

## 第一章 Java 的数据库编程（JDBC）术

知识要点：

### 第一节 JDBC 概述

为什么采用 JDBC、JDBC 基本程序结构、JDBC 的缺点、JDBC 的工作原理、JDBC 的结构、数据库应用的模型、通过 JDBC 实现对数据库的访问

### 第二节 创建与数据源的连接

JDBC 驱动程序、连接的一般方法、Access 的连接应用、SQL Server 2000 的连接应用、MySQL 的连接应用、Oracle 的连接应用、DB2 的连接应用、不同种类的数据库驱动程序配置

### 第三节 运用 JDBC 进行数据库操作

查询数据库的一些结构信息、查询数据库中的数据、使用宏语句、使用存储过程、处理大二进制字段

### 第四节 事务处理

为什么要处理事务、无事务的情况、事务处理的实例

### 第五节 游标操作

ResultSet 对象、游标操作的实例分析、更新数据库、插入操作、删除操作、批量操作

## 第一节 JDBC 概述

JDBC (Java DataBase Connectivity, Java 数据库连接技术) 是 Java 访问数据库资源的标准, JDBC 标准定义了一组 Java API, 允许我们写出 SQL 语句, 然后交给数据库。

有了 JDBC 从而可以使 Java 程序员用 Java 语言来编写完整的数据库方面的应用程序。另外也可以操作保存在多种不同的数据库管理系统中的数据, 而与数据库管理系统中数据存储格式无关。同时 Java 语言的与平台的无关性, 不必在不同的系统平台下编写不同的数据库应用程序。

### 一、为什么采用 JDBC

JDBC 可以说是最老的企业 Java 规范之一, 最早的起草日期要追溯到 1996 年, JDBC 与微软开发的开放数据连接 (Open Database Connectivity, ODBC) 标准具有同样的功能, 它提供一组通用的 API, 通过数据库特定的驱动程序来访问数据库。

如果没有 JDBC 或者 ODBC, 开发人员必须使用不同的一组 API 来访问不同的数据库, 而利用 JDBC 或者 ODBC, 则只需要使用一组 API, 再加上数据库厂商提供的数据库驱动程序就可以了。所以, 利用 JDBC, 我们就可以把同一个企业级 Java 应用移植到另一个数据库应用上。

JDBC 设计的目的

1) ODBC: 微软的 ODBC 是用 C 编写的, 而且只适用于 Windows 平台, 无法实现跨平台地操作数据库。

2) SQL 语言: SQL 尽管包含有数据定义、数据操作、数据管理等功能, 但它并不是一个完整的编程语言, 而且不支持流控制, 需要与其它编程语言相配合使用。

3) JDBC 的设计: 由于 Java 语言具有健壮性、安全、易使用并自动下载到网络等方面的优点, 因此如果采用 Java 语言来连接数据库, 将能克服 ODBC 局限于某一系统平台的缺陷; 将 SQL 语言与 Java 语言相互结合起来, 可以实现连接不同数据库系统, **即使用 JDBC 可以很容易**

地把 SQL 语句传送到任何关系型数据库中。

4) JDBC 设计的目的：它是一种规范，设计出它的最主要的目的是让各个数据库开发商为 Java 程序员提供标准的数据库访问类和接口，使得独立于 DBMS 的 Java 应用程序的开发成为可能（数据库改变，驱动程序跟着改变，但应用程序不变）。

JDBC 的主要功能：

- 1) 创建与数据库的连接；
- 2) 发送 SQL 语句到任何关系型数据库中；
- 3) 处理数据并查询结果。

## 二、JDBC 基本程序结构

（此结构有何问题？如何解决？）

Try

（Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")做什么样的工作？）

//(1)加载连接数据库的驱动程序

```
{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

// (2) 创建与数据库的连接

```
Connection
```

```
con=DriverManager.getConnection("jdbc:odbc:DatabaseDSN","Login","Password");
```

//(3).查询数据库: 创建 Statement 对象并执行 SQL 语句以返回一个 ResultSet 对象。发送 SQL 语句到数据库中

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

```
ResultSet rs=stmt.executeQuery("select * from DBTableName");
```

发送 SQL 语句到数据库中

// (5) 处理数据并查询结果。

```
while(rs.next())
```

```
{ String name=rs.getString("Name");
```

```
int age=rs.getInt("age");
```

```
float wage=rs.getFloat("wage");
```

```
}
```

// (6) 关闭

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

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

```
con.close();
```

```
}
```

```
catch(SQLException e)
```

```
{ System.out.println("SQLState:" + e.getSQLState());
```

```
System.out.println("Message:" + e.getMessage());
```

```
System.out.println("Vendor:" + e.getErrorCode());
```

```
}
```

## 三、JDBC 的缺点

（如何克服 JDBC 这样的缺点？）

1) JDBC 并不符合面向对象的要求，JDBC 要求你明确地处理数据字段，并且将它们映射到关系数据库的表中。开发人员被迫与两种区别非常大的数据模型、语言和数据访问手段打交道：

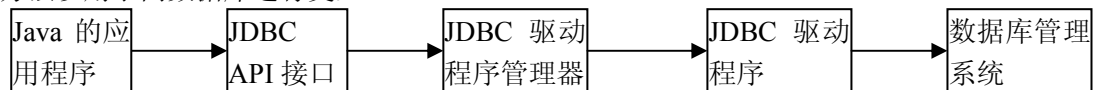
Java，以及 SQL 中的关系数据模型。

2) 在开发中实现从关系数据模型到 Java 对象模型的映射非常复杂，以致于多数开发人员从不为数据定义对象模型，而是简单地编写过程化的 Java 代码来对底层的关系数据库中的数据表进行操纵。

3) 最终结果是：开发人员根本不能从面向对象的开发中得到任何好处。

#### 四、JDBC 的工作原理

JDBC 的设计基于 X/Open SQL CLI（调用级接口）这一模型。它通过定义出一组 API 对象和方法以用于同数据库进行交互。



在 Java 程序中要操作数据库，一般应该通过如下几步（利用 JDBC 访问数据库的编程步骤）：

- 1) 加载连接数据库的驱动程序 `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
- 2) 创建与数据源的连接

```
String url="jdbc:odbc:DatabaseDSN";
```

```
Connection con=DriverManager.getConnection(url,"Login","Password");
```

- 3) 查询数据库：创建 Statement 对象并执行 SQL 语句以返回一个 ResultSet 对象。

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

```
ResultSet rs=stmt.executeQuery("select * from DBTableName");
```

- 4) 获得当前记录集中的某一记录的各个字段的值

```
String name=rs.getString("Name");
```

```
int age=rs.getInt("age");
```

```
float wage=rs.getFloat("wage");
```

- 5) 关闭查询语句及与数据库的连接（注意关闭的顺序先 rs 再 stmt 最后为 con）

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

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

```
con.close();
```

#### 五、JDBC 的结构

JDBC 主要包含两部分：面向 Java 程序员的 JDBC API 及面向数据库厂商的 JDBC Drive API。

##### 1) 面向 Java 程序员的 JDBC API:

（javaAPI 中的各种类之间具体什么关系？每个类的具体描述）

Java 程序员通过调用此 API 从而实现连接数据库、执行 SQL 语句并返回结果集等编程数据

库的能力，它主要是由一系列的接口定义所构成。

**java.sql.DriverManager:** 该接口主要定义了用来处理装载驱动程序并且为创建新的数据库连接提供支持。

**java.sql.Connection:** 该接口主要定义了实现对某一种指定数据库连接的功能。

**java.sql.Statement:** 该接口主要定义了一个在给定的连接中作为 SQL 语句执行声明的容器以实现对数据库的操作。它主要包含有如下的两种子类型。

**java.sql.PreparedStatement:** 该接口主要定义了用于执行带或不带 IN 参数的预编译 SQL 语句。

**java.sql.CallableStatement:** 该接口主要定义了用于执行数据库的存储过程的调用。

**java.sql.ResultSet:** 该接口主要定义了用于执行对数据库的操作所返回的结果集。

## 2) 面向数据库厂商的 JDBC Drive API:

**(现在的数据库厂商除了提供 API 接口之外，有一些厂商还提供了 DBMS 端的缓冲)**

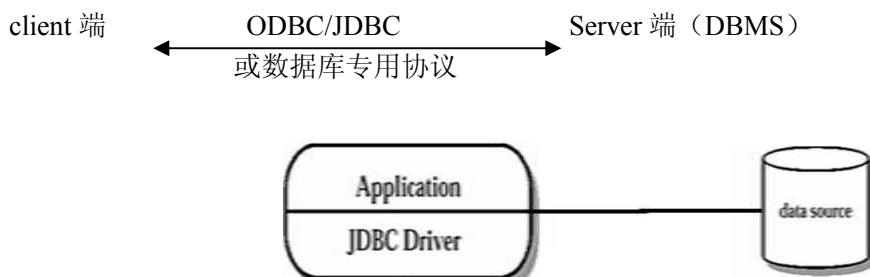
数据库厂商必须提供相应的驱动程序并实现 JDBC API 所要求的基本接口（每个数据库系统厂商必须提供对 DriverManager、Connection、Statement、ResultSet 等接口的具体实现），从而最终保证 Java 程序员通过 JDBC 实现对不同的数据库操作。

## 六、数据库应用的模型

### 1) 两层结构 (C/S):

**(两层结构具体不好使用在哪里？具体的代码展示)**

在此模型下，客户端的程序直接与数据库服务器相连接并发送 SQL 语句（但这时就需要在客户端安装被访问的数据库的 JDBC 驱动程序），DBMS 服务器向客户返回相应的结果，客户程序负责对数据的格式化。

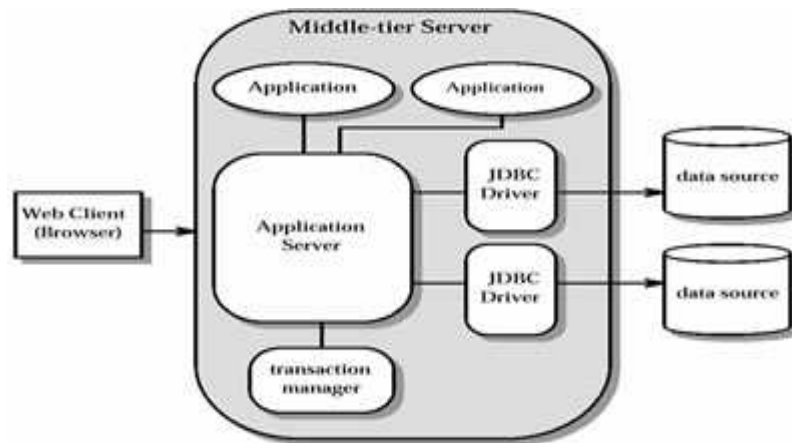


主要的缺点：受数据库厂商的限制，用户更换数据库时需要改写客户程序；受数据库版本的限制，数据库厂商一旦升级数据库，使用该数据库的客户程序需要重新编译和发布；对数据库的操作与处理都是在客户程序中实现，使客户程序在编程与设计时较为复杂。

### 2) 三（或多）层结构 (B/S):

**(三层结构的好处在哪里？JDBC 在三层结构中所处的位置和所起的作用，以及现在对 JDBC 的扩充和封装)**

在此模型下，主要在客户端的程序与数据库服务器之间增加了一个中间服务器（可以采用 C++ 或 Java 语言来编程实现），隔离客户端的程序与数据库服务器。客户端的程序（可以为通用的浏览器）与中间服务器进行通信，然后由中间服务器处理客户端程序的请求并管理与数据库服务器的连接。



## 七、通过 JDBC 实现对数据库的访问

### 1) 引用必要的包

```
import java.sql.*; //它包含有操作数据库的各个类与接口
```

### 2) 加载连接数据库的驱动程序类

为实现与特定的数据库相连接，JDBC 必须加载相应的驱动程序类。这通常可以采用 `Class.forName()` 方法显式地加载一个驱动程序类，由驱动程序负责向 `DriverManager` 登记注册并在与数据库相连接时，`DriverManager` 将使用此驱动程序。

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

注意：

1. 条语句直接加载了 sun 公司提供的 JDBC-ODBC Bridge 驱动程序类。
2. ODBC-JDBC 桥方式，可以采用 ODBC 数据源名称来指定数据库文件  
`String url="jdbc:odbc:studlist";`  
也可以直接采用指定数据库文件的物理位置（可以绝对路径,也可以是相对路径）  
`String url = "jdbc:odbc:driver={Microsoft Access Driver (*.mdb)};DBQ=d:\\stud.mdb";`

## 第二节 创建与数据源的连接

### 一、JDBC 驱动程序

一般来说，驱动程序是由数据库厂商提供的，目前 JDBC 驱动程序共有四种类型：

#### 第一种 JDBC-ODBC 桥加 ODBC 驱动程序

这是 Sun 公司最早实现的 JDBC 驱动程序，当时主要目的在于快速推广 JDBC，以便业界接纳这个标准。实际上这种驱动程序就是把 JDBC API 映射到 ODBC API 上。JDBC-ODBC 桥接方式利用微软的开放数据库互连接口(ODBC API)同数据库服务器通讯，客户端计算机首先应该安装并配置 ODBC driver 和 JDBC-ODBC bridge 两种驱动程序。

这种桥接方式最大的问题，是把 Java 不由自主的和 Windows 绑定在了一起，失去了 Java

跨平台的特点,这对于 Java 应用程序来说是不可接受的。另外 Sun 所提供的 ODBC-JDBC 桥不是多线程的,也就是说它不适合在需要并发执行的企业级应用中使用。所以就目前的使用上来看,这种形式的驱动程序已经很少使用了。

## 第二种 本地 API

这种类型的驱动程序把客户机 API 上的 JDBC 调用转换为 Oracle、Sybase、Informix、DB2 或其它 DBMS 的调用。注意,象桥驱动程序一样,这种类型的驱动程序要求将某些二进制代码加载到每台客户机上。

这种驱动方式将数据库厂商的特殊协议转换成 Java 代码及二进制类码,使 Java 数据库客户方与数据库服务器方通信。例如:Oracle 用 SQLNet 协议,DB2 用 IBM 的数据库协议。数据库厂商的特殊协议也应该被安装在客户机上。

## 第三种 JDBC 网络纯 Java 驱动程序

在 java 的早期阶段,Applet 非常流行的时候,需要用 Applet 直接访问数据库,然而 Applet 安全模式禁止它访问多个 Web 服务器上的数据库资源,为了解决这个问题,这个驱动程序就像 Applet 访问数据库的代理,目前这种形式的驱动程序已经很少用。

这种驱动程序将 JDBC 转换为与 DBMS 无关的网络协议,之后这种协议又被某个服务器转换为一种 DBMS 协议。这种网络服务器中间件能够将它的纯 Java 客户机连接到多种

不同的数据库上。所用的具体协议取决于提供者。通常,这是最为灵活的 JDBC 驱动程序。有可能所有这种解决方案的提供者都提供适合于 Intranet 用的产品。为了使这些产品也支持 Internet 访问,它们必须处理 Web 所提出的安全性、通过防火墙的访问等方面的额外要求。几家提供者正将 JDBC 驱动程序加到他们现有的数据库中间件产品中。

这种方式是纯 Java driver。数据库客户以标准网络协议(如 HTTP、SHTTP)同数据库访问服务器通信,数据库访问服务器然后翻译标准网络协议成为数据库厂商的专有特殊数据库访问协议(也可能用到 ODBC driver)与数据库通信。对 Internet 和 Intranet 用户而言这是一个理想的解决方案。Java driver 被自动的,以透明的方式随 Applets 自 Web 服务器而下载并安装在用户的计算机上。

## 第四种 本地协议纯 Java 驱动程序

这种类型的驱动程序将 JDBC 调用直接转换为 DBMS 所使用的网络协议。这将允许从客户机机器上直接调用 DBMS 服务器,是 Intranet 访问的一个很实用的解决方法。

这种方式也是纯 Java driver。数据库厂商提供了特殊的 JDBC 协议使 Java 数据库客户与数据库服务器通信。然而,将把代理协议同数据库服务器通信改用数据库厂商的特殊 JDBC driver。这对 Intranet 应用是高效的,可是数据库厂商的协议可能不被防火墙支持,缺乏防火墙支持在 Internet 应用中会存在潜在的安全隐患。

驱动程序的选择往往是比较难以定夺的事情,一般来说,第二种驱动程序具有多年使用的优势,而第四种驱动程序由于避开了本地代码存在的隐患,相对来说可能比较好,而且比较容易移植。

目前而言,Sybase、Informix、Sql Server 都可以使用第四种驱动程序,而 Oracle 可以使用第二种驱动程序。现在 Oracle 公司也免费提供第四种类型的驱动程序,可以访问下面的网址:



[http://technet.oracle.com/software/tech/java/sqlj\\_jdbc/software\\_index.html](http://technet.oracle.com/software/tech/java/sqlj_jdbc/software_index.html)

驱动程序的安装应该仔细阅读数据库厂商提供的资料，下面我们以最常用的几种数据库为例，讨论数据库驱动程序的配置问题。

## 二、连接的一般方法

```
String url="jdbc:odbc:DatabaseDSN";  
Connection con=DriverManager.getConnection(url,"Login","Password");
```

注意：

采用 **DriverManager** 类中的 **getConnection()** 方法实现与 **url** 所指定的数据源建立连接并返回一个 **Connection** 类的对象，以后对这个数据源的操作都是基于该 **Connection** 类对象；但对于 **Access** 等小型数据库，可以不用给出用户名与密码。

```
String url="jdbc:odbc:DatabaseDSN";  
Connection con=DriverManager.getConnection(url);  
System.out.println(con.getCatalog()); //取得数据库的完整路径及文件名
```

JDBC 借用了 **url** 语法来确定全球的数据库（数据库 URL 类似于通用的 URL），对由 **url** 所指定的数据源的表示格式为

**jdbc:<subprotocol>:[ database locator]**

**jdbc---**指出要使用 JDBC

**subprotocol---**定义驱动程序类型

**database locator---**提供网络数据库的位置和端口号(包括主机名、端口和数据库系统名等)

**jdbc:odbc://host.domain.com:port/databasefile**

主协议 **jdbc**，驱动程序类型为 **odbc**，它指明 JDBC 管理器如何访问数据库，该例指定为采用 JDBC-ODBC 桥接方式；其它为数据库的位置表示。

不同厂商的数据库系统，主要差别在 JDBC 的驱动程序及数据库的 **url** 格式。

下面讨论一些具体的问题。

## 三、Access 的连接应用

**Access** 数据库是微软 Office 自带的数据库，在安全性、方便性、用户界面方面有良好的表现，但它只适合小量用户使用。

由于它本身就必须在 Windows 下使用，这里正好用它说明 ODBC-JDBC 桥的应用。

数据源名：**FaqAccess**

测试代码：

```
import java.sql.*;

public class AccessDemo {
    public static void main(String[] args) throws Exception {

        //FaqAccess 指在 ODBC 上声明的数据库名字
        String url = "jdbc:odbc:FaqAccess";
        String query, subject, answer;

        Connection conn;           //建立连接类
        Statement statement;        //建立 Sql 语句执行类
        ResultSet resultSet = null; //建立结果集类

        //告诉程序使用 jdbc 与 odbc 桥创建数据库联接
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        //使用 DriverManager 类的 getConnection()方法建立联接,第一个字符参数定义
        //用户名,
        //第二个字符参数定义密码
        conn = DriverManager.getConnection(url, "", "");

        statement = conn.createStatement(); //创建 sql 语句执行类

        //获取数据的记录数
        query = "select count(*) as rowCount from faqs";
        //使用 executeQuery()方法执行 sql 命令
        resultSet = statement.executeQuery(query);

        resultSet.next(); //移到数据库的下一条记录

        //如果记录为空,那么加入十条记录
        if (resultSet.getInt("rowCount")==0){
            for(int i = 1; i < 11; ++i){
                String sql="insert into faqs values(" + i + ", '问题" + i +
                    "' , '答案" + i + "')";
                //使用 executeUpdate()方法执行除查询之外的 sql 命令
                statement.executeUpdate(sql);
            }
        }

        //获得数据库的所有记录
        query = "select * from faqs";
```



```

resultSet = statement.executeQuery(query);

//使用 next()方法游历数据库的每条记录
while (resultSet.next()) {
    //使用 getString()方法取得字段的内容
    subject = resultSet.getString("subject");
    answer = resultSet.getString("answer");
    System.out.print("问题内容 = " + subject);
    System.out.println(", 客案内容 = " + answer);
}

resultSet.close();    //关闭结果集
statement.close();    //关闭 sql 语句执行类
conn.close();         //关闭数据库联接类
}
}

```



#### 四、SQL Server 2000 的连接应用

微软的 SQL Server 在中小型数据库项目上应用非常普遍，它的连接速度快，支持的用户多，安全性好，而且是可视化界面，所以在国内许多 MIS 系统上应用非常广泛。

微软的驱动程序有三个程序：

```

msbase.jar
msutil.jar
mssqlserver.jar

```

我们可以在 C 盘根目录下建立一个文件夹：JDBCSQLSERVER2000。把这三个文件拷贝过去。

在工作目录下构造文件 setenv.cmd，

```

set path=%path%;c:\j2sdk1.4.0\bin;
set CLASSPATH=.% CLASSPATH %;c:\j2sdk1.4.0\lib
set
CLASSPATH=%CLASSPATH%.;c:\JDBCSQLSERVER2000\msbase.jar;c:\JDBCSQLSERVER2000

```

\msutil.jar;c:\JDBC\SQLSERVER2000\mssqlserver.jar

以设置 CLASSPATH 路径。

启动在运行前需要启动一下这个文件，设上路径。

注意：在 UNIX 操作系统中，路径设置为：

CLASSPATH=.;/home/user1/mssqlserver2000jdbc/lib/msbase.jar;/home/user1/mssqlserver2000jdbc/lib/msutil.jar;/home/user1/mssqlserver2000jdbc/lib/mssqlserver.jar

可以做一个小的例子，实验数据库调用情况：

文件名：SqlDemo.java

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

public class SqlDemo {

    public static void main(String[] args) throws Exception
    {

        //声明连接，SQL 语句执行对象和结果集变量
        java.sql.Connection conn        = null;
        java.sql.Statement stmt          = null;
        java.sql.ResultSet rs            = null;

        //加载数据库驱动程序
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        //创建连接
        //url=jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433
        //Properties
        //Password=
        //DatabaseName=pubs
        //User=sa
        //计算机名要根据实际情况更改
        conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=pubs;User=sa;Password="
);

        //创建 SQL 语句对象
        stmt = conn.createStatement();
        //执行 SQL 语句
        stmt.execute("select * from employee");
        //取得结果集
        rs = stmt.getResultSet();
```

```
//打印结果
while (rs.next()) {
    System.out.println(rs.getString("fname") + " - " +
                        rs.getString("lname") + " - " +
                        rs.getString("job_id"));
}

rs.close();    //关闭结果集
stmt.close();  //关闭 sql 语句执行类
conn.close();  //关闭数据库连接类
}
}
```

编译, 运行, 应该能够正常工作。

由于 SQL Server 比较普遍、比较小也比较标准, 后面大部分例子我们都将使用 SQL Server 来完成。

## 五、MySQL 的连接应用

MySQL 数据库是应用于网络的数据库, 在网络方面表现非常优越, 同时它是开放式技术标准的产品, 使用不需要付费, 因此得到了一大批网络用户的支持。它的运行速度快, 支持的用户多, 但管理上使用不太友好的 DOS 界面, 因此不如 SQL Server 方便。

```
import java.sql.*;

public class MysqlJDBCDemo {
    public static void main(String[] args) throws Exception {

        //定义 Mysql 的数据库连接类
        String mySqlDriver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost:3306/faq?user=root;password=";

        String query, subject, answer;

        Connection conn;           //建立连接类
        Statement statement;        //建立 Sql 语句执行类
        ResultSet resultSet = null; //建立结果集类

        //告诉程序使用 Mysql 的 jdbc 桥创建数据库连接
        Class.forName(mySqlDriver);

        //使用 DriverManager 类的 getConnection()方法建立联接,
        //第一个字符参数定义用户名,第二个字符参数定义密码
        conn = DriverManager.getConnection(url, "", "");

        statement = conn.createStatement(); //创建 sql 语句执行类
    }
}
```

```
//获得数据库的所有记录
query = "select * from faqs";
resultSet = statement.executeQuery(query);

//使用 next()方法游历数据库的每条记录
while (resultSet.next()) {
    //使用 getString()方法取得字段的内容
    subject = resultSet.getString("subject");
    answer = resultSet.getString("answer");
    System.out.print("问题内容 = " + subject);
    System.out.println(", 客案内容 = " + answer);
}

resultSet.close();    //关闭结果集
statement.close();    //关闭 sql 语句执行类
conn.close();         //关闭数据库联接类
}
}
```

## 六、Oracle 的连接应用

Oracle 是甲骨文公司的产品，是全球商业应用最广泛的数据库，它在安全、速度、数据挖掘与分析方面都领先于对手，它的口号是“坚不可摧”。但 Oracle 数据库操作复杂、成本高。不过，大多数 Java 产品后台数据库都是使用 Oracle。

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

```
public class OracleJDBCDemo {
    public static void main(String[] args) throws Exception {

        //ORCL.WORLD 指在 net8 的网络名
        String url = "jdbc:oracle:thin:bemyfriend:1521:ORCL.WORLD";
        String query, subject, answer;

        Connection conn;           //建立连接类
        Statement statement;        //建立 Sql 语句执行类
        ResultSet resultSet = null; //建立结果集类

        //告诉程序使用 Oracle 的 jdbc 桥创建数据库联接
        Class.forName ("oracle.jdbc.driver.OracleDriver");

        //使用 DriverManager 类的 getConnection()方法建立联接,第一个字符参数定义用户名,
        //第二个字符参数定义密码
    }
}
```

```
conn = DriverManager.getConnection(url, "user1", "pass1");

statement = conn.createStatement();    //创建 sql 语句执行类

//删除表内的所有记录
query = "delete from faqs";
statement.executeUpdate(query);
//向 faqs 数据表内插入十条记录
for(int i = 1; i < 11; ++i){
    String sql = "insert into faqs values(" + i + ", 'question" + i +
        "', 'answer" + i + ")";
    //使用 executeUpdate()方法执行除查询之外的 sql 命令
    statement.executeUpdate(sql);
}

//获得数据库的所有记录
query = "select * from faqs";
resultSet = statement.executeQuery(query);

//使用 next()方法游历数据库的每条记录
while (resultSet.next()) {
    //使用 getString()方法取得字段的内容
    subject = resultSet.getString("subject");
    answer = resultSet.getString("answer");
    System.out.print("问题内容 = " + subject);
    System.out.println(", 客案内容 = " + answer);
}
resultSet.close();    //关闭结果集
statement.close();    //关闭 sql 语句执行类
conn.close();        //关闭数据库联接类
}
}
```

## 七、DB2 的连接应用

IBM 公司的 DB2 是著名的海量数据库，它的性能非常优越，而且 Java 产品和 DB2 的关系也非常友好。

**DB2 UDB V8** 是 **IBM** 公司推出的新一代功能强大的关系型数据库管理系统，下面将讨论在 java 环境中的连接方法和应用的一些问题。

首先必须说明，当客户端需要使用 DB2 的时候，必须在客户端安装 DB2 的客户程序，在服务器端，安装的同时实际上已经安装了客户程序。

### 1) 设置驱动程序路径

在 C:\Program Files\IBM\SQLLIB\java 下面找到

db2java.zip （这是驱动程序文件）

sqlj.zip

Common.jar

db2fs.jar

db2jcc.jar

把它们拷贝到 C:\DB2DRIVER 目录下。其中第一个文件是最重要的，其它的是关于错误提示等的文件，有的时候会非常有用。

在普通应用的时候，也可以直接设置 Classpath。

但在现在的需求下在工作目录下找 setenv.cmd，Sql Server 设置以后的一行，写上：

set

CLASSPATH=%CLASSPATH%.;C:\DB2DRIVER\db2java.zip;C:\DB2DRIVER\Common.jar;C:\DB2DRIVER\db2fs.jar;C:\DB2DRIVER\db2jcc.jar;

启动的时候注意一下路径是不是设上去了。

## 2) 实验:

连接的方法是:

加载数据库驱动程序，注意大小写敏感:

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");  
conn = DriverManager.getConnection("jdbc:db2:数据库名","用户名","密码");
```

源代码: SqlDB2Demo.java

```
import java.*;  
import java.lang.*;  
import java.sql.*;  
  
public class SqlDB2Demo {  
  
    public static void main(String[] args) throws Exception  
    {  
  
        //声明连接，SQL 语句执行对象和结果集变量  
        java.sql.Connection conn          = null;  
        java.sql.PreparedStatement pstmt=null;  
  
        try  
        {
```

```
//加载数据库驱动程序

Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
conn = DriverManager.getConnection("jdbc:db2:BANK","db2admin","db2admin");

    //创建 SQL 语句对象
    pstmt=conn.prepareStatement("INSERT INTO administrator.ABC (NO,NAME) values (?,?)");

    pstmt.setInt(1,3);
    pstmt.setString(2,"张三");
    int opNum=pstmt.executeUpdate();
    pstmt.setInt(1,4);
    pstmt.setString(2,"李四");
    opNum=pstmt.executeUpdate();
}
catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    if (pstmt != null)
    {
        try{pstmt.close();}catch(Exception ignore){};
    }
    if (conn != null)
    {
        try{conn.close();}catch(Exception ignore){};
    }
}
}
```

注意一个问题，表的名字前面，应该加上表模式名：ADMINISTRATOR。这是当初构建表的时候选择的，当然当初构建表的时候也可以自己取名。

试验的结果，应该能送入数据。请注意，如果把 NO 字段设为主键，则数据不能重复，第二次运行会报错，这是正常的。

## 八、不同种类的数据库驱动程序配置

Java 在处理数据库的时候，正确的配置驱动程序是一个关键技术，由于现在流行的数据库比较多，同一种型号的数据库由于版本不同驱动程序的配置就会不完全一样，这是需要反复试



验积累经验的事情，下面在列出一些数据库驱动程序装载方式。

例如：装载 **mySQL JDBC** 驱动程序

```
Class.forName("com.mysql.jdbc.Driver");
jdbc:mysql://localhost/Jive?useUnicode=true&characterEncoding=gb2312
String url="
jdbc:mysql://localhost/testDB ? ?useUnicode=true&characterEncoding=gb2312"
//testDB 为你的数据库名
Connection conn= DriverManager.getConnection(url, "username", "password");
```

例如：装载 **Oracle JDBC OCI** 驱动程序（用 thin 模式）

```
Class.forName("oracle.jdbc.driver.OracleDriver ");
String url="jdbc:oracle:thin:@localhost:1521:orcl";
//orcl 为你的数据库的 SID
String user="scott";
String password="tiger";
Connection conn= DriverManager.getConnection(url,user,password);
注意：也可以通过 con.setCatalog("MyDatabase")来加载数据库。
```

例如：装载 **DB2** 驱动程序

```
Class.forName("COM.ibm.db2.jdbc.net.DB2Driver ")
String url="jdbc:db2://localhost:5000/sample";
//sample 为你的数据库名
String user="admin";
String password="";
Connection conn= DriverManager.getConnection(url,user,password);
```

例如：装载 **Microsoft SQLServer** 驱动程序

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
String url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=pubs";
//pubs 为你的数据库的
String user="sa";
String password="";
Connection conn= DriverManager.getConnection(url,user,password);
```

例如：装载 **Informix** 驱动程序

```
Class.forName("com.informix.jdbc.IfxDriver").newInstance();
String url="jdbc:informix-sqli://123.45.67.89:1533/testDB:INFORMIXSERVER=myserver;
user=testuser;password=testpassword"; //testDB 为数据库名
Connection conn= DriverManager.getConnection(url);
```

例如：装载 **Sybase** 驱动程序

```
Class.forName("com.sybase.jdbc.SybDriver").newInstance();
String url =" jdbc:sybase:Tds:localhost:5007/tsdata";    //tsdata 为数据库名
Properties sysProps = System.getProperties();
SysProps.put("user","userid");
SysProps.put("password","user_password");
Connection conn= DriverManager.getConnection(url, SysProps);
```

例如：装载 **PostgreSQL** 驱动程序

```
Class.forName("org.postgresql.Driver");
String url =" jdbc:postgresql:// localhost:5432/tsdata";    //tsdata 为数据库名
String user="userName";
String password="pswd";
Connection conn= DriverManager.getConnection(url,user,password);
```

### 第三节 运用 JDBC 进行数据库操作

#### 一、查询数据库的一些结构信息

这主要是获得数据库中的各个表，各个列及数据类型和存储过程等各方面的信息。根据这些信息，从而可以访问一个未知结构的数据库。这主要是通过 `DatabaseMetaData` 类的对象来实现并调用其中的方法来获得数据库的详细信息（即数据库的基本信息，数据库中的各个表的情况，表中的各个列的信息及索引方面的信息）。

```
DatabaseMetaData dbms=con.getMetaData();
System.out.println("数据库的驱动程序为 "+dbms.getDriverName());
```

#### 二、查询数据库中的数据

**（这些 statement 之间具体的区别是什么？在实际的开发中各用于什么样的情况？）**

在 JDBC 中查询数据库中的数据的执行方法可以分为三种类型，也就是三个接口：

**Statement**：用于执行不带参数的简单 SQL 语句字符串；

**PreparedStatement**：预编译 SQL 语句；

**CallableStatement**：主要用于执行存储过程）。

#### 1) 实现对数据库的一般查询 Statement

实现方法：

1、创建 `Statement` 对象（要想执行一个 SQL 查询语句，必须首先创建出 `Statement` 对象，它封装代表要执行的 SQL 语句）并执行 SQL 语句以返回一个 `ResultSet` 对象，这可以通过 `Connection` 类中的 `createStatement()` 方法来实现。

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

2、执行一个 SQL 查询语句，以查询数据库中的数据。`Statement` 接口提供了三种执行 SQL 语句的方法：`executeQuery()`、`executeUpdate()` 和 `execute()`。具体使用哪

一个方法由 SQL 语句本身来决定。

方法 `executeQuery` 用于产生单个结果集的语句，例如 `SELECT` 语句等。

方法 `executeUpdate` 用于执行 `INSERT`、`UPDATE` 或 `DELETE` 语句以及 SQL DDL（数据定义语言）语句，例如 `CREATE TABLE` 和 `DROP TABLE`。`INSERT`、`UPDATE` 或 `DELETE` 语句的效果是修改表中零行或多行中的一列或多列。`executeUpdate` 的返回值是一个整数，指示受影响的行数（即更新计数）。对于 `CREATE TABLE` 或 `DROP TABLE` 等不操作行的语句，`executeUpdate` 的返回值总为零。

方法 `execute` 用于执行返回多个结果集、多个更新计数或二者组合的语句。一般不会需要该高级功能。

下面给出通过 `Statement` 类中的 `executeQuery()` 方法来实现的代码段：

`executeQuery()` 方法的输入参数是一个标准的 SQL 查询语句，其返回值是一个 `ResultSet` 类的对象。

```
ResultSet rs=stmt. executeQuery ("select * from DBTableName");
```

要点：

①JDBC 在编译时并不对将要执行的 SQL 查询语句作任何检查，只是将其作为一个 `String` 类对象，直到驱动程序执行 SQL 查询语句时才知道其是否正确。对于错误的 SQL 查询语句，在执行时将会产生 `SQLException`。

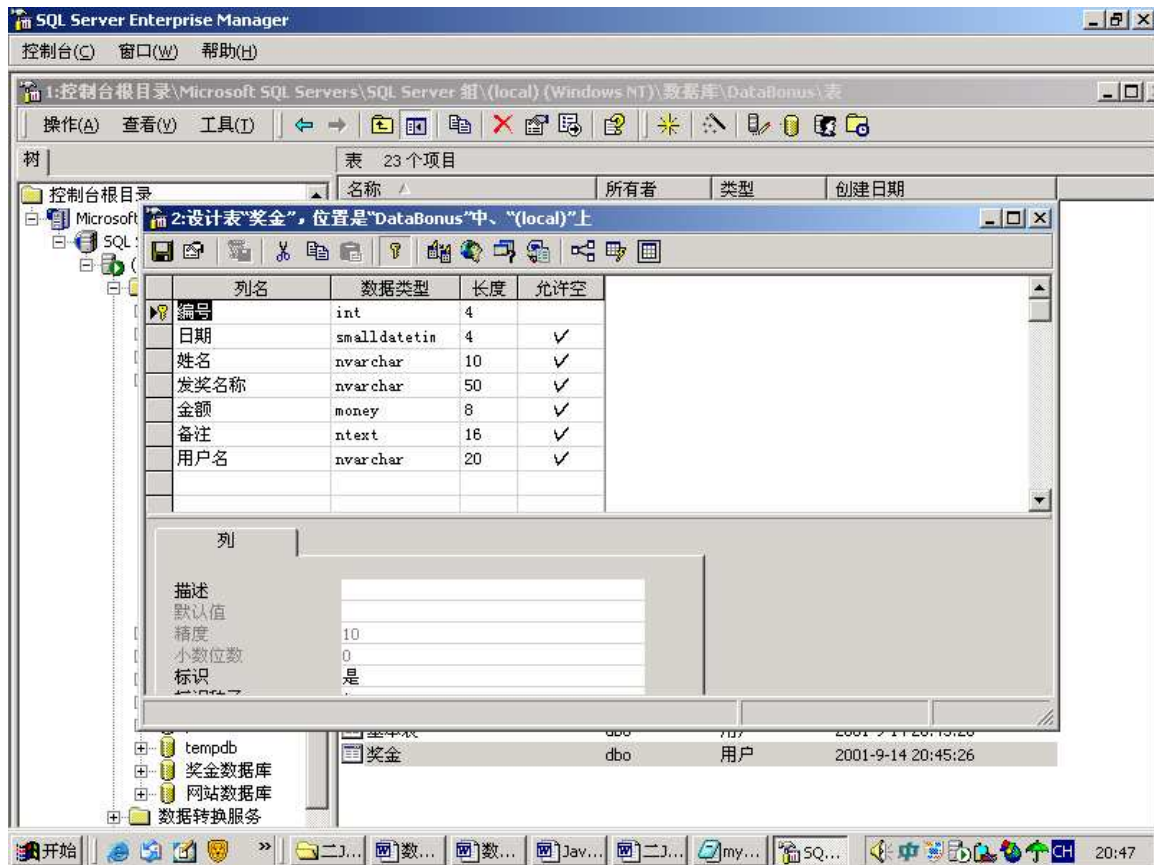
②一个 `Statement` 对象在同一时间只能打开一个结果集，对第二个结果集的打开隐含着对第一个结果集的关闭。（在 **JDBC3.0 规范中已经可以支持多个结果集**）

③如果想对多个结果集同时操作，必须创建出多个 `Statement` 对象，在每个 `Statement` 对象上执行 SQL 查询语句以获得相应的结果集。

④如果不需要同时处理多个结果集，则可以在一个 `Statement` 对象上顺序执行多个 SQL 查询语句，对获得的结果集进行顺序操作。

## 2) 查询操作的示例

数据库 `DataBonus` 中的表：奖金。



样例一：

package Datademo;

import java.io.\*;

import java.sql.\*;

```
public class mysqlserver {  
    public static void main(String[] args) {
```

```
        java.sql.Connection conn= null;
```

```
        java.sql.Statement stmt =null;
```

```
        java.sql.ResultSet rs=null;
```

```
        try
```

```
        {
```

```
            //加载数据库驱动程序
```

```
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

```
            //创建连接
```

```
            //计算机名要根据实际情况更改
```

```
            conn = DriverManager.getConnection
```

```
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Password=");
```

```
        //创建 SQL 语句对象
        stmt = conn.createStatement();
        //执行 SQL 语句
        stmt.execute("select * from 奖金");
        //取得结果集
        rs = stmt.getResultSet();
        //打印结果
        while (rs.next()) {
            System.out.println
(rs.getString("姓名")+"      "+rs.getString("发奖名称")+"      "+rs.getString("金额"));
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        if (rs != null)
        {
            try{rs.close();}catch(Exception ignore){};
        }
        if (stmt != null)
        {
            try{stmt.close();}catch(Exception ignore){};
        }
        if (conn != null)
        {
            try{conn.close();}catch(Exception ignore){};
        }
    }
}
}
```

样例二:

**(在何处断掉? 如何断掉? 如果不断掉会有何种后果?)**

数据库的操作应该注意, 任何操作完成后都要把连接断掉, 以减少数据库资源的消耗。

```
package Datademo;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

import java.*;
import java.lang.*;
import java.sql.*;
import java.util.*;

public class TableDemo extends JFrame {

    //构造一个容器对象
    JPanel contentPane = new JPanel();

    //定义一个表格和一个滚动条
    JTable table;
    JScrollPane jScrollPane1 = new JScrollPane();

    JTextField jTextField1 = new JTextField();
    JButton jButton1 = new JButton();
    JButton jButton2 = new JButton();
    JButton jButton3 = new JButton();

    //接收数据的数组
    Object[][] data;
    //列标题数组
    String[] columnNames = {"编号","姓名","发奖名称","金额","备注"};
    //行号
    int selectedRow;

    //构造方法
    public TableDemo() {
        try {jInit();} catch (Exception e) {}
    }

    //开始代码
    public static void main(String[] args) {
        TableDemo frame = new TableDemo();
        frame.setVisible(true);
        frame.seeData("Select * from 奖金");
    }

    //获取数据库的连接
```

```
java.sql.Connection MyConn()
{
    try
    {
        //加载数据库驱动程序
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

        //创建连接
        //计算机名要根据实际情况更改
        java.sql.Connection conn1 = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Passw
ord=");
        return conn1;
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return null;
    }
}
```

//提交方法

```
public void MUpdata(String Sql)
{

    //声明连接，SQL 语句执行对象和结果集变量
    java.sql.Connection conn        = null;
    java.sql.Statement stmt          = null;
    try
    {
        conn=MyConn();
        //创建 SQL 语句对象
        stmt = conn.createStatement();
        stmt.executeUpdate(Sql);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        if (stmt != null)
        {
```



```
        try{stmt.close();} catch(Exception ignore){};
    }
    if (conn != null)
    {
        try{conn.close();} catch(Exception ignore){};
    }
}
}
```

//查询方法

public void SeeData(String Sql)

{

//声明连接，SQL 语句执行对象和结果集变量

java.sql.Connection conn = null;

java.sql.Statement stmt = null;

java.sql.ResultSet rs = null;

try

{

conn=MyConn();

//创建 SQL 语句对象

stmt = conn.createStatement();

//执行 SQL 语句

stmt.execute(Sql);

//取得结果集

rs = stmt.getResultSet();

Collection c1=new ArrayList();

while (rs.next()) {

Object[] d =new Object[5];

d[0]= rs.getString("编号");

d[1]= rs.getString("姓名");

d[2]= rs.getString("发奖名称");

d[3]= rs.getString("金额");

d[4]= rs.getString("备注");

c1.add(d);

}

//打印结果

data=new Object[c1.size()][5];

c1.toArray(data);

table = new JTable(data, columnNames);

table.setRowHeight(20); //设置表格的行高度

//把表格加入滚动条对象

jScrollPane1.getViewport().add(table);

```
//定义表格的行对象
ListSelectionModel rowSM = table.getSelectionModel();
//加入行选择接收器
rowSM.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
        //当多种事件被激发的时候,不执行行接收器后面的代码
        if (e.getValueIsAdjusting()) return;
        ListRow(e);
    }
});

//当表格的内容改变的时候取得表格的内容
final TableModel model = table.getModel();
model.addTableModelListener(new TableModelListener() {
    public void tableChanged(TableModelEvent e) {

        //表格的行与列从 0 开始
        int row = e.getFirstRow();
        int column = e.getColumn();
        //取得列表框的数据
        Object data = model.getValueAt(row, column);

        String f="";
        if (column==1||column==2||column==4)
            f="";

        String Sql="UPDATE 奖金 Set "+columnNames[column]
            +"="+f+data.toString()+f
            +" where 编号="+ model.getValueAt(row,0).toString();
        MUpdata(Sql);
        table.requestFocus();
    }
});
}
catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    if (rs != null)
    {
        try{rs.close();}catch(Exception ignore){};
    }
}
```

```
        if (stmt != null)
        {
            try{stmt.close();} catch(Exception ignore){};
        }
        if (conn != null)
        {
            try{conn.close();} catch(Exception ignore){};
        }
    }
}
```

//行选择执行方法,注意，data 数组的内容实际上和表格是同步的

```
void ListRow(ListSelectionEvent e)
{
    //取得行对象
    ListSelectionModel lsm = (ListSelectionModel)e.getSource();
    selectedRow = lsm.getMinSelectionIndex();//取得行号
}
```

//窗口关闭时清空内存

```
protected void processWindowEvent(WindowEvent e) {
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
    }
}
```

//初始化代码

```
private void jbInit() throws Exception {
    contentPane=(JPanel)this.getContentPane();
    //设置窗口的标题,大小与布局
    setTitle("数据库表格程序演示");
    setSize(new Dimension(550, 340));
    contentPane.setLayout(null);

    //设置滚动条尺寸
    jScrollPane1.setBounds(new Rectangle(19, 10, 500, 230));

    jTextField1.setText("Select * from 奖金");
    jTextField1.setBounds(new Rectangle(19, 250, 400, 22));

    jButton1.setBounds(new Rectangle(420, 250, 73, 22));
}
```

```
jButton1.setText("查询");

jButton2.setBounds(new Rectangle(19, 280, 73, 22));
jButton2.setText("删除");

jButton3.setBounds(new Rectangle(100, 280, 73, 22));
jButton3.setText("添加");

//加入容器
contentPane.add(jScrollPane1, null);
contentPane.add(jTextField1, null);
contentPane.add(jButton1, null);
contentPane.add(jButton2, null);
contentPane.add(jButton3, null);

//单独加入对 jButton1 的处理事件
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {

        SeeData(jTextField1.getText());
    }
});

//单独加入对 jButton2 的处理事件
jButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String Sql=" DELETE From 奖金 Where 编号 = "+data[selectedRow][0].toString();
        MUpdata(Sql);
        SeeData(jTextField1.getText());
    }
});

//单独加入对 jButton3 的处理事件
jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String Sql=" INSERT INTO 奖金 (姓名) values (' ');
        MUpdata(Sql);
        SeeData(jTextField1.getText());
    }
});
}
}
```



编号	姓名	发奖名称	金额	备注
1	王秀琴	达标奖	550.0000	
2	朱莉文	达标奖	800.0000	
3	严毓英	工作优秀奖	400.0000	uuu
10	王秀琴	安全保证金利息	110.0000	
12	严毓英	安全保证金利息	110.0000	
14	朱莉文	安全奖	400.0000	
16	王秀琴	总厂年终奖	200.0000	
17	朱莉文	总厂年终奖	200.0000	
18	严毓英	总厂年终奖	200.0000	
19	王秀琴	1月奖金	183.2000	含科主任管理奖...
20	朱莉文	1月奖金	148.0000	

Select \* from 奖金

删除 添加 查询

### 三、使用宏语句

JDBC 规范支持宏语句，允许一个语句使用不同的参数重复的执行，这需要使用从 PreparedStatement 接口扩展过来的 prepareStatement 接口。

PreparedStatement 又称之为预编译方式执行 SQL 语句。

```
PreparedStatement prst=conn.prepareStatement("select * from DBTableName");
ResultSet rs=Prst.executeQuery();
```

```
Statement stat=conn.createStatement();
ResultSet rs=Stat.executeQuery(querysql);
```

由于 Statement 对象在每次执行 SQL 语句时都将该语句传给数据库，如果需要多次执行同一条 SQL 语句时，这样将导致执行效率特别低，此时可以采用 PreparedStatement 对象来封装 SQL 语句。如果数据库支持预编译，它可以将 SQL 语句传给数据库作预编译，以后每次执行该 SQL 语句时，可以提高访问速度；但如果数据库不支持预编译，将在语句执行时才传给数据库，其效果类同于 Statement 对象。

另外 PreparedStatement 对象的 SQL 语句还可以接收参数，可以用不同的输入参数来多次执行编译过的语句，较 Statement 灵活方便。

实现方法：

1) 创建 PreparedStatement 对象：从一个 Connection 对象上可以创建一个 PreparedStatement 对象，在创建时可以给出预编译的 SQL 语句。

```
PreparedStatement pstmt=con.prepareStatement("select * from DBTableName");
```

2) 执行 SQL 语句：可以调用 executeQuery() 来实现，但与 Statement 方式不同的是，它没有参数，因为在创建 PreparedStatement 对象时已经给出了要执行的 SQL 语句，系统并进行了预编译。

```
ResultSet rs=pstmt.executeQuery(); // 该条语句可以被多次执行
```

3) 关闭 PreparedStatement

```
pstmt.close(); //其实是调用了父类 Statement 类中的 close()方法
```

样例：

```
package Datademo;

import java.io.*;
import java.sql.*;

public class PreparedDemo {

    public static void main(String argv[])
    {
        java.sql.Connection conn= null;
        try
        {
            //加载数据库驱动程序
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
            //创建连接

            //计算机名要根据实际情况更改
            conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Passw
ord=");

            PreparedStatement pstmt=conn.prepareStatement("INSERT INTO 奖金 (姓名,发奖名称,金
额) values (?, ?, ?) ");

            pstmt.setString(1,"张三");
            pstmt.setString(2,"安全奖");
            pstmt.setInt(3,1200);

            int opNum=pstmt.executeUpdate();
            pstmt.setString(1,"李四");
            pstmt.setString(2,"生产奖");
            pstmt.setInt(3,2100);
            opNum=pstmt.executeUpdate();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            if (conn != null)
            {
                try{conn.close();} catch(Exception ignore){};
            }
        }
    }
}
```

```
    }  
}  
  
}  
}
```

其中，语句 `pstmt.setString(1,"张三")` 为送参数，而第一个数据是参数的位置。

样例二：

```
import java.io.*;  
import java.sql.*;  
  
public class prepareDemo1 {  
  
    public static void main(String argv[])  
    {  
        java.sql.Connection conn= null;  
        PreparedStatement pstmt=null;  
        java.sql.ResultSet rs=null;  
        try  
        {  
  
            //加载数据库驱动程序  
            //Driver Classname=com.microsoft.jdbc.sqlserver.SQLServerDriver  
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");  
  
            //创建连接  
  
            //计算机名要根据实际情况更改  
  
            conn = DriverManager.getConnection  
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Password=");  
  
            pstmt=conn.prepareStatement("select * from 奖金 Where 金额>?");  
            int A;  
            A=200;  
            pstmt.setInt(1,A);  
            //取得结果集  
            rs=pstmt.executeQuery();  
  
            //打印结果  
            while (rs.next()) {
```



```
        System.out.println(rs.getString(" 姓 名 ")+"          "+rs.getString(" 发 奖 名 称 ")+"
"+rs.getString("金额"));
    }
}
catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    if (rs != null)
    {
        try{rs.close();}catch(Exception ignore){};
    }
    if (pstmt != null)
    {
        try{pstmt.close();}catch(Exception ignore){};
    }
    if (conn != null)
    {
        try{conn.close();}catch(Exception ignore){};
    }
}
}
```

在可能的地方，应尽量使用宏语句，而不使用标准的 SQL 语句，使数据库能够把 SQL 语句编译成只要提供参数就能重复执行的语句，以便于提高执行速度，因为数据库不需要重复的重新编译 SQL。

#### 四、使用存储过程

**（在实际开发过程中何时使用存储过程？在返回参数和 resultset 的时候，可否同时返回，是否有顺序？）**

存储过程是一种特殊的 SQL 语句，用于对数据库进行操作。存储过程放在数据库中，它可以把复杂的查询与客户端隔离，而只给客户提供必要的查询接口。使用存储过程的好处是性能好，他在服务器上执行，离数据最接近，比直接发送 SQL 语句速度要快得多。

使用存储过程有如下几点好处：

- 1，允许模块化设计；
- 2，可以使用参数输入和输出；
- 3，减少网络流量；
- 4，更加快速的执行（创造时优化和分析，第一次使用以后，将使用内存版本，一般在数据库端计算）。

存储过程被定义以后，将成为数据库的一部分，显示为数据库对象，这就使它非常容易寻找和操纵。

存储过程还具有更好的安全性，它可以包装业务逻辑，对应用层隐藏数据库结构的改变。当然，使用存储过程也可以大大简化客户应用程序的编写。下面我们讨论 JDBC 对存储过程的调用。

例一：

首先讨论一个存储过程最简单的例子，说明 java 中是怎么调用存储过程的。

建立存储过程：

```
CREATE PROCEDURE LastName
@abc nvarchar(20)
AS
SELECT *
FROM 奖金
WHERE (姓名 LIKE '%' + @abc + '%')
```

代码：

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

```
public class SqlStore {
```

```
    public static void main(String[] args) throws Exception
    {
        //声明连接，SQL 语句执行对象和结果集变量
        java.sql.Connection conn = null;
        CallableStatement cstmt = null;
        java.sql.ResultSet rs = null;

        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Password=");

        //参数为 c
        cstmt = conn.prepareCall("{call LastName(?)}");
        cstmt.setString(1, "王");
        cstmt.execute();
        //取得结果集
        rs = cstmt.getResultSet();

        //打印结果
        while (rs.next()) {
```

```

        System.out.println(rs.getString(" 姓 名 ")+" "+rs.getString(" 发 奖 名 称 ")+" "+rs.getString("金额"));
    }
    pstmt.close();
    rs.close();    //关闭结果集
    conn.close();    //关闭数据库连接类
}
}

```

## 例二：接收输出参数

某些存储过程可能会返回输出参数，这时在执行这个存储过程之前，必须使用 `CallableStatement` 的 `registerOutParameter` 方法首先登记输出参数，在 `registerOutParameter` 方法中要给出输出参数的相应位置以及输出参数的 SQL 数据类型。在执行完存储过程以后，必须使用 `getXXX` 方法来获得输出参数的值。并在 `getXXX` 方法中要指出获得哪一个输出参数（通过序号来指定）的值。

比如：存储过程 `getTestData` 有三个输入参数并返回一个输出参数，类型分别为 `VARCHAR`。在执行完毕后，分别使用 `getString()` 方法来获得相应的值。

```

CallableStatement pstmt = con.prepareCall("{? = call getTestData (?,?,?)}");
pstmt.setString(1,Value);    //设置输入参数
pstmt.setInt(2,Value);
pstmt.setFloat(3,Value);
pstmt.registerOutParameter(1,java.sql.Types.VARCHAR); //登记输出参数
ResultSet rs = pstmt.executeQuery();    //执行存储过程
rs.getString(1);    //获得第一个字段的值
String returnResult=pstmt.getString(1);    //获得返回的输出参数的值

```

## 要点：

由于 `getXXX` 方法不对数据类型作任何转换，在 `registerOutParameter` 方法中指明数据库将返回的 SQL 数据类型，在执行完存储过程以后必须采用相应匹配的 `getXXX` 方法来获得输出参数的值。

## 实例：

我们已经说过，当存储过程需要用输出参数的时候，需要对输出参数注册，这个例子还显示了需要输入多个参数的情况，输出参数可以声明为 `output`，请注意，这个例子只是说明参数调用方法，并没有涉及数据库处理。

```

CREATE procedure MYproc
(@field1 int, @field2 int output)
as
begin

```

```
select @field2 = @field1 * @field1  
end
```

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

```
public class SqlStore2 {
```

```
    public static void main(String[] args) throws Exception  
    {
```

```
        //声明连接，SQL 语句执行对象和结果集变量
```

```
        java.sql.Connection conn          = null;  
        CallableStatement cstmt           =null;
```

```
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

```
        //假定计算机名为 COMMONOR-02A84C，要根据实际情况更改
```

```
        conn = DriverManager.getConnection  
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Passw  
ord=");
```

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

```
        //注册输出参数 2
```

```
        cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
```

```
        for (int i = 0; i < 100; i++) {
```

```
            cstmt.setInt(1, i);
```

```
            cstmt.execute();
```

```
            System.out.println(i + " " + cstmt.getInt(2));
```

```
        }
```

```
        cstmt.close();
```

```
        conn.close();          //关闭数据库联接类
```

```
    }
```

```
}
```

例三：

返回参数也可以设置成返回值，请看下面的例子。

```
CREATE procedure func_squareInt (@field1 int)  
as
```

```
begin return @field1 * @field1 end
```

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

```
public class SqlStore3 {
```

```
    public static void main(String[] args) throws Exception
    {
        //声明连接，SQL 语句执行对象和结果集变量
        java.sql.Connection conn          = null;
        CallableStatement cstmt           =null;
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

        //假定计算机名为 COMMONOR-02A84C，要根据实际情况更改
        conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Password=");

        cstmt = conn.prepareCall("{? = call func_squareInt(?)})");
        //注册输出参数 1
        cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
        for (int i = 0; i < 100; i++) {
            cstmt.setInt(2, i);
            cstmt.execute();
            System.out.println(i + " " + cstmt.getInt(1));
        }

        cstmt.close();
        conn.close();          //关闭数据库联接类
    }
}
```

通过研究这几个例子，更复杂的问题也能处理。

## 五、处理大二进制字段

处理大二进制字段主要利用字节数组，当然也需要利用流的知识。

### 1) 把图片加入数据库

```
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.*;
import java.sql.*;
import java.util.*;
import java.io.*;

public class SaveImage{

    public static void main(String[] args) throws Exception
    {
        java.sql.Connection con          = null;

        //声明连接，SQL 语句执行对象和结果集变量
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        //假定计算机名为 COMMONOR-02A84C，要根据实际情况更改
        con = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Password=");

        File files=new File("C:\\a.jpg");
        FileInputStream fis=new FileInputStream(files);
        PreparedStatement ps=con.prepareStatement("insert into photo(EMPID,photo)
values(?,?)");
        ps.setString(1,"aaa");
        ps.setBinaryStream(2,fis,(int)files.length());
        ps.executeUpdate();
        fis.close();
        con.close();
    }
}
```

## 2) 显示数据库的图片

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import java.io.*;

public class ImageFrame extends JFrame {
    JPanel contentPane;
    JLabel jLabel1 = new JLabel();

    //Construct the frame
```

```
public ImageFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) {

    ImageFrame frame = new ImageFrame();
    frame.setVisible(true);

}

//Component initialization
private void jbInit() throws Exception {
    contentPane = (JPanel) this.getContentPane();
    jLabel1.setText("jLabel1");
    jLabel1.setBounds(new Rectangle(25, 19, 353, 243));
    contentPane.setLayout(null);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Frame Title");
    contentPane.add(jLabel1, null);

    //声明连接，SQL 语句执行对象和结果集变量
    java.sql.Connection con        = null;
    java.sql.Statement stmt        = null;
    java.sql.ResultSet rs          = null;

    Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

    //假定计算机名为 COMMONOR-02A84C，要根据实际情况更改
    con = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=DataBonus;User=sa;Password=");

    //创建 SQL 语句对象
    stmt = con.createStatement();
    //执行 SQL 语句
    stmt.execute("select * from photo where id=38");
    //取得结果集
```



```
rs = stmt.getResultSet();
rs.next();
byte[] blob=rs.getBytes("PHOTO");
ImageIcon icon=new ImageIcon( blob);
jLabel1.setIcon(icon);
rs.close();
stmt.close();
con.close();
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
    }
}
}
```



## 第四节 事务处理

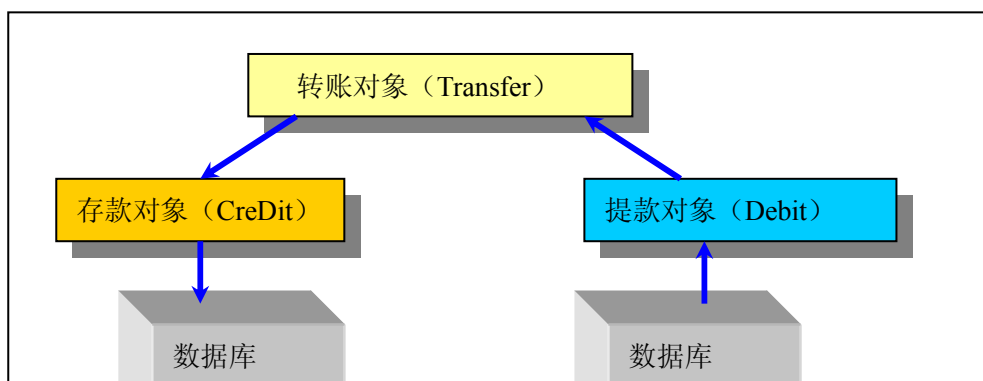
### 一、为什么要处理事务

#### （何时使用事务处理机制？）

事务是一种手段，用于保证一系列的数据库操作能够准确的完成。

使用事务的一个重要原因，是完成一个操作可能需要使用多步，其中任何一步操作的失败，都希望全部回滚。以银行转账系统为例：

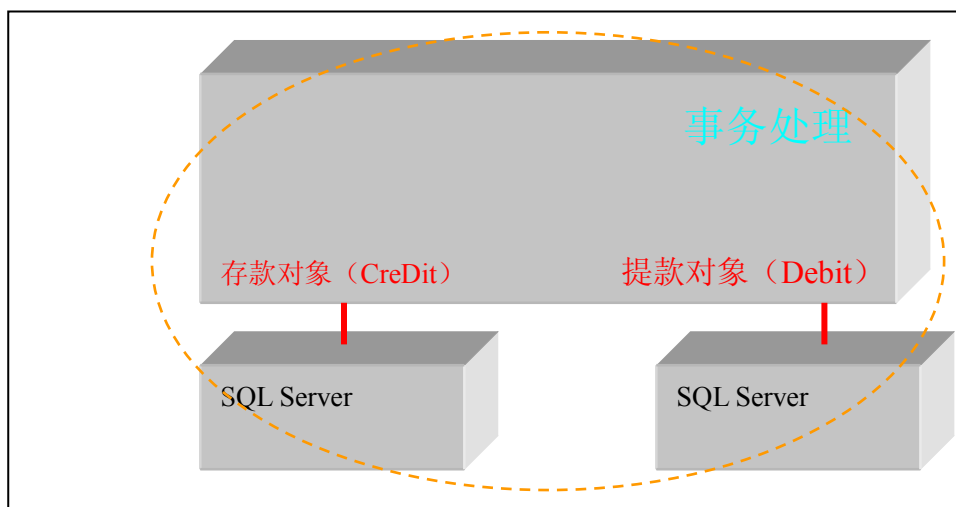
如下图所示，转账是通过两步完成的，但转账时，提款是正确的，但存款错误，总的账目将发生不平衡，显然这是不允许的。



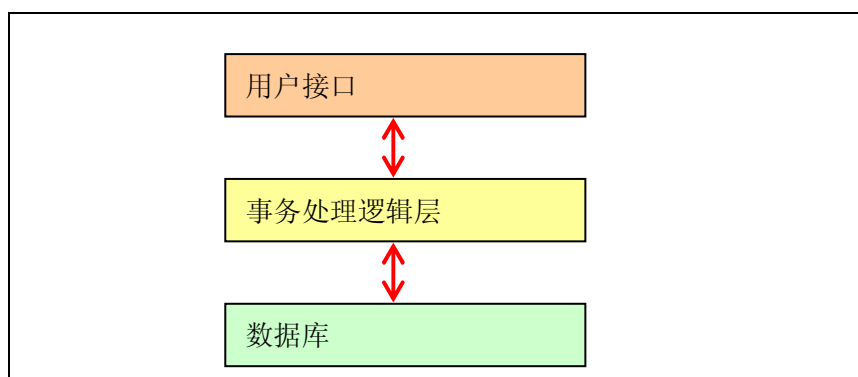
怎么解决这个问题呢？

你可以把存款、提款、转账这三个商业对象注册到一个事务中去，当某个对象无法完成操作的时候，其它的对象自动恢复成原来的状态。

这个关系可以用下面的图来表示。



进一步说，我们来考虑一个  $n$  层的应用程序。



其底层是数据库；

顶层是用户接口，在这一层中实际上不对数据做任何处理；

最令人感兴趣的是中间层，也就是“事务处理逻辑层”，在这一层中，将根据用户的要求，对数据进行各种处理。所谓事务处理，是把一个或多个链接处理单元放在一起作为整体的处理单元。

如果工作单元成功，则该工作将被提交。

如果失败，则回滚处理的每一个项目。

由于大部分电子事务处理都要处理商务，处理资金，处理交易等等，这些都要最低限度的控制风险，减少错误。

事务处理的例子。

设置两个数据库。

BankA

BankB

两个数据库都具有各自的表“账务”：

3	编号	int	4	0	主键
0	姓名	varchar	50	1	
0	金额	money	8	1	

## 二、无事务的情况

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

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

```
public class TransactionFrame extends JFrame {
```

```
    JPanel contentPane;
    JTextField jTextField1 = new JTextField();
    JTextField jTextField2 = new JTextField();
    JButton jButton1 = new JButton();
    JButton jButton2 = new JButton();
    JTextField jTextField3 = new JTextField();
    JTextPane jTextPane1 = new JTextPane();
```

```
JTextPane jTextPane2 = new JTextPane();
JTextPane jTextPane3 = new JTextPane();

java.sql.Connection conn1= null;
java.sql.Statement stmt1 =null;
java.sql.ResultSet rs1=null;

java.sql.Connection conn2= null;
java.sql.Statement stmt2 =null;
java.sql.ResultSet rs2=null;

//构造函数
public TransactionFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

//执行入口函数
public static void main(String[] args) {

    TransactionFrame frame = new TransactionFrame();
    frame.setVisible(true);
}

//控件初始化方法
private void jbInit() throws Exception {
    contentPane = (JPanel) this.getContentPane();
    jTextField1.setText("1");
    jTextField1.setBounds(new Rectangle(31, 62, 105, 25));
    contentPane.setLayout(null);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Frame Title");
    jTextField2.setText("1");
    jTextField2.setBounds(new Rectangle(243, 62, 105, 25));
    jButton1.setBounds(new Rectangle(147, 41, 81, 23));
    jButton1.setText("A > B");
    jButton1.addActionListener(new
Frame1_jButton1_actionAdapter(this));
    jButton2.setBounds(new Rectangle(146, 81, 81, 23));
```

```
        jButton2.setText("A < B");
        jButton2.addActionListener(new
Frame1_jButton2_actionAdapter(this));
        jTextField3.setText("0");
        jTextField3.setBounds(new Rectangle(132, 116, 116, 20));
        jTextPane1.setText("jTextPane1");
        jTextPane1.setBounds(new Rectangle(32, 175, 128, 25));
        jTextPane2.setText("jTextPane2");
        jTextPane2.setBounds(new Rectangle(220, 172, 128, 25));
        jTextPane3.setText("jTextPane3");
        jTextPane3.setBounds(new Rectangle(32, 220, 318, 25));
        contentPane.add(jTextField1, null);
        contentPane.add(jTextField2, null);
        contentPane.add(jButton1, null);
        contentPane.add(jButton2, null);
        contentPane.add(jTextField3, null);
        contentPane.add(jTextPane1, null);
        contentPane.add(jTextPane2, null);
        contentPane.add(jTextPane3, null);
    }

    //Overridden so we can exit when window is closed
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0);
        }
    }
}

//Banka 连接
java.sql.Connection MyConn1()
{
    try
    {
        //加载数据库驱动程序
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        //创建连接
        //计算机名要根据实际情况更改
        java.sql.Connection conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=BankA;User=sa;Password
=");

        return conn;
    }
    catch (Exception e)
    {

```

```
        e.printStackTrace();
        return null;
    }
}

//BankB 连接
java.sql.Connection MyConn2()
{
    try
    {
        //加载数据库驱动程序
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        //创建连接
        //计算机名要根据实际情况更改
        java.sql.Connection conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=BankB;User=sa;Password
=");

        return conn;

    }
    catch (Exception e)
    {
        e.printStackTrace();
        return null;
    }
}

//显示数据
void SeeData()
{
    try
    {
        conn1 = MyConn1();
        stmt1 = conn1.createStatement();
        stmt1.execute("select * from 账务 where 编号=" + jTextField1.getText());
        rs1 = stmt1.getResultSet();
        rs1.next();
        jPanel1.setText(rs1.getString("姓名")+"": " "+rs1.getString("金额"));
    }
    catch (Exception E)
    {
        E.printStackTrace();
    }
    finally
```

```
{
    if (rs1 != null)
    {
        try{rs1.close();}catch(Exception ignore){};
    }
    if (stmt1 != null)
    {
        try{stmt1.close();}catch(Exception ignore){};
    }
    if (conn1 != null)
    {
        try{conn1.close();}catch(Exception ignore){};
    }
}
try
{
    conn2 = MyConn2();
    stmt2 = conn2.createStatement();
    stmt2.execute("select * from 账务 where 编号=" + jTextField2.getText());
    rs2 = stmt2.getResultSet();
    rs2.next();
    jPanel2.setText(rs2.getString("姓名")+":  "+rs2.getString("金额"));
}
catch (Exception E)
{
    E.printStackTrace();
}
finally
{
    if (rs2 != null)
    {
        try{rs2.close();}catch(Exception ignore){};
    }
    if (stmt2 != null)
    {
        try{stmt2.close();}catch(Exception ignore){};
    }
    if (conn2 != null)
    {
        try{conn2.close();}catch(Exception ignore){};
    }
}
}
```

```
//转账 A -> B
void jButton1_actionPerformed(ActionEvent e) {

    boolean A=false;
    boolean B=false;
    try
    {
        conn1 = MyConn1();
        //开始事务
        //conn1.setAutoCommit(false);
        stmt1 = conn1.createStatement();
        stmt1.execute("select * from 账务 where 编号=" + jTextField1.getText());
        rs1 = stmt1.getResultSet();
        rs1.next();
        double D=Double.parseDouble(jTextField3.getText());
        if (rs1.getDouble("金额") >= D)
        {
            stmt1.executeUpdate("Update 账务 set 金额=金额-"+jTextField3.getText()+" where 编号 ="+jTextField1.getText());
            A=true;
        }
        conn2 = MyConn2();
        //开始事务
        //conn2.setAutoCommit(false);
        stmt2 = conn2.createStatement();
        double D1=Double.parseDouble(jTextField3.getText());
        if (D1 >= 0)
        {
            stmt2.executeUpdate("Update 账务 set 金额=金额+"+jTextField3.getText()+" where 编号 ="+jTextField2.getText());
            B=true;
        }
        if (A && B)
        {
            jTextField3.setText("转账正确");
            //事务有效
            //conn1.commit();
            //conn2.commit();
        }
        else
        {
            jTextField3.setText("转账错误");
            //事务回滚
            //conn1.rollback();
        }
    }
}
```



```
        //conn2.rollback();
    }
}
catch (Exception E)
{
    E.printStackTrace();
}
finally
{
    if (rs1 != null)
    {
        try{rs1.close();} catch(Exception ignore){};
    }
    if (stmt1 != null)
    {
        try{stmt1.close();} catch(Exception ignore){};
    }
    if (conn1 != null)
    {
        try{conn1.close();} catch(Exception ignore){};
    }
    if (rs2 != null)
    {
        try{rs2.close();} catch(Exception ignore){};
    }
    if (stmt2 != null)
    {
        try{stmt2.close();} catch(Exception ignore){};
    }
    if (conn2 != null)
    {
        try{conn2.close();} catch(Exception ignore){};
    }
}
SeeData();
}
```

```
void jButton2_actionPerformed(ActionEvent e) {
```

```
    boolean A=false;
    boolean B=false;
    try
    {
        conn2 = MyConn2();
```

```
//开始事务
//conn2.setAutoCommit(false);
stmt2 = conn2.createStatement();
stmt2.execute("select * from 账务 where 编号=" + jTextField2.getText());
rs2 = stmt2.getResultSet();
rs2.next();
double D=Double.parseDouble(jTextField3.getText());
if (rs2.getDouble("金额") >= D)
{
    stmt2.executeUpdate("Update 账务 set 金额=金额-"+jTextField3.getText()+" where 编号 ="+jTextField2.getText());
    A=true;
}
conn1 = MyConn1();
//开始事务
//conn1.setAutoCommit(false);
stmt1 = conn1.createStatement();
double D1=Double.parseDouble(jTextField3.getText());
if (D1 >= 0)
{
    stmt1.executeUpdate("Update 账务 set 金额=金额-"+jTextField3.getText()+" where 编号 ="+jTextField1.getText());
    B=true;
}
if (A && B)
{
    jTextPane3.setText("转账正确");
    //事务有效
    //conn1.commit();
    //conn2.commit();
}
else
{
    jTextPane3.setText("转账错误");

    //事务回滚
    //conn1.rollback();
    //conn2.rollback();
}
}
catch (Exception E)
{
    E.printStackTrace();
}
```

```
finally
{
    if (rs1 != null)
    {
        try{rs1.close();} catch(Exception ignore){};
    }

    if (stmt1 != null)
    {
        try{stmt1.close();} catch(Exception ignore){};
    }
    if (conn1 != null)
    {
        try{conn1.close();} catch(Exception ignore){};
    }
    if (rs2 != null)
    {
        try{rs2.close();} catch(Exception ignore){};
    }
    if (stmt2 != null)
    {
        try{stmt2.close();} catch(Exception ignore){};
    }
    if (conn2 != null)
    {
        try{conn2.close();} catch(Exception ignore){};
    }
}
SeeData();
}
```

```
class Frame1_jButtonon1_actionAdapter implements
java.awt.event.ActionListener {
    TransactionFrame adaptee;

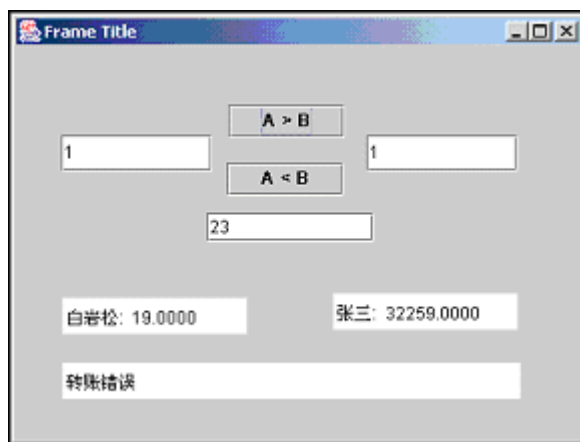
    Frame1_jButtonon1_actionAdapter(TransactionFrame adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee(jButtonon1_actionPerformed(e);
    }
}
```

```

class Frame1_jButtonon2_actionAdapter implements
java.awt.event.ActionListener {
    TransactionFrame adaptee;

    Frame1_jButtonon2_actionAdapter(TransactionFrame adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.jButton2_actionPerformed(e);
    }
}

```



可以看出来，当发生转账错误的时候，转入方的数据还是继续增加，这就造成了数据的错误，这是不能允许的。

### 三、事务处理的实例

下面讨论有事务的情况，注意，实际上是把上一个程序注释掉的部分恢复。

当发生转账错误的时候，转入方将发生回滚，结果是正确的。

注意：

事务是存在于数据库连接对象中，所以两个数据库应该各自处理各自的事务。

第二个要注意的问题是，由于事务处理使要消耗资源的，所以象查询数据这一类的操作，就不应该加入事务。

## 第五节 游标操作

下面详细讨论一下 ResultSet 对象。

### 一、ResultSet 对象

#### 1) 游标运行方式

执行完毕 SQL 语句后，将返回一个 `ResultSet` 类的对象，它包含所有的查询结果。但对 `ResultSet` 类的对象方式依赖于游标（Cursor）的类型，而对每一行中的各个列，可以按任何顺序进行处理（当然，如果按从左到右的顺序对各列进行处理可以获得较高的执行效率）；`ResultSet` 类中的 `Course`（光标）方式主要有：

### （在实际的项目之中如何使用移动光标？数据的网络传输和分页）

1. `ResultSet.TYPE_FORWARD_ONLY`（为缺省设置）：游标只能前进不能后退，也就是只能从第一个一直移动到最后一个。

对于从一个表中顺序读取所有记录的情况来说，`Forward-Only` 型的游标提供了最好的性能。获取表中的数据时，没有哪种方法比使用 `Forward-Only` 型的游标更快。但不管怎样，当程序中必须按无次序的方式处理数据行时，这种游标就无法使用了。

### 2. `ResultSet.TYPE_SCROLL_SENSITIVE`

`Scroll-Sensitive` 型游标，有时也称为 `Keyset-Driven` 游标，使用标识符。当每次在结果集移动游标时，会重新获得该标识符的数据，因为每次请求都会有网络往返，性能可能会很慢。

### 3. `ResultSet.TYPE_SCROLL_INSENSITIVE`

对于程序中要求与数据库的数据同步以及要能够在结果集中前后移动游标，使用 JDBC 的 `Scroll-Insensitive` 型游标是较理想的选择。

此类型的游标在第一次请求时就获取了所有的数据并且储存在客户端，因此，第一次请求会非常慢，特别是请求长数据时会更严重。而接下来的请求并不会造成任何网络往返并且处理起来很快。

因为第一次请求速度很慢，`Scroll-Insensitive` 型游标不应该被使用在单行数据的获取上，当有要返回长数据时，开发者也应避免使用 `Scroll-Insensitive` 型游标，因为这样可能会造成内存耗尽。

有些 `Scroll-Insensitive` 型游标的实现方式是在数据库的临时表中缓存数据来避免性能问题，但多数还是将数据缓存在应用程序中。

`ResultSet` 类中的数据是否允许修改主要有：

`ResultSet.CONCUR_READ_ONLY`（为缺省设置）：表示数据只能只读，不能更改。

`ResultSet.CONCUR_UPDATABLE`：表示数据允许被修改。

可以在创建 `Statement` 或 `PreparedStatement` 对象时指定 `ResultSet` 的这两个特性。

`Statement`

```
stmt=con.createStatement(ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_READ_ONLY);
```

或

```
PreparedStatement pstmt=con.prepareStatement("insert into bookTable values (?,?,?)",ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
```

## 2) 游标的移动

`ResultSet` 类的对象维持一个指向当前行的指针，利用 `ResultSet` 类的 `next()` 方法可以移动到下一行（在 JDBC 中，Java 程序一次只能看到一行数据），如果 `next()` 的返回值为 `false`，则说明已到记录集的尾部。另外 JDBC 也没有类似 ODBC 的书签功能的方法。

### 3) 获取游标数据

利用 ResultSet 类的 getXXX()方法可以获得某一列的结果，其中 XXX 代表 JDBC 中的 Java 数据类型，如 getInt()、getString()、getDate()等。访问时需要指定要检索的列（可以采用 int 值作为列号（从 1 开始计数）或指定列（字段）名方式，但字段名不区别字母的大小写）。

while(rs.next())

```
{ String name=rs.getString("Name");//采用“列名”的方式访问数据
  int age=rs.getInt("age");
  float wage=rs.getFloat("wage");
  String homeAddress=rs.getString(4); //采用“列号”的方式访问数据
}
```

### 4) 数据转换

利用 ResultSet 类的 getXXX()方法可以实现将 ResultSet 中的 SQL 数据类型转换为它所返回的 Java 数据类型。

### 5) NULL 结果值

**（为什么会有不同的结果？）**

要确定给定结果值是否是 JDBC NULL，必须先读取该列，然后使用 ResultSet.wasNull 方法检查该次读取是否返回 JDBC NULL。

当使用 ResultSet.getXXX 方法读取 JDBC NULL 时，方法 wasNull 将返回下列值之一：

（1）Javanull 值

对于返回 Java 对象的 getXXX 方法（例如 getString、getBigDecimal、getBytes、getDate、getTime、getTimestamp、getAsciiStream、getUnicodeStream、getBinaryStream、getObject 等）。

（2）零值：对于 getByte、getShort、getInt、getLong、getFloat 和 getDouble。

（3）false 值：对于 getBoolean

### 6) 获得结果集中的结构信息

利用 ResultSet 类的 getMetaData()方法来获得结果集中的一些结构信息（主要提供用来描述列的数量、列的名称、列的数据类型。利用 ResultSetMetaData 类中的方法）。

```
ResultSetMetaData rsmd=rs.getMetaData();
rsmd.getColumnCount(); //返回结果集中的列数
rsmd.getColumnLabel(1); //返回第一列的列名（字段名）
```

例如：

```
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from TableName");
for(int i=1; i<=rs.getMetaData().getColumnCount(); i++) //跟踪显示各个列的名称
```

```
        {    System.out.print(rs. getColumnName (i)+"\t");
        }
while(rs.next())
{ //跟踪显示各个列的值
    for(int j=1; j<=rs.getMetaData().getColumnCount(); j++)
    {    System.out.print(rs.getObject(j)+"\t");
    }
}
```

## 二、游标操作的实例分析

ResultSet 游标操作的方法如下表所示：

方法	使用说明
first()	移到第一条记录
beforeFirst()	移到数据集的开头
isFirst()	判断是否为第一条记录
isBeforeFirst()	判断是否为数据集的开头
last()	移到最后一条记录
afterLast()	移到数据集的结尾
isLast()	判断是否为最后一条记录
isAfterLast()	判断是否为数据集的结尾
next()	下移一条记录
previous()	上移一条记录
absolute(int row)	移到指定数字的记录
relative(int rows)	移到相对于当前位置的某条记录
moveToInsertRow()	移到新插入的记录

注意：

操作数据库的游标，不能使用 `Statement stmt=con.createStatement()` 来获取 `Statement` 对象，而应该使用：

```
Statement stmt=con.createStatement(ResultSet.TYPEY,ResultSet.CONCUR);
或
PreparedStatement pstmt=con.PrepareStatement("insert into bookTable values
(?,?,?)",ResultSet.TYPE,ResultSet.CONCUR);
```

其中： `ResultSet.TYPEY` 为游标类型；  
`ResultSet.CONCUR` 为数据是否能被修改。

下面举个例子：

```
import java.sql.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class CursorScrollableDemo extends JFrame {
    JPanel contentPane = new JPanel();
    String query, companyName, phone;
    int shipID;
    Connection conn;           //建立连接类
    Statement statement;       //建立 Sql 语句执行类
    ResultSet resultSet = null; //建立结果集类

    //建立可视化控件
    JTextField textFieldID = new JTextField();
    JTextField textFieldCompany = new JTextField();
    JTextField textFieldPhone = new JTextField();
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();
    JLabel jLabel3 = new JLabel();
    JPanel jPanel1 = new JPanel();
    JButton buttonFirst = new JButton();
    JButton buttonPrior = new JButton();
    JButton buttonNext = new JButton();
    JButton buttonLast = new JButton();

    //CursorScrollableDemo 的构造器
    public CursorScrollableDemo(){
        try {jbInit();} catch(Exception e) {}
    }

    //程序的主方法,创建并显示窗口
    public static void main(String[] args) throws Exception {
        new CursorScrollableDemo().setVisible(true);
    }

    private void jbInit() throws Exception {

        contentPane = (JPanel)this.getContentPane();
        contentPane.setLayout(null);
        this.setTitle("游标操作");
        this.setSize(new Dimension(400, 280));

        //退出窗口时清空内存及数据集的内容
```



```
this.addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        try{
            resultSet.close();    //关闭结果集
            statement.close();    //关闭 sql 语句执行类
            conn.close();        //关闭数据库联接类
        }catch(Exception e1){}
        System.exit(0);        //清空内存
    }
});
//加载数据库驱动程序

    Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

//创建连接

//计算机名要根据实际情况更改
//Northwind(罗斯文数据库)指 SQL Server 自带的样例数据库的名字
    conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=Northwind;User=sa;Password=");
//创建操作数据集游标的 Sql 执行类
statement = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                ResultSet.CONCUR_READ_ONLY);

//获得数据库的所有记录
query = "select * from Shippers";
resultSet = statement.executeQuery(query);
//获取第一条记录的内容
resultSet.first();
//设置标签的标题,位置与大小
jLabel1.setText("记录号");
jLabel1.setBounds(new Rectangle(88, 47, 50, 17));
jLabel2.setText("公司名");
jLabel2.setBounds(new Rectangle(88, 98, 63, 25));
jLabel3.setText("电话号码");
jLabel3.setBounds(new Rectangle(88, 158, 74, 19));
//设置编辑框的位置,大小
textFieldID.setBounds(new Rectangle(169, 39, 154, 37));
textFieldCompany.setBounds(new Rectangle(169, 91, 154, 37));
textFieldPhone.setBounds(new Rectangle(169, 143, 154, 37));
//在编辑框的显示数据库的第一条记录
showContent();
//加入工具栏,及设置四个按钮的事件
jPanel1.setBounds(new Rectangle(19, 203, 364, 44));
```

```
buttonFirst.setText("第一条");
buttonFirst.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        buttonFirst_actionPerformed(e);
    }
});
buttonPrior.setText("上一条");
buttonPrior.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        buttonPrior_actionPerformed(e);
    }
});
buttonNext.setText("下一条");
buttonNext.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        buttonNext_actionPerformed(e);
    }
});
buttonLast.setText("最后一条");
buttonLast.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        buttonLast_actionPerformed(e);
    }
});
jPanel1.add(buttonFirst, null);
jPanel1.add(buttonPrior, null);
jPanel1.add(buttonNext, null);
jPanel1.add(buttonLast, null);

//向容器类加入各种控件
contentPane.add(textFieldID, null);
contentPane.add(textFieldCompany, null);
contentPane.add(textFieldPhone, null);
contentPane.add(jLabel1, null);
contentPane.add(jLabel2, null);
contentPane.add(jLabel3, null);
contentPane.add(jPanel1, null);
}

//显示数据表记录的方法
void showContent(){
    try{
        //分别取得数据表的内容
        shipID = resultSet.getInt(1);
```

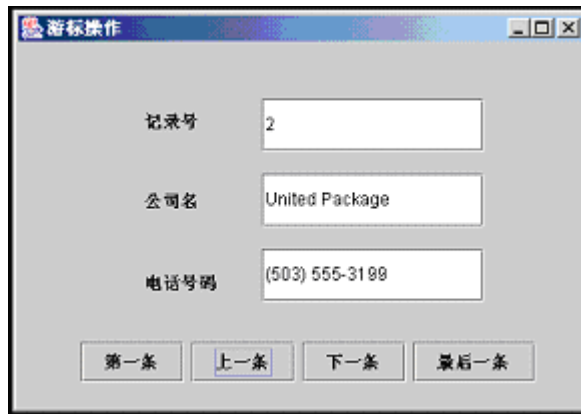
```
        companyName = resultSet.getString(2);
        phone = resultSet.getString(3);
        //在编辑框显示数据表的内容
        textFieldID.setText(String.valueOf(shipID));
        textFieldCompany.setText(companyName);
        textFieldPhone.setText(phone);
    } catch (Exception el) { //如果不存在该记录,显示"空记录"字符串
        textFieldID.setText("空记录");
        textFieldCompany.setText("空记录");
        textFieldPhone.setText("空记录");
    }
}

void buttonFirst_actionPerformed(ActionEvent e) {
    try {
        resultSet.first();    //移到数据集的第一条记录
        showContent();        //显示数据集的内容
    } catch (Exception el) { showContent(); }
}

void buttonPrior_actionPerformed(ActionEvent e) {
    try {
        resultSet.previous(); //上移一条记录
        showContent();        //显示数据集的内容
    } catch (Exception el) { showContent(); }
}

void buttonNext_actionPerformed(ActionEvent e) {
    try {
        resultSet.next();     //下移一条记录
        showContent();        //显示数据集的内容
    } catch (Exception el) { showContent(); }
}

void buttonLast_actionPerformed(ActionEvent e) {
    try {
        resultSet.last();     //移到数据集的最后一条记录
        showContent();        //显示数据集的内容
    } catch (Exception el) { showContent(); }
}
}
```



### 三、更新数据库

更新数据库数据，除了可以使用 SQL 语句以外，也可以使用 ResultSet 类的 updateXXX 方法，updateXXX 方法后面要加上适当的类型，如：updateInt、updateString 等。

ResultSet 类有关更新操作的方法如下：

1. updateXXX(int columnName, Type content)

其中：

columnName 需要更新的列；

content 更新后的数据。

2. updateRow()

向数据库提交更新的内容。

3. rowUpdated()

判断该行记录的更新是否已经向数据库提交。

4. cancelRowUpdates()

取消向数据库更新前的所有更新操作。

样例：

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

```
public class UpdateDemo {
```

```
    static String query, regionID, regionDescription;
```

```
    static Connection conn;           //建立连接类
```

```
    static Statement statement;       //建立 Sql 语句执行类
```

```
    static ResultSet resultSet = null; //建立结果集类
```

```
    public static void main(String[] args) throws Exception {
```

```
        //加载数据库驱动程序
```

```
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

```

//创建连接
//计算机名要根据实际情况更改
//Northwind(罗斯文数据库)指 SQL Server 自带的样例数据库的名字
conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=Northwind;User=sa;Passw
ord=");
//创建可以滚动,更新数据的 sql 语句执行类
statement = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                ResultSet.CONCUR_UPDATABLE);

//获得数据库的所有记录
query = "select * from Region";
resultSet = statement.executeQuery(query);
resultSet.absolute(2);
showContent(); //显示更新前的内容
resultSet.updateInt(1,2);
if(regionDescription.trim().equals("西方")){
    resultSet.updateString(2,"Western");//使用 updateString()方法更新
}else{
    resultSet.updateString(2,"西方");    //使用 updateString()方法更新
}
//resultSet.cancelRowUpdates();
resultSet.updateRow();
showContent(); //显示更新后的内容
resultSet.close();    //关闭结果集
statement.close();    //关闭 sql 语句执行类
conn.close();        //关闭数据库联接类
}

static void showContent() throws Exception{
    regionID = resultSet.getString(1);
    regionDescription = resultSet.getString(2);
    System.out.print("地区的标识 = " + regionID);
    System.out.println(", 地区的名字 = " + regionDescription);
}
}

```

#### 四、插入操作

插入数据库数据,除了可以使用 SQL 语句以外,也可以使用 ResultSet 类的方法进行插入操作。

插入操作的方法如下:

1) moveToInsertRow()

创建新的插入行。

2) insertRow()

向数据库提交插入操作。

样例：

```
import java.sql.*;

public class InsertDemo {
    static String query, regionID, regionDescription;

    static Connection conn;           //建立连接类
    static Statement statement;        //建立 Sql 语句执行类
    static ResultSet resultSet = null; //建立结果集类

    public static void main(String[] args) throws Exception {

        //加载数据库驱动程序
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

        //创建连接
        //计算机名要根据实际情况更改
        //Northwind(罗斯文数据库)指 SQL Server 自带的样例数据库的名字
        conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=Northwind;User=sa;Passw
ord=");

        //创建可以滚动,更新数据的 sql 语句执行类
        statement = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                           ResultSet.CONCUR_UPDATABLE);

        //获得数据库的所有记录
        query = "select * from Region";
        resultSet = statement.executeQuery(query);
        resultSet.moveToInsertRow(); //创建新插入的记录
        //注意，这是主键，不允许重复的，比要时可设为自动添加，这一句可以注释掉
        resultSet.updateInt(1,5); //更新字段 1 的内容
        resultSet.updateString(2,"东南方");//更新字段 2 的内容
        resultSet.insertRow(); //向数据库提交插入操作
        resultSet.absolute(5); //移到新插入记录的行
        showContent(); //显示插入的内容
        resultSet.close(); //关闭结果集
        statement.close(); //关闭 sql 语句执行类
        conn.close(); //关闭数据库联接类
    }

    static void showContent() throws Exception {
        regionID = resultSet.getString(1);
```

```
        regionDescription = resultSet.getString(2);
        System.out.print("地区的标识 = " + regionID);
        System.out.println(", 地区的名字 = " + regionDescription);
    }
}
```

## 五、删除操作

删除数据库数据，除了可以使用 SQL 语句以外，也可以使用 `ResultSet` 类的方法进行删除操作。

删除操作的方法如下：

`deleteRow()` 删除当前行的记录。

样例：

```
import java.sql.*;

public class DeleteDemo {
    static String query, regionID, regionDescription;

    static Connection conn;           //建立连接类
    static Statement statement;       //建立 Sql 语句执行类
    static ResultSet resultSet = null; //建立结果集类

    public static void main(String[] args) throws Exception {
        //加载数据库驱动程序
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        //创建连接
        //计算机名要根据实际情况更改
        //Northwind(罗斯文数据库)指 SQL Server 自带的样例数据库的名字
        conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=Northwind;User=sa;Password=");

        //创建可以滚动,更新数据的 sql 语句执行类
        statement = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                         ResultSet.CONCUR_UPDATABLE);

        //获得数据库的所有记录
        query = "select * from Region";
        resultSet = statement.executeQuery(query);
        resultSet.absolute(5);           //移到第 5 行
        resultSet.deleteRow();           //删除第 5 行的记录
        resultSet.close();               //关闭结果集
    }
}
```

```
statement.close();    //关闭 sql 语句执行类
conn.close();         //关闭数据库联接类
}

static void showContent() throws Exception{
    regionID = resultSet.getString(1);
    regionDescription = resultSet.getString(2);
    System.out.print("地区的标识 = " + regionID);
    System.out.println(", 地区的名字 = " + regionDescription);
}
}
```

## 六、批量操作

使用 Statement 类可以进行一系列的 SQL 命令执行，这种批量操作可以减少连接数据库的时间。批量操作的方法如下：

- 1) addBatch(String sql)  
向 Statement 类加入 SQL 命令。
- 2) executeBatch()  
执行一系列的 SQL 命令。

样例：

```
import java.sql.*;

public class BatchDemo {
    static String query, regionID, regionDescription;

    static Connection conn;           //建立连接类
    static Statement statement;       //建立 Sql 语句执行类
    static ResultSet resultSet = null; //建立结果集类

    public static void main(String[] args) throws Exception {

        //加载数据库驱动程序
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        //创建连接
        //计算机名要根据实际情况更改
        //Northwind(罗斯文数据库)指 SQL Server 自带的样例数据库的名字
        conn = DriverManager.getConnection
("jdbc:microsoft:sqlserver://COMMONOR-02A84C:1433;DatabaseName=Northwind;User=sa;Passw
ord=");

        //创建可以滚动,更新数据的 sql 语句执行类
```



```
statement = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                   ResultSet.CONCUR_UPDATABLE);

//使用 Batch 方法加入一系列 sql 命令
statement.addBatch("insert into region values(5,'东南方')");
statement.addBatch("insert into region values(6,'东北方')");
statement.addBatch("insert into region values(7,'西南方')");
statement.addBatch("insert into region values(8,'西北方')");
//使用 executeBatch()方法执行所有 sql 语句,返回的结果保存在数组 updateCounts
int[] updateCounts = statement.executeBatch();
for(int i = 0; i < updateCounts.length; ++i){
    //如果更新成功,返回的数值是 1
    System.out.println(updateCounts[i]);
}

statement.close();    //关闭 sql 语句执行类
conn.close();         //关闭数据库连接类
}

static void showContent() throws Exception{
    regionID = resultSet.getString(1);
    regionDescription = resultSet.getString(2);
    System.out.print("地区的标识 = " + regionID);
    System.out.println(", 地区的名字 = " + regionDescription);
}
}
```