1. 背景
2. 基本要求

2.1命名风格

2.2命名含义

1. 代码组积结构

3.1粒度大小控制

3.1.1类行数尽量控制在500行内

如行数太多，考虑根据功能业务拆成多个类

3.1.2方法行数尽量控制50行内

如果行数过多，分析该方法是多个行为功能，考虑拆分成多个职责单一的小方法

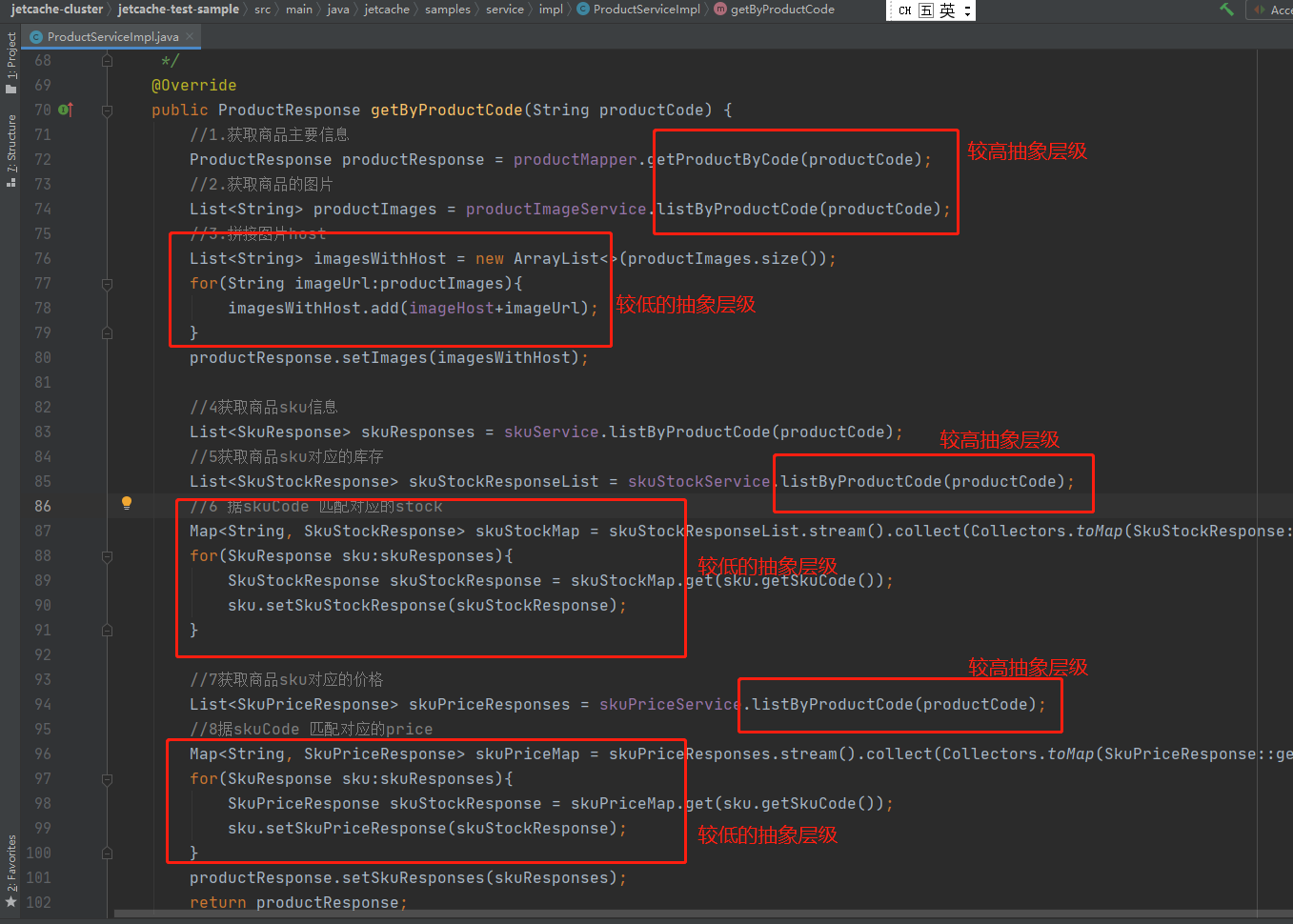
3.2 方法抽象级别保持一致

一个方法的实现，要么同时多个抽象的小方法法的组成，要么就是一个方法具体细节实现；一个方法内不能同时有两种抽像层级

假如有个据商品编码获取 商品详情的需求如下

1. 获取商品主要信息；
2. 获取商品图；
3. 获取商品sku信息
4. 获取sku 库存
5. 获取sku价格

反例:下面的代码同时有两种抽像级别



正例: 把上面的改造一下,每个动作功能都是一个方法，抽象层级都是一亲，代码变得清爽了很多，如下



3.3 高内聚、低耦合

高内聚与低耦合应该是相辅相成的；如果能做到高内聚，必须是低耦合的；换个说法是，如果一个事物对外依赖越少，那么其内聚性越高，耦合性越低，反之亦然；

为什么代码要高内聚、低耦合?我们应该不难理解，依赖越少应该是越易容维护，包括复用、移值、及变化扩展，安；

如上面3.2的例子，正例需然做到了抽象级别一致了，但我们可以发现内聚性并不高，

如获取商品的图片有两步，第一步是查图片url，第二步是拼接图片主机，该行为依赖两类ProductService 和ProductImageService，假如再来一个需求:据商品编码获取商图片图，那么得重新再写一段2、3步骤的一样的代码:



很明显，内聚性也会影响代码的复用度

正列:把上面商品详情需求代码进行进一步改造

1.把图片主机拼接代码写到ProductImageServcie里

2.把sku的库存及价格匹配的代码写的SkuService里

改造完的结果如下:

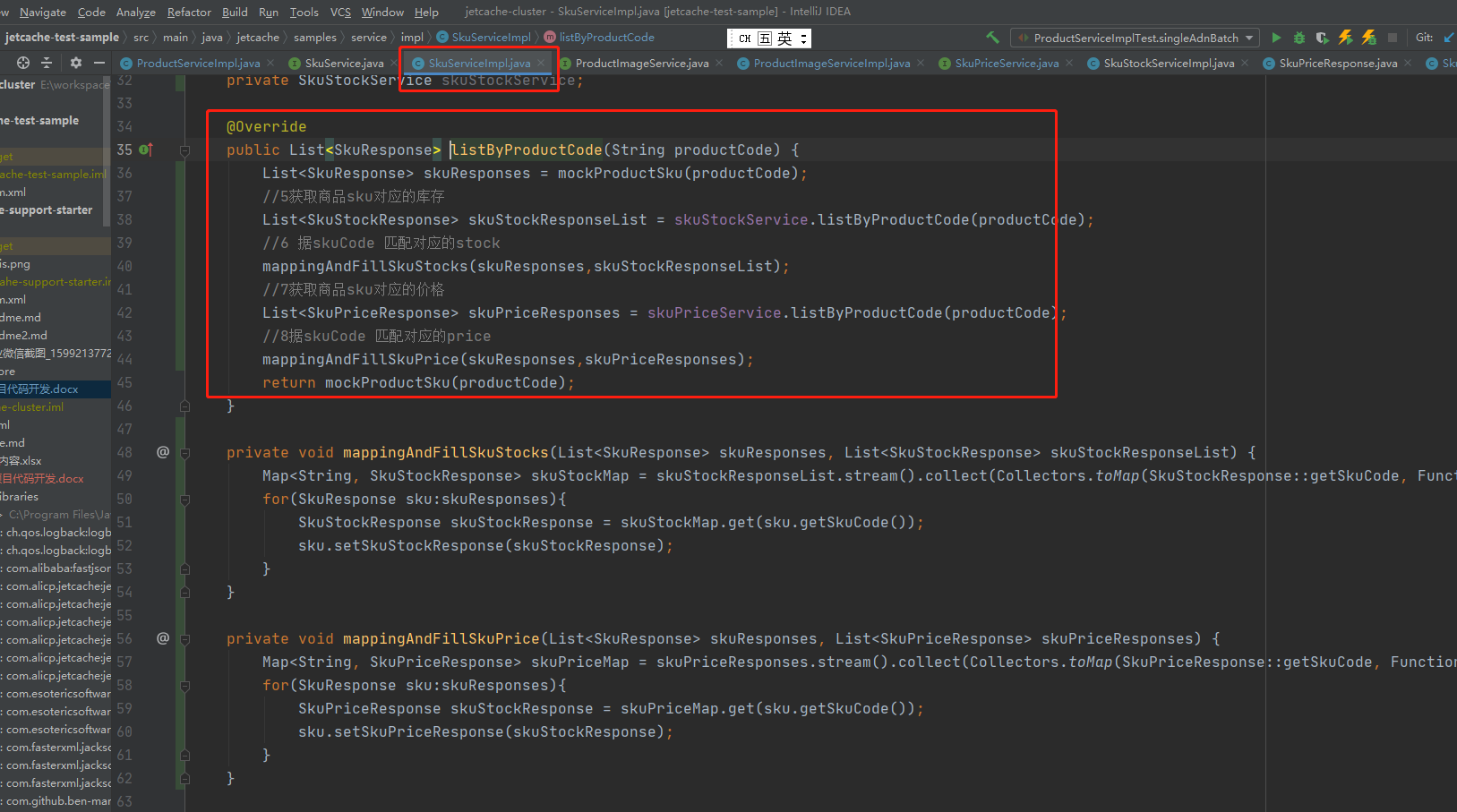
1.商品组合方法，主要由三个小方法组成，相对上面3.2的反例，代码简单了很多,代码时为原来的三份之一



1. 商品图片接口



1. 商品sku接口



虽然总体的代码量没有减少，但代变得单了，内聚性变高了耦合性

**安全性:**

另外高内聚的一个好处是，对数据安全访问比较容易控制，如上面访问商品图片的接，一开始访问图片有两个公开的方法，那么如果控制访问必须两个方法都得控制，访问入口越多越难控制，很有可能会漏掉一些入口；

耦合，

3.4 代码嵌套层次控制

通常如果方法粒度过大，抽象级别不一致，职责过多，往往会造方法嵌套层次，如果能按照上面两点去做，可很大程度代码嵌套过深难以维护;

另外，通过提前校验不满足条件提前返回，也可以减少嵌套

3.5 方法参数

时序性耦合及顺序依赖

3.5 职责单一

3.6 面向接口编程

3.7变量定义

不要远离使用的地方定义

1. 性能控制

4.1禁止循环远程调用

不管是远程调用服务还是数据库或远程缓存都是比较消耗资源的操作，如需要获取多个结果的操作要改为批量操作

1. 异常日志

1. 单元测试