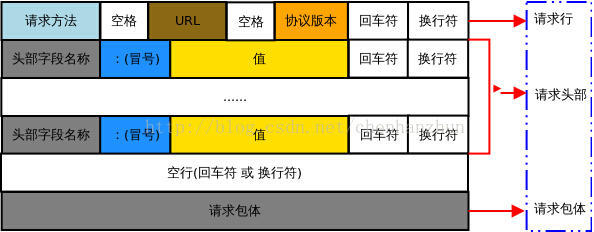
# **Android网络请求心路历程**

网络请求是android客户端很重要的部分。下面从入门级开始介绍下自己Android网络请求的实践历程。希望能给刚接触Android网络部分的朋友一些帮助。  
本文包含：

* [HTTP请求&响应](" \l "user-content-HTTP%E8%AF%B7%E6%B1%82&%E5%93%8D%E5%BA%94" \t "_blank)
* [Get&Post](" \l "user-content-Get&Post" \t "_blank)
* [HttpClient & HttpURLConnection](" \l "HttpClient & HttpURLConnection" \t "_blank)
* [同步&异步](" \l "%E5%90%8C%E6%AD%A5&%E5%BC%82%E6%AD%A5" \t "_blank)
* [HTTP缓存机制](" \l "HTTP%E7%BC%93%E5%AD%98%E6%9C%BA%E5%88%B6" \t "_blank)
* [Volley&OkHttp](" \l "Volley&OkHttp" \t "_blank)
* [Retrofit&RestAPI](" \l "Retrofit&RestAPI" \t "_blank)
* [网络图片加载优化](" \l "%E7%BD%91%E7%BB%9C%E5%9B%BE%E7%89%87%E5%8A%A0%E8%BD%BD%E4%BC%98%E5%8C%96" \t "_blank)
* [Fresco&Glide](" \l "Fresco&Glide" \t "_blank)
* [图片管理方案](" \l "%E5%9B%BE%E7%89%87%E7%AE%A1%E7%90%86%E6%96%B9%E6%A1%88" \t "_blank)

# HTTP请求&响应

既然说从入门级开始就说说Http请求包的结构。  
一次请求就是向目标服务器发送一串文本。什么样的文本？有下面结构的文本。  
HTTP请求包结构



请求包

例子：

POST /meme.php/home/user/login HTTP/1.1

Host: 114.215.86.90

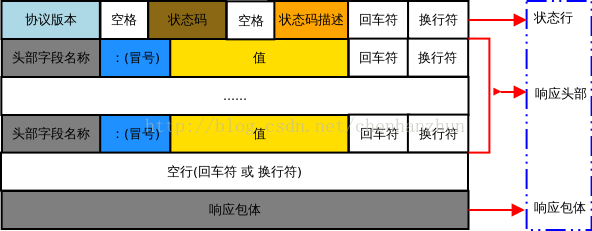
Cache-Control: no-cache

Postman-Token: bd243d6b-da03-902f-0a2c-8e9377f6f6ed

Content-Type: application/x-www-form-urlencoded

tel=13637829200&password=123456

请求了就会收到响应包(如果对面存在HTTP服务器)  
HTTP响应包结构



响应包

例子：

HTTP/1.1 200 OK

Date: Sat, 02 Jan 2016 13:20:55 GMT

Server: Apache/2.4.6 (CentOS) PHP/5.6.14

X-Powered-By: PHP/5.6.14

Content-Length: 78

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: application/json; charset=utf-8

{"status":202,"info":"\u6b64\u7528\u6237\u4e0d\u5b58\u5728\uff01","data":null}

Http请求方式有

| **方法** | **描述** |
| --- | --- |
| GET | 请求指定url的数据,请求体为空(例如打开网页)。 |
| POST | 请求指定url的数据，同时传递参数(在请求体中)。 |
| HEAD | 类似于get请求，只不过返回的响应体为空，用于获取响应头。 |
| PUT | 从客户端向服务器传送的数据取代指定的文档的内容。 |
| DELETE | 请求服务器删除指定的页面。 |
| CONNECT | HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。 |
| OPTIONS | 允许客户端查看服务器的性能。 |
| TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |

常用只有Post与Get。

# Get&Post

网络**请求**中我们常用键值对来传输参数(少部分api用json来传递,毕竟不是主流)。  
通过上面的介绍，可以看出虽然Post与Get本意一个是表单提交一个是请求页面，但本质并没有什么区别。下面说说参数在这2者的位置。

Get方式  
在url中填写参数：

http://xxxx.xx.com/xx.php?params1=value1&params2=value2

甚至使用路由

http://xxxx.xx.com/xxx/value1/value2/value3

这些就是web服务器框架的事了。

Post方式  
参数是经过编码放在请求体中的。编码包括x-www-form-urlencoded 与 form-data。  
x-www-form-urlencoded的编码方式是这样：

tel=13637829200&password=123456

form-data的编码方式是这样：

----WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="tel"

13637829200

----WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="password"

123456

----WebKitFormBoundary7MA4YWxkTrZu0gW

x-www-form-urlencoded的优越性就很明显了。不过x-www-form-urlencoded只能传键值对，但是form-data可以传二进制

因为url是存在于请求行中的。  
所以Get与Post区别本质就是参数是放在**请求行**中还是放在**请求体**中  
当然无论用哪种都能放在**请求头**中。一般在请求头中放一些发送端的常量。

有人说：

* Get是明文，Post隐藏  
  移动端不是浏览器,不用https全都是明文。
* Get传递数据上限XXX  
  胡说。有限制的是浏览器中的url长度，不是Http协议，移动端请求无影响。Http服务器部分有限制的设置一下即可。
* Get中文需要编码  
  是真的...要注意。URLEncoder.encode(params, "gbk");

还是建议用post规范参数传递方式。并没有什么更优秀，只是大家都这样社会更和谐。

上面说的是请求。下面说响应。  
请求是键值对，但返回数据我们常用Json。  
对于内存中的结构数据，肯定要用数据描述语言将对象序列化成文本，再用Http传递,接收端并从文本还原成结构数据。  
对象(服务器)<-->文本(Http传输)<-->对象(移动端) 。

服务器返回的数据大部分都是复杂的结构数据，所以Json最适合。  
Json解析库有很多Google的[Gson](https://github.com/google/gson" \t "_blank),阿里的[FastJson](https://github.com/alibaba/fastjson" \t "_blank)。  
Gson的用法看[这里](http://www.cnblogs.com/Jude95/p/Mr_Dentist.html" \t "_blank)。

# HttpClient & HttpURLConnection

HttpClient早被废弃了,谁更好这种问题也只有经验落后的面试官才会问。具体原因可以看[这里](http://blog.csdn.net/guolin_blog/article/details/12452307" \t "_blank)。

下面说说HttpURLConnection的用法。  
最开始接触的就是这个。

public class NetUtils {

public static String post(String url, String content) {

HttpURLConnection conn = null;

try {

// 创建一个URL对象

URL mURL = new URL(url);

// 调用URL的openConnection()方法,获取HttpURLConnection对象

conn = (HttpURLConnection) mURL.openConnection();

conn.setRequestMethod("POST");// 设置请求方法为post

conn.setReadTimeout(5000);// 设置读取超时为5秒

conn.setConnectTimeout(10000);// 设置连接网络超时为10秒

conn.setDoOutput(true);// 设置此方法,允许向服务器输出内容

// post请求的参数

String data = content;

// 获得一个输出流,向服务器写数据,默认情况下,系统不允许向服务器输出内容

OutputStream out = conn.getOutputStream();// 获得一个输出流,向服务器写数据

out.write(data.getBytes());

out.flush();

out.close();

int responseCode = conn.getResponseCode();// 调用此方法就不必再使用conn.connect()方法

if (responseCode == 200) {

InputStream is = conn.getInputStream();

String response = getStringFromInputStream(is);

return response;

} else {

throw new NetworkErrorException("response status is "+responseCode);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (conn != null) {

conn.disconnect();// 关闭连接

}

}

return null;

}

public static String get(String url) {

HttpURLConnection conn = null;

try {

// 利用string url构建URL对象

URL mURL = new URL(url);

conn = (HttpURLConnection) mURL.openConnection();

conn.setRequestMethod("GET");

conn.setReadTimeout(5000);

conn.setConnectTimeout(10000);

int responseCode = conn.getResponseCode();

if (responseCode == 200) {

InputStream is = conn.getInputStream();

String response = getStringFromInputStream(is);

return response;

} else {

throw new NetworkErrorException("response status is "+responseCode);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (conn != null) {

conn.disconnect();

}

}

return null;

}

private static String getStringFromInputStream(InputStream is)

throws IOException {

ByteArrayOutputStream os = new ByteArrayOutputStream();

// 模板代码 必须熟练

byte[] buffer = new byte[1024];

int len = -1;

while ((len = is.read(buffer)) != -1) {

os.write(buffer, 0, len);

}

is.close();

String state = os.toString();// 把流中的数据转换成字符串,采用的编码是utf-8(模拟器默认编码)

os.close();

return state;

}

}

注意网络权限！被坑了多少次。

<uses-permission android:name="android.permission.INTERNET"/>

# 同步&异步

这2个概念仅存在于多线程编程中。  
android中默认只有一个主线程，也叫UI线程。因为View绘制只能在这个线程内进行。  
所以如果你阻塞了(某些操作使这个线程在此处运行了N秒)这个线程，这期间View绘制将不能进行，UI就会卡。所以要极力避免在UI线程进行耗时操作。  
网络请求是一个典型耗时操作。  
通过上面的Utils类进行网络请求只有一行代码。

NetUtils.get("http://www.baidu.com");//这行代码将执行几百毫秒。

如果你这样写

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

String response ＝ Utils.get("http://www.baidu.com");

}

就会死。。  
这就是同步方式。直接耗时操作阻塞线程直到数据接收完毕然后返回。Android不允许的。  
异步方式：

//在主线程new的Handler，就会在主线程进行后续处理。

private Handler handler = new Handler();

private TextView textView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

textView = (TextView) findViewById(R.id.text);

new Thread(new Runnable() {

@Override

public void run() {

//从网络获取数据

final String response = NetUtils.get("http://www.baidu.com");

//向Handler发送处理操作

handler.post(new Runnable() {

@Override

public void run() {

//在UI线程更新UI

textView.setText(response);

}

});

}

}).start();

}

在子线程进行耗时操作，完成后通过Handler将更新UI的操作发送到主线程执行。这就叫异步。Handler是一个Android线程模型中重要的东西，与网络无关便不说了。关于Handler不了解就先去Google一下。  
[关于Handler原理一篇不错的文章](http://www.cnblogs.com/codingmyworld/archive/2011/09/12/2174255.html" \t "_blank)

但这样写好难看。异步通常伴随者他的好基友回调。  
这是通过回调封装的Utils类。

public class AsynNetUtils {

public interface Callback{

void onResponse(String response);

}

public static void get(final String url, final Callback callback){

final Handler handler = new Handler();

new Thread(new Runnable() {

@Override

public void run() {

final String response = NetUtils.get(url);

handler.post(new Runnable() {

@Override

public void run() {

callback.onResponse(response);

}

});

}

}).start();

}

public static void post(final String url, final String content, final Callback callback){

final Handler handler = new Handler();

new Thread(new Runnable() {

@Override

public void run() {

final String response = NetUtils.post(url,content);

handler.post(new Runnable() {

@Override

public void run() {

callback.onResponse(response);

}

});

}

}).start();

}

}

然后使用方法。

private TextView textView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

textView = (TextView) findViewById(R.id.webview);

AsynNetUtils.get("http://www.baidu.com", new AsynNetUtils.Callback() {

@Override

public void onResponse(String response) {

textView.setText(response);

}

});

是不是优雅很多。  
嗯，一个蠢到哭的网络请求方案成型了。  
愚蠢的地方有很多：

* 每次都new Thread，new Handler消耗过大
* 没有异常处理机制
* 没有缓存机制
* 没有完善的API(请求头,参数,编码,拦截器等)与调试模式
* 没有Https

# HTTP缓存机制

缓存对于移动端是非常重要的存在。

* 减少请求次数，减小服务器压力.
* 本地数据读取速度更快，让页面不会空白几百毫秒。
* 在无网络的情况下提供数据。

缓存一般由服务器控制(通过某些方式可以本地控制缓存，比如向过滤器添加缓存控制信息)。通过在请求头添加下面几个字端：

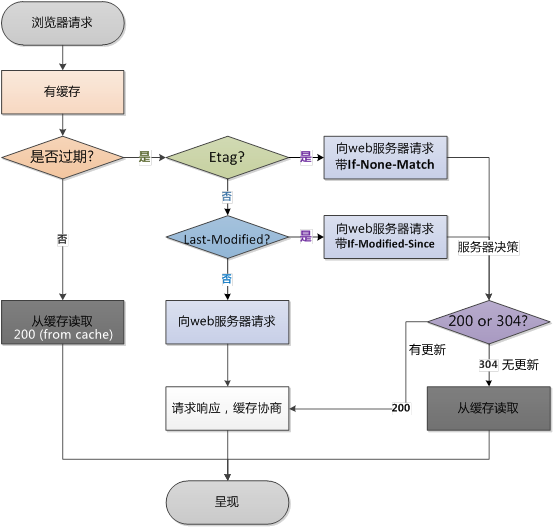
Request

| **请求头字段** | **意义** |
| --- | --- |
| If-Modified-Since: Sun, 03 Jan 2016 03:47:16 GMT | 缓存文件的最后修改时间。 |
| If-None-Match: "3415g77s19tc3:0" | 缓存文件的Etag(Hash)值 |
| Cache-Control: no-cache | 不使用缓存 |
| Pragma: no-cache | 不使用缓存 |

Response

| **响应头字段** | **意义** |
| --- | --- |
| Cache-Control: public | 响应被共有缓存，移动端无用 |
| Cache-Control: private | 响应被私有缓存，移动端无用 |
| Cache-Control:no-cache | 不缓存 |
| Cache-Control:no-store | 不缓存 |
| Cache-Control: max-age=60 | 60秒之后缓存过期（相对时间） |
| Date: Sun, 03 Jan 2016 04:07:01 GMT | 当前response发送的时间 |
| Expires: Sun, 03 Jan 2016 07:07:01 GMT | 缓存过期的时间（绝对时间） |
| Last-Modified: Sun, 03 Jan 2016 04:07:01 GMT | 服务器端文件的最后修改时间 |
| ETag: "3415g77s19tc3:0" | 服务器端文件的Etag[Hash]值 |

正式使用时按需求也许只包含其中部分字段。  
客户端要根据这些信息储存这次请求信息。  
然后在客户端发起请求的时候要检查缓存。遵循下面步骤：



浏览器缓存机制

注意服务器返回304意思是数据没有变动滚去读缓存信息。  
曾经年轻的我为自己写的网络请求框架添加完善了缓存机制，还沾沾自喜，直到有一天我看到了下面2个东西。（/TДT)/

# Volley&OkHttp

Volley&OkHttp应该是现在最常用的网络请求库。用法也非常相似。都是用构造请求加入请求队列的方式管理网络请求。

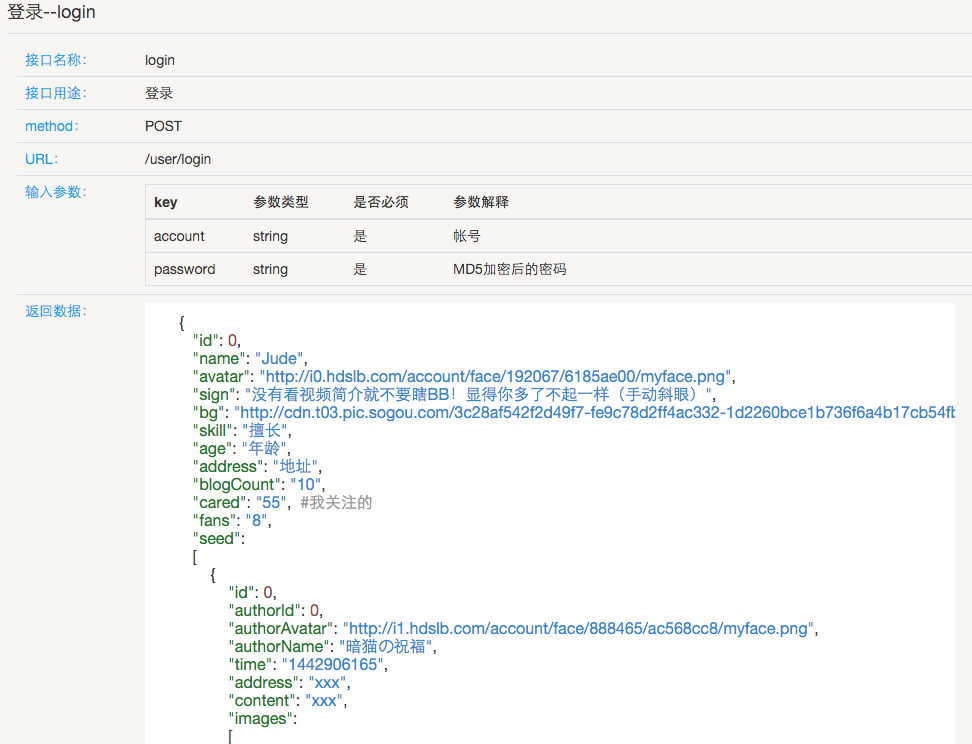
先说Volley:  
Volley可以通过[这个库](https://github.com/mcxiaoke/android-volley" \t "_blank)进行依赖.  
Volley在Android 2.3及以上版本，使用的是HttpURLConnection，而在Android 2.2及以下版本，使用的是HttpClient。  
Volley的基本用法，网上资料无数，这里推荐[郭霖大神的博客](http://blog.csdn.net/guolin_blog/article/details/17482095" \t "_blank)  
Volley存在一个缓存线程，一个网络请求线程池(默认4个线程)。  
Volley这样直接用开发效率会比较低，我将我使用Volley时的各种技巧封装成了一个库[RequestVolly](https://github.com/Jude95/RequestVolley" \t "_blank).  
我在这个库中将构造请求的方式封装为了函数式调用。维持一个全局的请求队列，拓展一些方便的API。

不过再怎么封装Volley在功能拓展性上始终无法与OkHttp相比。  
Volley停止了更新，而OkHttp得到了官方的认可，并在不断优化。  
因此我最终替换为了OkHttp

OkHttp用法见[这里](http://square.github.io/okhttp/" \t "_blank)  
很友好的API与详尽的文档。  
[这篇文章](http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0106/2275.html" \t "_blank)也写的很详细了。  
OkHttp使用[Okio](https://github.com/square/okio" \t "_blank)进行数据传输。都是[Square](http://square.github.io/" \t "_blank)家的。  
但并不是直接用OkHttp。Square公司还出了一个Retrofit库配合OkHttp战斗力翻倍。

# Retrofit&RestAPI

[Retrofit](http://square.github.io/retrofit/" \t "_blank)极大的简化了网络请求的操作，它应该说只是一个Rest API管理库，它是直接使用OKHttp进行网络请求并不影响你对OkHttp进行配置。毕竟都是[Square](http://square.github.io/" \t "_blank)公司出品。  
RestAPI是一种软件设计风格。  
服务器作为资源存放地。客户端去请求GET,PUT, POST,DELETE资源。并且是无状态的，没有session的参与。  
移动端与服务器交互最重要的就是API的设计。比如这是一个标准的登录接口。



Paste\_Image.png

你们应该看的出这个接口对应的请求包与响应包大概是什么样子吧。  
请求方式，请求参数，响应数据，都很清晰。  
使用Retrofit这些API可以直观的体现在代码中。



Paste\_Image.png

然后使用Retrofit提供给你的这个接口的实现类 就能直接进行网络请求获得结构数据。

注意Retrofit2.0相较1.9进行了大量不兼容更新。google上大部分教程都是基于1.9的。[这里](http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/1016/3588.html" \t "_blank)有个2.0的教程。

教程里进行异步请求是使用Call。Retrofit最强大的地方在于支持[RxJava](https://github.com/ReactiveX/RxAndroid" \t "_blank)。就像我上图中返回的是一个Observable。RxJava上手难度比较高，但用过就再也离不开了。Retrofit+OkHttp+RxJava配合框架打出成吨的输出，这里不再多说。

网络请求学习到这里我觉得已经到顶了。。

# 网络图片加载优化

对于图片的传输，就像上面的登录接口的avatar字段，并不会直接把图片写在返回内容里，而是给一个图片的地址。需要时再去加载。

如果你直接用HttpURLConnection去取一张图片，你办得到，不过没优化就只是个BUG不断demo。绝对不能正式使用。  
注意网络图片有些特点：

1. 它永远不会变  
   一个链接对应的图片一般永远不会变，所以当第一次加载了图片时，就应该予以永久缓存，以后就不再网络请求。
2. 它很占内存  
   一张图片小的几十k多的几M高清无码。尺寸也是64\*64到2k图。你不能就这样直接显示到UI，甚至不能直接放进内存。
3. 它要加载很久  
   加载一张图片需要几百ms到几m。这期间的UI占位图功能也是必须考虑的。

说说我在上面提到的[RequestVolley](https://github.com/Jude95/RequestVolley" \t "_blank)里做的图片请求处理(没错我做了，这部分的代码可以去github里看源码)。

##### 三级缓存

网上常说三级缓存－－服务器，文件，内存。不过我觉得服务器不算是一级缓存，那就是数据源嘛。

内存缓存  
首先内存缓存使用LruCache。LRU是Least Recently Used 近期最少使用算法，这里确定一个大小，当Map里对象大小总和大于这个大小时将使用频率最低的对象释放。我将内存大小限制为进程可用内存的1/8.  
内存缓存里读得到的数据就直接返回，读不到的向硬盘缓存要数据。

硬盘缓存  
硬盘缓存使用[DiskLruCache](http://developer.android.com/intl/zh-cn/samples/DisplayingBitmaps/src/com.example.android.displayingbitmaps/util/DiskLruCache.html" \t "_blank)。这个类不在API中。得复制使用。  
看见LRU就明白了吧。我将硬盘缓存大小设置为100M。

@Override

public void putBitmap(String url, Bitmap bitmap) {

put(url, bitmap);

//向内存Lru缓存存放数据时，主动放进硬盘缓存里

try {

Editor editor = mDiskLruCache.edit(hashKeyForDisk(url));

bitmap.compress(Bitmap.CompressFormat.JPEG, 100, editor.newOutputStream(0));

editor.commit();

} catch (IOException e) {

e.printStackTrace();

}

}

//当内存Lru缓存中没有所需数据时，调用创造。

@Override

protected Bitmap create(String url) {

//获取key

String key = hashKeyForDisk(url);

//从硬盘读取数据

Bitmap bitmap = null;

try {

DiskLruCache.Snapshot snapShot = mDiskLruCache.get(key);

if(snapShot!=null){

bitmap = BitmapFactory.decodeStream(snapShot.getInputStream(0));

}

} catch (IOException e) {

e.printStackTrace();

}

return bitmap;

}

DiskLruCache的原理不再解释了(我还解决了它存在的一个BUG，向Log中添加的数据增删记录时,最后一条没有输出,导致最后一条缓存一直失效。)

* 硬盘缓存也没有数据就返回空，然后就向服务器请求数据。

这就是整个流程。  
但我这样的处理方案还是有很多局限。

* 图片未经压缩处理直接存储使用
* 文件操作在主线程
* 没有完善的图片处理API

以前也觉得这样已经足够好直到我遇到下面俩。

# Fresco&Glide

不用想也知道它们都做了非常完善的优化，重复造轮子的行为很蠢。  
[Fresco](http://fresco-cn.org/" \t "_blank)是Facebook公司的黑科技。光看功能介绍就看出非常强大。使用方法官方博客说的够详细了。  
真三级缓存，变换后的BItmap(内存)，变换前的原始图片(内存)，硬盘缓存。  
在内存管理上做到了极致。对于重度图片使用的APP应该是非常好的。  
它一般是直接使用SimpleDraweeView来替换ImageView，呃～侵入性较强，依赖上它apk包直接大1M。代码量惊人。

所以我更喜欢[Glide](https://github.com/bumptech/glide" \t "_blank)，作者是bumptech。这个库被广泛的运用在google的开源项目中，包括2014年google I/O大会上发布的官方app。  
[这里](http://www.jianshu.com/p/4a3177b57949" \t "_blank)有详细介绍。直接使用ImageView即可，无需初始化，极简的API，丰富的拓展，链式调用都是我喜欢的。  
丰富的拓展指的就是[这个](https://github.com/wasabeef/glide-transformations" \t "_blank)。  
另外我也用过Picasso。API与Glide简直一模一样，功能略少，且有半年未修复的BUG。

# 图片管理方案

再说说图片存储。不要存在自己服务器上面，徒增流量压力，还没有图片处理功能。  
推荐[七牛](http://www.qiniu.com/pricing" \t "_blank)与[阿里云存储](http://www.aliyun.com/" \t "_blank)(没用过其它 π\_\_π )。它们都有很重要的一项图片处理。在图片Url上加上参数来对图片进行一些处理再传输。  
于是（七牛的处理代码）

public static String getSmallImage(String image){

if (image==null)return null;

if (isQiniuAddress(image)) image+="?imageView2/0/w/"+IMAGE\_SIZE\_SMALL;

return image;

}

public static String getLargeImage(String image){

if (image==null)return null;

if (isQiniuAddress(image)) image+="?imageView2/0/w/"+IMAGE\_SIZE\_LARGE;

return image;

}

public static String getSizeImage(String image,int width){

if (image==null)return null;

if (isQiniuAddress(image)) image+="?imageView2/0/w/"+width;

return image;

}

既可以加快请求速度，又能减少流量。再配合Fresco或Glide。完美的图片加载方案。  
不过这就需要你把所有图片都存放在七牛或阿里云，这样也不错。

图片/文件上传也都是使用它们第三方存储，它们都有SDK与官方文档教你。  
不过图片一定要压缩过后上传。上传1-2M大的高清照片没意义。