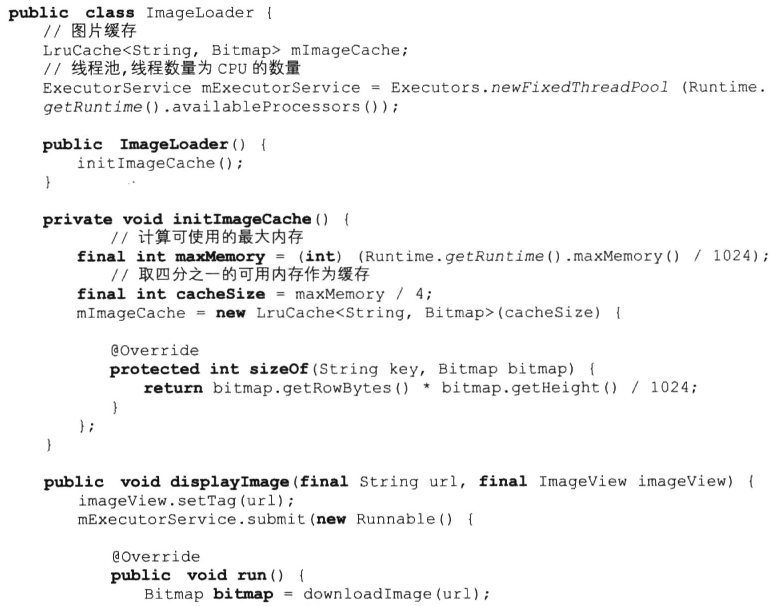
代码的六大原则

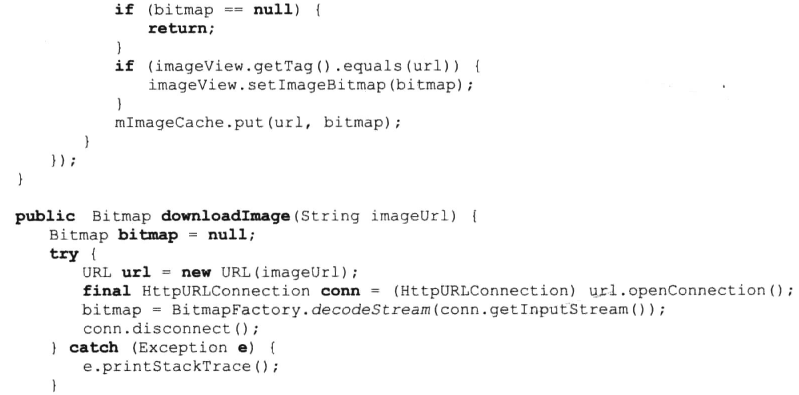
一、单一职责原则

定义：就一个类而言，应该仅有一个引起它变化的原因。一个类应该是一组相关性很高的函数、数据的封装。

案例：图片加载器（ImageLoader）

反例：主角小民应主管要求写了一个图片加载器（ImageLoader）。





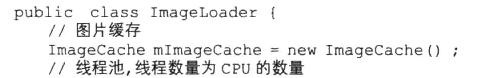


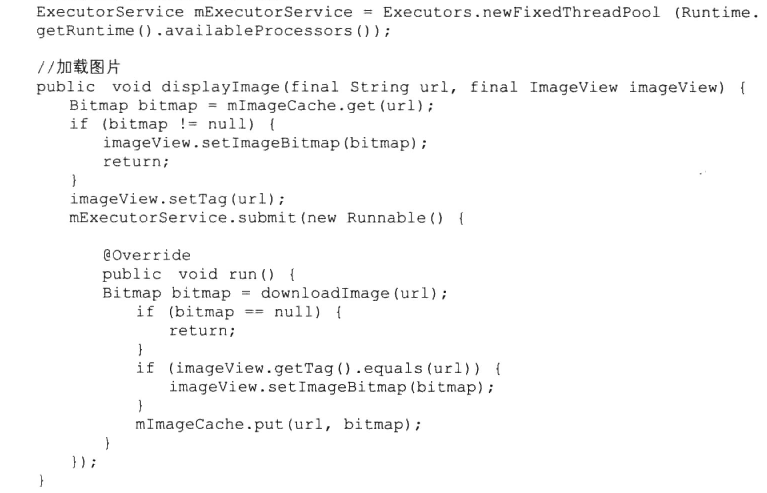
主管看后，说道，年轻人就是年轻人，这个类的耦合度太高了，没有遵从单一职责的原则，所有的功能都写在一个类里，当需求越来越多，这个类会变得特别臃肿，难以维护。

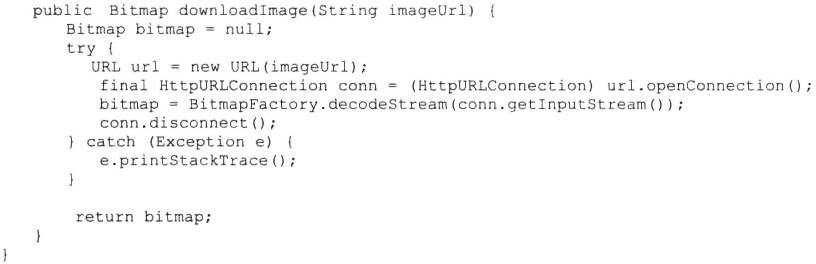
小民听了，感到惭愧，自己果然是一个小白，他Google了一下相关的关键词“单一职责”（百度听了会怎么想），经过思考，把原来的版本进行改进。

改进后的版本：

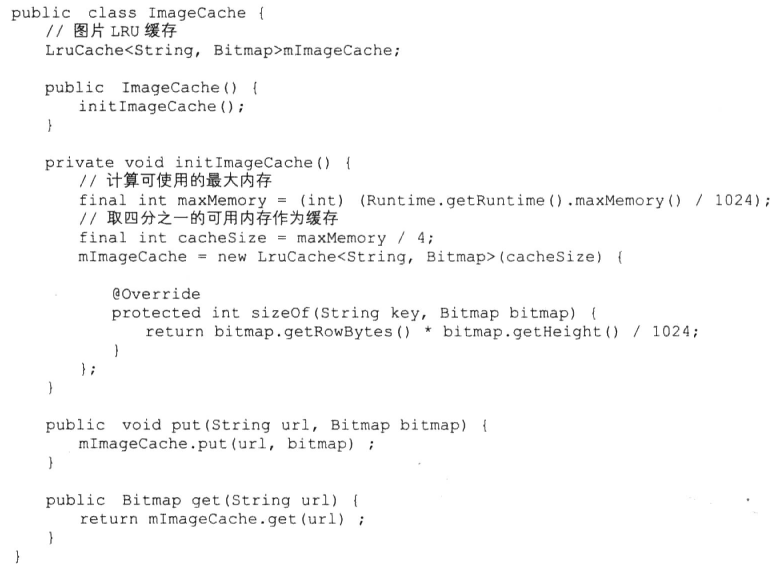
**ImageLoader.java:**







**ImageCache.java:**



这次把图片加载和图片缓存分割开来，比上一个版本更加灵活了。假如要修改图片缓存的功能，只要修改ImageCache类就可以了，对ImageLoader类不会有大的改动。主管看后，微微点头，小伙子还是有悟性滴，虽然效果没有特别理想，但还是有进步。

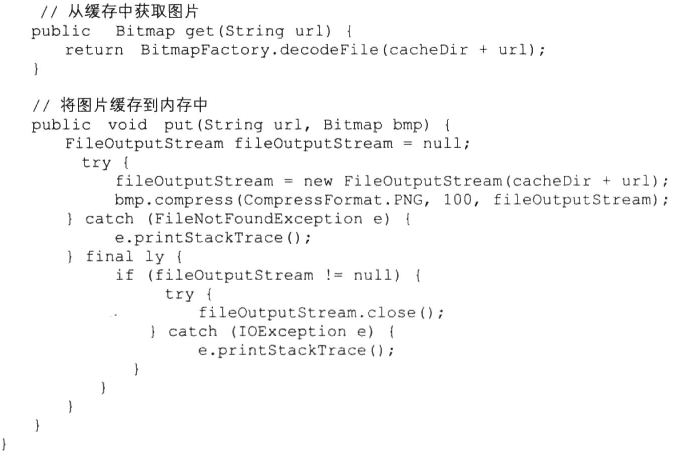
二、开闭原则

定义：软件中的对象对于扩展是开放的，对于修改是封闭的，当然这个是尽可能做到。

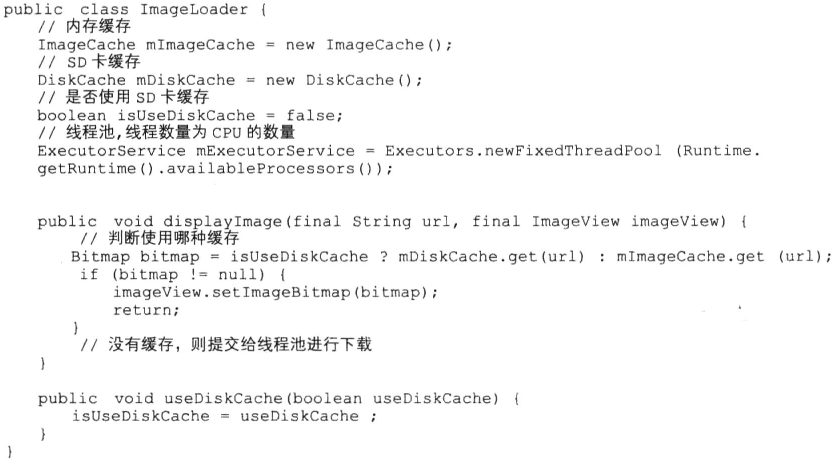
小民把他的图片加载器（ImageLoader）上传到了github上，很快就有人看了并且点赞。

有人建议增加SD卡缓存，于是他就加上了SD卡缓存。**DiskCache.java:**

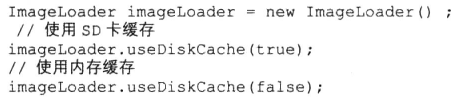




这个时候ImageLoader类就变成了这个样子：

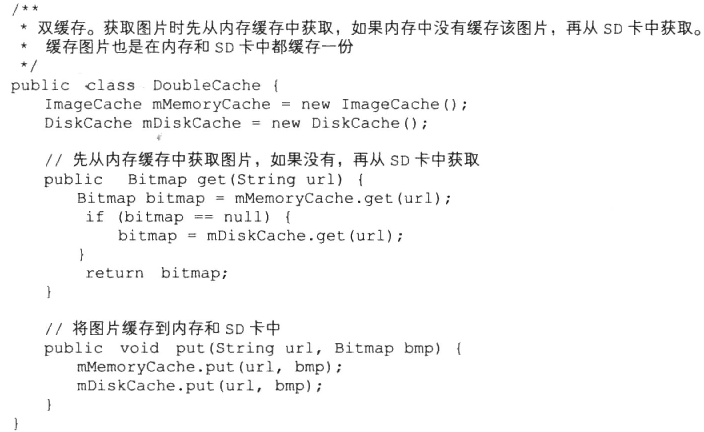


使用时候的代码：

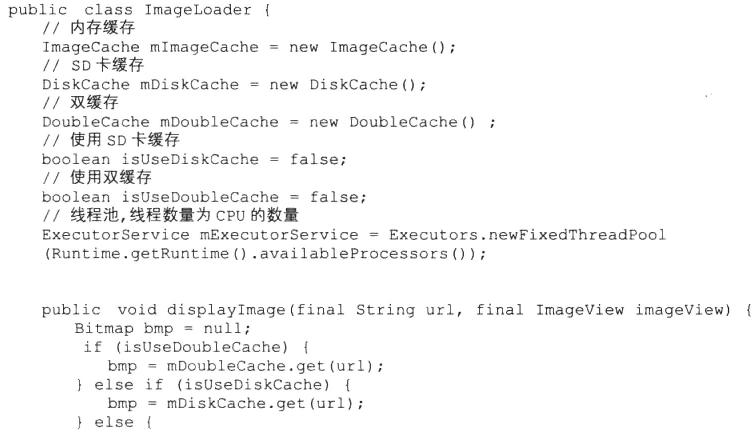


这个库只能使用SD卡缓存和内存缓存的其中一种，并不能同时使用两种策略。

于是小民使用了双缓存策略：



现在的ImageLoader类修改为如下模样：

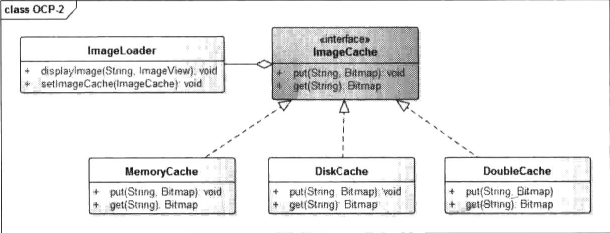




虽然能够实现功能，但是没有遵循开闭原则。这样的结果是，每次增加功能都必须修改ImageLoader类的代码，ImageLoader类会变的越来越臃肿，越来越难以维护。

主管决定亲自写图片加载器，让小民见识一下什么是真正的开闭原则。

UML图：

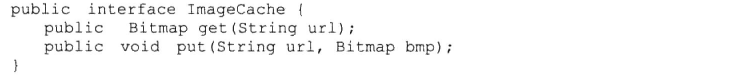


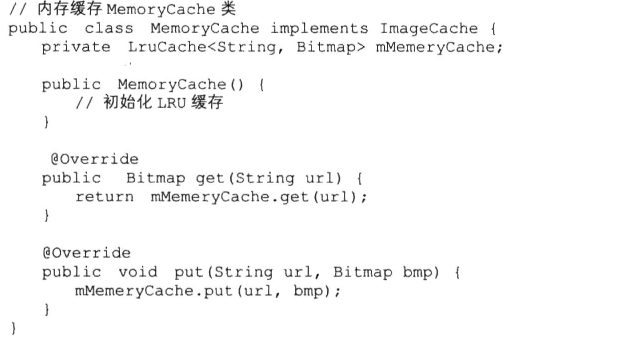
**ImageLoader.java:**

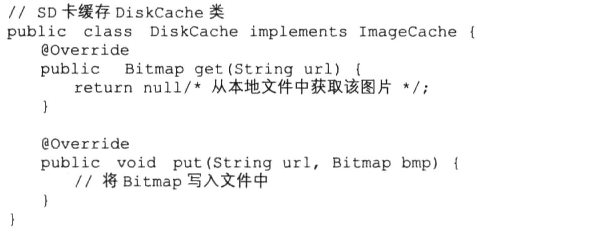




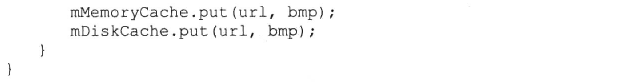
ImageCache.java变为了一个接口：











使用：

