

# JDBC

## 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

## 第1章 JDBC的概念

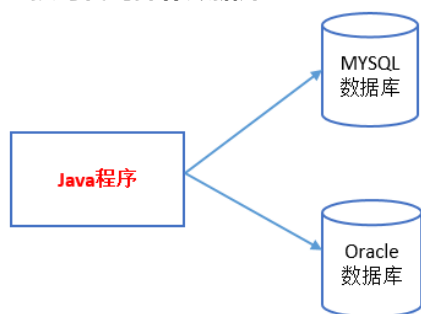
客户端操作MySQL数据库的方式

1. 使用第三方客户端来访问MySQL：SQLyog、Navicat、SQLWave、MyDB Studio、EMS SQL Manager for MySQL
2. 使用MySQL自带的命令行方式
3. 通过Java来访问MySQL数据库，今天要学习的内容

**什么是JDBC：**Java Data Base Connectivity (Java数据库连接) JDBC是Java访问数据库的 **标准规范** **JDBC的作用：**JDBC是用于执行SQL语句的Java API(**Java语言通过JDBC可以操作数据库**)

## 第2章 JDBC的由来

### 1. 直接写代码操作数据库



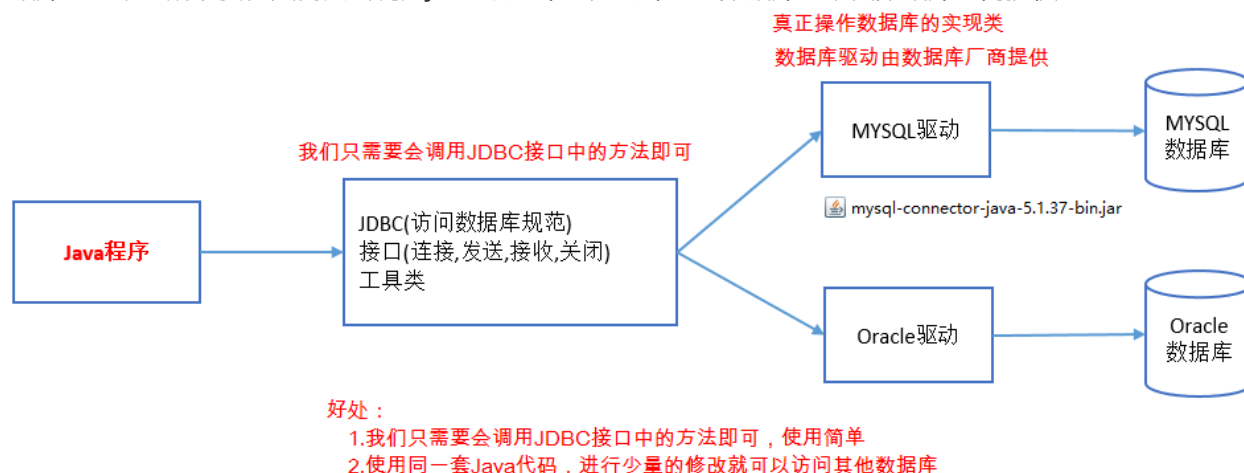
直接写代码操作数据库存在以下问题

1. 不知道MySQL数据库的操作方式，解析方式
2. 代码繁琐，写起来麻烦
3. MySQL和Oracle数据库的操作方式和解析方式不同，每个数据库都要写一套代码
4. MySQL和Oracle数据库相互切换麻烦

**直接写代码操作数据库存在的问题：**

1. 不知道MySQL数据库的操作方式，解析方式
2. 代码繁琐，写起来麻烦
3. MySQL和Oracle等其他数据库的操作方式和解析方式不同，每个数据库都要写一套代码
4. MySQL和Oracle等其他数据库相互切换麻烦

2. **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。



### JDBC的好处：

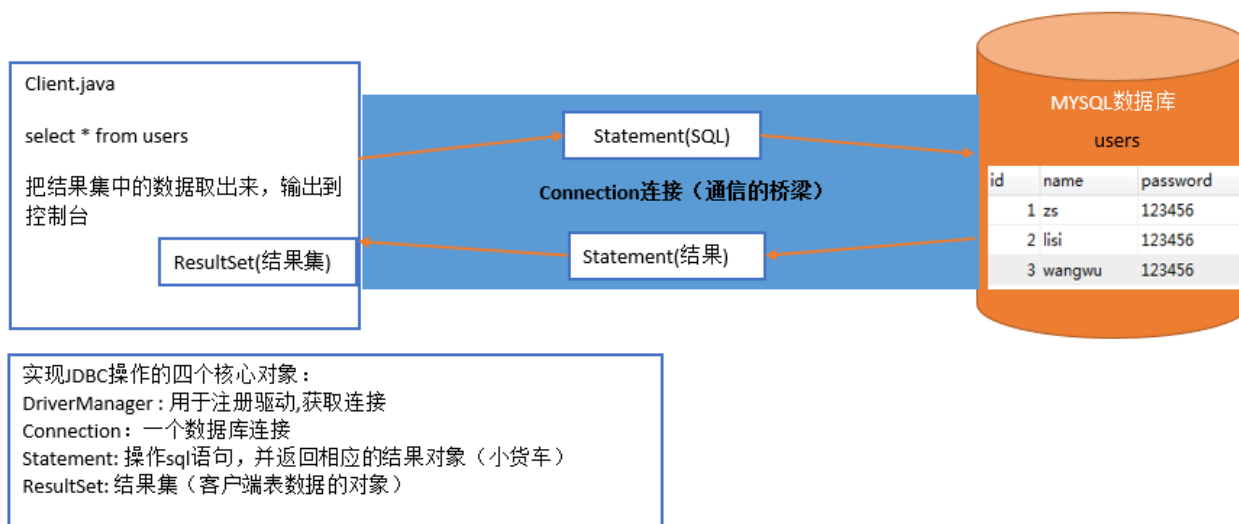
1. 我们只需要会调用JDBC接口中的方法即可，使用简单
2. 使用同一套Java代码，进行少量的修改就可以访问其他JDBC支持的数据库

### JDBC会用到的包：

1. java.sql：JDBC访问数据库的基础包，在javaSE中的包。如：java.sql.Connection
2. javax.sql：JDBC访问数据库的扩展包
3. 数据库的驱动，各大数据库厂商来实现。如：MySQL的驱动：com.mysql.jdbc.Driver

### JDBC四个核心对象 这几个类都是在java.sql包中

1. DriverManager: 用于注册驱动
2. Connection: 表示与数据库创建的连接
3. Statement: 执行SQL语句的对象
4. ResultSet: 结果集或一张虚拟表



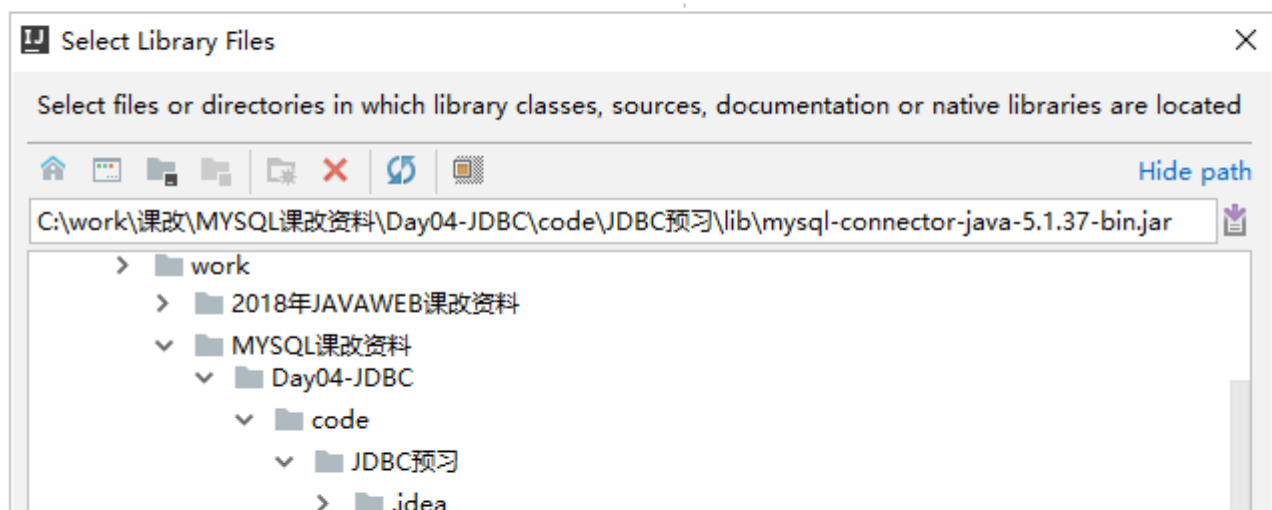
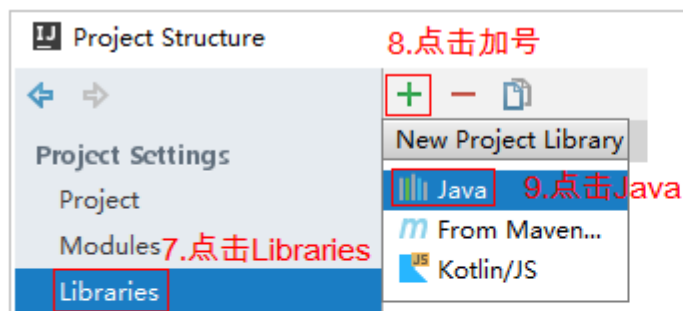
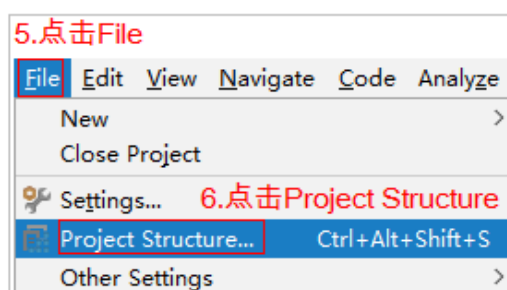
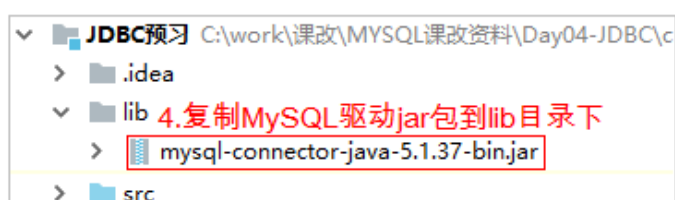
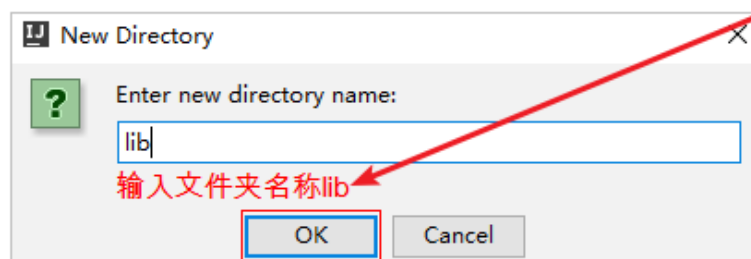
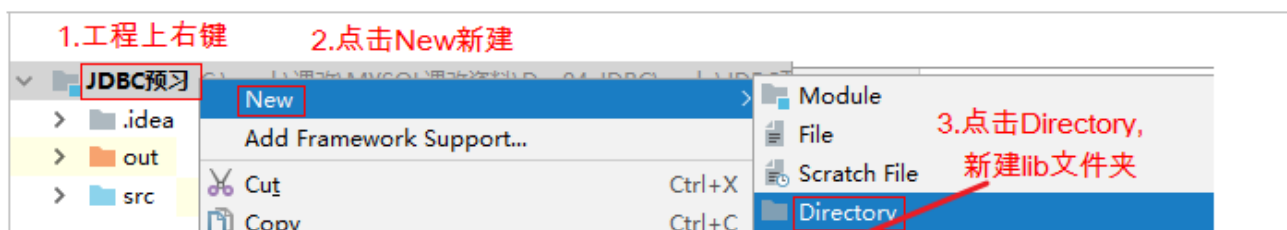
## 第3章 JDBC获取连接

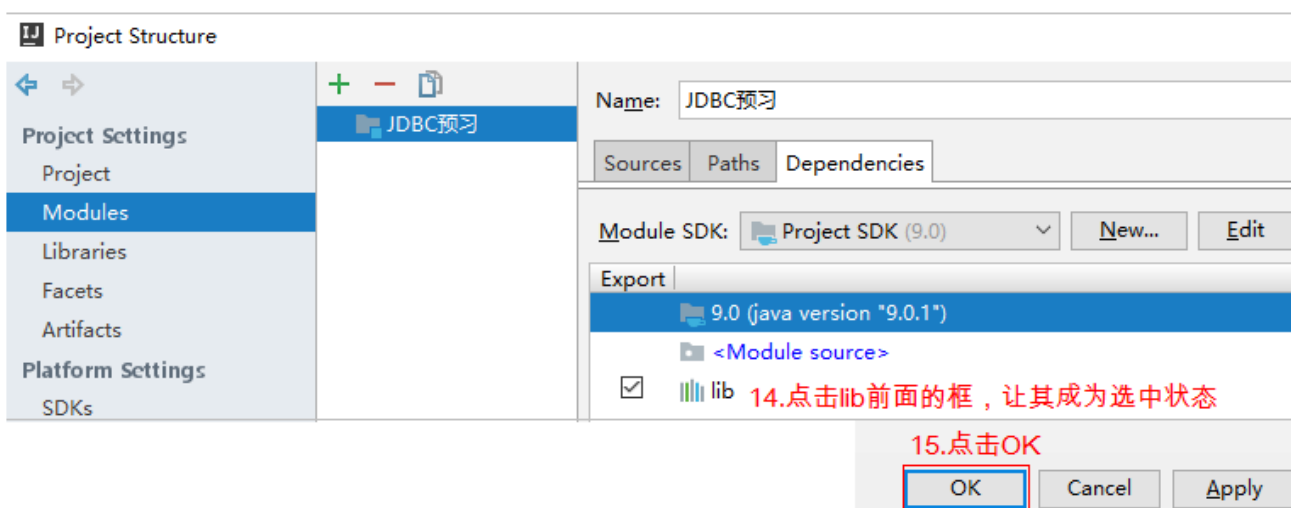
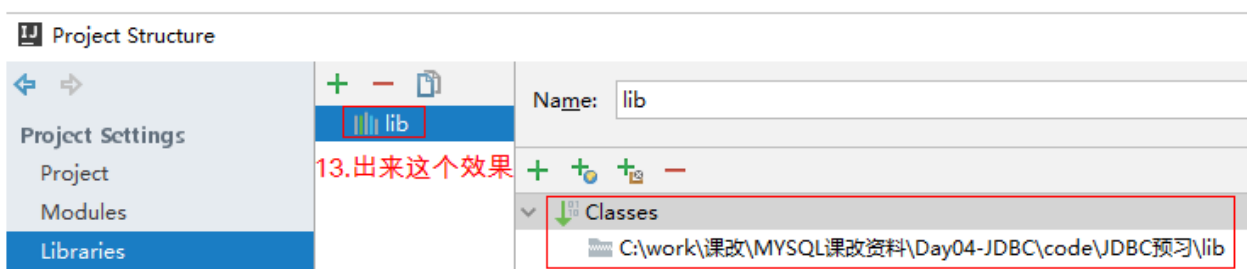
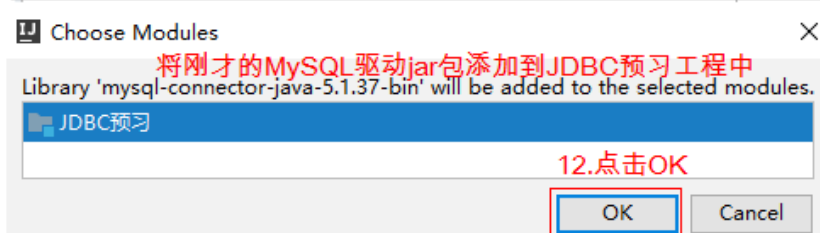
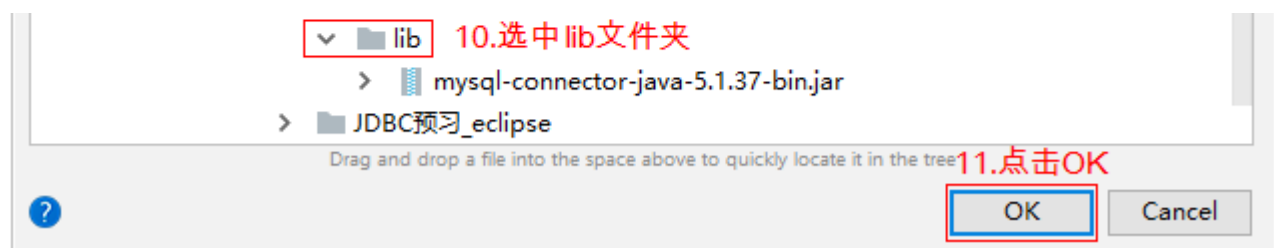
`Connection` 表示Java程序与数据库之间的连接，只有拿到Connection才能操作数据库。

JDBC获取连接步骤 1.导入驱动jar包 2.注册驱动 3.获取连接

### 3.1 导入驱动jar包

---



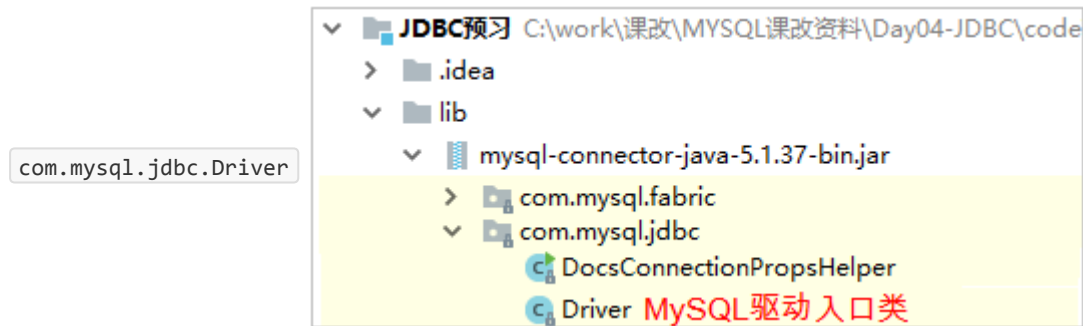


使用JDBC的好处：

- 1) 程序员如果要开发访问数据库的程序，只需要会调用JDBC接口中的方法即可，不用关注类是如何实现的。
- 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

## 3.2 注册驱动

我们Java程序需要通过数据库驱动才能连接到数据库，因此需要注册驱动。MySQL的驱动的入口类是：



### 3.2.1 API介绍

`java.sql.DriverManager` 类用于注册驱动。提供如下方法注册驱动

```
static void registerDriver(Driver driver)
```

向 `DriverManager` 注册给定驱动程序。

### 3.2.2 使用步骤

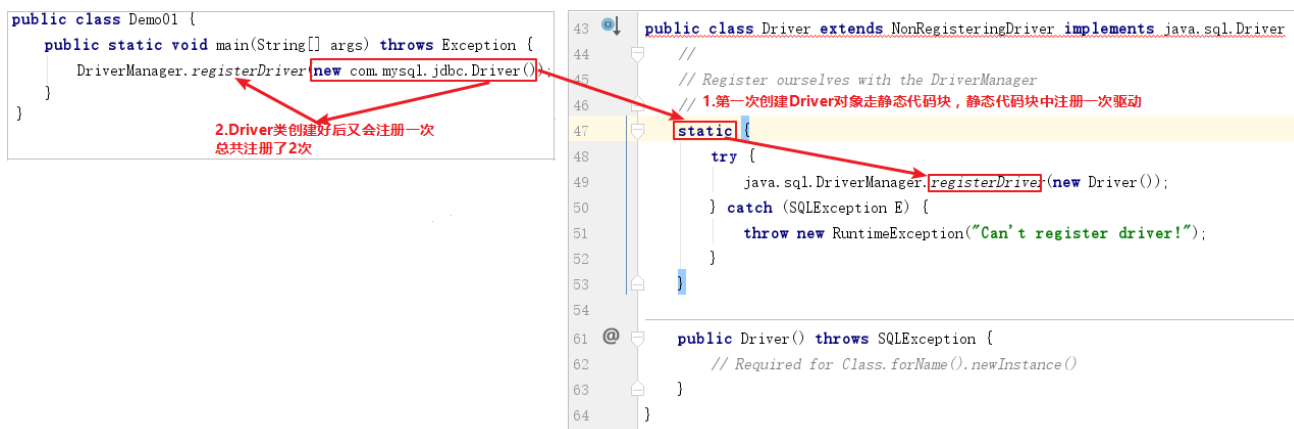
1. `DriverManager.registerDriver(驱动对象)`; 传入对应参数即可

### 3.2.3 案例代码

```
public class Demo01 {
    public static void main(String[] args) throws Exception {
        // 注册驱动
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());
    }
}
```

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 {
            // 自己自动注册
            java.sql.DriverManager.registerDriver(new Driver());
        } catch (SQLException E) {
            throw new RuntimeException("Can't register driver!");
        }
    }
    public Driver() throws SQLException {
    }
}
```



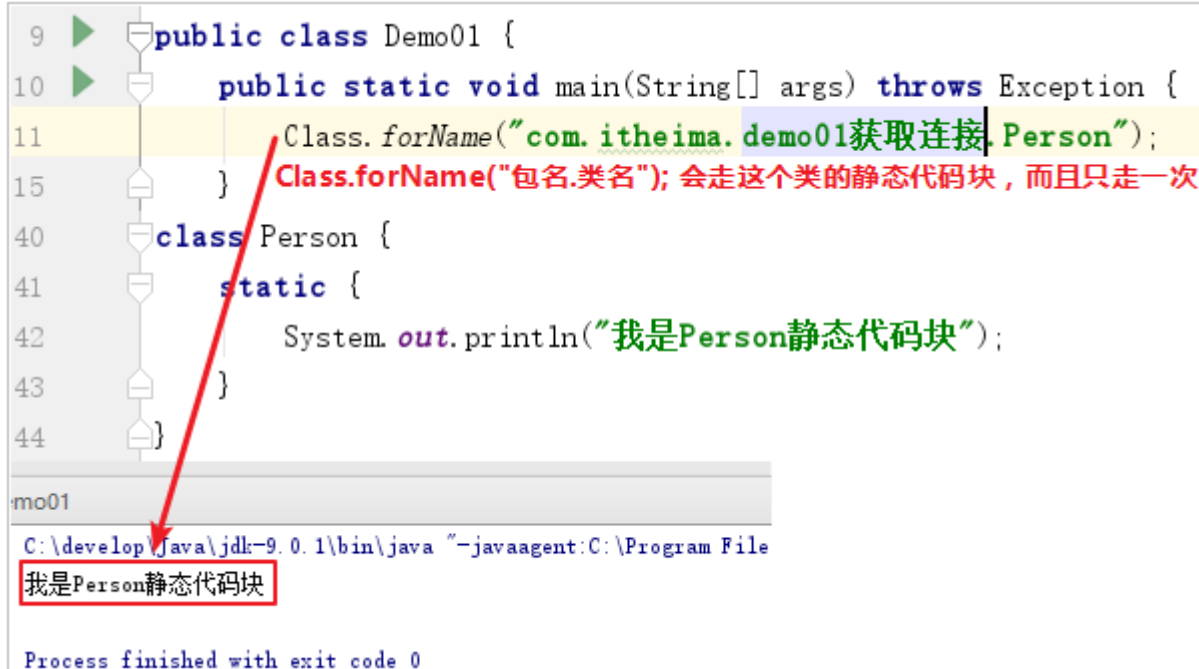
注意：使用 `DriverManager.registerDriver(new com.mysql.jdbc.Driver());`，存在两方面不足

1. 硬编码，后期不易于程序扩展和维护
2. 驱动被注册两次

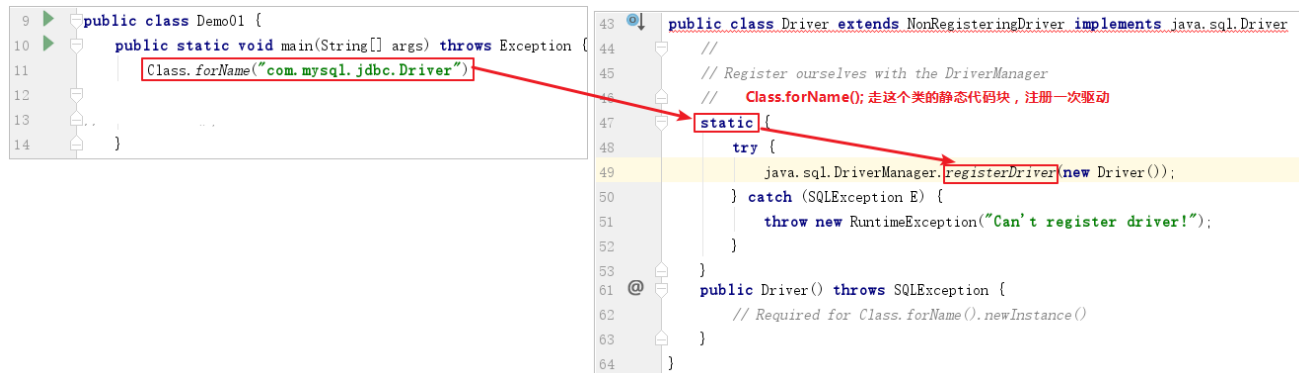
使用 `Class.forName("com.mysql.jdbc.Driver");` 加载驱动，这样驱动只会注册一次

```
public class Demo01 {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");
    }
}
```

演示：`Class.forName("包名.类名");` 会走这个类的静态代码块



通常开发我们使用Class.forName() 加载驱动。Class.forName("com.mysql.jdbc.Driver"); 会走Driver类的静态代码块。在静态代码块中注册一次驱动。



总结：注册MySQL驱动使用 `Class.forName("com.mysql.jdbc.Driver");`

## 3.3 获取连接

### 3.3.1 API介绍

`java.sql.DriverManager` 类中有如下方法获取数据库连接

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

连接到给定数据库 URL，并返回连接。

### 3.3.2 参数说明

1. `String url`：连接数据库的URL，用于说明连接数据库的位置
2. `String user`：数据库的账号
3. `String password`：数据库的密码

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：`jdbc:mysql://localhost:3306/day24`

如果是本地服务器，端口号是默认的3306，则可以简写：`jdbc:mysql:///day24`

### 3.3.3 注意事项

如果数据出现乱码需要加上参数: `?characterEncoding=utf8`，表示让数据库以UTF-8编码来处理数据。如：  
`jdbc:mysql://localhost:3306/day24?characterEncoding=utf8`

### 3.3.4 使用步骤

1. `DriverManager.getConnection(url, user, password);` 传入对应参数即可

### 3.3.5 案例代码



```

public class Demo01 {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");

        // 连接到MySQL
        // url: 连接数据库的URL
        // user: 数据库的账号
        // password: 数据库的密码
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/day24",
"root", "root");
        System.out.println(conn);
    }
}

```

### 3.3.6 案例效果



### 3.3.7 使用配置文件保存数据库账号密码（了解）

上面案例是将数据库的账号和密码写在代码中。还有一种方式是将数据库的账号密码写在Properties文件中

#### 3.3.7.1 API介绍

`java.sql.DriverManager` 类中有如下方法获取数据库连接

```

static Connection getConnection(String url, Properties info)

```

试图建立到给定数据库 URL 的连接。

#### 3.3.7.2 使用步骤

1. 在src目录下创建名为jdbc.properties的文件，内容如下：

```

user=root
password=root

```

2. 将jdbc.properties内容加载到Properties对象中
3. 调用 `getConnection` 传入url和Properties对象

#### 3.3.7.3 注意事项

`Demo01.class.getResourceAsStream("/jdbc.properties");` 会找到src/jdbc.properties加载，并返回 `InputStream` 对象。

### 3.3.7.4 案例代码

```
public class Demo01 {
    public static void main(String[] args) throws Exception {
        test02();
    }
    // 将src/jdbc.properties文件中的内容加载到Properties对象中
    private static void test02() throws Exception {
        InputStream is = Demo01.class.getResourceAsStream("/jdbc.properties");
        Properties pp = new Properties();
        pp.load(is);

        Class.forName("com.mysql.jdbc.Driver");
        Connection conn = DriverManager.getConnection("jdbc:mysql:///day24", pp);
        System.out.println(conn);
    }
}
```

## 第4章 JDBC实现对单表数据增、删、改、查

我们要对数据库进行增、删、改、查，需要使用 `Statement` 对象来执行SQL语句。

### 4.1 准备数据

```
-- 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');
```

### 4.2 JDBC实现对单表数据增、删、改

#### 4.2.1 API介绍

##### 4.2.1.1 获取Statement对象API介绍

在 `java.sql.Connection` 接口中有如下方法获取到 `Statement` 对象

```
Statement createStatement()
创建一个 Statement 对象来将 SQL 语句发送到数据库
```

##### 4.2.1.2 Statement的API介绍

1. `boolean execute(String sql)`  
此方法可以执行任意sql语句。返回boolean值，表示是否返回ResultSet结果集。仅当执行select语句，且有返回结果时返回true，其它语句都返回false;
2. `int executeUpdate(String sql)`  
根据执行的DML ( INSERT、UPDATE、DELETE ) 语句，返回受影响的行数
3. `ResultSet executeQuery(String sql)`  
根据查询语句返回结果集,只能执行SELECT语句

注意：在MySQL中，只要不是查询就是修改。 `executeUpdate`：用于执行增删改 `executeQuery`：用于执行查询

## 4.2.2 使用步骤

1. 注册驱动
2. 获取连接
3. 获取Statement对象
4. 使用Statement对象执行SQL语句
5. 释放资源

## 4.2.3 案例代码

```
public class Demo03 {  
    public static void main(String[] args) throws Exception {  
        Class.forName("com.mysql.jdbc.Driver");  
  
        Connection conn = DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");  
        System.out.println(conn);  
  
        // String sql = "SELECT * FROM category;";  
        // 从连接中拿到一个Statement对象  
        Statement stmt = conn.createStatement();  
  
        // 1.插入记录  
        String sql = "INSERT INTO category (cname) VALUES ('手机');";  
        int i = stmt.executeUpdate(sql);  
        System.out.println("影响的行数:" + i);  
  
        // 2.修改记录  
        sql = "UPDATE category SET cname='汽车' WHERE cid=4;";  
        i = stmt.executeUpdate(sql);  
        System.out.println("影响的行数:" + i);  
  
        // 3.删除记录  
        sql = "DELETE FROM category WHERE cid=1;";  
        i = stmt.executeUpdate(sql);  
        System.out.println("影响的行数:" + i);  
  
        // 释放资源
```

```

        stmt.close();
        conn.close();
    }
}

```

## 4.2.4 案例效果

前

cid	cname
1	家电
2	服饰
3	化妆品

后

cid	cname
2	汽车
3	化妆品
4	手机

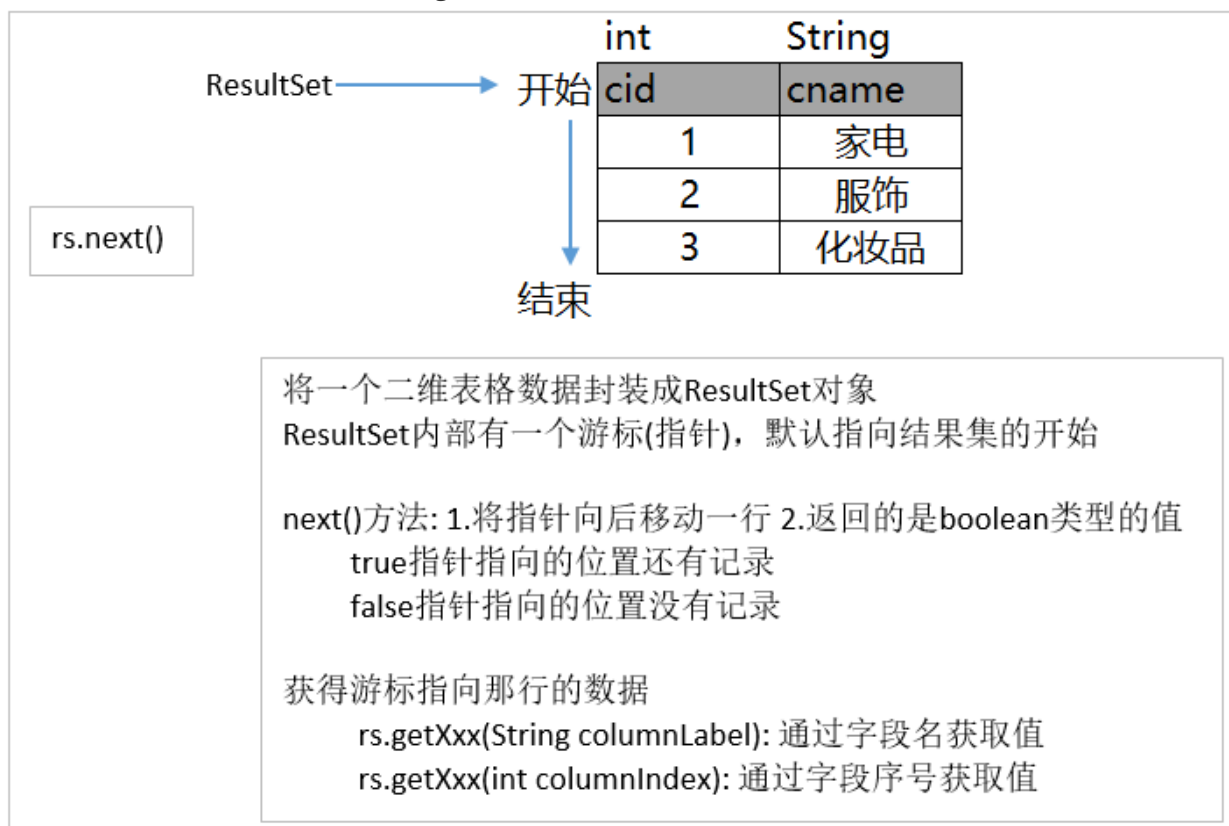
1.插入记录  
2.修改记录  
3.删除记录

## 4.3 JDBC实现对单表数据查询

`ResultSet` 用于保存执行查询SQL语句的结果。我们不能一次性取出所有的数据，需要一行一行的取出。

**ResultSet的原理：**

1. `ResultSet`内部有一个指针,刚开始记录开始位置
2. 调用`next`方法, `ResultSet`内部指针会移动到下一行数据
3. 我们可以通过`ResultSet`得到一行数据 `getXxx`得到某列数据



**ResultSet获取数据的API** 其实ResultSet获取数据的API是有规律的get后面加数据类型。我们统称 `getXXX()`

boolean	<code>getBoolean(String columnLabel)</code> 以 Java 编程语言中 boolean 的形式获取此 ResultSet 对象的当前行中指定列的值。
byte	<code>getByte(String columnLabel)</code> 以 Java 编程语言中 byte 的形式获取此 ResultSet 对象的当前行中指定列的值。
short	<code>getShort(String columnLabel)</code> 以 Java 编程语言中 short 的形式获取此 ResultSet 对象的当前行中指定列的值。
long	<code>getLong(String columnLabel)</code> 以 Java 编程语言中 long 的形式获取此 ResultSet 对象的当前行中指定列的值。
float	<code>getFloat(String columnLabel)</code> 以 Java 编程语言中 float 的形式获取此 ResultSet 对象的当前行中指定列的值。
double	<code>getDouble(String columnLabel)</code> 以 Java 编程语言中 double 的形式获取此 ResultSet 对象的当前行中指定列的值。
String	<code>getString(String columnLabel)</code> 以 Java 编程语言中 String 的形式获取此 ResultSet 对象的当前行中指定列的值。

### 使用JDBC查询数据库中的数据步骤

1. 注册驱动
2. 获取连接
3. 获取到Statement
4. 使用Statement执行SQL
5. ResultSet处理结果
6. 关闭资源

### 案例代码

```
public class Demo04 {  
    public static void main(String[] args) throws Exception {  
        Class.forName("com.mysql.jdbc.Driver");  
  
        Connection conn = DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");  
        Statement stmt = conn.createStatement();  
  
        String sql = "SELECT * FROM category;";  
        ResultSet rs = stmt.executeQuery(sql);  
  
        // 内部有一个指针,只能取指针指向的那条记录  
        while (rs.next()) { // 指针移动一行,有数据才返回true  
            // 取出数据  
            int cid = rs.getInt("cid");  
            String cname = rs.getString("cname");  
  
            System.out.println(cid + " == " + cname);  
        }  
  
        // 关闭资源  
        rs.close();  
        stmt.close();  
        conn.close();  
    }  
}
```


注意：

1. 如果光标在第一行之前，使用rs.getXXX()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.getXXX()获取列值，报错：After end of result set

数据库中数据

cid	cname
2	汽车
3	化妆品
4	手机

程序查询出数据

 Console 

develop\Java\j

2 == 汽车

3 == 化妆品

4 == 手机

案例效果

总结：其实我们使用JDBC操作数据库的步骤都是固定的。不同的地方是在编写SQL语句

1. 注册驱动
2. 获取连接
3. 获取到Statement
4. 使用Statement执行SQL
5. ResultSet处理结果
6. 关闭资源

## 第5章 JDBC事务

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。

### 5.1 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 ('张三', 1000), ('李四', 1000);
```

### 5.2 API介绍

Connection 接口中与事务有关的方法

1. `void setAutoCommit(boolean autoCommit) throws SQLException;`  
false：开启事务，ture：关闭事务
2. `void commit() throws SQLException;`  
提交事务

3. `void rollback() throws SQLException;`  
回滚事务

## 5.3 使用步骤

1. 注册驱动
2. 获取连接
3. 获取到Statement
4. **开启事务**
5. 使用Statement执行SQL
6. **提交或回滚事务**
7. 关闭资源

## 5.4 案例代码

```
public class Demo05 {
    public static void main(String[] args) {
        Connection conn = null;
        try {
            // 拿到连接
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql:///day24", "root", "root");

            // 开启事务
            conn.setAutoCommit(false);

            Statement pstmt = conn.createStatement();

            // 张三减500
            String sql = "UPDATE account SET balance = balance - 500 WHERE id=1;";
            pstmt.executeUpdate(sql);
            // 模拟异常
            // int i = 10 / 0;

            // 李四加500
            sql = "UPDATE account SET balance = balance + 500 WHERE id=2;";
            pstmt.executeUpdate(sql);

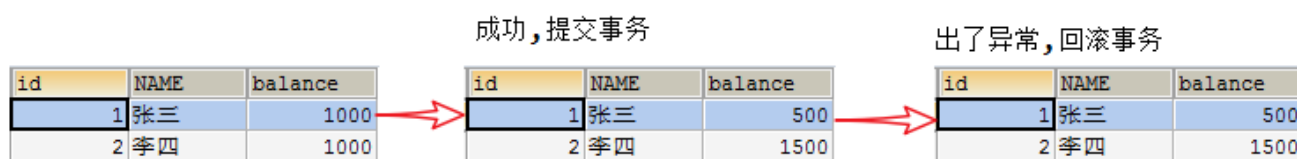
            pstmt.close();
            // 成功,提交事务
            System.out.println("成功,提交事务");
            conn.commit();
        } catch (Exception e) {
            // 失败,回滚事务
            try {
                System.out.println("出了异常,回滚事务");
                conn.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```

```

    } finally {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}
}

```

## 5.5 案例效果



## 第6章 JDBC获取连接与关闭连接工具类实现

通过上面案例需求我们会发现每次去执行SQL语句都需要注册驱动, 获取连接, 得到Statement, 以及释放资源。发现很多重复的劳动, 我们可以将重复的代码定义到某个类的方法中。直接调用方法, 可以简化代码。那么我们接下来定义一个 `JDBCUtils` 类。把注册驱动, 获取连接, 得到Statement, 以及释放资源的代码放到这个类的方法中。以后直接调用方法即可。

### 6.1 编写JDBC工具类步骤

1. 将固定字符串定义为常量
2. 在静态代码块中注册驱动(只注册一次)
3. 提供一个获取连接的方法 `static Connection getConnection();`
4. 定义关闭资源的方法 `close(Connection conn, Statement stmt, ResultSet rs)`
5. 重载关闭方法 `close(Connection conn, Statement stmt)`

### 6.2 案例代码

`JDBCUtils.java`

```

public class JDBCUtils {
    // 1. 将固定字符串定义为常量
    private static final String DRIVER_CLASS = "com.mysql.jdbc.Driver";
    private static final String URL = "jdbc:mysql:///day24";
    private static final String USER = "root";
    private static final String PASSWORD = "root";

    // 2. 在静态代码块中注册驱动(只注册一次)
    // 当这个类加载到内存的时候就走这个静态代码块, 再去触发Driver类中的静态代码块, 主动注册
    static {
        try {
            Class.forName(DRIVER_CLASS);
        } catch (ClassNotFoundException e) {}
    }
}

```



```

    }

    // 3.提供一个获取连接的方法
    static Connection getConneciton();
    // 我们面向JDBC编程
    public static Connection getConnection() throws SQLException {
        Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
        return conn;
    }

    // 5.重载关闭方法
    close(Connection conn, Statement stmt)
    public static void close(Connection conn, Statement stmt) {
        close(conn, stmt, null);
    }

    // 4.定义关闭资源的方法
    close(Connection conn, Statement stmt, ResultSet rs)
    public static void close(Connection conn, Statement stmt, ResultSet rs) {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {}
        }

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

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

```

Demo06.java

```

public class Demo06 {
    public static void main(String[] args) throws Exception {
        createTable();
        // addEmployee();
        // updateEmployee();
        // deleteEmployee();
    }

    // 删除员工
    public static void deleteEmployee() throws Exception {
        Connection conn = JDBCUtils.getConnection();
        Statement stmt = conn.createStatement();

        // 删除id为3的员工
        String sql = "DELETE FROM employee WHERE id=3;";
    }
}

```

```

        int i = stmt.executeUpdate(sql);
        System.out.println("影响的行数: " + i);

//        stmt.close();
//        conn.close();
//        JDBCUtils.close(conn, stmt, null);
//        JDBCUtils.close(conn, stmt);
    }

    // 修改员工
    public static void updateEmployee() throws Exception {
        Connection conn = JDBCUtils.getConnection();
        Statement stmt = conn.createStatement();

        // 将id为3的员工姓名改成田七, 地址改成天津
        String sql = "UPDATE employee SET address='天津', name='田七' WHERE id=3;";

        int i = stmt.executeUpdate(sql);
        System.out.println("影响的行数: " + i);

//        stmt.close();
//        conn.close();
//        JDBCUtils.close(conn, stmt, null);
//        JDBCUtils.close(conn, stmt);
    }

    // 定义添加员工
    public static void addEmployee() throws Exception {
        Connection conn = JDBCUtils.getConnection();
        Statement stmt = conn.createStatement();

        // 添加4个员工
        String sql = "INSERT INTO employee VALUES (NULL, '张三4', 20, '北京'),"
            + " (NULL, '李四4', 21, '南京'),"
            + " (NULL, '王五4', 18, '东京'),"
            + " (NULL, '赵六4', 17, '西安');";

        int i = stmt.executeUpdate(sql);
        System.out.println("影响的行数: " + i);

//        stmt.close();
//        conn.close();
//        JDBCUtils.close(conn, stmt, null);
//        JDBCUtils.close(conn, stmt);
    }

    // 创建表
    public static void createTable() throws Exception {
        Connection conn = JDBCUtils.getConnection();
        Statement stmt = conn.createStatement();

        String sql = "CREATE TABLE IF NOT EXISTS employee ("

```

```

        + " id INT PRIMARY KEY AUTO_INCREMENT,"
        + " name VARCHAR(20) UNIQUE NOT NULL,"
        + " age INT,"
        + " address VARCHAR(50)"
        + ");";

        int i = stmt.executeUpdate(sql);
        System.out.println("ok");

//        stmt.close();
//        conn.close();
//        JDBCUtils.close(conn, stmt, null);
//        JDBCUtils.close(conn, stmt);
    }
}

```

## 6.3 案例效果

```

public static void main(String[] args) throws Exception {
    createTable();
    // 添加4个员工
    addEmployee();
    // 将id为3的员工姓名改成田七，地址改成天津
    updateEmployee();
    // 删除id为3的员工
    deleteEmployee();
}

```

id	name	age	address
(Auto)	(NULL)	(NULL)	(NULL)

id	name	age	address
1	张三4	20	北京
2	李四4	21	南京
3	王五4	18	东京
4	赵六4	17	西安

id	name	age	address
1	张三4	20	北京
2	李四4	21	南京
3	田七	18	天津
4	赵六4	17	西安

id	name	age	address
1	张三4	20	北京
2	李四4	21	南京
4	赵六4	17	西安

## 第7章 JDBC实现登录案例

### 7.1 案例需求

模拟用户输入账号和密码登录网站

### 7.2 案例效果

1. 输入正确的账号，密码，显示登录成功

请输入账号：  
**admin**      输入正确的账号，密码  
请输入密码：  
**123**      登录成功  
欢迎您，**admin**

2. 输入错误的账号，密码，显示登录失败

请输入账号：  
**admin**      输入错误的账号或密码  
请输入密码：  
**aaa**      显示登录失败  
账号或密码错误...

## 7.3 案例分析

1. 使用数据库保存用户的账号和密码
2. 让用户输入账号和密码
3. 使用SQL根据用户的账号和密码去数据库查询数据
4. 如果查询到数据，说明登录成功
5. 如果查询不到数据，说明登录失败

## 7.4 实现步骤

1. 创建一个用户表保存用户的账号和密码，并添加一些数据，SQL语句如下：

```
CREATE TABLE USER (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    NAME VARCHAR(50),  
    PASSWORD VARCHAR(50)  
);  
INSERT INTO USER (NAME, PASSWORD) VALUES('admin', '123'), ('test', '123'), ('gm', '123');
```

2. 编写代码让用户输入账号和密码

```
public class Demo07 {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号：");  
        String name = sc.nextLine();  
        System.out.println("请输入密码：");  
        String password = sc.nextLine();  
    }  
}
```

3. 使用SQL根据用户的账号和密码去数据库查询数据

```
public class Demo07 {
```

```

public static void main(String[] args) throws Exception {
    // 让用户输入账号和密码
    Scanner sc = new Scanner(System.in);
    System.out.println("请输入账号: ");
    String name = sc.nextLine();
    System.out.println("请输入密码: ");
    String password = sc.nextLine();

    // 使用SQL根据用户的账号和密码去数据库查询数据
    Connection conn = JDBCUtils.getConnection();
    Statement stmt = conn.createStatement();
    String sql = "SELECT * FROM user WHERE name='" + name + "' AND password='" + password +
        "';";
    }
}

```

4. 如果查询到数据，说明登录成功，如果查询不到数据，说明登录失败

```

public class Demo07 {
    public static void main(String[] args) throws Exception {
        // 让用户输入账号和密码
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入账号: ");
        String name = sc.nextLine();
        System.out.println("请输入密码: ");
        String password = sc.nextLine();

        // 使用SQL根据用户的账号和密码去数据库查询数据
        Connection conn = JDBCUtils.getConnection();
        Statement stmt = conn.createStatement();
        String sql = "SELECT * FROM user WHERE name='" + name + "' AND password='" + password +
            "';";

        // 如果查询到数据，说明登录成功，如果查询不到数据，说明登录失败
        ResultSet rs = stmt.executeQuery(sql);

        if (rs.next()) {
            //能进来查询到了数据.
            String name2 = rs.getString("name");
            System.out.println("欢迎您," + name2);
        } else {
            //查询不到数据，说明登录失败
            System.out.println("账号或密码错误...");
        }

        JDBCUtils.close(conn, stmt, rs);
    }
}

```

## 7.5 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

请输入用户名：

hehe

请输入密码：

a' or '1'='1

问题分析：

// 代码中的SQL语句

```
"SELECT * FROM user WHERE name='" + name + "' AND password='" + password + "';";
```

// 将用户输入的账号密码拼接后

```
"SELECT * FROM user WHERE name='hehe' AND password='a' or '1'='1';"
```

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。

## 第8章 PreparedStatement预编译对象

继承结构：

```
java.sql
接口 PreparedStatement

所有超级接口：
    Statement, Wrapper

所有已知子接口：
    CallableStatement
```

---

```
public interface PreparedStatement
extends Statement
```

表示预编译的 SQL 语句的对象。

### 8.1 PreparedStatement的执行原理

我们写的SQL语句让数据库执行，数据库不是直接执行SQL语句字符串。和java一样，数据库需要执行编译后的SQL语句（类似java编译后的字节码文件）。

1. `Statement` 对象每执行一条SQL语句都会先将这条SQL语句发送给数据库编译，数据库再执行。

```
Statement stmt = conn.createStatement();
stmt.executeUpdate("INSERT INTO users VALUES (1, '张三', '123456');");
stmt.executeUpdate("INSERT INTO users VALUES (2, '李四', '666666');");
```

上面2条SQL语句我们可以看到大部分内容是相同的，只是数据略有不一样。数据库每次执行都编译一次。如果有1万条类似的SQL语句，数据库需要编译1万次，执行1万次，显然效率就低了。

2. `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。相当于调用方法多次传入不同的参数。

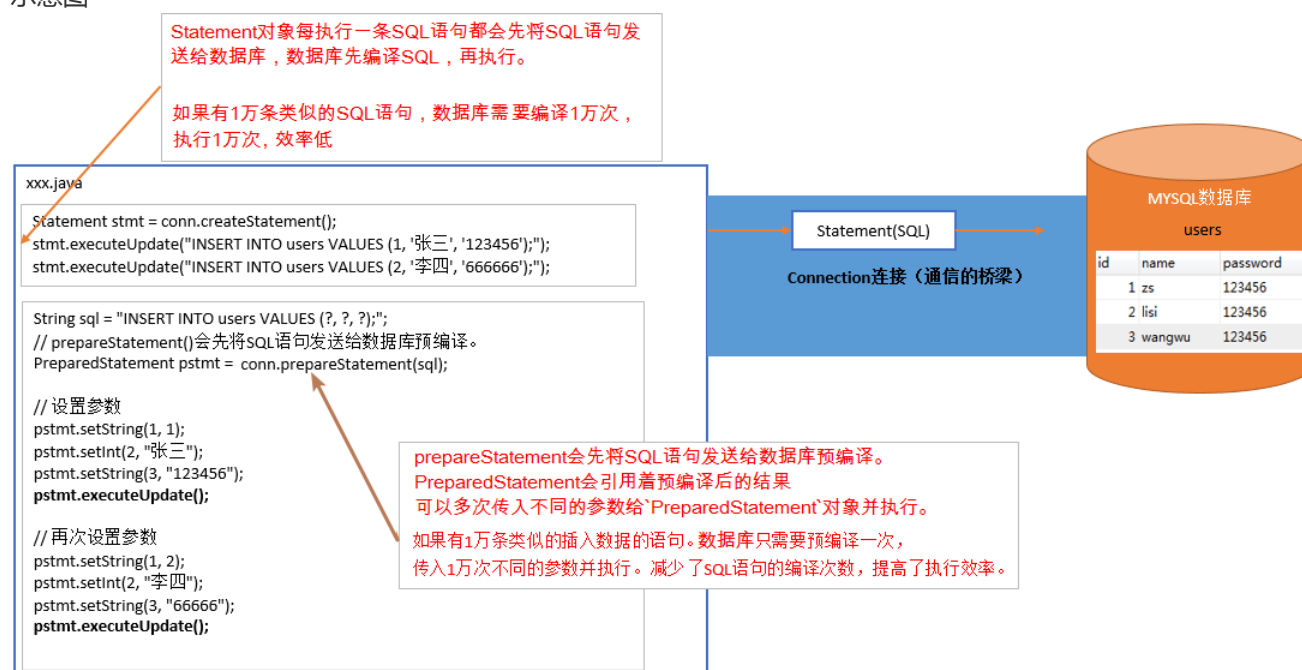
```
String sql = "INSERT INTO users VALUES (?, ?, ?)";
// 会先将SQL语句发送给数据库预编译。PreparedStatement会引用着预编译后的结果。
PreparedStatement pstmt = conn.prepareStatement(sql);

// 设置参数
pstmt.setString(1, 1);
pstmt.setInt(2, "张三");
pstmt.setString(3, "123456");
pstmt.executeUpdate();

// 再次设置参数
pstmt.setString(1, 2);
pstmt.setInt(2, "李四");
pstmt.setString(3, "666666");
pstmt.executeUpdate();
```

上面预编译好一条SQL，2次传入了不同的参数并执行。如果有1万条类似的插入数据的语句。数据库只需要预编译一次，传入1万次不同的参数并执行。减少了SQL语句的编译次数，提高了执行效率。

示意图



## 8.2 PreparedStatement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

## 8.2 PreparedStatement的基本使用

## 8.2.1 API介绍

### 8.2.1.1 获取PreparedStatement的API介绍

在 `java.sql.Connection` 有获取 `PreparedStatement` 对象的方法

```
PreparedStatement preparedStatement(String sql)
```

会先将SQL语句发送给数据库预编译。PreparedStatement对象会引用着预编译后的结果。

### 8.2.1.2 PreparedStatement的API介绍

在 `java.sql.PreparedStatement` 中有设置SQL语句参数，和执行参数化的SQL语句的方法

1. `void setDouble(int parameterIndex, double x)`  
将指定参数设置为给定 Java `double` 值。

2. `void setFloat(int parameterIndex, float x)`  
将指定参数设置为给定 Java `REAL` 值。

3. `void setInt(int parameterIndex, int x)`  
将指定参数设置为给定 Java `int` 值。

4. `void setLong(int parameterIndex, long x)`  
将指定参数设置为给定 Java `long` 值。

5. `void setObject(int parameterIndex, Object x)`  
使用给定对象设置指定参数的值。

6. `void setString(int parameterIndex, String x)`  
将指定参数设置为给定 Java `String` 值。

7. `ResultSet executeQuery()`  
在此 `PreparedStatement` 对象中执行 SQL 查询，并返回该查询生成的`ResultSet`对象。

8. `int executeUpdate()`  
在此 `PreparedStatement` 对象中执行 SQL 语句，该语句必须是一个 SQL 数据操作语言 (Data Manipulation Language, DML) 语句，比如 `INSERT`、`UPDATE` 或 `DELETE` 语句；或者是无返回内容的 SQL 语句，比如 `DDL` 语句。

## 8.2.2 PreparedStatement使用步骤

1. 编写SQL语句，未知内容使用?占位：`"SELECT * FROM user WHERE name=? AND password=?;"`
2. 获得PreparedStatement对象
3. 设置实际参数
4. 执行参数化SQL语句
5. 关闭资源



## 8.2.3 案例代码

```
public class Demo08 {

    public static void main(String[] args) throws Exception {
        // 获取连接
        Connection conn = JDBCUtils.getConnection();

        // 编写SQL语句，未知内容使用?占位
        String sql = "SELECT * FROM user WHERE name=? AND password=?";

        // preparedStatement()会先将SQL语句发送给数据库预编译。
        PreparedStatement pstmt = conn.prepareStatement(sql);

        // 指定?的值
        // parameterIndex: 第几个?, 从1开始算
        // x: 具体的值
        pstmt.setString(1, "admin");
        pstmt.setString(2, "123"); // 正确的密码
        // pstmt.setString(2, "6666"); // 错误的密码

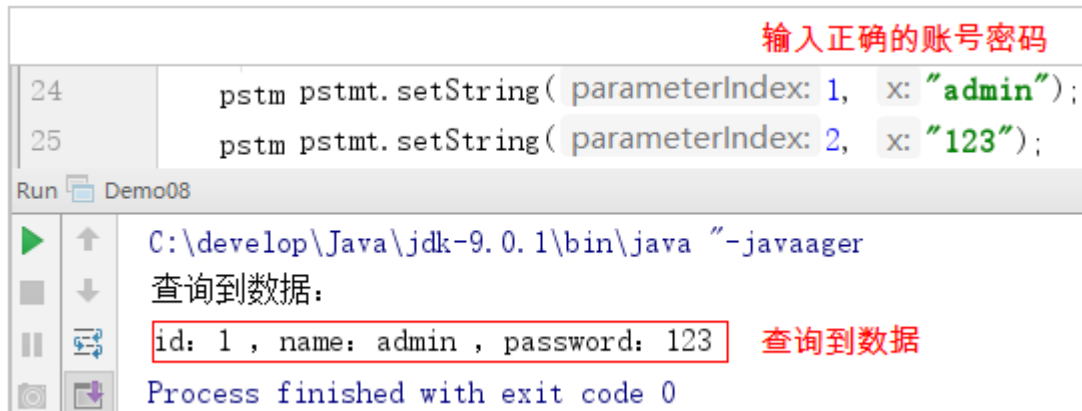
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            String name = rs.getString("name");
            System.out.println("name : " + name);
        } else {
            System.out.println("没有找到数据...");
        }

        JDBCUtils.close(conn, pstmt, rs);
    }
}
```

## 8.2.4 案例效果

1. 输入正确的账号密码：



2. 输入错误的密码：

```
24      pstmt pstmt.setString( parameterIndex: 1, x: "admin");
25      pstmt pstmt.setString( parameterIndex: 2, x: "6666");
```

Run Demo08

C:\develop\Java\jdk-9.0.1\bin\java "-javaager  
没有找到数据...

错误的密码找不到数据

Process finished with exit code 0

## 8.4 PreparedStatement实现增删查改

### 8.4.1 添加数据

向Employee表添加3条记录

```
// 添加数据：向Employee表添加3条记录
public static void addEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();
    String sql = "INSERT INTO employee VALUES (NULL, ?, ?, ?)";
    // preparedStatement()会先将SQL语句发送给数据库预编译。
    PreparedStatement pstmt = conn.prepareStatement(sql);

    // 设置参数
    pstmt.setString(1, "刘德华");
    pstmt.setInt(2, 57);
    pstmt.setString(3, "香港");
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    // 再次设置参数
    pstmt.setString(1, "张学友");
    pstmt.setInt(2, 55);
    pstmt.setString(3, "澳门");
    i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    // 再次设置参数
    pstmt.setString(1, "黎明");
    pstmt.setInt(2, 52);
    pstmt.setString(3, "香港");
    i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    JDBCUtils.close(conn, pstmt);
}
```

效果：

id	name	age	address
1	刘德华	57	香港
2	张学友	55	澳门
3	黎明	52	香港

## 8.4.2 修改数据

将id为2的学生地址改成台湾

```
// 修改数据：将id为2的学生地址改成台湾
public static void updateEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();

    String sql = "UPDATE employee SET address=? WHERE id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, "台湾");
    pstmt.setInt(2, 2);
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    JDBCUtils.close(conn, pstmt);
}
```

效果：

id	name	age	address
1	刘德华	57	香港
2	张学友	55	台湾
3	黎明	52	香港

## 8.4.3 删除数据

删除id为2的员工

```
// 删除数据：删除id为2的员工
public static void deleteEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();

    String sql = "DELETE FROM employee WHERE id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, 2);
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    JDBCUtils.close(conn, pstmt);
}
```

效果：

id	name	age	address
1	刘德华	57	香港
3	黎明	52	香港

## 8.4.4 查询数据

查询id小于8的员工信息,并保存到员工类中

```
public class Employee {
    private int id;
    private String name;
    private int age;
    private String address;
    public Employee() {
    }

    public Employee(int id, String name, int age, String address) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.address = address;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

```

    }

    @Override
    public String toString() {
        return "Employee2 [id=" + id + ", name=" + name + ", age=" + age + ", address=" +
address + "]\n";
    }
}

```

```

// 查询数据: 查询id小于8的员工信息,并保存到员工类中
public static void queryEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();
    String sql = "SELECT * FROM employee WHERE id<?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, 8);
    ResultSet rs = pstmt.executeQuery();

    // 创建集合存放多个Employee2对象
    ArrayList<Employee> list = new ArrayList<>();

    while (rs.next()) {
        // 移动到下一行有数据,取出这行数据
        int id = rs.getInt("id");
        String name = rs.getString("name");
        int age = rs.getInt("age");
        String address = rs.getString("address");

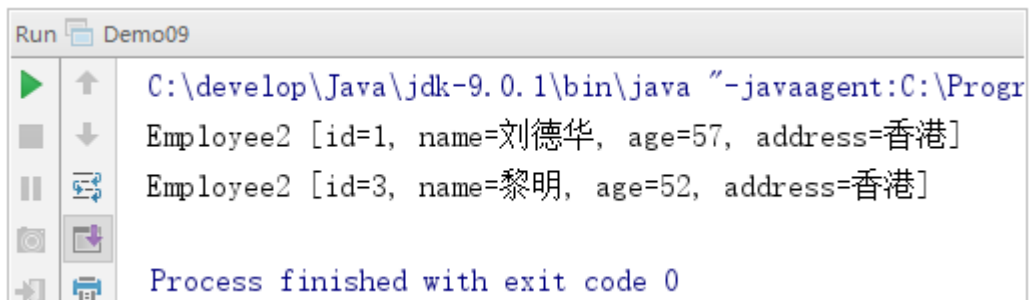
        // 创建Employee2对象
        Employee e = new Employee(id, name, age, address);
        // 将创建好的员工添加到集合中
        list.add(e);
    }

    // 输出对象
    for (Employee e : list) {
        System.out.println(e);
    }

    JDBCUtils.close(conn, pstmt, rs);
}

```

效果：



```

Run Demo09
C:\develop\Java\jdk-9.0.1\bin\java "-javaagent:C:\Progr
Employee2 [id=1, name=刘德华, age=57, address=香港]
Employee2 [id=3, name=黎明, age=52, address=香港]
Process finished with exit code 0

```

## 第9章 案例：PreparedStatement改造登录案例

### 9.1 案例需求

模拟用户输入账号和密码登录网站，防止SQL注入

### 9.2 案例效果

1. 输入正确的账号，密码，显示登录成功

请输入账号：  
**admin**      输入正确的账号，密码  
请输入密码：  
**123**      登录成功  
欢迎您,admin

2. 输入错误的账号，密码，显示登录失败








请输入账号：  
**admin**      输入错误的账号或密码  
请输入密码：  
**aaa**      显示登录失败  
账号或密码错误...

3. 输入 "a' or '1'='1" 作为密码，解决SQL注入：

输入"a' or '1'='1'"作为密码

```
24      pstmt.setString( parameterIndex: 1, x: "admin");
25      pstmt.setString( parameterIndex: 2, x: "a' or '1'='1");
```

Run Demo08

  C:\develop\Java\jdk-9.0.1\bin\java "-javaager  
 没有找到数据...  
  没有找到数据，解决了SQL注入的问题  
  Process finished with exit code 0

### 9.3 案例分析

1. 使用数据库保存用户的账号和密码
2. 让用户输入账号和密码
3. 编写SQL语句，账号和密码部分使用？占位
4. 使用PreparedStatement给？设置参数
5. 使用PreparedStatement执行预编译的SQL语句
6. 如果查询到数据，说明登录成功
7. 如果查询不到数据，说明登录失败

### 9.4 实现步骤

### 1. 编写代码让用户输入账号和密码

```
public class Demo07 {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
    }  
}
```

### 2. 编写SQL语句，账号和密码部分使用？占位，使用PreparedStatement给？设置参数，使用PreparedStatement执行预编译的SQL语句

```
public class Demo11 {  
    public static void main(String[] args) throws Exception {  
        // 让用户输入账号和密码  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
  
        // 获取连接  
        Connection conn = JDBCUtils.getConnection();  
        // 编写SQL语句，账号和密码使用？占位  
        String sql = "SELECT * FROM user WHERE name=? AND password=?";  
        // 获取到PreparedStatement对象  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
        // 设置参数  
        pstmt.setString(1, name);  
        pstmt.setString(2, password);  
        // pstmt.setString(2, "a' or '1'='1");  
    }  
}
```

### 3. 如果查询到数据，说明登录成功，如果查询不到数据，说明登录失败

```
public class Demo11 {  
    public static void main(String[] args) throws Exception {  
        // 让用户输入账号和密码  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
  
        // 获取连接  
        Connection conn = JDBCUtils.getConnection();  
        // 编写SQL语句，账号和密码使用？占位  
        String sql = "SELECT * FROM user WHERE name=? AND password=?";  
        // 获取到PreparedStatement对象
```

```
PreparedStatement pstmt = conn.prepareStatement(sql);
// 设置参数
pstmt.setString(1, name);
pstmt.setString(2, password);
// pstmt.setString(2, "a' or '1'='1");

// 如果查询到数据,说明登录成功,如果查询不到数据,说明登录失败
ResultSet rs = pstmt.executeQuery();

if (rs.next()) {
    // 能进来查询到了数据.
    String name2 = rs.getString("name");
    System.out.println("欢迎您," + name2);
} else {
    // 查询不到数据,说明登录失败
    System.out.println("账号或密码错误...");
}

JDBCUtils.close(conn, pstmt, rs);
}
}
```