开源代码 网址

Jdbc.jar和JdkSE完全开源

JDBC主要接口代码

先我们要先了解JDBC(Java Database Connectivity)是java数据库连接是一种用于执行SQL语句的java API可以为多种数据库提供统一的访问语言，它由一组用java语言编写的类和接口组成，JDBC提供了一种基准，据此可以构建更高级的工具和接口，如:Mybatis就是JDBC的一种扩充，简单来说JDBC就是为做三件事：与数据库**建立联系**，**发送**操作数据库的语句并**处理结果**。

传统链接方式：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class MyTest {

    public static void main(String[] args) {
        Connection con = null;
        Statement st = null;
        ResultSet rs = null;
        //源文件编码要与数据库编码一致
            String url = "jdbc:mysql://127.0.0.1:3306/stu_course?
useUnicode=true&characterEncoding=utf8";
        String user = "root";
        String pwd = "qwer15928198";
        try {
          // 获得MySQL驱动的实例
          Class.forName("com.mysql.jdbc.Driver").newInstance();
          // 获得连接对象(提供：地址，用户名，密码)
          con =  DriverManager.getConnection(url,user, pwd);
          if (!con.isClosed())
            System.out.println("Successfully connected ");
          else
            System.out.println("failed connected");.
          //建立一个Statement，数据库对象
          st = con.createStatement();
          // 运行SQL查询语句
          String sql = "select Sno,Sname,Ssex from student;";
          rs = st.executeQuery(sql);
          // 读取结果集
          System.out.println(" 学号     姓名    性别");
          while (rs.next()) {
              System.out.print( rs.getString(Sno));
              System.out.print("    ");
              System.out.print( rs.getString(Sname));
              System.out.print("    ");
              System.out.println( rs.getString(Ssex));
```

```
            }
            // 关闭链接
            con.close();
        } catch(Exception e) {
            System.err.println("Exception: " + e.getMessage());
        }
    }
}
```

就先以这段代码来开始分析：

一、加载驱动类：

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

com.mysql.jdbc.Driver的源代码：

```
package com.mysql.jdbc;

import java.sql.SQLException;

public class Driver extends com.mysql.cj.jdbc.Driver {
    public Driver() throws SQLException {
    }

    static {
        System.err.println("Loading class `com.mysql.jdbc.Driver'. This is
deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is
automatically registered via the SPI and manual loading of the driver class is
generally unnecessary.");
    }
}
```

com.mysql.cj.jdbc.Driver的源代码：

```
package com.mysql.cj.jdbc;

import java.sql.DriverManager;
import java.sql.SQLException;

public class Driver extends NonRegisteringDriver implements java.sql.Driver {
    public Driver() throws SQLException {
    }

    static {
        try {
            DriverManager.registerDriver(new Driver());
        } catch (SQLException var1) {
            throw new RuntimeException("Can't register driver!");
        }
    }
}
```

也就是说，在Class.forName加载完驱动类后，开始执行静态代码块时，会new一个Driver，并调用DriverManager的registerDriver把Driver给注册到自己的驱动程序管理器中。

ClassLoader.loadClass和Class.forName的区别

- `Classloader.loaderClass(String name)`

> 起始该方法内部调用的是:`Classloader.loadClass(name, false)`
>
> 方法：Classloader.loadClass(String name, boolean resolve)
>
> 1.参数name代表类的全限定类名
>
> 2.参数resolve代表是否解析

- Class.forName(String name)

> 其实该方法内部调用的是:`Class.forName(className, true,ClassLoader.getClassLoader(caller))`
>
> 方法：Class.forName(String name, boolean initialize, ClassLoader loader)
>
> 1.参数name代表全限定类名
>
> 2.参数initialize表示是否初始化
>
> 3.参数loader对应的类加载器

- 两者最大的区别

> Class.forName得到的class是已经初始化完成的
>
> Classloader.loaderClass得到的class是没有初始化的

由上述源码可以看见Driver类还继承了NonRegisteringDriver，而NonRegisteringDriver还实现了java.sql.Driver接口

java.sql.Driver接口源码：

```
package java.sql;

import java.util.logging.Logger;

public interface Driver {

    //获取Connection 方法。数据库的url，及info至少得包含user，password key
    Connection connect(String url, java.util.Properties info)
        throws SQLException;

    //判断是否是一个正确的url字符串。
    boolean acceptsURL(String url) throws SQLException;

    //得到驱动的属性(user，password，port等)。
    DriverPropertyInfo[] getPropertyInfo(String url, java.util.Properties info)
throws SQLException;

    //得到主要版本
    int getMajorVersion();

    //得到次要版本
    int getMinorVersion();

    //判断是否是一个正确的driver
    boolean jdbcCompliant();
```

```
    //------------------------ JDBC 4.1 -----------------------------------
    //返回父日志
    public Logger getParentLogger() throws SQLFeatureNotSupportedException;
}
```

Driver接口是每个数据库驱动都必须继承的接口。

继续com.mysql.cj.jdbc.Driver类

在Driver类中的静态代码块有DriverManager类(这是把Driver类注册到自己的驱动程序管理器中)的方法，此类没有继承和实现任何接口，它是管理一组JDBC驱动程序的基本服务。

DriverManager中有很多重要方法可以了解一下

二、获取数据库连接Connection

代表与数据库连接，并拥有创建SQL语句的方法，已完成基本的SQL操作，同时为数据库事务提供提交和回滚的方法

我以注释方法来说一下1Connection接口的重要方法：

```
package java.sql;

import java.util.Properties;
import java.util.concurrent.Executor;‘

public interface Connection  extends Wrapper, AutoCloseable {
    //创建statement
    Statement createStatement() throws SQLException;
    //创建prepareStatement
    PreparedStatement prepareStatement(String sql)
        throws SQLException;
    //创建CallableStatement
    CallableStatement prepareCall(String sql) throws SQLException;
    //转换sql为本机执行sql
    String nativeSQL(String sql) throws SQLException;
    //设置是否自动提交 状态
    void setAutoCommit(boolean autoCommit) throws SQLException;
    //判断是否是自动提交
    boolean getAutoCommit() throws SQLException;
    //提交
    void commit() throws SQLException;
    //回滚
    void rollback() throws SQLException;
    //关闭连接
    void close() throws SQLException;
    //判断是否关闭
    boolean isClosed() throws SQLException;
    //...
}
```

三、获取Statement、preparedstatement

两者都是用来执行sql的，但也有区别：

- PreparedStatement在使用时只需要编译一次，就可以运行多次，Statement每运行一次就编译一次，所以PreparedStatement的效率更高
- PreparedStatement需要的sql语句为用?(占位符)来替换值，Statement所需要的sql语句为字符串拼接
- PreparedStatement解决了sql注入的问题，Statement没有解决，因为PreparedStatement有一个预编译的过程，就算传入占位符的数据中有sql关键字也都被认为是值。Statement所需要的是字符串拼接，传入的整个字符串被默认为sql语句，如果用户手动拼接了字符串，那么会导致语句的改变

总之就是在开发过程中用preparedStatement比statement更好，以下是我认为preparedStatement中一些重要的方法：

```
package java.sql;

import java.math.BigDecimal;
import java.util.Calendar;
import java.io.Reader;
import java.io.InputStream;

public interface PreparedStatement extends Statement {
    //用于产生单个结果集的语句，例如 SELECT 语句
    ResultSet executeQuery() throws SQLException;
    //用于执行 INSERT、UPDATE 或 DELETE 语句以及 SQL DDL（数据定义语言）语句
    int executeUpdate() throws SQLException;
    //设置空值，必须穿入type，不然可能报空指针异常
    void setNull(int parameterIndex, int sqlType) throws SQLException;
    ...（同理有很多set的方法）
    //清空属性
    void clearParameters() throws SQLException;
    //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
    boolean execute() throws SQLException;
    //...
}
```

四、结果集ResultSet

结果集ResultSet是数据中查询结果的一种返回对象，可以说结果集是一个储存查询结果的对象，但是结果集并不仅仅具有储存的功能，它同时还具有操纵数据的功能，可以完成对数据的更新等。以下是ResultSet的一些重要方法：

```
package java.sql;

import java.math.BigDecimal;
import java.util.Calendar;
import java.io.Reader;
import java.io.InputStream;

public interface ResultSet extends Wrapper, AutoCloseable {
    //是否有下一个值
    boolean next() throws SQLException;
    //关闭
```

```
    void close() throws SQLException;
    //是否为空
    boolean wasNull() throws SQLException;
    //得到第几列的String类型数据
    String getString(int columnIndex) throws SQLException;
    boolean getBoolean(int columnIndex) throws SQLException;
    ...(太多get方法不一一列举)
    //得到列名为columnLabel的值
    String getString(String columnLabel) throws SQLException;

    //更新第几列为空（各种update方法）
    void updateNull(int columnIndex) throws SQLException;
    //插入（updateRow、deleteRow、refreshRow 等）
    void insertRow() throws SQLException;
    //...
}
```

以上就是JDBC中的主要的接口源码及个人解读、个人注释。