1.JDBC的问题分析：

1.数据库配置信息存在硬编码问题 => 配置文件

2.频繁创建释放数据库链接 => 连接池

3.sql语句、设置参数、获取结果集存在硬编码问题 => 配置文件

4.手动封装返回结果集，较为繁琐 => 反射、内省

2.自定义框架设计：

使用端（项目）：引入jar包。

提供两部分配置信息：

1.**sqlMapConfig.xml**：存放数据库配置信息，存放mapper.xml的全路径

2.**mapper.xml**:存放sql配置信息（sql语句、参数类型、返回值类型）

自定义持久层框架本身（工程）：封装JDBC。

1. **加载配置文件：**根据配置文件的路径，加载配置文件成字节输入流，存储在内存中

   创建Resources类 方法：InputSteam getResourceAsSteam(String path)
2. **创建两个javaBean(容器对象)：**存放的就是对配置文件解析出来的内容

   Configuration（核心配置类）：存放sqlMapConfig.xml解析出来的内容

   MappedStatement（映射配置类）：存放mapper.xml解析出来的内容
3. **解析配置文件**：dom4j

   创建类：SqlSessionFactoryBuilder 方法：build(Inputsteam in)

   第一：使用dom4j解析配置文件，将解析出来的内容封装到容器对象中

   第二:创建SqlSessionFactory对象：生产sqlSession: 会话对象（工厂模式）
4. **创建SqlSessionFactory借口口及实现类DefaultSqlSessionFactory**

   openSession(): 生产sqlSession
5. **创建SqlSession接口及实现类DefaultSession**

   定义对数据库的CRUD操作：selectList()、selectOne()、update()、delete()
6. **创建Executor接口及实现类SimpleExecutor实现类**

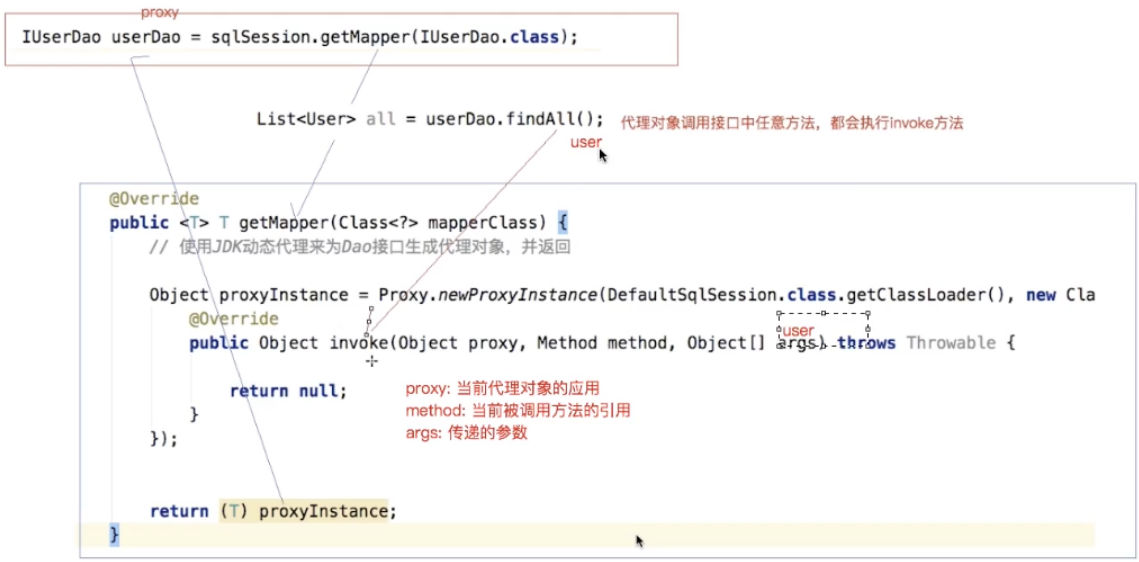
   query(Configuration, MappedStatement,Object... params): 执行JDBC代码

3.自定义持久层框架问题分析：

1.Dao层使用自定义持久层框架，存在代码复用，整个操作的过程模版重复（加载配置文件、创建sqlSessionFactory、生产sqlSession）。

2.statementId存在硬编码问题。

解决思路：使用代理模式生成Dao层接口的代理实现类。



getMapper->使用JDK动态代理，为你传递过来的Dao接口生成代理对象proxyInstance->返回给userDao，所以userDao的类型就是proxy类型。->需明确：代理对象调用接口中的任意方法，都会执行invoke方法。

4.mybatis简单回顾

1.传统开发方式:

```
@Test
public void test1() throws IOException {
    //1.Resources工具类，配置文件的加载，把配置文件加载成字节输入流
    InputStream resourceAsStream = Resources.getResourceAsStream("sqlMapConfig.xml");
    //2.解析了配置文件，并创建了sqlSessionFactory工厂
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(resourceAsStream);
    //3.生产sqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();// 默认开启一个事务，但是该事务不会自动提交
                                                            //在进行增删改操作时，要手动提交事务
    //4.sqlSession调用方法：查询所有selectList  查询单个：selectOne 添加：insert  修改：update 删除：delete
    List<User> users = sqlSession.selectList( s: "user.findAll");
    for (User user : users) {
        System.out.println(user);
    }
    sqlSession.close();

}
```

注：若在"openSession(b:ture)"自动提交，则不需要"sqlSession.commit()" 手动提交。

2.代理开发方式（主流）：

```
@Test
public void test5() throws IOException {
    InputStream resourceAsStream = Resources.getResourceAsStream("sqlMapConfig.xml");
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(resourceAsStream);
    SqlSession sqlSession = sqlSessionFactory.openSession();

    IUserDao mapper = sqlSession.getMapper(IUserDao.class);
    List<User> all = mapper.findAll();
    for (User user : all) {
        System.out.println(user);
    }
```

Mapper接口开发需要遵循以下规范：

①Mapper.xml文件汇总的namespace与mapper接口的全限定名相同

②Mapper接口方法名和Mapper.xml中定义的每个statement的id相同

③Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql的parameterType的类型相同

④Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resulttype的类型相同

Note：

resultType:声明实体类的全路径，才能通过**反射**获取实体类属性，才能完成字段值和属性值的自动映射封装。

StatementId：是namespace.id。

attributeValue:获取对象信息 => **getTextTrim**：获得除去空格的文本。