Android Source

Design Patterns

Analysis & Practice

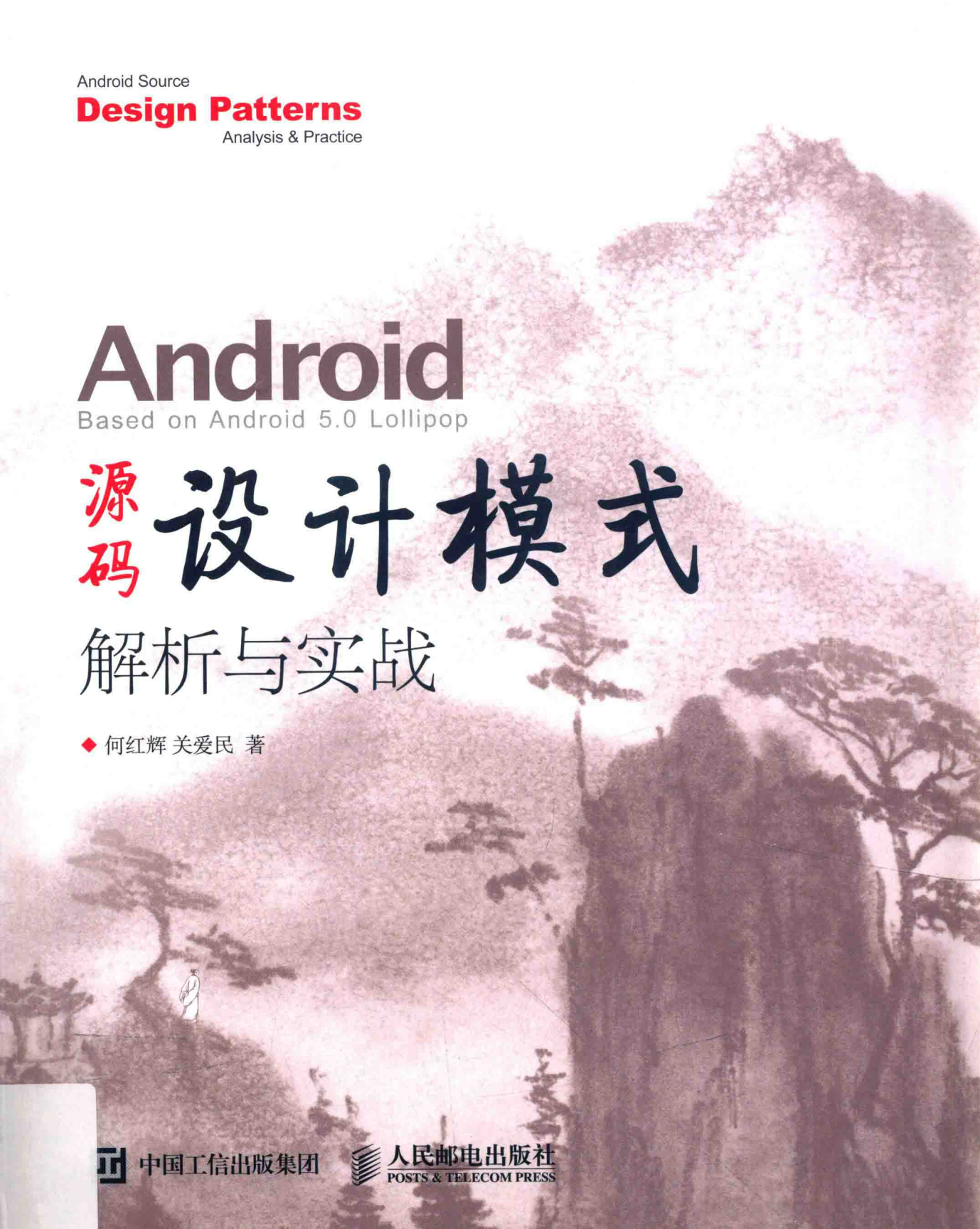
Android

Based on Android 5.0 Lollipop

源码设计模式

解析与实战

◆ 何红辉 关爱民 著



中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

Android Source
Design Patterns
Analysis & Practice

Android

Based on Android 5.0 Lollipop

源码设计模式

解析与实战

◆ 何红辉 关爱民 著

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

Android源码设计模式解析与实战 / 何红辉，关爱民著. -- 北京 : 人民邮电出版社，2015.11（2015.12重印）
ISBN 978-7-115-40671-2

Ⅰ. ①A… Ⅱ. ①何… ②关… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2015)第244188号

内 容 提 要

本书专门介绍 Android 源代码的设计模式，共 26 章，主要讲解面向对象的六大原则、主流的设计模式以及 MVC 和 MVP 模式。主要内容为：优化代码的第一步、开闭原则、里氏替换原则、依赖倒置原则、接口隔离原则、迪米特原则、单例模式、Builder 模式、原型模式、工厂方法模式、抽象工厂模式、策略模式、状态模式、责任链模式、解释器模式、命令模式、观察者模式、备忘录模式、迭代器模式、模板方法模式、访问者模式、中介者模式、代理模式、组合模式、适配器模式、装饰模式、享元模式、外观模式、桥接模式，以及 MVC 的介绍与实战和 MVP 应用架构模式。每个章节都对某个模式做了深入的分析，并且会对模式相关的技术点进行深入拓展，让读者在掌握模式的同时学习到 Android 中的一些重要知识，通过实战帮助读者达到学以致用的目的，且能够将模式运用于项目中，开发出高质量的程序。

本书适合的读者为初、中、高级 Android 工程师，也可以作为大专院校相关师生的学习用书和培训学校的教材。

◆ 著　　何红辉　关爱民
责任编辑　张　涛
责任印制　张佳莹　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：31.75
字数：441 千字　　2015 年 11 月第 1 版
印数：4001-5500 册　　2015 年 12 月北京第 3 次印刷

定价：79.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

推荐序

设计模式本身并不复杂，但是设计模式的出现，却是 GOF 大师们耗费无数心血，研究成百上千的例子，历经千锤百炼取其精华而得之，所以，它的重要性毋庸置疑。几年前，我曾见过高焕堂老师一本类似书籍的原稿，可惜此书未能出版，心中一直对此遗憾。但今天有幸一窥 CSDN 社区专家何红辉、关爱民老师精心撰写的这本以 Android 源码为案例的设计模式解析与实战一书时，激动之情勃然而发。是的，本书的确是国内第一本以 Android 为平台介绍设计模式的书，并且书中实例还不是简单的 Sample，而是作者在自己开发实践中的经历和一些实际使用的精彩代码段，实用性很强。

另外，我觉得《Android 源码设计模式解析与实战》中的主人公小民就是那些不断追求技术进步，从而得以不断成长的 IT 技术人的代表，小民的成长过程基本上反映了我们现在程序员的成长经历，他的成功很值得我们学习和借鉴。

学习设计模式，是程序员自我修炼、提升实力过程中必不可少的一关。读完此书的您或许已是设计模式的熟手，但我个人觉得，程序员的自我修炼远未结束，因为在设计模式之后，更有像 Pattern Oriented Software Architecture 5 卷本这样的、着眼于更高层次的书籍需要我们认真、刻苦地学习。希望何红辉和关爱民继续努力，将来为广大工程师提供层次更深、味道更丰富的精神食粮。

邓凡平

畅销书《深入理解 Android》系列的总策划和主笔，著有《深入理解 Android：卷 I》《深入理解 Android：卷 II》和《深入理解 Android：Wi-Fi，NFC 和 GPS 卷》。

自序一

想写一本 Android 设计模式的书的念头由来已久，也许是从我开始接触 Android 开发后就有了，于是很早就在自己的记事本上记录了一些相关学习心得。2014 年 4 月我就在博客上连载了《Android 源码分析之设计模式》系列，简单分析 Android 源码中的一些设计模式。到了 2014 年年底开始写一些开发框架相关的博客，并且在此期间发布了 AndroidEventBus 开源库，此后就一直活跃于 Github、博客圈。2015 年 3 月，我开始在 Github 上创建 Android 源码设计模式分析的开源项目，借助开源力量在一个月之内发布了十多篇 Android 源码中设计模式分析的文章，一经发布便得到了业界的普遍好评。

这些文章得到了业界的认可，让我又想起了最初出书的念头。原因很简单，Android 是一个开源的系统，很多优秀的思想、架构、设计模式必然在它的源码中得以体现，而在开源社区发布的文章还不够深入。从学习“Hello World”开始，我们都是先从学习他人如何做，然后再到学着做，最后经过自己的理解与思考再到自己做，因此，学习这些优秀的实现正是我们每个开发人员成长过程中的重要一步。在学习 Android 源码的优秀设计之后，我们如何将设计模式运用在 Android 开发上成了至关重要的问题，正所谓学以致用。因此，设计模式在 Android 开发中的实战又成了第二个关键。恰好，这两个领域目前都没有相关的书籍，我和关爱民老师就考虑出版这样的一本书籍。一来是通过写书实现自我提升以及对知识的梳理，二来也希望本书能够让更多的 Android 开发人员了解设计模式，从而提高自己的代码质量。如此一来，也算是尽了我们的绵薄之力。

何红辉

于北京

自序二

很多 Android 源码的实现都有设计模式的影子，对于很多从事 Android 开发的读者来说，学习 Android 源码的最大障碍往往是对其设计思想的理解而非源码本身，很多时候我们能看懂一段源码但却不明白其思想，看懂的是这一段源码的实现逻辑而不懂的则是为什么逻辑会是这样，对于开发者来说，知其然却又不知其所以然往往是进阶中最大的阻力。在今年早些时候，何红辉老师找到我说想构思一本关于 Android 源码设计模式的书，而此时的我刚好在深究 Android 源码的一些架构设计。市面上有很多关于 Android 基础方面的书籍，这些书籍帮助很多开发者入门并走向 Android 的开发殿堂，而对 Android 内核解析方面的书籍和资料也有不少，我曾经也阅读过很多关于 Android 内核的书籍，这些书籍对我从了解到深入内核层面有很大的帮助，但是，像一些基于设计和架构方面的 Android 书籍却很少看到，基于这样一个契机，我俩一拍即合决定写一本这样的书来帮助一些开发者进阶提升。我是一个喜欢分享、喜欢传授且不拘一束的人，借此机会，我想将我的一些经验或方法通过此书分享给大家，希望大家在今后的开发道路上少走一些弯路。

写书是需要付出极大的努力，且远比想象中困难，特别是在开发技术飞速发展的今天想要跟上技术迭代的脚步极不容易，在这里我想先感谢阅读本书的读者们，因为你们的认可才给予我们最大的写作动力；其次是帮我审稿、阅稿，以及在我学习过程中不断指点我的李志豪、常正宇和刘峰前辈，还有人民邮电出版社的张涛编辑，在出版的过程中给予我们很多建议和教导；再次要感谢的是一起写书的搭档何红辉老师，因为他的想法才有本书的诞生，并且在百忙之中还帮助我阅稿并指正错误；最后要感谢的是我的家人和朋友，因为你们的理解才能让我在几个月的时间中集中精力写好本书。

关爱民

于重庆

前　言

本书的特色

本书的第一部分是面向对象六大原则，通过具体的示例讲解六大原则的定义与作用，以及遵循这些原则会存在什么问题，遵循这些原则又会得到什么好处。通过第一部分使得读者对于面向接口编程以及 OOP 的基本原则有一个深入的认识。

第二部分就是本书的核心部分，每个章节分析一个设计模式，每个模式分为如下几个部分：

（1）模式基本介绍，介绍模式的定义、使用场景、UML 类图，使读者对相关模式有一个大致的认识；

（2）模式的简单实现，该模式的经典实现，使读者从代码的角度进一步理解相关模式；

（3）Android 源码中的模式实现，模式在 Android 源码中的运用与分析；

（4）深入拓展，进一步了解运用该模式模块的核心机制，使得读者彻底了解 Android 的一些基本原理；

（5）模式实战，通过一个示例让读者学习到如何将该模式运用于 Android 开发中；

（6）总结，总结该模式的优、缺点。

每个章节都对某个模式做了深入的分析，并且会对与模式相关的技术点进行深入拓展，让读者在掌握模式的同时学习到 Android 中的一些重要知识，通过实战则使读者达到学以致用的效果，能够将模式运用于开发当中。

本书的第三部分则是简单介绍了 MVC 及 MVP 模式，使读者从更高的应用架构角度看待应用开发，从整体上把握应用的基本结构。

本书涵盖了基本的 OOP 原则、设计模式的详细介绍、分析、实战，使得读者能够通过设计模式解决一些局部的设计问题，从而达到可扩展、灵活的局部架构。最后的 MVC 与 MVP 则从架构的高度帮助读者避免出现臃肿的类型、紧耦合等问题，使得应用真正实现高内聚、低耦合的应用架构。

面向的读者

在行业内很多初、中级工程师甚至高级工程师由于某些原因都还停留在功能实现层面，甚至对设计模式、面向对象知之甚少，因此，很少考虑代码的设计问题。本书从源码的角度由浅入深地剖析设计模式的运用，让工程师们把设计与模式重视起来，提升自己的设计能力与代码质量。因此，本书适合的读者为初、中、高级 Android 工程师。另外，本书强调的是编程思想，而思想都是相通的，因此，其他开发领域的工程师也能从中受益。

本书并不是一本 Android 开发或者设计模式的入门书籍，因此，阅读本书时你最好掌握了 Android 以及设计模式的相关知识。如果还没有，那么《第一行代码》和《Android 开发艺术探索》都是 Android 领域的优秀书籍；对于设计模式而言，《设计模式之禅》和《设计模式：可复用面向对象软件的基础》都是很好的选择。

如何阅读本书

本书分为 3 部分，分别为面向对象六大原则、23 种设计模式解析、MVC 与 MVP。其中第二部分的各个章节之间没有相关的联系，因此读者可以从中选择自己感兴趣的章节进行阅读。初、中级工程师建议至少阅读第一部分、第二部分的常用模式以及第三部分，高级工程师可以选择自己感兴趣的部分进行阅读。判定你是否需要阅读某个章节的标准是，当你看到标题时是否对这个知识点了然于心，如果答案是否定的，那么阅读该章节还是很有必要的。当然，通读全书自然是最好的选择。

最后需要说明的一点是，编写任何一本书籍都难免会有一些错误或不准确甚至不正确的地方，我们很乐意听到读者对我们的意见或建议，您可以通过发邮件（邮箱地址：simplecoder.h@gmail.com）的方式向我们反馈，在这里致以我们诚挚的谢意。编辑联系邮箱为：zhangtao@ptpress.com.cn。

编者

致　谢

任何一本书的诞生都不是某个个人努力的产物，本书也一样。经过数月的笔耕不辍，本书最终得以出版，在这期间有很多人为此付出了自己的辛勤劳动。

首先要感谢我的搭档关爱民老师，他不仅分担了本书一半的写作任务，而且在写作的过程中任劳任怨，挤出一切闲暇时间以保证按时甚至提前完成写作，只为能让本书尽快出版。其次是要感谢张涛编辑，在本书的出版过程中给了我们很多鼓励，并且为本书的出版付出了很多的努力。还要感谢秦汉、潘超群（Joker）两位前辈在百忙之中抽出时间帮助审稿，保证了本书的质量。最后要感谢我的家人，在我写作的时候给我建议并帮我校稿，在整个过程中给予我很大的支持。

何红辉

2015 年 8 月 4 日于北京

写书是需要付出极大的努力与勇气的，且远比想象中困难，特别是在开发技术飞速发展的今天，想要跟上技术迭代的脚步极不容易。在这里我想先感谢阅读本书的读者们，因为你们的认可才能给予我们最大的写作动力；其次是帮我审稿以及在我学习过程中不断指点我的李志豪、常正宇和刘峰前辈；人民邮电出版社的张涛编辑，在出版的过程中给予我们很多建议和帮助；再次要感谢的是一起写书的搭档何红辉老师，因为他的想法才有了本书的诞生，并且在百忙之中还帮助我阅稿、指正错误；最后要感谢我的家人和朋友，因为你们的理解才能让我在几个月的时间中集中精力写好本书。

关爱民

2015 年 8 月 4 日于重庆

目　录

第1章 走向灵活软件之路——面向对象的六大原则

1.1 优化代码的第一步——单一职责原则

单一职责原则的英文名称是 Single Responsibility Principle，缩写是 SRP。SRP 的定义是：就一个类而言，应该仅有一个引起它变化的原因。简单来说，一个类中应该是一组相关性很高的函数、数据的封装。就像秦小波老师在《设计模式之禅》中说的："这是一个备受争议却又及其重要的原则。只要你想和别人争执、怄气或者是吵架，这个原则是屡试不爽的"。因为单一职责的划分界限并不是总是那么清晰，很多时候都是需要靠个人经验来界定。当然，最大的问题就是对职责的定义，什么是类的职责，以及怎么划分类的职责。

对于计算机技术，通常只单纯地学习理论知识并不能很好地领会其深意，只有自己动手实践，并在实际运用中发现问题、解决问题、思考问题，才能够将知识吸收到自己的脑海中。下面以我的朋友小民的事情说起。

自从 Android 系统发布以来，小民就是 Android 的铁杆粉丝，于是在大学期间一直保持着对 Android 的关注，并且利用课余时间做些小项目，锻炼自己的实战能力。毕业后，小民如愿地加入了心仪的公司，并且投入到了他热爱的 Android 应用开发行业中。将爱好、生活、事业融为一体，小民的第一份工作也算是顺风顺水，一切尽在掌握中。

在经历过一周的适应期以及熟悉公司的产品、开发规范之后，小民的开发工作就正式开始了。小民的主管是个工作经验丰富的技术专家，对于小民的工作并不是很满意，尤其小民最薄弱的面向对象设计，而 Android 开发又是使用 Java 语言，程序中的抽象、接口、六大原则、23 种设计模式等名词把小民弄得晕头转向。小民自己也察觉到了自己的问题所在，于是，小民的主管决定先让小民做一个小项目来锻炼这方面的能力。正所谓养兵千日用兵一时，磨刀不误砍柴工，小民的开发之路才刚刚开始。

在经过一番思考之后，主管挑选了使用范围广、难度也适中的图片加载器（ImageLoader）作为小民的训练项目。既然要训练小民的面向对象设计，那么就必须考虑到可扩展性、灵活性，而检测这一切是否符合需求的最好途径就是开源。用户不断地提出需求、反馈问题，小民的项目需要不断升级以满足用户需求，并且要保证系统的稳定性、灵活性。在主管跟小民说了这一特殊任务之后，小民第一次感到了压力，"生活不容易！"年仅 22 岁的小民发出了如此深刻的感叹！

挑战总是要面对的，何况是从来不服输的小民。主管的要求很简单，要小民实现图片加载，并

且要将图片缓存起来。在分析了需求之后，小民一下就放心下来了，“这么简单，原来我还以为很难呢……”小民胸有成足地喃喃自语。在经历了 10 分钟的编码之后，小民写下了如下代码：

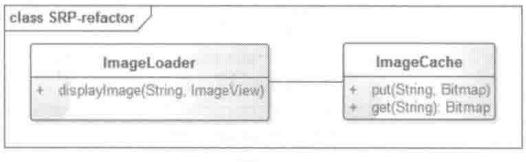
```
/**
 * 图片加载类
 */
public class ImageLoader {
    // 图片缓存
    LruCache<String, Bitmap> mImageCache;
    // 线程池,线程数量为 CPU 的数量
    ExecutorService mExecutorService = Executors.newFixedThreadPool (Runtime.
    getRuntime().availableProcessors());

    public ImageLoader() {
        initImageCache();
    }

    private void initImageCache() {
            // 计算可使用的最大内存
        final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);
            // 取四分之一的可用内存作为缓存
        final int cacheSize = maxMemory / 4;
        mImageCache = new LruCache<String, Bitmap>(cacheSize) {

            @Override
            protected int sizeOf(String key, Bitmap bitmap) {
                return bitmap.getRowBytes() * bitmap.getHeight() / 1024;
            }
        };
    }

    public void displayImage(final String url, final ImageView imageView) {
        imageView.setTag(url);
        mExecutorService.submit(new Runnable() {

            @Override
            public void run() {
                Bitmap bitmap = downloadImage(url);
                if (bitmap == null) {
                    return;
                }
                if (imageView.getTag().equals(url)) {
                    imageView.setImageBitmap(bitmap);
                }
                mImageCache.put(url, bitmap);
            }
        });
    }

    public Bitmap downloadImage(String imageUrl) {
        Bitmap bitmap = null;
        try {
            URL url = new URL(imageUrl);
            final HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            bitmap = BitmapFactory.decodeStream(conn.getInputStream());
            conn.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        return bitmap;
    }
}
```

并且使用 Git 软件进行版本控制，将工程托管到 Github 上，伴随着 git push 命令的完成，小民的 ImageLoader 0.1 版本就正式发布了！如此短的时间内就完成了这个任务，而且还是一个开源项目，小民暗暗自喜，并幻想着待会儿被主管称赞。

在小民给主管报告了 ImageLoader 的发布消息的几分钟之后，主管就把小民叫到了会议室。这下小民纳闷了，怎么夸人还需要到会议室。“小民，你的 ImageLoader 耦合太严重啦！简直就没有设计可言，更不要说扩展性、灵活性了。所有的功能都写在一个类里怎么行呢，这样随着功能的增多，ImageLoader 类会越来越大，代码也越来越复杂，图片加载系统就越来越脆弱……”这简直就是当头棒喝，小民的脑海里已经听不清主管下面说的内容了，只是觉得自己之前没有考虑清楚就匆匆忙忙完成任务，而且把任务想得太简单了。

“你还是把 ImageLoader 拆分一下，把各个功能独立出来，让它们满足单一职责原则。”主管最后说道。小民是个聪明人，敏锐地捕捉到了单一职责原则这个关键词，他用 Google 搜索了一些资料之后，总算是对单一职责原则有了一些认识，于是打算对 ImageLoader 进行一次重构。这次小民不敢过于草率，也是先画了一幅 UML 图，如图 1-1 所示。



▲图 1-1

ImageLoader 代码修改如下所示：

```
/**
 * 图片加载类
 */
public  class ImageLoader {
    // 图片缓存
    ImageCache mImageCache = new ImageCache() ;
    // 线程池,线程数量为 CPU 的数量
    ExecutorService mExecutorService = Executors.newFixedThreadPool (Runtime.
    getRuntime().availableProcessors());

    //加载图片
    public  void displayImage(final String url, final ImageView imageView) {
        Bitmap bitmap = mImageCache.get(url);
        if (bitmap != null) {
            imageView.setImageBitmap(bitmap);
            return;
        }
        imageView.setTag(url);
        mExecutorService.submit(new Runnable() {

            @Override
            public  void run() {
            Bitmap bitmap = downloadImage(url);
                if (bitmap == null) {
                    return;
                }
                if (imageView.getTag().equals(url)) {
                    imageView.setImageBitmap(bitmap);
                }
                mImageCache.put(url, bitmap);
            }
        });
    }
```

```
    public  Bitmap downloadImage(String imageUrl) {
        Bitmap bitmap = null;
        try {
            URL url = new URL(imageUrl);
            final HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            bitmap = BitmapFactory.decodeStream(conn.getInputStream());
            conn.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return bitmap;
    }
}
```

并且添加了一个 ImageCache 类用于处理图片缓存，具体代码如下：

```
public  class ImageCache {
    // 图片 LRU 缓存
    LruCache<String, Bitmap>mImageCache;

    public  ImageCache() {
        initImageCache();
    }

    private void initImageCache() {
        // 计算可使用的最大内存
        final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);
        // 取四分之一的可用内存作为缓存
        final int cacheSize = maxMemory / 4;
        mImageCache = new LruCache<String, Bitmap>(cacheSize) {

            @Override
            protected int sizeOf(String key, Bitmap bitmap) {
                return bitmap.getRowBytes() * bitmap.getHeight() / 1024;
            }
        };
    }

    public  void put(String url, Bitmap bitmap) {
        mImageCache.put(url, bitmap) ;
    }

    public  Bitmap get(String url) {
        return mImageCache.get(url) ;
    }
}
```

如图 1-1 和上述代码所示，小民将 ImageLoader 一拆为二，ImageLoader 只负责图片加载的逻辑，而 ImageCache 只负责处理图片缓存的逻辑，这样 ImageLoader 的代码量变少了，职责也清晰了；当与缓存相关的逻辑需要改变时，不需要修改 ImageLoader 类，而图片加载的逻辑需要修改时也不会影响到缓存处理逻辑。主管在审核了小民的第一次重构之后，对小民的工作给予了表扬，大致意思是结构变得清晰了许多，但是可扩展性还是比较欠缺。虽然没有得到主管的完全肯定，但也是颇有进步，再考虑到自己确实有所收获，小民原本沮丧的心里也略微地好转起来。

从上述的例子中我们能够体会到，单一职责所表达出的用意就是“单一”二字。正如上文所说，如何划分一个类、一个函数的职责，每个人都有自己的看法，这需要根据个人经验、具体的业务逻