**JDBC**

# 回顾

## 表连接

问：笛卡尔积的记录是多少条？

左表中行数x右表中行数

## 表连接的分类

|  |  |  |
| --- | --- | --- |
| **表连接** | **分类** | **语法** |
| **内连接** | 显式 | select \* from 左表 inner join 右表 on 条件 |
| 隐式 | select \* from 左表,右表 where 条件 |
| **外连接** | 左连接 | select \* from 左表 left join 右表 on 条件 |
| 右连接 | select \* from 左表 right join 右表 on 条件 |

## 子查询的三种情况

|  |  |
| --- | --- |
| **子查询的结果** | **使用运算符** |
| **单行单列(一个值)** | 比较 > < = |
| **多行单列(多个值)** | in () |
| **多行多列(一个表)** | 做成一些虚拟表再次查询，给个别名 |

## 手动提交事务

|  |  |
| --- | --- |
| **功能** | **MySQL语句** |
| **开启事务** | **start transaction** |
| **提交事务** | **commit** |
| **回滚事务** | **rollback** |
| **设置回滚点** | **savepoint 名字** |
| **回到回滚点** | **rollback to 名字** |

## 事务的四大特性

|  |
| --- |
| **事务特性** |
| **原子性** |
| **一致性** |
| **隔离性** |
| **持久性** |

## 事务的并发访问会引发哪三种问题

|  |
| --- |
| **并发访问的问题** |
| **脏读** |
| **不可重复读** |
| **幻读** |

## 用户操作权限操作

|  |  |
| --- | --- |
| **操作语句** | **语法** |
| **给用户授权** | grant 权限 on 库.表 to '用户名'@'主机名' |
| **撤销授权** | revoke 权限 on 库.表 from '用户名'@'主机名' |

# 学习目标

1. 能够理解JDBC的概念
2. 能够使用DriverManager类
3. 能够使用Connection接口
4. 能够使用Statement接口
5. 能够使用ResultSet接口
6. 能够说出SQL注入原因和解决方案
7. 能够通过PreparedStatement完成增、删、改、查
8. 能够完成PreparedStatement改造登录案例

# JDBC入门

### 什么是JDBC

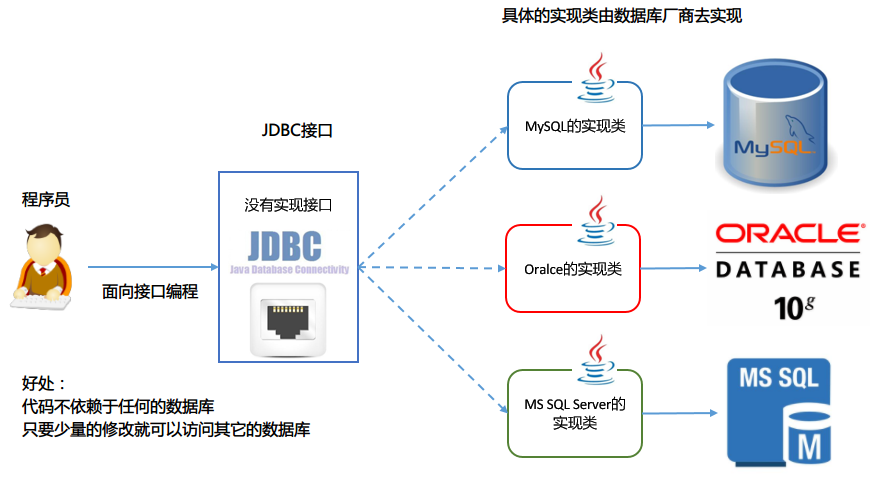
Java DataBase Connectivity Java数据库连接技术，用来访问所有的数据库，而且代码几乎一样。

**JDBC规范定义接口，具体的实现由各大数据库厂商来实现**。

JDBC是Java访问数据库的标准规范，真正怎么操作数据库还需要具体的实现类，也就是数据库驱动。每个数据库厂商根据自家数据库的通信格式编写好自己数据库的驱动。所以我们只需要会调用JDBC接口中的方法即可，数据库驱动由数据库厂商提供。

### 使用JDBC的好处：

1. 我们只需要会调用JDBC接口中的方法即可，使用简单
2. 使用同一套Java代码，进行少量的修改就可以访问其他JDBC支持的数据库



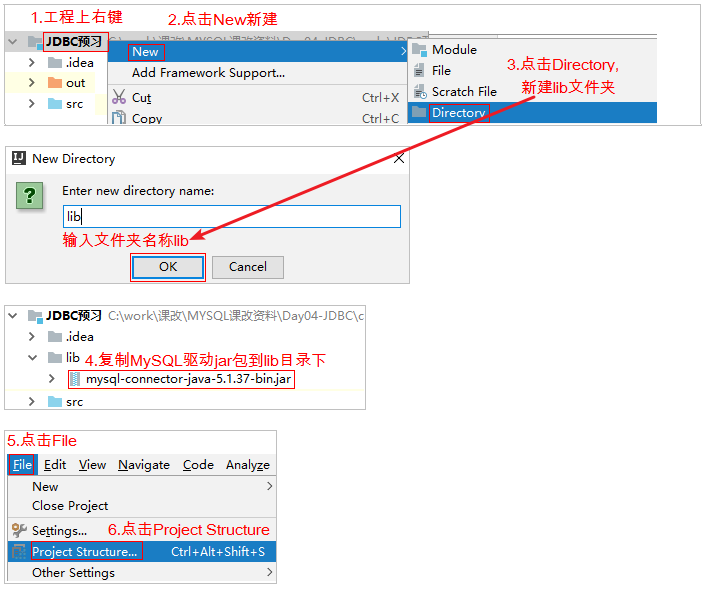
### 使用JDBC开发使用到的包：

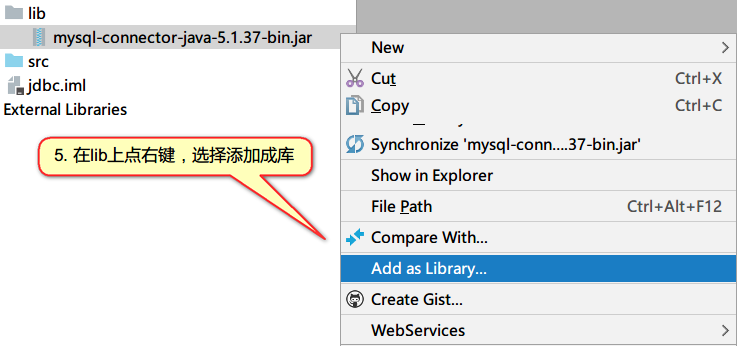
|  |  |
| --- | --- |
| **会使用到的包** | **说明** |
| **java.sql** | JDBC访问数据库的基本包，今天学习的内容 |
| **javax.sql** | 访问数据库的扩展名，JDBC高级特性。如：连接池。 |
| **数据库的驱动** | 由数据库厂商来实现。得到mysql驱动：com.mysql.jdbc.Driver |

## JDBC的核心API

|  |  |
| --- | --- |
| **接口或类** | **作用** |
| **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 |

## 导入驱动Jar包





## 加载和注册驱动

|  |  |
| --- | --- |
| **加载和注册驱动的方法** | **描述** |
| **Class.forName(数据库驱动实现类)** | 用来加载指定的驱动程序 |

* 疑问：为什么这样可以注册驱动？

因为Class.forName可以加载一个类，并且调用static代码块

* com.mysql.jdbc.Driver源代码

public class Driver extends NonRegisteringDriver implements java.sql.Driver {

public Driver() throws SQLException {

}

static {

try {

DriverManager.registerDriver(new Driver()); //注册驱动

} catch (SQLException var1) {

throw new RuntimeException("Can't register driver!"); //如果注册失败抛出异常

}

}

}

* + 通常情况下，第1步都应该加载和注册驱动，但JDBC3 以后的版本，注册驱动可以省略了。如果加上是为了兼容以前版本。

# DriverManager类

## DriverManager作用：

1. 注册驱动
2. 得到连接Connection对象

## 类中的方法：

|  |  |
| --- | --- |
| **DriverManager类中的静态方法** | **描述** |
| **Connection getConnection (String url, String user, String password)** | 指定URL、用户名、密码创建一个连接对象 |
| **Connection getConnection (String url, Properties info)** | 指定URL和属性集合得到一个连接对象 |

## 使用JDBC连接数据库的四个参数：

|  |  |
| --- | --- |
| **JDBC连接数据库的四个参数** | **说明** |
| **用户名** | root |
| **密码** | root |
| **连接字符串URL** | 客户端连接数据库服务器的地址，不同的数据库URL不同。 |
| **驱动类的字符串名** | com.mysql.jdbc.Driver |

## 连接数据库的URL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** |

### MySQL写法：

### MySQL中可以简写：

|  |
| --- |
| **jdbc:mysql:///数据库 前提：当访问本机数据库，而且端口号是3306默认的时候** |

### 乱码的处理

如果数据库出现乱码，可以指定参数: ?characterEncoding=utf8，表示让数据库以UTF-8编码来处理数据。

jdbc:mysql://localhost:3306/day24?characterEncoding=utf8

## 案例：得到MySQL的数据库连接对象

1. 使用用户名、密码、URL得到连接对象



可以看到具体的实现类是由mysql厂商去实现的。

1. 使用属性集合和url得到连接对象

package com.itheima;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.Properties;

public class Demo02Connection {

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

//1) 使用用户名、密码、URL得到连接对象

Connection con1 = DriverManager.getConnection("jdbc:mysql://localhost:3306/day24","root","root");

System.out.println("得到连接对象：" + con1);

//2) 使用属性集合和url得到连接对象

//创建一个属性

Properties info = new Properties();

info.setProperty("user","root");

info.setProperty("password","root");

Connection con2 = DriverManager.getConnection("jdbc:mysql:///day24",info);

System.out.println("得到连接对象：" + con2);

}

}

# Connection接口：

## Connection作用：

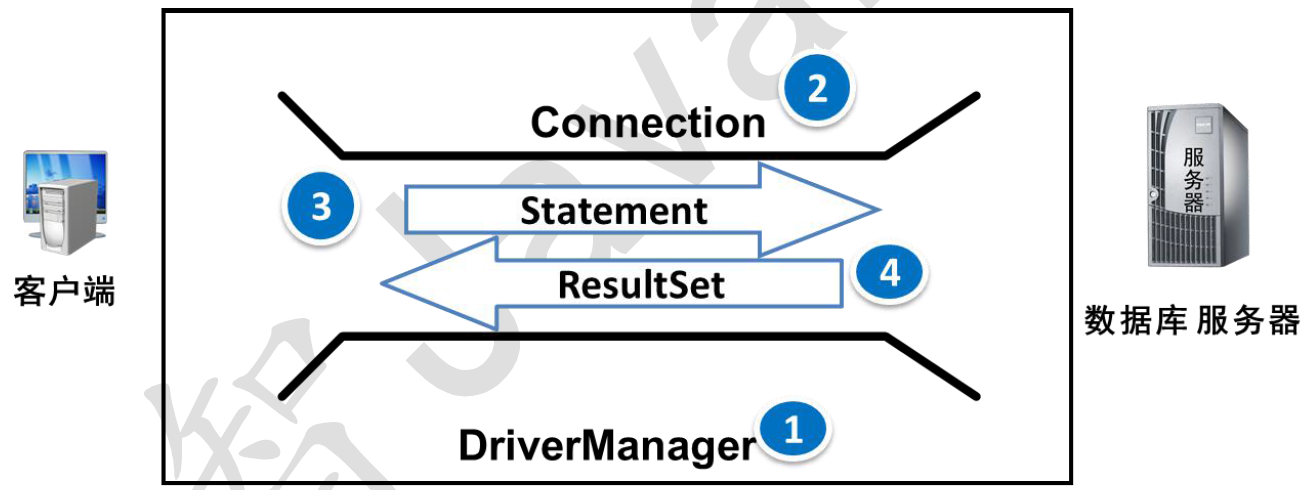
代表一个连接对象，每次访问数据库都需要创建一个连接对象。

## Connection方法：

|  |  |
| --- | --- |
| **Connection接口中的方法** | **描述** |
| **Statement createStatement()** | 通过连接对象创建一个SQL语句对象 |

# Statement接口

## JDBC访问数据库的步骤



1. 通过DriverManager创建一个连接对象Connection
2. 由连接对象创建一个SQL语句对象Statement
3. 将SQL语句发送给服务器
4. 在服务器上运行SQL语句，查询数据库中表。将查询到的结果发送给客户端，封装成一个ResultSet结果集。
5. 客户端得到结果集对象，从结果集中取出数据。
6. 释放资源，关闭连接对象。

## Statement接口作用：

创建好连接之后，发送SQL语句给服务器。使用Statement代表一个语句对象。

## Statement中的方法：

|  |  |
| --- | --- |
| **Statement接口中的方法** | **描述** |
| **int executeUpdate(String sql)** | 作用：用于增删改的操作  参数：SQL语句字符串  返回值：整数，对表中影响的行数 |
| **ResultSet executeQuery(String sql)** | 作用：用于查询数据的操作  参数：SQL语句  返回值：一个查询的结果集 |

## 释放资源

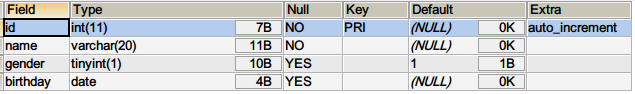
1. 需要释放的对象：Connection、Statement、ResultSet
2. 释放顺序：先开的后关，后开的先关。 关闭顺序：ResultSet 🡪 Statement 🡪 Connection
3. 关闭语句放在哪个代码块中：finally

## 执行DDL操作

如果使用exeuteUpdate执行DDL语句，返回的是0

* 需求：使用JDBC在MySQL的数据库中创建一张学生表

1. id是主键，整数类型，自增长
2. name是varchar(20)，非空
3. 性别是boolean类型
4. 生日是date类型



* 开发步骤：

1. 创建连接
2. 通过连接对象得到语句对象
3. 通过语句对象发送SQL语句给服务器，执行SQL
4. 通过语句对象发送SQL语句给服务器
5. 释放资源

* 代码：

package com.itheima;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

public class Demo03DDL {

public static void main(String[] args) {

Connection connection = null;

Statement statement = null;

try {

//1) 创建连接

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/day24","root","root");

//2) 通过连接对象得到语句对象

statement = connection.createStatement();

//3) 通过语句对象发送SQL语句给服务器，执行SQL

int row = statement.executeUpdate("CREATE TABLE Student (id int PRIMARY KEY auto\_increment, name VARCHAR(20) not null, gender boolean, birthday date)");

System.out.println(row);

//4) 释放资源

} catch (SQLException e) {

e.printStackTrace();

} finally {

if (statement!=null) {

try {

statement.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (connection!=null) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

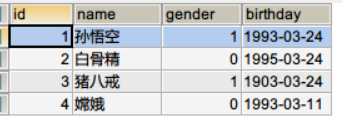
}

}

}

## 执行DML操作

* 需求：向学生表中添加4条记录，主键是自动增长



* 开发步骤：

1. 创建连接对象
2. 创建Statement语句对象
3. 执行SQL语句：executeUpdate(sql)
4. 返回影响的行数
5. 释放资源

* 代码：

package com.itheima;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

public class Demo04DML {

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

// 1) 创建连接对象

Connection connection = DriverManager.getConnection("jdbc:mysql:///day24","root","root");

// 2) 创建Statement语句对象

Statement statement = connection.createStatement();

// 3) 执行SQL语句：executeUpdate(sql)

int row = statement.executeUpdate("INSERT INTO student VALUES (NULL,'孙悟空',true,'1999-9-9'), (NULL,'白骨精',false,'1999-9-9'), (NULL,'孙悟饭',true,'1999-9-9'), (NULL,'牛魔王',true,'1999-9-9')");

// 4) 返回影响的行数

System.out.println("插入了" + row + "条记录");

// 5) 释放资源

statement.close();

connection.close();

}

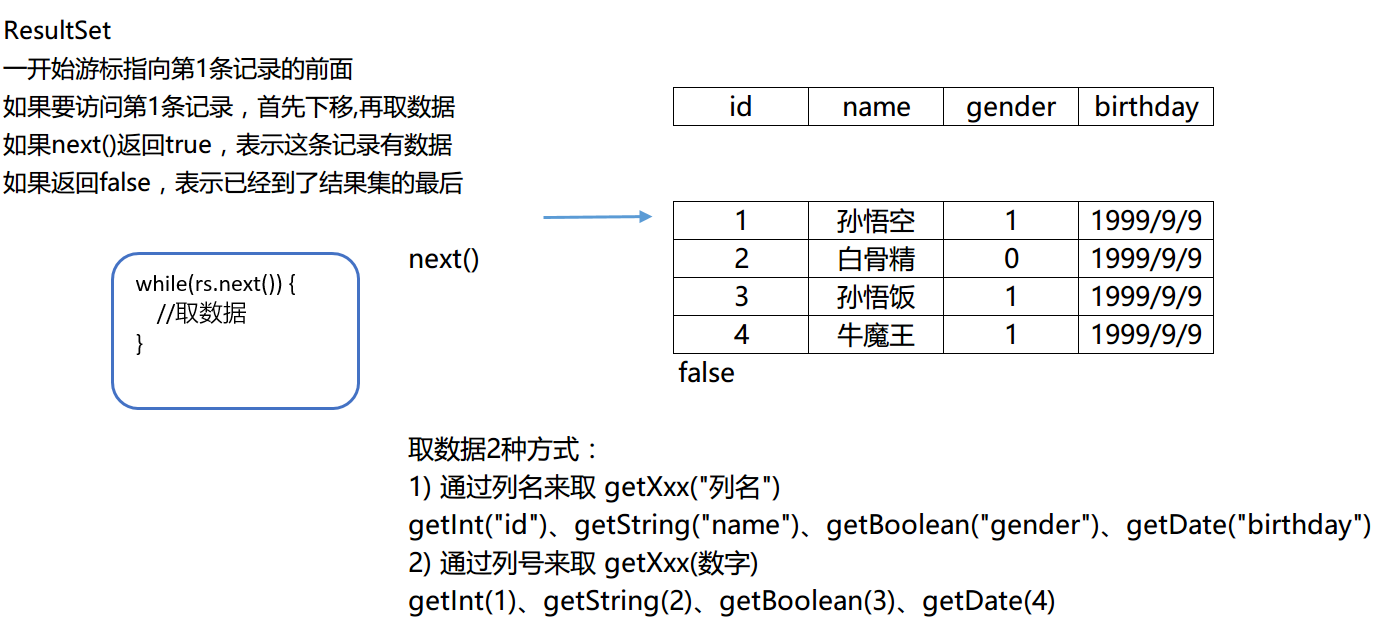
}

## 执行DQL操作

### ResultSet接口：

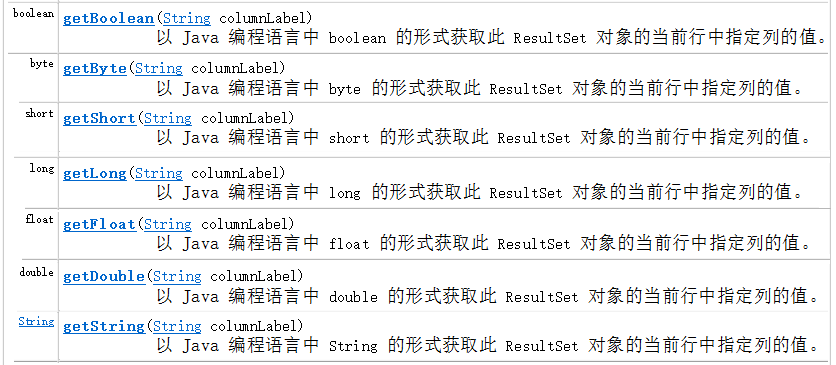
* 作用：

用来封装查询的结果集，后续要调用它的方法从结果集中取出数据。



* 接口中的方法：

|  |  |
| --- | --- |
| **ResultSet接口中的方法** | **描述** |
| **boolean next()** | 1) 向下移动一行  2) 判断当前所指的是否是记录，如果是记录则返回true，如果是最后一行的后面，则返回false |
| **数据类型 getXxx(参数)** | getInt("id")、getString("name")、getBoolean("gender")、getDate("birthday") |



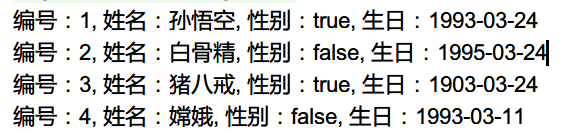
### 常用数据类型转换表

|  |  |  |
| --- | --- | --- |
| **SQL类型** | **Jdbc对应方法** | **Java返回类型** |
| **BIT(1) bit(n)** | **getBoolean()** | **boolean** |
| **TINYINT** | **getByte()** | **byte** |
| **SMALLINT** | **getShort()** | **short** |
| **INT** | **getInt()** | **int** |
| **BIGINT** | **getLong()** | **long** |
| **CHAR,VARCHAR** | **getString()** | **String** |
| **DATE** | **getDate()** | **java.sql.Date 只表示日期** |
| **TIME** | **getTime()** | **java.sql.Time 只表示时间** |
| **TIMESTAMP** | **getTimestamp()** | **java.sql.Timestamp 同时有日期和时间** |

* 注：java.sql.Date、java.sql.Time、java.sql.Timestamp(时间戳)，三个共同父类是：java.util.Date

### 需求：确保数据库中有3条以上的记录，查询所有的学员信息

* 查询结果：



* 开发步骤：

1. 得到连接对象
2. 得到语句对象
3. 执行SQL语句后得到结果集ResultSet对象
4. 循环遍历取出每一条记录
5. 输出的控制台上
6. 释放资源

* 代码

package com.itheima;

import java.sql.\*;

public class Demo05DQL {

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

//1) 得到连接对象

Connection connection = DriverManager.getConnection("jdbc:mysql:///day24","root","root");

//2) 得到语句对象

Statement statement = connection.createStatement();

//3) 执行SQL语句后得到结果集ResultSet对象

ResultSet resultSet = statement.executeQuery("select \* from student");

//4) 循环遍历取出每一条记录

while(resultSet.next()) {

//5) 输出的控制台上

System.out.println("编号：" + resultSet.getInt("id"));

System.out.println("姓名：" + resultSet.getString("name"));

System.out.println("性别：" + (resultSet.getBoolean("gender")?"男":"女"));

System.out.println("生日：" + resultSet.getDate("birthday"));

System.out.println("========================");

}

//6) 释放资源

resultSet.close();

statement.close();

connection.close();

}

}

### 关于ResultSet接口中的注意事项：

1. 如果光标在第一行之前，使用rs.getXX()获取列值，报错：Before start of result set
2. 如果光标在最后一行之后，使用rs.getXX()获取列值，报错：After end of result set
3. 使用完毕以后要关闭结果集ResultSet，再关闭Statement，再关闭Connection

# 数据库工具类JdbcUtils

* 什么时候需要创建自己的工具类？

如果一个功能经常要使用，而且代码相对不变。可以做成一个工具类。

* 需求：

上面使用Statement写的代码中出现了很多重复的代码，可以把这些公共代码抽取出来。

* 创建类JdbcUtils包含3个方法：

1. 可以把几个字符串定义成常量：用户名，密码，URL，驱动类
2. 注册驱动，为了兼容以前的程序
3. 得到数据库的连接：getConnection()
4. 关闭所有打开的资源：  
   close(Connection conn, Statement stmt)，  
   close(Connection conn, Statement stmt, ResultSet rs)

* 代码

package com.itheima.utils;

import java.sql.\*;

/\*\*

\* JDBC的工具类

\*/

public class JdbcUtils {

//1) 可以把几个字符串定义成常量：用户名，密码，URL，驱动类

public static final String USER = "root";

public static final String PASSWORD = "root";

public static final String URL = "jdbc:mysql://localhost:3306/day24";

public static final String DRIVER = "com.mysql.jdbc.Driver";

static {

//2) 注册驱动，为了兼容以前的程序

try {

Class.forName(DRIVER);

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

/\*\*

\* 得到数据库的连接

\*/

public static Connection getConnection() throws SQLException {

return DriverManager.getConnection(URL,USER,PASSWORD);

}

/\*\*

\* 关闭所有打开的资源

\*/

public static void close(Connection conn, Statement stmt) {

if (stmt!=null) {

try {

stmt.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (conn!=null) {

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 关闭

\*/

public static void close(Connection conn, Statement stmt, ResultSet rs) {

if (rs!=null) {

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

//调用上面的方法关闭

close(conn,stmt);

}

}

## 案例：用户登陆

### 需求：

1. 有一张用户表，添加几条用户记录

-- 创建用户表

create table user (

id int primary key auto\_increment,

name varchar(20),

password varchar(20)

)

insert into user values (null, 'jack','123'),(null,'rose','456');

select \* from user;

-- 登录成功

select \* from user where name='jack' and password='123';

-- 登录失败

select \* from user where name='jack' and password='abc';

select \* from user where name='newboy' and password='hello';

1. 使用Statement字符串拼接的方式实现用户的登录, 用户在控制台上输入用户名和密码。

### 开发步骤：

1. 得到用户从控制台上输入的用户名和密码
2. 调用下面写的登录方法来实现登录
3. 写一个登录的方法
   1. 通过工具类得到连接
   2. 创建语句对象，使用拼接字符串的方式生成SQL语句
   3. 查询数据库，如果有记录则表示登录成功，否则登录失败
   4. 释放资源

### 代码：

package com.itheima.utils;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Scanner;

public class Demo06Login {

// 1) 得到用户从控制台上输入的用户名和密码

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("请输入用户名：");

String name = scanner.nextLine();

System.out.println("请输入密码：");

String password = scanner.nextLine();

//2) 调用下面写的登录方法来实现登录

login(name, password);

}

//3) 写一个登录的方法

private static void login(String name, String password) {

// a) 通过工具类得到连接

Connection connection = null;

Statement statement = null;

ResultSet rs = null;

try {

connection = JdbcUtils.getConnection();

// b) 创建语句对象，使用拼接字符串的方式生成SQL语句

//select \* from user where name='jack' and password='123'

String sql = "select \* from user where name='" + name + "' and password='" + password + "'";

statement = connection.createStatement();

// c) 查询数据库，如果有记录则表示登录成功，否则登录失败

rs = statement.executeQuery(sql);

//如果结果集中有数据，则登录成功

if (rs.next()) {

System.out.println("欢迎您，登录成功。" + name);

}

else {

System.out.println("登录失败");

}

} catch (SQLException e) {

e.printStackTrace();

}

// d) 释放资源

finally {

JdbcUtils.close(connection,statement,rs);

}

}

}

### SQL注入问题

* 当我们输入以下密码，我们发现我们账号和密码都不对竟然登录成功了

a' or '1'='1

* 问题分析：

select \* from user where name='newboy' and password='a' or '1'='1'

name='newboy' false

password='a' false

'1'='1' true

select \* from user where false and false or true;

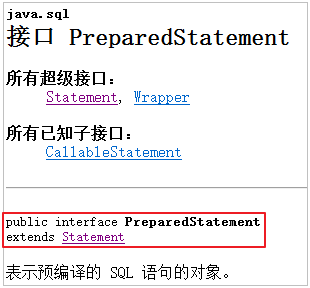
select \* from user where true;

select \* from user; -- 相当于查询所有的记录

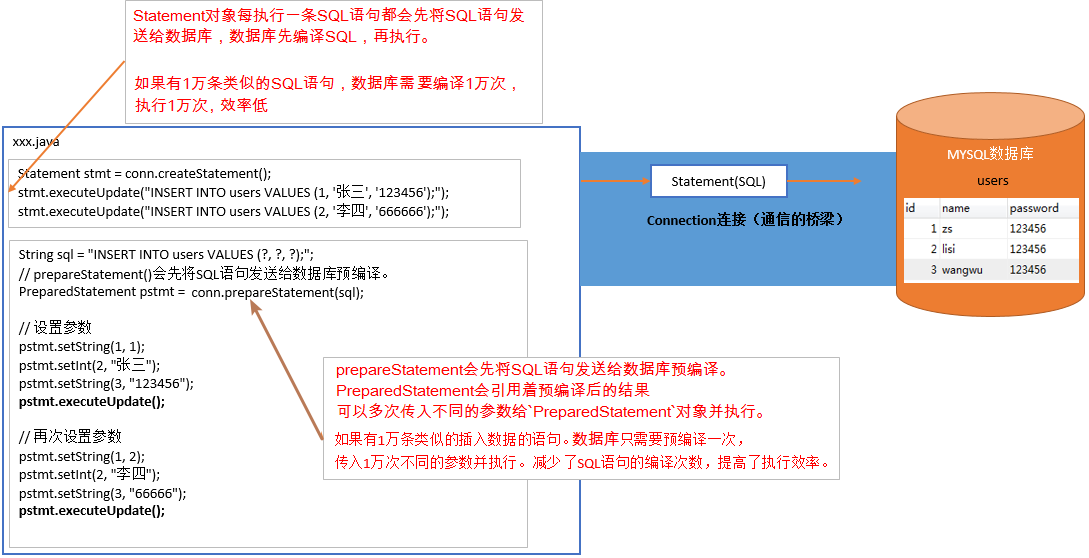
我们让用户输入的密码和SQL语句进行字符串拼接。用户输入的内容作为了SQL语句语法的一部分，改变了原有SQL真正的意义，以上问题称为SQL注入。要解决SQL注入就不能让用户输入的密码和我们的SQL语句进行简单的字符串拼接。

# PreparedStatement接口

## 继承结构与作用：



## PreparedSatement的执行原理



## Connection创建PreparedStatement对象

|  |  |
| --- | --- |
| **Connection接口中的方法** | **描述** |
| **PreparedStatement prepareStatement(String sql)** | 通过连接对象创建一个语句对象，是Statement接口的子接口，有预编译的功能，效率更高。 |

## PreparedStatement接口中的方法：

|  |  |
| --- | --- |
| **PreparedStatement接口中的方法** | **描述** |
| **int executeUpdate()** | 用于增删改操作 |
| **ResultSet executeQuery()** | 用于查询操作，注：没有提供SQL语句。在创建PreparedStatement接口时就提供了SQL语句。 |

## PreparedSatement的好处

1. prepareStatement()会先将SQL语句发送给数据库预编译。PreparedStatement会引用着预编译后的结果。可以多次传入不同的参数给PreparedStatement对象并执行。减少SQL编译次数，提高效率。
2. 安全性更高，没有SQL注入的隐患。
3. 提高了程序的可读性

## 使用PreparedStatement的步骤：

1. 编写SQL语句，未知内容使用?占位："SELECT \* FROM user WHERE name=? AND password=?";
2. 通过Connection获得PreparedStatement对象，提供SQL语句
3. 设置实际参数，调用setXxx()，不同的数据类型。
4. 执行参数化SQL语句
5. 关闭资源

|  |  |
| --- | --- |
| **PreparedStatement中设置参数的方法** | **描述** |
| **void setDouble(int parameterIndex, double x)** | 第1个参数是位置，从1开始。  第2个参数是真实的值  将指定参数设置为给定 Java double 值。 |
| **void setFloat(int parameterIndex, float x)** | 将指定参数设置为给定 Java REAL 值。 |
| **void setInt(int parameterIndex, int x)** | 将指定参数设置为给定 Java int 值。 |
| **void setLong(int parameterIndex, long x)** | 将指定参数设置为给定 Java long 值。 |
| **void setObject(int parameterIndex, Object x)** | 使用给定对象设置指定参数的值。 |
| **void setString(int parameterIndex, String x)** | 将指定参数设置为给定 Java String 值。 |

* 使用PreparedStatement改写上面的登录程序，看有没有SQL注入的情况

package com.itheima;

import com.itheima.utils.JdbcUtils;

import java.sql.\*;

import java.util.Scanner;

public class Demo07Login {

// 1) 得到用户从控制台上输入的用户名和密码

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("请输入用户名：");

String name = scanner.nextLine();

System.out.println("请输入密码：");

String password = scanner.nextLine();

//2) 调用下面写的登录方法来实现登录

login(name, password);

}

//3) 写一个登录的方法

private static void login(String name, String password) {

// a) 通过工具类得到连接

Connection connection = null;

PreparedStatement statement = null;

ResultSet rs = null;

try {

connection = JdbcUtils.getConnection();

// b) 创建语句对象，使用拼接字符串的方式生成SQL语句

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

statement = connection.prepareStatement(sql);

//要设置占位符真实的值

statement.setString(1,name);

statement.setString(2,password);

// c) 查询数据库，如果有记录则表示登录成功，否则登录失败

rs = statement.executeQuery();

//如果结果集中有数据，则登录成功

if (rs.next()) {

System.out.println("欢迎您，登录成功。" + name);

}

else {

System.out.println("登录失败");

}

} catch (SQLException e) {

e.printStackTrace();

}

// d) 释放资源

finally {

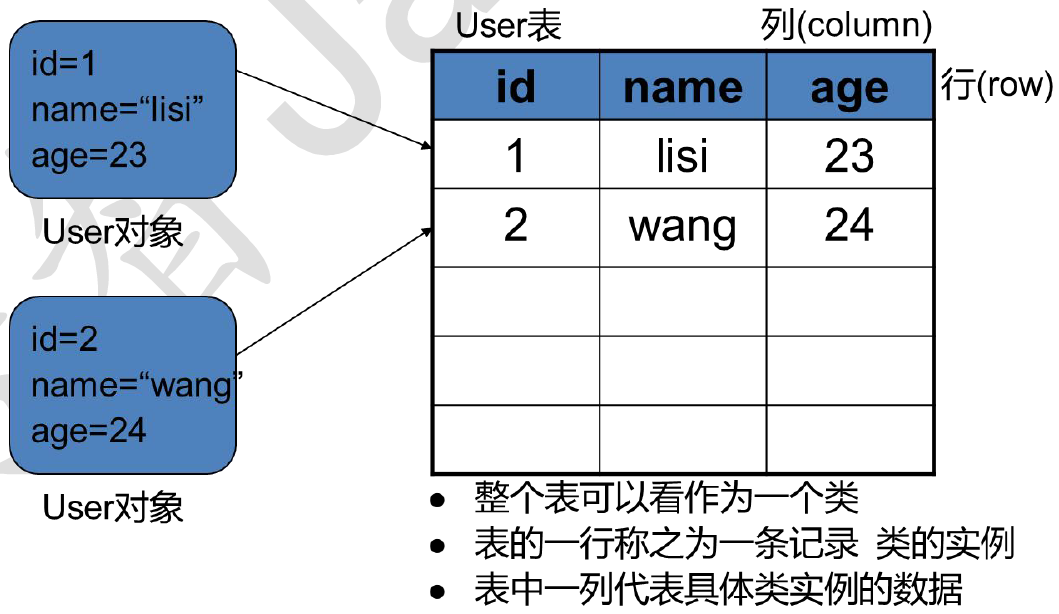
JdbcUtils.close(connection,statement,rs);

}

}

}

## 表与类的关系



### 案例：使用PreparedStatement查询id为1的一条学生数据，封装成一个学生Student对象

* 开发步骤：

1. 创建一个学生对象
2. SQL设置占位符
3. 传递参数，替换占位符
4. 将结果封装成一个学生对象
5. 释放资源
6. 使用数据，输出到控制台

* 代码：

@Test

public void testFindOne() throws SQLException {

//1) 创建一个学生对象

Student student = new Student();

Connection connection = JdbcUtils.getConnection();

// 2) SQL设置占位符

PreparedStatement ps = connection.prepareStatement("select \* from student where id=?");

// 3) 传递参数，替换占位符

ps.setInt(1,3);

// 4) 将结果封装成一个学生对象

ResultSet rs = ps.executeQuery();

if (rs.next()) {

student.setId(rs.getInt("id"));

student.setName(rs.getString("name"));

student.setGender(rs.getBoolean("gender"));

student.setBirthday(rs.getDate("birthday"));

}

// 5) 释放资源

JdbcUtils.close(connection,ps,rs);

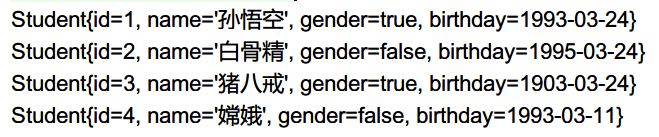
// 6) 使用数据，输出到控制台

System.out.println(student);

}

### 案例：将多条记录封装成集合，集合中每个元素是一个学生对象

* 需求： 查询所有的学生类，封装成List<Student>返回
* 执行效果：



* 开发步骤：

1. 创建一个集合用于封装所有的记录
2. SQL语句没有占位符
3. 每次循环封装一个学生对象
4. 把数据放到集合中
5. 关闭连接
6. 使用数据，特殊输出学生对象

* 代码：

public void testFindAll() throws SQLException {

// 1) 创建一个集合用于封装所有的记录

List<Student> students = new ArrayList<>();

Connection connection = JdbcUtils.getConnection();

// 2) SQL语句没有占位符

PreparedStatement ps = connection.prepareStatement("select \* from student");

ResultSet rs = ps.executeQuery();

// 3) 每次循环封装一个学生对象

while(rs.next()) {

Student student = new Student();

student.setId(rs.getInt("id"));

student.setName(rs.getString("name"));

student.setGender(rs.getBoolean("gender"));

student.setBirthday(rs.getDate("birthday"));

// 4) 把数据放到集合中

students.add(student);

}

// 5) 关闭连接

JdbcUtils.close(connection,ps,rs);

// 6) 使用数据，特殊输出学生对象

students.forEach(System.out::println);

}

## PreparedStatement执行DML操作

* 需求：

1. 向学生表中添加1条记录代码
2. 将id为2的用户，姓名更新为"猪八戒"
3. 将id为4的学员删除

* 代码：

@Test

public void testAdd() throws SQLException {

Connection conn = JdbcUtils.getConnection();

PreparedStatement ps = conn.prepareStatement("insert into student values(null, ?,?,?)");

//设置参数

ps.setString(1,"潘金莲");

ps.setBoolean(2, false);

ps.setDate(3,java.sql.Date.valueOf("1994-01-23"));

//执行更新

int i = ps.executeUpdate();

//关闭

JdbcUtils.close(conn, ps);

System.out.println("插入" + i + "条记录");

}

@Test

public void testUpdate() throws SQLException {

//得到连接

Connection conn = JdbcUtils.getConnection();

PreparedStatement ps = conn.prepareStatement("update student set name=?, gender=? where id=?");

//设置参数

ps.setString(1,"猪八戒");

ps.setBoolean(2, true);

ps.setInt(3,2);

//执行更新

int i = ps.executeUpdate();

//关闭

JdbcUtils.close(conn, ps);

System.out.println("更新" + i + "条记录");

}

//将id为4的学员删除

@Test

public void testDelete() throws SQLException {

//得到连接

Connection conn = JdbcUtils.getConnection();

PreparedStatement ps = conn.prepareStatement("delete from student where id=?");

//设置参数

ps.setInt(1,4);

//执行更新

int i = ps.executeUpdate();

//关闭

JdbcUtils.close(conn, ps);

System.out.println("删除" + i + "条记录");

}

# JDBC事务的处理

之前我们是使用MySQL的命令来操作事务。接下来我们使用JDBC来操作银行转账的事务。

## 准备数据

CREATE TABLE account (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 NAME VARCHAR(10),  
 balance DOUBLE  
);  
-- 添加数据  
INSERT INTO account (NAME, balance) VALUES ('Jack', 1000), ('Rose', 1000);

## API介绍

|  |  |
| --- | --- |
| **Connection接口中与事务有关的方法** | **说明** |
| **void setAutoCommit(boolean autoCommit)** | 设置SQL语句是否自动提交  参数如果是false，关闭自动提交，手动提交。 |
| **void commit()** | 提交事务 |
| **void rollback()** | 回滚 |

## 开发步骤

* 先试一下没有事务的转账情况
* 使用事务的情况

1. 获取连接
2. 开启事务
3. 获取到PreparedStatement
4. 使用PreparedStatement执行两次更新操作
5. 正常情况下提交事务
6. 出现异常回滚事务
7. 最后关闭资源

* 案例代码

package com.itheima;

import com.itheima.utils.JdbcUtils;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class Demo09Transaction {

//先试一下没有事务的转账情况

public static void main(String[] args) {

Connection conn = null;

PreparedStatement ps = null;

try {

conn = JdbcUtils.getConnection();

//开启事务

conn.setAutoCommit(false);

//得到PreparedStatement对象

ps = conn.prepareStatement("update account set balance =balance - 500 where name='jack'");

//执行第1条

ps.executeUpdate();

//出现异常

System.out.println(100 / 0);

ps = conn.prepareStatement("update account set balance =balance + 500 where name='rose'");

ps.executeUpdate();

//提交事务

conn.commit();

System.out.println("转账成功");

} catch (Exception e) {

e.printStackTrace();

System.out.println("转账失败");

//事务回滚

try {

conn.rollback();

} catch (SQLException e1) {

e1.printStackTrace();

}

} finally {

JdbcUtils.close(conn,ps);

}

}

}