JDBC代码重构

1、JDBC操作模板：jdbc Temp late

- DML操作模板
  - 第一步：链接数据源

    utils包 下的 JDBCTemplete类

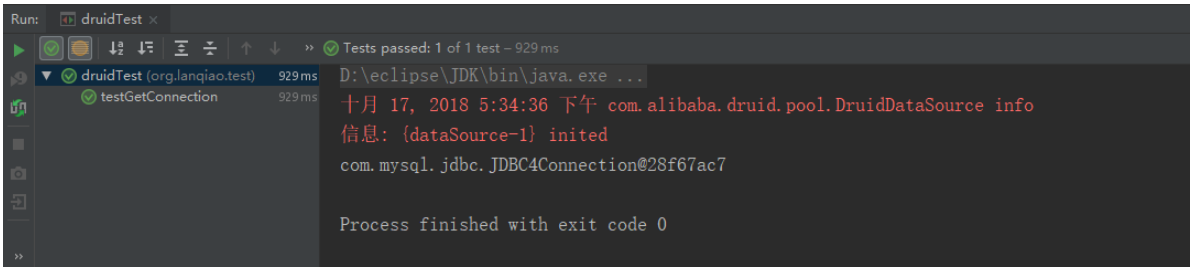
```
public class JDBCUtils {
    private static  String driverClassName;
    private static String url;
    private static  String username ;
    private static String password ;
    private static int maxActive;
    private static int initSize;
    private static DruidAbstractDataSource druid= null;
    static {
        //读取文件，得到一个输入流
        //首先通过当前类获取自己的Class对象，在获取类加载器，得到输入流
        InputStream inputStream =
JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties");
        //创建Properties对象(属性对象)
        Properties properties = new Properties();
        try {
            //将属性流加载到属性对象中
            properties.load(inputStream);
            //获取其中内容
            druid = new DruidDataSource();//创建一个数据源
            //设置连接属性
            driverClassName = properties.getProperty("driverClassName");
            url = properties.getProperty("url");
            username = properties.getProperty("username");
            password = properties.getProperty("password");
            initSize = Integer.valueOf(properties.getProperty("initialSize"));
            maxActive = Integer.valueOf(properties.getProperty("maxActive"));
            druid.setDriverClassName(driverClassName);
            druid.setUrl(url);
            druid.setUsername(username);
            druid.setPassword(password);

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

    public static Connection getConnection(){
        Connection connection = null;
        try {
            connection = druid.getConnection();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        return connection;
    }

    public static void releaseSource(ResultSet resultSet, Statement statement,
Connection connection){
        try {
            if (resultSet != null){
                resultSet.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            try {
                if (statement != null){
                    statement.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }finally {
                try {
                    if (connection != null){
                        connection.close();
                    }
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

- 第二部：进行测试 test包 下的druidTest类

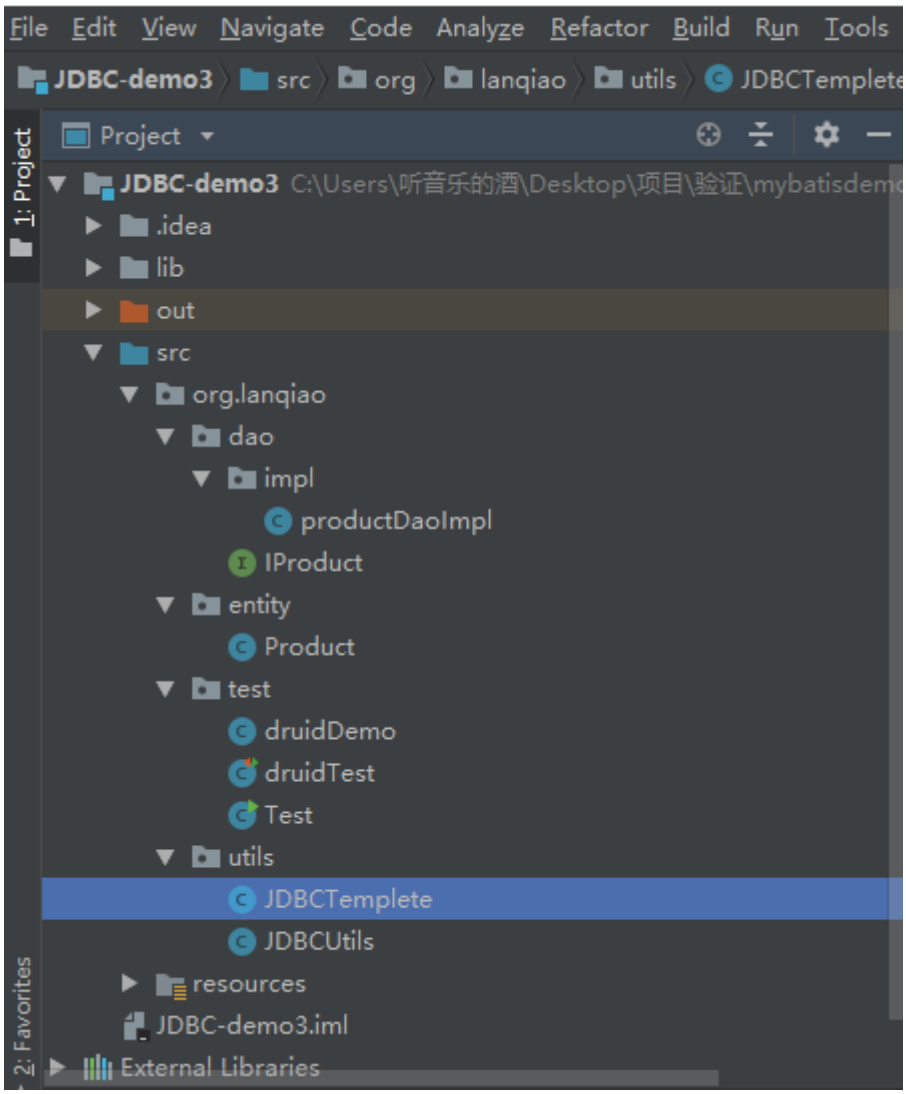
```
public class druidTest {

    @Test
    public void testGetConnection(){
            Connection connection = JDBCUtils.getConnection();
            System.out.println(connection);
    }
}
```

链接成功：如图



- 第三部：进行增删改操作
  - 项目的目录

    如图所示：



  - 各个包下的具体实现
    - utils包 下的JDBCTemplete类

```
//增删改操作
    //参数: insert,delete,update
    public  static  void execUpdate(String sql,Object ... args){
        Connection connection =  JDBCUtils.getConnection();
        PreparedStatement preparedStatement = null;
        try {
            connection.setAutoCommit(false);//手动关闭
```

```
                preparedStatement = connection.prepareStatement(sql);//此
处传哪个sql进来? 就执行哪个sql
            //设置参数
            //对参数进行for循环
            for(int i = 0;i < args.length;i++){
                preparedStatement.setObject(i+1,args[i]);//参数从哪开始
            }
            preparedStatement.executeUpdate();//进行更新操作，更新操作都需
要事务
            connection.commit();//手动提交
        } catch (SQLException e) {
            try {
                connection.rollback();//若出现异常则进行回滚操作
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
        }finally {
            //释放资源

 JDBCUtils.releaseSource(null,preparedStatement,connection);
        }
    }
```

- test包 下的Test类

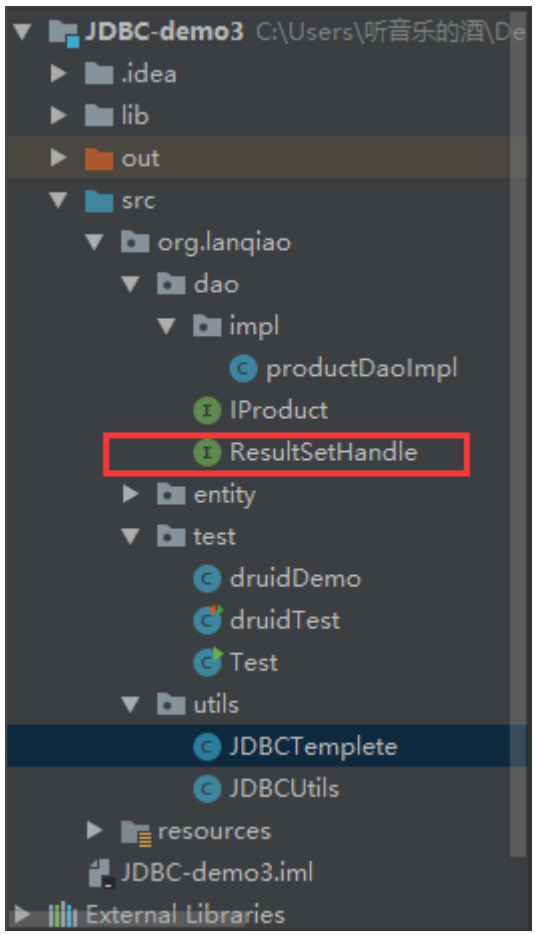
```
public class Test {
    public static void main(String[] args) throws Exception{
        IProduct iProductp = new productDaoImpl();
        /*//根据ID查询
        Product product = iProductp.getProductById(1);
        System.out.println(product);
        System.out.println("----------------------------------------
------------");
        //根据产品名称查询
        List<Product> productList = iProductp.getProductByName("M");
        for (Product product1:productList){
            System.out.println(product1);
        }
        System.out.println("----------------------------------------
-------------");
        //查询所有
        List<Product> productAll = iProductp.getAll();
        for (Product productList2:productAll){
            System.out.println(productList2);
        }
        System.out.println("----------------------------------------
------------------");
        //更新
        Product product3 = iProductp.getProductById(1);
         product3.setSalePrice(666);
         iProductp.updateProductById(product3);*/
       //删除
```

```
            iProductp.deleteProductById(10);
        }
    }
```

        - 当当
    - 当当
- DQL操作模板
    - 结果集处理
    - 具体实现代码
        - utils包 下的 JDBCTemplete类

```
//查询操作
    public static  <T>T  execSelect(String sql, ResultSetHandle<T> rsh,
Object ... args){
        Connection connection =  JDBCUtils.getConnection();
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            preparedStatement = connection.prepareStatement(sql);
            //传递参数(给它一个参数)
            //对参数进行循环(参数设置,Object ... args[])
            for(int i = 0;i < args.length;i++){
                preparedStatement.setObject(i + 1,args[i]);
            }
            //返回一个结果集
            resultSet = preparedStatement.executeQuery();
            //进行结构集处理
            //新建一个处理结果集的接口
            //添加一个结果集处理器
            //结果集的处理
           T t =rsh.handle(resultSet);//得到一个泛型对象
           return t;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return  null;
    }
```

        - 在Dao层下添加一个接口
            - 项目结构目录



- 代码

```
public interface ResultSetHandle<T> {
    T handle(ResultSet resultSet) throws SQLException;//结果集处理后，返回一个T对象
}
```

- 当当

- test包 下的Test类

```
//基于DQL操作模板的查询
        Product product = iProductp.getProductById(1);
        System.out.println(product);
```

- 方法泛型参数设计
  - 返回结果
    - 对象
    - 集合
    - 使用统计函数返回一个单个数字
  - 提供了一个泛型方法，根据不同结果集提供不同实现类
  - 如图：

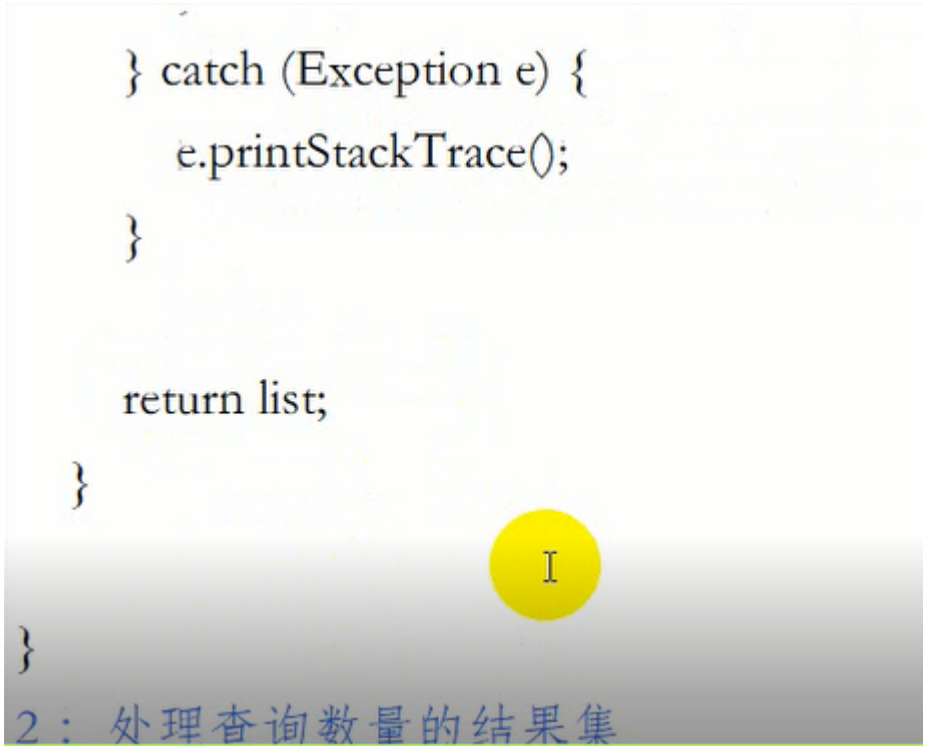
实现了ResultHandle<T>的处理结果集类

1： 处理list结果集

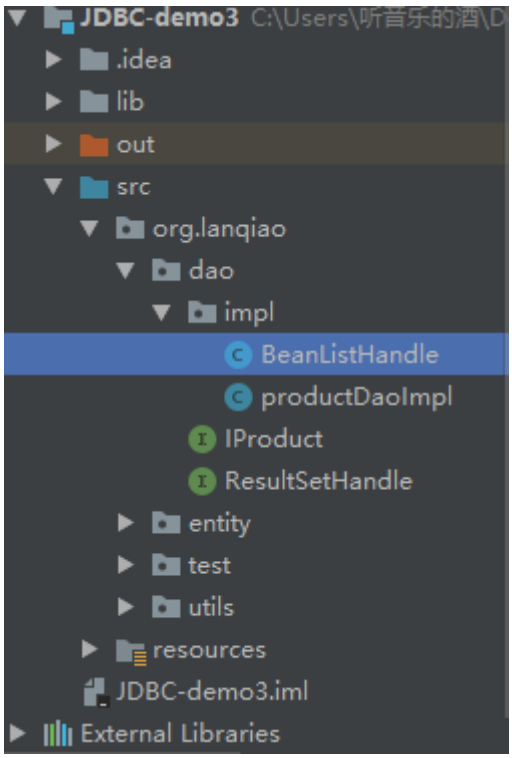
```
public class BeanListHandle<T> implements ResultHandle<List<T>>{
    private Class<T> clazz;
    public BeanListHandle(Class<T> clazz){
        this.clazz=clazz;
    }
    @Override
    public List<T> handle(ResultSet rs) {
```

```
    @Override
    public List<T> handle(ResultSet rs) {
        List<T> list=new ArrayList<T>();
        try {
            while(rs.next()){
                T t=clazz.newInstance();
                BeanInfo beanInfo = Introspector.getBeanInfo(clazz, Object.class);
                PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();
                for (PropertyDescriptor pd : pds) {
```

```
                for (PropertyDescriptor pd : pds) {
                    String name = pd.getName();
                    Method method = pd.getWriteMethod();
                    method.invoke(t,rs.getObject(name));
                }
                list.add(t);

            }
```



- 新建一个类：BeanListHandle（dao层下的Impl包中）
  - 如图：项目结构图



  - 代码

```
public class BeanListHandle<T> implements ResultSetHandle<List<T>>{
    //定义成员属性
    private  Class<T> clazz;
    //构造方法
    public BeanListHandle(Class<T> clazz){
        this.clazz = clazz;
```

```
        }

        //实现接口的方法
        @Override
        public List<T> handle(ResultSet resultSet) throws SQLException,
    IllegalAccessException, InstantiationException,
    IntrospectionException, InvocationTargetException {
            //进行结果集的处理
            //1、创建一个List
            List<T> list = new ArrayList<>();
            while (resultSet.next()){
                //创建一个对象
                T t = clazz.newInstance();
                //获取一个包含属性的对象
                BeanInfo bean =
    Introspector.getBeanInfo(clazz,Object.class);
                //得到一个数组
                PropertyDescriptor[] pds = bean.getPropertyDescriptors();
                //对数组进行for循环
                for(int i = 0;i < pds.length;i++){
                    //获取属性名称
                    String name = pds[i].getName();
                    //获取所有写的方法
                    Method method = pds[i].getWriteMethod();
                    //执行
                    method.invoke(t,resultSet.getObject(name));
                }
                list.add(t);
            }
            return list;
        }
    }
```

- 修改productDaoImpl类中getProductById(int id)方法的代码

```
@Override//根据ID查询
    public Product getProductById(int id)  {
        String sql = "select * from product where id = ?";
        Product product = new Product();
        List<Product> list = JDBCTemplete.execSelect(sql,new
 BeanListHandle<>(Product.class) , id);
        return list.get(0);
    }
```

- 当当

- 当当

- 小结：
  - 数据库连接池：dbcp c3p0 druid
  - 代码重构：
    - 重点应用的知识：反射和泛型

        - 查询中难点：结果集的处理
            - 提供一个接口：

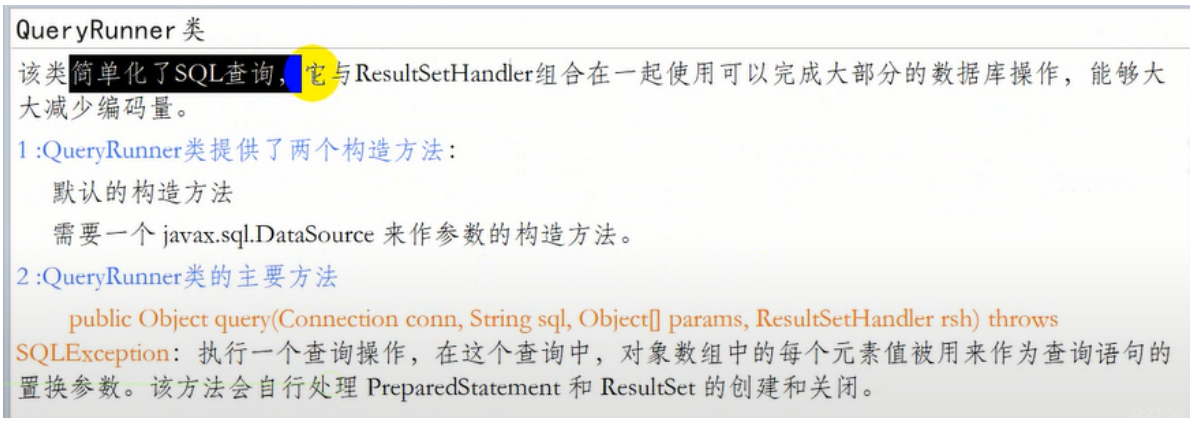
```
```

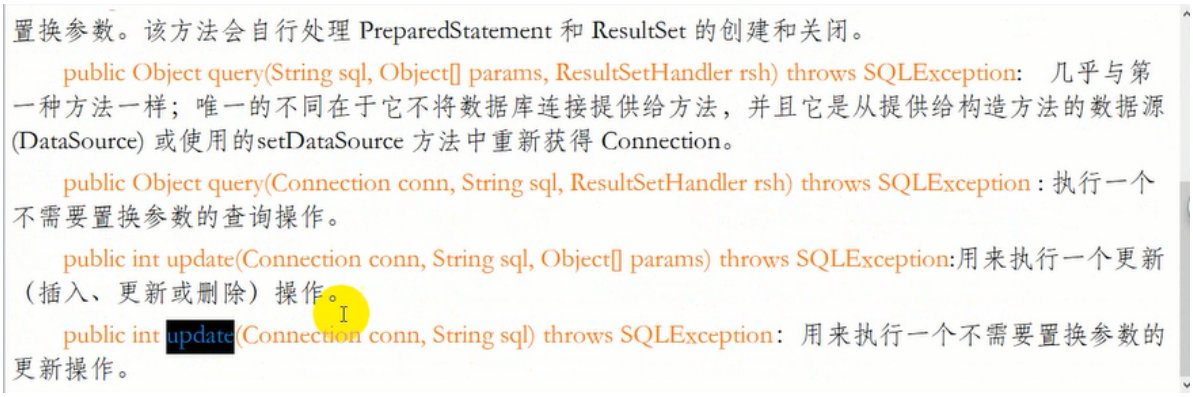
        - 当当
    - 当当
- 当当

2、DCUtils（第三方JDBC框架）

- 简介
    - 如图：

    - 当当
- QueryRunner类
    - 使用QueryRunner类实现CRUD
    - 如图:





- 新建一个实例
  - 使用QueryRunner类实现CRUD,queryRunner类中的构造方法需要一个数据源，在工具类中，提供一个获取 DataSource(数据源) 的方法

    utils包 下 JDBCUtils类

```
public class JDBCUtil {
    private static DruidDataSource ds =new DruidDataSource();
    static {
        InputStream in =
JDBCUtil.class.getClassLoader().getResourceAsStream("druid.properties");
        Properties properties = new Properties();
        try {
            properties.load(in);

ds.setDriverClassName(properties.getProperty("driverClassName"));
            ds.setUrl(properties.getProperty("url"));
            ds.setUsername(properties.getProperty("username"));
            ds.setPassword(properties.getProperty("password"));

ds.setInitialSize(Integer.valueOf(properties.getProperty("initialSize")));

ds.setMaxActive(Integer.valueOf(properties.getProperty("maxActive")));
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }

    public static DataSource getDataSource() {
        return ds;
    }
    public static void ReleaseSource(ResultSet res ,Statement
stmt,Connection conn) {

            try {
                if(res !=null) {
                    res.close();
                }

            } catch (SQLException e) {
             // TODO Auto-generated catch block
```

```
            e.printStackTrace();
        }finally {

            try {
                if(stmt!=null) {
                stmt.close();
                }
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally {
                if(conn !=null) {
                    try {
                        conn.close();
                    } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }

            }

    }



    }
}
```

- 进行测试

  test包 下的 Test类:测试链接数据库是否成功

```
public class Test {
    public static void main(String[] args){
        System.out.println(JDBCUtils.getDataSource());
    }
}
```

- 创建一个实体类

  entity包 下的 User类

```
public class User {
    private int id;
    private String name;
    private int age;
    private String sex;
    private String password;

    public User(){

    }
    。。。
    。。。
}
```

- Dao层
  - IUser接口

```
public interface IUser {
    //添加
    public  void  insert(User user) throws SQLException;
    //更新(修改)
    public void update (User user) throws SQLException;
    //删除
    public void delete(int id) throws SQLException;
    //根据ID查询
    public User getUserByID(int id) throws SQLException;
    //查询所有
    public List<User> getUserAll() throws SQLException;
}
```

  - 接口的实现类：UserDaoImpl类

```
public class UserDaoImpl implements IUser {

    //创建一个QueryRunner对象（由JDBCUtils工具类提供一个数据源）
    QueryRunner queryRunner = new
QueryRunner(JDBCUtils.getDataSource());

    @Override
    //添加
    public void insert(User user) throws SQLException {
        String sql = "insert into stu(name,password) values(?,?)";
        //sql,结果集处理器，参数，执行完返回一个对象
        User user1 = queryRunner.insert(sql,new BeanHandler<>
(User.class),user.getName(),user.getPassword());
    }

    @Override
    public void update(User user) throws SQLException {
        String sql = "update stu set name='李超',password='88888' where
id = 1";
```

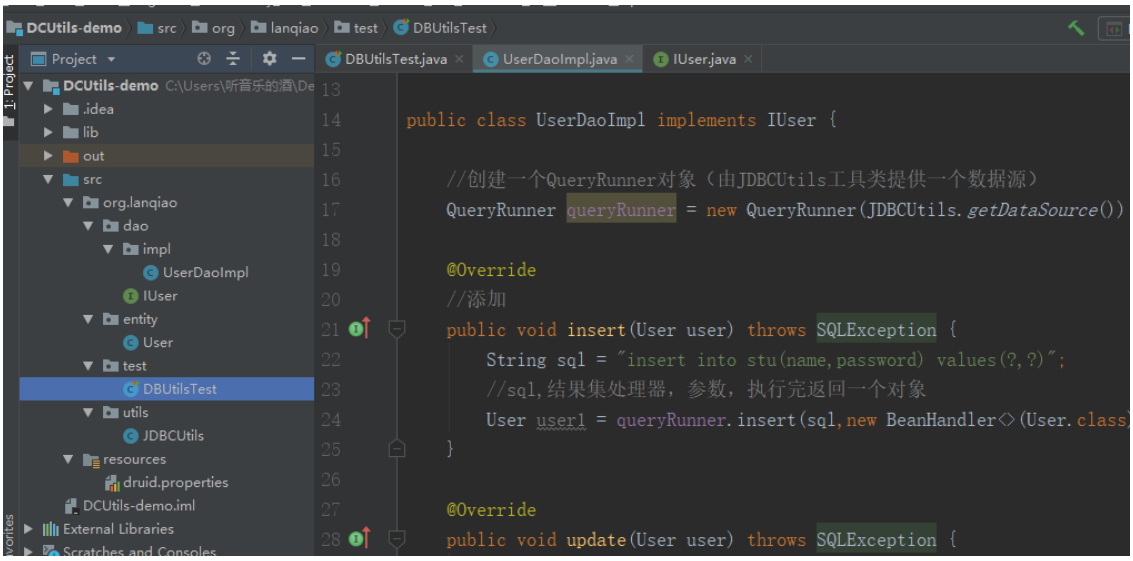
```
            queryRunner.update(sql);
        }

        @Override
        //删除
        public void delete(int id) throws SQLException {
            String sql = "delete from stu where id = ?";
            queryRunner.update(sql,id);
        }

        @Override
        //根据ID查询
        public User getUserByID(int id) throws SQLException {
            String sql = "select * from stu where id = ?";
            //结果集处理器，如何去处理？有一个结果集处理器：Interface
    ResultSetHandler<T>(他是一个接口，接口不能创建对象)，创建一个BeanHandler对象
            User user = queryRunner.query(sql,new BeanHandler<>
    (User.class),id);//类型：User.class
            return user;
        }

        @Override
        public List<User> getUserAll() {
            String sql = "select * from stu";
            List<User> list = new ArrayList<>();
            try {
               list = queryRunner.query(sql,new BeanListHandler<>
    (User.class));
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return list;
        }
    }
```

- test包下的 测试类：Test

```
public class DBUtilsTest {
    public static void main(String[] args){
        System.out.println(JDBCUtils.getDataSource());
    }

    @Test
    //添加
    public void insertTest() throws SQLException {
        IUser iUser = new UserDaoImpl();
        User user = new User();
        user.setName("火女");
        user.setPassword("2222");
    }

    @Test
    //删除
```

```
    public void deleteTest() throws SQLException {
        IUser iUser = new UserDaoImpl();
        iUser.delete(13);
    }

    @Test
    //根据ID查询
    public void getUserByIdTest() throws SQLException {
        IUser iUser = new UserDaoImpl();
        User user = iUser.getUserByID(1);
        System.out.println(user);
    }

    @Test
    //更新
    public void updateTest() throws SQLException {
        IUser iUser = new UserDaoImpl();
        User user = new User();
        iUser.update(user);
    }

    @Test
    //查询所有
    public void getUserAllTest() throws SQLException {
        IUser iUser = new UserDaoImpl();
        List<User> list = iUser.getUserAll();
        for (User str:list){
            System.out.println(str);
        }
    }
}
```

- 项目结构图



- 当当

  - 当当

- ResultSetHandler接口

- 当当

3、JDBC调用存储过程和函数