如鹏网 jdbc 入门通透讲解

1.1 JDBC 入门

1.1.1 JDBC 简介

JDBC 是 Java 平台上的数据库访问标准，几乎所有数据库都支持通过 JDBC 进行访问。和微软平台的 ODBC、ADO、ADO.NET 等类似，JDBC 的作用是屏蔽 Java 程序访问各种不同数据库操作的差异性。使用 JDBC，开发人员可以通过同样的程序接口访问不同的数据库，大大增强了系统的可移植性，同时也简化了开发人员的工作。

需要注意的就是，JDBC 只是一个定义了访问接口的标准规范，针对不同的数据库有不同的实现，这些不同的时间被称为对应数据库平台的“JDBC 驱动”。不过开发人员无需关心这些“JDBC 驱动”的实现细节，只需在程序初始化的时候指定采用哪个“JDBC 驱动”即可，其后的数据库操作完全是标准的 JDBC 操作。

1.1.2 JDBC 驱动

不同的数据库有相对应的不同的 JDBC 驱动，这些 JDBC 驱动通常是以 Jar 包的形式存在的，在使用这些驱动的时候需要到相应网站上去下载，下面列出常用数据库的驱动下载地址：

MySQL:http://www.mysql.com

Oracle:http://www.oracle.com/

Microsoft SQLServer: http://www.microsoft.com

DB2:http://www.ibm.com

下载完成以后把下载包中的相应的 jar 包添加到 CLASSPATH 中即可。这些 JDBC 驱动中有一个驱动类是暴露给开发人员的，在使用相应的 JDBC 驱动之前必须首先加载这个驱动类，在后边将讲解如何加载这个 JDBC 驱动类，这里只列出这些驱动类的名称：

MySQL: com.mysql.jdbc.Driver

Oracle:oracle.jdbc.driver.OracleDriver

Microsoft SQLServer:com.microsoft.jdbc.sqlserver.SQLServerDriver

DB2:Com.ibm.db2.jdbc.net.DB2Driver

1.1.3 连接字符串

在连接数据库的时候，我们必须告诉 JDBC 关于要连接数据库的具体信息，比如数据库服务器的 IP 地址是多少、要连接的数据库名字是什么等等。由于这些连接信息的复杂性以及在各个不同数据库平台上信息的差异性，因此我们需要把这些信息以字符串的形式发送给 JDBC。不同的 JDBC 驱动对字符串的格式要求也是不同的，这是学习 JDBC 的一个难点，因此下面我们列出这些产用的 JDBC 驱动的连接字符串的格式：

（1） MySQL:jdbc:mysql://数据库的主机名或者 IP 地址:3306/数据库名。例如我要连接 IP 地址为 192.168.88.88 的 MySQL 服务器上的名字为 afa 的数据库，那么正确的连接字符串即为：jdbc:mysql://192.168.88.88:3306/afa。

（2） Oracle：jdbc:oracle:thin:@数据库的主机名或者 IP 地址:1521:数据库名。例如我要连接 IP 地址为 192.168.88.88 的 Oracle 服务器上的名字为 afa 的数据库，那么正确的连接字符串即为：jdbc:oracle:thin:@192.168.88.88:1521:afa。

（3） Microsoft SQLServer：jdbc:microsoft:sqlserver://数据库的主机名或者 IP 地址:1433;databaseName=数据库名。例如我要连接 IP 地址为 192.168.88.88 的 Microsoft SQLServer 服务器上的名字为 afa 的数据库，那么正确的连接字符

串即为：sqlserver://192.168.88.88:1433;databaseName=afa。

（4） DB2：jdbc:db2://数据库的主机名或者 IP 地址:6789/数据库名。例如我要连接 IP 地址为 192.168.88.88 的 DB2 服务器上的名字为 afa 的数据库，那么正确的连接字符串即为：jdbc:db2://192.168.88.88:6789/afa。

注意：如果要连接本机的数据库，则“数据库的主机名或者 IP 地址”一般填为 127.0.0.1 或者 localhost 即可。

1.1.4 JDBC 的常见概念

连接（Connection）：“连接”是建立在 Java 程序和数据库服务器之间的一条通路，在 Java 程序对数据库进行操作之前必须首先建立一个“连接”，在操作完成后需要关闭“连接”。

语句（Statement）：“语句”是 Java 程序向数据库服务器发起的操作命令，也就是它代表了 Java 程序要求数据库服务器执行的 SQL 语句。

结果集（ResultSet）：“结果集”是数据库服务器执行“语句”的执行结果，Java 程序可以通过读取“结果集”来取得 SQL 语句的执行结果。

1.2 JDBC 实战

本节我们将以一个实例介绍如何通过 JDBC 访问 MySQL 数据库。

第一步、启动 MySQL 数据库，在 MySQL 数据库中创建一个数据库 afa，并创建一张代表人员的表:T_Person，建表 SQL 语句为：

```
Create Table T_Person(FId Varchar(50) not null,FName Varchar(50) not null,FAge int not null,primary key(FId));
```

在数据库中插入一条记录：

```
Insert into T_Person(FId,FName,FAge) values('1','tom',22);
```

第二步、将 MySQL 的 JDBC 驱动 Jar 包放到 CLASSPATH 中；

第三步、建立一个 Java 文件 JDBCTest：

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

    }
}
```

第四步、加载MySQL的JDBC驱动：

```
Class.forName("com.mysql.jdbc.Driver");
```

可以看到加载数据库驱动的方法就是：Class.forName(驱动类名);

由于加载数据库驱动的时候有可能会发生驱动找不到的异常，因此需要捕捉这个异常并且将这个异常情况报告给客户，这里我们使用最简单的输出字符串的方式报告：

```
try
{
    Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException e)
{
    System.out.println("数据库驱动找不到！");
    return;
```

```
        }
```

第五步、建立和数据库的连接：

JDBC中建立和数据库连接的方式非常简单，只要调用java.sql.DriverManager类的静态方法getConnection即可。getConnection接受三个字符串类型的参数：第一个参数为连接字符串；第二个参数为连接数据库的用户名；第二个参数为连接数据库的密码。getConnection的返回值为代表数据库连接的java.sql.Connection对象。

在这里，我们要建立IP地址为本地（即localhost），数据库名称为afa的MySQL数据库，用户名为root，密码为空，因此如下建立连接：

```
Connection conn = null;
conn = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/afa","root","");
```

第六步、创建查询SQL语句：

建立连接以后就可以创建要执行的SQL语句，调用数据库连接的prepareStatement方法即得到代表SQL语句的java.sql.PreparedStatement对象，prepareStatement的参数为要执行的SQL语句。这里要查询T_Person表中的所有数据，因此如下创建SQL语句对象：

```
PreparedStatement ps = null;
ps = conn.prepareStatement("select * from T_Person");
```

第七步、执行SQL语句对象：

PreparedStatement有两个主要的方法：executeQuery用于执行Select等有返回结果集的SQL语句，而execute则用于执行Delete、Update语句等没有返回结果集的SQL语句。这里我们要执行的是select语句，因此使用executeQuery方法执行：

```
ResultSet rs = null;
rs = ps.executeQuery();
```

executeQuery方法的返回值类型为java.sql.ResultSet，它代表执行以后的结果集，读取ResultSet即可得到所有的执行结果。

第八步、读取执行结果集

因为执行结果集数量非常大，为了减少资源占用，ResultSet采用了类似于游标的处理方式，也就是每次只处理一行结果集，当要处理其他行的时候必须移动游标到相应行上。游标可以位于结果集的任何一行，也可以位于结果集的第一行之前的位置，也可以位于结果集的最后一行之后的位置。初始状态时游标位于结果集的第一行之前的位置。

ResultSet中提供了很多移动游标的方法，主要包括：

next()：向下一行游标。

beforeFirst()：将游标移动到第一行之前的位置。

afterLast()：将游标移动到最后一行之后的位置。

first()：将游标移动到第一行。

last()：将游标移动到最后一行。

absolute(int row)：将游标移动到第row行。

所有数据库都支持next()方法移动游标，而其他移动游标的方法在很多情况下不被支持，因此我们通常只使用next()方法进行游标控制，而且实际业务中也通常是从上到下逐行进行结果集读取，因此next()方法一般也就够用了。 next()方法的返回值表示是否已经到了结果集的最后一行之后的位置。

ResultSet提供了读取当前行中各个列中数据值的方法：

- String getString(String columnName)：取得列名为columnName的值，返回类型为String。

- boolean getBoolean(String columnName)：取得列名为columnName的值，返回类型为boolean。
- int getInt(String columnName)：取得列名为columnName的值，返回类型为int。
- java.sql.Date getDate(String columnName)：取得列名为columnName的值，返回类型为java.sql.Date。

ResultSet也提供了根据列索引号来取得值的方法，但是出于程序可读性和可维护性的考虑，我们不推荐这种使用方式。

讲解了这些基础知识，下面就来显示第七步的执行结果：

```
while (rs.next())
{
    StringBuffer sb = new StringBuffer();
    sb.append("主键:").append(rs.getString("FId"))
    .append(";姓名:").append(rs.getString("FName"))
    .append(";年龄:").append(rs.getInt("FAge"));
    System.out.println(sb);
}
```

这里使用ResultSet的next()方法做为循环退出的判断条件。next()有两个作用，第一个作用是将游标下移一个位置，第二个作用就是判断是否已经处理到最后一行。执行结果如下：

主键:1;姓名:tom;年龄:22

第九步、异常处理和资源回收

SQL执行的过程中有可能发生各种异常，这些异常的类型通常是SQLException，我们需要对这些异常进行处理，这里我们采用最简单的打印异常消息的处理方式：

```
catch (SQLException e)
        {
System.out.println("数据库执行过程中发生异常情况:" +
e.getMessage());
        }
```

执行完数据库操作以后我们需要对数据库连接、SQL语句对象、结果集等进行资源关闭等回收操作，否则会造成数据库资源泄露。为了保证资源一定会被回收我们将资源回收的工作放到finally块中来执行：

```
finally
    {
        if (rs != null)
        {
            try
            {
                rs.close();
            } catch (SQLException e)
            {

            }
        }
```

```
            if (ps != null)
            {
                try
                {
                    ps.close();
                } catch (SQLException e)
                {

                }
            }

            if (conn != null)
            {
                try
                {
                    conn.close();
                } catch (SQLException e)
                {

                }
            }
}
```

需要注意的是，在调用close的方法中也有可能抛出SQLException异常，为了不影响后续的资源回收操作，这里通常要对close方法抛出的异常捕获。

下面是这个例子的全部代码：

```
package com.test;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JDBCTest
{
    public static void main(String[] args)
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e)
        {
            System.out.println("数据库驱动找不到！");
            return;
```

```
        }
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try
        {
            conn = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/afa", "root", "");
            ps = conn.prepareStatement("select * from T_Person");
            rs = ps.executeQuery();
            while (rs.next())
            {
                StringBuffer sb = new StringBuffer();
                sb.append("主键:").append(rs.getString("FId")).append(";
姓名:")
                        .append(rs.getString("FName")).append(";年
龄:").append(
                                rs.getInt("FAge"));
                System.out.println(sb);
            }
        } catch (SQLException e)
        {
            System.out.println("数据库执行过程中发生异常情况:" +
e.getMessage());
        } finally
        {
            if (rs != null)
            {
                try
                {
                    rs.close();
                } catch (SQLException e)
                {

                }
            }

            if (ps != null)
            {
                try
                {
                    ps.close();
                } catch (SQLException e)
                {
```

```
            }
        }

        if (conn != null)
        {
            try
            {
                conn.close();
            } catch (SQLException e)
            {

            }
        }

    }
}
```

对应的Java工程为：

1.3 JDBC其他问题

1.3.1 执行没有结果集的SQL

```
            ps = conn.prepareStatement("Update T_Person set Fage=20");
            rs = ps.execute();
```

1.3.2 参数化SQL

在执行SQL的时候经常需要拼一些查询条件，比如查询名称等于用户输入的name值的人员，我们可以采用拼SQL语句的方法来进行：

```
            ps = conn.prepareStatement("select * from T_Person where
Fname='"+name+"'");
            rs = ps.executeQuery();
```

这样有可能会被人采用注入SQL的方式进行攻击，而且也会有执行效率的问题。更好的方式是使用参数化SQL来解决。参数化SQL将SQL语句中需要在运行期才能确定的参数用？表示，并且在运行时再通过JDBC提供的方法为这些参数赋值。

比如我们要查询名称等于用户输入的name值的人员，只要如下编写即可：

```
package com.test;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;

public class JDBCParamTest
{
    public static void main(String[] args)
    {
        //从控制台中读取用户输入的参数
        System.out.print("请输入要查询的用户的名称: ");
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String name = "";
        try
        {
            name = br.readLine();
        } catch (IOException e)
        {
            e.printStackTrace();
            return;
        }


        try
        {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e)
        {
            System.out.println("数据库驱动找不到! ");
            return;
        }
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try
        {
            conn = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/afa", "root", "");
            ps = conn.prepareStatement("select * from T_Person where
FName=?");
            ps.setString(1, name);
            rs = ps.executeQuery();
            while (rs.next())
            {
                StringBuffer sb = new StringBuffer();
```

```
                sb.append("主键:").append(rs.getString("FId")).append(";
姓名:")
                        .append(rs.getString("FName")).append(";年
龄:").append(
                                rs.getInt("FAge"));
                System.out.println(sb);
            }
        } catch (SQLException e)
        {
            System.out.println("数据库执行过程中发生异常情况:" +
e.getMessage());
        } finally
        {
            if (rs != null)
            {
                try
                {
                    rs.close();
                } catch (SQLException e)
                {

                }
            }

            if (ps != null)
            {
                try
                {
                    ps.close();
                } catch (SQLException e)
                {

                }
            }

            if (conn != null)
            {
                try
                {
                    conn.close();
                } catch (SQLException e)
                {

                }
```

```
            }

        }

    }
}
```

代码一开始从控制台中读取用户输入的值，并将输入的值赋值给字符串name。从控制台中读取用户输入的值的代码涉及到了流操作，如果你目前不熟悉也无所谓，只要知道这段代码的作用就可以了。

这段代码的核心就是红色粗体字部分：

```
ps = conn.prepareStatement("select * from T_Person where FName=?");
ps.setString(1, name);
```

这里采用了参数化的SQL："select * from T_Person where FName=?"，可以看到采用？为Fname的值预留了位置；接着调用ps的setString方法为这个参数赋值，注意在这里参数的索引号是从1开始的，而非通常的0，这是初学者经常犯错误的地方。还可以为参数赋予其他类型的参数值，比如赋整数类型的setInt、赋日期类型的setDate等。

还需要特别注意的就是参数只适用于where语句以及计算列等部分，而不能将表名、字段名等部分也用参数代替。比如根据用户输入的表名来查询某个表，很多初学者会这么写(示例代码)：

```
ps = conn.prepareStatement("select * from ? ");
ps.setString(1, "T_Person");
```

这是不对的，在运行的时候会报语法错误。