디폴트는 최적화 40의 경우는 HideInternalMethods이고 최적화가 0 - 30인 경우에는 NoHideInternalMethods입니다.
InlineArrayCopy	스칼라 배열의 일부 경우에 System.arraycopy 메소드의 인라인닝을 유발합니다.
NoInlineArrayCopy	System.arraycopy 메소드의 인라인닝을 금지합니다.
InlineInit	java.lang 클래스에 대한 초기 메소드를 인라인합니다.
NoInlineInit	초기 메소드를 인라인하지 않습니다.
InlineMiscFloat	java.lang.Math의 기타 float/double 메소드를 인라인합니다.
NoInlineMiscFloat	기타 float/double 메소드를 인라인하지 않습니다.
InlineMiscInt	java.lang.Math의 기타 int/long 메소드를 인라인합니다.
NoInlineMiscInt	기타 int/long 메소드를 인라인하지 않습니다.
InlineStringMethods	java/lang/String에서 특정 메소드의 인라인을 허용합니다.
NoInlineStringMethods	java/lang/String에서 특정 메소드의 인라인을 허용하지 않습니다.
InlineTransFloat	java.lang.Math의 고급 float/double 메소드를 인라인합니다.
NoInlineTransFloat	초월 float/double 메소드를 인라인화합니다.
OptimizeJsr	단일 목표가 있는 ″jsr″ 바이트코드에 대해 보다 나은 코드를 생성합니다.
NoOptimizeJsr	단일 목표가 있는 ″jsr″ 바이트코드에 대해 보다 나은 코드의 생성을 억제합니다.

스트링	설명
PreloadExtRef	참조된 클래스가 메소드 입력시에 사전로드(클래스 초기화없이)될 수 있음을 표시합니다.
NoPreloadExtRef	참조된 클래스가 메소드 입력시에 사전로드될 수 없음을 표시합니다. 그러나 PreresolveExtRef 매개변수는 이 설정을 대체하며 참조된 클래스가 사전로드되고 초기화되도록 만듭니다
PreresolveExtRef	메소드 항목에서 참조한 메소드를 사전해석합니다.
NoPreresolveExtRef	처음 접촉시 메소드 참조를 해석합니다. 다른 기계에서 실행되는 프로그램에서 "클래스를 찾을 수 없음" 예외를 분석하는 데 사용합니다.
ProgramSizeFactor	JAR 파일의 크기가 여러 Java 프로그램을 요구하기에 충분한 경우 이 숫자 값(디폴트 100)은 각 프로그램이 증가할 수 있는 크기를 판별하는 데 사용됩니다.
ShortCktAthrow	설정하면 athrow 단락을 시도합니다.
NoShortCktAthrow	설정하지 않으면 athrows 단락을 시도하지 않습니다.
ShortCktExSubclasses	설정하면 Exception의 일부 서브클래스를 인식하고 직접 단락시킵니다.
NoShortCktExSubclasses	설정하지 않으면 Exception의 일부 서브클래스를 인식하고 직접 단락시킵니다.
StrictFloat	Java 스펙을 엄격하게 준수하지 않는 부동 소수점 최적화를 금지합니다.
NoStrictFloat	Java 스펙을 엄격하게 준수하지 않는 부동 소수점 최적화를 허용합니다.

이중 별표(**)는 이 스트링이 stringname=number(사이에 공백 없음) 구문에 입력을 위한 값이 필수임을 의미합니다.

예: Java 프로그램 작성(CRTJVAPGM) 명령

Java(TM) 프로그램을 작성하고 이를 클래스 파일(myJavaClassFileName)과 연관시키려면, CRTJVAPGM(Java 프로그램 작성) 명령을 사용하십시오. OPTIMIZE(*INTERPRET)를 사용하여 작성하면 Java 프로그램 클래스 파일 바이트코드가 해석됩니다. 프로그램을 시작하려면 RUNJVA(Java 실행) 명령을 사용하십시오.

예 1: 해석된 Java 프로그램 작성

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
CRTJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class')
   OPTIMIZE(*INTERPRET)
```

이 예는 프로그램이 최적화된다는 것을 제외하면 예 1과 같습니다. OPTIMIZE(40)으로 작성되므로, 프로그램에는 Java 프로그램 시작시 실행되는 컴파일 기계 명령어가 포함됩니다.

예 2: 최적화된 Java 프로그램 작성

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
CRTJVAPGM CLSF('/projectB/team2/myJavaclassfile.class')
   OPTIMIZE(40)
```

구문 다이어그램 및 매개변수에 대한 자세한 내용은 CRTJVAPGM(Java 프로그램 작성) 명령을 참조하십시오.

예: Java 프로그램 삭제(DLTJVAPGM) 명령

Java(TM) 프로그램 삭제(DLTJVAPGM) 명령은 이름이 myJavaClassFileName인 지정된 클래스 파일과 연관된 Java 프로그램을 삭제합니다.

주: DLTJVAPGM 명령은 클래스 파일 또는 ZIP 파일을 삭제하지 않습니다.

예 1: Java 프로그램 삭제

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
DLTJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class')
```

구문 다이어그램 및 매개변수에 대한 자세한 내용은 DLTJVAPGM(Java 프로그램 삭제) 명령을 참조하십시오.

예: JVM(Java Virtual Machine) 덤프(DMPJVM) 명령

JVM(Java(TM) Virtual Machine) 덤프(DMPJVM) 명령은 지정된 작업에 대해 JVM에 대한 정보를 덤프합니다.

예 1: JVM 덤프

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
DMPJVM JOB(099246/FRED/QJVACMDSRV)
```

DMPJVM 명령이 099246/FRED/QJVACMDSRV라는 이름의 작업에서 실행 중인 JVM에 관한 정보를 덤프합니다.

출력 예:

```
JAVA VIRTUAL MACHINE INFORMATION: 099246/FRED/QJVACMDSRV
 ..................................................................
 . Classpath
 ..................................................................
/QIBM/ProdData/Java400/jdk117/lib/jdkptf117.zip:/QIBM/ProdData/Java400/jdk1
17/lib/classes.zip:/QIBM/ProdData/Java400/ext/IBMmisc.jar:/QIBM/ProdData/Ja
va400/ext/db2_classes.jar:/QIBM/ProdData/Java400/ext/jssl.jar:/QIBM/ProdDat
a/Java400/ext/ibmjssl.jar:/QIBM/ProdData/Java400/:/home/fred
 ..................................................................
 . Garbage collection
 ..................................................................
Garbage collector parameters
   Initial size: 2048 K
   Max size: *NOMAX
```

```
Current values
   Heap size: 9476 K
   Garbage collections: 0
 ..................................................................
 . Thread information
 ..................................................................
Information for 3 thread(s) of 3 thread(s) processed
Thread: 00000001 Thread-0
  TDE: B000200002941000
  Thread priority: 5
  Thread status: Destroy wait
  Thread group: main
  Runnable: java/lang/Thread
 Stack:
   None
 Locks:
   None
 ..................................................................
Thread: 00000003 t2
  TDE:  B000100005B37000
  Thread priority: 5
  Thread status: Timed wait
  Thread group: main
  Runnable: dbgtest2
 Stack:
    java/io/BufferedInputStream.read()I+11 (BufferedInputStream.java:154)
    pressEnter.theFirstMethod(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V+1
        0 (dbgtest2.java:15)
    dbgtest2.run()V+69 (dbgtest2.java:44)
    java/lang/Thread.run()V+11 (Thread.java:466)
 Locks:
   None
 ..................................................................
Thread: 00000002 t1
  TDE:  B000100005B33000
  Thread priority: 5
  Thread status: Java wait
  Thread group: main
  Runnable: dbgtest2
 Stack:
    pressEnter.theFirstMethod(Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;)V+0 (dbgtest2.java:14)
    dbgtest2.run()V+69 (dbgtest2.java:44)
    java/lang/Thread.run()V+11 (Thread.java:466)
 Locks:
   None
 ..................................................................
```

예: Java 프로그램 표시(DSPJVAPGM) 명령

DSPJVAPGM(Java 프로그램 표시) 명령은 myJavaClassFileName으로 명명된 지정 클래스 파일과 연관된 Java 프로그램을 표시합니다.

예 1: Java 프로그램 표시

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
DSPJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class') OUTPUT(*)
```

구문 다이어그램 및 매개변수에 대한 자세한 내용은 DSPJVAPGM(Java 프로그램 표시) 명령을 참조하십시오.

JAVA 명령

JAVA 명령은 Run Java(TM) (RUNJVA) 명령과 동일한 기능을 합니다. 이 명령을 상호 교환하여 사용할 수 있습니다. JAVA 명령과 함께 사용할 수 있는 매개변수와 정보는 RUNJVA(Java 실행) 명령 실행을 참조하십시오.

예: RUNJVA(Java 실행) 명령 사용

RUNJVA(Java(TM) 실행) 명령은 클래스와 연관된 iSeries Java 프로그램을 실행합니다.

예 1: Java 프로그램 실행

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
RUNJVA CLASS('/projectA/myJavaclassname')
```

구문 다이어그램 및 매개변수에 대한 자세한 내용은 RUNJVA(Java 실행) 명령 실행을 참조하십시오.

Java가 지원하는 iSeries Navigator 명령

iSeries Navigator는 Windows(R) 데스크탑을 위한 그래픽 인터페이스입니다. Windows용 iSeries Access의 일부이며 관리자나 사용자가 일상 작업을 수행하는데 필요한 많은 iSeries 기능을 담당합니다.

iSeries Navigator는 iSeries Access for Windows의 파일 시스템 옵션에 들어있는 플러그 인으로서 Java(TM)를 지원합니다. iSeries Navigator Java 플러그인을 사용하려면 iSeries에 IBM Developer Kit for Java를 설치해야 합니다. 그런 다음, 퍼스널 컴퓨터에 Java 플러그인을 설치하려면 설치 폴더에서 선택적 설정을 통해 파일 시스템을 선택하십시오.

클래스, JAR, ZIP 및 Java 파일은 통합 파일 시스템에 상주합니다. iSeries Navigator를 사용하면 오른쪽 분할 창에서 이 파일들을 볼 수 있습니다. 사용하려는 클래스, JAR, ZIP 또는 Java 파일에서 오른쪽 버튼을 클릭하십시오. 문맥 메뉴가 나타납니다.

문맥 메뉴에서 **연관된 Java 프로그램 —> 신규...**를 선택하면 Java 변환 프로그램이 시작되어 클래스, JAR 또는 ZIP 파일과 연관된 iSeries Java 프로그램을 작성합니다.

대화 상자를 사용하여 프로그램 작성 방법에 대한 세부사항을 지정할 수 있습니다. Java 변환이나 Java 해석을 위한 프로그램을 작성할 수 있습니다.

주: 변환을 선택하면 클래스 파일의 바이트코드가 해석을 사용할 때보다 성능이 더 향상되는 RISC 명령어로 변환됩니다.

문맥 메뉴에서 **연관된 Java 프로그램 —> 편집...**을 선택하면 Java 클래스 파일, ZIP 파일 또는 JAR 파일에 접속된 Java 프로그램의 속성이 변경됩니다.

문맥 메뉴에서 **연관된 Java 프로그램 —> 실행...**을 선택하면 iSeries 서버에서 클래스 파일이 실행됩니다.

JAR 또는 ZIP 파일을 선택하고 이 JAR 또는 ZIP 파일 내에 위치한 클래스 파일을 실행할 수도 있습니다. 대화 상자를 사용하여 프로그램 실행 방법의 세부사항을 지정할 수 있습니다.

이미 **연관된 Java 프로그램 —> 신규...**를 선택했으면 프로그램을 실행할 때 클래스 파일과 연관된 iSeries Java 프로그램을 사용합니다.

iSeries Java 프로그램이 클래스 파일과 아직 연관되지 않았으면 프로그램을 실행하기 전에 iSeries Java 프로그램을 작성합니다.

문맥 메뉴에서 **연관된 Java 프로그램 —> 삭제...**를 선택하면 클래스, JAR 또는 ZIP 파일과 연관된 iSeries Java 프로그램을 삭제합니다.

문맥 메뉴에서 **등록 정보**를 선택하면 **Java 프로그램** 및 **Java 옵션** 탭이 들어 있는 등록 정보 대화 상자가 표시됩니다. 이러한 탭을 통해 연관된 iSeries Java 프로그램이 클래스, JAR 또는 ZIP 파일에 대해 어떻게 작성되었는지에 대한 세부사항을 볼 수 있습니다.

주: 이러한 패널들은 Java 프로그램 정보표시입니다.

문맥 메뉴에서 **Java 파일 컴파일**을 선택하면 선택한 Java 파일이 클래스 파일 바이트코드로 변환됩니다.

새 Java 프로그램, **Java 프로그램 편집**, **Java 프로그램 실행**, **Java 프로그램**, **Java 옵션**, **Java 파일 컴파일** 및 **Java 프로그램 삭제** iSeries Navigator 대화 상자의 매개변수와 옵션에 대해서는 iSeries Navigator에 포함된 도움말 정보를 참조하십시오.

제 7 장 선택적 패키지

선택적 패키지는 핵심 Java 플랫폼 API를 확장하는 API(Application Programming Interfaces)를 정의합니다. 다음은 IBM Developer Kit for Java(TM)와 함께 사용할 수 있는 선택적인 패키지입니다.

JAAS(Java Authentication and Authorization Service)
JAAS(Java Authentication and Authorization Service)를 사용하면 사용자나 ID를 현재 Java 스레드에 연관시킬 수 있습니다.

JCE(Java Cryptography Extension)
JCE(Java Cryptography Extension)는 암호화, 키 생성 및 키 동의, 메세지 인증 코드(MAC) 알고리즘에 대한 구조와 구현을 제공합니다. JCE는 보안 스트림과 비밀 오브젝트도 지원합니다.

Java 명명 및 디렉토리 인터페이스
JNDI(Java 명명 및 디렉토리 인터페이스)는 JavaSoft의 플랫폼 어플리케이션 프로그램 인터페이스(API)의 일부입니다. JNDI로 복수 명명 및 디렉토리 서비스에 연결할 수있습니다. 이 인터페이스를 사용하여 강력하고 이식성 있는 디렉토리를 사용할 수 있는 Java 어플리케이션을 빌드할 수 있습니다.

JSSL(Java Secure Socket Layer)
JSSL(Java Secure Socket Layer)은 보안 인터넷 통신을 가능하게 하는 Java 패키지 세트입니다. SSL 및 TLS(Transport Layer Security) 프로토콜의 Java 버전을 구현하며 자료 암호화, 서버 인증, 메세지 무결성 및 선택적 클라이언트 인증을 위한 기능이 들어 있습니다.

JavaMail
JavaMail API는 전자(전자 우편) 시스템을 모델화하는 추상 클래스 세트를 제공합니다. API는 Java 기반 전자 우편 및 메세징 어플리케이션을 빌드할 수 있도록 플랫폼과 상관 없으며 프로토콜과도 상관 없는 구조를 제공합니다.

JavaPrintService
Java 인쇄 서비스 API를 사용하면 모든 Java 플랫폼에서 인쇄가 가능합니다. Java 1.4는 Java 런타임 환경 및 써드 파티가 PDF, Postscript 및 AFP(Advanced Function Presentation)와 같이 다양한 인쇄 형식을 작성하기 위한 스트림 생성기 플러그인을 제공할 수 있는 구조를 제공합니다.

Java 명명 및 디렉토리 인터페이스

JNDI(Java(TM) Naming and Directory Interface)는 JavaSoft의 플랫폼 API(어플리케이션 프로그램 인터페이스)의 한 파트입니다. JNDI로 복수 명명 및 디렉토리 서비스에 연속적으로 연결할 수 있습니다. 이 인터페이스를 사용하여 강력하고 이식성 있는 디렉토리를 사용할 수 있는 Java 어플리케이션을 빌드할 수 있습니다.

JavaSoft는 IBM, SunSoft, Novell, Netscape 및 Hewlett-Packard Co.를 비롯한 업계 선두 협력 업체와 함께 JNDI 스펙을 개발했습니다.

JNDI에 대한 자세한 정보는 Sun Microsystems, Inc.의 Java 명명 및 디렉토리 인터페이스를 참조하십시오. IBM 고유 정보의 경우에는 IBM JNDI LDAP 공급자 프로그래밍 안내서를 참조하십시오.

IBM JNDI LDAP 제공자 프로그래밍 안내서

이 프로그래밍 안내서에서는 사용자가 JNDI(Java(TM) Naming and Directory Interface)와 LDAP(Lightweight Directory Access Protocol) 조작 방법에 익숙한 것으로 가정합니다. 자세한 정보는 Sun Microsystems에서 JNDI 문서를 참조하십시오.

IBM은 SDK 또는 JRE(Java Runtime Environment) 1.2.2와 함께 사용할 수 있는 JNDI용 LDAP 서비스 제공자를 제공합니다. IBM JNDI LDAP 제공자를 SDK 또는 JRE 1.3 이상과 함께 사용하는 것은 지원되지 않습니다. 그 대신 JNDI와 Sun Microsystems, Inc. JNDI LDAP 제공자를 사용해야 하며 SDK 및 JRE 1.3의 일부입니다. Sun Microsystems, Inc. JNDI LDAP 제공자를 SDK 및 JRE 1.2.2와 함께 사용할 수도 있지만 이러한 구성요소는 Sun Microsystems, Inc. JNDI 웹 사이트에서 다운로드해야 하며 지원은 Sun에서 제공합니다. 이 프로그래밍 안내서는 IBM JNDI LDAP 제공자를 SDK 또는 JRE 1.2.2와 함께 사용하는 것을 설명합니다.

IBM JNDI LDAP 제공자를 사용하여 코드를 컴파일하거나 실행하려면 클래스 경로에 다음을 추가하십시오.

```
/QIBM/ProdData/Java400/ext/ibmjndi.jar:/QIBM/ProdData/Java400/ext/jndi.jar
```

프로그램 안내서는 다음의 주제를 언급합니다:

초기 문맥 작성
이 주제는 LDAP 서버에 연결할 초기 문맥을 작성하는 방법을 설명합니다. JNDI는 클라이언트가 LDAP(Lightweight Directory Access Protocol) 서버에 대해 작업할 수 있도록 두 가지 방법을 지원합니다.

- 클라이언트가 문맥 작성 시에 서버를 식별합니다.
- URL 스트링을 문맥의 메소드에 직접 전달합니다.

LDAP V3 URL
이 주제는 LDAP URL 구문을 정의합니다.

서버 바인딩 및 SASL 지원
특정 조작이 허용되기 전에 서버는 클라이언트를 인증해야 합니다. LDAP는 이를 서버로의 바인딩으로 언급합니다. LDAP 프로토콜은 SASL(Simple Authentication and Security Layer) 메카니즘도 지원하기 위해 인증을 확대했습니다. 이들 메카니즘을 사용하면 사용자의 ID와 암호를 명확한 텍스트로 보내어 사용자의 보안을 불필요하게 위협하지 않고 서버에 클라이언트를 식별하는 보다 복잡한 방법을 사용할 수 있습니다.

속성 탐색 및 확보
JNDI는 LDAP(Lightweight Directory Access Protocol) 디렉토리 탐색 시 유연성을 제공합니다.

디렉토리에 항목 추가 및 삭제
JNDI는 디렉토리에 항목을 추가 및 삭제하게 합니다. 이 주제에는 이러한 타스크의 수행 방법에 대한 예가 들어 있습니다.

489 페이지의『속성 변경』
JNDI는 디렉토리 항목에서 속성을 변경, 작성 또는 제거할 수 있게 합니다.

489 페이지의『디렉토리 항목 이름 변경』
JNDI는 기본 문맥과 관련된 곳이면 어느 곳에서나 디렉토리 항목의 이름을 변경할 수 있게 합니다. 이 주제는 디렉토리 항목의 이름을 변경하기 위해 사용하는 rename 메소드에 영향을 주는 등록 정보를 식별합니다.

참조 및 탐색 참조
LDAP 서버는 참조 또는 탐색 참조를 리턴할 수 있습니다. 참조는 어느 조작에 대해서나 리턴할 수 있으며 서버가 요구의 목표 항목을 보유하지 않음을 나타냅니다. 탐색 참조는 탐색 조작에 대해서만 리턴됩니다.

LDAP 제어
LDAP v3 스펙은 확장 정보를 송수신하기 위한 제어를 추가했습니다. 서버로 보낸 제어를 요구 제어라고 합니다. 서버로부터 받은 제어를 응답 제어라고 합니다.

2진 속성
LDAP 프로토콜은 검색한 2진 속성과 텍스트 속성 간의 구별을 제공하지 않습니다. 그 대신 클라이언트 어플리케이션이 자료를 처리하는 방법을 알 것으로 예상합니다. 이 주제에서는 검색된 속성에 발생한 상태의 세 가지 처리 방법과 스트링으로 변환되었는지의 여부를 설명합니다.

스키마
LDAP 서버의 스키마 구조를 검색하고 보고 갱신할 수 있습니다. LDAP 스펙이 정의한 스키마 정보를 제공하는 서버만을 지원합니다.

SASL 플러그인
사용자는 자신의 SASL(Simple Authentication and Security Layer) 플러그인을 작성할 수 있습니다. 이 주제는 플러그인의 작성을 시작하는 데 도움이 되는 코드 예를 제공합니다.

클라이언트측 캐싱
캐싱은 최근에 요구된 정보를 로컬로 저장하는 방법을 제공합니다. 이미 확보한 정보에 대해 리모트 서버로 되돌아가는 대신 반복된 조회를 로컬로 검색하여 성능을 향상시킵니다.

IBMJNDI 클래스 버전 검색
이 주제는 LDAP의 IBMJNDI 클래스 버전을 검색하기 위해 사용할 수 있는 정적 메소드를 나타냅니다.

500 페이지의 『준수 고려사항 및 추가 등록 정보』
IBM LDAP 제공자와 Sun의 JNDI Implementor Guidelines for LDAP Service Providers를 사용할 때 고려해야 하는 사항을 알려면 이 주제를 참조하십시오. 지원되는 등록 정보와 제거된 등록 정보도 나타냅니다.

초기 문맥 작성

JNDI(Java(TM) Naming and Directory Interface)는 클라이언트가 LDAP(Lightweight Directory Access Protocol) 서버와 작업하기 위한 두 개의 다른 방법을 제공합니다. 우선적이고 가장 일반적인 방법은 클라이언트가 문맥 작성 시에 서버를 식별하도록 하는 것입니다. 그러면 문맥의 메소드에 DN 기반 이름을 전달하여 이 열린 연결에 대한 조작을 수행합니다. 다음의 두 등록 정보는 이 조작 유형을 지원합니다.

java.naming.factory.initial(Context.INITIAL_CONTEXT_FACTORY)
이 등록 정보는 com.ibm.jndi.LDAPCtxFactory로 설정해야 합니다.

java.naming.provider.url(Context.PROVIDER_URL)
이 등록 정보는 URL 스트링의 형태로 LDAP 서버의 이름과 포트를 식별합니다. LDAP 서버의 이름과 포트를 식별할 수 없으면 IBM JNDI LDAP 제공자는 ldap://localhost:389를 디폴트로 사용합니다.

다음의 코드는 호스트 ldapserver에 대한 연결을 작성하고 항목을 검색합니다.

예 1: 호스트 ldapserver로 연결 작성

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
     Properties env = new Properties();
     env.put("java.naming.factory.initial", "com.ibm.jndi.LDAPCtxFactory");
     env.put("java.naming.provider.url", "ldap://ldapserver");
DirContext ctx = new InitialDirContext(env);
     Attributes entry = ctx.getAttributes("cn=example,o=IBM,c=US");
```

JNDI를 사용하여 LDAP 서버에 대해 작업하는 두 번째 방법은 URL 스트링을 문맥의 메소드에 직접 전달하는 것입니다. 그러나 이 프로세스는 각 조작에 대한 새로운 연결을 작성하는 부담이 있으며 단일 서버에 대해 모든 조작을 바인드할 경우에는 피해야 합니다. 다음 등록 정보는 조작의 이 유형을 지원합니다:

java.naming.factory.url.pkgs(Context.URL_PKG_PREFIXES)

문맥의 메소드에 대한 이름 입력으로 URL 스트링 전달과 LDAP 서버로의 연결이 문맥 작성 시에 필요하지 않으면 이 등록 정보를 com.ibm.jndi로 설정해야 합니다.

다음의 코드는 이전 예와 중복되지만 getAttributes 메소드를 호출할 때까지 서버로의 연결이 지연됩니다.

예 2: 호스트 ldapserver에 대한 연결 작성 및 서버로의 연결 지연

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
     Properties env = new Properties();
     env.put("java.naming.factory.url.pkgs", "com.ibm.jndi");
DirContext ctx = new InitialDirContext(env);
     Attributes entry = ctx.getAttributes("ldap://ldapserver/cn=example,o=IBM,c=US");
```

제공자는 앞의 두 메소드 혼합도 지원합니다. 즉, java.naming.factory.initial을 사용하여 LDAP 서버에 연결을 설정하고 메소드에 대한 이름 입력으로 URL 스트링을 전달할 수 있습니다. java.naming.factory.url.pkgs의 정의 여부와 상관 없이 이러한 작업이 이루어집니다.

주: InitialDirContext에 의해 열린 LDAP 서버로의 연결은 close 메소드를 호출하여 닫아야 합니다.

LDAP V3 URL

IBM JNDI(Java(TM) Naming and Directory Interface) LDAP(Lightweight Directory Access Protocol) 제공자는 RFC 2255 에서 정의된 LDAP URL(Uniform Resource Locator) 형식을 완전히 지원합니다. LDAP URL은 다음의 구문을 사용하여 정의합니다.

```
scheme "://" [host [ ":" port ]] ["/"
    [dn ["?" [attributes] ["?" [scope]
    ["?" [filter] ["?" extensions]]]]]]
```

여기서,

- *scheme*은 URL 체계를 나타냅니다. 이 클래스 라이브러리는 일반 LDAP 연결에 대해서는 종래의 `ldap`을 지원하고 SSL(Secure Socket Layer) 연결에 대해서는 `ldaps`를 지원합니다.
- *host*는 LDAP 서버의 이름입니다. LDAP 서버명을 지정하지 않은 경우에 디폴트는 localhost입니다.
- *port*는 LDAP 서버의 포트 번호를 나타냅니다. 포트 번호를 지정하지 않았으면 디폴트는 비SSL의 경우에는 389이고 SSL의 경우에는 636입니다.
- *dn*은 조작에 대한 기본 오브젝트를 식별합니다.
- *attributes*는 리턴될 속성들을 쉼표로 구분한 리스트를 나타냅니다. 속성 리스트를 지정하지 않으면 디폴트는 모든 속성을 리턴하는 것입니다.
- *scope*은 탐색의 범위를 나타냅니다. 이 파일의 유효한 값은 다음과 같습니다.

- **base**
  이 값은 기본 오브젝트를 나타냅니다.
- **sub**
  이 값은 파일 계층의 하위 명령을 나타냅니다.
- **one**
  이 값은 파일 계층의 한 레벨을 나타냅니다.

범위를 지정하지 않으면 디폴트는 base입니다.

- *filter*는 탐색 필터를 나타냅니다. 탐색 필터를 지정하지 않으면 디폴트는 (objectclass=*)입니다.
- *extension*은 LDAP URL에 확장 메카니즘을 제공하여 URL의 기능을 확장할 수 있도록 합니다. IBM JNDI LDAP 제공자가 지원하는 확장은 bindname 뿐입니다.

서버 바인딩 및 SASL 지원

많은 경우에 서버는 특정 조작이 허용되기 전에 클라이언트를 인증해야 합니다. LDAP(Lightweight Directory Access Protocol)은 서버에 대한 바인딩으로 이를 언급합니다.

서버에 바인드할 때 클라이언트는 사용하려는 LDAP 프로토콜을 지정합니다. V2와 V3인 두 가지 LDAP 프로토콜 버전을 정의합니다. 서버가 V2만을 지원하면 클라이언트가 V3 클라이언트로 바인드하려고 시도할 때 프로토콜 오류가 리턴됩니다. IBM JNDI(Java(TM) Naming and Directory Interface) LDAP 제공자는 V2 또는 V3 클라이언트로서 바인딩을 제공합니다.

서버에 바인드할 때 다음의 등록 정보를 사용할 수 있습니다.

java.naming.ldap.version
이 등록 정보는 LDAP 프로토콜 버전을 지정합니다. 유효한 값은 2 또는 3입니다. 이 등록 정보를 설정하지 않으면 제공자는 V3 클라이언트로 바인드하려고 시도하고 프로토콜 오류가 리턴되면 자동으로 V2로 단계를 내립니다. 이 등록 정보를 설정하면 제공자는 단계를 내리려고 시도하지 않습니다.

프로토콜 버전의 설정 이외에 바인드는 인증을 위해 서버에 사용자를 식별합니다.

java.naming.security.principal(Context.SECURITY_PRINCIPAL)
이 등록 정보는 클라이언트 ID를 지정합니다. 거의 모든 경우에 식별 이름의 형태로 지정합니다.

java.naming.security.credentials(Context.SECURITY_CREDENTIALS)
이 등록 정보는 클라이언트의 증명서(즉, 클라이언트의 암호)를 지정합니다.

LDAP는 여러 가지 유형의 인증 메카니즘을 지원합니다. V2 LDAP 프로토콜은 간단한 바인드라고 하는 한 가지 바인드 유형만을 지원했습니다. 이 메카니즘을 사용하면 명확한 텍스트 ID 및 증명서를 서버에 보냅니다. V3 프로토콜은 SASL(Simple Authentication and Security Layer) 메카니즘도 지원하기 위해 인증을 확대했습니다. 이들 메카니즘을 사용하면 사용자의 ID와 암호를 명확한 텍스트로 보내어 사용자의 보안을 불필요하게 위협하지 않고 서버에 클라이언트를 식별하는 보다 복잡한 방법을 사용할 수 있습니다.

제공자는 두 가지 인증 메카니즘 지정 방법을 지원합니다. 한 방법에서는 인증 클래스의 이름이 필요합니다. 이 방법을 사용하면 제공자 외부의 인증 클래스를 지정하여 제공자를 확대할 수 있습니다. 따라서 사용자 자신의 SASL 플러그인을 작성할 수 있습니다. 다음의 등록 정보는 이러한 인증 메카니즘 지정 방법을 지원합니다.

java.naming.security.sasl

이 등록 정보는 사용할 인증 클래스의 이름을 지정합니다. 다음의 클래스가 제공자의 일부로 제공됩니다.

com.ibm.ldap.LDAPSimpleBind

이 등록 정보는 인증을 위해 서버에 보내는 명확한 텍스트 ID 및 증명서를 지정합니다. 이 메카니즘은 V2 및 V3 서버가 모두 지원합니다. 낮은 계층에서 인증 또는 암호화를 수행하지 않고 있을 때에는 개방 네트워크를 통한 명확한 텍스트 암호 사용이 권장되지 않는다는 점을 참고하십시오.

com.ibm.ldap.LDAPSaslExternal

외부 SASL 메소드는 SSL과 같이 이미 조정된 기초 보안 프로토콜을 사용하여 바인드하려고 시도합니다. 대부분의 경우에 보안 프린시펄과 증명서는 초기화하지 않은 상태로 두어야 합니다.

com.ibm.ldap.LDAPSaslCRAM_MD5

CRAM-MD5 SASL은 챌린지 응답 프로토콜을 사용하여 인증을 위해 서버에 보안 프린시펄과 증명서를 보냅니다.

com.ibm.ldap.LDAPSaslGSSAPI

GSSAPI SASL 메소드는 kinit 또는 통합 로그인과 같은 별도의 방법을 통해 증명서를 확보한 후에 Kerberos 인증을 사용하여 바인드하려고 시도합니다. 대부분의 경우에 보안 프린시펄 및 인증 바인드 인수는 초기화하지 않은 상태로 두어야 합니다.

java.naming.sasl.mode

이 등록 정보는 로드된 SASL 플러그인에 전달된 모드 설정을 지정합니다. 제공자에서 모든 사전정의 SASL 플러그인은 이 설정을 무시합니다.

두 번째 지원 방법은 인증 메카니즘을 Sun의 LDAP 툴킷과 호환할 수 있도록 지정하는 것입니다. 로드할 인증 클래스를 표시하는 대신 인증 메카니즘의 이름을 지정합니다. java.naming.security.sasl 등록 정보를 설정하지 않았으면 IBM JNDI LDAP 제공자는 인증 메카니즘을 지정할 때 이 방법을 사용합니다.

java.naming.security.authentication(Context.SECURITY_AUTHENTICATION)

이 등록 정보는 사용할 인증 메카니즘의 이름을 지정합니다. 이 등록 정보가 다음의 값을 지원합니다.

- none
  인증을 수행하지 않습니다(익명 바인드).
- simple
  간단한 인증을 사용합니다.
- EXTERNAL
  외부 SASL 메카니즘을 사용합니다.

- CRAM-MD5
  CRAM-MD5 SASL 메카니즘을 사용합니다.
- GSSAPI
  GSS 또는 Kerberos SASL 메카니즘을 사용합니다.

클라이언트가 성공적으로 인증된 후에 이 클래스 라이브러리가 다음의 등록 정보를 설정합니다.

java.naming.authorization.identity
이 등록 정보는 클라이언트의 권한 ID로 설정합니다. 일반적으로 지정된 클라이언트 ID와 동일합니다. 그러나 SASL 메카니즘은 초기 바인드 DN을 다른 값에 맵핑할 수 있습니다. 권한 ID가 클라이언트의 인증에 저장된 외부 SASL이 그 한 예입니다.

다음의 예는 버전 프로토콜이 3이고 CRAM-MD5 메카니즘을 사용하여 Larry Meade로 인증함을 나타내는 설정 등록 정보를 보여줍니다.

예: 등록 정보 설정

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
    Properties env = new Properties();
    env.put("java.naming.factory.initial", "com.ibm.jndi.LDAPCtxFactory");
    env.put("java.naming.ldap.version", "3");
    env.put("java.naming.provider.url", "ldap://ldapserver");
    env.put(Context.SECURITY_PRINCIPAL, "cn=Larry Meade, o=IBM, c=US");
    env.put(Context.SECURITY_CREDENTIALS, "secret");
    env.put(Context.SECURITY_AUTHENTICATION, "CRAM-MD5");
DirContext ctx = new InitialDirContext(env);
```

SASL 클래스명을 명시적으로 표시하도록 앞의 예를 변경할 수 있습니다. 앞의 예에서 SECURITY_AUTHENTICATION 행을 다음의 행으로 대체해야 합니다.

```
    env.put("java.naming.security.sasl", "com.ibm.ldap.LDAPSaslCRAM_MD5");
```

속성 탐색 및 확보

JNDI(Java(TM) Naming and Directory Interface)에서는 LDAP(Lightweight Directory Access Protocol) 디렉토리를 검색시 상당한 유연성을 제공합니다. IBM JNDI LDAP 제공자에서 가장 자주 사용하는 두 가지 메소드는 search와 getAttributes입니다. 그러나 다음의 메소드도 LDAP 서버에서 자료를 검색합니다.

- lookup
- lookupLink
- list
- listBindings
- getSchema
- getSchemaClassDefinition

다음 등록 정보는 탐색 조작에 영향을 줍니다.

java.naming.ldap.derefAliases

이 등록 정보는 별명 오브젝트(X.501에 정의됨)를 처리하는 방법을 정의합니다. 이 등록 정보는 다음 값을 갖습니다.

- always
  이 값은 탐색의 기본 오브젝트를 탐색하고 찾을 때 별명을 참조로 사용하지 않습니다. 이것이 디폴트입니다.
- never
  이 값은 탐색의 기본 오브젝트를 탐색하거나 찾을 때 별명을 참조로 사용하지 않습니다. 성능을 향상시키기 위해 이 설정은 권장되는 설정입니다.
- finding
  이 값은 탐색의 기본 오브젝트를 찾을 때에는 별명을 참조로 사용하지 않지만 기본 오브젝트의 종속 오브젝트를 찾을 때에는 참조로 사용합니다.
- searching
  이 값은 탐색 시에 기본 오브젝트의 종속 오브젝트에서 별명을 참조로 사용하지 않지만 탐색의 기본 오브젝트를 찾을 때에는 참조로 사용합니다.

java.naming.batchsize(Context.BATCHSIZE)

이 값은 리턴된 NamingEnumeration이 보유한 탐색 결과 수의 권장 크기 한계를 설정합니다. 값을 지정하지 않은 경우에 디폴트 batchsize는 1입니다. 이 값은 클래스 라이브러리가 가장 작은 메모리 흔적을 남길 수 있도록 합니다. 값 0은 batchsize를 작동 불가능하게 하고 모든 결과를 수집할 때까지 탐색이 정지됨을 나타냅니다.

java.naming.ldap.typesOnly

이 등록 정보는 getAttributes 및 search 메소드와 관련이 있으며 후자는 리턴하는 오브젝트 플래그가 false일 경우에만 가능합니다. 이 등록 정보는 다음 값을 갖습니다.

- true
  이 값은 값이 아닌 속성 ID만을 리턴합니다.
- false
  이 값은 속성 ID와 값을 모두 리턴합니다. 이것이 디폴트입니다.

탐색 호출의 결과는 NamingEnumeration입니다. 결과를 얻으려면 종래의 hasMoreElements 및 nextElement 메소드나 NamingEnumeration 특정 hasMore 및 next 메소드를 사용하여 계산을 진행해야 합니다. 후자의 두 메소드는 ReferralException을 추적하거나 보려는 경우에 예외를 포착할 수 있게 합니다.

주:

- 할당된 자원과 연결이 예상치 않게 계속 열려 있는 것을 방지하려면 NamingEnumeration을 끝까지 진행하거나(즉, hasMore 및 hasMoreElements 메소드가 false를 리턴할 때까지) 계산의 close 메소드를 호출해야 합니다.

- IBM JNDI LDAP 제공자는 식별 이름과 속성 유형 이름에서 유효하지 않은 UTF-8 문자 인코딩을 * 문자로 자동 대체합니다. 하나의 유효하지 않은 값으로 인해 전체 및 긴 탐색이 실패하는 것을 방지하기 위해 이러한 작업을 수행합니다.

다음의 예는 성이 smith인 모든 항목에 대한 탐색을 수행하고 cn 속성만의 리턴을 나타냅니다.

예: cn 속성 탐색 및 확보

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
     SearchControls constraints = new SearchControls();
constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
     String attrList[] = {"cn"};
constraints.setReturningAttributes(attrList);
NamingEnumeration results =
         ctx.search("o=IBM,c=US", "(sn=smith)", constraints);
while (results.hasMore()) {
    SearchResult si =(SearchResult)results.next();
    System.out.println(si.getName());
    Attributes attrs = si.getAttributes();
    if (attrs == null) {
             System.out.println("   No attributes");
        continue;
         }
    NamingEnumeration ae = attrs.getAll();
    while (ae.hasMoreElements()) {
        Attribute attr =(Attribute)ae.next();
        String id = attr.getID();
        Enumeration vals = attr.getAll();
        while (vals.hasMoreElements())
                 System.out.println("   "+id + ": " + vals.nextElement());
         }
     }
```

다음의 예는 list 메소드를 사용하여 기본 식별 이름(DN) 하에 이름을 표시합니다.

예: 기본 DN 하에 이름 표시

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
     String url="ldap://ldapserver:389/o=IBM,c=US";
NamingEnumeration listResults=ctx.list(url);
while (listResults.hasMore()) {
    NameClassPair ncp = (NameClassPair) listResults.next();
    System.out.println(ncp.getName());
     }
```

디렉토리에서 항목 추가 및 삭제

JNDI(Java(TM) Naming and Directory Interface)를 사용하면 디렉토리에서 항목을 추가하고 삭제할 수 있습니다. 다음의 예는 objectclass, roomnumber 및 telephonenumber 속성을 사용하여 새로운 항목을 추가합니다.

예: 디렉토리에 항목 추가

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
BasicAttribute objClasses = new BasicAttribute("objectclass");
objClasses.add("person");
objClasses.add("organizationalPerson");
objClasses.add("inetOrgPerson");

BasicAttributes attrs = new BasicAttributes();
attrs.put(objClasses);
attrs.put("roomnumber", "2000");
attrs.put("telephonenumber", "1-800-use-LDAP");

ctx.createSubcontext(name, attrs);
```

다음의 예는 항목 제거입니다.

예: 디렉토리에서 항목 삭제

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
ctx.destroySubcontext(name);
```

속성 변경: JNDI는 디렉토리 항목에서 속성을 변경, 작성 또는 제거할 수 있게 합니다. 다음의 예는 항목의 roomnumber 속성을 대체합니다.

예: 속성 변경

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
ctx.modifyAttributes(name,
    DirContext.REPLACE_ATTRIBUTE,
    new BasicAttributes("roomnumber", "5000"));
```

다음의 예는 새로운 telephonenumber 속성 값을 항목에 추가하고 roomnumber 속성을 제거합니다.

예: 디렉토리에서 항목 변경

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
ModificationItem[] mods=new ModificationItem[2];
mods[0] = new ModificationItem(DirContext.ADD_ATTRIBUTE,
    new BasicAttribute("telephonenumber", "456-7777"));
mods[1] = new ModificationItem(DirContext.REMOVE_ATTRIBUTE,
    new BasicAttribute("roomnumber));
ctx.modifyAttributes(name, mods);
```

디렉토리 항목 이름 변경: rename 메소드를 사용하여 기본 문맥과 관련된 곳이면 어느 곳에서나 디렉토리 항목의 이름을 변경할 수 있습니다.

다음의 등록 정보는 이름 변경 메소드에 영향을 줍니다.

java.naming.ldap.deleteRDN

이 등록 정보는 항목의 이름을 변경할 때 이전 RDN을 제거합니다. 디폴트 설정은 참입니다.

이 등록 정보를 false로 설정하면 이전 RDN을 항목의 속성 값으로 보유합니다.

다음은 이름 변경 메소드 호출의 예입니다.

예: 디렉토리 항목 이름 변경

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
String oldname="cn=bill smith";
String newname="cn=bill smith, ou=programmer";
ctx.rename(oldname, newname);
```

참조 및 탐색 참조

LDAP(Lightweight Directory Access Protocol) 서버는 참조 또는 탐색 참조를 리턴할 수 있습니다. 참조는 어느 조작에 대해서나 리턴할 수 있으며 서버가 요구의 목표 항목을 보유하지 않음을 나타냅니다. 탐색 참조는 탐색 조작에 대해서만 리턴됩니다. 탐색 참조는 서버가 baseObject가 참조한 항목을 찾을 수 있지만 baseObject 이하의 범위에서 모든 항목을 탐색할 수는 없음을 나타냅니다. 서버는 하나 이상의 탐색 참조를 리턴할 수 있습니다.

세 가지 방법 중 하나로 참조 및 탐색 참조를 처리하도록 문맥을 구성할 수 있습니다.

1. 참조를 자동으로 따르고 서버를 참조한 곳에서 조작을 수행하도록 설정할 수 있습니다. IBM JNDI(Java(TM) Naming and Directory Interface) LDAP 제공자는 참조 루프 즉, 체인에서 이전에 추적했던 포인트를 다시 가리키는 상황을 자동으로 인식하고 방지합니다.
2. 참조 또는 탐색 참조를 받았을 때 ReferralException을 유발하도록 설정할 수 있습니다. 예를 들어, 각 서버에 다른 바인딩이 필요한 경우와 같이 자동 처리가 다소 부족한 경우에 이러한 설정이 유용합니다.
3. 참조를 무시하고 아무 것도 발생하지 않은 것처럼 계속하도록 설정할 수 있습니다. 탐색 참조의 경우에는 시작 서버에서 발견한 항목만을 리턴함을 의미합니다.

다음의 환경 등록 정보는 IBM JNDI LDAP 제공자가 참조 및 탐색 참조를 처리하기 위해 정의합니다.

java.naming.referral(Context.REFERRAL)

이 등록 정보는 follow, throw 또는 ignore로 설정합니다. 이 등록 정보를 설정하지 않으면 참조를 자동으로 따르도록 설정하는 것이 디폴트입니다.

java.naming.ldap.referral.limit

이 등록 정보는 참조를 추적할 때 클래스 라이브러리가 작성하는 참조 홉의 수를 정의합니다. 등록 정보 값을 지정하지 않으면 디폴트는 10입니다.

java.naming.ldap.referral.bind

이 등록 정보를 true로 설정하면 참조를 자동으로 따를 때 시작 문맥과 동일한 SASL 메카니즘 및 증명서를 사용하여 참조된 서버에 클래스를 바인드합니다. false로 설정하면 클래스를 바인드하지 않습니다(즉, 익명 액세스). 디폴트 작동은 바인드입니다.

참조 또는 탐색 참조를 추적할 때 다음의 규칙을 적용합니다.

- 참조에 포트가 들어 있으면 이 포트를 사용합니다. 그렇지 않으면 1차 연결의 포트를 사용합니다.
- 1차 연결의 보안 연결 유형을 유지보수합니다. 즉, 1차 연결이 SSL을 통해 이루어진 경우에 추적한 모든 참조도 SSL을 통해 이루어집니다.

주: follow로 설정된 경우에도 문맥은 계속 ReferralException을 유발할 수 있습니다. 참조 홉 한계를 초과했거나 참조된 서버에 문맥을 연결 또는 바인드할 수 없는 경우에 이러한 상태가 발생할 수 있습니다.

다음의 예는 탐색 요구 시에 참조 및 탐색 참조를 포착하고 표시합니다. 참조 예외 처리에 대한 자세한 정보는, Sun Microsystems, Inc.의 ReferralException  를 참조하십시오.

예: 참조 및 탐색 참조 포착 및 표시

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
ctx.addToEnvironment(ctx.REFERRAL, "throw");
try {
    NamingEnumeration results = ctx.search(url);
    while (true) {
           try {
            if (!results.hasMore())
        break;
            SearchResult si =(SearchResult) results.next();
    System.out.println(si.getName());
        } catch (ReferralException re) {
            System.out.println("Reference caught");
    do {
        System.out.println(re.getReferralInfo());
    } while (re.skipReferral());
        }
    }
} catch (ReferralException re) {
    System.out.println("Referral caught");
    do {
        System.out.println(re.getReferralInfo());
    } while (re.skipReferral());
}
```

LDAP 제어

LDAP(Lightweight Directory Access Protocol) v3 스펙은 확장 정보를 송수신할 수 있도록 제어를 추가했습니다. 서버에 보낸 제어를 **요구 제어**라고 하며 지정된 속성에 따라 탐색 결과를 정렬하도록 서버에 지시하는 것을 예로 들 수 있습니다. 지원되는 요구 제어는 전적으로 서버에 따라 다릅니다(즉, 제어는 한 서버 유형에서 작동할 수 있지만 다른 서버 유형에서는 실패할 수 있습니다). 서버에서 받은 제어는 **응답 제어**라고 합니다.

JNDI(Java(TM) Naming and Directory Interface) 1.2 스펙은 2개의 별도 범주 즉, 서버로 연결될 때 사용되는 것과 다른 작업에 사용되는 것으로 요구 제어를 분리합니다. 요구 제어에 대한 자세한 정보는 Sun의 JNDI 문서에서 LdapContext를 참조하십시오.

IBM JNDI LDAP 제공자는 ManageDsaIT라고 하는 하나의 사전정의 제어를 제공합니다. 이 제어는 서버가 탐색 참조를 일반 LDAP 항목으로 취급하도록 강요하여 LDAP 항목이 참조하는 자료 대신 LDAP 항목을 보고 변경할 수 있도록 합니다. 다음의 예는 이 제어를 작동할 수 있는 방법을 보여줍니다.

예: ManageDsaIT 제어 작동

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
import com.ibm.jndi.ldap.control.ManageDsaIT;

Control[] cntl = new Control[1];
cntl[0] = new ManageDsaIT();
ctx.setRequestControls(cntl);
```

연결 제어는 클라이언트를 서버에 바인드할 때에만 활동 상태가 됩니다.

다음은 연결 제어를 작동할 수 있게 하는 방법이 예입니다.

```
LdapContext ctx = new InitialLdapContext(env, cntl);
```

다음의 API를 사용하여 마지막으로 받은 응답 제어를 검색합니다.

```
Control[] cntl = ctx.getResponseControls();
```

특정 제어 클래스에 일반 제어 자료를 맵핑할 수 있는 제어 팩토리 작동에 대한 세부사항은 Sun의 문서를 참조하십시오.

2진 속성

LDAP(Lightweight Directory Access Protocol) 프로토콜은 검색된 2진 및 텍스트 속성 간의 구별을 제공하지 않습니다. 그 대신 클라이언트 어플리케이션이 자료를 처리하는 방법을 알 것으로 예상합니다. IBM JNDI LDAP 제공자는 텍스트 속성을 변환하여 Java 스트링으로 리턴하는 데 도움이 됩니다. 그러나 제공자는 어느 속성이 2진이고 어느 속성이 문자 자료인지 알아야 합니다. 제공자는 검색된 속성에 발생한 상태의 처리 및 스트링으로 변환되었는지의 여부에 대해 세 가지 방법을 지원합니다.

속성을 검색할 때 제공자는 알려진 2진 속성명의 리스트를 검사합니다. 제공자는 다음의 공통 LDAP 2진 속성 세트를 인식하도록 프로그램되어 있습니다.

- userPassword
- userCertificate
- cACertificate
- authorityRevocationList
- certificateRevocationList
- deltaRevocationList
- crossCertificatePair
- x500UniqueIdentifier
- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedObject
- thumbnailPhoto
- thumbnailLogo
- supportedAlgorithms
- protocolInformation

다음의 등록 정보를 사용하여 자신의 2진 속성명 리스트를 지정할 수 있습니다.

java.naming.ldap.attributes.binary(LDAPCtx.ATTRIBUTES_BINARY)
제공자가 정의한 디폴트 세트 이외에 공백으로 구분된 사용자 정의 2진 속성명 리스트입니다.

다음 예는 두 개의 추가 사용자 정의 2진 속성을 식별합니다.

예: 2진 속성명 리스트 지정

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
ctx.addToEnvironment(LDAPCtx.ATTRIBUTES_BINARY,
    "gifPhoto fingerPrint");
```

2진 속성을 처리하기 위한 두 번째 방법은 일부 V3 서버가 지원하는 2진 설명 옵션을 인식하는 것입니다. ″jpegPhoto;binary″와 같은 속성명에 ″;binary″가 추가되었으면 2진 값이 지정됩니다. 제공자는 V3 사용자로 바인드될 때 2진 속성 설명 옵션을 인식합니다.

마지막으로 제공자는 2진으로 정의되지 않았으며 2진 속성 설명 옵션이 포함되지 않은 속성을 변환하려고 시도합니다. 변환이 실패하면 대신 자료가 2진으로 리턴됩니다. 그러나 비UTF-8 자료가 잘못 변환되었을 가능성

이 있기 때문에 이 작업에 의존해서는 안됩니다.

스키마

LDAP(Lightweight Directory Access Protocol) 서버의 스키마 구조를 검색하고 보고 갱신할 수 있습니다. LDAP(Lightweight Directory Access Protocol)(v3) 문서에 정의된 스키마 정보를 제공하는 서버만을 지원합니다.

서버의 스키마를 검색하려면 getSchema 메소드를 사용하십시오. 리턴된 스키마는 계층으로 표시하며 이 계층에서 각 종속 레벨은 스키마의 다른 구성요소입니다. IBM JNDI(Java(TM) Naming and Directory Interface) LDAP 제공자는 다음 스키마 구성요소를 분석하는 기능을 제공합니다.

- AttributeDefinition 하위 명령 아래에 저장된 AttributeTypes
- ClassDefinition 하위 명령 하에 저장된 오브젝트 클래스
- SyntaxDefinition 하위 명령 하에 저장된 구문 설명
- MatchingRule 하위 명령 하에 저장된 일치하는 규칙
- IBMAttributeDefinition 하위 명령 하에 저장된 IBM Attribute Types

IBMAttributeDefinition을 제외한 항목의 내용은 Lightweight Directory Access Protocol(v3): Attribute Syntax Definitions 에 정의된 스키마와 일대일로 일치합니다.

IBMAttributeDefinition은 IBM 특정 정보를 보유하도록 속성 스키마를 확장합니다. 다음의 BNF(Backus-Naur form)로 정의합니다.

```
IBMAttributeTypesDescription = "(" whsp
    numericoid whsp
    [ "DBNAME"   qdescrs ]            ; at most 2 names (table, column)
    [ "ACCESS-CLASS" whsp IBMAccessClass whsp ]
    [ "LENGTH" wlen whsp ]            ; maximum length of attribute
    [ "EQUALITY" [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "ORDERING" [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "APPROX"   [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "SUBSTR"   [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "REVERSE"  [ IBMwlen ] whsp ]   ; reverse index for substring
    whsp ")"

IBMAccessClass =
    "NORMAL"            / ; this is the default
    "SENSITIVE"         /
    "CRITICAL"          /
    "RESTRICTED"        /
    "SYSTEM"            /
    "OBJECT"

IBMwlen = whsp len
```

서버에서 리턴했지만 이 클래스 라이브러리가 지원하지 않는 스키마 정의를 저장하며 분석하지 않습니다. 이러한 항목에는 정확히 두 개의 속성이 있는데, 하나는 스키마 유형 이름과 동일한 오브젝트 클래스를 사용하는 것이고(예: objectclass=adddef) 다른 하나는 값 리스트를 사용하는 것입니다. 지원되지 않는 스키마 정의는 볼 수 있지만 갱신할 수 없습니다.

다음 예는 전체 스키마 계층을 검색합니다.

예: 스키마 계층 검색

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
DirContext schemaCtx = ctx.getSchema("");
SearchControls cons = new SearchControls();
cons.setSearchScope(SearchControls.SUBTREE_SCOPE);
NamingEnumeration ne = schemaCtx.search("",
    "(|(NUMERICOID=*)(objectclass=*))", cons);
```

다음 예는 cn 속성에 대한 스키마를 검색합니다:

예: cn 속성에 대한 스키마 검색

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
DirContext schemaCtx = ctx.getSchema("");
Attributes attrs = schemaCtx.getAttributes("AttributeDefinition/cn");
```

다음의 예는 새로운 오브젝트 클래스에 대한 스키마 정의 추가를 시도합니다.

예: 스키마 정의 추가

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
DirContext schemaCtx = ctx.getSchema("");
BasicAttributes attrs = new BasicAttributes();
attrs.put("NAME", "javaObject");
attrs.put("NUMERICOID", "1.3.6.1.4.1.42.2.27.4.2.2");
Attribute may = new BasicAttribute("MAY");
may.add("javaClassName");
may.add("javaSerializedObject");
attrs.put(may);
attrs.put("DESC", "Serialized Java object");
attrs.put("AUXILIARY", "true");
attrs.put("SUP", "top");
schemaCtx.createSubcontext("ClassDefinition/javaObject", attrs);
```

com.ibm.jndi.LDAPSchemaCtx 클래스는 특히 파일의 스키마 정의에 대해 작업하도록 확장되었습니다. 두 개의 공용 구성자는 디스크에서 스키마 정보 읽기를 지원합니다. 하나의 구성자는 단일 파일명을 인수로 사용하고 다른 구성자는 파일명의 배열을 인수로 사용합니다. dumpSchema 메소드는 스키마 정의를 파일에 저장합니다. 다음은 이 지원을 보여줍니다.

예: dumpSchema 메소드

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
LDAPSchemaCtx ctx = new LDAPSchemaCtx("schema.file");
ctx.dumpSchema("schema.sav");
```

SASL 플러그인

사용자는 자신의 SASL(Simple Authentication and Security Layer) 플러그인을 작성할 수 있습니다. SASL 플러그인은 추상 기본 클래스 LDAPSaslBind에서 비롯된 것이어야 합니다. bind 메소드를 구현해야 하며 이 메소드는 SendBindRequest 메소드를 호출하여 서버와 통신합니다. 다음은 간단한 바인드 플러그인의 예를 보여줍니다.

예: 바인드 플러그인

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
import java.io.IOException;
import com.ibm.asn1.ASN1Exception;
import com.ibm.ldap.*;

public class SimpleBind extends LDAPSaslBind
{
    public boolean bind(String dn, String credentials)
        throws IOException, ASN1Exception, LDAPException
        {
                return SendBindRequest("SIMPLE", dn, credentials);
        }

}
```

SASL 프로토콜에 서버 챌린지가 들어 있으면 SendBindRequest 메소드를 여러 번 호출하여 getServerCredentials 메소드를 통해 서버 챌린지 정보를 검색할 수 있게 해야 합니다. 다음은 CRAM-MD5 SASL 플러그인의 예를 보여줍니다.

예: CRAM-MD5 SASL 플러그인

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
import java.net.*;
import java.io.*;
import java.security.*;
import com.ibm.util.*;
import com.ibm.asn1.ASN1Exception;
import com.ibm.ldap.*;

/*

    Challenge-Response Authentication Mechanism / MD5 hash.
    See RFC 2195 ("IMAP/POP AUTHorize Extension for Simple
    Challenge/Response") and draft-ietf-ldapext-authmeth-02
    ("Authentication Methods for LDAP") for details.
```

```
*/
public class CramMD5 extends LDAPSaslBind
{
    public boolean bind(String dn, String credentials)
        throws IOException, ASN1Exception, LDAPException
    {
        String clientCreds;

        // Send initial bind request
        if (SendBindRequest("CRAM-MD5", dn, null) == true)
            return false;

        // Generate md5 hash from client's secret and server's
        // challenge and send to server
           try {
            clientCreds = "dn: " + new String(stringUTF(dn)) + " " +
                HMAC_MD5(credentials, getServerCredentials());
        } catch (NoSuchAlgorithmException nsae) {
            throw new IOException(nsae.toString());
        }
        putCredentials(clientCreds);
        return SendBindRequest();
    }

    /*

        Hashed Message Authentication Code. See RFC 2104
        ("HMAC: Keyed-Hashing for Message Authentication")
        for details.

    */
    public static String HMAC_MD5(String secret, String text)
        throws NoSuchAlgorithmException
    {
        MessageDigest md5;
        byte[] ipad, opad, key;
    int i;

        // Initialize
        md5 = MessageDigest.getInstance("MD5");
        ipad = new byte[64];
        opad = new byte[64];
        key = secret.getBytes();

        // If key is larger than block size then hash key
        if (key.length > 64)
        key = md5.digest(key);

        // Perform XOR of ipad and opad with key (padded to 64 bytes).
        for (i = 0; i < key.length; ++i) {
            ipad[i] = (byte)(0x36 ^ key[i]);
            opad[i] = (byte)(0x5c ^ key[i]);
        }
        while (i < 64) {
            ipad[i] = 0x36;
            opad[i++] = 0x5c;
        }
```

```
            // Hash ipad XOR result and text
            md5.update(ipad);
            key = md5.digest(text.getBytes());

            // Hash opad XOR result and previous hash
            md5.update(opad);
            key = md5.digest(key);

            // Return hex representation of hash (32 bytes)
            return Hex.toString(key, false);
        }
    }
```

클라이언트측 캐싱

캐싱은 최근에 요구한 정보를 로컬로 저장하는 방법을 제공합니다. 이미 확보한 정보에 대해 리모트 서버로 되돌아가는 대신 반복된 조회를 로컬로 검색하여 성능을 향상시킵니다. 반대로 캐시의 값이 검색 이후로 변경되었는지의 여부를 판별할 수 있는 방법이 없습니다. 인터넷 초안 A Simple Caching Scheme for Lightweight Directory Access Protocol(LDAP) and X.500 Directories는 항목이 실효되기 전에 캐시에 합리적으로 남아 있을 수 있는 시간을 나타내는 ″time-to-live″ 속성을 정의하여 이 문제를 언급합니다. 클라이언트측 캐싱의 구현으로 이 클래스 라이브러리가 이 초안을 완전히 지원합니다.

이 클래스 라이브러리에서 사용된 캐싱 알고리즘은 탐색 요청에 완전히 기초합니다. 자료는 로컬로 복제하지 않으므로 캐시에서 두 가지 방법으로 항목을 조회할 수 없습니다. 예를 들어, ″cn=Joe*″를 탐색한 후에 ″cn=Joe Smith″를 조회하면 결과가 ″cn=Joe*″ 결과 세트의 일부로서 로컬로 상주하더라도 캐시에서 값을 검색하지 않습니다. LDAP 자료의 복제에는 민감한 부분이 많으며, 특히 Java(TM)에서는 이 캐싱 체계의 범위를 벗어납니다. 최적화된 서버에서 결과를 검색하는 것보다 Java에서는 복잡한 조회를 로컬로 검색하는 데 시간이 오래 걸릴 수 있습니다.

빠르게 변경하는 자료나 중요한 정보를 검색할 때에는 캐싱을 사용해서는 안됩니다. 이 클래스 라이브러리에서는 디폴트로 캐싱이 작동 불가능하며 작동되면 캐시를 통과하고 서버에서 직접 결과를 검색하는 방법을 제공합니다. 그러나 LDAP 자료는 일반적으로 정적이고 종종 정보를 기반으로 하기 때문에 정확한 time-to-live 값을 사용한 캐싱은 많은 어플리케이션의 합리적인 모델입니다.

캐싱이 작동되면 서버에서 자료를 검색하는 조작에서 사용하며 다음의 메소드가 들어 있습니다.

- getAttributes
- search
- lookup
- lookupLink
- list

- listBindings
- getSchema
- getSchemaClassDefinition

유형이 LDAPCache인 오브젝트를 인스턴스화할 때 캐시가 작성됩니다. 여러 문맥이 단일 캐시를 공유할 수 있으며 한 문맥이 캐시에 넣은 값을 다른 문맥이 캐시에서 검색할 수 있음을 의미합니다. IBM JNDI LDAP 제공자에서의 캐싱도 스레드세이프입니다. LDAPCache의 다른 피처는 디스크에 일련화하고 나중에 복원할 수 있다는 점입니다. 검색한 항목 수에 기초하여 캐시의 크기를 변경할 수 있습니다.

문맥은 LDAPCache를 직접 사용하지 않지만 LDAPCacheControl에 의존하여 캐시를 관리합니다. LDAPCacheControl은 공유 캐시에 개별화된 보기를 제공합니다. LDAPCacheControl은 time-to-live 값과 같은 요구당 기준과 캐시를 바이패스하고 서버에서 직접 결과를 검색할 것인지에 따라 설정을 조정할 수 있게 합니다. LDAPCacheControl은 요구당 기준으로 사용할 수 있지만 여러 문맥이 공유할 수도 있습니다. 문맥에서의 LDAPCacheControl 참조는 lookup, lookupLink 및 listBindings 메소드에 의해 작성된 하위 문맥으로 전달됩니다.

java.naming.control.cache 등록 정보는 제공자가 캐시 제어를 문맥과 연관시키는 한 방법으로 정의합니다. 이러한 정의는 문맥 작성 시에 또는 나중에 addToEnvironment 메소드를 사용하여 가능합니다.

캐싱을 작동시키려면 유형이 LDAPCacheControl인 오브젝트를 인스턴스화하고 문맥과 연관시켜야 합니다. LDAPCacheControl 구성자에는 인수로 LDAPCache 오브젝트가 필요합니다. 다음은 캐싱을 작동시켜서 문맥을 작성하는 예입니다.

예: 캐싱이 작동할 수 있는 문맥 작성

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

```
import com.ibm.jndi.LDAPCtx;
import com.ibm.ldap.LDAPCache;
import com.ibm.ldap.LDAPCacheControl;

LDAPCache cache = new LDAPCache();
LDAPCacheControl cacheControl = new LDAPCacheControl(cache);
env.put("java.naming.control.cache", cacheControl);
DirContext ctx = new InitialDirContext(env);
```

요구 기준으로 설정을 조정할 수 있습니다. 설정은 지워질 때까지 계속 유효합니다. 다음은 time-to-live 값을 60초로 제한합니다.

```
cacheControl.putTTL(60);
cacheControl.putHonorServerTTL(false);
ctx.search(...);
```

다음은 캐시에 저장된 값을 바이패스하고 서버에서 직접 결과를 검색합니다.

```
cacheControl.putReadFlag(false);
ctx.search(...);
```

주: java.naming.batchsize 등록 정보로 인해 결과를 완전히 열거할 때까지 캐시에 항목이 추가됩니다.

IBMJNDI 클래스 버전 검색

LDAP(Lightweight Directory Access Protocol) 클래스의 버전은 다음의 정적 메소드를 사용하여 확보할 수 있습니다.

```
String version = com.ibm.jndi.LDAPCtx.getVersion();
```

준수 고려사항 및 추가 등록 정보

IBM JNDI(Java(TM) Naming and Directory Interface) LDAP 제공자와 Sun의 JNDI Implementor Guidelines for LDAP Service Providers(Draft 0.2) 사이에는 다음과 같은 알려진 차이점이 있습니다.

1. 명명 연합이 지원되지 않습니다.
2. SSL(Secure Socket Layer) 프로토콜의 사용은 iSeries 서버에서 IBM JNDI LDAP 제공자가 지원하지 않습니다.
3. IBM LDAP 제공자는 LDAP 디렉토리에서 일련화된 Java 오브젝트를 저장할 때 사용해서는 안됩니다.
4. 필요한 양 이상으로 URL 구성요소를 전달하면 ConfigurationException 또는 InvalidNameException이 발생하지 않습니다.
5. IBM JNDI LDAP 제공자는 자체의 SASL(Simple Authentication and Security Layer) 플러그인 지원을 가지고 있습니다. API는 현재 미리보기만 지원되는 패키지로 존재하기 때문에 Java SASL API(Application Programming Interface)를 지원하지 않습니다.
6. 제공자는 이름공간 또는 오브젝트 변경 이벤트가 아닌 자발적 통지 이벤트를 지원합니다.
7. java.naming.referral의 디폴트 값은 ignore가 아닌 follow입니다. 이 등록 정보가 ignore로 설정되었으면 제공자는 요구에 ManageDSAIt 제어를 자동으로 추가하지 않으며 참조를 받았을때 PartialResultException을 유발하지 않습니다.

다음 등록 정보도 지원됩니다.

com.ibm.jndi.ldap.so_timeout

이 등록 정보는 문맥 블록이 서버로부터 자료를 기다리는 밀리초 수를 정의합니다. 디폴트 시간종료는 5분입니다. 시간종료 값 0은 무한 시간종료 값으로 해석합니다.

다음의 등록 정보가 제거되었습니다.

- java.naming.ldap.noBind
- java.naming.control.server

JSSL

JSSL(Java Secure Socket Layer)은 보안 인터넷 통신을 가능하게 하는 Java 패키지 세트입니다. SSL 및 TLS(Transport Layer Security) 프로토콜의 Java 버전을 구현하며 자료 암호화, 서버 인증, 메세지 무결성 및 선택적 클라이언트 인증을 위한 기능이 들어 있습니다. JSSL을 사용하면 TCP/IP를 통해 어플리케이션 프로토콜(예: HTTP, Telnet, NNTP 및 FTP)을 실행 중인 클라이언트와 서버 사이에서 자료의 보안 이동을 위해 제공되는 어플리케이션을 개발할 수 있습니다.

JSSL에 대한 자세한 정보는 Sun Microsystems, Inc.의 Java Secure Socket Extension(JSSE)  을 참조하십시오.

JavaMail

JavaMail(TM) API는 전자(전자 우편) 시스템을 모델화하는 추상 클래스 세트를 제공합니다. API는 메일을 읽고 보내기 위해 일반 메일 기능을 제공하며 서비스 제공자가 프로토콜을 구현해야 합니다.

서비스 제공자는 특정 프로토콜을 구현합니다. 예를 들어, 단순 우편 전송 프로토콜(SMTP)은 전자 우편을 보내기 위한 전송 프로토콜입니다. POP3(Post Office Protocol 3)는 전자 우편을 받기 위한 표준 프로토콜입니다. IMAP(Internet Message Access Protocol)은 POP3의 대체 프로토콜입니다.

서비스 제공자 이외에 JavaMail에는 보통 텍스트가 아닌 메일 내용을 처리하기 위해 JAF(JavaBeans Activation Framework)가 필요합니다. 여기에는 MIME(Multipurpose Internet Mail Extensions), URL(Uniform Resource Locator) 페이지 및 파일 접속이 포함됩니다.

모든 JavaMail 구성요소는 IBM Developer Kit for Java의 일부로 제공됩니다. 이러한 구성요소는 다음과 같습니다.

- **mail.jar**
  이 JAR 파일에는 JavaMail API, SMTP 서비스 제공자, POP3 서비스 제공자 및 IMAP 서비스 제공자가 들어 있습니다.
- **activation.jar**
  이 JAR 파일에는 JavaBeans Activation Framework가 들어 있습니다.

자세한 정보는 Sun Microsystems, Inc. JavaMail  문서를 참조하십시오.

Java 인쇄 서비스

Java(TM) 인쇄 서비스 API를 사용하면 모든 Java 플랫폼에서 인쇄할 수 있습니다. Java 1.4는 Java 런타임 환경 및 써드 파티가 PDF, Postscript 및 AFP(Advanced Function Presentation)와 같이 다양한 인쇄 형식을 작성하기 위한 스트림 생성기 플러그인을 제공할 수 있는 구조를 제공합니다. 이러한 플러그인은 2차원(2D) 그래픽 호출에서 출력 형식을 생성합니다.

자세한 정보는 Sun Microsystems Java Print Service 문서를 참조하십시오.

제 8 장 IBM Developer Kit for Java를 사용한 프로그램 디버그

Java(TM) 프로그램을 디버그해야 하는 경우, 다음 옵션중 하나를 선택하십시오.

- Java 프로그램 디버그
- Java 및 원시 메소드 프로그램 디버그
- 다른 화면에서 Java 프로그램 디버그
- »

  사용자 정의 클래스 로더를 통해 로드한 Java 클래스 디버그

  «
- 서브릿 디버그

Java 프로그램을 디버그할 때 사용자의 Java 프로그램은 BCI(일괄처리) 작업으로 JVM(Java Virtual Machine)에서 실제로 실행됩니다. 소스 코드가 대화식 화면에 나타나지만 화면에서 Java 프로그램이 실행되지 않습니다. 즉 다른 작업에서 실행하며 이것이 서비스를 제공 받는 작업입니다. Java 프로그램이 종료할 때 서비스를 받는 작업이 종료되며 서비스 작업 종료 중이라는 메세지가 표시됩니다.

JIT(Just-In-Time) 컴파일러를 사용하여 실행 중인 Java 프로그램은 디버그할 수 없습니다. 파일에 연관된 Java 프로그램이 없으면 디폴트는 JIT를 실행하는 것입니다. 디버깅을 허용하기 위해 몇 가지 방법으로 이 컴파일러를 작동 불가능하게 할 수 있습니다.

- JVM(Java Virtual Machine)을 시작할 때 등록 정보 `java.compiler=NONE`을 지정하십시오.
- RUNJVA(Java 실행) 명령에 OPTION(*DEBUG)을 지정하십시오.
- Java 실행(RUNJVA) 명령에 INTERPRET(*YES)를 지정하십시오.
- JVM을 시작하기 전에 CRTJVAPGM OPTIMIZATION(10)을 사용하여 연관된 Java 프로그램을 작성하십시오.

  주: 이러한 해결책 중 어느 것도 실행 중인 JVM에 영향을 주지 않습니다. 이들 해결책 중 하나를 사용하여 JVM을 시작하지 않았으면 디버그를 위해 JVM을 중지하고 다시 시작해야 합니다.

RUNJVA(Java 실행) 명령에 *DEBUG 옵션을 지정하면 두 작업 사이에 인터페이스가 설정됩니다.

시스템 디버거에 대한 자세한 정보는 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  책 및 온라인 도움말 정보를 참조하십시오.

Java 프로그램 디버그

Java(TM) 프로그램을 디버그하는 데에는 여러 가지 다른 방법이 있습니다. *DEBUG 옵션을 사용하여 프로그램을 실행하기 전에 소스 코드를 볼 수 있습니다. 그런 다음, 프로그램이 실행되는 동안 중단점을 설정하거나 오류를 분석하기 위해 프로그램을 건너 뛰거나 프로그램 안으로 들어갈 수 있습니다.

Java 프로그램을 디버그하려면 이 단계를 따르십시오.

1. javac 툴에서 -g 옵션인 DEBUG 옵션을 사용하여 Java 프로그램을 컴파일하십시오. 자세한 정보는 *DEBUG 옵션을 사용하여 Java 프로그램 디버그를 참조하십시오.
2. iSeries 서버의 같은 디렉토리에 클래스 파일(.class)과 소스 파일(.java)을 삽입하십시오.
3. iSeries 명령행에서 RUNJVA(Java 실행) 명령을 사용하여 Java 프로그램을 실행하십시오. RUNJVA(Java 실행) 명령에 OPTION(*DEBUG)을 지정하십시오.

   주: 클래스만을 디버그할 수 있습니다. CLASS 키워드에 대해 JAR 파일명을 입력하면 OPTION(*DEBUG)이 지원되지 않습니다.
4. Java 프로그램 소스가 표시됩니다.
5. 중단점을 설정하려면 F6(중단점 추가/지우기) 키를 누르거나 프로그램을 이동하려면 F10(단계) 키를 누르십시오. 중단점 설정에 대한 자세한 내용은 중단점 설정을 참조하십시오. 이동에 대한 자세한 정보는 디버그하기 위해 Java 프로그램 이동을 참조하십시오.

추가 정보:

1. 중단점과 단계를 사용하는 동안 Java 프로그램의 논리 흐름을 체크한 후 필요에 따라 변수를 보고 변경하십시오.
2. RUNJVA 명령에서 OPTION(*DEBUG)을 사용하면 JIT(Just-In-Time) 컴파일러가 작동 불가능해집니다. 연관된 Java 프로그램이 없는 파일은 해석 모드에서 실행됩니다.

*DEBUG 옵션을 사용한 Java 프로그램 디버그

프로그램을 실행하기 전에 소스 코드를 보려면 *DEBUG 옵션을 사용하십시오. *DEBUG 옵션을 사용하면 코드안에 중단점을 설정할 수 있습니다.

*DEBUG 옵션을 사용하려면, 명령행에서 classfile의 이름과 OPTION(*DEBUG) 다음에 RUNJVA(Run Java(TM)) 명령을 실행하십시오. 예를 들어, iSeries 명령행은 다음과 같습니다.

```
RUNJVA CLASS(classname) OPTION(*DEBUG)
```

주: STRSRVJOB(서비스 작업 시작) 명령을 사용하기 위한 권한을 부여받지 않은 경우, OPTION(*DEBUG)이 무시됩니다.

디버깅 화면을 보려면 Java 프로그램에 대한 초기 디버깅 화면을 참조하십시오

Java 프로그램에 대한 초기 디버깅 표시장치

Java(TM) 프로그램을 디버그할 경우, 사용자 프로그램에 대해 다음의 화면 예를 따르십시오. 이 화면은 Hellod라는 프로그램 예를 보여줍니다.

- ADDENVVAR ENVVAR(CLASSPATH) VALUE('/MYDIR')을 입력하십시오.
- 명령을 입력하십시오(RUNJVA CLASS(HELLOD) OPTION(*DEBUG)). HELLOD 대신에 사용자 Java 프로그램명을 입력하십시오.
- 모듈 소스 표시 화면이 나타날 때까지 기다리십시오. 다음이 HELLOD Java 프로그램용 소스입니다.

```
+------------------------------------------------------------------------------+
|                            모듈 소스 표시                                    |
|                                                                              |
|  클래스 파일명:   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()                                                     |
|                                                                  계속...     |
|  디버그 . . .                                                                |
|                                                                              |
|  F3=프로그램 종료   F6=중단점 추가/지우기   F10=단계   F11=변수 표시         |
|  F12=재개     F17=변수 감시     F18=감시에 대한 작업     F24=추가 키         |
|                                                                              |
+------------------------------------------------------------------------------+
```

- F14(모듈 리스트에 대한 작업) 키를 누르십시오.
- 모듈 리스트에 대한 작업 화면이 표시됩니다. 옵션 1(프로그램 추가)을 사용하여 디버그할 다른 클래스와 프로그램을 추가할 수 있습니다. 옵션 5(모듈 소스 표시)를 사용하여 소스를 표시하십시오.

```
+------------------------------------------------------------------------------+
|                          모듈 리스트에 대한 작업                             |
|                                                  시스템:   AS400             |
|  옵션을 입력한 후 Enter 키를 누르십시오.                                     |
|    1=프로그램 추가   4=프로그램 제거   5= 모듈 소스 표시                     |
|    8=모듈 중단점에 대한 작업                                                 |
|                                                                              |
|  Opt     프로그램/모듈      라이브러리       유형                             |
|                             *LIBL            *SRVPGM                         |
|          HELLOD                              *CLASS      선택됨              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                  맨 아래     |
```

```
| 명령                                                                          |
| ===>                                                                          |
| F3=나감    F4=프롬트    F5=화면정리    F9=검색    F12=취소                     |
| F22=클래스 파일명 표시                                                        |
|                                                                               |
+-------------------------------------------------------------------------------+
```

- 디버그할 클래스를 추가할 때 프로그램/모듈 입력 필드보다 긴 패키지 규정 클래스명을 입력해야 합니다. 더 긴 이름을 입력하려면 다음 단계를 따르십시오.
  1. 옵션 1(프로그램 추가)을 입력하십시오.
  2. 프로그램/모듈 필드를 공백으로 두십시오.
  3. 라이브러리 필드를 *LIBL로 두십시오.
  4. 유형에 *CLASS를 입력하십시오.
  5. Enter 키를 누르십시오.
  6. 패키지 규정 클래스 파일명을 입력할 더 많은 공간이 있는 팝업 창이 표시됩니다.

중단점 설정

중단점을 사용하여 프로그램 실행을 제어할 수 있습니다. 중단점은 특정 명령문에서 실행 중인 프로그램을 중단시킵니다.

중단점을 설정하려면 다음 단계를 수행하십시오.

1. 중단점을 설정하려는 코드 행에 커서를 놓으십시오.
2. 중단점을 설정하려면 F6(중단점 추가/지우기) 키를 누르십시오.
3. 프로그램을 실행하려면 F12(재개) 키를 누르십시오.

주: 중단점이 설정되어 있는 코드 행을 실행하기 바로 전에 중단점에 도달했음을 나타내는 프로그램 소스가 표시됩니다.

```
+-------------------------------------------------------------------------------+
|                            모듈 소스 표시                                     |
|                                                                               |
|현재 스레드:   00000019     중단 스레드:  00000019                             |
|클래스 파일명:   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();                                                           |
```

```
|디버그 . . .                                                                  |
|                                                                              |
|F3=프로그램 종료   F6=중단점 추가/지우기   F10=단계   F11=변수 표시           |
|F12=재개   F17=변수 감시   F18=감시에 대한 작업   F24=추가 키                 |
|41 행에 중단점이 추가됨.                                                      |
+------------------------------------------------------------------------------+
```

중단점을 만날 때 현재 스레드 내에서만 만난 중단점을 설정하려는 경우 TBREAK 명령을 사용하십시오.

시스템 디버거 명령에 대한 자세한 정보는 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  책 및 온라인 도움말 정보를 참조하십시오.

중단점에서 프로그램 실행을 중단할 때의 변수 평가에 대해서는 Java(TM) 프로그램에서 변수 평가를 참조하십시오.

디버그를 위한 Java 프로그램에서의 이동

디버그하는 동안 프로그램 안에서 이동할 수 있습니다. 다른 함수로 건너 뛰거나 들어갈 수 있습니다. Java(TM) 프로그램과 원시 메소드는 이동 기능을 사용할 수 있습니다.

프로그램 소스가 처음 표시될 때 이동을 시작할 수 있습니다. 첫 번째 명령문을 실행하기 전에 프로그램이 중단합니다. F10(이동) 키를 누르십시오. 프로그램 안에서 이동하려면 F10(단계) 키를 계속 누르십시오. 프로그램이 호출하는 함수 안으로 진입하려면 F22(단계 진입) 키를 누르십시오. 중단점을 만날 때마다 이동(stepping)을 시작할 수 있습니다. 중단점 설정에 대한 자세한 정보는 중단점 설정을 참조하십시오.

```
+------------------------------------------------------------------------------+
|                            모듈 소스 표시                                    |
|                                                                              |
|현재 스레드:   00000019     중단 스레드:  00000019                            |
|클래스 파일명:   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();                                                          |
|디버그 . . .                                                                  |
|                                                                              |
|F3=프로그램 종료   F6=중단점 추가/지우기   F10=단계   F11=변수 표시           |
|F12=재개   F17=변수 감시   F18=감시에 대한 작업   F24=추가 키                 |
| 스레드 00000019의 42 행에서 단계가 완성되었습니다.                           |
+------------------------------------------------------------------------------+
```

이동(stepping)을 중단하고 프로그램을 계속 실행하려면 F12(재개) 키를 누르십시오.

이동(stepping)에 대한 자세한 정보는 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  책 및 온라인 도움말 정보를 참조하십시오.

한 단계에서 프로그램 실행이 중단될 때의 변수 평가에 대한 자세한 내용은 Java 프로그램의 변수 평가를 참조하십시오.

Java 프로그램의 변수 평가

중단점이나 단계에서 프로그램 실행이 중단될 때 변수를 평가하는 방법에는 두 가지가 있습니다.

- 디버그 명령 행에 `EVAL VariableName`을 입력합니다.
- 표시된 소스 코드의 변수명에 커서를 놓고 F11(변수 표시) 키를 누릅니다.

Java(TM) 프로그램에서 변수를 평가하는 EVAL 명령을 사용하십시오.

주: EVAL 명령을 사용하여 변수 내용을 변경할 수도 있습니다. EVAL 명령의 변화에 대해서는 WebSphere Development Studio: ILE C/C++ Programmer's Guide, SC09-2712  책 및 온라인 도움말 정보를 참조하십시오.

Java 프로그램에서 변수를 찾을 때 다음을 주의하십시오.

- Java 클래스의 인스턴스인 변수를 평가할 경우 화면의 첫 번째 행은 오브젝트의 종류를 나타냅니다. 또한 오브젝트의 ID도 보여줍니다. 첫 번째 행 다음에 오브젝트에 있는 각 필드의 내용이 표시됩니다. 변수가 널(null)인 경우 화면의 첫 번째 행이 변수가 널임을 나타냅니다. 별표(*)는 각 필드(널 오브젝트) 내용을 보여줍니다.
- Java 스트링 오브젝트인 변수를 평가하는 경우 해당 스트링의 내용이 표시됩니다. 스트링이 널인 경우 널이 표시됩니다.
- 스트링인 변수를 변경할 수 없습니다.
- 배열인 변수를 평가하는 경우 해당 배열 ID 뒤에 'ARR'이 표시됩니다. 변수명의 문자를 사용하여 배열 요소를 평가할 수 있습니다. 배열이 널인 경우 널이 표시됩니다.
- 배열인 변수를 변경할 수 없습니다. 스트링이나 오브젝트의 배열이 아닌 경우 배열의 요소는 변경할 수 있습니다.
- 배열인 변수의 경우 배열에 있는 요소의 수를 알아 보려면 arrayname.length를 지정할 수 있습니다.
- 클래스의 필드인 변수 내용을 보려는 경우, classvariable.fieldname을 지정할 수 있습니다.
- 초기설정 이전에 변수를 평가하는 경우, 다음 두 가지 상황 중에 하나가 발생할 수 있습니다. 변수를 표시할 수 없음 메세지가 표시되거나 이상한 값으로 보이는 초기 설정되지 않은 변수가 표시됩니다.

Java 및 원시 메소드 프로그램 디버그

Java(TM) 프로그램과 원시 메소드 프로그램을 동시에 디버그할 수 있습니다. 대화식 화면에서 소스를 디버그할 때 서비스 프로그램(*SRVPGM) 내에 있는 C 프로그램으로 이루어진 원시 메소드를 디버그할 수 있습니다.

»

*SRVPGM은 디버그 자료를 사용하여 컴파일하고 작성해야 합니다.

«

Java 프로그램과 원시 메소드 프로그램을 동시에 디버그하려면 다음과 같이 하십시오.

1. 모듈 리스트에 대한 작업(WRKMODLST) 화면을 표시하려면 Java 프로그램 소스가 표시될 때 F14(모듈 리스트에 대한 작업) 키를 누르십시오.
2. 서비스 프로그램을 추가하려면 옵션 1(프로그램 추가)을 선택하십시오.
3. 옵션 5(표시 화면 모듈 소스)를 선택하여 디버그 하려는 *MODULE과 소스를 표시 화면에 나타내십시오.
4. F6(중단점 추가/지우기) 키를 눌러 서비스 프로그램에 중단점을 설정하십시오. 중단점 설정에 대한 자세한 내용은 중단점 설정을 참조하십시오.
5. 프로그램을 실행하려면 F12(재개) 키를 누르십시오.

주: 서비스 프로그램에서 중단점을 만나면 프로그램은 실행을 중단하고 서비스 프로그램 소스를 표시합니다.

다른 화면에서 Java 프로그램 디버그

Java(TM) 프로그램을 디버깅할 때, 중단점을 만날 때 마다 프로그램 소스가 표시됩니다. 인터페이스는 Java 프로그램의 화면 출력에 영향을 줄 수 있습니다. 이것을 피하려면 다른 화면에서 Java 프로그램을 디버그하십시오. Java 프로그램의 출력은 Java 명령이 실행되는 곳에 표시되고 프로그램 소스는 다른 화면에서 나타납니다.

이미 실행 중인 Java 프로그램이 JIT(Just-In-Time) 컴파일러를 사용하고 있지 않으면 이 Java 프로그램을 디버그할 수도 있습니다.

다른 화면에서 Java를 디버그하려면 다음과 같이 하십시오.

1. Java 프로그램은 디버그 설정을 시작하는 동안 보류시켜야 합니다. 프로그램을 다음 상태로 만들어서 Java 프로그램을 보류시킬 수 있습니다.
   - 키보드의 입력을 대기합니다.
   - 시간 간격 동안 대기합니다.
   - 변수를 테스트하기 위해 루프 처리를 합니다. 이것은 Java 프로그램이 결국에는 루프에서 벗어날 값을 설정해야 한다는 것을 의미합니다.
2. 일단 Java 프로그램이 보류되면 다른 화면으로 가서 다음 단계를 수행하십시오.
   a. 명령 행에 WRKACTJOB(활동 작업에 대한 작업) 명령을 입력하십시오.

b. Java 프로그램을 실행 중인 일괄처리 즉시(BCI) 작업을 찾으십시오. 서브시스템/작업 리스트에서 QJVACMDSRV를 찾으십시오. 사용자 리스트에서 사용자 ID를 찾으십시오. BCI 유형에서 찾으십시오.

c. 해당 작업에 대해 작업하려면 옵션 5를 입력하십시오.

d. 작업(Job)에 대한 작업 화면의 맨 위에 번호, 사용자 및 작업이 표시됩니다. STRSRVJOB Number/User/Job을 입력하십시오.

e. STRDBG CLASS(classname)를 입력하십시오. Classname은 디버그하려는 Java 클래스명입니다. 클래스명은 Java 명령에 지정한 클래스명이거나 다른 클래스명일 수 있습니다.

f. 해당 클래스의 소스가 모듈 소스 표시 화면에 나타납니다.

g. Java 클래스에서 중단하려고 할 때마다 F6(중단점 추가/지우기) 키를 눌러 중단점을 설정하십시오. 디버그할 다른 클래스, 프로그램 또는 서비스 프로그램을 추가하려면 F14 키를 누르십시오. 중단점 설정에 대한 자세한 내용은 중단점 설정을 참조하십시오.

h. 프로그램을 계속 실행하려면 F12(재개) 키를 누르십시오.

3. 기본 Java 프로그램의 보류를 중단하십시오. 중단점을 만나면 모듈 소스 표시 화면은 STRSRVJOB(서비스 작업 시작) 명령과 STRDBG(디버그 시작) 명령이 입력되었던 화면에 나타납니다. Java 프로그램이 끝날 때 서비스 작업이 종료됨 메세지가 나타납니다.
4. ENDDBG(디버그 종료) 명령을 입력하십시오.
5. ENDSRVJOB(서비스 작업 종료) 명령을 입력하십시오.

주: 원래의 작업에서 JVM(Java Virtual Machine)을 시작할 때 JIT(Just-In-Time)를 작동 불가능하게 했는지 확인하십시오. 이 작업은 java.compiler=NONE 등록 정보를 사용하여 수행할 수 있습니다. 디버그하는 동안 JIT를 실행하면 예상하지 못한 결과가 발생할 수 있습니다.

JVM(Java Virtual Machine)을 호출하기 전에 BCI 작업 대기 여부를 제어하는 변수에 대한 자세한 내용은 QIBM_CHILD_JOB_SNDINQMSG 환경 변수를 참조하십시오.

QIBM_CHILD_JOB_SNDINQMSG 환경 변수

QIBM_CHILD_JOB_SNDINQMSG 환경 변수는 JVM(Java(TM) virtual machine)이 실행하는 일괄처리 즉시(BCI) 작업이 JVM을 시작할 때까지 대기하는 지를 제어하는 변수입니다.

RUNJVA(Java 실행) 명령을 실행할 때 환경 변수를 1로 설정하면 사용자의 메세지 대기행렬로 메세지가 전송됩니다. 메세지는 BCI 작업에서 JVM(Java Virtual Machine)이 시작되기 전에 전송됩니다. 메세지는 다음과 같습니다.

```
생성된 (하위) 프로세스 023173/JOB/QJVACMDSRV가 중단됨(G C)
```

이 메세지를 보려면 SYSREQ를 입력한 후 옵션 4를 선택하십시오.

이 메세지에 응답을 입력할 때까지 BCI 작업이 대기합니다. (G) 응답은 JVM(Java Virtual Machine)을 시작합니다.

메세지에 응답하기 전에 BCI 작업이 호출하는 *SRVPGM 또는 *PGM에 중단점을 설정할 수 있습니다.

주: 현재 JVM(Java Virtual Machine)이 시작되지 않았기 때문에 Java 클래스에 중단점을 설정할 수 없습니다.

사용자 정의 클래스 로더를 통해 로드한 Java 클래스 디버그

사용자 정의 클래스 로더를 통해 로드한 클래스를 디버그하려면 다음의 단계를 수행하십시오.

1. DEBUGSOURCEPATH 환경 변수를 소스 코드가 포함된 디렉토리로 설정하거나 패키지 규정 클래스의 경우에는 패키지명의 시작 디렉토리로 설정하십시오.

   예를 들어, 사용자 정의 클래스 로더가 /MYDIR 디렉토리 하에 있는 클래스를 로드한 경우에는 다음을 수행하십시오.

   ```
   ADDENVVAR ENVVAR(DEBUGSOURCEPATH) VALUE('/MYDIR')
   ```

2. 모듈 소스 표시 화면에서 디버그 보기에 클래스를 추가하십시오.

   클래스가 이미 JVM(Java(TM) virtual machine)에 로드되었으면, 보통의 경우와 같이 *CLASS를 추가하고 디버그할 소스 코드를 표시하십시오.

   예를 들어, pkg1/test14.class의 소스를 보려면 다음을 입력하십시오.

   ```
        Opt     Program/module     Library        Type
   1      pkg1.test14_       *LIBL          *CLASS
   ```

   JVM에 클래스가 로드되지 않았으면 동일한 단계를 수행하여 앞에 표시된 대로 *CLASS를 추가하십시오. **Java 클래스 파일을 사용할 수 없습니다.** 메세지가 표시됩니다. 이 때 프로그램 처리를 재개할 수 있습니다. 제공된 이름과 일치하는 클래스의 메소드를 입력하면 JVM이 자동으로 중지됩니다. 클래스의 소스 코드가 표시되므로 디버그할 수 있습니다.

서브릿 디버그

서브릿 디버그는 사용자 정의 클래스 로더를 통해 로드한 클래스를 디버그하는 특수 경우입니다. 서브릿은 IBM HTTP Server의 Java(TM) 런타임에서 실행됩니다. 서브릿을 디버그하기 위한 한 가지 방법은 사용자 정의 클래스 로더를 통해 로드한 클래스에 대한 지침을 따르는 것입니다.

서브릿을 디버그하는 또 다른 방법은 다음과 같습니다.

1. Qshell 인터프리터 내의 `javac -g` 명령을 사용하여 서브릿을 컴파일하십시오.

2. 소스 코드(.java 파일)와 컴파일 코드(.class 파일)를 /QIBM/ProdData/Java400에 복사하십시오.
3. Java 작성 프로그램(CRTJVAPGM) 명령을 최적화 레벨 10, OPTIMIZE(10)를 사용하는 .class 파일에 대해 실행하십시오.
4. 서버를 시작하십시오.
5. 서비스 시작 작업(STRSRVJOB) 명령을 서브릿이 실행할 작업에서 실행하십시오.
6. STRDBG CLASS(myServlet)를 입력하십시오. 여기서 myServlet이 서브릿의 이름입니다. 소스가 표시 화면에 나타나야 합니다.
7. 서브릿에서 중단점을 설정하고 F12 키를 누르십시오.
8. 서브릿을 실행하십시오. 서브릿이 중단점과 일치할 때 디버깅을 계속할 수 있습니다.

JPDA(Java Platform Debugger Architecture)

JPDA(Java(TM) Platform Debugger Architecture)는 다음 세 파트로 구성됩니다.

- 『JVMDI(Java Virtual Machine Debug Interface)』
- 513 페이지의 『JDWP(Java Debug Wire Protocol)』
- 513 페이지의 『JDI(Java Debug Interface)』

JPDA의 세 부분은 모두 JDWP를 사용하여 디버깅 조작을 수행하는 디버거의 프론트 엔드를 가능하게 합니다. 디버거 프론트 엔드는 리모트로 실행하거나 iSeries 어플리케이션으로 실행할 수 있습니다.

JVMDI(Java Virtual Machine Debug Interface)

Java(TM) 2 SDK(J2SDK), 표준판 버전 1.2 이상에서, JVMDI(Java Virtual Machine Debug Interface)는 Sun Microsystems, Inc. 플랫폼 API의 일부입니다. JVMDI를 사용하면 누구나 iSeries C 코드로 iSeries 서버에 대한 Java 디버거를 작성할 수 있습니다. 디버거는 JVM이 JVMDI 인터페이스를 사용하므로 내부 구조를 알 필요가 없습니다. JVMDI는 JVM에 가장 가까운 JPDA의 최저 레벨 인터페이스입니다.

디버거는 JVM과 동일한 멀티스레드 가능한 작업에서 실행합니다. 디버거는 JNI(Java Native Interface) 호출 API를 사용하여 JVM을 작성합니다. 그런 다음 사용자 클래스 원시 메소드의 시작에 후크를 배치하고 원시 메소드를 호출합니다. 원시 메소드가 시작할 때 후크가 일치되고 디버깅이 시작합니다. 중단점 설정, 이동(stepping), 변수 표시 및 변수 변경과 같은 일반적인 디버그 기능을 사용할 수 있습니다.

디버거는 JVM이 실행 중인 작업과 사용자 인터페이스를 처리하는 작업 사이의 통신을 처리합니다. 이 사용자 인터페이스는 iSeries 서버나 또 다른 시스템 중 하나에 있습니다.

QSYS 라이브러리에 상주하는 QJVAJVMDI라는 서비스 프로그램이 JVMDI 기능을 지원합니다.

JDWP(Java Debug Wire Protocol)

JDWP(Java Debug Wire Protocol)는 디버거 프로세스와 JVMDI 사이의 정의된 통신 프로토콜입니다. JDWP는 리모트 시스템에서 또는 로컬 소켓을 통해 사용할 수 있습니다. JVMDI에서 제거된 한 계층이지만 보다 복잡한 인터페이스입니다.

QShell에서 JDWP 시작

JDWP를 시작하고 Java 클래스 SomeClass를 실행하려면 QShell에 다음의 명령을 입력하십시오.

```
    java -interpret -Xrunjdwp:transport=dt_socket,
address=8000,server=y,suspend=n SomeClass
```

이 예에서 JDWP는 TCP/IP 포트 8000에서 리모트 디버거로부터의 연결을 청취하지만 원하는 아무 포트 번호나 사용할 수 있습니다. dt_socket은 JDWP 전송을 처리하는 SRVPGM의 이름이며 변경되지 않습니다.

-Xrunjdwp와 함께 사용할 수 있는 추가 옵션은 Sun Microsystems, Inc.의 Sun VM Invocation Options 를 참조하십시오.

CL 명령행에서 JDWP 시작

CL 명령과 함께 -Xrun 옵션을 사용하려면 os400.xrun.option 등록 정보를 QShell 명령행에서 사용했던 스트링과 동일하게 정의하면 됩니다. JDWP를 시작하고 Java 클래스 SomeClass를 실행하려면 다음의 명령을 입력하십시오.

```
JAVA CLASS(SomeClass) INTERPRET(*YES)
PROP((os400.xrun.option 'jdwp:transport=dt_socket,address=8000,
                                        server=y,suspend=n'))
```

많은 JVMDI 기능은 최적화 레벨 10과 20에서 작동하지 않습니다. 따라서 모든 기능을 작동시키려면 인터프리터를 사용하여 어플리케이션을 실행하는 것이 좋습니다.

JDI(Java Debug Interface)

JDI(Java Debug Interface)는 툴 개발을 위해 제공되는 상위 레벨 Java 언어 인터페이스입니다. JDI는 JVMDI 및 JDWP의 복잡성을 일부 Java 클래스 정의 뒤로 숨깁니다. JDI는 rt.jar 파일에 포함되어 있으므로 디버거의 프론트 엔드는 Java를 설치한 어느 플랫폼에나 존재합니다.

Java를 위한 디버거를 작성하려면 JDI를 사용해야 합니다. JDI는 가장 간단한 인터페이스이며 코드는 플랫폼과 상관없기 때문입니다.

JDPA에 대한 자세한 정보는 Sun Microsystems, Inc.의 Java Platform Debugger Architecture Overview 를 참조하십시오.

메모리 누출 찾기

ANZJVM은 지정된 시간 간격으로 분리된 가비지 콜렉션 힙에 대해 두 개의 사본을 작성하여 오브젝트 유출을 찾습니다. 오브젝트 유출을 찾기 위해 힙에서 각 클래스의 인스턴스 수를 확인합니다. 특히 많은 인스턴스가 있는 클래스는 유출 가능성이 있는 것으로 주목해야 합니다.

가비지 콜렉션 힙의 두 사본 사이에 각 클래스의 인스턴스 수가 변경되었음을 확인해야 합니다. 클래스의 인스턴스 수가 계속 증가하면 이 클래스는 유출 가능성이 있음을 주목해야 합니다. 두 사본 사이의 시간 간격이 길수록 오브젝트가 실제로 유출되고 있음이 확실해집니다. 보다 큰 시간 간격으로 연속해서 ANZJVM을 실행하면 보다 확실하게 유출되고 있는 것을 진단할 수 있습니다.

제 9 장 IBM Developer Kit for Java 문제 해결

IBM Developer Kit for Java(TM)를 사용하는 도중 문제점을 발견하면, 다음 단계를 수행하여 문제점의 원인을 판별합니다.

- »

  IBM Developer Kit for Java를 사용할 때 제한사항이 있음을 알 수 있습니다. 이 주제는 모든 알려진 한계, 제한 사항 또는 고유 작동 방식을 식별합니다.

  «

- Java 명령을 실행한 작업에서 작업기록부 찾기. 또한 장애의 원인을 분석하기 위해 Java 프로그램이 실행된 일괄처리 즉시(BCI) 작업에서도 작업 기록부를 찾으십시오.
- APAR(공식 프로그램 결함 수정 의뢰서)에 대한 유용한 자료를 수집하십시오.
- PTF(프로그램 임시 수정)를 적용하십시오.
- IBM Developer Kit for Java에서 잠재적인 결함을 감지한 경우 지원 확보 방법을 알아두십시오.

제한사항

IBM Developer Kit for Java(TM) 사용시, 사용법에 일부 제한이 있음을 알 수 있습니다. 이 리스트는 모든 알려진 한계, 제한사항 또는 고유 작동 방식을 식별합니다.

- 클래스가 로드되고 그의 수퍼클래스를 모르는 경우, 원래 클래스가 없었음을 오류가 표시합니다. 예를 들어, 클래스 B가 클래스 A를 확장하고, 클래스 B를 로드할 때 클래스 A가 없는 경우 발견되지 않은 것이 실제로는 클래스 A인 경우에도 오류는 클래스 B가 없었다고 표시합니다. 클래스가 없었음을 표시하는 오류를 볼 때 클래스 및 그의 모든 수퍼클래스가 CLASSPATH에 있는지 확인하십시오. 이것은 또한 로드될 클래스가 수행하는 인터페이스에도 적용됩니다.
- »

  가비지 콜렉션 힙은 132GB로 제한됩니다.
- 구축된 오브젝트 수는 132가 한계입니다.

  «

- iSeries 서버의 java.net backlog 매개변수는 다른 플랫폼에서와 다르게 활동할 수 있습니다. 예를 들면 다음과 같습니다.
  - Listen backlogs 0, 1
    - Listen(0)은 하나의 지연중 조건을 허용함을 의미합니다. 즉, 소켓을 작동 불가능하게 하지 않습니다.
    - Listen(1)은 하나의 지연중 주석을 허용함을 의미하며, Listen(0)과 같음을 의미합니다.
  - Listen backlogs > 1

- 많은 미결 요구가 청취 대기행렬에 남아 있도록 허용합니다. 새로운 연결 요구가 도착하고 대기행렬이 한계에 있는 경우 요구는 미결 요구 중 하나를 삭제합니다.

- 

  사용하고 있는 JDK 버전과 상관 없이 멀티 스레드가 가능한(즉, 스레드세이프) 환경에서 JVM(Java Virtual Machine)만을 사용할 수 있습니다. iSeries 서버는 스레드세이프이지만 일부 파일 시스템은 그렇지 않습니다. 통합 파일 시스템 주제에는 스레드세이프가 아닌 파일 시스템 리스트가 있습니다.

Java 문제점 분석을 위한 작업 기록부 찾기

Java(TM) 명령을 실행한 작업의 작업 기록부 및 Java 프로그램이 실행된 일괄처리 즉시 (BCI) 작업 기록부를 사용하여 Java 실패의 원인을 분석하십시오. 둘다 중요한 오류 정보를 포함할 수 있습니다.

BCI 작업에 대한 작업 기록부를 찾는 두 가지 방법이 있습니다. Java 명령을 실행한 작업의 작업 기록부에 기록된 BCI 작업명을 찾을 수 있습니다. 그런 다음, 그 작업명을 사용하여 BCI 작업에 대한 작업 기록부를 찾으십시오.

또한 다음 단계로 BCI 작업에 대한 작업 기록부를 찾을 수도 있습니다.

1. iSeries 명령행에 WRKSBMJOB(제출된 작업에 대한 작업) 명령을 입력하십시오.
2. 리스트의 맨 아래로 가십시오.
3. QJVACMDSRV라는 리스트에서 최종 작업을 찾으십시오.
4. 그 작업에 대해 옵션 8(스풀 파일에 대한 작업)을 입력하십시오.
5. QPJOBLOG라는 파일이 표시됩니다.
6. 스풀 파일의 보기 2를 보려면 F11 키를 누르십시오.
7. 날짜 및 시간이 실패가 발생한 날짜 및 시간과 일치하는지 확인하십시오.

   주: 날짜 및 시간이 사용자가 종료한 날짜 및 시간과 일치하지 않은 경우 제출된 작업 리스트에서 계속 찾으십시오. 사용자가 종료(Sign Off)한 시기와 일치하는 날짜 및 시간을 갖는 QJVACMDSRV 작업 기록부를 찾으십시오.

BCI 작업에 대한 작업 기록부를 찾을 수 없는 경우 생성되지 않았을 수 있습니다. 이것은 QDFTJOBD 작업 설명에 대한 ENDSEP 값을 너무 높게 설정하거나 QDFTJOBD 작업 설명에 대한 LOG 값이 *NOLIST를 지정하는 경우에 발생합니다. 이들 값을 확인하고, BCI 작업에 대해 작업 기록부가 생성되도록 값을 변경하십시오.

RUNJVA(Java 실행) 명령을 실행한 작업에 대한 작업 기록부를 생성하려면 다음의 단계를 수행 하십시오.

1. SIGNOFF *LIST를 입력하십시오.
2. 그런 다음, 다시 시작(sign on)하십시오.
3. iSeries 명령행에 WRKSPLF(스풀 파일에 대한 작업) 명령을 입력하십시오.

4. 리스트의 맨 아래로 가십시오.
5. QPJOBLOG라는 파일을 찾으십시오.
6. F11 키를 누르십시오.
7. 날짜 및 시간이 signoff 명령을 입력한 날짜 및 시간과 일치하는지 확인하십시오.

   주: 날짜 및 시간이 사용자가 종료한 날짜 및 시간과 일치하지 않은 경우 제출된 작업 리스트에서 계속 찾으십시오. 사용자가 종료(Sign Off)한 시기와 일치하는 날짜 및 시간을 갖는 QJVACMDSRV 작업 기록부를 찾으십시오.

Java 문제점 분석 자료 모음

APAR을 위한 자료를 수집하려면 다음과 같이 하십시오.

1. 문제점의 완벽한 설명을 포함시키십시오.
2. 실행하는 도중 문제점을 발생시킨 Java(TM) 클래스 파일을 저장하십시오.
3. 통합 파일 시스템에서 오브젝트를 저장하기 위해 SAV 명령을 사용할 수 있습니다. 실행을 위해 이 프로그램이 필요로 하는 다른 클래스 파일을 저장해야 할 경우도 있습니다. 또한 필요한 경우, 문제점을 재생할 때 사용하기 위해 전체 디렉토리를 저장하고 IBM으로 송신해야 할 수도 있습니다. 다음은 전체 디렉토리를 저장하는 방법의 예입니다.

   예: 디렉토리 저장

   주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

   ```
   SAV DEV('/QSYS.LIB/TAP01.DEVD') OBJ(('/mydir'))
   ```

   가능한한 문제점에 포함되는 모든 Java 클래스 소스 파일을 저장하십시오. 이것은 IBM이 문제점을 재생하고 분석하는 데 도움이 됩니다.
4. 프로그램을 실행하는 데 필요한 원시 메소드가 들어 있는 모든 서비스 프로그램을 저장하십시오.
5. Java 프로그램을 실행하는 데 필요한 모든 자료 파일을 저장하십시오.
6. 문제점 재생 방법에 대한 상세한 설명을 추가하십시오. 다음은 포함시켜야 하는 항목들입니다.
   - CLASSPATH 환경 변수 값.
   - 실행된 Java 명령의 설명.
   - 프로그램이 필요로 하는 모든 입력에 대한 응답 방법의 설명.
7. 장애 발생 시간에 인접해 발생한 VLIC(수직 사용권 내부 코드) 기록부를 포함시키십시오.
8. JVM(Java Virtual Machine)이 실행 중이던 대화식 작업 및 BCI 작업의 두 가지 작업 기록부를 추가하십시오.

IBM Developer Kit for Java 지원 확보

IBM Developer Kit for Java(TM)에 대한 지원 서비스는 iSeries 소프트웨어 제품의 거래 조건 하에서 제공됩니다. 지원 서비스는 프로그램 서비스, 음성 지원 및 자문 서비스를 포함합니다. 자세한 정보는

IBM iSeries 홈 페이지 "지원" 주제를 참조하십시오. IBM Support Services for 5722-JV1(IBM Developer Kit for Java)을 사용하십시오. 또는 IBM 영업대표에게 문의하십시오.

IBM의 지시하에 계속적인 프로그램 서비스를 받으려면 IBM Developer Kit for Java의 현재 레벨이 필요합니다. 자세한 내용은 복수 JDK(Java Development Kits)에 대한 지원을 참조하십시오.

IBM Developer Kit for Java 프로그램의 결함 해결은 프로그램 서비스 또는 음성 지원에서 지원합니다. 어플리케이션 프로그래밍 또는 디버깅 문제에 대한 해결은 자문 서비스에서 지원합니다.

다음에 해당하지 않는 경우 IBM Developer Kit for Java 어플리케이션 프로그램 인터페이스(API) 호출은 자문 서비스에서 지원합니다.

1. 상대적으로 간단한 프로그램으로 재작성해본 결과 명백하게 Java API 결함입니다.
2. 문서 교정을 요구하는 문제점입니다.
3. 샘플 또는 문서의 위치에 관한 문제점입니다.

모든 프로그래밍 지원은 자문 서비스에서 지원합니다. 여기에는 IBM Developer Kit for Java 사용권 프로그램(LP) 제품에서 제공되는 프로그램 샘플이 포함됩니다. 추가 샘플은 인터넷의

IBM iSeries 홈 페이지에서 지원되지 않는 기본을 참조하십시오.

IBM Developer Kit for Java LP는 문제점 해결에 대한 정보를 제공합니다. IBM Developer Kit for Java API에 잠재적인 결함이 있다고 생각되는 경우, 오류를 설명하는 간단한 프로그램이 필요합니다.

제 10 장 IBM Developer Kit for Java의 코드 예

다음은 IBM Developer Kit for Java(TM)의 코드 예 목록입니다.

주: 중요한 법률 정보에 대해서는 코드 예 면책사항을 참조하십시오.

CL 명령

- 

  ANZJVM

- CHGJVAPGM
- CRTJVAPGM
- DLTJVAPGM
- DMPJVM
- DSPJVAPGM
- RUNJVA

국제화

- DateFormat
- NumberFormat
- ResourceBundle

JDBC

- 

  액세스 등록 정보
- Blob

- CallableStatement 인터페이스
- 

  다른 명령문의 커서를 통해 명령문을 사용하여 값 변경
- Clob
- JNDI로 UDBDataSource 작성 및 바인드
- UDBDataSource 작성 및 사용자 ID와 암호 확보

- UDBDataSourceBind 작성 및 DataSource 등록 정보 설정

- DatabaseMetaData 인터페이스
- 

  JNDI로 UDBDataSource 작성 및 바인드
- Datalink
- 고유 유형

- SQL문 삽입
- 

  트랜잭션 종료
- 유효하지 않은 사용자 ID 및 암호
- JDBC
- 트랜잭션에서 작용하는 복수 연결
- UDBDataSource를 바인드하기 전에 초기 문맥 확보
- ParameterMetaData
- 다른 명령문의 커서를 통해 표에서 값 제거

- ResultSet 인터페이스
- 

  ResultSet 민감도
- 민감한 ResultSet와 민감하지 않은 ResultSet
- UDBDataSource 및 UDBConnectionPoolDataSource를 사용하여 연결 풀 설정
- SQLException

- 

  트랜잭션 일시중단 및 재개
- 일시중단된 ResultSet
- 연결 풀 성능 테스트
- 두 DataSource의 성능 테스트
- BLOB 갱신

- CLOB 갱신
- 복수 트랜잭션으로 연결 사용
- BLOB 사용
- CLOB 사용
- 209 페이지의『DB2CachedRowSet 등록 정보 및 DataSource 사용』
- 210 페이지의『DB2CachedRowSet 등록 정보 및 JDBC URL 사용』
- 트랜잭션을 처리하기 위한 JTA 사용
- 둘 이상의 열이 있는 메타데이터 ResultSet 사용
- 원시 JDBC 및 Toolbox JDBC 동시 사용
- ResultSet를 얻기 위해 PreparedStatement 사용
- 211 페이지의『기존 데이터베이스 연결을 사용하기 위해 execute(Connection) 메소드 사용』
- 212 페이지의『데이터베이스 요구를 그룹화하기 위해 execute(int) 메소드 사용』
- 209 페이지의『populate 메소드 사용』
- 211 페이지의『기존 데이터베이스 연결을 사용하기 위해 setConnection(Connection) 메소드 사용』
- 명령문 오브젝트의 executeUpdate 메소드 사용

Java Authentication and Authorization Service
- JAAS HelloWorld 예
- JAAS SampleThreadSubjectLogin 예

Java 일반 보안 서비스
- 샘플 비JAAS 클라이언트 프로그램
- 샘플 비JAAS 서버 프로그램
- 샘플 JAAS 작동 가능 클라이언트 프로그램
- 샘플 JAAS 작동 가능 서버 프로그램

Java 명명 및 디렉토리 인터페이스
- 디렉토리에서 항목 추가(489 페이지 참조)
- 디렉토리에서 항목 삭제(489 페이지 참조)
- 디렉토리에서 항목 추가(489 페이지 참조)
- 디렉토리 항목 이름 변경(490 페이지 참조)

- 2진 속성 이름 목록 지정(493 페이지 참조)

Java 보안 소켓 확장

- SSLContext 오브젝트를 사용한 SSL 클라이언트 및 서버

기타 프로그래밍 언어를 사용한 Java

- CL 프로그램 호출
- CL 명령 호출
- 다른 Java 프로그램 호출
- C에서 Java 호출
- RPG에서 Java 호출
- 입력 및 출력 스트림
- 호출 API
- Java용 OS/400 PASE 원시 메소드
- 소켓

선택적 패키지

- JCE

성능 분석 툴

- JPDC(Java Performance Data Converter)

GUI없이 호스트 실행

- 리모트 AWT 설정

SQLJ

- Java 어플리케이션의 내장 SQL문

보안 소켓층

- 소켓 팩토리
- 서버 소켓 팩토리
- 보안 소켓층
- 보안 소켓층 서버

제 11 장 IBM Developer Kit for Java 참조

다음은 Developer Kit for Java(TM)의 참조 사항입니다.

Javadoc

- iSeries 특정 JAAS Javadoc
- JAAS API 스펙

Java 2 플랫폼, 표준판, 버전 1.3.1

- Java 2 플랫폼, 표준판, v1.3.2 API 스펙
- AWT(Abstract Window Toolkit)
- Java IDL
- 입력 메소드 구조
- 국제화
- JDBC API
- JNI - Java 원시 인터페이스
- Java RMI(Remote Method Invocation)
- RMI - Remote Method Invocation
- 보안
- Java 2 SDK 툴

코드 면책사항 정보

이 문서에는 프로그래밍 예제가 들어 있습니다.

IBM은 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.

모든 샘플 예제는 IBM에 의해 예시 목적으로만 제공됩니다. 이러한 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.

여기에 포함된 모든 프로그램은 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 어떠한 종류의 보증 없이 "현상태대로" 제공됩니다.

Printed in U.S.A