DBUtils

1. 介绍

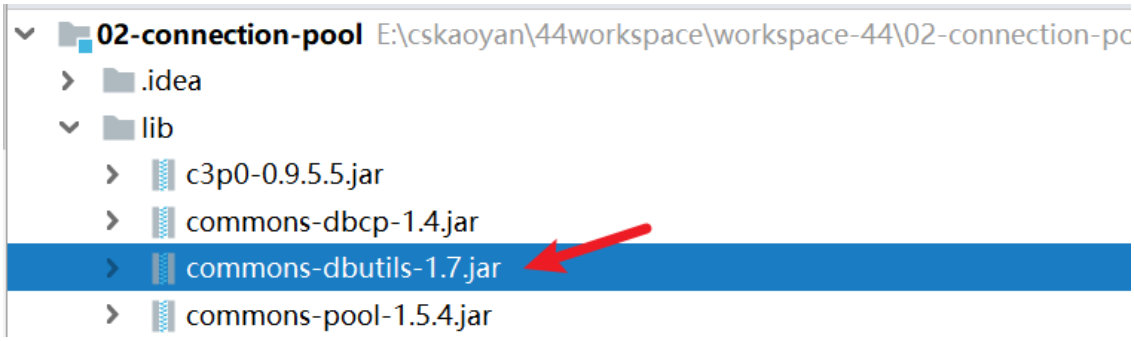
DBUtils是一个 Apache开源的 关于操作JDBC的一个简单工具类库。可以帮助我们简化 JDBC的操作。

```
// 1. 注册驱动

// 2. 获取连接

// 3. 获取Statement对象

// 4. 执行SQL语句

// 5. 解析结果集

// 6. 关闭资源
```

DBUtils可以帮助我们简化 第三步、第四步、第五步的操作。

2. 使用

- 导包



- 配置

  DBUtils不需要额外的任何配置，可以直接使用

- 使用

3. API

DBUtils

DBUtils仅仅是提供了一些非常简单的方法封装，例如：

- 关闭资源（close）
- 安静的关闭资源（在方法内部处理异常，并且不打印异常的栈信息）
- 提交
- 回滚

由于封装的代码非常简单，一般不使用。

QueryRunner

QueryRunner这个类实际上可以帮助我们简化SQL语句的执行过程，对创建Statement过程无感知。

```
// 1. 注册驱动

// 2. 获取连接

// 3. 获取Statement对象

// 4. 执行SQL语句

// 5. 解析结果集

// 6. 关闭资源
```

有了QueryRunner之后，我们不用自己来创建Statement。

```
// 无参构造方法
QueryRunner queryRunner = new QueryRunner();

/*使用这个QueryRunner的时候，在执行SQL语句的时候，需要传入Connection对象*/

// 有参构造（传入数据库连接池）
QueryRunner queryRunner1 = new QueryRunner(DruidUtils.getDataSource());
// 使用这个传入数据库连接池的QueryRunner1的时候，我们在执行SQL语句的时候，不需要传入
Connection对象；QueryRunner会自己从数据库连接池中获取连接来执行对应的SQL语句。
```

无参构造

```
// 无参构造方法
QueryRunner queryRunner = new QueryRunner();

// 获取Connection
// 连接对象可见，可以控制事务
Connection connection = DruidUtils.getConnection();

int affectedRows = queryRunner.update(connection,"update user set username =
?,nickname=? where id = ?",
        "王英","矮脚虎","1004");


System.out.println("affectedRows:" + affectedRows);

connection.close();
```

有参构造

```
// 传入数据库连接池
QueryRunner queryRunner = new QueryRunner(DruidUtils.getDataSource());


int affectedRows = queryRunner.update("update user set username = ?,nickname=?
where id = ?",
        "林冲","豹子头","1004");


System.out.println("affectedRows:" + affectedRows);
```

> QueryRunner去执行查询语句的时候，需要配合ResultSetHandler一起来使用。

ResultSetHandler

这个类本质上是一个接口。这个接口中定义了一个解析结果集的方法。

```
// 源代码
public interface ResultSetHandler<T> {

    // handle方法的本质实际上就是把ResultSet结果集解析为Java对象
    T handle(ResultSet var1) throws SQLException;
}
```

自己实现

自己实现ResultSetHandler接口

```
public class MyResultSetHandler implements ResultSetHandler {
    @Override
    public Object handle(ResultSet resultSet) throws SQLException {

        if (resultSet.next()) {

            int id = resultSet.getInt("id");
            String username = resultSet.getString("username");
            String password = resultSet.getString("password");
            String nickname = resultSet.getString("nickname");

            User user = new User();

            user.setId(id);
            user.setUsername(username);
            user.setPassword(password);
            user.setNickname(nickname);

            return user;

        }else {
            return null;
        }


    }
```

```
}
```

> 弊端：
>
> 1. 自己实现比较麻烦，并没有帮助我们简化JDBC的流程，还是需要我们手动来解析结果集
> 2. 自己实现的ResultSetHandler接口是不能通用的
>
> 其实DBUtils给我们提供了一些ResultSetHandler接口的实现类，我们只需要来使用这些实现类即可。

> 注意：在使用DBUtils的自动映射的ResultSetHandler的实现类的时候，需要让查询的结果的临时表的列名和 Java对象中的成员变量名保持一致，不然对应的字段会映射失败。

BeanHandler

> 可以帮助我们把结果集对象解析为 单个Java对象

```
// 1. 构建QueryRunner
QueryRunner queryRunner = new QueryRunner(DruidUtils.getDataSource());

// 2. 执行查询语句
User user = queryRunner.query("select id,username as name,password,nickname as
xxxname from user where id = ?",new BeanHandler<>(User.class),1002);

// 3. 打印对象
System.out.println(user);
```

BeanListHandler

> 可以帮助我们把结果集对象解析为 Java对象的集合

```
// 1. 获取一个QueryRunner
QueryRunner queryRunner = new QueryRunner(DruidUtils.getDataSource());

// 2. 执行SQL语句
List<User> userList = queryRunner.query("select * from user where id in
(?,?,?)",new BeanListHandler<>(User.class),1001,1002,1003);

// 3. 打印
System.out.println(userList);
```

ColumnListHandler

> 假如SQL语句的执行结构是单列值，那么可以把这单列值解析为一个List

```
// 1. 构建QueryRunner
QueryRunner queryRunner = new QueryRunner(DruidUtils.getDataSource());

// 2. 执行SQL语句
List<String> nameList = queryRunner.query("select username from user",new
ColumnListHandler<>());

// 3. 打印
System.out.println(nameList);
```

ScalarHander

> 假如SQL语句的执行结果是单个值，可以使用ScalarHander来直接解析

```
// 1. 构建QueryRUnner
QueryRunner queryRunner = new QueryRunner(DruidUtils.getDataSource());

// 2. 执行SQL
Long count = queryRunner.query("select count(*) from user", new ScalarHandler<>
());


System.out.println("count:" + count);
```

MapListHandler

```
// 1. 获取QueryRUnner
QueryRunner queryRunner = new QueryRunner(DruidUtils.getDataSource());

// 2. 执行SQL语句
List<Map<String,Object>> mapList =   queryRunner.query("select * from user",new
MapListHandler());


System.out.println(mapList);
```

4. 总结

DBUtils有什么优缺点呢?

- 优点：可以帮助我们简化JDBC的流程，尤其是解析获取结果集的过程
- 缺点：
    1. SQL语句和代码是硬编码的（正常来说SQL语句要和代码分开管理）
    2. 每次解析结果集的时候需要我们自己去选择使用哪个ResultSetHandler，不太智能
    3. 复杂映射（多表查询）不够强大的

> 在公司里面，我们一般在什么情况下使用DBUtils呢?
>
> 假如项目比较简单，访问量不太大（轻量级），可以考虑使用DBUtils；否则都会使用 Mybatis。

Junit

1. 介绍

Junit是一个开源的测试工具。目前Junit已经发展到了 Junit5，但是在大量使用的是还是Junit4，所以推荐大家使用Junit4。

版本要求：4.12

官网

2. 使用

导包

配置

不需要任何配置

使用

Junit给我们提供了一些注解。

```
// @Test

// @Before

// @After

// @BeforeClass

// @AfterClass
```

- @Test

  修饰在测试方法上，让这个测试方法可以像main方法一样运行

- @Before

  这个注解修饰的方法并不能直接运行，是在@Test注解修饰的方法运行之前执行

- @After

  这个注解修饰的方法是在测试方法运行之后执行

- @BeforeClass

  是在这个测试类进行类加载的时候执行

- @AfterClass

  在这个测试类类销毁的之前执行

3. 规范

1. 测试类命名的时候，建议大家使用 XxxTest
2. @Test注解修饰的方法（测试方法）的方法名建议使用 testXxx();
3. @Test注解修饰的方法：

- 必须是public
- 必须是void
- 必须是无参的方法

4. @BeforeClass、@AfterClass修饰的方法必须是静态的

> 思考题：我们在学习JavaSE的时候，我们都知道可以直接运行的方法就是main方法，main方法也是Java程序执行的入口。那么为什么一个方法添加了@Test注解之后就可以直接执行呢？
>
> 根本上还是执行的Main方法，不过是执行的JunitStarter这个类中main方法，然后在这个main方法中通过反射来调用我们当前执行的测试方法。