

网络框架分析 – 全是套路

2016-12-18 安卓开发精选

(点击上方公众号，可快速关注)

来源：伯乐在线专栏作者 - PleaseCallMeCoder

链接：<http://android.jobbole.com/85268/>

[点击 → 了解如何加入专栏作者](#)

前言

这几天抽时间啃完了Volley和Picasso的源码，收获颇多，所以在这里跟大家分享一下。

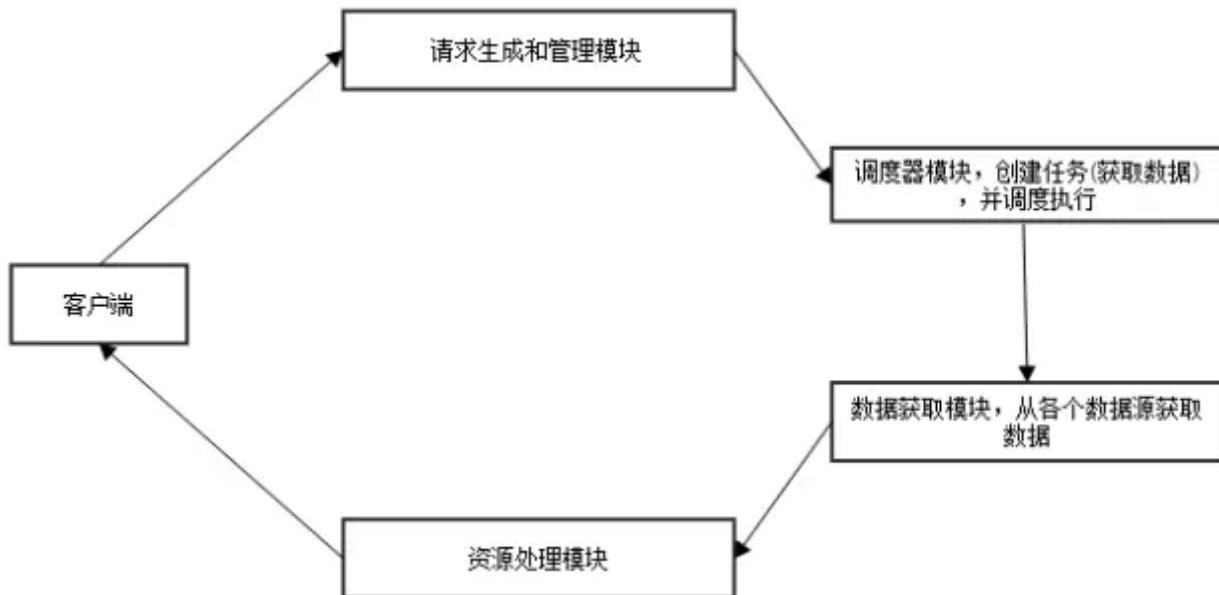
对于网络请求框架或者图片加载框架来说，我们的理想型大体应该是这样的：

- 简单：框架的出现当然是为了提升我们的开发效率，使我们的开发变得简单，所以在保证质量的情况下简单是第一位的
- 可配置：天下没有完全相同的两片树叶，也没有完全相同的两个项目，所以某些差异应该是可配置的，比如缓存位置、缓存大小、缓存策略等等
- 方便扩展：框架在设计的时候就要考虑到变化，并且封装起来。举个例子，比如有了更好的Http客户端，我们应该能很方便的修改并且不能对我们之前的代码产生太大影响

但万变不离其宗，这些框架的骨架其实基本上都是一样的，今天我们就来讨论下这些框架中的套路。

基本模块

既然我们说这些框架的结构其实基本上都是一样的，那么我们就先来看看它们之间类似的模块结构。



整体流程大概是这样的：

客户端请求->生成框架封装的请求类型->调度器开始处理任务->调用数据获取模块->对获取的数据进行处理->回调给客户端

生产者消费者模型

框架中请求管理和任务调度模块一般会用到生产者消费者模型。

为什么会有生产者消费者模型

在线程世界里，生产者就是生产数据的线程，消费者就是消费数据的线程。在多线程开发当中，如果生产者处理速度很快，而消费者处理速度很慢，那么生产者就必须等待消费者处理完，才能继续生产数据。同样的道理，如果消费者的处理能力大于生产者，那么消费者就必须等待生产者。为了解决这个问题于是引入了生产者和消费者模型。

什么是生产者消费者模型

生产者消费者模式是通过一个容器来解决生产者和消费者的强耦合问题。生产者和消费者彼此之间不直接通讯，而通过阻塞队列来进行通讯，所以生产者生产完数据之后不用等待消费者处理，直接扔给阻塞队列，消费者不找生产者要数据，而是直接从阻塞队列里取，阻塞队列就相当于一个缓冲区，平衡了生产者和消费者的处理能力。

生产者消费者模型的使用场景

Java中的线程池类其实也是一种生产者和消费者模式的实现方式，但是实现方法更高明。生产者把任务丢给线程池，线程池创建线程并处理任务，如果将要运行的任务数大于线程池的基本线程

数就把任务扔到阻塞队列里，这种做法比只使用一个阻塞队列来实现生产者和消费者模型显然要高明很多，因为消费者能够处理直接就处理掉了，这样速度更快，而生产者先存，消费者再取这种方式显然慢一些。

框架中的应用

对于上述的使用场景我们分别可以在框架中找到实现。

Volley源码中实现方式是用一个优先级阻塞队列来实现生产者消费者模型。生产者是往队列里添加数据的线程，消费者是一个默认4个元素的线程数组（不包括处理缓存的线程），来不停的取出消息处理。

而Picssso是一个比较典型的线程池实现的生产者消费者模型，这里就不做过多介绍了。

这两个框架使用的数据结构都是PriorityBlockingQueue（优先级阻塞队列），目的是为了做排序，保证优先级高的请求先被处理。

顺便说一下Android的消息处理机制其实也是一个生产者消费者模型。

一个小问题

这里博主当时想到了一个小问题：那就是唤醒消费者的时候唤醒的顺序是怎样的？

这里涉及到一个概念叫公平访问队列，所谓公平访问队列是指所有阻塞的生产者线程或者消费者线程，当队列可用时，可以按照阻塞的先后顺序访问队列，即先阻塞的生产者线程，可以先往队列里插入元素，先阻塞的消费者线程，可以先从队列里获取元素。通常情况下为了保证公平性会降低吞吐量。

缓存

Android缓存分为内存缓存和文件缓存（磁盘缓存）。

一般网络框架是不需要处理内存缓存的，但是图片加载框架需要。在Android3.1以后，Android推出了LruCache这个内存缓存类，LruCache中的对象是强引用的。Picasso的内存缓存就是使用的LruCache实现的。对于磁盘缓存，Google提供的一种解决方案是使用DiskLruCache（DiskLruCache并没有集成到Android源码中，在Android Doc的例子中有讲解）。Picasso的磁盘缓存是基于okhttp的，使用了DiskLruCache。而Volley的磁盘缓存是在DiskBasedCache中实现得，也是基于Lru算法的。

至于其他缓存算法、缓存命中率等等概念这里我就不做过多介绍了。

异步的处理

我们知道Android是单线程模型，我们应该避免在UI线程中进行耗时操作，网络请求算是一个比较典型的耗时操作，所以网络相关的框架中都会对异步操作进行一些封装。

其实这里没什么复杂的地方，无非就是利用Handler进行线程间通信，然后配合回调机制，把结果返回到主线程里。这里可以参考我之前的文章《Android Handler 消息机制（解惑篇）》和《当观察者模式和回调机制遇上Android源码》。

我们以Volley为例来简单看一下，ExecutorDelivery类的职责是分发子线程产生的responses数据或者错误信息。初始化是在RequestQueue类里。

```
public RequestQueue(Cache cache, Network network, int threadPoolSize) {
    this(cache, network, threadPoolSize,
        new ExecutorDelivery(new Handler(Looper.getMainLooper())));
}
```

这里传入的是主线程的Handler对象，而这个ExecutorDelivery对象会被传入到NetworkDispatcher和CacheDispatcher中，这两个类是继承于Thread的，负责处理队列中的请求。所以处理请求的操作是发生在子线程的。

然后我们看下ExecutorDelivery类的构造方法

```
public ExecutorDelivery(final Handler handler) {
    // Make an Executor that just wraps the handler.
    mResponsePoster = new Executor() {
        @Override
        public void execute(Runnable command) {
            handler.post(command);
        }
    };
}
```

这里用Executor对Handler进行了一层包装。Volley中的responses数据或者错误信息都会通过Executor发送出去，这样消息就到了主线程中。

Picasso比Volley要稍稍复杂了一点，由Picasso会对图片进行变换等操作，属于耗时操作，所以在Picasso中请求的分发和结果的处理会单独放到一个线程中。这个线程是一个带有消息队列的线程，用来执行循环性任务，即对获取到的数据进行处理。当它对结果处理完成之后，才会通过主线程的Handler把结果发送回主线程进行显示等操作。

设计模式

优秀的框架会合理的利用设计模式，使代码易于扩展和后期的维护。这里有一些出现频率比较高的设计模式。

- 静态工厂方法：由一个工厂对象决定创建出哪一种产品类的实例
- 单例模式：确保有且只有一个对象被创建
- 建造者模式：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示
- 外观模式：简化一群类的接口
- 命令模式：封装请求成为对象
- 策略模式：封装可以互选的行为，并使用委托来决定使用哪一个

框架入口

一般框架为了调用简洁，并不会让客户端直接通过new实例化一个入口对象。这里就需要用到创建型模式。

Volley的入口使用的是静态工厂方法，与Android源码中Bitmap的实例化类似，具体可以参考《Android源码中的静态工厂方法》

```
/**
 * Creates a default instance of the worker pool and calls {@link RequestQueue#start()}
 * on it.
 *
 * @param context A {@link Context} to use for creating the cache dir.
 * @return A started {@link RequestQueue} instance.
 */
public static RequestQueue newRequestQueue(Context context) {
    return newRequestQueue(context, null);
}
```

Picasso的入口方法则用到了双重锁的单例模式

```
static volatile Picasso singleton = null;
public static Picasso with(Context context) {
    if (singleton == null) {
        synchronized (Picasso.class) {
            if (singleton == null) {
                singleton = new Builder(context).build();
            }
        }
    }
}
```

```
    }  
}  
return singleton;  
}
```

同时由于可配置项太多，所以Picasso还使用了Builder模式。

同时一些框架为了给客户端提供一个简洁的API，会使用外观模式定义一个高层接口，使得框架中的各个模块更加容易使用。外观模式是一种结构型模式。

外观模式可以参考《Android源码中的外观模式》

命令模式

命令模式的定义是将一个请求封装成一个对象，从而使你可用不同的请求对客户进行参数化，对请求排队或记录请求日志，以及支持可撤销的操作。在网络请求框架中都会将请求做一个封装成对象，方便传递和使用。比如Volley中的Request，Picasso中的Request和Action。

命令模式可以参考《Android源码中的命令模式》

策略模式

策略模式也是大部分框架都会用到的一个模式，作用是封装可以互选的行为，并使用委托来决定使用哪一个。

Volley中就大量使用了面向接口编程的编程思想。这里我们看下Volley的入口方法

```
public static RequestQueue newRequestQueue(Context context, HttpStack stack, int maxDiskCacheBytes) {  
    //~省略部分无关代码~  
    if (stack == null) {  
        if (Build.VERSION.SDK_INT >= 9) {  
            stack = new HurlStack();  
        } else {  
            // Prior to Gingerbread, HttpURLConnection was unreliable.  
            // See: http://android-developers.blogspot.com/2011/09/androids-http-clients.html  
            stack = new HttpClientStack(AndroidHttpClient.newInstance(userAgent));  
        }  
    }  
  
    Network network = new BasicNetwork(stack);  
    //~省略部分无关代码~
```

}

这里会根据API版本选择不同的Http客户端，它们实现了一个共同的接口

```
/**  
 * An HTTP stack abstraction.  
 */  
  
public interface HttpStack {  
    /**  
     * Performs an HTTP request with the given parameters.  
     *  
     * <p>A GET request is sent if request.getPostBody() == null. A POST request is sent  
     * otherwise,  
     * and the Content-Type header is set to request.getPostBodyContentType().</p>  
     *  
     * @param request the request to perform  
     * @param additionalHeaders additional headers to be sent together with  
     *     {@link Request#getHeaders()}  
     * @return the HTTP response  
     */  
  
    public HttpResponse performRequest(Request<?> request, Map<String, String> additionalHeaders)  
        throws IOException, AuthFailureError;  
  
}
```

当然我们也可以自己实现这个接口，然后把Http客户端换成okhttp。

后记

网络相关的框架套路基本上就这些了，具体细节大家可以去自己看下相关源码。如果有什么不完善或者不对的地方也请大家多指教。

专栏作者简介（[点击 → 加入专栏作者](#)）

PleaseCallMeCoder：我是 PleaseCallMeCoder，一个小小的90后程序员。热衷于移动开发，喜欢研究新技术，奔跑在成为大神的路上。

微信扫一扫 支付



向霸王皇后 [星耀] 转账

赞赏你发在伯乐在线的文章

¥1.80

打赏支持作者写出更多好文章，谢谢！

关注「安卓开发精选」

看更多精选安卓技术文章



安卓开发精选

分享 Android 相关技术干货 · 资讯 · 高薪职位 · 教程



微信号：AndroidPD

长按识别二维码关注

伯乐在线 旗下微信公众号

商务合作QQ：2302462408

[阅读原文](#)
