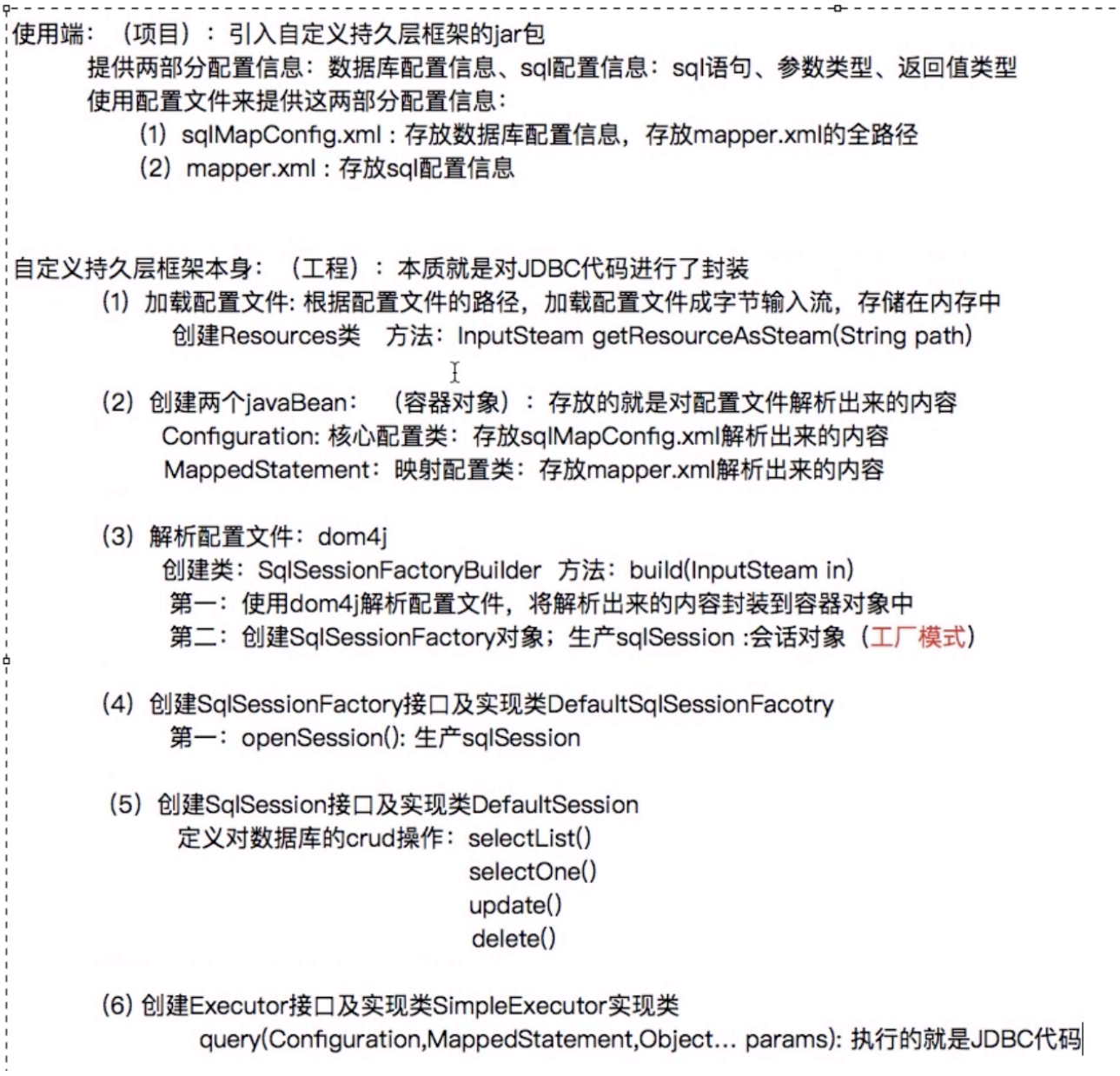
1. 框架分析

讲义1.1 + 1.2

2. 框架设计思路

总体分成 使用端 和框架本身 两大部分。

自定义持久层框架设计思路：



3. 框架实现

Create a project named "IPersistence_Test"

Create a new module named "IPersistence"

> 为了在"IPersistence_Test"中引入 "IPersistence" 的依赖，要先在"IPersistence" 的lifycycle下面选install，相当于把它打包。

3.1 创建配置文件

1. 核心配置文件 sqlMapConfig.xml，包括加载外部的properties文件，实体类别名，运行环境配置，映

射配置文件等。
2. 映射配置文件 IUserMapper.xml，包括sql语句，statement类型，输入参数Java类型，输出参数Java类型等。
3. pom.xml 中导入需要的dependency

3.2 pojo folder - 实体类

1. Configuration
2. MappedStatement

3.3 io folder - Resources

根据配置文件的路径，将配置文件加载成字节输入流，存储在内存中。

3.4 config folder

1. XMLConfigBuilder: parseConfig, 使用dom4j对配置文件进行解析，封装Configuration
2. XMLMapperBuilder: parse InputStream

3.5 sqlSession folder

1. SqlSessionFactory Interface
2. SqlSession Interface
3. Executor Interface
4. DefaultSqlSessionFactory implements SqlSessionFactory
5. DefaultSqlSession implements SqlSession
6. simpleExecutor implements Executor --> utils.xxx
7. SqlSessionFactoryBuilder

3.6 utils folder

1. TokenHandler Interface
2. ParameterMappingTokenHandler implements TokenHandler
3. ParameterMapping
4. GenericTokenParser

Step 6

- Vedio_10 实现 simpleExecutor.query 方法中的Step 1 + 2 + 3

- Vedio_11 Step 4 参数设置，即 query(Configuration, MappedStatement, Object... params) input里面的params
- Vedio_12 Step 5 + 6, 执行sql 和 封装返回结果
- 反射（Reflection）和内省（Introspector）是什么？看博客

4. 框架优化

4.1 IUserDao Interface

把与数据库的交互行为放在IUserDao Interface中。
然后创建一个实现类 UserDaoImpl Class。
但是这里有2个问题：

1. Dao实现类中存在代码重复，如 加载配置文件、创建sqlSessionFactory、生产sqlSession
2. statementId 是 hardcode的，如果把 userMapper.xml 配置文件里面的namespace更改了，那么Dao中的statementId也要改。

Solution：
删掉实现类，使用代理模式生成Dao Interface的代理实现类: 在 sqlSession interface 中添加新发方法 getMapper，然后在其实现类 DefaultSqlSession 中使用JDK动态代理来为Dao interface生成代理对象，并返回：

> 1. getMapper 方法 return的是 mapperClass 类的ProxyInstance

2. 代理对象调用Interface中任意方法，都会执行invoke方法。invoke方法的参数：
3. proxy：当前代理对象的引用
4. method：当前被调用方法的引用
5. args：调用方法时的input（即参数）

Proxy的invoke方法：

```
public class DefaultSqlSession implements SqlSession {
...

    @Override
    public <T> T getMapper(Class<?> mapperClass) {
        // 使用JDK动态代理来为Dao接口生成代理对象，并返回

        Object proxyInstance =
Proxy.newProxyInstance(DefaultSqlSession.class.getClassLoader(), new Class[]
{mapperClass}, new InvocationHandler() {
            @Override
            public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable {
                /*
                * 底层都还是去执行JDBC代码。
                * 根据不同情况，来调用selctList或者selectOne。但是由于在proxy这里没有办法
get到userMapper.xml
                * 中的namespace 和 id，就不能通过statementId来唯一确定想要执行那条语句。
                * 所以，为了能够通过 statementId 定位到想要执行的sql语句，这里将
userMapper.xml中的
                * namespace 和 id 的名字分别修改成为 接口全限定名
(com.lagou.dao.IUserDao) 和 方法名(findAll / findByCondition)
                *
                * 这样，就可以直接用Interface name 和 方法名 来组装 statementId.
                * */
                // 准备参数 1：statmentid = interface_class_name.method_name

                // 方法名：findAll / findByCondition
                String methodName = method.getName();
                // interface class name
                String className = method.getDeclaringClass().getName();

                String statementId = className+"."+methodName;

                // 准备参数2：params:args
                // 获取被调用方法的返回值类型
                Type genericReturnType = method.getGenericReturnType();
                // 判断是否进行了 泛型类型参数化
                // return true when genericReturnType是一个集合类型
                if(genericReturnType instanceof ParameterizedType){
                    List<Object> objects = selectList(statementId, args);
                    return objects;
                }

                return selectOne(statementId,args);
```

```
            }
        });

        return (T) proxyInstance;
    }
}
```