JDBC从入门到熟练使用——功能类详解、增删改查(CRUD)、sql注入、事务、连接池

1.jdbc的概念

- JDBC（Java DataBase Connectivity：java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系型数据库提供统一访问，它是由一组用Java语言编写的类和接口组成的。
- JDBC的作用：可以通过java代码操作数据库

2.jdbc的本质

- 其实就是java官方提供的一套规范(接口)。用于帮助开发人员快速实现不同关系型数据库的连接!

3.jdbc的快速入门程序

1. 导入jar包
2. 注册驱动

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

3. 获取连接

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/db2", "root", "root");
```

4. 获取执行者对象

```
Statement stat = conn.createStatement();
```

5. 执行sql语句，并接收返回结果

```
String sql = "SELECT * FROM user";
ResultSet rs = stat.executeQuery(sql);
```

6. 处理结果

```
while(rs.next()) {
    System.out.println(rs.getInt("id") + "\t" + rs.getString("name"));
}
```

7. 释放资源

```
rs.close();
stat.close();
conn.close();
```

8. 创建一个java项目：JDBC基础
9. 将jar包导入，并添加到引用类库
10. 新建com.lichee01.JDBCDemo01

```
public class JDBCDemo01 {
    public static void main(String[] args) throws Exception{
        //1.导入jar包
        //2.注册驱动
        Class.forName("com.mysql.jdbc.Driver");

        //3.获取连接 （连接的数据库名是db2，第二个第三个参数是连接数据库的用户名密码）
        Connection conn =
DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/db2","root","lichee");

        //4.获取执行者对象 （statement：表现，声明，跟程序意思不匹配）
        Statement stat = conn.createStatement();

        //5.执行sql语句，并且接收结果
        String sql = "SELECT * FROM user";
        ResultSet rs = stat.executeQuery(sql); //execute执行，query：查询，resultset：结果
集

        //6.处理结果
        while(rs.next()) {
            System.out.println(rs.getInt("id") + "\t" + rs.getString("name"));
        }

        //7.释放资源
        conn.close();
        stat.close();
        conn.close();
    }
}
```

二、JDBC各个功能类详解

1.DriverManager

- DriverManager：驱动管理对象
    - 注册驱动(告诉程序该使用哪一个数据库驱动)

        - 注册给定的驱动程序：static void registerDriver(Driver driver) （DriverManager的方法）
            - 我们在刚刚的入门案例中并没有注册驱动，也成功了，咋回事呢
        - 这是因为我们使用了Class.forName：Class.forName(“com.mysql.jdbc.Driver”)
            - 我们通过了给forName指定了是mysql的驱动
            - 它会帮助我们注册驱动，如下：
        - 在com.mysql.jdbc.Driver类中存在静态代码块（通过查看源码发现）

```
//这是com.mysql.jdbc.Driver的静态代码块，只要使用这个类，就会执行这段代码
//而Class.forName("com.mysql.jdbc.Driver")就正好使用到了这个类
static {
        try {
                java.sql.DriverManager.registerDriver(new Driver());
        } catch (SQLException E) {
                throw new RuntimeException("Can't register driver!");
        }
}
```

        - **注意**：我们不需要通过DriverManager调用静态方法registerDriver()，因为只要Driver类被使用，则会执行其静态代码块完成注册驱动
        - mysql5之后可以省略注册驱动的步骤。在jar包中，存在一个java.sql.Driver配置文件，文件中指定了com.mysql.jdbc.Driver
        - 所以后边我们其实可以省略注册驱动的步骤（可以注释掉上个案例的注册驱动的步骤，也可以查询到数据）

- 获取数据库连接(获取到数据库的连接并返回连接对象)
    - static Connection getConnection(String url, String user, String password);
        - 返回值：Connection数据库连接对象
        - 参数
            - url：指定连接的路径。语法：**jdbc:mysql://ip地址(域名):端口号/数据库名称**
            - user：用户名
            - password：密码

2.Connection

- Connection：数据库连接对象
    - 获取执行者对象
        - 获取普通执行者对象：Statement createStatement();
        - 获取预编译执行者对象：PreparedStatement prepareStatement(String sql);
    - 管理事务
        - 开启事务：setAutoCommit(boolean autoCommit); 参数为false，则开启事务。
        - 提交事务：commit();
        - 回滚事务：rollback();
    - 释放资源
        - 立即将数据库连接对象释放：void close();

3.Statement

- Statement：执行sql语句的对象
    - 执行DML语句：int executeUpdate(String sql);
        - 返回值int：**返回影响的行数。**
        - 参数sql：可以执行insert、update、delete语句。
    - 执行DQL语句：ResultSet executeQuery(String sql);
        - 返回值ResultSet：**封装查询的结果。**
        - 参数sql：可以执行select语句。
    - 释放资源
        - 立即将执行者对象释放：void close();

4.ResultSet

- ResultSet：结果集对象
    - 判断结果集中是否还有数据：boolean next();
        - 有数据返回true，并将索引向下移动一行
        - 没有数据返回false
    - 获取结果集中的数据：XXX getXxx(“列名”);
        - XXX代表数据类型(要获取某列数据，这一列的数据类型)
        - 例如：String getString(“name”); int getInt(“age”);
    - 释放资源
        - 立即将结果集对象释放：void close();

三、JDBC案例student学生表的CRUD

1. 数据准备

- 数据库和数据表

```
-- 创建student表
CREATE TABLE student(
        sid INT PRIMARY KEY AUTO_INCREMENT,     -- 学生id
        NAME VARCHAR(20),                                       -- 学生姓名
        age INT,                                                        -- 学生年龄
        birthday DATE                                           -- 学生生日
);

-- 添加数据
INSERT INTO student VALUES (NULL,'张三',23,'1999-09-23'),(NULL,'李四',24,'1998-08-10'),(NULL,'王五',25,'1996-06-06'),(NULL,'赵六',26,'1994-10-20');
```

- 实体类
    - Student类，**成员变量对应表中的列**
    - **注意**：所有的基本数据类型需要使用包装类，以防null值无法赋值
        - 数据库的查询结果可能是 null，因为自动拆箱，用基本数据类型接收有 空指针 风险
    - 新建com.lichee02.domain.Student (保存实体类的包名，可以用bean，也可以用domain)

```
public class Student {
    private Integer sid;
    private String name;
    private Integer age;
    private Date birthday;

    //Constructor
    //Getter and Setter
    //toString
}
```

- 本案例会使用分层思想，所以需要新建几个包
- dao：数据持久层：操作数据库

```
//定义dao接口：StudentDao
//本案例要处理五个功能，所以dao接口中定义五个抽象方法
/*
    Dao层接口
 */
public interface StudentDao {
    //查询所有学生信息
    public abstract ArrayList<Student> findAll();
    //条件查询，根据id获取学生信息
    public abstract Student findById(Integer id);
    //新增学生信息
    public abstract int insert(Student stu);
    //修改学生信息
    public abstract int update(Student stu);
    //删除学生信息
    public abstract int delete(Integer id);
}
```

- service：业务层：处理业务逻辑，调用dao

```
//定义service接口：StudentService
/*
    Service层接口
 */
public interface StudentService {
    //查询所有学生信息
    public abstract ArrayList<Student> findAll();
    //条件查询，根据id获取学生信息
    public abstract Student findById(Integer id);
    //新增学生信息
    public abstract int insert(Student stu);
```

```
    //修改学生信息
    public abstract int update(Student stu);
    //删除学生信息
    public abstract int delete(Integer id);
}
```

- controller：控制层，调用业务层方法，将数据返回给前端界面（不过目前我们这个案例没有与界面结合）

```
//定义控制层类：StudentController，提前准备好对应的测试方法
public class StudentController {
    /*
        查询所有学生信息
     */
    @Test
    public void findAll() {
    }

    /*
        条件查询，根据id查询学生信息
     */
    @Test
    public void findById() {
    }

    /*
        添加学生信息
     */
    @Test
    public void insert() {
    }

    /*
        修改学生信息
     */
    @Test
    public void update() {
    }

    /*
        删除学生信息
     */
    @Test
    public void delete() {

    }
}
```

2. 需求一：查询全部（查询所有学生信息）

- 持久层：新建StudentDaoImpl实现StudentDao，并重写所有方法，我们先来处理第一个逻辑：

```
/*
     查询所有学生信息
*/
@Override
public ArrayList<Student> findAll() {
    ArrayList<Student> list = new ArrayList<>();
    Connection conn = null;
    Statement stat = null;
    ResultSet rs = null;
    try{
        //1.注册驱动
        Class.forName("com.mysql.jdbc.Driver");

        //2.获取数据库连接
        conn = DriverManager.getConnection("jdbc:mysql://192.168.59.129:3306/db14", "root",
"lichee");

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        String sql = "SELECT * FROM student";
        rs = stat.executeQuery(sql);

        //5.处理结果集
        while(rs.next()) {
            Integer sid = rs.getInt("sid");
            String name = rs.getString("name");
            Integer age = rs.getInt("age");
            Date birthday = rs.getDate("birthday");

            //封装Student对象
            Student stu = new Student(sid,name,age,birthday);

            //将student对象保存到集合中
            list.add(stu);
        }

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
```

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

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

        if(rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    //将集合对象返回
    return list;
}
```

- 业务层：新建StudentServiceImpl，实现StudentService，重写所有方法，然后先处理第一个逻辑：

```
private StudentDao dao = new StudentDaoImpl();
/*
    查询所有学生信息
*/
@Override
public ArrayList<Student> findAll() {
    return dao.findAll();
}
```

- 控制层

```
private StudentService service = new StudentServiceImpl();
/*
    查询所有学生信息
*/
@Test
```

```
public void findAll() {
    ArrayList<Student> list = service.findAll();
    for(Student stu : list) {
        System.out.println(stu);
    }
}
```

3. 需求二：条件查询（根据id查询学生信息）

- 持久层

```
/*
    条件查询，根据id查询学生信息
*/
@Override
public Student findById(Integer id) {
    Student stu = new Student();
    Connection conn = null;
    Statement stat = null;
    ResultSet rs = null;
    try{
        //1.注册驱动
        Class.forName("com.mysql.jdbc.Driver");

        //2.获取数据库连接
        conn = DriverManager.getConnection("jdbc:mysql://192.168.59.129:3306/db14", "root",
"lichee");

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        String sql = "SELECT * FROM student WHERE sid='"+id+"'";
        rs = stat.executeQuery(sql);

        //5.处理结果集
        while(rs.next()) {
            Integer sid = rs.getInt("sid");
            String name = rs.getString("name");
            Integer age = rs.getInt("age");
            Date birthday = rs.getDate("birthday");

            //封装Student对象
            stu.setSid(sid);
            stu.setName(name);
            stu.setAge(age);
            stu.setBirthday(birthday);
```

```
        }

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        if(conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

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

        if(rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    //将对象返回
    return stu;
}
```

- 业务层

```
/*
    条件查询，根据id查询学生信息
*/
@Override
public Student findById(Integer id) {
    return dao.findById(id);
}
```

- 控制层

```
/*
    条件查询，根据id查询学生信息
*/
@Test
public void findById() {
    Student stu = service.findById(3);
    System.out.println(stu);
}
```

4.需求三：新增数据 （添加学生）

- 持久层

```
/*
      添加学生信息
*/
@Override
public int insert(Student stu) {
    Connection conn = null;
    Statement stat = null;
    int result = 0;
    try{
        //1.注册驱动
        Class.forName("com.mysql.jdbc.Driver");

        //2.获取数据库连接
        conn = DriverManager.getConnection("jdbc:mysql://192.168.59.129:3306/db14", "root",
"lichee");

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        Date d = stu.getBirthday();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String birthday = sdf.format(d);
        String sql = "INSERT INTO student VALUES
('"+stu.getSid()+"','"+stu.getName()+"','"+stu.getAge()+"','"+birthday+"')";
        result = stat.executeUpdate(sql);

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        if(conn != null) {
            try {
                conn.close();
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if(stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    //将结果返回
    return result;
}
```

- 业务层

```
/*
    新增学生信息
*/
@Override
public int insert(Student stu) {
    return dao.insert(stu);
}
```

- 控制层

```
/*
        新增学生信息
*/
@Test
public void insert() {
    Student stu = new Student(5,"周七",27,new Date());
    int result = service.insert(stu);
    if(result != 0) {
        System.out.println("新增成功");
    }else {
        System.out.println("新增失败");
    }
}
```

5. 需求四：修改数据（修改学生信息）

- 持久层

```
/*
    修改学生信息
*/
@Override
public int update(Student stu) {
    Connection conn = null;
    Statement stat = null;
    int result = 0;
    try{
        //1.注册驱动
        Class.forName("com.mysql.jdbc.Driver");

        //2.获取数据库连接
        conn = DriverManager.getConnection("jdbc:mysql://192.168.59.129:3306/db14", "root",
"lichee");

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        Date d = stu.getBirthday();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String birthday = sdf.format(d);
        String sql = "UPDATE student SET
sid='"+stu.getSid()+"',name='"+stu.getName()+"',age='"+stu.getAge()+"',birthday='"+birthday+"'
WHERE sid='"+stu.getSid()+"'";
        result = stat.executeUpdate(sql);

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        if(conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if(stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
```

```
        }
    }
    //将结果返回
    return result;
}
```

- 业务层

```
/*
    修改学生信息
*/
@Override
public int update(Student stu) {
    return dao.update(stu);
}
```

- 控制层

```
/*
    修改学生信息
*/
@Test
public void update() {
    Student stu = service.findById(5);
    stu.setName("周七七");

    int result = service.update(stu);
    if(result != 0) {
        System.out.println("修改成功");
    }else {
        System.out.println("修改失败");
    }
}
```

6. 需求五：删除数据（删除学生）

- 持久层

```
/*
    删除学生信息
*/
@Override
public int delete(Integer id) {
    Connection conn = null;
    Statement stat = null;
```

```
    int result = 0;
    try{
        //1.注册驱动
        Class.forName("com.mysql.jdbc.Driver");

        //2.获取数据库连接
        conn = DriverManager.getConnection("jdbc:mysql://192.168.59.129:3306/db14", "root",
"lichee");

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        String sql = "DELETE FROM student WHERE sid='"+id+"'";
        result = stat.executeUpdate(sql);

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        if(conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if(stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    //将结果返回
    return result;
}
```

- 业务层

```
/*
    删除学生信息
*/
@Override
```

```
public int delete(Integer id) {
    return dao.delete(id);
}
```

- 控制层

```
/*
    删除学生信息
*/
@Test
public void delete() {
    int result = service.delete(5);

    if(result != 0) {
        System.out.println("删除成功");
    }else {
        System.out.println("删除失败");
    }
}
```

四、JDBC工具类

- 为啥需要工具类呢?
- 因为在上个案例中的dao层的代码，很多都是重复的
- 程序员一但碰到重复代码，就要想办法解决

1.工具类的抽取

- 配置文件(在src下创建config.properties)

```
driverClass=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/db1
username=root
password=root
```

- 工具类：com.lichee02.utils.JDBCUtils

```
/*
    JDBC工具类
 */
public class JDBCUtils {
    //1.私有构造方法
    private JDBCUtils(){};

    //2.声明配置信息变量
    private static String driverClass;
```

```
    private static String url;
    private static String username;
    private static String password;
    private static Connection conn;

    //3.静态代码块中实现加载配置文件和注册驱动
    static{
        try{
            //通过类加载器返回配置文件的字节流
            InputStream is =
JDBCUtils.class.getClassLoader().getResourceAsStream("config.properties");

            //创建Properties集合，加载流对象的信息
            Properties prop = new Properties();
            prop.load(is);

            //获取信息为变量赋值
            driverClass = prop.getProperty("driverClass");
            url = prop.getProperty("url");
            username = prop.getProperty("username");
            password = prop.getProperty("password");

            //注册驱动
            Class.forName(driverClass);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //4.获取数据库连接的方法
    public static Connection getConnection() {
        try {
            conn = DriverManager.getConnection(url,username,password);
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return conn;
    }

    //5.释放资源的方法
    public static void close(Connection conn, Statement stat, ResultSet rs) {
        if(conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
```

```
            }
        }

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

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

    public static void close(Connection conn, Statement stat) {
        close(conn,stat,null);
    }
}
```


2.使用工具类优化student表的CRUD

- 查询全部：修改StudentDaoImpl 中的方法

```
/*
    查询所有学生信息
*/
@Override
public ArrayList<Student> findAll() {
    ArrayList<Student> list = new ArrayList<>();
    Connection conn = null;
    Statement stat = null;
    ResultSet rs = null;
    try{

        conn = JDBCUtils.getConnection();

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
```

```
        String sql = "SELECT * FROM student";
        rs = stat.executeQuery(sql);

        //5.处理结果集
        while(rs.next()) {
            Integer sid = rs.getInt("sid");
            String name = rs.getString("name");
            Integer age = rs.getInt("age");
            Date birthday = rs.getDate("birthday");

            //封装Student对象
            Student stu = new Student(sid,name,age,birthday);

            //将student对象保存到集合中
            list.add(stu);
        }

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        JDBCUtils.close(conn,stat,rs);
    }
    //将集合对象返回
    return list;
}
```

- 条件查询

```
/*
    条件查询，根据id查询学生信息
*/
@Override
public Student findById(Integer id) {
    Student stu = new Student();
    Connection conn = null;
    Statement stat = null;
    ResultSet rs = null;
    try{

        conn = JDBCUtils.getConnection();

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
```

```
        String sql = "SELECT * FROM student WHERE sid='"+id+"'";
        rs = stat.executeQuery(sql);

        //5.处理结果集
        while(rs.next()) {
            Integer sid = rs.getInt("sid");
            String name = rs.getString("name");
            Integer age = rs.getInt("age");
            Date birthday = rs.getDate("birthday");

            //封装Student对象
            stu.setSid(sid);
            stu.setName(name);
            stu.setAge(age);
            stu.setBirthday(birthday);
        }

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        JDBCUtils.close(conn,stat,rs);
    }
    //将对象返回
    return stu;
}
```

- 新增数据

```
/*
     添加学生信息
*/
@Override
public int insert(Student stu) {
    Connection conn = null;
    Statement stat = null;
    int result = 0;
    try{
        conn = JDBCUtils.getConnection();

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        Date d = stu.getBirthday();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
```

```
        String birthday = sdf.format(d);
        String sql = "INSERT INTO student VALUES
('"+stu.getSid()+"','"+stu.getName()+"','"+stu.getAge()+"','"+birthday+"')";
        result = stat.executeUpdate(sql);

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        JDBCUtils.close(conn,stat);
    }
    //将结果返回
    return result;
}
```


- 修改数据

```
/*
     修改学生信息
*/
@Override
public int update(Student stu) {
    Connection conn = null;
    Statement stat = null;
    int result = 0;
    try{
        conn = JDBCUtils.getConnection();

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        Date d = stu.getBirthday();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String birthday = sdf.format(d);
        String sql = "UPDATE student SET
sid='"+stu.getSid()+"',name='"+stu.getName()+"',age='"+stu.getAge()+"',birthday='"+birthday+"'
WHERE sid='"+stu.getSid()+"'";
        result = stat.executeUpdate(sql);

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        JDBCUtils.close(conn,stat);
    }
```

```
    //将结果返回
    return result;
}
```

- 删除数据

```
/*
   删除学生信息
*/
@Override
public int delete(Integer id) {
    Connection conn = null;
    Statement stat = null;
    int result = 0;
    try{
        conn = JDBCUtils.getConnection();

        //3.获取执行者对象
        stat = conn.createStatement();

        //4.执行sql语句，并且接收返回的结果集
        String sql = "DELETE FROM student WHERE sid='"+id+"'";
        result = stat.executeUpdate(sql);

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        //6.释放资源
        JDBCUtils.close(conn,stat);
    }
    //将结果返回
    return result;
}
```

五、SQL注入攻击

1.sql注入攻击的演示

- sql注入攻击：就是利用sql语句的漏洞来对系统进行攻击
- 分析：
    - 登陆的时候，会调用UserDaoImpl .findByLoginNameAndPassword 方法
    - 此方法中的sql语句如下

```
//2.定义SQL语句
String sql = "SELECT * FROM user WHERE loginname='"+loginName+"' AND
password='"+password+"'";
System.out.println(sql);
//3.获取操作对象，执行sql语句，获取结果集
st = conn.createStatement();
rs = st.executeQuery(sql);
//4.获取结果集
if (rs.next()) {
    //5.封装
    user = new User();
    user.setUid(rs.getString("uid"));
    ....
}
//6.返回
return user;
```

- 输入aaa和aaa' or '1'='1 (注意引号)最终sql拼接完变为：

```
SELECT * FROM user WHERE loginname='aaa' AND password='aaa' or '1'='1'
```

- 这样的sql语句，会将user表中所有数据都查询出来
- 所以最终查询到用户了，就登陆进去了

2.sql注入攻击的原理

- 按照正常道理来说，我们在密码处输入的所有内容，都应该认为是密码的组成
- 但是现在Statement对象在执行sql语句时，将一部分内容当做查询条件来执行了

3.PreparedStatement的介绍

- 预编译sql语句的执行者对象。在执行sql语句之前，将sql语句进行提前编译
- 明确sql语句的格式后，就不会改变了。剩余的内容都会认为是参数
- 参数使用?作为占位符
- 为参数赋值的方法：setXxx(参数1,参数2);
  - 参数1：?的位置编号(编号从1开始)
  - 参数2：?的实际参数
- 执行sql语句的方法
  - 执行insert、update、delete语句：int executeUpdate();
  - 执行select语句：ResultSet executeQuery();

4.PreparedStatement的使用

```
/*
        使用PreparedStatement的登录方法，解决注入攻击
*/
```

```
@Override
public User findByLoginNameAndPassword(String loginName, String password) {
    //定义必要信息
    Connection conn = null;
    PreparedStatement pstm = null;
    ResultSet rs = null;
    User user = null;
    try {
        //1.获取连接
        conn = JDBCUtils.getConnection();
        //2.创建操作SQL对象
        String sql = "SELECT * FROM user WHERE loginname=? AND password=?";
        pstm = conn.prepareStatement(sql);
        //3.设置参数
        pstm.setString(1,loginName);//设置第一个?参数
        pstm.setString(2,password);//设置第二个?参数
        System.out.println(sql);
        //4.执行sql语句，获取结果集
        rs = pstm.executeQuery();
        //5.获取结果集
        if (rs.next()) {
            //6.封装
            user = new User();
            user.setUid(rs.getString("uid"));
            user.setUcode(rs.getString("ucode"));
            user.setUsername(rs.getString("username"));
            user.setPassword(rs.getString("password"));
            user.setGender(rs.getString("gender"));
            user.setDutydate(rs.getDate("dutydate"));
            user.setBirthday(rs.getDate("birthday"));
            user.setLoginname(rs.getString("loginname"));
        }
        //7.返回
        return user;
    }catch (Exception e){
        throw new RuntimeException(e);
    }finally {
        JDBCUtils.close(conn,pstm,rs);
    }
}
```


六、JDBC管理事务

1. 介绍

1. JDBC如何管理事务

- 管理事务的功能类：Connection
- 开启事务：setAutoCommit(boolean autoCommit); 参数为false，则开启事务。
- 提交事务：commit();
- 回滚事务：rollback();

2. 演示

- 演示批量添加数据并在业务层管理事务
- **注意**：事务的管理需要在业务层实现，因为dao层的功能要给很多模块提供功能的支撑，而有些模块是不需要事务的。
- 批量添加三个用户

3. 具体操作

- UserService 接口

```
/*
         批量添加 ：  batch：一批，成批处理
*/
void batchAdd(List<User> users);
```

- UserServiceImpl 实现类：事务需要在service添加

```
/*
        事务要控制在此处
*/
@Override
public void batchAdd(List<User> users) {
    //获取数据库连接
    Connection connection = JDBCUtils.getConnection();
    try {
        //开启事务
        connection.setAutoCommit(false);
        for (User user : users) {
            //1.创建ID,并把UUID中的-替换
            String uid = UUID.randomUUID().toString().replace("-", "").toUpperCase();
            //2.给user的uid赋值
            user.setUid(uid);
            //3.生成员工编号
            user.setUcode(uid);

            //模拟异常
            //int n = 1 / 0;

            //4.保存
            userDao.save(connection,user);
```

```
        }
        //提交事务
        connection.commit();
    }catch (Exception e){
        try {
            //回滚事务
            connection.rollback();
        }catch (Exception ex){
            ex.printStackTrace();
        }
        e.printStackTrace();
    }finally {
        JDBCUtils.close(connection,null,null);
    }
}
```

- UserDao 接口

```
/**
        支持事务的添加
*/
void save(Connection connection,User user);
```

- UserDaoImpl 实现类

```
/*
        支持事务的添加
*/
@Override
public void save(Connection connection, User user) {
    //定义必要信息
    PreparedStatement pstm = null;
    try {
        //1.获取连接
        connection = JDBCUtils.getConnection();
        //2.获取操作对象
        pstm = connection.prepareStatement("insert into
user(uid,ucode,loginname,password,username,gender,birthday,dutydate)values(?,?,?,?,?,?,?,?)");
        //3.设置参数
        pstm.setString(1,user.getUid());
        pstm.setString(2,user.getUcode());
        pstm.setString(3,user.getLoginname());
        pstm.setString(4,user.getPassword());
        pstm.setString(5,user.getUsername());
        pstm.setString(6,user.getGender());
        pstm.setDate(7,new Date(user.getBirthday().getTime()));
```

```
            pstm.setDate(8,new Date(user.getDutydate().getTime()));
            //4.执行sql语句，获取结果集
            pstm.executeUpdate();
        }catch (Exception e){
            throw new RuntimeException(e);
        }finally {
            JDBCUtils.close(null,pstm,null);
        }
    }
```

七、数据库连接池

1. 数据库连接池的概念

- 数据库连接背景
  - 数据库连接是一种关键的、有限的、昂贵的资源，这一点在多用户的网页应用程序中体现得尤为突出
  - 对数据库连接的管理能显著影响到整个应用程序的伸缩性和健壮性，影响到程序的性能指标
  - 数据库连接池正是针对这个问题提出来的
- 数据库连接池
  - 数据库连接池负责分配、管理和释放数据库连接
  - 它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个
  - 这项技术能明显提高对数据库操作的性能
- 数据库连接池原理
  - 之前的程序：来一个访问就会创建一个连接，使用完了关闭连接。频繁的创建连接和关闭连接是非常耗时的。
  - 优化后的程序：提前准备一些数据库连接，使用的时候从池中获取，用完后重新归还给池中。

2. 自定义连接池

- java.sql.DataSource接口：数据源(数据库连接池)。**java官方提供的数据库连接池规范(接口)**
- 如果想完成数据库连接池技术，就必须实现 **DataSource** 接口
- 核心功能：获取数据库连接对象：**Connection getConnection();**
- 自定义连接池：
  1. 定义一个类，实现 DataSource 接口。
  2. 定义一个容器，用于保存多个 Connection 连接对象。
  3. 定义静态代码块，通过 JDBC 工具类获取 10 个连接保存到容器中。
  4. 重写 getConnection 方法，从容器中获取一个连接并返回。
  5. 定义 getSize 方法，用于获取容器的大小并返回。
- 具体实现：
  - 新建java项目jdbc高级，新建libs目录，添加mysql驱动jar包，并且添加引用库
  - 将上个项目的config.properties复制到src中
  - 将上个项目的JDBCUtils复制到com.lichee.utils中
  - 新建数据库连接池类： com.lichee01.MyDataSource，代码如下：

```
/*
        自定义连接池类
```

```
*/
public class MyDataSource implements DataSource{
    //定义集合容器，用于保存多个数据库连接对象 （创建的是一个线程安全的ArrayList）
    private static List<Connection> pool = Collections.synchronizedList(new ArrayList<Connection>
());

    //静态代码块，生成10个数据库连接保存到集合中
    static {
        for (int i = 0; i < 10; i++) {
            Connection conn = JDBCUtils.getConnection();
            pool.add(conn);
        }
    }

    //返回连接池的大小
    public int getSize() {
        return pool.size();
    }

    //从池中返回一个数据库连接
    @Override
    public Connection getConnection() {
        if(pool.size() > 0) {
            //从池中获取数据库连接
            return pool.remove(0); //注意：从集合中移除一个，并返回（我们每次都是从集合中拿走一个
去用，所以是remove，这个方法正好是能够返回移除的这个对象）
        }else {
            throw new RuntimeException("连接数量已用尽");
        }
    }

    //剩下的其他方法，先不用具体处理
}
```


3. 自定义连接池测试

- 测试：通过自定义数据库连接池完成查询学生表的全部信息
- 具体代码：新建com.lichee01.MyDataSourceTest

```
public class MyDataSourceTest {
    public static void main(String[] args) throws Exception{
        //创建数据库连接池对象
        MyDataSource dataSource = new MyDataSource();

        System.out.println("使用之前连接池数量：" + dataSource.getSize());
```

```
        //获取数据库连接对象
        Connection conn = dataSource.getConnection();
        System.out.println(conn.getClass());// JDBC4Connection

        //查询学生表全部信息
        String sql = "SELECT * FROM student";
        PreparedStatement pst = conn.prepareStatement(sql);
        ResultSet rs = pst.executeQuery();

        while(rs.next()) {
            System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" +
rs.getInt("age") + "\t" + rs.getDate("birthday"));
        }

        //释放资源
        rs.close();
        pst.close();
                //目前的连接对象close方法，是直接关闭连接，而不是将连接归还池中
        conn.close();

        System.out.println("使用之后连接池数量：" + dataSource.getSize());
    }
}
```

4. 归还连接

- 归还方式：
    1. 继承方式
    2. 装饰设计模式
    3. 适配器设计模式
    4. 动态代理方式

4.1 继承(无法解决)

1. 继承方式归还数据库连接的思想

- 通过打印连接对象，发现 DriverManager 获取的连接实现类是 JDBC4Connection
    - JDBC4Connection继承了ConnectionImpl
    - ConnectionImpl实现了MySQLConnection
    - MySQLConnection继承了Connection
    - 所以可以理解：JDBC4Connection就是Connection的一个实现类
- 那我们就可以自定义一个类，继承 JDBC4Connection 这个类，重写 close() 方法，完成连接对象的归还

2. 继承方式归还数据库连接的实现步骤

    1. 定义一个类，继承 JDBC4Connection
    2. 定义 Connection 连接对象和连接池容器对象的成员变量

3. 通过有参构造方法完成对成员变量的赋值
4. 重写 close 方法，将连接对象添加到池中

3. 继承方式归还数据库连接存在的问题

- 通过查看 JDBC 工具类获取连接的方法发现：我们虽然自定义了一个子类，完成了归还连接的操作。
- 但是 DriverManager 获取的还是 JDBC4Connection 这个对象，并不是我们的子类对象，
- 而我们又不能整体去修改驱动包中类的功能，所继承这种方式行不通！
- 这种方式行不通，通过查看JDBC工具类获取连接的方法我们发现：我们虽然自定义了一个子类，完成了归还连接的操作。但是DriverManager获取的还是JDBC4Connection这个对象，并不是我们的子类对象。而我们又不能整体去修改驱动包中类的功能！

4.2 装饰设计模式

1. 装饰设计模式归还数据库连接的思想

- 我们可以自定义一个类，实现 Connection 接口。这样就具备了和 JDBC4Connection 相同的行为了
- 重写 close() 方法，完成连接的归还。其余的功能还调用 mysql 驱动包实现类原有的方法即可
- **然后我们可以使用我们自定义的类对JDBC4Connection对象进行包装**

2. 装饰设计模式归还数据库连接的实现步骤
    1. 定义一个类，实现 Connection 接口
    2. 定义 Connection 连接对象和连接池容器对象的成员变量
    3. 通过有参构造方法完成对成员变量的赋值
    4. 重写 close() 方法，将连接对象添加到池中
    5. 剩余方法，只需要调用 mysql 驱动包的连接对象完成即可
    6. 在自定义连接池中，将获取的连接对象通过自定义连接对象进行包装
3. 装饰设计模式归还数据库连接存在的**问题**
    - **实现 Connection 接口后，有大量的方法需要在自定义类中进行重写**
4. 实现：自定义连接类

```
/*
    自定义Connection类。通过装饰设计模式，实现和mysql驱动包中的Connection实现类相同的功能！
    实现步骤：
        1.定义一个类，实现Connection接口
        2.定义Connection连接对象和连接池容器对象的变量
        3.提供有参构造方法，接收连接对象和连接池对象，对变量赋值
        4.在close()方法中，完成连接的归还
        5.剩余方法，只需要调用mysql驱动包的连接对象完成即可
 */
public class MyConnection2 implements Connection {

    //2.定义Connection连接对象和连接池容器对象的变量
    private Connection conn;
    private List<Connection> pool;

    //3.提供有参构造方法，接收连接对象和连接池对象，对变量赋值
```

```
    public MyConnection2(Connection conn,List<Connection> pool) {
        this.conn = conn;
        this.pool = pool;
    }

    //4.在close()方法中，完成连接的归还
    @Override
    public void close() throws SQLException {
        pool.add(conn);
    }

    //5.剩余方法，只需要调用mysql驱动包的连接对象完成即可
}
```

- 自定义连接池类

```
public class MyDataSource implements DataSource{
    //定义集合容器，用于保存多个数据库连接对象
    private static List<Connection> pool = Collections.synchronizedList(new ArrayList<Connection>
());

    //静态代码块，生成10个数据库连接保存到集合中
    static {
        for (int i = 0; i < 10; i++) {
            Connection conn = JDBCUtils.getConnection();
            pool.add(conn);
        }
    }

    //返回连接池的大小
    public int getSize() {
        return pool.size();
    }

    //从池中返回一个数据库连接
    @Override
    public Connection getConnection() {
        if(pool.size() > 0) {
            //从池中获取数据库连接
            Connection conn = pool.remove(0);
            //通过自定义连接对象进行包装
            MyConnection2 mycon = new MyConnection2(conn,pool);
            //返回包装后的连接对象
            return mycon;
        }else {
            throw new RuntimeException("连接数量已用尽");
```

```
        }
    }
}
```

4.3 适配器设计模式

1. 适配器设计模式归还数据库连接的思想

- 我们可以提供一个适配器类，实现 Connection 接口，将所有方法进行实现(除了close方法)
- 自定义连接类只需要继承这个适配器类，重写需要改进的 close() 方法即可

2. 适配器设计模式归还数据库连接的实现步骤。

    1. 定义一个适配器类，实现 Connection 接口
    2. 定义 Connection 连接对象的成员变量
    3. 通过有参构造方法完成对成员变量的赋值
    4. 重写所有方法(除了 close)，调用mysql驱动包的连接对象完成即可
    5. 定义一个连接类，继承适配器类
    6. 定义 Connection 连接对象和连接池容器对象的成员变量，并通过有参构造进行赋值。
    7. 重写 close() 方法，完成归还连接
    8. 在自定义连接池中，将获取的连接对象通过自定义连接对象进行包装

3. 适配器设计模式归还数据库连接存在的**问题**

    - **自定义连接类虽然很简洁了，但适配器类还是我们自己编写的，也比较的麻烦**

4. 适配器类

```
/*
    适配器抽象类。实现Connection接口。
    实现所有的方法，调用mysql驱动包中Connection连接对象的方法
 */
public abstract class MyAdapter implements Connection {

    // 定义数据库连接对象的变量
    private Connection conn;

    // 通过构造方法赋值
    public MyAdapter(Connection conn) {
        this.conn = conn;
    }

    // 所有的方法，均调用mysql的连接对象实现
}
```

- 自定义连接类

```
/*
    自定义Connection连接类。通过适配器设计模式。完成close()方法的重写
        1.定义一个类，继承适配器父类
        2.定义Connection连接对象和连接池容器对象的变量
        3.提供有参构造方法，接收连接对象和连接池对象，对变量赋值
        4.在close()方法中，完成连接的归还
 */
public class MyConnection3 extends MyAdapter {
    //2.定义Connection连接对象和连接池容器对象的变量
    private Connection conn;
    private List<Connection> pool;

    //3.提供有参构造方法，接收连接对象和连接池对象，对变量赋值
    public MyConnection3(Connection conn,List<Connection> pool) {
        super(conn);    // 将接收的数据库连接对象给适配器父类传递
        this.conn = conn;
        this.pool = pool;
    }

    //4.在close()方法中，完成连接的归还
    @Override
    public void close() throws SQLException {
        pool.add(conn);
    }
}
```

- 自定义连接池类

```
public class MyDataSource implements DataSource{
    //定义集合容器，用于保存多个数据库连接对象
    private static List<Connection> pool = Collections.synchronizedList(new ArrayList<Connection>
());

    //静态代码块，生成10个数据库连接保存到集合中
    static {
        for (int i = 0; i < 10; i++) {
            Connection conn = JDBCUtils.getConnection();
            pool.add(conn);
        }
    }

    //返回连接池的大小
    public int getSize() {
        return pool.size();
    }
```

```
    //从池中返回一个数据库连接
    @Override
    public Connection getConnection() {
        if(pool.size() > 0) {
            //从池中获取数据库连接
            Connection conn = pool.remove(0);

            //通过自定义连接对象进行包装
            //MyConnection2 mycon = new MyConnection2(conn,pool);
            MyConnection3 mycon = new MyConnection3(conn,pool);

            //返回包装后的连接对象
            return mycon;
        }else {
            throw new RuntimeException("连接数量已用尽");
        }
    }
}
```


4.4 动态代理

- 经过我们适配器模式的改进，自定义连接类中的方法已经很简洁了。剩余所有的方法已经抽取到了适配器类中。
- 但是适配器这个类还是我们自己编写的，也比较麻烦！所以可以使用动态代理的方式来改进。
- **动态代理：在不改变目标对象方法的情况下对方法进行增强**
- 组成
    - 被代理对象：真实的对象
    - 代理对象：内存中的一个对象
- 要求
    - 代理对象必须和被代理对象实现相同的接口
- 实现
    - Proxy.newProxyInstance()
- 自定义数据库连接池类

```
public class MyDataSource implements DataSource{
    //定义集合容器，用于保存多个数据库连接对象
    private static List<Connection> pool = Collections.synchronizedList(new ArrayList<Connection>
());

    //静态代码块，生成10个数据库连接保存到集合中
```

```
    static {
        for (int i = 0; i < 10; i++) {
            Connection conn = JDBCUtils.getConnection();
            pool.add(conn);
        }
    }

    //返回连接池的大小
    public int getSize() {
        return pool.size();
    }

    //动态代理方式
    @Override
    public Connection getConnection() {
        if(pool.size() > 0) {
            //从池中获取数据库连接
            Connection conn = pool.remove(0);

            Connection proxyCon =
(Connection)Proxy.newProxyInstance(conn.getClass().getClassLoader(), new Class[]
{Connection.class}, new InvocationHandler() {
                /*
                    执行Connection实现类所有方法都会经过invoke
                    如果是close方法，则将连接还回池中
                    如果不是，直接执行实现类的原有方法
                 */
                @Override
                public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
{
                    if(method.getName().equals("close")) {
                        pool.add(conn);
                        return null;
                    }else {
                        return method.invoke(conn,args);
                    }
                }
            });

            return proxyCon;
        }else {
            throw new RuntimeException("连接数量已用尽");
        }
    }
}
```

5. 开源连接池的使用

5.1 C3P0

- C3P0 数据库连接池的使用步骤：

  1. 导入 jar 包：
     - c3p0-0.9.5.2.jar
     - mchange-commons-java-0.2.12.jar
  2. 导入配置文件到 src 目录下：
     - 修改配置文件中的数据库url，用户名和密码
  3. 创建 C3P0 连接池对象
  4. 获取数据库连接进行使用
  5. 注意：C3P0 的配置文件会自动加载，但是必须叫 c3p0-config.xml 或 c3p0-config.properties 。

- 配置文件：c3p0-config.xml

```
<c3p0-config>
  <!-- 使用默认的配置读取连接池对象 -->
  <default-config>
        <!-- 连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://127.0.0.1:3306/db2</property>
    <property name="user">root</property>
    <property name="password">root</property>

    <!-- 连接池参数 -->
    <property name="initialPoolSize">5</property>
    <property name="maxPoolSize">10</property>
    <property name="checkoutTimeout">3000</property>
  </default-config>
</c3p0-config>
```

- 基本使用：新建com.lichee03.C3P0Test1

```
public class C3P0Test1 {
    public static void main(String[] args) throws Exception{
        //1.创建c3p0的数据库连接池对象
        DataSource dataSource = new ComboPooledDataSource();

        //2.通过连接池对象获取数据库连接
        Connection conn = dataSource.getConnection();

        //3.执行操作
        String sql = "SELECT * FROM student";
        PreparedStatement pst = conn.prepareStatement(sql);

        //4.执行sql语句，接收结果集
```

```
        ResultSet rs = pst.executeQuery();

        //5.处理结果集
        while(rs.next()) {
            System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" +
rs.getInt("age") + "\t" + rs.getDate("birthday"));
        }

        //6.释放资源
        rs.close();
        pst.close();
        conn.close();
    }
}
```

- 配置演示：
    - 在上个案例中，conn.close()是将连接归还了么？我们来测试一下：
    - 新建C3P0Test2

```
public class C3P0Test2 {
    public static void main(String[] args) throws Exception{
        //1.创建c3p0的数据库连接池对象
        DataSource dataSource = new ComboPooledDataSource();

        //2.测试
        for(int i = 1; i <= 11; i++) {
            Connection conn = dataSource.getConnection();
            System.out.println(i + ":" + conn);

            if(i == 5) {//如果没有这段代码，我们循环获取11个连接是会报错的，因为我们配置的
最大连接数量是10
                conn.close();
            }
        }
    }
}
```

- 测试：
    - 添加过if判断之后，就不会报错，然后观察到打印的对象，发现第五个和第六个是一个对象
- 这说明，close之后，连接是还给连接池了，这样第六个才能继续获取新的连接对象

5.2 Druid

- Druid 数据库连接池的使用步骤：

    1. 导入 jar 包：
        - druid-1.0.9\druid-1.0.9.jar
    2. 编写配置文件，放在 src 目录下：
        - 修改url，用户名和密码
    3. 通过 Properties 集合加载配置文件
    4. 通过 Druid 连接池工厂类获取数据库连接池对象
    5. 获取数据库连接进行使用
    6. **注意**：Druid 不会自动加载配置文件，需要我们手动加载，但是文件的名称可以自定义。

- 配置文件：druid.properties

```
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/db2
username=root
password=root
initialSize=5
maxActive=10
maxWait=3000
```

- 基本使用：新建com.lichee04.DruidTest1

```
/*
    1.通过Properties集合，加载配置文件
    2.通过Druid连接池工厂类获取数据库连接池对象
    3.通过连接池对象获取数据库连接进行使用
 */
public class DruidTest1 {
    public static void main(String[] args) throws Exception{
        //获取配置文件的流对象
        InputStream is =
DruidTest1.class.getClassLoader().getResourceAsStream("druid.properties");

        //1.通过Properties集合，加载配置文件
        Properties prop = new Properties();
        prop.load(is);

        //2.通过Druid连接池工厂类获取数据库连接池对象
        DataSource dataSource = DruidDataSourceFactory.createDataSource(prop);

        //3.通过连接池对象获取数据库连接进行使用
        Connection conn = dataSource.getConnection();


        String sql = "SELECT * FROM student";
        PreparedStatement pst = conn.prepareStatement(sql);
```

```
        //4.执行sql语句，接收结果集
        ResultSet rs = pst.executeQuery();

        //5.处理结果集
        while(rs.next()) {
            System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" +
rs.getInt("age") + "\t" + rs.getDate("birthday"));
        }

        //6.释放资源
        rs.close();
        pst.close();
        conn.close();//也是将连接归还
    }
}
```


- 抽取工具类com.lichee.utils.DataSourceUtils

```
/*
    数据库连接池的工具类
 */
public class DataSourceUtils {
    //1.私有构造方法
    private DataSourceUtils(){}

    //2.声明数据源变量
    private static DataSource dataSource;

    //3.提供静态代码块，完成配置文件的加载和获取数据库连接池对象
    static{
        try{
            //完成配置文件的加载
            InputStream is =
DataSourceUtils.class.getClassLoader().getResourceAsStream("druid.properties");

            Properties prop = new Properties();
            prop.load(is);

            //获取数据库连接池对象
            dataSource = DruidDataSourceFactory.createDataSource(prop);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //4.提供一个获取数据库连接的方法
```

```
    public static Connection getConnection() {
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }

    //5.提供一个获取数据库连接池对象的方法
    public static DataSource getDataSource() {
        return dataSource;
    }

    //6.释放资源
    public static void close(Connection conn, Statement stat, ResultSet rs) {
        if(conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

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

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

    public static void close(Connection conn, Statement stat) {
        if(conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
```

```
            }
        }

        if(stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- 使用工具类

```
public class DruidTest2 {
    public static void main(String[] args) throws Exception{
        //1.通过连接池工具类获取一个数据库连接
        Connection conn = DataSourceUtils.getConnection();

        String sql = "SELECT * FROM student";
        PreparedStatement pst = conn.prepareStatement(sql);

        //2.执行sql语句，接收结果集
        ResultSet rs = pst.executeQuery();

        //3.处理结果集
        while(rs.next()) {
            System.out.println(rs.getInt("sid") + "\t" + rs.getString("name") + "\t" +
rs.getInt("age") + "\t" + rs.getDate("birthday"));
        }

        //4.释放资源
        DataSourceUtils.close(conn,pst,rs);
    }
}
```

5.3 C3P0 与Druid的区别

- c3p0是一个开放源代码的JDBC连接池，Hibernate的发行包中默认使用此连接池，性能很好
- Druid是阿里出品，淘宝和支付宝专用数据库连接池，但它不仅仅是一个数据库连接池，它还包含一个ProxyDriver，一系列内置的JDBC组件库，一个 SQL Parser。
    - 支持所有JDBC兼容的数据库，包括Oracle、MySql、Derby、Postgresql、SQL Server、H2等等

- Druid针对Oracle和MySql做了特别优化，比如Oracle的PS Cache内存占用优化，MySql的ping检测优化