第07天 JDBC

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【理解】JDBC的概述及入门案例**
2. 【应用】阐述JDBC连接数据库的原理
3. 【应用】独立完成JDBC入门案例的准备工作
4. 【应用】独立完成JDBC入门案例
5. **【应用】JDBC的API详解**
6. 【应用】阐述注册驱动代码的原理及注意事项
7. 【应用】独立编写代码获得连接、获得语句执行者对象、结果集对象
8. 【应用】独立编写代码JDBC资源的释放
9. **【应用】JDBC的API详解**
10. 【应用】独立编写JDBC工具类
11. 【应用】独立编写代码使用JDBC工具类完成增删改查的操作
12. **【应用】JDBC的API详解**
13. 【应用】独立编写代码演示SQL注入的问题
14. 【应用】独立编写代码使用预处理对象完成数据的增删改查操作
15. 【应用】独立代码使用预处理对象解决SQL注入问题

# 第1章JDBC的概述及入门案例

## JDBC概述和原理

### JDBC概述

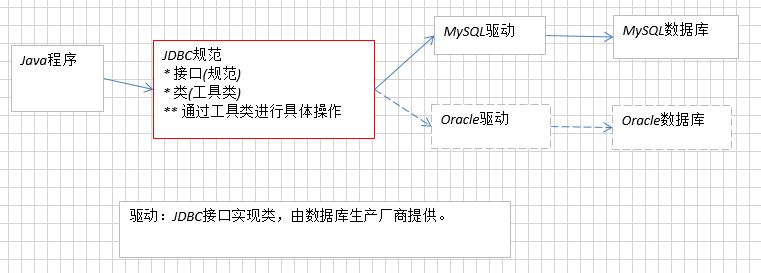
JDBC（Java Data Base Connectivity,java数据库连接）是一种用于执行SQL语句的Java API。JDBC是Java访问数据库的标准规范，可以为不同的关系型数据库提供统一访问，它由一组用Java语言编写的接口和类组成。

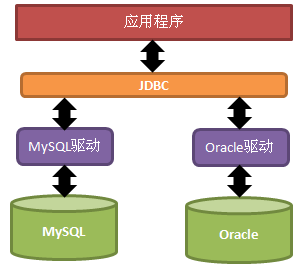
JDBC需要连接驱动，驱动是两个设备要进行通信，满足一定通信数据格式，数据格式由设备提供商规定，设备提供商为设备提供驱动软件，通过软件可以与该设备进行通信。

今天我们使用的是mysql的驱动mysql-connector-java-5.1.37-bin.jar

### JDBC原理

Java提供访问数据库规范称为JDBC，而生产厂商提供规范的实现类称为驱动。





JDBC是接口，驱动是接口的实现，没有驱动将无法完成数据库连接，从而不能操作数据库！每个数据库厂商都需要提供自己的驱动，用来连接自己公司的数据库，也就是说驱动一般都由数据库生成厂商提供。

## JDBC入门案例—准备工作

### 准备数据

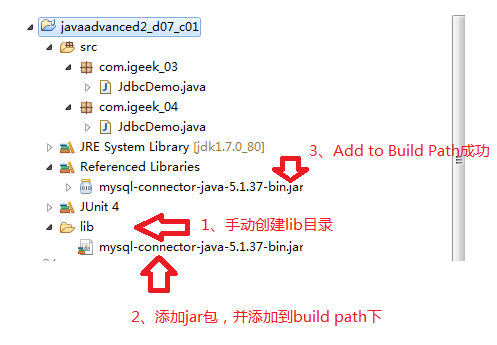
之前我们学习了sql语句的使用，并创建的分类表category，今天我们将使用JDBC对分类表进行增删改查操作。

|  |
| --- |
| #创建数据库  create database mydb;  #使用数据库  use mydb;  ###创建分类表  create table category(  cid int PRIMARY KEY AUTO\_INCREMENT ,  cname varchar(100)  );  #初始化数据  insert into category (cname) values('家电');  insert into category (cname) values('服饰');  insert into category (cname) values('化妆品'); |

### 1.2.2导入驱动jar包

创建lib目录，用于存放当前项目需要的所有jar包

选择jar包，右键执行build path / Add to Build Path



## JDBC入门案例—案例实现

### 开发步骤

1. 注册驱动.
2. 获得连接.
3. 获得语句执行平台
4. 执行sql语句
5. 处理结果
6. 释放资源.

### 案例代码一

|  |
| --- |
| **package** com.igeek\_03;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: JdbcDemo  \* **@Description**: JDBC入门案例  \* **@date** 2018年2月7日 上午10:25:36  \* Company www.igeekhome.com  \*  \*/  **public** **class** JdbcDemo {    /\*\*  \* **@Title**: demo01  \* **@Description**: 了解JDBC开发步骤  \* **@throws** Exception  \*/  @Test  **public** **void** demo01() **throws** Exception{  // 查询所有的分类信息  // 注意：使用JDBC规范，采用都是 java.sql包下的内容    //1 注册驱动  Class.*forName*("com.mysql.jdbc.Driver");    //2 获得连接  Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mydb", "root", "niit");    //3获得语句执行者  Statement st = conn.createStatement();    //4执行SQL语句  ResultSet rs = st.executeQuery("select \* from category");    //5处理结果集  **while**(rs.next()){  // 获得一行数据  Integer cid = rs.getInt("cid");  String cname = rs.getString("cname");    System.***out***.println(cid + " , " + cname);  }  //6释放资源  rs.close();  st.close();  conn.close();  }  } |

# JDBC的API详解

## JDBC API详解--注册驱动

### 注册驱动

|  |
| --- |
| 代码：Class.forName("com.mysql.jdbc.Driver"); |

JDBC规范定义驱动接口：java.sql.Driver，MySql驱动包提供了实现类： com.mysql.jdbc.Driver

DriverManager工具类，提供注册驱动的方法 registerDriver()，方法的参数是java.sql.Driver，所以我们可以通过如下语句进行注册:

DriverManager.registerDriver(new com.mysql.jdbc.Driver());

以上代码不推荐使用，存在两方面不足

1. 硬编码，后期不易于程序扩展和维护
2. 驱动被注册两次。

通常开发我们使用Class.forName() 加载一个使用字符串描述的驱动类。

如果使用Class.forName()将类加载到内存，该类的静态代码将自动执行。

通过查询com.mysql.jdbc.Driver源码，我们发现Driver类“主动”将自己进行注册

|  |
| --- |
| **public** **class** Driver **extends** NonRegisteringDriver **implements** java.sql.Driver{  **static** {  **try** {  java.sql.DriverManager.*registerDriver*(**new** Driver());  } **catch** (SQLException E) {  **throw** **new** RuntimeException("Can't register driver!");  }  }  ......  } |

### 案例代码二

|  |
| --- |
| **package** com.igeek\_01;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: ApiDemo  \* **@Description**: JDBC API详解--注解驱动  \* **@date** 2018年2月7日 上午10:59:10  \* Company www.igeekhome.com  \*  \*/  **public** **class** ApiDemo {  /\*\*  \* **@Title**: demo01  \* **@Description**: 测试注解驱动  \* **@throws** Exception  \*/  @Test  **public** **void** demo01() **throws** Exception{  /\* 获得驱动  \* 1. JDBC规范规定，如果需要连接数据库，必须提供驱动接口的实现类  \* 驱动接口：java.sql.Driver  \* 每一个数据库提供驱动jar 都实现该接口  \* 2. MySQL 提供实现类：com.mysql.jdbc.Driver  \* 源码：public class com.mysql.jdbc.Driver implements java.sql.Driver {  \* 3. JDBC规范提供了，注册实现方式  \* DriverManager.registerDriver( new com.mysql.jdbc.Driver() );  \* 但，如果遵循上面语句，Java代码与 mysql实现类耦合(直接关系)，之后切换数据库将不能进行。  \* 希望提供方案时，只要切换数据驱动，就可以切换使用数据库  \* 4. 通常注册驱动使用标准写法  \* Class.forName("com.mysql.jdbc.Driver")  \* 1) 使用反射的方式加载指定的类  \* 2) 具体加载的类以字符串(类的全限定名称)体现，内容就可以存放到配置文件中，通过修改配置文件方便切换数据库  \* 3) 一个类被加载到内存，静态代码块将执行，static{ ... }  \* 4) com.mysql.jdbc.Driver 源码分析  \* static{  \* java.sql.DriverManager.registerDriver(new Driver());  \*/  //结论：注册驱动  Class.*forName*("com.mysql.jdbc.Driver");  }  } |

## 注册驱动--注意事项

注册驱动注意事项

手动注册驱动，驱动注册了几次？

DriverManager.registerDriver(new com.mysql.jdbc.Driver() );

注册了2次

第一次：new Driver() 时，Driver类加载，静态代码块执行，注册一次

第二次：手动注册

## JDBC API详解--获得连接

### 外链接概述

|  |
| --- |
| 代码：Connection conn = DriverManager.getConnection  ("jdbc:mysql://localhost:3306/mydb", "root", "niit"); |

获取连接需要方法 DriverManager.getConnection(url,username,password)，三个参数分别表示，url 需要连接数据库的位置（网址） user用户名 password 密码

url比较复杂，下面是mysql的url：

|  |
| --- |
| jdbc:mysql://localhost:3306/mydb  JDBC规定url的格式由三部分组成，每个部分中间使用冒号分隔。  第一部分是jdbc，这是固定的；  第二部分是数据库名称，那么连接mysql数据库，第二部分当然是mysql了； |

第三部分是由数据库厂商规定的，我们需要了解每个数据库厂商的要求，mysql的第三部分分别由数据库服务器的IP地址（localhost）、端口号（3306），以及DATABASE名称(mydb)组成。

### 案例代码三

|  |
| --- |
| **package** com.igeek\_03;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: ApiDemo  \* **@Description**: JDBC API详解--获取连接  \* **@date** 2018年2月7日 上午11:07:55  \* Company www.igeekhome.com  \*  \*/  **public** **class** ApiDemo {  /\*\*  \* **@Title**: demo01  \* **@Description**: 测试获取连接  \* **@throws** Exception  \*/  @Test  **public** **void** demo01() **throws** Exception{  /\* 获得连接  \* JDBC提供工具类 DriverManager(驱动管理器)  \* getConnection() 通过设置具体参数向不同的数据库创建新的连接  \* 参数1：url ，数据访问路径  \* 参数2：user ， 数据库用户名  \* 参数3：password ， 数据库密码  \* url访问路径  \* 格式 jdbc:mysql://ip地址:端口号/数据库名称  \* 例如 jdbc:mysql://localhost:3306/mydb  \*  \* jdbc固定  \* mysql 表示mysql数据库，一般情况会根据数据库不同而不同  \* localhost:3306 表示数据具体地址，为默认值，及可以省略  \* mydb 表示数据名称  \*  \* 路径简化版 jdbc:mysql:///mydb  \*  \*  \*/    Class.*forName*("com.mysql.jdbc.Driver");  //结论：获得连接  Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mydb", "root", "niit");  }  } |

## JDBC API详解--获得语句执行者

### 概述

|  |
| --- |
| String sql = "某SQL语句";  获取Statement语句执行平台：Statement stmt = con.createStatement(); |

常用方法：

* + int executeUpdate(String sql); --执行insert update delete语句.
  + ResultSet executeQuery(String sql); --执行select语句.
  + boolean execute(String sql); --执行select返回true 执行其他的语句返回false.

### 案例代码四

|  |
| --- |
| **package** com.igeek\_02\_api;  **package** com.igeek\_04;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: ApiDemo  \* **@Description**: JDBC API详解--获得语句执行者  \* **@date** 2018年2月7日 上午11:13:12  \* Company www.igeekhome.com  \*  \*/  **public** **class** ApiDemo {    /\*\*  \* **@Title**: demo01  \* **@Description**: 测试获得语句执行者  \* **@throws** Exception  \*/  @Test  **public** **void** demo01() **throws** Exception{  /\* 获得语句执行者  \* 通过Connection就可以获得针对不同数据库sql语句的执行对象，常用 createStatement()  \* 通过Statement对象可以执行任意的SQL语句  \* st.executeUpdate(sql) 执行DML语句(增删改 insert、delete、update) ，返回为整形，表示影响行数。  \* st.executeQuery(sql) 执行DQL语句(查询 select) ，返回ResultSet结果集对象(查询所有数据)  \*  \* st.execute(sql) 了解，可以执行任意sql语句。返回为boolean  \* true，表示执行DQL语句，需要通过ts.getResultSet() 获得查询结果  \* false，表示DML语句，需要通过 ts.getUpdateCount() 获得影响行数。  \*/    Class.*forName*("com.mysql.jdbc.Driver");  Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mydb", "root", "niit");  //结论：获得语句执行者  Statement st = conn.createStatement();  st.execute(**null**);  **int** executeUpdate = st.executeUpdate("");  ResultSet executeQuery = st.executeQuery("");  }  } |

## JDBC API详解--结果集对象

### 概述

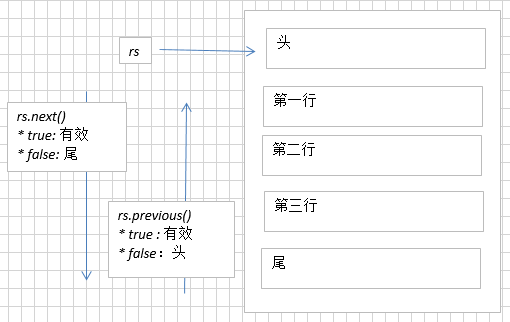
ResultSet实际上就是一张二维的表格，我们可以调用其boolean next()方法指向某行记录，当第一次调用next()方法时，便指向第一行记录的位置，这时就可以使用ResultSet提供的getXXX(int col)方法(与索引从0开始不同个，列从1开始)来获取指定列的数据：

|  |
| --- |
| rs.next();//指向第一行  rs.getInt(1);//获取第一行第一列的数据 |

常用方法：

* Object getObject(int index) / Object getObject(String name) 获得任意对象
* String getString(int index) / Object getObject(String name) 获得字符串
* int getInt(int index) / Object getObject(String name) 获得整形
* double getDouble(int index) / Object getObject(String name) 获得双精度浮点型

### 图解



### 案例代码五

|  |
| --- |
| **package** com.igeek\_05;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: ApiDemo  \* **@Description**: JDBC API详解--结果集对象  \* **@date** 2018年2月7日 上午11:20:15  \* Company www.igeekhome.com  \*  \*/  **public** **class** ApiDemo {    /\*\*  \* **@Title**: demo01  \* **@Description**: 测试结果集对象  \* **@throws** Exception  \*/  @Test  **public** **void** demo01() **throws** Exception{  /\* 获得语句执行者  \* 移动游标  \* rs.next(); 下一个  \* rs.previous(); 上一个  \* 获得指定列数据  \* rs.getXxx(String) ,通过字段名称获得内容  \* rs.getXxx(Integer) ,通过字段索引号获得内容  \*  \* 例如 ：  \* rs.getString("cname") 获得指定名称  \* rs.getDouble(1) 获得第二列  \*/    Class.*forName*("com.mysql.jdbc.Driver");  Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mydb", "root", "niit");  //结论：获得语句执行者  Statement st = conn.createStatement();  ResultSet rs = st.executeQuery("select \* from category");  **while**(rs.next()){  }  }  } |

## JDBC API详解--释放资源

与IO流一样，使用后的东西都需要关闭！关闭的顺序是先得到的后关闭，后得到的先关闭。

|  |
| --- |
| rs.close();  stmt.close();  con.close(); |

### 案例代码六

|  |
| --- |
| **package** com.igeek\_06;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: ApiDemo  \* **@Description**: JDBC API详解--释放资源  \* **@date** 2018年2月7日 上午11:25:14  \* Company www.igeekhome.com  \*  \*/  **public** **class** ApiDemo {    /\*\*  \* **@Title**: demo01  \* **@Description**: 测试释放资源  \*/  @Test  **public** **void** demo01(){  /\* 释放资源  \*/    Connection conn = **null**;  //结论：获得语句执行者  Statement st = **null**;  ResultSet rs = **null**;  **try** {  Class.*forName*("com.mysql.jdbc.Driver");  conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/mydb", "root", "niit");  st = conn.createStatement();  rs = st.executeQuery("select \* from category");  } **catch** (Exception e) {  e.printStackTrace();  } **finally**{    **if**(rs != **null**) {  **try** {  rs.close();  } **catch** (SQLException e) {  }  }    **if**(st != **null**) {  **try** {  st.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }      **if**(conn != **null**){  **try** {  conn.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  }  } |

# JDBC工具类

## JDBC工具类

“获得数据库连接”操作，将在以后的增删改查所有功能中都存在，可以封装工具类JDBCUtils。提供获取连接对象的方法，从而达到代码的重复利用。

该工具类提供方法：public static Connection getConn ()。

### 案例代码七

|  |
| --- |
| **package** com.igeek\_01;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;  /\*\*  \* **@ClassName**: JdbcUtils  \* **@Description**: JDBC工具类  \* **@date** 2018年2月7日 下午12:28:51  \* Company www.igeekhome.com  \*  \*/  **public** **class** JdbcUtils {  **private** **static** String *driver* = "com.mysql.jdbc.Driver";  **private** **static** String *url* = "jdbc:mysql://localhost:3306/mydb";  **private** **static** String *user* = "root";  **private** **static** String *password* = "niit";    **static**{  **try** {  //注册驱动  Class.*forName*(*driver*);  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  }    }    /\*\*  \* **@Title**: getConnection  \* **@Description**: 获得连接  \* **@return**  \* **@throws** SQLException  \*/  **public** **static** Connection getConnection() **throws** SQLException{  //获得连接  Connection conn = DriverManager.*getConnection*(*url*, *user*, *password*);  **return** conn;  }    /\*\*  \* **@Title**: closeResource  \* **@Description**: 释放资源  \* **@param** conn  \* **@param** st  \* **@param** rs  \*/  **public** **static** **void** closeResource(Connection conn , Statement st , ResultSet rs){    **if**(rs != **null**){  **try** {  rs.close();  } **catch** (SQLException e) {  }  }    **if**(st != **null**){  **try** {  st.close();  } **catch** (SQLException e) {  }  }  **if**(conn != **null**){  **try** {  conn.close();  } **catch** (SQLException e) {  }  }    }  } |

## JDBC增删改查操作

### 案例代码八

|  |
| --- |
| **package** com.igeek\_02;  **import** java.sql.Connection;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: CRUDDemo  \* **@Description**: JDBC增删改查操作  \* **@date** 2018年2月7日 下午12:36:31  \* Company www.igeekhome.com  \*  \*/  **public** **class** CRUDDemo {  /\*\*  \* **@Title**: demo00  \* **@Description**: 获得连接  \*/  @Test  **public** **void** demo00(){  //模板    Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;    **try** {  //1 获得连接  conn = JdbcUtils.*getConnection*();    //操作      } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  //释放资源  JdbcUtils.*closeResource*(conn, st, rs);  }    }      /\*\*  \* **@Title**: demo01  \* **@Description**: 新增操作  \*/  @Test  **public** **void** demo01(){  //添加    Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;    **try** {  //1 获得连接  conn = JdbcUtils.*getConnection*();    //操作  //1) 获得语句执行者  st = conn.createStatement();  //2) 执行sql语句  **int** r = st.executeUpdate("insert into category(cname) values('测试')");    //3) 处理结果  System.***out***.println(r);    } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  //释放资源  JdbcUtils.*closeResource*(conn, st, rs);  }    }      /\*\*  \* **@Title**: demo02  \* **@Description**: 修改操作  \*/  @Test  **public** **void** demo02(){  //修改  Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    st = conn.createStatement();  **int** r = st.executeUpdate("update category set cname='测试2' where cid = 4");  System.***out***.println(r);    } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, st, rs);  }    }      /\*\*  \* **@Title**: demo03  \* **@Description**: 删除操作  \*/  @Test  **public** **void** demo03(){  //删除  Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    //操作  st = conn.createStatement();  **int** r = st.executeUpdate("delete from category where cid = 4");  System.***out***.println(r);    } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, st, rs);  }    }      /\*\*  \* **@Title**: demo04  \* **@Description**: 根据id查询详情  \*/  @Test  **public** **void** demo04(){  //通过id查询详情  Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    //操作  st = conn.createStatement();  rs = st.executeQuery("select \* from category where cid = 1");    **if**(rs.next()){  String cid = rs.getString("cid");  String cname = rs.getString("cname");  System.***out***.println(cid + " @ " + cname );  } **else** {  System.***out***.println("没有数据");  }    } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, st, rs);  }  }    /\*\*  \* **@Title**: demo05  \* **@Description**: 查询所有数据  \*/  @Test  **public** **void** demo05(){  //查询所有  Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    //操作  st = conn.createStatement();  rs = st.executeQuery("select \* from category");    **while**(rs.next()){  String cid = rs.getString("cid");  String cname = rs.getString("cname");  System.***out***.println(cid + " @ " + cname );  }    } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, st, rs);  }  }  } |

# JDBC 预处理对象--解决SQL注入

## JDBC 预处理对象--SQL注入问题

### SQL注入概述

SQL注入：用户输入的内容作为了SQL语句语法的一部分，改变了原有SQL真正的意义。

假设有登录案例SQL语句如下**:**

|  |
| --- |
| SELECT \* FROM 用户表 WHERE NAME = 用户输入的用户名 AND PASSWORD = 用户输的密码; |

此时，当用户输入正确的账号与密码后，查询到了信息则让用户登录。但是当用户输入的账号为XXX 密码为：XXX’ OR ‘a’=’a时，则真正执行的代码变为：

|  |
| --- |
| SELECT \* FROM 用户表 WHERE NAME = ‘XXX’ AND PASSWORD =’ XXX’ OR ’a’=’a’; |

此时，上述查询语句时永远可以查询出结果的。那么用户就直接登录成功了，显然我们不希望看到这样的结果，这便是SQL注入问题。

为此，我们使用PreparedStatement来解决对应的问题

### 准备工作

|  |
| --- |
| #创建一个表  CREATE TABLE USER(  id VARCHAR(32) PRIMARY KEY,  username VARCHAR(50),  PASSWORD VARCHAR(32)  );  INSERT INTO USER(id,username,PASSWORD) VALUES('u001','jack','1234');  INSERT INTO USER(id,username,PASSWORD) VALUES('u002','rose','1234'); |

### 案例代码九

|  |
| --- |
| **package** com.igeek\_01;  **import** java.sql.Connection;  **import** java.sql.ResultSet;  **import** java.sql.Statement;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: PrepareDemo  \* **@Description**: JDBC 预处理对象--SQL注入问题  \* **@date** 2018年2月7日 下午12:48:37  \* Company www.igeekhome.com  \*  \*/  **public** **class** PrepareDemo {    /\*\*  \* **@Title**: demo01  \* **@Description**: 模拟用户登录  \*/  @Test  **public** **void** demo01(){  //#模拟用户登录  String username = "jack";  String password = "1234";    Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;  **try** {  conn = JdbcUtils.*getConnection*();    st = conn.createStatement();  rs = st.executeQuery("select \* from user where username='"+username+"' and password='"+password+"'");    **if**(rs.next()){  System.***out***.println("用户登录");  } **else** {  System.***out***.println("不能登录");  }      } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, st, rs);  }    }      /\*\*  \* **@Title**: demo02  \* **@Description**: 模拟用户不能登录  \*/  @Test  **public** **void** demo02(){  //#演示SQL注入  String username = "jack";  String password = "12345";    Connection conn = **null**;  Statement st = **null**;  ResultSet rs = **null**;  **try** {  conn = JdbcUtils.*getConnection*();    st = conn.createStatement();  String sql = "select \* from user where username='"+username+"' and password='"+password+"'";  System.***out***.println(sql);  rs = st.executeQuery(sql);    **if**(rs.next()){  System.***out***.println("用户登录");  } **else** {  System.***out***.println("不能登录");  }      } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, st, rs);  }    }  } |

## JDBC 预处理对象--API详解

PreparedStatement预处理对象，处理的每条sql语句中所有的实际参数，都必须使用占位符?替换。

String sql = "select \* from user where username = ? and password = ?";

PreparedStatement使用，需要通过以下3步骤完成：

1. PreparedStatement预处理对象代码：

#获得预处理对象，需要提供已经使用占位符处理后的SQL语句

PreparedStatement psmt = conn.prepareStatement(sql)

1. 设置实际参数

void setXxx(int index, Xxx xx) 将指定参数设置指定类型的值

参数1：index 实际参数序列号，从1开始。

参数2：xxx 实际参数值，xxx表示具体的类型。

例如：

setString(2, "1234") 把SQL语句中第2个位置的占位符?替换成实际参数 "1234"

1. 执行SQL语句:

int executeUpdate(); --执行insert update delete语句.

ResultSet executeQuery(); --执行select语句.

boolean execute(); --执行select返回true 执行其他的语句返回false.

## JDBC 预处理对象--增删改查

使用PreparedStatement对象完成数据库的增删改查

### 案例代码十

|  |
| --- |
| **package** com.igeek\_03;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: PrepareDemo  \* **@Description**: JDBC 预处理对象--增删改查  \* **@date** 2018年2月7日 下午12:56:11  \* Company www.igeekhome.com  \*  \*/  **public** **class** PrepareDemo {    /\*\*  \* **@Title**: demo01  \* **@Description**: 新增操作  \*/  @Test  **public** **void** demo01(){  //添加：向分类表中添加数据  Connection conn = **null**;  PreparedStatement psmt = **null**;  ResultSet rs = **null**;    **try** {  //1 获得连接  conn = JdbcUtils.*getConnection*();  //2 处理sql语句  String sql = "insert into category(cname) values(?)";  //3获得预处理对象  psmt = conn.prepareStatement(sql);  //4设置实际参数  psmt.setString(1,"预处理");  //5执行  **int** r = psmt.executeUpdate();    System.***out***.println(r);    } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  //6释放资源  JdbcUtils.*closeResource*(conn, psmt, rs);  }      }    /\*\*  \* **@Title**: demo02  \* **@Description**: 修改操作  \*/  @Test  **public** **void** demo02(){  //修改  Connection conn = **null**;  PreparedStatement psmt = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    //1 sql语句  String sql = "update category set cname = ? where cid = ?";  //2 获得预处理对象  psmt = conn.prepareStatement(sql);  //3设置实际参数  psmt.setString(1, "测试数据");  psmt.setInt(2, 4);  //4执行  **int** r = psmt.executeUpdate();  System.***out***.println(r);      } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, psmt, rs);  }      }      /\*\*  \* **@Title**: demo03  \* **@Description**: 删除操作  \*/  @Test  **public** **void** demo03(){  //删除  Connection conn = **null**;  PreparedStatement psmt = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    //1 sql语句  String sql = "delete from category where cid = ?";  //2 获得预处理对象  psmt = conn.prepareStatement(sql);  //3设置实际参数  psmt.setInt(1, 5);  //4执行  **int** r = psmt.executeUpdate();  System.***out***.println(r);      } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, psmt, rs);  }  }      /\*\*  \* **@Title**: demo04  \* **@Description**: 查询所有数据  \*/  @Test  **public** **void** demo04(){  //查询所有  Connection conn = **null**;  PreparedStatement psmt = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    String sql = "select \* from category";  psmt = conn.prepareStatement(sql);  rs = psmt.executeQuery();  **while**(rs.next()){  String cname = rs.getString("cname");  System.***out***.println(cname);  }        } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, psmt, rs);  }  }    /\*\*  \* **@Title**: demo05  \* **@Description**: 根据id查询详情  \*/  @Test  **public** **void** demo05(){  //通过id查询  Connection conn = **null**;  PreparedStatement psmt = **null**;  ResultSet rs = **null**;    **try** {  conn = JdbcUtils.*getConnection*();    String sql = "select \* from category where cid = ?";  psmt = conn.prepareStatement(sql);  psmt.setInt(1, 2);  rs = psmt.executeQuery();  **if**(rs.next()){  System.***out***.println("查询到");  } **else** {  System.***out***.println("查询不到");  }      } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, psmt, rs);  }  }  } |

## JDBC 预处理对象--解决SQL注入

使用PreparedStatement对象解决SQL注入问题.

### 案例代码十一

|  |
| --- |
| **package** com.igeek\_04;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** org.junit.Test;  /\*\*  \* **@ClassName**: PrepareDemo  \* **@Description**: JDBC 预处理对象--解决SQL注入  \* **@date** 2018年2月7日 下午1:05:24  \* Company www.igeekhome.com  \*  \*/  **public** **class** PrepareDemo {    /\*\*  \* **@Title**: demo02  \* **@Description**: 模拟用户登录  \*/  @Test  **public** **void** demo02(){  //#解决SQL注入  String username = "jack";  String password = "1234";    Connection conn = **null**;  PreparedStatement psmt = **null**;  ResultSet rs = **null**;  **try** {  conn = JdbcUtils.*getConnection*();    //1 准备sql语句  String sql = "select \* from user where username=? and password=?";  //2 获得预处理对象  psmt = conn.prepareStatement(sql);  //3设置实际参数  psmt.setString(1, username);  psmt.setString(2, password);  //4执行sql语句 , 注意：没有实际参数  rs = psmt.executeQuery();    **if**(rs.next()){  System.***out***.println("用户登录");  } **else** {  System.***out***.println("不能登录");  }      } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally**{  JdbcUtils.*closeResource*(conn, psmt, rs);  }  }  } |

重点和总结

1、理解JDBC的概念和原理

2、熟练掌握JDBC的开发步骤

3、熟练掌握JDBC的API

4、熟练掌握JDBC的工具类

5、熟练掌握预处理对象