

JDBC

1_持久化概述

持久化(persistence): 把数据保存到可掉电式存储设备中以供之后使用。

大多数情况下，特别是企业级应用，数据持久化意味着将内存中的数据保存到硬盘上加以“固化”，而持久化的实现过程大多通过各种关系数据库来完成。就是将内存中的数据存储在关系型数据库中，当然也可以存储在磁盘文件、XML数据文件中。而在 Java 中，数据库存取技术只能通过 JDBC 来访问数据库。

JDBC 访问数据库的形式主要有两种：

1. 直接使用 JDBC 的 API 去访问数据库服务器 (MySQL/Oracle).
2. 间接地使用 JDBC 的 API 去访问数据库服务器.
第三方 O/R Mapping 工具，如 Hibernate, MyBatis 等.(底层依然是 JDBC)
JDBC 是 Java 访问数据库的基石,其他技术都是对 JDBC 的封装.

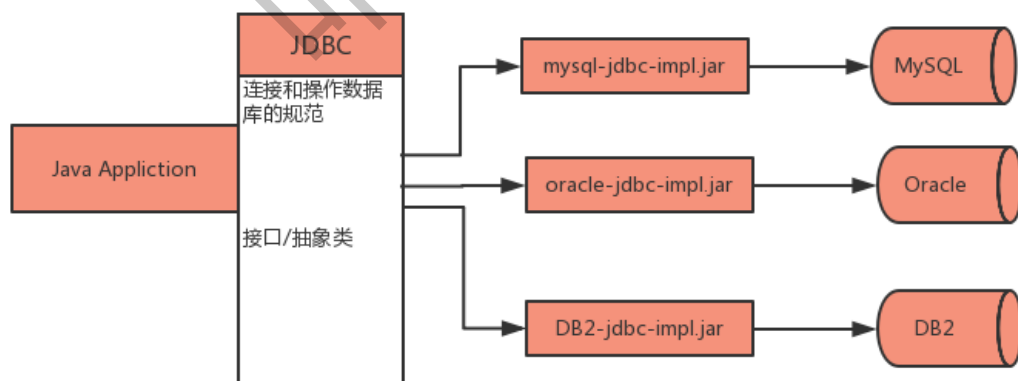
2_JDBC 概述

JDBC (Java DataBase Connectivity)

是一种用于执行 SQL 语句的 Java API，**可以为多种关系数据库提供统一访问**，它由一组用 Java 语言编写的类和接口组成。JDBC 提供了一种基准，使数据库开发人员能够编写数据库应用程序。

JDBC 为访问不同的数据库提供了一种统一的途径，为开发者屏蔽了一些细节问题。

JDBC 的目标是使 Java 程序员使用 JDBC 可以连接任何提供了 JDBC 实现 (驱动程序) 的数据库系统，这样就使得程序员无需对特定的数据库系统的特点有过多的了解，从而大大简化和加快了开发过程。



总结: JDBC 本身是 Java 连接数据库的一个标准,是进行数据库连接的抽象层,由 Java 编写的一组类和接口组成，接口的实现由各个数据库厂商来完成。

3_JDBC的基本操作

3.1_获取数据库连接对象

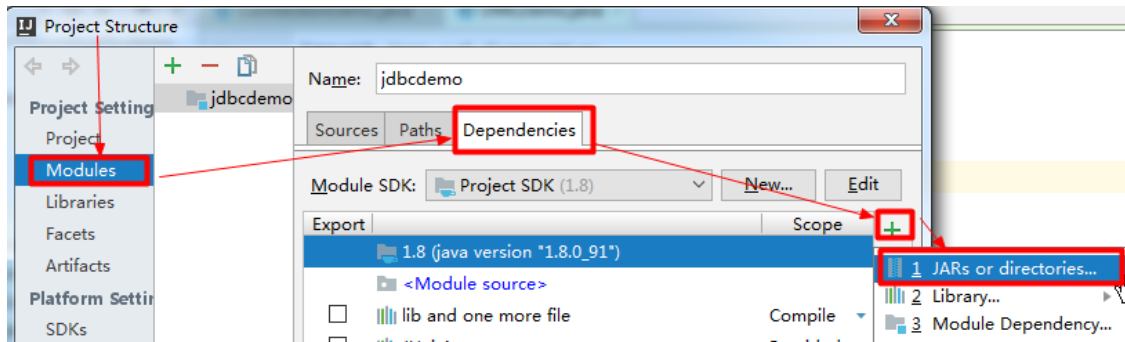
3.1.1_环境准备

1. 拷贝 MySQL 的 JDBC 驱动,到 Java 项目中: mysql-connector-java-5.1.26-bin.jar

注意: 是 jar 包,不是 zip 包.

2. 选择 jar,把 jar 引用到 classpath 路径.

- o idea 项目中创建一个目录 lib
- o File -> Project Structure -> Modules -> Dependencies-> + -> jar or directories -> 选择jar包路劲



JDBC 操作数据库需要和数据库建立关系, 所以第一步需 获取 JDBC 和数据库的连接对象 / Connection对象.

3.1.2_操作步骤(贾琰)

1. 加载注册驱动.

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

上述代码是如何完成注册驱动的?

```
static {
    try {
        java.sql.DriverManager.registerDriver(new Driver());
    } catch (SQLException E) {
        throw new RuntimeException("Can't register driver!");
    }
}
```

1. 把 com.mysql.jdbc.Driver 这一份字节码加载进 JVM.
2. 字节码被加载进JVM,就会执行其静态代码块.而其底层的静态代码块在完成注册驱动工作,将驱动注册到DriverManger 中.

2. 获取连接对象.

使用 DriverManager 的 getConnection 方法创建 Connection 对象

```
Connection conn = DriverManager.getConnection(url,username,password);
// url=jdbc:mysql://localhost:3306/jdbcdemo
// 如果连接的是本机的 MySQL,并且端口是默认的 3306 ,则可以简写:
url=jdbc:mysql:///jdbcdemo
// username:当前访问数据库的用户名
// password:当前访问数据库的密码
```

注意: Java6 开始,JDBC4.0有一个新特性-无需加载注册驱动.

规范要求: JDBC 4.0 的驱动必须包括 META-INF/services/java.sql.Driver 文件。此文件已经包含驱动名称。

程序会自动从 META-INF/services/java.sql.Driver 去读取当前的驱动类的全限定名,所有目前写不写加载注册驱动都没问题,但是web项目必须写上加载注册驱动代码,否则无法连接数据库。

3.2_插入操作

需求: 使用 JDBC 的 API 完成向学生表中插入一条数据.

3.2.1_操作 JDBC 口诀

贾琰欲执事

1. 加载注册驱动.
2. 获取连接对象.
3. 创建/获取语句对象.
4. 执行SQL语句.
5. 释放资源.

3.2.2_实现 API

Connection接口: JDBC 的连接对象.

```
//常用方法:
Statement createStatement(): // 创建一个静态的语句对象.写死的SQL语句
PreparedStatement prepareStatement(String sql); //创建一个预编译语句对象.动态SQL,使用带有占位符(?)的SQL语句的模板.
close(); //释放资源
```

Statement 接口: 执行静态 SQL 语句并返回它所生成结果.

```
// 常用方法:
int executeUpdate(String sql); //执行 DDL 或 DML语句.
// 若当前SQL是DDL语句,则返回0.
// 若当前SQL是DML语句,则返回受影响的行数.
ResultSet executeQuery(String sql); //执行DQL语句,返回结果集.
close(); //释放资源
```

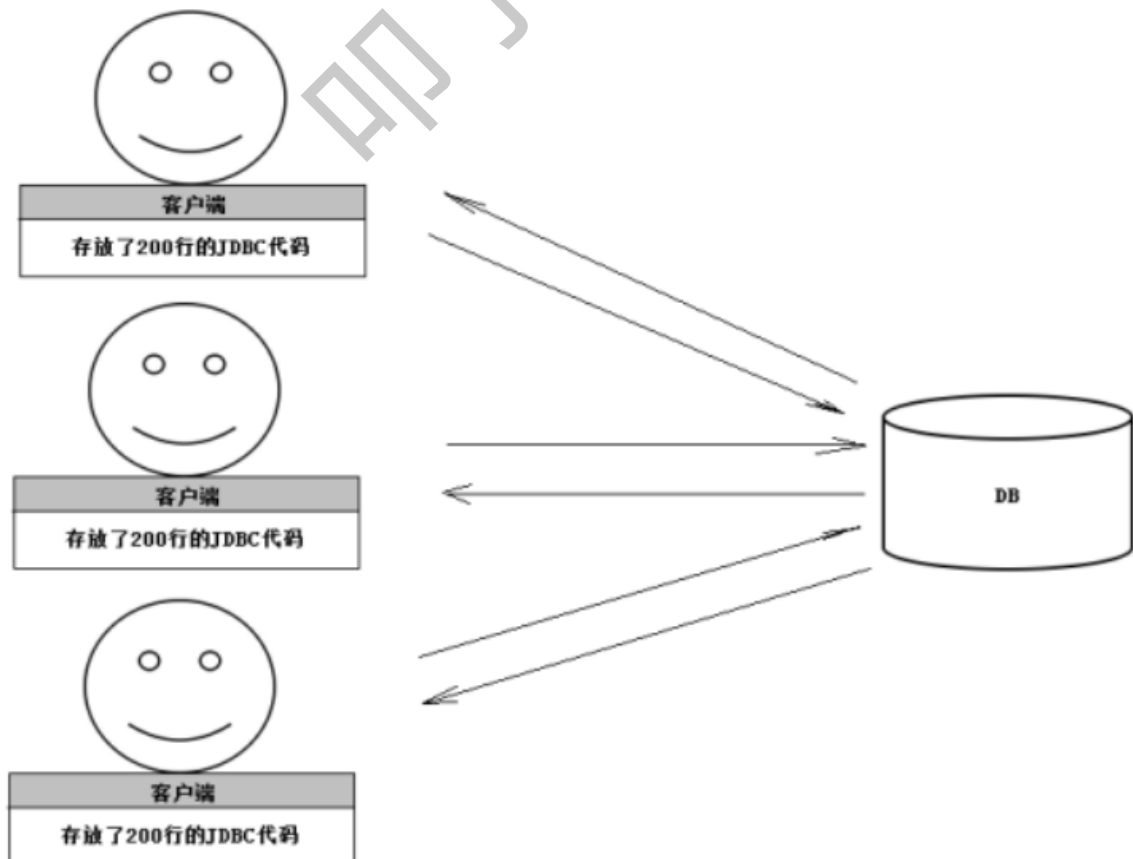
3.2.3_插入实现

```
@Test
public void testInsert() throws Exception {
    String sql = "INSERT INTO t_student(name,email,age)
VALUES('xiaoming2','xiao2@',18)";
    // 贾琰欲执事
    // 1 加载注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 2 获取连接对象
    Connection conn = DriverManager.getConnection("jdbc:mysql:///jdbcdemo",
"root", "admin");
    // 3 获取语句对象
    Statement st = conn.createStatement();
    // 4 执行语句
    st.executeUpdate(sql);
    // 5 释放资源
    st.close();
    conn.close();
}
```

4_DAO 思想

4.1_没有 DAO 的问题

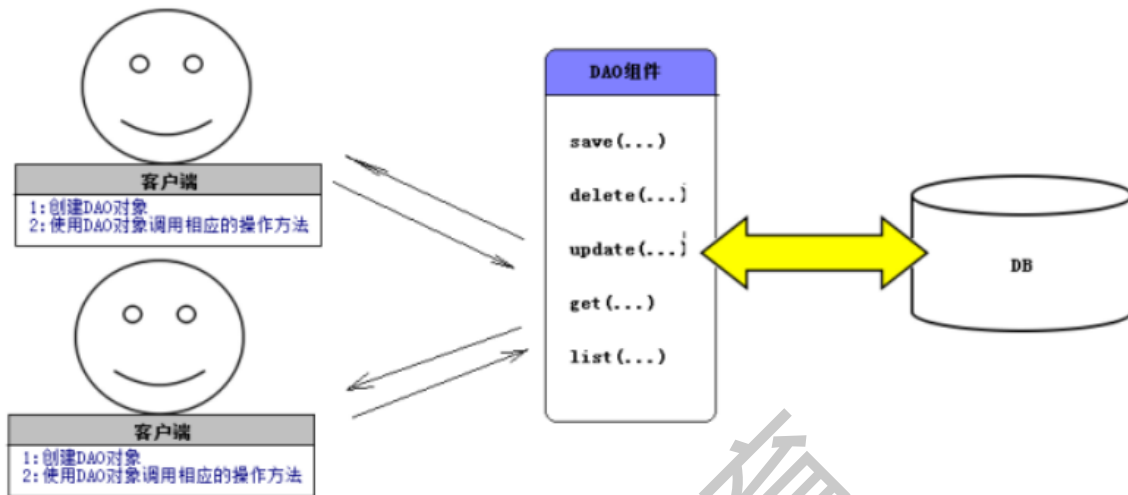
在没有 DAO 的时候,我们的代码存在大量的重复。



4.2_DAO 介绍和方法设计

DAO(Data Access Object) **数据访问对象**是一个**面向对象**的**数据库**接口. 顾名思义就是与数据库打交道, 夹在业务逻辑与数据库资源中间, 将所有对数据源的访问操作抽象封装在一个公共 API 中。程序书写就是建立一个接口, 接口中定义了此应用程序中将会用到的所有事务方法。

DAO 中的主要操作: 增删改查(CRUD). 引入 DAO 之后,此时设计如下图



通过以上图, DAO 作为组件, 那其主要的的是方法的设计, 方法设计需要注意什么呢?

1. 在保存功能中,调用者需要传递多个参数进来,然后把这些数据保存到数据库中
2. 在查询功能中,结果集的每行数据有多个列的值,然后把这些数据返回给调用者

意识: 在开发过程中,如果遇到需要传递的数据有多个的时候,通常需要使用 JavaBean 对其进行封装

最终方法设计如下:

```
//调用者将需要保存的数据封装到Student对象中,然后传递进来
void save(Student stu);
//在查询之后,将每行数据封装到Student对象中,再返回给调用者
Student selectOne(long id);
```

4.3_DAO层开发规范(重要,背)

规范就是王八的屁股--->龟腩(规定)

DAO 其实是一个组件(可以重复使用),包括:

分包规范:

域名倒写.项目模块名.组件;

cn.wolfcode.pss.util; // 存放工具类

cn.wolfcode.pss.domain; //装pss模块的domain类,模型对象.(Student)

cn.wolfcode.pss.dao; //装pss模块的dao接口.

```
cn.wolfcode.pss.dao.impl; //装pss模块的dao接口的实现类.  
cn.wolfcode.pss.test;      //暂时存储DAO的测试类,以后的测试类不应该放这里.
```

命名规范:

以下的 Xxx 表示一个模型对象, 比如 Employee,Department,Student

- DAO 接口: 表示对某个模型的 CRUD 操作做规范, 以 I 开头,interface
 - 标准: IXxxDAO
 - 例: IEmployeeDAO/IStudentDAO
- DAO 实现类: 表示对某个 DAO 接口的实现
 - 标准: XxxDAOImpl
 - 例: EmployeeDAOImpl/StudentDAOImpl
- DAO 测试类: 测试 DAO 组件中的所有方法
 - 标准: XxxDAOTest: XxxDAO 组件的测试类,
 - 例: EmployeeDAOTest,StudentDAOTest

开发建议: 面向接口编程,声明 DAO 对象

传统的做法 : EmployeeDAOImpl dao = new EmployeeDAOImpl();

面向接口编程: IEmployeeDAO dao = new EmployeeDAOImpl();

把实现类赋给接口类型,体现多态的特性:可以屏蔽不同子类之间实现的差异.

5_按照规范搭建项目

5.1_搭建项目规范

步骤:

1. 创建项目
2. 导入数据库驱动包
3. 创建表和模型包以及模型对象 (domain/Student)
4. 创建 DAO 包和 DAO 接口,设计 DAO 接口方法 (dao/IStudentDAO)
5. 创建 DAO 实现包, 实现 DAO 接口(dao.impl/StudentDAOImpl)
6. 创建测试目录, 生成测试类和方法(test/StudentDAOTest)
7. 书写实现, 实现一个方法测试一个方法并且测试通过

5.2_测试现行

```
public class StudentDAOTest {  
    private IStudentDAO studetDAO = new StudentDAOImpl();  
  
    @Test  
    public void insert() {  
        Student stu = new Student(null,"小明","ming@",18);  
        studetDAO.insert(stu);  
    }  
  
    @Test  
    public void delete() {
```

```

        studetDAO.delete(2L);
    }

    @Test
    public void update() {
        Student stu = new Student(1L, "小明", "ming@", 19);
        studetDAO.update(stu);
    }

    @Test
    public void selectOne() {
        Student stu = studetDAO.selectOne(1L);
        System.out.println(stu);
    }

    @Test
    public void selectList() {
        List<Student> list = studetDAO.selectList();
        for (Student stu: list) {
            System.out.println(stu);
        }
    }
}

```

5.3_DAO 之保存操作

```

public void insert(Student stu) {
    String sql = "INSERT INTO t_student(name,email,age) VALUES('" +
        stu.getName() + "','" +
        + stu.getEmail() + "','" +
        + stu.getAge() + "')";
    System.out.println(sql);
    // 贾琰欲执事
    Connection conn = null;
    Statement st = null;
    try {
        // 1 加载注册驱动
        Class.forName("com.mysql.jdbc.Driver");
        // 2 获取连接对象
        conn = DriverManager.getConnection("jdbc:mysql:///jdbcdemo", "root",
            "admin");
        // 3 获取语句对象
        st = conn.createStatement();
        // 4 执行sql 语句
        st.executeUpdate(sql);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (st != null) {
                st.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

```

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

```

在完成保存的过程中,执行 SQL 需要的参数是调用者传递进来的,所以需要将其拼接到 SQL 里面

```

String sql = "INSERT INTO t_student(name,email,age) VALUES('" + stu.getName() +
    "','"+
        + stu.getEmail() + "','"+
        + stu.getAge() + ")";
System.out.println(sql);

```

可以看出,这里的拼接操作极其恶心,容易错,还不容易发现,拼接多了语义很不清晰,所以这里需要优化。(PreparedStatement)

6_预编译语句对象

PreparedStatement 接口: 是 Statement 接口的子接口,享有 Statement 中的方法.

使用的预编译语句对象,sql 语句中使用 ? 来作为值的占位符.

Connection API:

```
PreparedStatement conn对象的.prepareStatement(String sql)
```

PreparedStatement API:

```

// 常用方法:
void setXxx(int parameterIndex,xxx value); //设置第几个占位符的真正参数值.
    // Xxx 表示数据类型,比如 String,int,long,Date等.
void setObject(int parameterIndex, Object x); //设置第几个占位符的真正参数值.

int executeUpdate(); //执行DDL/DML语句. 注意:没有参数
    // 若当前 SQL是 DDL语句,则返回 0.
    // 若当前 SQL是 DML语句,则返回受影响的行数.
ResultSet executeQuery(); //执行DQL语句,返回结果集.
close(); //释放资源

```

有了PreparedStatement 就可以使用占位符? 来代替拼接,这样语义更加清晰,数据设置也很清晰,所以以后都使用 PreparedStatement 预编译语句对象。

调整之后的插入操作:

```
public void insert(Student stu) {
```



```

String sql = "INSERT INTO t_student(name,email,age) VALUES(?,?,?)";
System.out.println(sql);
// 贾琰欲执事
Connection conn = null;
PreparedStatement pst = null;
try {
    // 1 加载注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 2 获取连接对象
    conn = DriverManager.getConnection("jdbc:mysql:///jdbcdemo", "root",
"admin");
    // 3 获取预编译语句对象
    pst = conn.prepareStatement(sql);
    // 给sql 的? 设置数据
    pst.setObject(1,stu.getName());
    pst.setObject(2,stu.getEmail());
    pst.setObject(3,stu.getAge());

    // 4 执行sql 语句
    pst.executeUpdate();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (pst != null) {
            pst.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    try {
        if (conn != null) {
            conn.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
}

```

7_DAO之更新和删除

更新和删除都属于 DML 操作，它们和插入的不同仅仅就 SQL 语句不同而已，其他的都是相同的。

DML 操作模板：

```

//String sql = "INSERT INTO t_student(name,email,age) VALUES(?,?,?)";
//String sql = "DELETE FROM t_student WHERE id=?";
String sql = "UPDATE t_student SET name=?,email=?,age=? WHERE id=?";
// 贾琰欲执事
Connection conn = null;
PreparedStatement pst = null;
try {

```

```

// 1 加载注册驱动
Class.forName("com.mysql.jdbc.Driver");
// 2 获取连接对象
conn = DriverManager.getConnection("jdbc:mysql:///jdbcdemo", "root",
"admin");

// 3 获取预编译语句对象
pst = conn.prepareStatement(sql);
// 给sql 的? 设置数据
// 添加的sql 的? 设置数据
//pst.setString(1, stu.getName());
//pst.setString(2, stu.getEmail());
//pst.setInt(3, stu.getAge());

// 删除的sql 的? 设置数据
// pst.setLong(1,id);

// 修改sql 的? 设置数据
//pst.setString(1,stu.getName());
//pst.setString(2,stu.getEmail());
//pst.setInt(3,stu.getAge());
//pst.setLong(4,stu.getId());

// 4 执行sql 语句
pst.executeUpdate();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (pst != null) {
            pst.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    try {
        if (conn != null) {
            conn.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

在此 DML 操作就完成了，目前就差 DQL 了。

8_DAO之查询操作

DQL 操作和DML 操作的不同之处是 DQL 需要获取返回的数据，而这些数据都被存在一个叫 ResultSet 的对象中

```

ResultSet Statement对象的 executeQuery(String sql); //执行DQL语句,返回结果集.
ResultSet PreparedStatement对象的 executeQuery(); //执行DQL语句,返回结果集.

```

8.1_查询结果对象

ResultSet 接口: 通过执行 DQL 语句查询之后的结果对象。

ResultSet 对象具有指向其当前数据行的光标。最初, 光标被置于第一行之前。next 方法将光标移动到下一行; 因为该方法在 ResultSet 对象没有下一行时返回 false, 所以可以在 while 循环中使用它来迭代结果集

```
// 常用方法:select id,name,age
boolean next(); //判断当前光标是否能向下移动,能向下移动返回true,并同时光标移动到下一行`.

Xxx getXxx(int columnIndex); //取出当前光标所在行的第columnIndex列的数据(columnIndex从1开始算).
Xxx getXxx(String columnName); //取出当前光标所在行的列名为columnName列的数据,columnName可以是别名.
// Xxx 表示数据类型,比如String,int,long,Date等. 推荐使用列名来取数据.

close(); // 释放资源
```

ResultSet 中存的数据则为查询出来的结果, 这个结果就是一张表的结果。

id	name	email	age
1	小明	xiao@	18
2	小东	dong@	18

而我们要做的就是从 ResultSet 中来取出这些数据。

操作步骤

1. 先取行: 默认光标在第一行, 则为标题行 (数据)
使用 ResultSet 的 next 方法完成
2. 再取列: id name age

```
// 取出当前光标所在行的第columnIndex列的数据(columnIndex 从 1 开始算).
Xxx getXxx(int columnIndex); // 索引可能会变化,所以不建议使用
// 取出当前光标所在行的列名为columnName列的数据,columnName可以是别名.(推荐)
Xxx getXxx(String columnName);
```

8.2_查询实现

查询单个学生的信息

```
public Student selectOne(Long id)
{
    String sql = "SELECT * FROM t_student WHERE id=?";
    Connection conn = null;
    PreparedStatement pst = null;
    ResultSet rs = null;
```

```

try {
    // 1 加载注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 2 获取连接对象
    conn = DriverManager.getConnection("jdbc:mysql:///jdbcdemo", "root",
"admin");
    // 3 获取语句对象
    pst = conn.prepareStatement(sql);
    // 设置sql 的? 的值
    pst.setLong(1, id);
    // 4 执行语句操作
    rs = pst.executeQuery();
    // 从ResultSet 中来获取数据
    if (rs.next()) {
        // 获取列的数据
        long resultId = rs.getLong("id");
        String name = rs.getString("name");
        String email = rs.getString("email");
        int age = rs.getInt("age");
        Student stu = new Student(id, name, email, age);
        return stu;
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    // 释放资源
}
}

```

查询所有学生信息

```

public List<Student> selectList() {
    // 创建一个List 集合,用来存放获取到的每一行数据封装成的 Student 对象
    List<Student> list = new ArrayList<>();

    String sql = "SELECT * FROM t_student";
    Connection conn = null;
    PreparedStatement pst = null;
    ResultSet rs = null;
    try {
        // 1 加载注册驱动
        Class.forName("com.mysql.jdbc.Driver");
        // 2 获取连接对象
        conn = DriverManager.getConnection("jdbc:mysql:///jdbcdemo", "root",
"admin");
        // 3 获取语句对象
        pst = conn.prepareStatement(sql);
        // 4 执行语句操作
        rs = pst.executeQuery();
        // 从ResultSet 中来获取数据
        while (rs.next()) {
            // 获取列的数据
            long resultId = rs.getLong("id");
            String name = rs.getString("name");
            String email = rs.getString("email");

```

```

        int age = rs.getInt("age");
        Student stu = new Student(resultId, name, email, age);
        list.add(stu);
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    // 释放资源
}
return list;
}

```

完成符合 DAO 规范的 CRUD 操作

9_重构设计

在 DAO 的实现过程中,发现释放资源的代码非常恶心,而且还需要反复编写,这种代码我们通常可以抽取到工具类中: JDBCUtil

9.1_抽取 JDBCUtil 工具类

```

// JDBCUtil.java
// 释放资源
public static void close(Connection conn, Statement pst, ResultSet rs) {
    try {
        if (rs != null) {
            rs.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // 释放资源
        try {
            if (pst != null) {
                pst.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
}

```

在获取连接的过程,需要指定连接的信息: 驱动类的全限定名 / url / 用户名 / 密码 (连接数据库的四要素), 但是每次都需要编写对应的字符串,麻烦, 重复, 也容易错。

将获取连接对象的操作抽取到工具类中,统一编写连接数据库的信息

```
//JDBCUtil.java
public static Connection getConnection() throws Exception {
    // 加载注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 获取连接对象
    return DriverManager.getConnection("jdbc:mysql://localhost:3306/javaweb",
    "root", "admin");
}
```

加载注册驱动在整个程序只需要执行一次,所以将加载的代码放到静态代码块中

```
//JDBCUtil.java
static {
    // 加载注册驱动
    // 该操作,在整个程序中只需要执行一次
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

上面的代码中,我们是将连接数据库的信息编写在 Java 代码中的,这当中包括数据库的用户名和密码

Java代码会编译成字节码文件,然后把字节码文件交给用户去使用,当用户需要修改数据库密码时,此时不方便

问题: 硬编码

解决方案: 配置文件

properties(key-value) || xml

由于四要素是key=value 格式, 所有选择使用 properties

9.2_抽取db.properties

```
# db.properties
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/javaweb
username=root
password=admin
```

将文件中的数据加载到内存中的 Properties 对象, 然后再从 Properties 中获取数据, 设置给 JDBC

```
// JDBCUtil.java
private JdbcUtil() {}

private static Properties p;
static {
    // 加载注册驱动
    // 该操作,在整个程序中只需要执行一次
    try {

        // properties配置文件只需要加载一次
        InputStream in = Thread.currentThread().getContextClassLoader()
            .getResourceAsStream("db.properties");
        p = new Properties();
        p.load(in);

        Class.forName(p.getProperty("driverClassName"));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// 获取连接对象
public static Connection getConnection() throws Exception {
    // 获取连接对象
    return DriverManager.getConnection(
        p.getProperty("url"),
        p.getProperty("username"),
        p.getProperty("password"));
}
```

10 JDBC 事务操作

事务在 JDBC 中非常重要，没有事务是一件非常恐怖的事情，如下，所以咱们需要使用 InnoDB 存储引擎。

10.1 银行转账案例

案例: 银行转账, 从张无忌账户上给赵敏转 1000 块钱.

准备: account(账户表)

id	name(账号,唯一)	balance(余额)
1	张无忌	20000
2	赵敏	0

转账操作步骤:

1. 查询张无忌的账户余额是否大于等于1000.

```
SELECT * FROM account WHERE name = '张无忌' AND balance >= 1000;
```

余额小于1000 : 温馨提示:亲,你的余额不足.

余额大于等于1000: GOTO 2;

2. 从张无忌的账户余额中减少1000.

```
UPDATE account SET balance = balance - 1000 WHERE name = '张无忌';
```

3. 在赵敏的账户余额中增加1000.

```
UPDATE account SET balance = balance + 1000 WHERE name = '赵敏';
```

悲剧的事情来了:当程序执行到第②步和第③步之间,突然出现一个异常,此时会造成转账前后数据不一致的问题

造成这个问题的根本原因是,转入转出是两个单独的操作,其中一个失败后,不会影响到另一个的执行,但是在转账这个业务中,我们需要保证进出两个操作要么都成功,要么都失败.

所以这里我们需要使用事务管理来解决这个问题

案例实现:

```
@Test
public void testTx() throws Exception {
    // 贾琰欲执事
    // 1 查询张无忌的账户余额是否大于等于1000
    Connection conn = JDBCUtil.getConnection();
    String sql = "SELECT * FROM account WHERE balance>=? AND name=?";
    PreparedStatement pst = conn.prepareStatement(sql);
    // 给 ? 设置数据
    pst.setBigDecimal(1, new BigDecimal("1000"));
    pst.setString(2, "张无忌");
    ResultSet rs = pst.executeQuery();
    if(!rs.next()){
        System.out.println("余额不足");
        return;
    }
    // 2 从张无忌的账户余额中减少1000.
    sql = "UPDATE account SET balance = balance-? WHERE name=?";
    pst = conn.prepareStatement(sql);
    //设置? 的数据
    pst.setBigDecimal(1, new BigDecimal("1000"));
    pst.setString(2, "张无忌");
    pst.executeUpdate();

    // 模拟出异常
    int a = 10/0;

    // 3 在赵敏的账户余额中增加1000.
    sql = "UPDATE account SET balance = balance+? WHERE name=?";
    pst = conn.prepareStatement(sql);
    //设置? 的数据
    pst.setBigDecimal(1, new BigDecimal("1000"));
    pst.setString(2, "赵敏");
    pst.executeUpdate();

    // 释放资源
```



```
JDBCUtil.close(conn,pst,rs);  
}
```

10.2 JDBC 的事务操作

事务(Transaction,简称为tx)

在数据库中,所谓事务是指一组逻辑操作单元,使数据从一种状态变换到另一种状态。

为确保数据库中数据的一致性,数据的操纵应当是成组的逻辑单元:当每个逻辑操作单元全部完成时,数据的一致性可以保持,而当这个单元中的一部分操作失败,整个事务应全部视为错误,所有从起始点以后的操作应全部回退到开始状态。

10.2.1_事务的ACID属性

1. 原子性 (Atomicity) :原子在化学中,是最小单位,不可以再分割了.
原子性是指事务是一个不可分割的工作单位, 事务中的操作要么都发生, 要么都不发生。
2. 一致性 (Consistency): 包装数据的完整性.
事务必须使数据库从一个一致性状态变换到另外一个一致性状态。(数据不被破坏)
3. 隔离性 (Isolation) :Hibernate再讲
事务的隔离性是指一个事务的执行不能被其他事务干扰, 即一个事务内部的操作及使用的数据对并发的其他事务是隔离的, 并发执行的各个事务之间不能互相干扰。
4. 持久性 (Durability) :
持久性是指一个事务一旦被提交, 它对数据库中数据的改变就是永久性的, 接下来的其他操作和数据库故障不应该对其有任何影响

10.2.2_事务的操作步骤

1. 先定义开始一个事务,然后对数据作修改操作,
2. 执行过程中,如果没有问题就提交(commit)事务,此时的修改将永久地保存下来
3. 如果执行过程中有问题(异常),回滚事务(rollback),数据库管理系统将放弃您所作的所有修改而回到开始事务时的状态。

10.2.3_事务的操作模板(掌握)

```
try{  
    //取消事务的自动提交机制,设置为手动提交.  
    connection对象.setAutoCommit(false);  
    //操作1  
    //操作2  
    //异常  
    //操作3  
    //....  
  
    //手动提交事务  
    connection对象.commit();  
}catch(Exception e){  
    //处理异常
```

```
//回滚事务
connection对象.rollback();
}
```

事务相关注意事项

1. 默认情况下,事务在执行完 DML 操作就自动提交.
2. 查询操作,其实是不需要事务的.但是,一般的,我们在开发中都把查询放入事务中.
3. 开发中,代码完全正确,没有异常,但是就是数据库中数据不变.

意识 : 没有提交事务

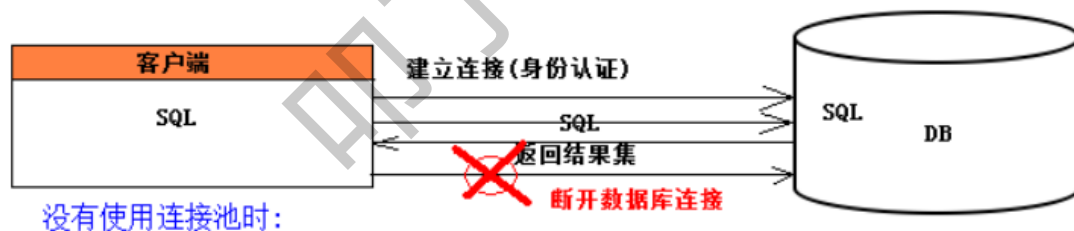
4. 在 MySQL 中,只有 InnoDB 存储引擎支持事务,支持外键,MyISAM 不支持事务.
5. 以后事务我们不应该在 DAO 层处理,应该在 service 层控制.
6. 事务在讲解 MyBatis, Spring, 项目的时候都会再讲.

11_连接池思想

11.1_连接池引入和介绍

普通的 JDBC 数据库连接(Connection对象)使用 DriverManager 来获取, 每次向数据库建立连接的时候都要将 Connection 加载到内存中, 再验证用户名和密码(得花费0.05s ~ 1s的时间),数据库的连接是比较昂贵的(创建的成本比较大)。

需要数据库连接的时候, 就向数据库要求一个, 执行完成后再断开连接。这样的方式将会消耗大量的资源和时间。



数据库的连接资源并没有得到很好的重复利用.若同时有几百人甚至几千人在线, 频繁的进行数据库连接

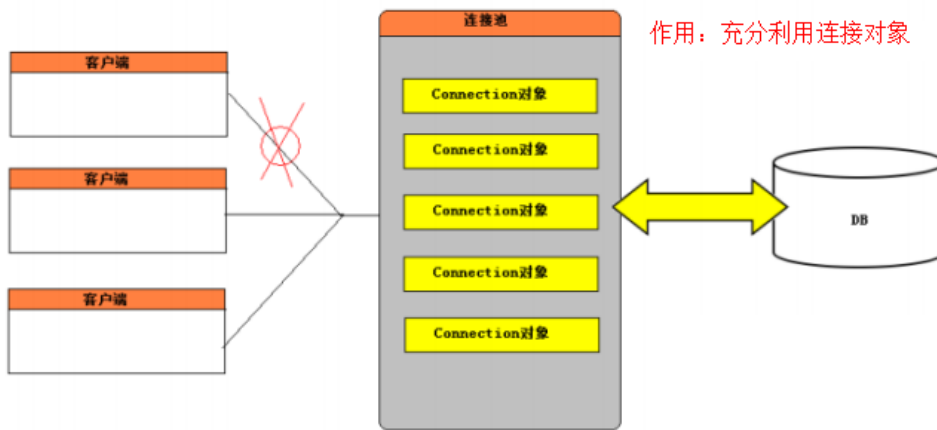
操作将占用很多的系统资源, 严重的甚至会造成服务器的崩溃。

对于每一次数据库连接, 使用完后都得断开。否则, 如果程序出现异常而未能关闭, 将会导致数据库系

统中的内存泄漏, 最终将导致重启数据库。

这种开发不能控制被创建的连接对象数, 系统资源会被毫无顾及的分配出去, 如连接过多, 也可能导致

内存泄漏, 服务器崩溃。



连接池属性分析：联想春运去火车站购票

基本属性：连接池存了连接对象，而连接对象依赖四要素，所以四要素是基本要求

- driverClassName,url,username,password

其他属性：对连接对象做限制的配置

- 初始化连接数：5 在连接池中事先准备好5个Connection对象
- 最多连接数：10 在连接池中最多有10个Connection对象,其他客户端进入等待状态
- 最少连接数：3 在连接池中最少存在3个Connection对象
- 最长等待时间：5 min 使用5分钟来申请获取Connection对象,如果时间到还没有申请到,则提示,自动放弃
- 最长超时时间：10min 如果你在10分钟之内没有任何动作,则认为是自动放弃Connection对象.

在Java 中,连接池使用 javax.sql.DataSource 接口来表示连接池.

DataSource(数据源)和连接池(Connection Pool)是同一个.

注意:DataSource 仅仅只是一个接口,由各大服务器厂商来实现(Tomcat,JBoss).

11.2_常见的 DataSource 实现

DBCP：Spring 框架推荐的

C3P0：Hibernate 框架推荐的

druid：阿里巴巴的连接池(号称 Java 语言中性能最好的连接池).

Java7 Benchmark Result

Jdbc Connection Pool	1 thread	2 threads	5 threads	10 threads	20 threads	50 thread
Druid	898	1,191	1,324	1,362	1,325	1,459
tomcat-jdbc	1,269	1,378	2,029	2,103	1,879	2,025
DBCP	2,324	5,055	5,446	5,471	5,524	5,415
BoneCP	3,738	3,150	3,194	5,681	11,018	23,125
jboss-datasource	4,377	2,988	3,680	3,980	32,708	37,742
C3P0	10,841	13,637	10,682	11,055	14,497	20,351
Proxool	16,337	16,187	18,310 (Exception)	25,945	33,706 (Exception)	39,501 (Exception)

结论

1. Druid是性能最好的数据库连接池，tomcat jdbc和druid性能接近。
2. proxool在激烈并发时会抛异常，完全不靠谱。
3. c3p0和proxool都相当慢，慢到影响sql执行效率的地步。
4. bonecp性能并不优越，采用LinkedTransferQueue并没有能够获得性能提升。
5. 除了bonecp，其他的在JDK 7上跑得比JDK 6上快
6. jboss-datasource虽然稳定，但是性能很糟糕

11.3_使没使用连接池的区别

如何获取 Connection 对象:

- 没有使用连接池: `Connection conn = DriverManager.getConnection(url,username,password);`
- 使用 连接池:`Connection conn = DataSource对象.getConnection();`

只要获取了Connection对象,接下来的操作和以前是一模一样的.

如何释放Connection对象(Connection对象.close());

- 没有使用连接池: 是和数据库服务器断开.
- 使用连接池: 是把Connection对象返还给连接池中,并没有和数据库服务器断开.

关键在于:如何创建DataSource对象,所以需要来学习DataSource实现的使用.

11.4 druid 连接池使用

druid:是阿里巴巴研发出来的号称 Java语言领域性能最高的连接池.

wiki地址:<https://github.com/alibaba/druid/wiki>

使用起来,类似于 DBCP 连接池.

方便检测性能/状态.

支持: MySQL,Oracle,DB2,MS Server等.

支持: 对配置文件的密码加密.

拷贝 jar: druid-1.0.15.jar.

最基本的写法:

```
@Test
public void testDruidDataSource() throws Exception {
    // 创建一个连接池对象
    DruidDataSource ds = new DruidDataSource();
    ds.setDriverClassName("com.mysql.jdbc.Driver");
    ds.setUrl("jdbc:mysql://localhost:3306/jdbcdemo");
    ds.setUsername("root");
    ds.setPassword("admin");

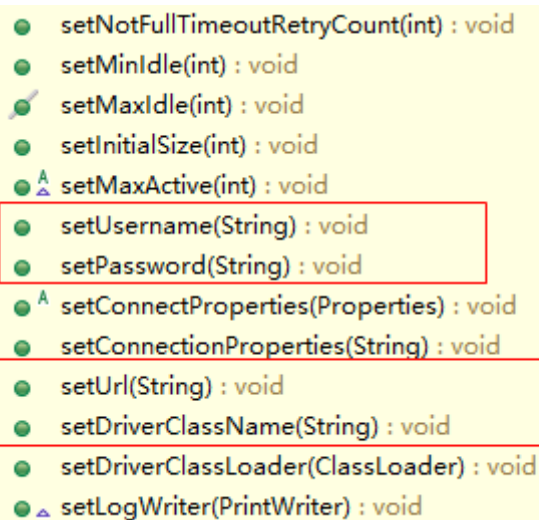
    ds.setInitialSize(5); // 初始化创建连接的个数
    Connection conn = ds.getConnection();
    System.out.println(conn.getClass());
}
```

处理Druid连接池使用过程中的硬编码

在创建连接池对象时,使用到的连接数据库的信息应该编写到 Properties 配置文件中,然后再读取到内存中来使用

db.properties

```
#这里的 key 一定要和 DruidDataSource 中对应的属性名一致
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/javaweb
username=root
password=admin
```



```
● setNotFullTimeoutRetryCount(int) : void
● setMinIdle(int) : void
● setMaxIdle(int) : void
● setInitialSize(int) : void
● setMaxActive(int) : void
● setUsername(String) : void
● setPassword(String) : void
● setConnectProperties(Properties) : void
● setConnectionProperties(String) : void
● setUrl(String) : void
● setDriverClassName(String) : void
● setDriverClassLoader(ClassLoader) : void
● setLogWriter(PrintWriter) : void
```

```

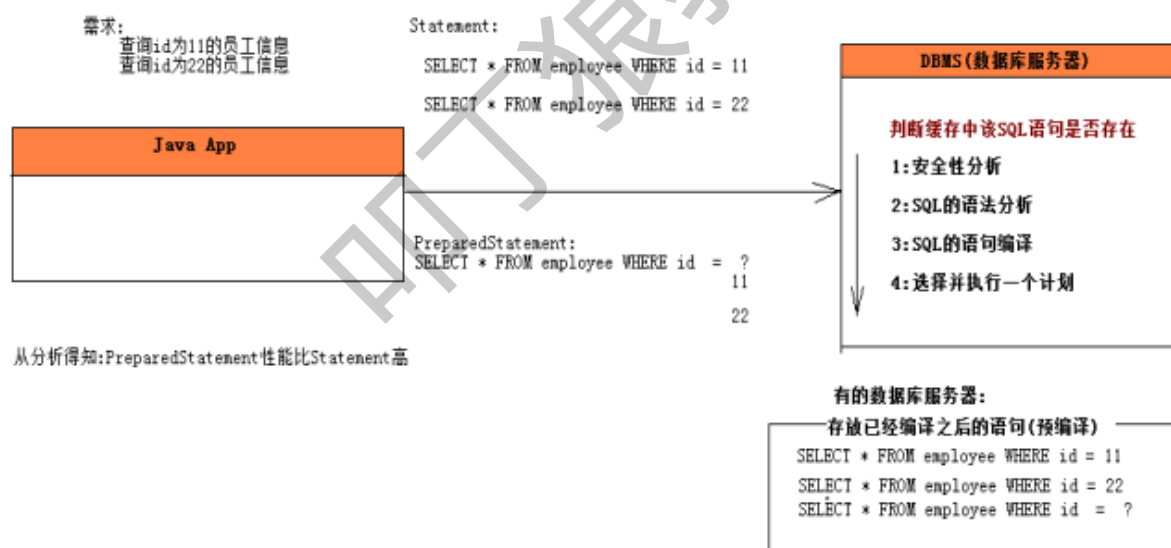
public class DruidUtil {
    private DruidUtil() {
    }
    private static Properties p = new Properties();
    private static DataSource dataSource = null;
    static {
        try {
            //从classpath的根路径去寻找dbcp.properties文件
            InputStream inStream = Thread.currentThread()
                .getContextClassLoader()
                .getResourceAsStream("dbcp.properties");
            p.load(inStream);
            dataSource = DruidDataSourceFactory.createDataSource(p);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

12_Statement 和 PreparedStatement的区别(了解)

Statement 和 PreparedStatement 的区别: PreparedStatement 存在的优势:

1. 更好的可读性,可维护性.
2. 可以提供更好的性能(预编译). MySQL 不支持 PreparedStatement 性能优化.



3. 更安全,可以防止 SQL 注入的问题

如果使用Statement语句对象,我们是把参数直接拼接到 SQL 中然后执行,此时,如果参数会改变SQL的语法结构,执行的结果就会存在问题了,如,在登录查询的 SQL 中,如果用户名参数值为: ' or 1=1 or ',此时,填写的密码无论是多少,登录都会成功,这就是 SQL 注入的问题.

```

@Test
public void testLoginByStatement() throws Exception {
    //String sql = "SELECT * FROM t_student WHERE name = 'admin' AND password='1234'";
    String sql = "SELECT * FROM t_student WHERE name = ' ' OR 1=1 OR ' ' AND password='SB'";
    Connection conn = JdbcUtil.getConn();
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery(sql);
    if(rs.next()){
        System.out.println("登陆成功");
    }else{
        System.out.println("登陆失败");
    }
}
}

```

这个问题可以使用预编译语句对象完美解决

1. 预先发送带有占位符的 SQL 到数据库中进行编译, 语句结构固定下来
2. 设置参数给对应的占位符,然后再执行 SQL

这里无论是什么参数,都不会再改变 SQL 的语法结构,达到防止 SQL 注入问题的目的

```

@Test
public void testLoginByPreparedStatement() throws Exception {
    String sql = "SELECT * FROM t_student WHERE name = ? AND password=?";
    Connection conn = JdbcUtil.getConn();
    PreparedStatement ps = conn.prepareStatement(sql);
    ps.setString(1, " ' OR 1=1 OR '");
    ps.setString(2, "1234");
    ResultSet rs = ps.executeQuery();
    if(rs.next()){
        System.out.println("登陆成功");
    }else{
        System.out.println("登陆失败");
    }
}
}

```

13_查询操作代码抽取(了解)

```

/**
 * 处理查询操作
 * <T>:是声明泛型类型
 * List<T>, Class<T>使用声明好的T
 * @param sql 要执行的SQL语句
 * @param type 将每行数据封装的对象类型
 * @param params 执行的SQL需要的参数
 * @return 返回查询到的结果,统一放到List集合中
 */
public static <T> List<T> executeQuery(String sql,Class<T> type,
Object...params){
    Connection conn = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    List<T> list = new ArrayList<>();
    try {
        conn = DruidUtil.getConnection();
        ps = conn.prepareStatement(sql);
    }
}

```

```

//为占位符设置
for (int i = 0; i < params.length; i++) {
    ps.setObject(i + 1, params[i]);
}

rs = ps.executeQuery();
// 直到next方法返回false时结束
while (rs.next()) {
    // 获取这一行中的指定列的数据
    //使用反射创建对象
    T t = type.newInstance();
    //将数据从结果集中获取到,并设置给对象t
    //通常情况下,属性名和列名一样,所以我们可以根据属性名去获取对应列的值
    BeanInfo beanInfo = Introspector.getBeanInfo(t.getClass(),
object.class);
    PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();
    //操作每个属性
    for (PropertyDescriptor pd : pds) {
        //获取到属性名
        String name = pd.getName();
        //根据这个属性名从结果集中获取到数据
        Object value = rs.getObject(name);
        //获取到属性对应的set方法
        Method writeMethod = pd.getWriteMethod();
        writeMethod.invoke(t, value);
    }
    list.add(t);
}
} catch (Exception e) {
    e.printStackTrace();
} finally {
    JdbcUtil.close(conn, ps, rs);
}
return list;
}

```

小结

- 1 重点掌握 JDBCUtil 的抽取精华
- 2 重点掌握 db.properties 文件的抽取(解决硬编码)
- 3 重点掌握程序如何使用事物
- 4 重点掌握 druid 连接池的使用

作业： 可新建项目也可直接拷贝旧的项目,删除类去重新练习

- 1 从零开始书写一遍基于 DAO 规范的 crud 操作.(表自行定义,目的: 掌握 crud 操作以及 DAO 规范)
- 2 抽取 JDBCUtil 工具类
- 3 抽取db.properties

4 书写带有事务 TX 的程序(只要在程序中加入事物即可,不知道的情况: 使用课堂上的转账案例去巩固事物操作)

5 把程序改为使用 druid 连接池 (修改 JDBCUtil 也可以 书写一个新的工具类 DruidUtil)

6 以上操作不熟悉,重复写

7 复习 sql 语句,简单的 crud 操作,基本的过滤条件,排序,分页语法 (MySQL练习.txt)

拓展题: 不做要求,有时间有精力去研究下

模拟登录操作:控制台版本的登录操作(测试类+DAO)

1 当用户输入的账号错误时,提示用户账号错误

2 当用户输入的密码错误时,提示用户密码错误

叮丁狼教育