Tibero

JDBC 개발자 안내서

Tibero 7

Copyright © 2022 TmaxTibero Co., Ltd. All Rights Reserved.

Copyright Notice

Copyright © 2022 TmaxTibero Co., Ltd. All Rights Reserved.

대한민국 경기도 성남시 분당구 황새울로258번길 29, BS 타워 9층 우)13595

Website

http://www.tmaxtibero.com

기술서비스센터

Tel : +82-1544-8629

E-Mail : info@tmax.co.kr

Restricted Rights Legend

All TmaxTibero Software (Tibero®) and documents are protected by copyright laws and international convention. TmaxTibero software and documents are made available under the terms of the TmaxTibero License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxTibero Co., Ltd. Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxTibero trademarks, logos, or any other brand features.

This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

이 소프트웨어(Tibero®) 사용설명서의 내용과 프로그램은 저작권법과 국제 조약에 의해서 보호받고 있습니다. 사용설명서의 내용과 여기에 설명된 프로그램은 TmaxTibero Co., Ltd.와의 사용권 계약 하에서만 사용이 가능하며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부분을 TmaxTibero의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단을 사용하여 전송, 복제, 배포, 2차적 저작물작성 등의 행위를 하여서는 안 됩니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 아니하며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보의 제공만을 목적으로 하고, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 아니하며, 사용설명서 상의 내용은 법적 또는 상업적인 특정한 조건을 만족시키는 것을 보장하지는 않습니다. 사용설명서의 내용은 제품의 업그레이드나 수정에 따라 그 내용이 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 아니합니다.

Trademarks

Tibero® is a registered trademark of TmaxTibero Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

Tibero®는 TmaxTibero Co., Ltd.의 등록 상표입니다. 기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상표로서 참조용으로만 사용됩니다.

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

Detailed Information related to the license can be found in the following directory : ${INSTALL_PATH}/license/oss_licenses

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

관련 상세한 정보는 제품의 다음의 디렉터리에 기재된 사항을 참고해 주십시오. : ${INSTALL_PATH}/license/oss_licenses

안내서 정보

안내서 제목: Tibero JDBC 개발자 안내서

발행일: 2024-08-22

소프트웨어 버전: Tibero 7.2.2

안내서 버전: v7.2.2

내용 목차

그림 목차

안내서에 대하여

안내서의 대상

본 안내서는 Tibero®(이하 Tibero)에서 제공하는 JDBC 기능을 이용하여 프로그램을 개발하려는 애플리케이션 개발자를 대상으로 기술한다.

안내서의 전제 조건

본 안내서를 원활히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 데이터베이스의 이해
- RDBMS의 이해
- Java 프로그래밍의 이해

안내서의 제한 조건

본 안내서는 JDBC 애플리케이션 프로그램을 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하고 있지 않다. 따라서 JDBC를 활용한 운용과 관리 또는 고급 프로그래밍에 대해서는 각 제품 안내서를 참고하기 바란다.

안내서 구성

Tibero JDBC 개발자 안내서는 총 11개의 장으로 이루어져 있다.

각 장의 주요 내용은 다음과 같다.

- 제1장: Tibero JDBC 소개

  JDBC의 기본 개념과 Tibero에서 제공하는 Tibero JDBC를 소개한다.

- 제2장: JDBC 표준 지원

  tbJDBC가 지원하는 JDBC 표준을 기능별로 기술한다.

- 제3장: tbJDBC의 사용

  tbJDBC를 사용하는 기본적인 방법을 기술한다.

- 제4장: DataSource 객체와 데이터베이스 URL

  DataSource 객체를 이용하여 데이터베이스에 연결하는 방법과 데이터베이스 URL을 기술한다.

- 제5장: 분산 트랜잭션

  분산 트랜잭션 기능을 기술한다.

- 제6장: 결과 집합 확장기능

  결과 집합의 확장기능을 기술한다.

- 제7장: Row Set

  Row Set 기능을 기술한다.

- 제8장: LOB 데이터 처리

  LOB 데이터를 처리하는 방법을 기술한다.

- 제9장: Failover와 Load balancing

  Failover와 로드 밸런싱(Load balancing) 기능을 기술한다.

- 제10장: SSL

  SSL의 기본 개념과 사용 방법을 기술한다.

- 제11장 사용자 정의 데이터 타입

  tbJDBC에서 사용자 정의 데이터 타입을 처리하는 방법을 설명한다.

안내서 규약

표기	의미
<<AaBbCc123>>	프로그램 소스 코드의 파일명
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일 계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
주의	주의할 사항
[그림 1.1]	그림 이름
[예 1.1]	예제 이름
`AaBbCc123`	Java 코드, XML 문서
[*command argument*]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택 사항. 예) A|B: A나 B 중 하나
…	파라미터 등이 반복되어서 나옴
${ }	환경변수

시스템 사용 환경

	요구 사항
Platform	HP-UX 11i v3(11.31)
	Solaris (Solaris 11)
	AIX (AIX 7.1/AIX 7.2/AIX 7.3)
	GNU (X86, 64, IA64)
	Red Hat Enterprise Linux 7 kernel 3.10.0 이상
	Windows(x86) 64bit
Hardware	최소 2.5GB 하드디스크 공간
	1GB 이상 메모리 공간
Compiler	PSM (C99 지원 필요)
	tbESQL/C (C99 지원 필요)

관련 안내서

안내서	설명
Tibero 설치 안내서	설치 시 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero 애플리케이션 개발자 안내서	각종 애플리케이션 라이브러리를 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero External Procedure 안내서	External Procedure를 소개하고 이를 생성하고 사용하는 방법을 기술한 안내서이다.
Tibero tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbPSM 안내서	저장 프러시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브 프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero tbPSM 참조 안내서	저장 프러시저 모듈인 tbPSM의 패키지를 소개하고, 이러한 패키지에 포함된 각 프러시저와 함수의 프로토타입, 파라미터, 예제 등을 기술한 참조 안내서이다.
Tibero 관리자 안내서	Tibero의 동작과 주요 기능의 원활한 수행을 보장하기 위해 DBA가 알아야 할 관리 방법을 논리적 또는 물리적 측면에서 설명하고, 관리를 지원하는 각종 도구를 기술한 안내서이다.
Tibero 유틸리티 안내서	데이터베이스와 관련된 작업을 수행하기 위해 필요한 유틸리티의 설치 및 환경설정, 사용 방법을 기술한 안내서이다.
Tibero TAS 안내서	Tibero Active Cluster (TAS)를 사용해서 Tibero의 파일을 관리하고자 하는 관리자를 대상으로 기술한 안내서이다.
Tibero	Tibero를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.

안내서	설명
에러 참조 안내서	
Tibero 참조 안내서	Tibero의 동작과 사용에 필요한 초기화 파라미터와 데이터 사전, 정적 뷰, 동적 뷰를 기술한 참조 안내서이다.
Tibero SQL 참조 안내서	데이터베이스 작업을 수행하거나 애플리케이션 프로그램을 작성할 때 필요한 SQL 문장을 기술한 참조 안내서이다.
Tibero Spatial 참조 안내서	Tibero에서 Geometry 타입에 대한 설명과 Spatial 기능 관련 프러시저 함수 목록 및 사용 방법 등을 기술한 안내서이다.
Tibero TEXT 참조 안내서	Tibero의 제공하는 Text Index를 소개하고, Text Index를 생성 하고 사용하는 방법을 기술하는 안내서이다.
Tibero TDP.NET 안내서	Tibero Data Provider for .NET 기능을 기술하는 안내서이다.
Tibero IMCS 안내서	Tibero에서 제공하는 In-Memory Column Store(이하 IMCS) 기능을 기술하는 안내서이다.

제1장 Tibero JDBC 소개

본 장에서는 JDBC의 기본 개념과 Tibero에서 제공하는 Tibero JDBC(이하 tbJDBC)를 소개한다.

1.1. JDBC

JDBC(Java Database Connectivity)는 Java로 개발된 프로그램 안에서 SQL 문장을 실행하기 위해 데이터베이스를 연결해 주는 API(Application Program Interface)이다.

JDBC를 사용하면, 다음과 같은 이점이 있다.

- 어떠한 관계형 데이터베이스에서도 SQL 문장을 사용할 수 있다.

  DBMS 벤더별로 데이터베이스에 접근하는 프로그램을 따로 만들 필요가 없다.

- 업무 프로그램을 Java로 작성하면, 플랫폼별로 다르게 작성하지 않아도 된다.

  Java로 작성된 애플리케이션은 어디에서나 동작할 수 있다.

- Java의 기능을 확장할 수 있다.

  예를 들어 원격 데이터베이스에서 얻은 정보를 애플릿으로 구성하거나 하나 이상의 내부 데이터베이스를 연결하는 데에도 사용할 수 있다. 또한 다른 곳에 저장된 정보도 JDBC를 이용하여 쉽게 접근할 수 있으며, 새로운 애플리케이션을 개발하는 데 걸리는 시간도 단축할 수 있다.

1.2. tbJDBC

Tibero에서는 다음 장에서 설명할 JDBC 표준을 준수함은 물론 별도의 API를 추가로 제공하고 있다. 이렇게 구성된 API를 **tbJDBC**(Tibero의 Java Database Connectivity)라 한다.

tbJDBC는 다음과 같은 특징이 있다.

- 클래스와 인터페이스 메소드로 이루어져 있다.
- SQL 표준인 SQL-99를 지원한다.
- DBMS 종류와 관계 없이 독립적으로 프로그램을 개발할 수 있다.
- tbJDBC는 Java 패키지(java.sql.*, javax.sql.*)를 상속한다.

  JDBC의 버전은 Java 표준 스펙에 따라 달라진다.

  - JDBC 3.0(J2SE 1.4)
  - JDBC 4.0(Java SE 6)

1.2.1. 동작 구조

JDBC 표준을 이용하여 애플리케이션 프로그램 개발자는 해당 JDBC 표준에 맞는 Driver를 만들어 배포할 수 있다.

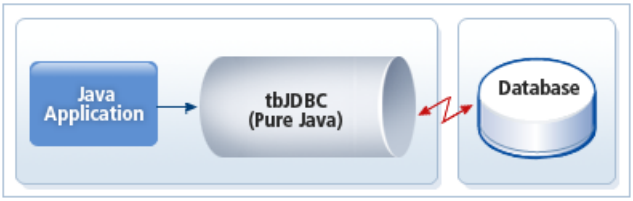
JDBC는 다음과 같이 네 가지 타입의 Driver를 제공한다.

타입	설명
JDBC-ODBC Bridge Driver	JDBC로 데이터베이스에 직접 연결하지 않고 ODBC를 사용한다.
Native-API Driver	JDBC 명령을 DBMS 고유의 클라이언트 프로그램에 맞게 변환한다.
Net-Protocol Driver	데이터베이스에 종속되지 않은 프로토콜(WAS용)로 JDBC를 변환한 후 WAS에서 연결하여 처리한다.
Native-Protocol Driver	DBMS 벤더에서 지원하는 JDBC Driver로 JDBC 문장을 DBMS 고유의 프로토콜로 변환한다.

tbJDBC는 데이터베이스 서버를 설치하지 않아도 100% Java로 작성된 애플리케이션 프로그램을 개발할 수 있는 JDBC Driver를 제공한다. 위 표에서 설명한 Native-Protocol Driver(또는 Thin Driver) 타입의 Driver이다.

tbJDBC는 다음과 같이 동작한다.

[그림 1.1] tbJDBC의 동작 구조



1.2.2. 기본 경로

Tibero를 서버에 설치한 후 tbJDBC는 $TB_HOME/client/lib/jar 디렉터리에 생성된다. 단, JDK 버전에 따라 생성되는 파일명이 다르다.

JDK 버전	설명
1.4 이상	– tibero7-jdbc-14.jar : tbJDBC 파일 – tibero7-jdbc-14-dbg.jar : 디버깅용 tbJDBC 파일
1.6 이상	– tibero7-jdbc.jar : tbJDBC 파일 – tibero7-jdbc-dbg.jar : 디버깅용 tbJDBC 파일
1.8 이상	– tibero7-jdbc-18.jar : tbJDBC 파일

JDK 버전	설명
	– tibero7-jdbc-18-dbg.jar : 디버깅용 tbJDBC 파일

1.2.3. 제약 사항

tbJDBC를 사용할 때 다음과 같은 제약 사항이 있다.

- 데이터베이스 서버를 설치할 때 반드시 자동으로 생성되는 tbJDBC를 사용한다.
- 이전 버전에 대한 호환성(backward compatibility)을 제공하지 않는다.
- JDK 1.4 이상이 반드시 설치되어 있어야 한다.

  다음의 위치에서 JDK를 다운로드할 수 있다.

  ```
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
  ```

  만약 시스템이 Oracle사의 JDK를 사용하지 않는다면 각각의 시스템에 적합한 JDK를 찾아 설치한다. 예를 들어 HP-UX는 HP, AIX는 IBM에서 JDK를 다운로드 받아 설치한다.

  시스템별로 JDK를 설치하는 방법은 다음 위치에서 확인할 수 있다.

  ```
  http://www.oracle.com/technetwork/java/index.html
  ```

제2장 JDBC 표준 지원

tbJDBC는 JDBC 4.0을 준수하고 있다. 다만, 표준에 따르면 필수적으로 구현해야 하는 부분과 벤더별로 구현여부를 선택 가능한 부분을 구분하여 명시하고 있으며, 이러한 추가 기능에 대해서는 지원하지 않을 수도 있다. 본 장에서는 tbJDBC가 지원하는 JDBC 표준을 기능별로 설명한다.

2.1. JDBC 2.0

2.1.1. 인터페이스 메소드

tbJDBC에서 지원하는 주요한 인터페이스 메소드는 다음과 같다.

- java.sql.Array
- java.sql.Blob
- java.sql.CallableStatement
- java.sql.Clob
- java.sql.Connection
- java.sql.DatabaseMetaData
- java.sql.Driver
- java.sql.PreparedStatement
- java.sql.ResultSet
- java.sql.ResultSetMetaData
- java.sql.Statement
- java.sql.Struct

tbJDBC에서 지원하지 않는 인터페이스 메소드는 다음과 같다.

- java.sql.Ref
- java.sql.SQLData
- java.sql.SQLInput
- java.sql.SQLOutput

2.1.2. 주요 기능

다음은 JDBC 2.0 표준 지원하는 주요 기능에 대한 설명이다.

- 결과 집합

  tbJDBC에서 지원하는 결과 집합 기능은 다음과 같다.

  – Scrolling 기능

  Scrolling 기능을 지원하는 스크롤 가능한 결과 집합(Scrollable ResultSet)을 생성할 수 있다. 이 기능은 결과 집합을 전 방향 또는 후 방향으로 탐색할 수 있다. 또한 결과 집합 내에서 상대적이거나 절대적인 위치 이동도 할 수 있다.

  – 결과 집합 타입

  결과 집합은 다음과 같이 3가지 타입으로 지정할 수 있다.

타입	**설명**
Forward-only	최초 생성될 때의 데이터 값으로 고정되며, 오직 정방향으로만 탐색할 수 있다.
Scroll-insensitive	Forward-only 결과 집합과 마찬가지로 최초 생성될 때의 데이터 값으로 고정되지만, 스크롤 가능한 결과 집합의 특성을 가지므로 정방향, 역방향, 상대위치, 절대위치 등으로 탐색할 수 있다.
Scroll-sensitive	데이터베이스 변화에 민감하며, 데이터가 변경되는 경우 즉시 결과 집합에 반영된다. 따라서 새로운 데이터를 볼 수 있다.

  – Concurrency 타입

  결과 집합에 대해 다음과 같이 2가지 Concurrency 타입을 지정할 수 있다.

타입	**설명**
Read-only	데이터를 수정할 수 없다. 따라서 여러 트랜잭션 때문에 발생할 수 있는 잠금(Lock) 현상은 없다.
Updatable	데이터를 수정할 수 있다. 단, 서로 다른 사용자가 같은 데이터에 접근할 경우 잠금 현상때문에 동시 수행을 할 수 없다.

- Batch update

  Batch update 기능은 여러 개의 DML 문장을 한꺼번에 처리할 수 있는 기능이다. 또한 개별로 INSERT 문을 수행하거나 준비된 문장(Prepared Statement)을 사용하여 파라미터만 바꿔가며 수행할 수도 있다. 이 때문에 시스템 성능 향상에 많은 도움을 준다.

- 데이터 타입

  tbJDBC는 BLOB(Binary Large OBject)과 CLOB(Character Large OBject) 데이터 타입을 지원한다. 사용자는 BLOB과 CLOB 데이터 타입의 데이터를 받아오기 위해 getClob(), getBlob() 메소드를 사용할 수 있으며, 데이터를 저장하기 위해 setClob(), setBlob() 메소드를 추가로 사용할 수 있다.

2.2. JDBC 3.0

2.2.1. 인터페이스 메소드

tbJDBC에서 지원하는 주요한 인터페이스 메소드는 다음과 같다.

- java.sql.ParameterMetaData
- java.sql.Savepoints

tbJDBC에서 지원하는 javax.sql 패키지의 인터페이스 메소드는 다음과 같다.

- javax.sql.ConnectionEventListener
- javax.sql.ConnectionPoolDataSource
- javax.sql.DataSource
- javax.sql.PooledConnection
- javax.sql.RowSet
- javax.sql.RowSetInternal
- javax.sql.RowSetListener
- javax.sql.RowSetMetaData
- javax.sql.RowSetReader
- javax.sql.RowSetWriter
- javax.sql.XAConnection
- javax.sql.XADataSource

2.2.2. 주요 기능

tbJDBC는 JDBC 3.0 표준의 일부를 제외한 모든 기능을 지원한다.

지원 기능

다음은 JDBC 3.0 표준 지원하는 주요 기능에 대한 설명이다.

- 저장점

Savepoint 인터페이스를 구현하여 임의의 트랜잭션에 대한 저장점 설정과 커밋 및 롤백 기능을 제공한다. 단, 특정한 저장점을 해제하는 Connection.releaseSavepoint java.sql 메소드는 제공하지 않는다.

- 문장 풀링(Pooling)

  접속 풀링과 관련된 문장의 풀링을 제공한다.

- 접속 풀링(Pooling)

  ConnectionPoolDataSource 인터페이스에서 설정할 수 있는 내용을 지원한다. 사용자는 DataSource 객체에 의해 PooledConnection이 어떠한 특성을 갖고 생성될지를 결정할 수 있다.

- 파라미터 메타데이터

  ParameterMetaData 인터페이스를 구현한 JDBC 클래스는 준비된 문장(Prepared Statement)에서 사용한 파라미터 개수의 정보를 제공한다. 단, 데이터 타입과 속성(Property)에 대한 메타데이터(metadata)는 제공하지 않는다.

- 자동 생성 키

  SQL 문장을 실행한 후 결과 집합으로부터 `getGeneratedKeys` 함수를 사용하여 결과 로우에 대한 키 또는 자동으로 생성된 컬럼의 값을 얻는다.

- 이름 있는 파라미터

  CallableStatement 객체의 파라미터를 저장할 때 파라미터 이름으로 구별할 수 있는 이름 있는 파라미터(named parameter) 기능을 지원한다.

- 결과 집합의 유지성

  결과 집합이 열려있는 동안에 커밋이 발생했을 때 결과 집합을 그대로 유지할지 아니면 닫을지를 설정한다. tbJDBC에서는 결과집합을 유지하는 방법만을 지원한다.

- 여러 개의 결과 집합 반환

  한 SQL 문장이 여러 개의 결과 집합을 열 수 있도록 지원한다.

- BLOB와 CLOB 객체에 존재하는 데이터의 수정

  updateXXX API를 통해 BLOB와 CLOB 객체에 포함된 데이터를 수정하는 기능을 제공한다.

미지원 기능

tbJDBC에서 지원하지 않는 기능은 다음과 같다.

- BOOLEAN, DATALINK, URL 데이터 타입
- REF 객체의 수정
- 그룹 변환 및 데이터 타입의 매핑

2.3. JDBC 4.0

2.3.1. 인터페이스 메소드

tbJDBC에서 지원하는 주요한 인터페이스 메소드는 다음과 같다.

- java.sql.NClob
- java.sql.RowId
- java.sql.SQLXML
- java.sql.Wrapper

tbJDBC에서 지원하는 javax.sql 패키지의 인터페이스 메소드는 다음과 같다.

- javax.sql.StatementEventListener

2.3.2. 주요 기능

지원 기능

다음은 JDBC 4.0 표준 지원하는 주요 기능에 대한 설명이다.

- XML 데이터 타입

  SQLXML 인터페이스를 통해 SQL:2003에 추가된 XML 데이터 타입을 사용할 수 있다.

- 자동 드라이버 검출

  자동으로 드라이버를 로딩할 수 있게 되어 Class.forName를 사용한 java.sql.Driver 클래스의 로드 없이도 드라이버 객체를 사용할 수 있다.

- 국가별 캐릭터 셋 지원

  데이터베이스에서 별도로 지정해 사용하는 국가별 캐릭터 셋을 지원하기 위한 API가 추가되었다.

- 향상된 SQLException

  연쇄적으로 연결된 예외를 생성할 수 있게 되어 보다 자세한 원인을 전달해 줄 수 있으며, 새로운 종류의 예외 타입이 추가되었다.

- 향상된 Blob/Clob 기능

  Blob/Clob 객체를 생성/해제할 수 있는 API를 지원한다.

- ClientInfo 설정 기능

setClientInfo/getClientInfo 함수를 사용한 ClientInfo 설정/조회를 지원한다.

- SQL ROWID 데이터 타입의 지원

  RowId 인터페이스를 이용하여 SQL ROWID 타입을 사용할 수 있다.

- 실제 JDBC 객체에 대한 접근 허용

  Wrapper 인터페이스를 이용하여 애플리케이션 서버나 접속 풀링 환경에서도 실제 JDBC 객체에 접근해 사용할 수 있다.

- 접속 풀링 환경에서 실제 접속 상태에 대한 통지

  접속 풀링 환경에서 실제 접속이 닫히거나 유효하지 않게 되었을 때 그 상태를 풀링된 문장에 통지해 준다.

- 사용자 정의 타입 조회, 사용자 정의 타입을 인자로 하는 PSM 호출

  Struct, Array 타입을 생성해서 inbind하거나 서버로부터 select한 후 outbind해서 구조적으로 탐색할 수 있다.

2.4. JDBC 4.1

2.4.1. 인터페이스 메소드

tbJDBC에서 지원하는 주요한 인터페이스 메소드는 다음과 같다.

- java.sql.Connection
  - void abort(Executor executor)
  - void setSchema(String schema)
  - String getSchema()
- java.sql.Statement
  - void closeOnCompletion()
  - boolean isCloseOnCompletion()
- java.sql.ResultSet
  - <T> T getObject(int parameterIndex, Class<T> type)

2.4.2. 주요 기능

지원 기능

다음은 JDBC 4.1 표준 지원하는 주요 기능에 대한 설명이다.

- BIGINT

  java.math.BigInteger와 JDBC Type BIGINT를 지원한다.

- Connection

  `abort`함수를 사용한 연결 종료를 지원한다.

  `setSchema`함수를 사용한 schema 지정을 지원한다.

  `setNetworkTimeout`함수를 사용한 timeout 지정을 지원한다.

- Statement

  `closeOnCompletion`함수를 사용해 결과 집합이 닫힐 때 닫히도록 설정할 수 있다.

- ResultSet

  `getObject`함수의 인자로 Class를 사용하여 반환 타입 지정을 할 수 있도록 지원한다.

참고

JDBC 4.1은 Tibero 7 FS02 릴리즈 부터 지원한다.

2.5. JDBC 4.2

2.5.1. 인터페이스 메소드

tbJDBC에서 지원하는 주요한 인터페이스 메소드는 다음과 같다.

- java.sql.CallableStatement
  - void registerOutParameter(int parameterIndex, SQLType sqlType)
- java.sql.PreparedStatement
  - void setObject(int parameterIndex, Object x, SQLType targetSqlType)
- java.sql.Statement

  - long executeLargeUpdate()
- java.sql.SQLInput
  - <T> T readObject(Class<T> type)
- java.sql.SQLOutput
  - void writeObject(Object x, SQLType sqlType)

2.5.2. 주요 기능

지원 기능

다음은 JDBC 4.2 표준 지원하는 주요 기능에 대한 설명이다.

- java.time

  LocalDate, LocalTime, LocalDateTime, OffsetTime, OffsetDateTime 타입을 지원한다.

- SQLType

  `setObject, registerOutParameter`함수의 인자로 SQLType을 사용한 타입 지정을 지원한다.

- Statement

  `executeLargeUpdate`함수를 사용해 업데이트 횟수를 long으로 받을 수 있다.

참고

JDBC 4.2는 Tibero 7 FS02 릴리즈 부터 지원한다.

제3장 tbJDBC의 사용

본 장에서는 tbJDBC를 사용하여 애플리케이션 프로그램을 개발하는 과정을 순서대로 설명하고, 추가로 데이터 타입, Stream 등을 설명한다.

3.1. 개발 과정

다음은 tbJDBC를 사용하여 애플리케이션 프로그램을 개발하는 과정이다.

1. **패키지 import**

   기본이 되는 Java 패키지와 tbJDBC 패키지를 import한다.

```
/* 기본 Java 패키지 */
import java.sql.*;

/* tbJDBC 패키지 */
import com.tmax.tibero.jdbc.*;
import com.tmax.tibero.jdbc.ext.*;
```

2. **데이터베이스 연결**

   데이터베이스에 연결하기 위해서 Connection 객체를 생성한다. Connection 객체를 생성하기 위해서 tbJDBC를 로딩한 후 DriverManager를 이용한다.

```
Class.forName("com.tmax.tibero.jdbc.TbDriver");
Connection conn = DriverManager.getConnection
                 ("jdbc:tibero:thin:@localhost:8629:dbsvr",
                  "tibero", "tmax");
```

   또는 다음과 같이 java.util.Properties를 이용하여 Conneciton할 때 설정 정보를 설정할 수 있다.

```
Class.forName("com.tmax.tibero.jdbc.TbDriver");
Properties prop = new Properties();
prop.setProperty("user", "tibero");
prop.setProperty("password", "tmax");
Connection conn =
    DriverManager.getConnection("jdbc:tibero:thin:@localhost:8629:dbsvr", prop);
```

   또는 다음과 같이 DataSource 객체를 이용하여 Connection 객체를 생성한다.

```
TbDataSource ds = new TbDataSource();
ds.setURL("jdbc:tibero:thin:@localhost:8629:dbsvr");
```

```
ds.setUser("tibero");
ds.setPassword("tmax");
Connection conn = ds.getConnection();
```

3. **Statement 객체 생성**

   데이터베이스에 연결되면, Connection 객체를 이용하여 Statement 객체를 생성한다.

```
Statement         stmt  = conn.createStatement();
PreparedStatement pstmt = conn.prepareStatement("SELECT empno, name FROM emp");
CallableStatement cstmt = conn.prepareCall("BEGIN ... END;");
```

4. **질의문 수행과 ResultSet 객체 받기**

   질의문을 수행하고 ResultSet 객체를 받는다.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM ALL_OBJECTS;");
```

5. **ResultSet 객체 처리**

   ResultSet 객체는 next() 메소드를 이용하여 모든 로우에 접근할 수 있다. 이때 getXXX() 메소드를 호출하면 원하는 타입별로 로우 데이터를 가져올 수 있다. 만약 next() 메소드가 false라면, 더는 결과 집합이 존재하지 않음을 의미한다.

```
while (rs.next())
{
    System.out.println(rs.getString(1));
    System.out.println(rs.getInt(2));
}
```

6. **커밋 또는 롤백 수행**

   기본적으로 DML 문(INSERT, UPDATE, DELETE)을 수행한 다음에는 자동으로 커밋된다. 이러한 기능을 auto-commit이라고 한다. 사용자는 Connection 클래스에서 제공하는 다음의 메소드를 이용하여 이 기능을 임의로 해제할 수 있다.

```
void setAutoCommit(boolean flag)
```

   이 메소드를 이용하여 auto-commit 기능을 비활성화하면, 사용자는 반드시 DML 문을 사용한 다음 커밋이나 롤백을 수행해야 데이터베이스에 적용된다.

```
conn.commit();
conn.rollback();
```

7. **ResultSet 객체와 Statement 객체 소멸**

ResultSet 객체와 Statement 객체는 반드시 소멸시켜야 한다. 왜냐하면, tbJDBC 내부에는 별도의 finalizer 메소드가 없기 때문이다. 이 때문에 예상치 못한 시점에 심각한 메모리 누수 현상이 발생할 수 있고, 데이터베이스 내부 커서의 최대 허용 범위를 초과하여 에러가 발생할 수도 있다.

```
rs.close();
stmt.close();
```

8. **데이터베이스 연결 해제**

마지막으로 모든 작업이 끝나면 반드시 데이터베이스 연결을 해제해야 한다. 그렇지 않으면 현재 세션이 연결된 상태로 남아 있기 때문에 다음 연결할 때에는 최대 허용 세션을 초과할 수 있다. 따라서 데이터베이스 연결에 실패할 수도 있다.

```
conn.close();
```

3.2. Connection Properties

다음은 DB를 연결할 때에 Properties로 설정할 수 있는 속성들이다.

속성명	타입	설명
databaseName	String	서버에 존재하는 특정 데이터베이스의 이름이다.
dataSourceName	String	데이터소스의 이름이다.
description	String	데이터소스에 대한 설명이다.
networkProtocol	String	서버와 통신하는 네트워크 프로토콜의 이름이다. (기본값: TCP)
password	String	서버 접속을 위한 패스워드이다.
user	String	서버 접속을 위한 사용자 이름이다.
portNumber	int	서버 리스너의 포트 번호이다.
serverName	String	데이터베이스의 이름이다.
login_timeout	int	소켓의 read timeout을 지정한다. 최소 소켓을 생성할 때부터 DB 연결 생성이 완료될 때까지 적용된다. DB 연결 생성이 완료된 뒤에는 read_timeout 속성이 적용된다. 만약 응답이 없이 지정된 시간이 지나면 timeout을 발생시키지만 설정값이 0일 경우에는 timeout을 발생시키지 않는다. (단위: millisecond, 기본값: 0)
read_timeout	int	DB 연결 생성이 완료된 이후의 소켓의 read timeout을 지정한다. 만약 응답이 없이 지정된 시간이 지나면 timeout을 발생시키지만 설정값이 0일 경우에는 timeout을 발생시키지 않는다. (단위: millisecond, 기본값: 0)

속성명	타입	설명
characterset	String	JDBC에서 인코딩, 디코딩 Character Set을 지정한다. (기본값: 서버의 문자 셋을 사용)
program_name	String	프로그램 이름이다. (기본값: JDBC Thin Client)
includeSynonyms	String	DatabaseMetaData.getColumn()에서 synonyms 객체를 포함할지 여부를 지정한다. (기본값: false)
mapDateToTimestamp	String	DATE 컬럼에 대해서 결과 타입으로 Timestamp(true)로 돌려줄지 Date(false)로 여부를 지정한다. (기본값: true)
defaultNChar	String	PreparedStatement.setString() API로 설정한 문자열을 national charset 설정을 이용하여 서버로 전송하도록 강제한다. (기본값: false)
self_keepalive	String	SELF KEEP ALIVE 기능을 활성화할지 여부를 지정한다. 활성화한 경우에는 self_keepidle, self_keepintvl, self_keepcnt 속성 설정에 따라 연결 대상에 대한 네트워크 접근에 문제가 없는지 확인한다. 전체 확인 과정에 실패하면, 해당 네트워크 연결을 강제로 종료시킨다. (기본값: false)
self_keepidle	int	접속 완료, DB 요청/응답 처리 등의 정상적인 네트워크 사용 이후, 다음 번의 정상 처리된 시간이 갱신되지 않을 때, 정상 상황으로 간주할 최대 시간을 지정한다. 지정된 시간이 지나면 네트워크 접근에 대한 확인 절차가 시작된다. **self_keepalive 설정이 true인 경우에만 유효하다.** (단위: 초, 기본값: 60)
self_keepintvl	int	매 번 확인할 때의 간격 및 한 번 확인할 때 최대 대기 시간을 지정한다. **self_keepalive 설정이 true인 경우에만 유효하다.** (단위: 초, 기본값: 10)
self_keepcnt	int	확인 절차를 총 몇 회까지 수행할 것인지를 지정한다. 지정된 횟수만큼 연속으로 실패하여야만 전체 확인과정이 실패한 것으로 처리된다. **self_keepalive 설정이 true인 경우에만 유효하다.** (기본값: 3)
failover_retry_count	int	Failover 기능이 활성화되었을 때 연결 복원을 시도하는 최대 횟수를 지정한다. (기본값: 3) Failover 기능에 대한 자세한 설명은 “제9장 Failover와 Load balancing”의 내용을 참고한다.

3.3. 데이터 타입

tbJDBC는 JDBC 타입을 지원할 뿐만 아니라 Interval 타입을 추가로 제공한다.

다음은 JDBC 타입과 tbJDBC 타입과의 대응 관계를 나타내는 표이다.

JDBC 타입(표준)	Java 타입(표준)	tbJDBC 타입
java.sql.Types.CHAR	java.lang.String	java.lang.String
java.sql.Types.VARCHAR	java.lang.String	java.lang.String
java.sql.Types.LONGVARCHAR	java.lang.String	java.lang.String
java.sql.Types.NUMERIC	java.math.BigDecimal	java.math.BigDecimal
java.sql.Types.DECIMAL	java.math.BigDecimal	java.math.BigDecimal
java.sql.Types.BIT	boolean	boolean
java.sql.Types.TINYINT	byte	byte
java.sql.Types.SMALLINT	short	short
java.sql.Types.INTEGER	int	int
java.sql.Types.BIGINT	long	long
java.sql.Types.REAL	float	float
java.sql.Types.FLOAT	double	double
java.sql.Types.DOUBLE	double	double
java.sql.Types.BINARY	byte[]	byte[]
java.sql.Types.VARBINARY	byte[]	byte[]
java.sql.Types.LONGVARBINARY	byte[]	byte[]
java.sql.Types.DATE	java.sql.Date	java.sql.Date
java.sql.Types.TIME	java.sql.Time	java.sql.Time
java.sql.Types.TIMESTAMP	java.sql.Timestamp	java.sql.Timestamp
java.sql.Types.BLOB	java.sql.Blob	com.tmax.tibero.jdbc.TbBlob
java.sql.Types.CLOB	java.sql.Clob	com.tmax.tibero.jdbc.TbClob
java.sql.Types.NCLOB	java.sql.NClob	com.tmax.tibero.jdbc.TbNClob
java.sql.Types.NCHAR	java.sql.NCHAR	java.lang.String
java.sql.Types.NVARCHAR	java.sql.NVARCHAR	java.lang.String
java.sql.Types.SQLXML	java.sql.SQLXML	com.tmax.tibero.jdbc.TbSQLXML
com.tmax.tibero.jdbc.data.DataType.VARRAY	java.sql.Array	com.tmax.tibero.jdbc.TbArray
com.tmax.tibero.jdbc.data.DataType.Struct	java.sql.Struct	com.tmax.tibero.jdbc.TbUpStruct

JDBC 타입(표준)	Java 타입(표준)	tbJDBC 타입
com.tmax.tibero.jdbc.data.DataType.CURSOR	-	com.tmax.tibero.jdbc.TbResultSet
com.tmax.tibero.jdbc.data.DataType.ITV_DTS	-	com.tmax.tibero.jdbc.TbIntervalDts
com.tmax.tibero.jdbc.data.DataType.ITV_YTM	-	com.tmax.tibero.jdbc.TbIntervalYtm
com.tmax.tibero.jdbc.data.DataType.RowId	-	com.tmax.tibero.jdbc.TbRowId
com.tmax.tibero.jdbc.data.DataType.TIMESTAMP_TZ	java.sql.Timestamp	com.tmax.tibero.jdbc.TbTimestampTZ
com.tmax.tibero.jdbc.data.DataType.TIMESTAMP_LTZ	java.sql.Timestamp	java.sql.Timestamp

참고

Tibero에서 TIME 데이터 타입을 지원하지만 DATE 데이터 타입을 사용하기를 권장한다.

TIME 컬럼에 대해서 PreparedStatement.setTime() 메소드로 지정할 수 없다. PreapredStatement.setTime()은 DATE 데이터 타입의 컬럼에 대해서 지원하고 있다. TIME 컬럼에 대한 값을 지정하기 위해서는 문자열을 TIME 값으로 변환하는 TO_TIME 함수를 통해서 입력할 수 있다.

3.4. tbJDBC Stream

tbJDBC에서는 특정 데이터 타입에 대해 다음과 같은 Stream 기능을 제공한다.

- 컬럼에 대한 Stream 생성
  - CHAR, VARCHAR, RAW, NCHAR, NVARCHAR
  - LONG, LONG RAW
  - LOB
- Stream 소멸

tbJDBC에서는 다음과 같이 4종류의 Stream 메소드를 제공한다.

메소드	반환 값	설명
getAsciiStream	Java.io.InputStream	ASCII 형태로 변환된 데이터 Stream이다.
getBinaryStream	Java.io.InputStream	byte 형태로 변환된 데이터 Stream이다.

메소드	반환 값	설명
getUnicodeStream	Java.io.InputStream	Unicode 형태로 변환된 데이터 Stream이다. 이 메소드는 JDBC 4.0에서 폐기(deprecated)되었으므로 호환성을 고려하여 가급적 사용하지 않는 것이 좋다.
getCharacterStream	Java.io.Reader	문자열 형태로 변환된 데이터 Reader이다.

각각의 메소드를 이용하면 해당 데이터 타입에서 허용하는 최대 크기까지 점진적으로 데이터를 읽을 수 있다. 메소드에서 반환되는 객체는 InputStream과 Reader이고, read() 메소드를 사용하면 데이터를 읽을 수 있다.

3.4.1. 컬럼에 대한 Stream 생성

CHAR, VARCHAR, RAW 컬럼에 Stream 메소드를 사용하는 경우 전체 데이터를 한꺼번에 받아와서 Stream 객체로 생성한다. 반면, LONG이나 LONG RAW, LOB 컬럼에 대해서는 최초 일부의 데이터만 읽어온 후 Stream 객체를 생성한다. 단, 사용자가 추가로 요청하면 차례대로 데이터를 더 읽어온 후 Stream 객체를 생성하게 된다.

다음은 getBinaryStream()을 이용하여 데이터를 읽어오는 예이다.

```
ResultSet rs = stmt.executeQuery("SELECT name, image FROM backgrounds");

while (rs.next()) {
    InputStream imageIS = rs.getBinaryStream(2);
    try {
        FileOutputStream fos = new FileOutputStream(rs.getString(1));
        int chunk = 0;
        while ((chunk = imageIS.read()) != -1)
            fos.write(chunk);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    finally {
        if (fos != null)
            fos.close();
    }
}
```

다음은 getBinaryStram() 대신 getBytes()를 이용하여 데이터를 읽어오는 예이다. **getBytes()**는 해당 컬럼에 저장된 모든 데이터를 한꺼번에 받아오기 때문에 그 크기만큼 메모리를 차지한다는 점을 주의해야 한다.

```
ResultSet rs = stmt.executeQuery("SELECT name, image FROM backgrounds");
```

```
while (rs.next()) {
    bytes[] images = rs.getBytes(2);
    try {
        FileOutputStream fos = new FileOutputStream(rs.getString(1));
        fos.write(images);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    finally {
        if (fos != null)
            fos.close();
    }
}
```

3.5. 내장 함수 호출

3.5.1. PSM의 사용

tbJDBC에서는 PSM을 사용할 수 있다.

다음과 같이 SQL-92의 ESCAPE 문법과 PSM 문법을 제공한다.

```
// SQL92 ESCAPE 문법
CallableStatement cstmt = conn.prepareCall("{call proc(?)}");
CallableStatement cstmt = conn.prepareCall("{? = call func(?)}");

// PSM 문법
CallableStatement cstmt = conn.prepareCall("begin proc(?); end;");
CallableStatement cstmt = conn.prepareCall("begin ? := func(?); end;");
```

다음은 PSM 문법을 사용하여 내장 함수를 호출하는 예이다.

```
create or replace function concat(x varchar2, y varchar2)
return varchar2 is
begin
    return x || y;
end;
```

```
CallableStatement cstmt = conn.prepareCall("begin ? := concat(?, ?); end;");

cstmt.regiterOutParameter(1, Types.VARCHAR);
cstmt.setString(2, "Tmax");
cstmt.setString(3, "Tibero");
```

```
cstmt.executeUpdate();

String result = cstmt.getString(1);
```

3.5.2. Java Stored Procedure의 사용

이전 절에서 설명한 방법과 같이 Java Stored Procedure를 사용할 수 있다.

참고

Java Stored Procedure에 대한 자세한 내용은 "Tibero External Procedure 안내서"를 참고한다.

3.6. 예외 처리

tbJDBC에서는 예외 상황을 처리하기 위해 java.sql.SQLException 클래스 객체를 정의할 수 있다. 예외 상황은 tbJDBC 내부에서 발생할 수도 있고, 데이터베이스 내부에서 발생할 수도 있다. 예외 상황의 내용은 에러 코드와 에러가 발생한 위치 등의 정보가 포함되어 있다.

예외 상황에 대한 정보를 얻기 위해 다음과 같은 메소드가 제공된다.

메소드	설명
getMessage()	에러가 발생한 원인이다.
getSQLState()	SQL 상태 정보이다.
getErrorCode()	에러 코드이다.
printStackTrace()	에러가 발생한 스택의 트레이스 정보(Stack trace)이다.

다음은 위의 메소드를 사용하여 예외 상황을 처리하는 예이다.

```
try {
    stmt.execute("drop table not_exist_table");
}
catch (SQLException e) {
    System.out.println("ERROR[" + e.getErrorCode() + "]" + e.getMessage());
    e.printStackTrace();
}
```

실행하면 다음과 같은 정보가 화면에 출력된다.

```
ERROR[-7071] Schema object'NOT_EXIST_TABLE' was not found or is invalid.
java.sql.SQLException: Schema object 'NOT_EXIST_TABLE' was not found or is invalid.
   at com.tmax.tibero.jdbc.msg.common.TbMsgError.readErrorStackInfo(TbMsgError.java
:108)
   at com.tmax.tibero.jdbc.msg.TbMsgEreply.deserialize(TbMsgEreply.java:61)
```

```
at com.tmax.tibero.jdbc.comm.TbStream.readMsgBody(TbStream.java:327)
at com.tmax.tibero.jdbc.comm.TbCommType4.executeDirect(TbCommType4.java:460)
at com.tmax.tibero.jdbc.TbStatement.executeInternal(TbStatement.java:1051)
at com.tmax.tibero.jdbc.TbStatement.execute(TbStatement.java:560)
at TestSimple.main(TestSimple.java:102)
```

제4장 DataSource 객체와 데이터베이스 URL

DataSource 객체는 데이터베이스의 모든 리소스를 지칭하는 포괄적인 개념으로 JDBC 2.0 표준의 확장 API로 처음 소개되었다. 여기서 설명하는 DataSource는 하나의 데이터베이스에 대응된다.

본 장에서는 tbJDBC에서 제공하는 DataSource 객체를 이용하여 데이터베이스에 연결하는 방법과 데이터베이스 URL에 대해 설명한다.

4.1. DataSource 객체

javax.sql.DataSource에 정의된 기본 인터페이스는 다음과 같다.

```
public interface DataSource
{
    Connection getConnection() throws SQLException;
    Connection getConnection(String username, String password) throws SQLException;
    ...
}
```

Tibero에서는 com.tmax.tibero.jdbc.ext 패키지를 제공하고 있다. 즉, 각종 DataSource 객체의 속성을 설정할 수 있는 메소드를 제공하여 애플리케이션 프로그램 개발자의 편의를 제공하고 있다.

4.1.1. DataSource 객체의 속성

다음은 DataSource 객체에 설정할 수 있는 속성이다.

속성	타입	설명
databaseName	String	서버에 존재하는 특정 데이터베이스의 이름이다.
dataSourceName	String	DataSource의 이름이다.
description	String	DataSource에 대한 설명이다.
networkProtocol	String	서버와 통신하는 네트워크 프로토콜의 이름이다. (기본값: TCP)
password	String	서버 접속을 위한 패스워드이다.
portNumber	int	서버 리스너의 포트 번호이다.
serverName	String	데이터베이스의 이름이다.
user	String	서버 접속을 위한 사용자 이름이다.

다음은 위 표에서 설명한 DataSource 객체의 속성을 설정할 수 있는 메소드이다.

- public String getDatabaseName()
- public void setDatabaseName(String databaseName)
- public String getDataSourceName()
- public void setDataSourceName(String dataSourceName)
- public String getDescription()
- public void setDescription(String description)
- public String getNetworkProtocol()
- public void setNetworkProtocol(String networkProtocol)
- public String getPassword()
- public void setPassword(String password)
- public int getPortNumber()
- public void setPortNumber(int portNumber)
- public String getServerName()
- public void setServerName(String serverName)
- public String getUser()
- public void setUser(String user)

4.1.2. DataSource 객체의 추가 속성

다음은 추가로 설정할 수 있는 DataSource 객체의 속성이다.

속성	타입	설명
driverType	String	JDBC Driver의 타입이다.
login_timeout	int	소켓의 read timeout을 지정한다. 최소 소켓을 생성할 때부터 DB 연결 생성이 완료될 때까지 적용된다. DB 연결 생성이 완료된 뒤에는 read_timeout 속성이 적용된다. 만약 응답이 없이 지정된 시간이 지나면 timeout을 발생시키지만 설정값이 0일 경우에는 timeout을 발생시키지 않는다. (단위: millisecond, 기본값: 0)
logWriter	java.io.PrintWriter	DataSource 객체를 위한 Log Writer이다.
maxStatements	int	애플리케이션 프로그램 캐시에서 저장할 문장의 최댓값이다.

속성	타입	설명
read_timeout	int	DB 연결 생성이 완료된 이후의 소켓의 read timeout을 지정한다. 만약 응답이 없이 지정된 시간이 지나면 timeout을 발생시키지만 설정값이 0일 경우에는 timeout을 발생시키지 않는다. (단위: millisecond, 기본값: 0)
program_name	String	프로그램 이름이다. (기본값: JDBC Thin Client)
self_keepalive	String	SELF KEEP ALIVE 기능을 활성화할지 여부를 지정한다. 활성화한 경우에는 self_keepidle, self_keepintvl, self_keepcnt 속성 설정에 따라 연결 대상에 대한 네트워크 접근에 문제가 없는지 확인한다. 전체 확인 과정에 실패하면, 해당 네트워크 연결을 강제로 종료시킨다. (기본값: false)
self_keepidle	int	접속 완료, DB 요청/응답 처리 등의 정상적인 네트워크 사용 이후, 다음 번의 정상 처리된 시간이 갱신되지 않을 때, 정상 상황으로 간주할 최대 시간을 지정한다. 지정된 시간이 지나면 네트워크 접근에 대한 확인 절차가 시작된다. **self_keepalive 설정이 true인 경우에만 유효하다.** (단위: 초, 기본값: 60)
self_keepintvl	int	매 확인할 때의 간격 및 한 번 확인할 때 최대 대기 시간을 지정한다. **self_keepalive 설정이 true인 경우에만 유효하다.** (단위: 초, 기본값: 10)
self_keepcnt	int	확인 절차를 총 몇 회까지 수행할 것인지를 지정한다. 지정된 횟수만큼 연속으로 실패하여야만 전체 확인과정이 실패한 것으로 처리된다. **self_keepalive 설정이 true인 경우에만 유효하다.** (기본값: 3)
URL	String	데이터베이스 연결을 위한 데이터베이스 URL이다.

다음은 위 표에서 설명한 DataSource 객체의 추가 속성을 설정할 수 있는 메소드이다.

- public String getDriverType()
- public void setDriverType(String driverType)
- public int getLoginTimeout()
- public void setLoginTimeout(int time)
- public PrintWriter getLogWriter()
- public void setLogWriter(PrintWriter writer)

- public int getMaxStatements()
- public void setMaxStatements(int maxStatements)
- public int getReadTimeout()
- public void setReadTimeout(int readTimeout)
- public String getProgramName()
- public void setProgramName(String name)
- public boolean getSelfKeepalive()
- public void setSelfKeepalive(boolean keepalive)
- public int getSelfKeepidle()
- public void setSelfKeepidle(int keepidle)
- public int getSelfKeepintvl()
- public void setSelfKeepintvl(int keepintvl)
- public int getSelfKeepcnt()
- public void setSelfKeepcnt(int keepcnt)
- public String getURL()
- public void setURL(String url)

4.2. DataSource 객체를 이용한 연결

DataSource 객체를 이용하여 데이터베이스에 연결하는 방법은 JNDI를 사용하지 않는 방법과 사용하는 방법이 있다. JNDI(Java Naming and Directory Interface)는 Java로 작성된 애플리케이션 프로그램이 DNS, NDS 등과 같은 네이밍, 디렉터리 서비스에 접근하기 위한 API이다.

4.2.1. JNDI를 사용하지 않는 방법

다음은 JNDI를 사용하지 않고 DataSource 객체를 사용하여 데이터베이스에 연결하는 가장 일반적인 방법이다. 방법은 TbDataSource 객체를 생성한 후 기본 속성을 설정하면 된다.

예를 들면 다음과 같다.

```
TbDataSource ds = new TbDataSource();

ds.setDriverType("thin");
ds.setServerName("tmaxh4");
ds.setNetworkProtocol("tcp");
```

```
ds.setDatabaseName("tibero");
ds.setPortNumber(8888);
ds.setUser("tibero");
ds.setPassword("tmax");

Connection conn = ds.getConnection();
```

4.2.2. JNDI를 사용한 방법

다음은 JNDI를 사용하여 데이터베이스에 연결하는 예이다.

1. TbDataSource 객체를 생성한 후 기본 속성을 설정한다.

```
TbDataSource ds = new TbDataSource();

ds.setDriverType("thin");
ds.setServerName("tmaxh4");
ds.setNetworkProtocol("tcp");
ds.setDatabaseName("tibero");
ds.setPortNumber(8888);
ds.setUser("tibero");
ds.setPassword("tmax");
```

2. 초기화된 DataSource 객체를 JNDI에 등록한다.

```
Context ctx = new InitialContext();
ctx.bind("tibero/webdb", ds);
```

InitialContext()를 호출하면 JNDI를 참조하는 context 객체가 생성되고, ctx.bind()를 호출하면 DataSource 객체와 JNDI 이름이 연결된다. 즉, 이것은 위에서 연결시킨 'tibero/webdb'만을 이용하여 다음의 예처럼 언제든지 데이터베이스에 연결할 수 있다는 말이다.

```
TbDataSource ds = (TbDataSource)ctx.lookup("tibero/webdb");
Connection conn = ds.getConnection();
```

4.3. 데이터베이스 URL과 데이터베이스 지시자

데이터베이스 URL(Uniform Resource Locator)은 다음과 같이 문자열 값으로 사용한다.

```
jdbc:tibero:thin:@database_sepcifier
```

URL에 user name, password를 포함하는 형태도 제공한다.

```
jdbc:tibero:thin:user/password@database_sepcifier
```

Tibero에서 제공하는 데이터베이스 지시자(database_specifier)는 다음과 같이 사용할 수 있다.

```
hostname:port:[service_name]
```

또한 여러 개의 데이터베이스 연결을 지정하기 위한 DESCRIPTION 형태도 제공한다.

```
(DESCRIPTION=
(LOAD_BALANCE=ON)
(FAILOVER=ON)
(PROTOCOL=TCP)
(ADDRESS_LIST=
 (ADDRESS=(HOST=dbsvr1)(PORT=8629))
 (ADDRESS=(HOST=dbsvr2)(PORT=7629))
 ...
))
```

위의 예에서 LOAD_BALANCE와 FAILOVER 기능은 선택적으로 적용할 수 있다. 단, ADDRESS_LIST는 하나 이상의 ADDRESS를 반드시 명시해야 한다.

제5장 분산 트랜잭션

본 장에서는 tbJDBC에서 제공하는 분산 트랜잭션(Distributed Transaction) 기능을 설명한다. 분산 트랜잭션의 대표적인 예로는 XA(Extended Architecture)가 있으며 이와 관련된 설명을 주로 한다.

5.1. 개요

분산 트랜잭션은 보통 전역 트랜잭션이라고도 하는데, 통합으로 관리되는 하나 이상의 트랜잭션 집합을 말한다. 분산 트랜잭션에 참여하는 트랜잭션은 동일한 데이터베이스에 존재할 수도 있고, 다른 데이터베이스에 존재할 수도 있다. 이러한 각각의 트랜잭션을 **트랜잭션 브랜치**(Transaction Branch)라고 한다.

분산 트랜잭션은 JDBC 2.0 표준의 확장 API로서 접속 풀링 기능을 기반으로 제공되거나 X/Open DTP(Distributed Transaction Processing) 규약의 XA 표준으로 제공되기도 한다.

분산 트랜잭션은 다음과 같은 특징이 있다.

- 분산 트랜잭션은 개별 트랜잭션을 관리하기 위해 외부의 트랜잭션 관리자를 이용한다.

  트랜잭션 관리자는 Java Transaction API 표준을 구현한 소프트웨어 요소로서 여러 벤더(vendor)에서 XA 호환 JTA 모듈을 제공한다.

- XA 기능은 애플리케이션 프로그램과는 별도로 구분되며, 대부분 애플리케이션 프로그램 서버(Application Server)와 같은 Middle-tier에서 사용된다.

- 리소스 매니저(Resource Manager)는 데이터나 다른 종류의 리소스를 지칭하는 용어로 본 장에서는 데이터베이스를 가리킨다.

5.2. 구성요소

다음은 XA 구성요소별 기능 설명이다.

- XADataSource

  XADataSource는 Connection Pool DataSource나 다른 DataSource와 비슷한 개념과 기능을 가진다. 분산 트랜잭션에서 사용되는 각각의 리소스 매니저(데이터베이스)에는 하나의 XADataSource 객체가 존재하고, 이 XADataSource가 XAConnection을 생성한다.

- XAConnection

  XAConnection은 Pooled Connection의 확장이며, 개념이나 기능면에서는 비슷하다.

XAConnection은 물리적인 데이터베이스 연결에 대한 임시 핸들이 되고, 하나의 XA Connection은 하나의 데이터베이스 세션에 해당한다.

- XAResource

  XAResource는 분산 트랜잭션의 각 트랜잭션 브랜치를 조합하기 위해 트랜잭션 관리자에 의해 사용된다. 각각의 XAConnection으로부터 하나의 XAResource 객체를 얻어올 수 있고, 1:1 관계를 가진다.

  따라서 하나의 XAResource 객체는 하나의 데이터베이스 세션에 해당된다. 하나의 XAResource 객체는 실행 중인 오직 하나의 트랜잭션 브랜치만 있을 수 있다. 그러나 실행 중인 트랜잭션과는 별개로 정지된 트랜잭션도 있을 수 있다. 각각의 XAResource 객체는 해당 세션에서 수행되며 시작, 종료, 준비, 커밋, 롤백 등의 기능이 있다.

- XID

  각각의 트랜잭션 브랜치를 구분하기 위해 XID(트랜잭션 ID)를 사용하고, 하나의 XID는 트랜잭션 브랜치 ID와 분산 트랜잭션 ID로 구성된다.

5.3. 전역 트랜잭션과 지역 트랜잭션 간의 변환

JDBC 3.0 표준에서는 전역 트랜잭션과 지역 트랜잭션 사이에서 커넥션을 공유할 수 있고, 각각으로 변환할 수 있다.

일반적으로 커넥션은 다음의 세 가지 모드 중에 반드시 하나를 갖는다.

모드	**설명**
NO_TXN	현재 커넥션을 사용하는 활성화된 트랜잭션이 없는 모드이다.
LOCAL_TXN	auto-commit 모드를 비활성화시킨 상태로 현재 커넥션을 사용하는 활성화된 트랜잭션이 있는 모드이다. 커넥션 모드에 따라 prepare(), commit(), rollback(), forget(), end() 메소드를 호출할 수 없고, XAException이 발생한다.
GLOBAL_TXN	현재 커넥션을 사용하는 활성화된 트랜잭션이 있는 모드이다. 커넥션 모드에 따라 commit(), rollback(), setAutoCommit(), setSavepoint() 메소드를 호출할 수 없고, SQLException이 발생한다.

각 커넥션은 실행 상태에 따라 다음과 같이 세 가지 모드 사이에서 자동으로 바뀐다. 단, 커넥션이 초기화될 때에는 항상 NO_TXN 모드로 동작한다.

현재 모드	NO_TXN으로 변환	LOCAL_TXN으로 변환	GLOBAL_TXN으로 변환
NO_TXN	-	auto-commit 모드를 비활성화시키고 DML 문을 수행했을 때	XAConnection에서 얻은 XAResource에 end() 메소드를 호출했을 때
LOCAL_TXN	DDL 문이 수행되거나 commit() 또는 rollback() 메소드를 호출했을 때	-	불가능
GLOBAL_TXN	XAConnection에서 얻은 XAResource에 end() 메소드를 호출했을 때	불가능	-

5.4. Tibero XA 패키지

Tibero에서는 XA 표준에 따른 분산 트랜잭션 패키지인 com.tmax.tibero.jdbc.ext 내부에 다음과 같은 클래스를 지원한다.

- TbXAConnection
- TbXADataSource
- TbXAException
- TbXAResource
- TbXid

5.5. XA 인터페이스의 구성요소

본 절에서는 JDBC 2.0 표준 패키지에 정의된 XA 인터페이스의 구성요소를 설명한다.

5.5.1. XADataSource 인터페이스

javax.sql.XADataSource에 정의된 인터페이스는 다음과 같다.

```
public interface XADataSource
{
  XAConnection getXAConnection() throws SQLException;
  XAConnection getXAConnection(String user, String password) throws SQLException;
  ...
}
```

tbJDBC에서는 com.tmax.tibero.jdbc.ext.TbXADataSource 클래스로 제공된다. 또한 TbConnectionPoolDataSource를 확장하여 일반적인 DataSource에서 사용할 수 있는 커넥션의 특성을 사용할 수도 있다.

5.5.2. XAConnection 인터페이스

javax.sql.XAConnection에 정의된 인터페이스는 다음과 같다.

```
public interface XAConnection extends PooledConnection
{
  javax.transaction.xa.XAResource getXAResource() throws SQLException;
}
```

tbJDBC에서는 com.tmax.tibero.jdbc.ext.TbXAConnection 클래스로 제공된다.

5.5.3. XAResource 인터페이스

javax.transaction.xa.XAResource에 정의된 인터페이스는 다음과 같다.

```
public interface XAResource
{
  void commit(Xid xid, boolean onePhase) throws XAException;
  void end(Xid xid, int flags) throws XAException;
  void forget(Xid xid) throws XAException;
  int prepare(Xid xid) throws XAException;
  int rollback(Xid xid) throws XAException;
  void start(Xid xid, int flags) throws XAException;
  boolean isSameRM(XAResource xares) throws XAException;
}
```

tbJDBC에서는 com.tmax.tibero.jdbc.ext.TbXAResource 클래스로 제공된다.

XAResource 인터페이스에서 설정할 수 있는 메소드는 다음과 같다.

메소드	설명
Start	XID와 관련된 트랜잭션의 특정 브랜치를 시작하거나 이미 존재하는 트랜잭션을 재시작 또는 변경 사항을 조인한다.
End	XID와 관련된 트랜잭션의 특정 브랜치에 대한 종료 상태(정상 또는 실패)를 알리거나 이미 존재하는 트랜 잭션을 멈춘다.
Prepare	현재의 트랜잭션 브랜치에서 수행된 변경사항을 적용하기 위해 준비한다.
Commit	현재의 트랜잭션 브랜치의 변화를 반영한다.
Rollback	현재의 트랜잭션 브랜치의 변화를 롤백시킨다.
Forget	리소스 매니저가 지정한 트랜잭션 브랜치를 무시하도록 한다.
Recover	현재 준비가 완료되었거나 수행이 끝난 트랜잭션 브랜치의 목록을 반환한다.
isSameRM	두 개의 XA 리소스 객체가 동일한 리소스 매니저에 속해 있는지의 여부를 반환한다.

Start

XID와 관련된 트랜잭션의 특정 브랜치를 시작하거나 이미 존재하는 트랜잭션을 재시작 또는 변경 사항을 조인시키기 위해 사용하는 메소드이다.

- 문법

```
void start(Xid xid, int flags)
```

- 파라미터

파라미터	설명
xid	현재 리소스 객체와 관련된 글로벌 트랜잭션의 ID이다.
flags	flags 파라미터는 반드시 다음의 값 중의 하나여야 한다. – XAResource.TMNOFLAGS : 새로운 트랜잭션 브랜치가 시작된다. – XAResource.TMJOIN : 이미 존재하는 트랜잭션 브랜치에 다음 동작을 조인시킨다. – XAResource.TMRESUME : 정지된 트랜잭션을 다시 시작시킨다. – TbXAResource.TBRTMSERIALIZABLE : Serializable 트랜잭션을 시작시킨다. – TbXAResource.TBRTMREADONLY : Read-only 트랜잭션을 시작시킨다. – TbXAResource.TBRTMREADWRITE : Read/write 트랜잭션을 시작시킨다. – TbXAResource.TBRTRANSLOOSE : Loosely-coupled 트랜잭션을 시작시킨다.

End

XID와 관련된 트랜잭션의 특정 브랜치에 대한 종료 상태(정상 또는 실패)를 알리거나 이미 존재하는 트랜잭션을 멈추기 위해 사용하는 메소드이다.

- 문법

```
void end(Xid xid, int flags)
```

- 파라미터

파라미터	설명
xid	현재 리소스 객체와 관련된 글로벌 트랜잭션의 ID이다.
flags	flags 파라미터는 반드시 다음의 값 중의 하나여야 한다.

파라미터	설명
	– XAResource.TMSUCCESS : 현재 트랜잭션 브랜치가 성공했음을 알린다. – XAResource.TMFAIL : 현재 트랜잭션 브랜치가 실패했음을 알린다. – XAResource.TMSUSPEND : 현재 트랜잭션 브랜치를 정지시킨다.

Prepare

현재의 트랜잭션 브랜치에서 수행된 변경사항을 적용하기 위해 준비하는 과정으로 Two-phase commit 과정의 첫 번째 단계이다. 만약 분산 트랜잭션 내부에 오직 하나의 트랜잭션만 있는 경우라면 prepare()를 수행할 필요가 없다.

- 문법

```
int prepare(Xid xid)
```

- 파라미터

파라미터	설명
xid	현재 리소스 객체와 관련된 글로벌 트랜잭션의 ID이다.

- 반환 값

  이 메소드는 반드시 다음의 값 중의 하나를 반환한다.

반환 값	설명
XAResource.XA_RDONLY	현재의 트랜잭션 브랜치는 read-only 모드로 동작하므로 SELECT 문만 수행할 수 있다.
XAResource.XA_OK	현재의 트랜잭션 브랜치에서는 어떠한 수정 작업도 수행할 수 있다.

Commit

현재의 트랜잭션 브랜치의 변화를 반영하는 과정으로 Two-phase commit 과정의 두 번째 단계이다. 단, 모든 트랜잭션 브랜치가 prepare를 완료한 후에 수행된다.

- 문법

```
void commit(Xid xid, boolean isOnePhase)
```

- 파라미터

파라미터	설명
xid	현재 리소스 객체와 관련된 글로벌 트랜잭션의 ID이다.

파라미터	설명
isOnePhase	isOnePhase 파라미터에 설정된 값에 따라 다음과 같이 동작한다. – true : Two-phase commit 과정이 아닌 One-phase commit 과정으로 동작한다. – false : Two-phase commit 과정으로 동작한다.

Rollback

현재의 트랜잭션 브랜치의 변화를 롤백시킨다.

- 문법

```
void rollback(Xid xid)
```

- 파라미터

파라미터	설명
xid	현재 리소스 객체와 관련된 글로벌 트랜잭션의 ID이다.

Forget

리소스 매니저가 지정한 트랜잭션 브랜치를 무시하도록 한다.

- 문법

```
void forget(Xid xid)
```

- 파라미터

파라미터	설명
xid	현재 리소스 객체와 관련된 글로벌 트랜잭션의 ID이다.

Recover

현재 준비가 완료되었거나 수행이 끝난 트랜잭션 브랜치의 목록을 반환한다.

- 문법

```
Xid[] recover(int flag)
```

- 파라미터

파라미터	설명
flag	flags 파라미터는 반드시 다음의 값 중의 하나여야 한다. – XAResource.TMSTARTSCAN : 현재 준비가 완료된 트랜잭션을 반환한다. – XAResource.TMENDSCAN : 현재 수행이 끝난 트랜잭션을 반환한다. – XAResource.TMNOFLAGS : 모든 트랜잭션을 반환한다.

isSameRM

두 개의 XA 리소스 객체가 동일한 리소스 매니저에 속해 있는지의 여부를 반환한다.

- 문법

```
boolean isSameRM(XAResource aResource)
```

- 파라미터

파라미터	설명
aResource	현재 리소스 객체와 비교할 리소스 객체이다.

5.5.4. XID 인터페이스

트랜잭션 관리자는 XID 객체를 생성하고, 이를 이용하여 분산 트랜잭션의 브랜치를 관리한다. 각각의 트랜잭션 브랜치는 개별적으로 XID를 부여받는다.

XID는 다음의 정보를 포함한다.

- 포맷 지시자(4bytes)

  Java 트랜잭션 관리자를 가리키며, NULL이 될 수 없다. 이 정보를 얻기 위한 메소드는 다음과 같다.

```
public int getFormatId()
```

- 전역 트랜잭션 지시자(64bytes)

  동일한 분산 트랜잭션에 속하는 트랜잭션 브랜치의 경우 모두 같은 값을 가진다. 이 정보를 얻기 위한 메소드는 다음과 같다.

```
public byte[] getGlobalTransactionId()
```

- 브랜치 지시자(64bytes)

  이 정보를 얻기 위한 메소드는 다음과 같다.

```
public byte[] getBracheQualifier()
```

javax.transaction.xa.Xid에 정의된 인터페이스는 다음과 같다.

```
public TbXid(int formatId, byte[] globalId, byte[] branchId) throws XAException
```

tbJDBC에서는 com.tmax.tibero.jdbc.ext.TbXid 클래스로 제공된다.

5.6. 분산 트랜잭션 예제

다음은 두 개의 트랜잭션 브랜치로 이루어진 Two-phase 분산 트랜잭션 환경을 구현하는 순서이다.

1. 트랜잭션 브랜치 #1을 시작한다.
2. 트랜잭션 브랜치 #2를 시작한다.
3. 트랜잭션 브랜치 #1에서 DML 문장을 수행한다.
4. 트랜잭션 브랜치 #2에서 DML 문장을 수행한다.
5. 트랜잭션 브랜치 #1의 트랜잭션을 종료한다.
6. 트랜잭션 브랜치 #2의 트랜잭션을 종료한다.
7. 트랜잭션 브랜치 #1을 준비한다.
8. 트랜잭션 브랜치 #2를 준비한다.
9. 트랜잭션 브랜치 #1을 커밋한다.
10. 트랜잭션 브랜치 #2를 커밋한다.

다음의 예는 위 순서에 맞게 구현된 Java 프로그램 소스 코드이다.

```
import java.sql.*;
import javax.sql.XAConnection;
import javax.transaction.xa.XAResource;
import com.tmax.tibero.jdbc.ext.*;

public class TwoBranchXA
{
  public static void main(String args[]) throws SQLException
  {
    createBaseTable();
    try {
          // Create XADataSource instance
          TbXADataSource xads1 = new TbXADataSource();
          xads1.setUrl("jdbc:tibero:thin:@localhost:7629:dbsvr");
          xads1.setUser("tibero");
          xads1.setPassword("tmax");
```

```
TbXADataSource xads2 = new TbXADataSource();
xads2.setUrl("jdbc:tibero:thin:@localhost:8629:dbsvr");
xads2.setUser("wrpark");
xads2.setPassword("tmax");

// Get the XA connection
XAConnection xacon1 = xads1.getXAConnection();
XAConnection xacon2 = xads2.getXAConnection();

// Get the physical connection
Connection conn1 = xacon1.getConnection();
Connection conn2 = xacon2.getConnection();

// Get the XA resource
XAResource xars1 = xacon1.getXAResource();
XAResource xars2 = xacon2.getXAResource();

// Create the Xid
Xid xid1 = createXid(1);
Xid xid2 = createXid(2);

// Start the resource
xars1.start(xid1, XAResource.TMNOFLAGS);
xars2.start(xid2, XAResource.TMNOFLAGS);

// Execute SQL operations
execute1(conn1);
execute2(conn2);

// End both the branches
xars1.end(xid1, XAResource.TMSUCCESS);
xars2.end(xid2, XAResource.TMSUCCESS);

// Prepare the resource manager
int pre1 = xars1.prepare(xid1);
int pre2 = xars2.prepare(xid2);

// Commit or rollback
if (pre1 == XAResource.XA_RDONLY || pre1 == XAResource.XA_OK)
    xars1.commit(xid1, false);
else
    xars1.rollback(xid1);

if (pre2 == XAResource.XA_RDONLY || pre2 == XAResource.XA_OK)
    xars2.commit(xid2, false);
else
```

```
            xars2.rollback(xid2);

          // Clear
          conn1.close();
          conn1 = null;
          conn2.close();
          conn2 = null;

          xacon1.close();
          xacon1 = null;
          xacon2.close();
          xacon2 = null;
        }
    catch (Exception se) {
          se.printStackTrace();
        }
  }
}
```

제6장 결과 집합 확장기능

본 장에서는 JDBC 2.0 표준에서 제공하는 기능인 Scrollable, Updatable 결과 집합의 확장기능을 설명한다.

6.1. JDBC 2.0 표준

JDBC 2.0 표준에서는 scrollability, positioning, sensitivity, updatability에 대한 확장기능을 제공한다.

- 결과 집합 타입에 의해 scrollability, positioning, sensitivity가 결정된다.
- 동시성 타입에 의해 updatability가 결정된다.

6.1.1. Scrollability, Positioning, Sensitivity

Scrollability, Positioning은 결과 집합에 대해 정방향뿐만 아니라 역방향으로도 움직일 수 있고, 상대 위치나 절대 위치 등의 임의의 위치로 움직일 수 있는 기능이다. 여기서 상대 위치는 현재 열의 위치로부터 정방향 또는 역방향으로의 이동을 말하고, 절대 위치는 결과 집합의 시작이나 끝에서부터의 이동을 말한다.

주의

Scrollablility 기능을 사용할 경우 결과 집합의 모든 열은 사용자의 메모리에 로드되기 때문에 이를 주의해야 한다. 그리고 될 수 있으면 용량이 큰 데이터를 갖는 결과 집합은 사용하지 말 것을 권장한다.

Scrollability, Positioning 결과 집합을 생성할 때는 반드시 Sensitivity를 설정해야 하는데, 이는 현재의 결과 집합에 관계없이 데이터베이스에 적용된 변경 사항을 반영할 것인지의 여부를 결정한다. Sensitivity는 데이터베이스에 적용된 변경 사항이 바로 적용되어 새로운 데이터를 볼 수 있다. 반면에 Insensitivity는 최초 결과 집합이 생성된 시점의 데이터만 볼 수 있다.

결과 집합을 생성할 때 결정할 수 있는 결과 집합의 타입은 다음과 같다.

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_SENSITIVE
- ResultSet.TYPE_SCROLL_INSENSITIVE

6.1.2. Updatability

Updatability는 결과 집합에 직접 수정한 후에 데이터베이스에 반영할 수 있는 기능이다. 예를 들어 새로운 열을 추가하거나 기존의 열을 지우거나 수정하는 것이 이에 해당된다.

주의

이 기능은 데이터베이스에 접근해야 하므로, 잠금(Lock) 설정이 필요할 수도 있다는 점을 주의해야 한다.

결과 집합을 생성할 때 결정할 수 있는 동시성 타입은 다음과 같다.

- ResultSet.CONCUR_UPDATABLE
- ResultSet.CONCUR_READ_ONLY

6.2. Scrollable, Updatable 결과 집합 생성

6.2.1. Statement 객체 생성

tbJDBC에서는 Connection 클래스에 다음과 같은 메소드를 제공한다.

- Statement createStatement(int resultSetType, int resultSetConcurrency)
- PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)
- CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)

다음은 Statement 객체를 생성하는 예이다.

```
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                      ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT empno FROM emp");
```

6.2.2. 결과 집합 특성 확인

Statement, PreparedStatement, CallableStatement 객체를 생성한 후에는 다음의 메소드를 사용하여 결과 집합 타입과 동시성 타입을 확인할 수 있다.

- int getResultSetType() throws SQLException
- int getResultSetConcurrent() throws SQLException

6.2.3. 제약 사항

Updatable 결과 집합을 생성할 때는 다음과 같은 제약 사항이 존재한다.

- 오직 하나의 테이블에 대한 질의문만 사용할 수 있고, 조인(join)은 사용할 수 없다.
- 'SELECT *'와 같은 형태는 사용할 수 없고, 'SELECT T.*'와 같은 형태만 사용할 수 있다.
- 오직 테이블의 컬럼에 대해서만 질의문을 사용할 수 있다.

Scroll-sensitive 결과 집합을 생성할 때는 다음과 같은 제약 사항이 존재한다.

- 오직 하나의 테이블에 대한 질의문만 사용할 수 있다.
- 'SELECT *'와 같은 형태는 사용할 수 없고, 'SELECT t.*'와 같은 형태만 사용할 수 있다.

6.3. Scrollable 결과 집합 탐색

다음은 Scrollable 결과 집합을 위해 제공되는 메소드로, 새로운 위치로 이동할 수 있다.

메소드	설명
boolean next() throws SQLException	현재 위치에서 다음 열로 이동한다. 만약 더는 다음 열이 존재하지 않는 경우 false를 반환한다.
boolean previous() throws SQLException	현재 위치에서 이전 열로 이동한다. 만약 더는 이전 열이 존재하지 않는 경우 false를 반환한다.
boolean first() throws SQLException	결과 집합의 첫 번째 열로 이동한다. 만약 열 데이터가 하나도 없을 경우 false를 반환한다.
boolean last() throws SQLException	결과 집합의 마지막 열로 이동한다. 만약 열 데이터가 하나도 없을 경우 false를 반환한다.
boolean absolute(int row) throws SQLException	결과 집합의 처음이나 마지막 열의 위치에서부터 정해진 값만큼 절대 위치로 이동한다. 만약 입력 값이 양수인 경우에는 처음 열부터 정방향으로 이동하고, 음수인 경우에는 마지막 열부터 역방향으로 이동한다. 만약 결과 집합의 개수보다 더 큰 양수 값을 사용할 경우에는 마지막 열 다음으로 이동하며, 이는 afterLast()와 같은 효과를 가진다. 마찬가지로 결과 집합의 개수보다 더 큰 음수 값을 사용할 경우에는 beforeFirst()와 같은 효과를 가진다.
boolean relative(int row) throws SQLException	현재 열의 위치로부터 시작하여 양수 값이면 정방향으로 이동하고, 음수 값이면 역방향으로 이동한다. 만약 결과 집합의 개수보다 더 큰 양수 값을 사용할 경우에는 마지막 열 다음으로 이동하며, 이는 afterLast()와 같은 효과를 가진다. 마찬가지로 결과 집합의 개수보다 더 큰 음수 값을 사용한 경우에는 beforeFirst()와 같은 효과를 가진다.

메소드	설명
	주의할 점은 반드시 현재 열의 위치가 적절해야 한다는 것이다. 처음 열의 위치 이전이나 마지막 열의 위치 다음에서부터 상대 위치로의 이동은 할 수 없으며, SQLException을 발생시킨다.
void beforeFirst() throws SQLException	결과 집합의 첫 번째 열 이전으로 이동한다. 이는 정방향으로 결과 집합을 탐색할 때의 상태이며 바로 사용할 수 있는 열 데이터는 없다.
void afterLast() throws SQLException	결과 집합의 마지막 열 다음으로 이동한다. 이는 역방향으로 결과 집합을 탐색할 때의 상태이며 바로 사용할 수 있는 열 데이터는 없다.

현재의 위치 정보를 알기 위해 tbJDBC에서는 다음과 같은 메소드를 제공한다.

메소드	설명
int getRow() throws SQLException	현재 열의 위치를 반환하며, 적합한 열의 위치가 아닌 경우에는 0을 반환한다.
boolean isFirst() throws SQLException	현재 열의 위치가 첫 번째 열인지 확인한다.
boolean isLast() throws SQLException	현재 열의 위치가 마지막 열인지 확인한다.
boolean isBeforeFirst() throws SQLException	현재 열의 위치가 첫 번째 열의 이전 열인지 확인한다.
boolean isAfterLast() throws SQLException	현재 열의 위치가 마지막 열의 다음 열인지 확인한다.

6.4. Updatable 결과 집합 탐색

사용자는 Updatable 결과 집합을 이용하여 열의 데이터를 업데이트할 수 있고, 삭제할 수도 있으며, 새로운 열의 데이터를 입력할 수도 있다.

UPDATE나 INSERT를 수행한 후에는 반드시 별도의 단계를 거쳐 데이터베이스에 변경 사항을 반영해야 한다. 그렇지 않으면 변경된 모든 데이터는 소멸된다. 반면에 DELETE를 수행한 경우에는 별도의 단계 없이 즉시 데이터베이스에 반영된다.

6.4.1. INSERT

INSERT 작업은 다음의 과정을 거쳐 데이터베이스에 반영된다.

1. moveToInsertRow() 메소드를 이용하여 INSERT를 수행하기 위한 임시 열 데이터 저장 공간으로 이동한다. 결과 집합은 기존의 열의 위치를 내부적으로 기억하고 있으므로 moveToCurrentRow() 메소드를 사용할 경우 원래 열의 위치로 이동할 수 있다.

2. 적절한 updateXXX() 메소드를 수행하여 컬럼의 데이터를 작성한다. 만약 특정 컬럼의 데이터를 작성하지 않는다면, NULL 상태로 남아 있게 된다.

3. insertRow() 메소드를 수행하여 데이터베이스에 반영한다.

   만약 INSERT를 수행한 후에 이를 취소하려면, 다른 열의 위치로 이동하여 원래의 데이터로 복원하면 된다. 그러나 반드시 다음의 사항을 주의해야 한다.

   – insertRow() 메소드를 수행한 경우 데이터베이스에 반영이 되므로 롤백하기 전에는 취소되지 않는다.

   – 어떠한 결과 집합의 타입도 INSERT 작업에 의해 수행된 열의 데이터를 볼 수 없다.

다음은 INSERT 작업을 수행하는 예이다.

```
rs.moveToInsertRow();
rs.updateString(1, "tibero");
rs.insertRow();
rs.moveToCurrentRow();
```

6.4.2. UPDATE

UPDATE 작업은 다음의 두 단계를 거쳐야 데이터베이스에 최종으로 반영된다.

1. updateXXX() 메소드를 수행한다.

2. updateRow() 메소드를 수행하여 데이터베이스에 반영한다. 이때 auto-commit 모드에 따라 커밋이 될 수도 있다.

   만약 UPDATE를 수행한 후에 이를 취소하려면, cancelRowUpdates() 메소드를 수행하거나 다른 열의 위치로 이동할 경우 원래의 데이터로 복원하면 된다. 그러나 updateRow() 메소드를 수행한 경우라면 데이터베이스에 반영이 되므로 롤백하기 전에는 취소되지 않음을 주의해야 한다.

다음은 UPDATE 작업을 수행하는 예이다.

```
rs.absolute(5);
rs.updateString(1, "tibero");
rs.updateRow();
```

6.4.3. DELETE

DELETE 작업을 수행할 때는 반드시 다음의 사항을 주의해야 한다.

- DELETE 작업을 수행할 경우 데이터베이스에 바로 적용되므로 auto-commit 모드의 설정 값에 따라 바로 커밋이 될 수도 있다.

- 사용자에 의해 지워진 열 데이터의 경우 데이터베이스에는 반영되지만 결과 집합 자체에는 남아 있게 된다. 반대로 Scrollable 결과 집합인 경우에는 바로 재조정이 일어난다. 즉, 현재 열의 위치는 바로 이전의 열의 데이터를 가리키게 되고, 다음 열의 위치는 자동으로 당겨지게 된다.

다음의 메소드를 사용하여 DELETE 작업을 수행할 수 있다.

```
void deleteRow() throws SQLException
```

다음은 DELETE 작업을 수행하는 예이다.

```
rs.absolute(5);
rs.deleteRow();
```

제7장 Row Set

본 장에서는 tbJDBC에서 제공하는 Row Set 기능을 설명한다.

7.1. 개요

Row Set이란 문자 그대로 로우 데이터의 집합을 포함하는 객체이다. javax.sql.RowSet 인터페이스 메소드를 통해 접근할 수 있다.

일반적으로 Row Set은 다음과 같이 세 가지로 나뉜다.

- Cached Row Set(tbJDBC에서는 현재 이 Row Set만 제공한다.)
- JDBC Row Set
- Web Row Set

7.2. Row Set Listener

Row Set에서는 여러 개의 리스너를 등록하여 사용할 수 있다. 등록할 때는 addRowSetListener() 메소드를 사용하고, 제거할 때는 removeRowSetListener() 메소드를 사용하면 된다. 이때 사용되는 리스너는 반드시 javax.sql.RowSetListener 인터페이스를 통해서 구현되어야 한다.

RowSetListener 인터페이스에서 제공하는 이벤트는 다음과 같이 3가지이다.

이벤트	설명
cursorMoved	next()나 previous() 등의 메소드를 통해 열의 이동이 있을 때 마다 발생한다.
rowChanged	새로운 열이 추가되거나 기존의 열이 수정되거나 삭제될 때 발생한다.
rowSetChanged	전체 Row Set이 생성되거나 변경될 때 발생한다.

RowSetListener 인터페이스의 이벤트를 사용하는 방법은 다음의 예와 같다.

1. 리스너 클래스를 생성한다.

```
public class MyListener implements RowSetListener
{
    public void cursorMoved(RowSetEvent event) {
        // do work
    }
    public void rowChanged(RowSetEvent event) {
```

```
        // do work
    }
    public void rowSetChanged(RowSetEvent event) {
        // do work
    }
}
```

2. 리스너를 RowSet 객체에 등록한다.

```
MyListener mListener = new MyListener();
rowset.addRowSetListener(mListener);
```

7.3. Cached Row Set

Cached Row Set은 모든 열을 캐시에 저장하고, 데이터베이스와의 연결을 유지하지 않도록 구현된 Row Set의 한 형태이다. tbJDBC에서는 TbCachedRowSet 클래스로 제공된다.

7.3.1. RowSet 객체 생성

RowSet 객체는 execute() 메소드를 통해 사용할 수 있는 준비를 마치며, 그 이후에는 java.sql.ResultSet 객체를 사용하는 방법과 동일하게 사용할 수 있다.

본 절에서는 RowSet 객체를 생성하는 과정을 두 가지로 나누어 설명한다.

질의문을 이용한 RowSet 객체 생성

다음의 예는 질의문을 이용하여 RowSet 객체를 생성하는 과정이다.

1. TbCachedRowSet 객체를 생성한다.

```
TbCachedRowSet rowset = new TbCachedRowSet();
```

2. URL, 사용자 이름, 패스워드, 질의문을 설정한 후에 execute() 메소드를 수행하여 RowSet 객체를 생성한다.

```
rowset.setUrl("jdbc:tibero:thin:@localhost:8629:dbsvr");
rowset.setUsername("tibero");
rowset.setPassword("tmax");
rowset.setCommand("SELECT * FROM emp");
rowset.execute();
```

결과 집합을 이용한 RowSet 객체 생성

다음의 예는 기존에 존재하는 결과 집합을 이용하여 RowSet 객체를 생성하는 과정이다.

1. TbCachedRowSet 객체를 생성한다.

```
TbCachedRowSet rowset = new TbCachedRowSet();
```

2. 결과 집합 객체(rset)를 이용하여 populate() 메소드를 수행하여 RowSet 객체를 생성한다.

```
ResultSet rset = pstmt.executeQuery();
rowset.populate(rset);
```

7.3.2. 열 데이터 탐색

RowSet 객체를 이용하여 정방향이나 역방향으로 이동하면서 열의 데이터를 탐색할 수 있다.

예를 들면 다음과 같다.

```
rowset.beforeFirst();
while (rowset.next()) {
    System.out.println(rowset.getString(1));
}

rowset.afterLast();
while (rowset.previous()) {
    System.out.println(rowset.getString(1));
}
```

또한 열의 데이터를 삽입, 삭제, 수정할 수도 있다. 단, acceptChanges() 메소드를 수행해야 커밋이 실행된다.

예를 들면 다음과 같다.

```
rowset.absolute(5);
rowset.moveToInsertRow();
rowset.updateString(1, "tibero");
rowset.insertRow();
rowset.acceptChanges();
```

7.3.3. 제약 사항

Updatable 결과 집합에 적용되는 제약 사항과 동일하게 적용된다.

제8장 LOB 데이터 처리

본 장에서는 tbJDBC에서 LOB 데이터를 처리하는 방법을 설명한다.

8.1. 개요

JDBC 표준에서는 LOB 데이터를 처리하기 위해 두 가지 타입을 제공한다. BLOB(바이너리 데이터)와 CLOB(문자열 데이터)가 이에 해당한다. tbJDBC에서는 이 두 가지 타입을 효율적으로 처리하기 위해 지시자(locator)라는 개념을 사용한다.

일반적으로 LOB 데이터는 데이터베이스에 저장된 후에 지시자를 생성하여 데이터가 저장된 위치를 찾아갈 수 있도록 한다. 따라서 사용자는 이 지시자만을 사용하여 필요할 때마다 LOB 데이터를 읽고 쓸 수 있다. 이 때문에 시스템 성능을 향상시키는 데 도움을 줄 수 있다.

다음은 tbJDBC에서 제공하는 LOB 클래스이다.

- com.tmax.tibero.jdbc.TbBlob
- com.tmax.tibero.jdbc.TbClob

8.2. LOB 지시자

8.2.1. LOB 지시자 얻어오기

JDBC 표준에서 제공하는 ResultSet 객체나 CallableStatement 객체의 메소드를 이용하여 LOB 지시자를 얻어올 수 있다.

- ResultSet 객체

```
ResultSet.getBlob()
ResultSet.getClob()
ResultSet.getObject()
```

다음은 ResultSet 객체를 이용하여 LOB 지시자를 얻어오는 예이다.

```
ResultSet rs = stmt.executeQuery("SELECT document, image FROM library");
while (rs.next())
{
    Clob document = rs.getClob(1);
    // 또는 Clob document = (Clob)rs.getObject(1);
```

```
    Blob image = rs.getBlob(2);
    // 또는 Blob image = (Blob)rs.getObject(1);
}
```

- CallableStatement 객체

```
CallableStatement.getBlob()
CallableStatement.getClob()
CallableStatement.getObject()
```

다음은 CallableStatement 객체를 이용하여 LOB 지시자를 얻어오는 예이다.

```
CallableStatement cstmt = conn.prepareCall("{call proc(?)}");
cstmt.registerOutParameter(1, Types.CLOB);
cstmt.execute();

Clob document = cstmt.getClob(1);
// 또는 Clob document = (Clob)cstmt.getObject(1);
```

8.2.2. LOB 지시자 넘겨주기

JDBC 표준에서 제공하는 PreparedStatement 객체나 CallableStatement 객체의 메소드를 이용하여 LOB 지시자를 넘겨줄 수 있다.

- PreparedStatement 객체

```
PreparedStatement.setBlob()
PreparedStatement.setClob()
PreparedStatement.setObject()
```

다음은 PreparedStatement 객체를 이용하여 LOB 지시자를 넘겨주는 예이다.

```
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO library
VALUES (?, ?)");

pstmt.setClob(1, document);
// 또는 pstmt.setObject(1, document, Types.CLOB);

pstmt.setBlob(2, image);
// 또는 pstmt.setObject(1, image, Types.BLOB);

pstmt.executeUpdate();
```

- CallableStatement 객체

```
CallableStatement.setBlob()
CallableStatement.setClob()
CallableStatement.setObject()
```

다음은 CallableStatement 객체를 이용하여 LOB 지시자를 넘겨주는 예이다.

```
CallableStatement cstmt = conn.prepareCall("{call proc(?, ?)}");

cstmt.setClob(1, document);
// 또는 cstmt.setObject(1, document, Types.CLOB);

cstmt.setBlob(2, image);
// 또는 cstmt.setObject(1, image, Types.BLOB);

cstmt.execute();
```

8.3. LOB 데이터 읽고 쓰기

LOB 지시자를 얻어오면 JDBC 표준에서 제공하는 API를 이용하여 데이터를 읽고 쓸 수 있다. LOB 데이터는 보통 Byte 배열이나 스트림 형태로 읽어올 수 있는데, 일반적인 스트림 형태와 달리 데이터베이스 서버 내부에 데이터가 있으므로 연결이 유지되는 한 언제든지 접근할 수 있다.

JDBC 표준에서는 LOB 데이터를 읽고 쓸 수 있도록 다음과 같은 API를 제공한다.

- InputStream Clob.getAsciiStream()
- Reader Clob.getCharacterStream()
- String Clob.getSubString()
- OutputStream Clob.setAsciiStream()
- Writer Clob.setCharacterStream()
- int Clob.setString()
- InputStream Blob.getBinaryStream()
- byte[] Blob.getBytes()
- OutputStream Blob.setBinaryStream()
- int Blob.setBytes()

다음은 LOB 데이터를 읽는 예이다.

```
// BLOB 데이터
InputStream is = image.getBinaryStream();
byte[] imageData = new byte[1000];
```

```
int readLength = is.read(imageData);

// CLOB 데이터
Reader reader = document.getCharacterStream();
char[] docData = new char[1000];
int readLength = reader.read(docData, 0, docData.length);
```

다음은 LOB 데이터를 쓰는 예이다.

```
// BLOB 데이터
byte[] imageData = {32, 53, 78, 19, 2, 93};
OutputStream os = image.setBinaryStream();
os.write(imageData);

// CLOB 데이터
char[] docData = {'J', 'D', 'B', 'C', '안', '내', '서'};
Writer writer = document.setCharacterStream();
writer.write(docData);
writer.flush();
writer.close();
```

위의 예에서 OutputStream이나 Writer 객체는 데이터를 쓸 때마다 데이터베이스에 직접 전송되므로 별도의 UPDATE 문장을 실행할 필요가 없다. 하지만 LOB 자체는 트랜잭션 범위에 포함되므로 반드시 커밋을 수행해야 데이터베이스에 최종적으로 반영된다.

8.4. 임시 LOB

tbJDBC에서는 대용량 데이터를 LOB 형태로 저장하기 위해 임시 LOB을 제공한다. 단, 이 대용량 데이터는 테이블에 저장되지 않는다.

이를 위해 다음과 같은 메소드를 제공한다.

- Connection.createBlob() return Blob
- Connection.createClob() return Clob
- Connection.createNClob() return NClob
- Blob.free()
- Clob.free()
- NClob.free()

임시 LOB을 사용하여 저장한 데이터는 임시 테이블 스페이스에 저장된다. 이 영역에 저장된 데이터가 더는 필요하지 않게 되면, 사용자는 반드시 이것을 해제해야 한다. 그렇지 않으면 이 영역의 데이터는 계속 남아있게 된다.

임시 LOB는 LOB 컬럼의 데이터를 바인딩할 때 주로 사용한다. 예를 들면 다음과 같다.

```
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO
library VALUES (?, ?)");
pstmt.setInt(1, 20090811);

Clob doc = conn.createClob();
doc.setString(1, "JDBC Guide<br>writer:Rachael<br>proofreader:Patrick");
pstmt.setClob(2, doc);
pstmt.executeUpdate();

doc.free();
pstmt.close();
```

위에서 나열한 메소드들은 JDBC 표준 API들로 일부 메소드는 Java 버전에 따라 사용할 수 없는 것들이 존재한다. 이를 위해 tbJDBC에서는 별도로 다음과 같은 구현 메소드들을 제공하고 있다. 이 메소드들을 이용할 경우 다른 JDBC와의 호환성은 떨어지지만 tbJDBC에 한하여 Java 버전에 관계 없이 사용할 수 있는 장점이 있다.

- TbConnection.createTbBlob() return TbBlob
- TbConnection.createTbClob() return TbClob
- TbConnection.createTbNClob() return TbNClob

Tibero 4.0 SP1까지의 tbJDBC에서는 구현 메소드를 다음과 같은 형태로 제공하였으나, 호환성에 문제가 있어 폐기(deprecated)하고 위의 메소드들로 대체되었다.

- static TbBlob.createTemporary(Connection conn) return TbBlob
- static TbClob.createTemporary(Connection conn) return TbClob
- static TbClob.createTemporaryNClob(Connection conn) return TbClob
- TbBlob.freeTemporary()
- static TbBlob.freeTemporary(TbBlob blob)
- TbClob.freeTemporary()
- static TbClob.freeTemporary(TbClob clob)
- TbNClob.freeTemporary()
- static TbNClob.freeTemporary(TbNClob nclob)

8.5. API 목록

8.5.1. 표준 API

LOB와 관련된 표준 API의 정보는 JDBC 표준 문서를 참고한다.

8.5.2. 확장 API

tbJDBC에서는 LOB와 관련된 API로 BLOB, CLOB, NCLOB 메소드를 추가로 제공한다.

CONNECTION API

TbConnection 클래스에서는 다음과 같은 API를 추가로 제공한다.

API	설명
TbBlob createTbBlob()	데이터베이스에 임시 BLOB을 생성한 후에 TbBlob 객체를 반환한다.
TbClob createTbClob()	데이터베이스에 임시 CLOB을 생성한 후에 TbClob 객체를 반환한다.
TbNClob createTbNClob()	데이터베이스에 임시 NCLOB을 생성한 후에 TbNClob 객체를 반환한다.

BLOB API

TbBlob 클래스에서는 다음과 같은 API를 추가로 제공한다.

API	설명
TbBlob createEmptyBlob()	EMPTY_BLOB을 생성하여 반환한다.
void close()	BLOB을 닫는다.
OutputStream getBinaryOutputStream()	setBinaryStream(1L)과 동일하다.
OutputStream getBinaryOutputStream(long)	setBinaryStream(long)과 동일하다.
void open(int)	int 모드로 BLOB을 연다.

CLOB API

TbClob 클래스에서는 다음과 같은 API를 추가로 제공한다.

API	설명
TbClob createEmptyClob()	EMPTY_CLOB을 생성하여 반환한다.
void close()	CLOB을 닫는다.
int getBufferSize()	서버와 원활하게 통신할 수 있도록 32KB의 버퍼 크기를 반환한다.
long getChars(long offset, char[] buffer)	offset 위치부터 CLOB 데이터를 읽어서 buffer에 저장한다.
long getChars(long offset, char[] buffer, long numChars)	offset 위치부터 numChars만큼의 CLOB 데이터를 읽어서 buffer에 저장한다.
long getChars(long offset, char[] buffer, long bufOffset, long numChars)	offset 위치부터 numChars만큼의 CLOB 데이터를 읽어서 buffer의 bufOffset 위치에 저장한다.
void open(int)	int 모드로 CLOB을 연다.
long putChars(long offset, char[] buffer)	buffer 데이터를 읽어서 offset 위치에 저장한다.
long putChars(long offset, char[] buffer, long numChars)	buffer 데이터를 numChars만큼 읽어서 offset 위치에 저장한다.
long putChars(long offset, char[] buffer, long bufOffset, long numChars)	buffer 데이터를 bufOffset 위치부터 numChars만큼 읽어서 offset 위치에 저장한다.

NCLOB API

TbNClob 클래스에서는 다음과 같은 API를 추가로 제공한다.

API	설명
TbNClob createEmptyNClob()	EMPTY_NCLOB을 생성하여 반환한다.
void close()	NCLOB을 닫는다.
int getBufferSize()	서버와 원활하게 통신할 수 있도록 32KB의 버퍼 크기를 반환한다.
long getChars(long offset, char[] buffer)	offset 위치부터 NCLOB 데이터를 읽어서 buffer에 저장한다.
long getChars(long offset, char[] buffer, long numChars)	offset 위치부터 numChars만큼의 NCLOB 데이터를 읽어서 buffer에 저장한다.

API	설명
long getChars(long offset, char[] buffer, long bufOffset, long numChars)	offset 위치부터 numChars만큼의 NCLOB 데이터를 읽어서 buffer의 bufOffset 위치에 저장한다.
void open(int)	int 모드로 NCLOB을 연다.
long putChars(long offset, char[] buffer)	buffer 데이터를 읽어서 offset 위치에 저장한다.
long putChars(long offset, char[] buffer, long numChars)	buffer 데이터를 numChars만큼 읽어서 offset 위치에 저장한다.
long putChars(long offset, char[] buffer, long bufOffset, long numChars)	buffer 데이터를 bufOffset 위치부터 numChars만큼 읽어서 offset 위치에 저장한다.

제9장 Failover와 Load balancing

본 장에서는 tbJDBC에서 제공하는 Failover 기능과 로드 밸런싱(Load balancing) 기능을 설명한다.

9.1. Failover

tbJDBC는 일반 애플리케이션 프로그램 환경이나 TAC 환경에서 최초 연결을 맺을 때 또는 임의의 작업을 수행하는 중에 연결이 끊어졌을 때, 일부 자원들을 자동으로 복원해주는 기능을 제공한다. 이를 **Failover** 라고 한다.

9.1.1. Failover 기능 설정

Failover 기능을 사용하려면 DriverManager.getConnection()의 URL을 description 형태로 작성하고 **FAILOVER**를 설정하여 사용한다.

FAILOVER 항목에 설정할 수 있는 값은 다음과 같다.

설정값	**설명**
NONE	Failover 기능을 비활성화한다. (기본값)
SESSION	연결에 실패했을 때, 다음 노드로 재접속을 시도하여 연결을 복원한다.
CURSOR	이 옵션은 SESSION 복원을 포함하며, 사용중이던 java.sql.ResultSet을 복원한다. 복원 가능한 경우는 다음과 같다. – SELECT 문장을 직접 수행하는 경우 – ResultSet 타입이 TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE인 경우 – ResultSet concurrency 모드가 CONCUR_READ_ONLY인 경우 – 재사용 가능한 파라미터만 사용한 경우(Reader, Stream 등의 휘발성 데이터나 또는 varchar 타입 최대 크기를 초과하는 String 데이터가 존재하면 복구 불가)

참고

이전 버전에서 지원하던 Failover 옵션 OFF는 NONE으로 ON은 SESSION으로 동작한다.

다음은 Failover 기능을 활성화하는 예이다. svr1로 연결이 실패하면 자동으로 svr2로 연결을 시도한다. 단, 모든 서버에 대한 연결 시도가 실패할 때는 에러가 발생한다.

```
Connection conn =
    DriverManager.getConnection("jdbc:tibero:thin:@(description=(failover=session)" +
                                "(address_list="+
                                "(address=(host=svr1)(port=8629))"+
                                "(address=(host=svr2)(port=7629))"+
                                ")(DATABASE_NAME=TACUAC))", "tibero", "tmax");
```

9.1.2. Failover 동작과 관련된 연결 속성

Failover SESSION 이상의 모드인 경우 연결 복원 과정에서 재시도 횟수를 다음의 속성을 통해 지정할 수 있다.

- failover_retry_count

다음의 연결 속성을 이용하여 실제로 장애가 발생하였더라도 클라이언트에서 연결 종료를 감지하지 못해 무한정 대기하는 경우를 회피할 수 있다.

- login_timeout
- read_timeout
- self_keepalive

참고

각 연결 속성별 설명 및 사용 방법은 “제3장 tbJDBC의 사용”의 “3.2. Connection Properties”를 참고한다.

9.2. 로드 밸런싱

tbJDBC는 일반 애플리케이션 프로그램 환경이나 TAC 환경에서 여러 개의 노드로 사용자를 분산시켜 데이터베이스 서버의 효율성을 향상시키기 위한 기능을 제공한다. 이를 **로드 밸런싱(Load balancing)**이라고 한다. 로드 밸런싱 기능을 사용하려면 DriverManager.getConnection()의 URL을 description 형태로 작성하고 **(load_balance=on)**만 추가하면 된다.

다음은 로드 밸런싱 기능을 활성화하는 예로 최초 연결을 맺을 때 내부 로직에 의해 svr1과 svr2 사용자로 분산시켜 연결을 맺는다.

```
Connection conn =
    DriverManager.getConnection("jdbc:tibero:thin:@(description=(load_balance=on)"
+
```

```
                                "(address_list="+
                                "(address=(host=svr1)(port=8629))"+
                                "(address=(host=svr2)(port=7629))"+
                                ")(DATABASE_NAME=TACUAC))", "tibero", "tmax");
```

참고

로드 밸런싱 기능은 Failover 기능과 혼용하여 설정할 수 있다.

제10장 SSL

본 장에서는 tbJDBC에서 제공하는 SSL 기능의 기본 개념과 사용 방법을 설명한다.

10.1. 개요

Tibero에 저장된 데이터는 간혹 주민등록번호나 은행 계좌번호 등과 같은 매우 중요한 개인정보 등을 담고 있다. 이러한 정보는 악의적인 사용자에 의해 누출될 수도 있기 때문에 이를 막기 위해 서버와 클라이언트의 진위를 정확하게 가릴 수 있는 기능이 필요하다.

tbJDBC에서는 **SSL** 기능을 제공하여 위와 같은 문제를 해결할 수 있다. tbJDBC의 SSL 기능은 Java Secure Socket Extension(JSSE) 패키지를 이용하여 구현되었다. 이 기능을 사용하면 tbJDBC와 Tibero 서버 사이의 정보를 안전하게 보호할 수 있다.

10.2. SSL 사용

SSL 기능을 사용하기 위해서는 Tibero 서버와 tbJDBC를 설정해야 한다. 이 두 가지 설정이 끝난 다음에는 별도의 추가 작업 없이 Tibero 서버와 tbJDBC 사이에 주고받는 모든 정보를 보호할 수 있다.

10.2.1. Tibero 서버 설정

Tibero 서버는 다음과 같은 순서로 설정되어야 한다.

1. 인증 파일의 존재 유무를 확인한다.

```
$TB_HOME/client/ssl/certs/certificate.pem
$TB_HOME/client/ssl/private/private.key
```

2. $TB_SID.tip 파일에 다음의 초기화 파라미터를 추가한다.

```
CERTIFICATE_FILE="$TB_HOME/client/ssl/certs/certificate.pem"
PRIVATE_KEY="$TB_HOME/client/ssl/private/private.key"
```

3. Tibero 서버를 다시 기동한다.

10.2.2. tbJDBC 설정

tbJDBC는 다음과 같이 설정되어야 한다.

```
Connection conn =
    DriverManager.getConnection("jdbc:tibero:thin:@(description=(protocol=tcps)" +
                                "(address=(host=svr1)(port=8629))"+
                                "(DATABASE_NAME=dbsvr))", "tibero", "tmax");
```

제11장 사용자 정의 데이터 타입

본 장에서는 tbJDBC에서 사용자 정의 데이터 타입을 처리하는 방법을 설명한다.

11.1. 개요

JDBC 표준에서는 사용자 정의 데이터 타입(User Defined Types) 지원을 위해서 java.sql.Array, java.sql.Struct 인터페이스를 제공하고 있다.

- Array 타입은 동일한 타입의 원소들을 모아서 저장하기 위한 타입이다.
- Struct 타입은 여러 가지 타입의 원소들을 저장하기 위한 타입이다.

11.2. Array 타입

tbJDBC는 java.sql.Array 인터페이스를 구현한 com.tmax.tibero.jdbc.TbArray 클래스를 제공한다.

11.2.1. Array 데이터 타입 선언

사용자는 다음 같이 Array 타입을 정의할 수 있다.

```
create or replace type t_varr      is varray(16) of varchar2(128);

create or replace type t_tbl       is table      of varchar2(128);
```

11.2.2. Array 데이터 타입 IN

Array를 객체를 만들고 PSM 프러시저의 IN 인자로 전달하기 위해서 다음의 과정으로 처리한다.

1. Connection 객체로부터 Array를 생성한다.

   ```
   public Array Connection.createArrayOf(String typeName, Object[] elements)
   throws SQLException;
   ```

2. 프러시저에 Array 객체를 바인딩한다.

   CallabaleStatement의 setObject()를 통해서 Array 객체를 PSM 프러시저에 넘길 수 있다.

다음은 Array를 객체를 만들고 PSM 프러시저의 IN 인자로 전달하는 예제이다.

```
StringBuffer sbProc = new StringBuffer();

sbProc.append("begin ");
sbProc.append("    print_varr(?); ");
sbProc.append("end; ");

Object[] attributes = new Object[16];
for (int i = 0; i < 16; i++) {
      attributes[i] = "abc" + i;
}


Array arr = conn.createArrayOf("TIBERO.T_VARR", attributes);

CallableStatement cstmt = conn.prepareCall(sbProc.toString());

cstmt.setObject(1, arr);
cstmt.execute();

int status = 0;
int i = 0;
while (status == 0) {
      cstmt = conn.prepareCall("{call sys.dbms_output.get_line(?, ?)}");
      cstmt.registerOutParameter(1, java.sql.Types.VARCHAR);
      cstmt.registerOutParameter(2, java.sql.Types.NUMERIC);
      cstmt.execute();

      status = cstmt.getInt(2);
      if (status != 0)
            break;
      String retStr = cstmt.getString(1);
      String ret[] = retStr.split(":");
      System.out.println(ret[1]);
}
```

11.2.3. Array 데이터 타입 OUT

프러시저로부터 OUT 인자 형태로 Array 객체를 받아오기 위해서는 다음의 단계로 처리한다.

1. OUT 바인딩을 통한 프러시저를 호출하면 프러시저는 Array 타입 객체를 돌려준다.

2. CallableStatement.getArray() 함수를 통해 Array 객체를 받아온다.

3. Array 객체로부터 값을 받아온다.

다음은 프러시저로부터 OUT 인자 형태로 Array 객체를 받아오는 예제이다. p_out_t_varr 프러시저는 16개의 문자열 배열을 돌려주도록 작성되어 있다.

```
sbProc.append("begin ");
sbProc.append("    p_out_t_varr(?, ?); ");
sbProc.append("end; ");

CallableStatement cstmt = conn.prepareCall(sbProc.toString());
cstmt.registerOutParameter(1, Types.ARRAY, "TIBERO.T_VARR");
cstmt.setString(2, "abc");
cstmt.execute();

TbArray arr = (TbArray) cstmt.getArray(1);

Object obj = arr.getArray();

String[] bdArr = (String[]) obj;
assertEquals(16, bdArr.length);
for (int i = 0; i < bdArr.length; i++) {
    System.out.println(bdArr[i]);
}
```

11.3. Struct 타입

build-in 타입의 collection을 처리하는 Array와 달리 Struct는 여러 종류의 타입을 묶어서 나타내는 경우 사용한다. tbJDBC는 java.sql.Struct 인터페이스를 구현한 com.tmax.tibero.jdbc.Tb Struct 클래스를 제공한다.

11.3.1. Struct 데이터 타입 선언

다음과 같이 Struct 타입을 선언할 수 있다.

```
create or replace type t_obj is object (obj_id number, name varchar(128), d date);
```

11.3.2. Struct 데이터 타입 IN

Struct 객체를 IN 인자로 전달하기 위해서는 다음의 단계로 처리한다.

1. Connection 객체로부터 Struct를 생성한다.

   ```
   public Struct Connection.createStruct(String typeName, Object[] attributes)
   ```

2. 프러시저에 Struct 객체를 바인딩한다.

CallabaleStatement의 setObject()를 통해서 Struct 객체를 PSM 프러시저에 넘길 수 있다.

다음은 Struct 객체를 IN 인자로 전달하는 예제이다. print_obj 프러시저는 Struct 객체를 인자로 받아서 처리하는 기능을 담당한다.

```
StringBuffer sbProc = new StringBuffer();

sbProc.append("begin ");
sbProc.append("    print_obj(?); ");
sbProc.append("end; ");

Object[] attributes = new Object[3];
attributes[0] = 1;
attributes[1] = "abc1";
Calendar cal = Calendar.getInstance();
cal.set(2012, 11, 21); // 2012. 12. 21
attributes[2] = new Date(cal.getTimeInMillis());

TbStruct struct = conn.createStruct("TIBERO.T_OBJ", attributes);

CallableStatement cstmt = conn.prepareCall(sbProc.toString());
cstmt.setObject(1, struct);
int row = cstmt.executeUpdate();
```

11.3.3. Struct 데이터 타입 OUT

프러시저로부터 Struct 객체를 OUT 인자 형태로 받아오기 위해서는 다음의 단계로 처리하면 된다.

1. OUT 바인딩을 통한 프러시저를 호출한다. 프러시저는 Struct 타입의 객체를 돌려준다.
2. CallableStatement.getStruct() 함수를 통해 Struct 객체를 받아온다.
3. Struct 객체로부터 값을 받아온다.

다음은 프러시저로부터 Struct 객체를 OUT 인자형태로 받아오는 예제이다. p_out_t_obj 프러시저는 NUMBER, VARCHAR, DATE 형태의 object 객체를 돌려준다.

```
StringBuffer sbProc = new StringBuffer();

sbProc.append("begin ");
sbProc.append("    p_out_t_obj(?); ");
sbProc.append("end; ");

CallableStatement cstmt = conn.prepareCall(sbProc.toString());
cstmt.registerOutParameter(1, Types.STRUCT, "TIBERO.T_OBJ");
cstmt.execute();
```

```
Struct struct = (Struct) cstmt.getObject(1);
Object[] structArr = struct.getAttributes();

System.out.println(structArr.length);

System.out.println("number : " + structArr[0]);
System.out.println("varchar : " + structArr[1]);
System.out.println("date : " + structArr[2]);
```

11.4. Array, Struct 타입 테스트 코드

다음은 위의 Array, Struct 타입 생성 및 테스트를 위한 프러시저 코드이다.

```
create or replace type t_obj is object (obj_id number, name varchar(128), d date);
/
create or replace type t_varr is varray(16) of varchar2(128);
/
create or replace type t_varr_varr is varray(8) of t_varr;
/
create or replace type t_tbl is table of varchar2(128);
/
create or replace type t_tbl_tbl is table of t_tbl;
/
create or replace type t_varr_obj is varray(4) of t_obj;
/
create or replace type t_tbl_obj is table of t_obj;
/
create or replace procedure p_out_t_obj(o out t_obj)
is
begin
    o := t_obj(1,'abc',to_date('1900-01-01', 'YYYY-MM-DD'));
end;
/
create or replace procedure p_out_t_varr(o out t_varr, str varchar2)
as
    o_val t_varr;
begin
    o_val := t_varr();

    for i in  1 .. 16 loop
        o_val.extend;
        o_val(i) :=  str || to_char(i);
    end loop;
    o := o_val;
end;
```

```
/
create or replace procedure p_out_t_varr_varr(o out t_varr_varr)
as
  o_val t_varr_varr;
begin
    o_val := t_varr_varr();

    for i in 1 .. 8 loop
        o_val.extend;
        p_out_t_varr(o_val(i),  'elem' || to_char(i));
    end loop;
    o := o_val;
end;
/
create or replace procedure p_out_t_tbl(o out t_tbl, str varchar2)
as
    o_val t_tbl;
begin
    o_val := t_tbl();

    for i in  1 .. 4 loop
        o_val.extend;
        o_val(i) :=  str || to_char(i);
    end loop;
    o := o_val;
end;
/
create or replace procedure p_out_t_tbl_tbl(o out t_tbl_tbl)
as
  o_val t_tbl_tbl;
begin
    o_val := t_tbl_tbl();

    for i in 1 .. 2 loop
        o_val.extend;
        p_out_t_tbl(o_val(i),  'elem' || to_char(i));
    end loop;
    o := o_val;
end;
/
create or replace procedure p_out_t_varr_obj(o out t_varr_obj)
as
  o_val t_varr_obj;
begin
    o_val := t_varr_obj();
    for i in 1 .. 4 loop
        o_val.extend;
```

```
        p_out_t_obj(o_val(i));
    end loop;
    o := o_val;
end;
/
create or replace procedure p_out_t_tbl_obj(o out t_tbl_obj)
as
  o_val t_tbl_obj;
begin
    o_val := t_tbl_obj();
    for i in 1 .. 6 loop
        o_val.extend;
        p_out_t_obj(o_val(i));
    end loop;
    o := o_val;
end;
/
create or replace procedure print_obj(p in t_obj)
as
begin
    dbms_output.put_line('obj_id:' || p.obj_id);
    dbms_output.put_line('name  :' || p.name);
    dbms_output.put_line('d     :' || p.d);
end;
/
create or replace procedure print_varr(p in t_varr)
as
begin
    for i in  1 .. 16 loop
        dbms_output.put_line(to_char(i) || ':' ||  p(i));
    end loop;
end;
/
create or replace procedure print_varr_varr(p in t_varr_varr)
as
begin
    for i in  1 .. 8 loop
        print_varr(p(i));
    end loop;
end;
/
create or replace procedure print_tbl(p in t_tbl)
as
begin
    for i in  1 .. 4 loop
        dbms_output.put_line(to_char(i) || ':' ||  p(i));
    end loop;
```

```
end;
/
create or replace procedure print_tbl_tbl(p in t_tbl_tbl)
as
begin
    for i in  1 .. 2 loop
        print_tbl(p(i));
    end loop;
end;
/
create or replace procedure print_varr_obj(p in t_varr_obj)
as
begin
    for i in 1 .. 4 loop
        print_obj(p(i));
    end loop;
end;
/
create or replace procedure print_tbl_obj(p in t_tbl_obj)
as
begin
    for i in 1 .. 6 loop
        print_obj(p(i));
    end loop;
end;
/
```

색인

I

J

L

N

P

R

S

ㅈ

ㅋ

ㅌ

ㅍ