1. JDBC
2. 概述

Java DataBase Connectivity，从名称来看，JDBC是用于Java连接数据库的。JDBC的本质是Java定义的访问数据库的标准接口，Java并没有定义如何访问各种数据库，具体的访问各种数据库的实现类是由各大数据库厂商实现的，这些实现类称为数据库驱动，厂商一般都会提供相应的jar包下载。

1. JDBC的优势
2. 开发者只需面向接口编程，不用关注实现类。
3. 使用同一套Java代码，只需少量修改就可访问JDBC支持的数据库。
4. JDBC的核心API

DriverManager 类 1) 管理和注册数据库驱动

2) 得到数据库连接对象

Connection 接口 一个连接对象，可用于创建

Statement 和 PreparedStatement 对象

Statement 接口 一个 SQL 语句对象，用于将 SQL 语句发送给数据库服务器。

PreparedStatemen 接口 一个 SQL 语句对象，是 Statement 的子接口

ResultSet 接口 用于封装数据库查询的结果集，返回给客户端 Java 程序

1. DriverManager类
2. 功能：

（1）管理和注册驱动

（2）获得数据库连接对象

1. 注册驱动

（1）导入数据库驱动jar包

首先我们需要导入数据库驱动jar包，在项目下设置一个Lib文件夹，并将它右键Add as Library，设置为库。

（2）注册驱动

告诉了程序该使用什么数据库驱动jar包。首先要注册驱动，通过下列语句实现：

Class.forName( 数据库驱动实现类)

加载和注册数据库驱动，数据库驱动由 mysql 厂商提供（com.mysql.jdbc.Driver）

Class.forName("com.mysql.jdbc.Driver");实现注册mysql驱动。（mysql8.0之后的注册驱动为com.mysql.cj.jdbc.Driver）

深入com.mysql.jdbc.Driver源码查看实现，

try {

DriverManager.registerDriver(new Driver()); //注册数据库驱动

} catch (SQLException var1) {

throw new RuntimeException("Can't register driver!");

}

可以看到，注册驱动调用的是DriverManager类中的registerDriver方法。

注意：mysql5之后的驱动jar包可以省略注册驱动步骤，但是建议写上。

1. 获取数据库连接对象
2. 方法

静态方法 描述

Connection getConnection (String url, String user, String password)

通过连接字符串，用户名，密码来得到数据

库的连接对象

Connection getConnection (String url, Properties info)

通过连接字符串，属性对象来到连接对象

1. URL的地址格式

协议名: 子协议:// 服务器名或 IP 地址: 端口号/ 数据库名? 参数= 参数值

如：

Jdbc:mysql://localhost:3306/数据库?参数

如连接本地服务器，且端口号为3306，则可省略localhost:3306

1. 连接数据库测试

package JDBC;  
  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
public class test {  
 public static void main(String[] args) throws IOException, SQLException {  
  
 String url = "jdbc:mysql://localhost:3306/h";  
 Connection con = DriverManager.*getConnection*(url, "root", " ");  
 System.*out*.println(con);  
 }  
}

注意，可能会出现TIME\_ZONE问题，这是数据库的系统时间设置，需要改为东八区”+8:00”。

三，Connection接口

Connection接口的具体实现类由厂商实现，代表一个连接对象。

1. 获取执行SQL的对象

方法：返回一个可以执行sql语句的对象。

Statement createStatement()

preparedStatement prepareStatement(String sql)

1. 管理事务
2. 开启事务

setAutoCommit(Boolean autoCommit)

该方法用于设置自动提交，参数设为false即为手动提交，即手动开启事务。

1. 提交事务

commit()

1. 回滚事务

rollback()

* 1. Statement接口

（一）作用

Statement对象为一个语句对象，用于发送SQL语句给服务器，用于执行静态的SQL语句并返回他所生成结果的对象。

（二）方法：

boolean execute(String sql) 执行任意的SQL语句

int executeUpdate(String sql) 执行DML（insert, update, delete）语句，DDL（create, alter, drop）语句，返回的是该语句影响的行数，通过这个行数判断SQL语句是否执行成功，行数大于0则成功。

ResultSet executeQuery(String sql) 执行DQL（select）语句

* 1. ResultSet接口

1. 作用

封装数据库的查询结果集，用于对结果集进行遍历，取出每一条数据。

1. 使用

ResultSet结果集使用游标进行遍历，游标按照行进行遍历的，每次游标指向一行。一开始的游标指向第一条记录的上面，要获取第一条记录，我们需要将游标下移一行。

1. 方法

boolean next() 游标下移一行，返回boolean，如有下一条记录返回true

数据类型getxx(String/ int) xx是数据类型，参数可以为字段名getString(“name”)或者是列号getString(1)，列号从1开始

如： double getDouble(String columnLabel)

double getDouble(Int columnNumber)

1. 遍历ResultSet

用游标的next()函数的返回值判断是否是最后一行，可以不断循环知道最后一行。

package JDBC;  
  
import java.sql.\*;  
  
public class ResultSet使用 {  
 public static void main(String[] args) {  
 Connection con = null;  
 Statement stm = null;  
 ResultSet rs = null;  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 con = DriverManager.*getConnection*("jdbc:mysql:///heima", "root", " ");  
 String sql = "select \* from student";  
 stm = con.createStatement();  
 rs = stm.executeQuery(sql);  
 while(rs.next()){  
 int id = rs.getInt(1);  
 String name = rs.getString("name");  
 int age = rs.getInt("age");  
 System.*out*.println(name + "的id是" + id + " 年龄是" + age );  
 }  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }finally {  
 if(rs != null){  
 try {  
 rs.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(stm != null){  
 try {  
 stm.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(con != null){  
 try {  
 con.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
}

* 1. PreparedStatement接口

1. 概述

表示预编译的SQL语句对象，SQL已经预编译并存储在PreparedStatement对象中。使用该对象可以解决SQL的注入问题。PreparedStatement对象会先将SQL语句发送给数据库预编译，然后该对象引用预编译后的结果，可以多次传入不同的参数给PreparedStatement对象并执行，减少了SQL编译次数，提高了效率。

1. SQL注入问题

在拼接SQL时，有一些SQL的特殊关键字参与字符串的拼接中，会造成意想不到的安全问题。

1. 使用PreparedStatement
2. ？占位符

在定义SQL语句时，使用？占位符作为SQL的参数，如：

Select \* from user where username=? and password=?;

1. Connection.preparedStatement(String sql)

获取该对象时要传入SQL语句

1. 给？占位符赋值

方法：setxx(参数1，参数2)

Xx为参数的类型；

参数1：？占位符的位置编号，从1开始，即第几个？

参数2：？占位符的值

如：

Sql语句为： select \* from user where username=? and password=?

赋值语句为：  
setString(1, username);

setString(2, password);

1. PreparedStatment对象执行SQL语句不需要传参
   1. 释放资源
2. 需要释放的对象

ResultSet结果集，Statement语句，Connection连接

1. 释放顺序

先创建的后关闭，后创建的先关闭。

ResultSet -> Statement -> Connection

1. 位置

释放资源是必须做的，因此不能放在try代码段中，因为可能有不执行的风险，我们需要将该代码放在finally代码块中。

八． JDBC表中插入数据

package JDBC;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
public class InsertTest {  
 public static void main(String[] args) {  
 Statement stm = null;  
 Connection con = null;  
  
 try {  
 //1.注册驱动  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 //2.定义SQL  
 String sql = "insert into student(id,name,age) values(14,'dd',24)";  
 //3.获取Connection对象  
 con = DriverManager.*getConnection*("jdbc:mysql:///heima", "root", " ");  
 //4.获取执行SQL的对象Statement  
 stm = con.createStatement();  
 //5.执行SQL  
 int count = stm.executeUpdate(sql);  
 //6.处理结果  
 System.*out*.println(count);  
 if(count > 0){  
 System.*out*.println("添加成功！");  
 }  
 else{  
 System.*out*.println("添加失败！");  
 }  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }finally {  
 //释放资源  
 //判断空指针异常  
 if(stm != null){  
 try {  
 stm.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(con != null){  
 try {  
 con.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 }  
}

注意：SQL语句中的字段名称没有带引号。

九． JDBC的工具类

1. 创建工具类

用JDBC每次连接数据库都要重复一些步骤 ，为了简化流程，我们可以将一些功能封装成一个工具类，在不同的地方复用。

在使用JDBC的模块下创建一个类：JDBCUtils

1. 工具类的方法
2. 抽取一个方法获取连接对象
3. 抽取一个方法释放资源
4. 使用配置文件

使用配置文件，提高工具类的通用性。将Driver，User，Password，Url等可能会变化的字符串写到配置文件中，可以省去在代码中修改的麻烦。

1. 使用静态代码块读取配置文件

静态代码块读取配置文件，只需读取一次即可。

1. 使用类加载器获取配置文件路径

类加载器可以用于找到配置文件的文件路径，通过配置对象的加载将配置文件加载到配置对象中，进而导出文件中的属性值。

1. 工具类代码示例

package Utils;  
  
import java.io.FileReader;  
import java.io.IOException;  
import java.net.URL;  
import java.sql.\*;  
import java.util.Properties;  
  
public class JDBCUtils {  
  
 private static String *url*;  
 private static String *user*;  
 private static String *password*;  
 private static String *driver*;  
  
 static{  
 //读取配置文件  
 //创建Properties集合类  
 Properties pro = new Properties();  
 //加载文件  
 //通过名称找到该配置文件  
 //类加载器  
 ClassLoader classLoader = JDBCUtils.class.getClassLoader();  
 //类加载器找资源  
 URL res = classLoader.getResource("jdbc.properties");  
 //获取该资源的绝对路径  
 String path = res.getPath();  
  
 try {  
 pro.load(new FileReader(path));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
   
 //获取数据  
 *url* = pro.getProperty("url");  
 *user* = pro.getProperty("user");  
 *password* = pro.getProperty("password");  
 *driver* = pro.getProperty("driver");  
 //注册驱动  
 try {  
 Class.*forName*(*driver*);  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* 获取连接  
 \** ***@return*** *连接对象  
 \*/* public static Connection getConnection() throws SQLException {  
  
 return DriverManager.*getConnection*(*url*, *user*, *password*);  
  
 }  
  
 */\*\*  
 \* 释放资源  
 \*/* public static void close(Statement stm, Connection con){  
 if(stm != null){  
 try {  
 stm.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(con != null){  
 try {  
 con.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public static void close(ResultSet rs, Statement stm, Connection con){  
 if(rs != null){  
 try {  
 rs.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(stm != null){  
 try {  
 stm.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(con != null){  
 try {  
 con.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

* 1. JDBC管理事务

1. 开启事务

使用setAutoCommit(Boolean autoCommit)开启事务，在执行SQL语句之前开启事务。

1. 提交事务

执行完所有SQL语句后提交事务。

1. 回滚事务

用try-catch-语句将事务包起来，try代码段中写执行SQL语句，catch代码段中写回滚事务rollback()。当出现异常时，catch异常代码块中会执行事务的回滚，将事务状态回滚到之前的状态。

1. 代码实现

package JDBC;  
  
import Utils.JDBCUtils;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
  
public class Transaction {  
 public static void main(String[] args) {  
 Connection connecton = null;  
 PreparedStatement ps = null;  
 try{  
 //连接  
 connecton = JDBCUtils.*getConnection*();  
 //开启事务  
 connecton.setAutoCommit(false);  
 String sql = "update student set age = age - ? where name = ?";  
 //执行语句对象  
 ps = connecton.prepareStatement(sql);  
 //设置值  
 ps.setInt(1, 2);  
 ps.setString(2, "xx");  
 //更新  
 ps.executeUpdate();  
 //设置异常  
 System.*out*.println(100 / 0);  
 //提交事务  
 connecton.commit();  
 System.*out*.println("修改成功");  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 try {  
 connecton.rollback();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 System.*out*.println("修改失败");  
 }finally {  
 //关闭资源  
 JDBCUtils.*close*(ps, connecton);  
 }  
 }  
}