面试后的总结

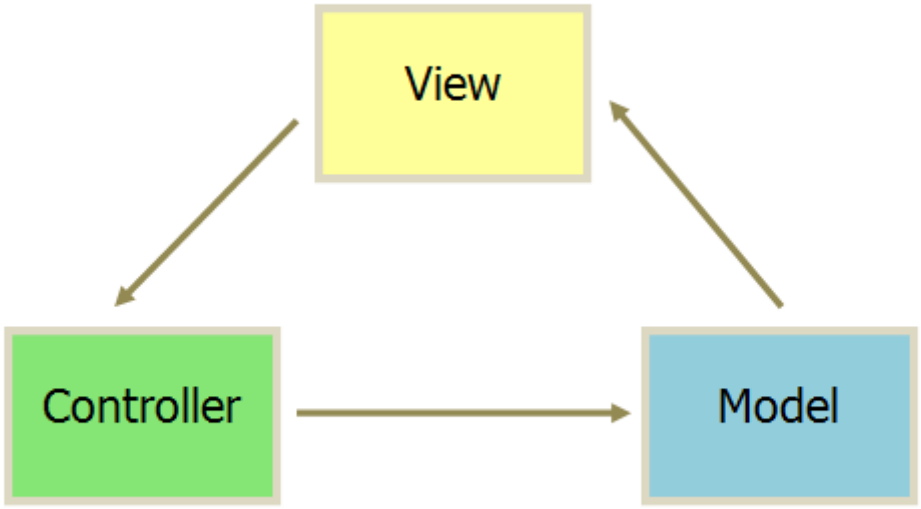
2016-03-08

摘要

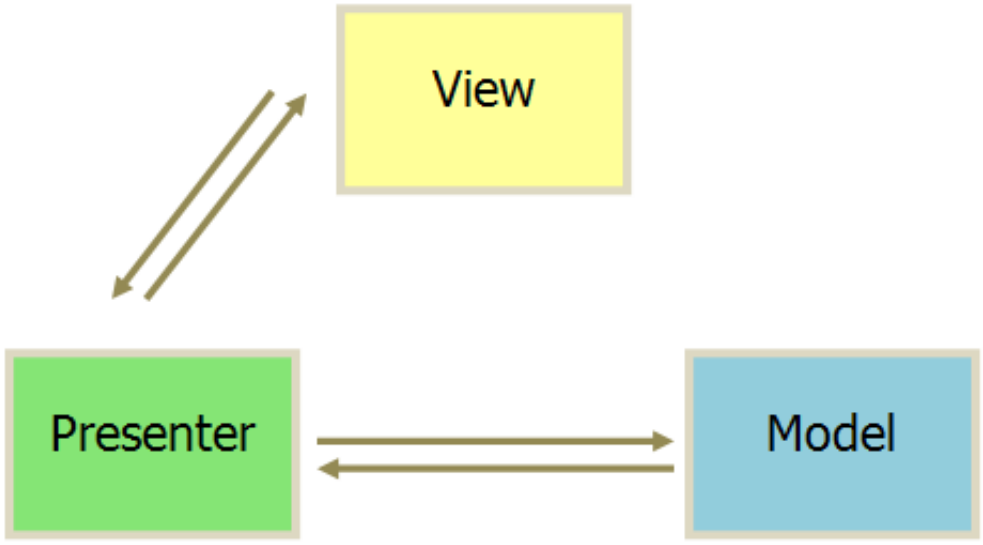
本文原创，转载请注明地址：http://kymjs.com/code/2016/03/08/01 (http://kymjs.com/code/2016/03/08/01)
“基础 Android 知识掌握的不错，学习能力也不错。但是基础知识部分比较薄弱，有些概念和逻辑掌握不清。” 感谢春林的这句话。

MVC，MVP 和 MVVM

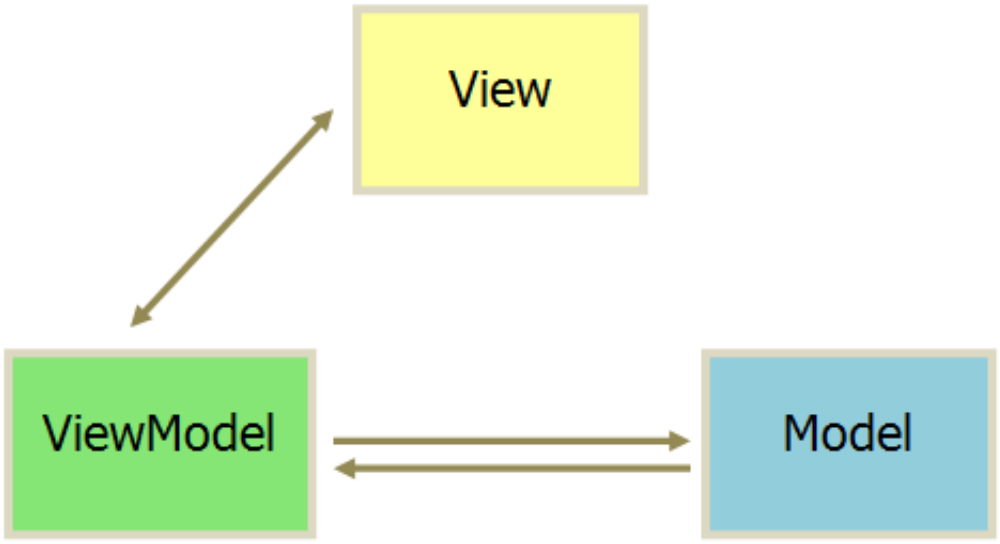
- MVC 通信方式，环形方式：
  1、View 传送指令到 Controller
  2、Controller 起到不同层面间的组织作用，用于控制应用程序的流程。它处理事件并作出响应。“事件”包括用户的行为和数据 Model 上的改变。
  3、Model 将新的数据发送到 View，用户得到反馈
  所有通信都是单向的。



- MVP 通信方式：
  1、各部分之间的通信，都是双向的。
  2、View 与 Model 不发生联系，都通过 Presenter 传递。
  3、View 非常薄，不部署任何业务逻辑，称为”被动视图”（Passive View），即没有任何主动性，而 Presenter非常厚，所有逻辑都部署在那里。



- MVVM 模式是 MVP 的升级：
  基本上与 MVP 模式完全一致。唯一的区别是，它采用双向绑定：View的变动，自动反映在 ViewModel，反之亦然。



(以上内容取自：http://www.ruanyifeng.com/blog/2015/02/mvcmvp_mvvm.html (http://www.ruanyifeng.com/blog/2015/02/mvcmvp_mvvm.html))

我们针对业务模型，建立的数据结构和相关的类，就可以理解为AndroidApp 的 Model，Model 是与 View 无关，而与业务相关的，例如数据库读取数据，应该是属于model层的事情。（感谢@Xander的讲解）
我的猜想：

> 至于为什么我们通常直接去在 Activity 中去写数据库数据读取，我的猜想是因为简单。试想，如果是为了规范，首先定义一个getDataFromDB()的接口，再写个类实现getDataFromDB()方法，以后如果改了请求数据所用的方法，直接改写实现类，听起来确实不错，可是仅仅是为了从数据库读点数据，额外添加了至少两个类文件真的有意义吗。
> 当然网络请求，是属于业务逻辑层C层。

MVP中 Presenter 真正需要处理的并非业务逻辑，而应该是视图逻辑。业务逻辑应该是视图无关的，可以是单独的一个类中，也可以是在P中。
P与V是一对多关系
EventBus应该作用于P层，在P层发送，在P层接收。

MVVM中，M层改变并不是直接改变V层，而是通过VM层去改变V层。M与V依旧是不直接操作的。

相关介绍：http://www.tianmaying.com/tutorial/AndroidMVC (http://www.tianmaying.com/tutorial/AndroidMVC)

架构的定义

有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计。
总结一下，就是一整个软件工程项目中的骨架，是一种宏观的规划。

Volley相关

Volley的磁盘缓存

在面试的时候，聊到 Volley 请求到网络的数据缓存。当时说到是 Volley 会将每次通过网络请求到的数据，采用 `FileOutputStream` ，写入到本地的文件中。
那么问题来了：这个缓存文件，是声明在一个SD卡文件夹中的(也可以是getCacheFile())。如果不停的请求网络数据，这个缓存文件夹将无限制的增大，最终达到SD卡容量时，会发生无法写入的异常(因为存储空间满了)。
这个问题的确以前没有想到，当时也没说出怎么回事。回家了赶紧又看了看代码才知道，原来 Volley 考虑过这个问题(汗!想想也是)
翻看代码 `DiskBasedCache#pruneIfNeeded()`

```
private void pruneIfNeeded(int neededSpace) {
    if ((mTotalSize + neededSpace) < mMaxCacheSizeInBytes) {
        return;
    }

    long before = mTotalSize;
    int prunedFiles = 0;
    long startTime = SystemClock.elapsedRealtime();

    Iterator<Map.Entry<String, CacheHeader>> iterator = mEntries.entr
ySet().iterator();
    while (iterator.hasNext()) {
        Map.Entry<String, CacheHeader> entry = iterator.next();
        CacheHeader e = entry.getValue();
        boolean deleted = getFileForKey(e.key).delete();
        if (deleted) {
            mTotalSize -= e.size;
        } else {
        //print log
        }
        iterator.remove();
        prunedFiles++;
        if ((mTotalSize + neededSpace) < mMaxCacheSizeInBytes * HYSTE
RESIS_FACTOR) {
            break;
        }
    }
}
```

其中 `mMaxCacheSizeInBytes` 是构造方法传入的一个缓存文件夹的大小，如果不传默认是5M的大小。
通过这个方法可以发现，每当被调用时会传入一个 `neededSpace` ，也就是需要申请的磁盘大小(即要新缓存的那个文件所需大小)。首先会判断如果这个 `neededSpace` 申请成功以后是否会超过最大可用容量，如果会超过，则通过遍历本地已经保存的缓存文件的header(header中包含了缓存文件的缓存有效期、占用大小等信息)去删除文件，直到可用容量不大于声明的缓存文件夹的大小。
其中 `HYSTERESIS_FACTOR` 是一个值为0.9的常量，应该是为了防止误差的存在吧(我猜的)。

Volley缓存命中率的优化

如果让你去设计Volley的缓存功能，你要如何增大它的命中率。
可惜了，如果上面的缓存功能是昨天看的，今天的面试这个问题就能说出来了。
还是上面的代码，在缓存内容可能超过缓存文件夹的大小时，删除的逻辑是直接遍历header删除。这个时候删除的文件有可能是我们上一次请求时刚刚保存下来的，屁股都还没坐稳呢，现在立即删掉，有点舍不得啊。
如果遍历的时候，判断一下，首先删除超过缓存有效期的(过期缓存)，其次按照LRU算法，删除最久未使用的，岂不是更合适？

Volley缓存文件名的计算

这个是我一直没弄懂的问题。
如下代码：

```
private String getFilenameForKey(String key) {
    int firstHalfLength = key.length() / 2;
    String localFilename = String.valueOf(key.substring(0, firstHalfL
ength).hashCode());
    localFilename += String.valueOf(key.substring(firstHalfLength).ha
shCode());
    return localFilename;
}
```

为什么会要把一个key分成两部分，分别求hashCode，最后又做拼接。
这个问题之前在stackoverflow上问过 #链接 (http://stackoverflow.com/questions/34984302/why-volley-diskbasedcache-splicing-without-direct-access-to-the-cache-file-name/34987423#34987423)
原谅我，别人的回答我最初并没有看懂。直到最近被问到，如果让你设计一个HashMap，如何避免value被覆盖，我才想到原因。
先来看一下 `String#hashCode()` 的实现：

```
@Override public int hashCode() {
    int hash = hashCode;
    if (hash == 0) {
        if (count == 0) {
            return 0;
        }
        final int end = count + offset;
        final char[] chars = value;
        for (int i = offset; i < end; ++i) {
            hash = 31*hash + chars[i];
        }
        hashCode = hash;
    }
    return hash;
}
```

从上面的实现可以看到，String的hashcode是根据字符数组中每个位置的字母的int值再加上上次hash值乘以31，这种算法求出来的，至于为什么是31，我也不清楚。
但是可以肯定一点，hashcode并不是唯一的。不信你运行下面这两个输出：

```
System.out.print("======" + "vFrKiaNHfF7t[9::E[XsX?L7xPp3DZSteIZvdRT8
CX:w6d;v<_KZnhsM_^dqoppe".hashCode());
System.out.print("======" + "hI4pFxGOfS@suhVUd:mTo_begImJPB@Fl[6WJ?ai
=RXfIx^=Aix@9M;;?Vdj_Zsi".hashCode());
```

这两个字符串是根据hashcode的算法逆向出来的，他们的hashcode都是12345。逆向算法请见这里 (http://my.oschina.net/backtract/blog/169310)
再回到我们的问题，为什么会要把一个key分成两部分。现在可以肯定的答出，目的是为了尽可能避免hashcode重复造成的文件名重复(求两次hash两次都与另一个url重复的概率总要比一次重复的概率小吧)。
顺带再提一点，就像上面说的，概率小并不代表不存在。但是Java计算hashcode的速度是很快的，应该是在效率和安全性上取舍的结果吧。

推送心跳包是TCP包还是UDP包或者HTTP包

其实聊起这个问题是因为最近看到 @睡不着起不来的万宵 (http://weibo.com/u/2951317192) 同学写的一篇文章《Android推送技术研究 (http://www.jianshu.com/p/584707554ed7)》结果就产生了这个没回答出来的问题(妈蛋，自己给自己挖坑 - -)
最后看了这篇文章(好像是转的，没找到原地址)android 心跳的分析

(http://blog.csdn.net/wangliang198901/article/details/16542567)
原来心跳包的实现是调用了 `socket.sendUrgentData(0xFF)` 这句代码实现的，所以，当然是TCP包。

ARGB_8888占用内存大小

首先说说本题的答案，是4byte，即ARGB各占用8个比特来描述。当时回答错了，详细解答看这里你的 Bitmap 究竟占多大内存 (http://bugly.qq.com/bbs/forum.php?mod=viewthread&tid=498)
但是——
这个问题引出了一个大大的闹剧，请听我慢慢道来。😂😂😂
不知道怎么就聊到 Bitmap 压缩上了，他说他们的Bitmap居然都是不压缩的😂😂😂
还是直接甩代码吧。。。。

```
public static Bitmap create(byte[] bytes, int maxWidth, int maxHeight
) {
                //上面的省略了
        option.inJustDecodeBounds = true;
        BitmapFactory.decodeByteArray(bytes, 0, bytes.length, option)
;
        int actualWidth = option.outWidth;
        int actualHeight = option.outHeight;

        // 计算出图片应该显示的宽高
        int desiredWidth = getResizedDimension(maxWidth, maxHeight, a
ctualWidth, actualHeight);
        int desiredHeight = getResizedDimension(maxHeight, maxWidth,
actualHeight, actualWidth);

        option.inJustDecodeBounds = false;
        option.inSampleSize = findBestSampleSize(actualWidth, actualH
eight,
                desiredWidth, desiredHeight);
        Bitmap tempBitmap = BitmapFactory.decodeByteArray(bytes, 0, b
ytes.length, option);

        // 做缩放
        if (tempBitmap != null
                && (tempBitmap.getWidth() > desiredWidth || tempBitma
p
                .getHeight() > desiredHeight)) {
            bitmap = Bitmap.createScaledBitmap(tempBitmap, desiredWid
th,
                    desiredHeight, true);
            tempBitmap.recycle();
        } else {
            bitmap = tempBitmap;
        }
    }

    return bitmap;
}
```

你这么做，decodeByteArray两次不是更占内存吗？😂😂😂
第一次设置inJustDecodeBounds = true 时候是不占内存的，因为返回的是null
一脸不相信我的说：噢，这地方我下去再看看。
吓得我回来了以后赶紧又看了看，还好没有记错，见源码注释

```
/**
 * If set to true, the decoder will return null (no bitmap), but
 * the out... fields will still be set, allowing the caller to query
 * the bitmap without having to allocate the memory for its pixels.
 */
public boolean inJustDecodeBounds;
```

Activity中类似onCreate、onStart运用了哪种设计模式，优点是什么

这个回答的太多了，我当时说的是代理模式，因为 `AppCompatActivity` 中的确是使用的代理模式。这一点还要感谢@代码家 (http://weibo.com/daimajia) 当时说让我看看 `AppCompatDelegate` 类的设计。其主要目的就是通过使用组合来替代继承，降低了耦合。

不过回家后再想一想，对方想听到的应该是模板方法模式吧。在父类中实现一个算法不变的部分，并将可变的行为留给子类来实现。生命周期方法原本就是在基类中做出了Activity不同状态时回调的一系列方法，而这些方法具体需要做的可变部分交给子类去完成。

HashMap的底层实现

HashMap内部是通过数组实现的，诶，大学时候数据结构有讲过啊，都忘记了。根据hash算法，求出当前key应该存放在数组的那个index处，如果有值了，则存在相邻的下一个位置。

根据如果自己实现HashMap如何防止value覆盖。同上面 Volley 中讲到的。

Atomic、volatile、synchronized区别

面Java基础的时候遇上了这个问题，说如果只有一个i++;的时候， `volatile` 和 `synchronized` 能否互换。当时也不知道，感觉 `volatile` 作为修饰变量的时候，变量自加会出现加到一半发生线程调度。再看看当时蒙对了。

`volatile` 可以保证在一个线程的工作内存中修改了该变量的值，该变量的值立即能回显到主内存中，从而保证所有的线程看到这个变量的值是一致的。但是有个前提，因为它不具有操作的原子性，也就是它不适合在对该变量的写操作依赖于变量本身自己。就比如i++、i+=1;这种。但是可以改为num=i+1;如果i是一个 `volatile` 类型，那么num就是安全的，总之就是不能作用于自身。

`synchronized` 是基于代码块的，只要包含在 `synchronized` 块中，就是线程安全的。既然都说了线程安全，就多了解几个：

`AtomicInteger` ，一个轻量级的 `synchronized` 。使用的并不是同步代码块，而是

Lock-Free算法(我也不懂，看代码就是一个死循环调用了底层的比较方法直到相同后才退出循环)。最终的结果就是在高并发的时候，或者说竞争激烈的时候效率比 `synchronized` 高一些。

`ThreadLocal` ，线程中私有数据。主要用于线程改变内部的数据时不影响其他线程，使用时需要注意 `static` 。

详细分析见这篇文章 (http://blog.csdn.net/u010687392/article/details/50549236) 。

再补一个，才学到的。利用 `clone()` 方法，如果是一个类的多个对象想共用对象内部的一个变量，而又不想这个变量static，可以使用浅复制方式。(查看设计模式原型模式)

其他

做内部库设计时，最重要的考虑是jar的成本，方法数、体积。

设计模式不应该是去记忆，而应该是用的时候自然而然的用上。

> 3月11日更新
> 面试真的是有够烦的，因为题目是随机的，而知识是无穷的。直到被很多答案都是没有标准的。就好像上面提到的 MV* ，也许到现在上面的理解依旧有问题，但是我觉得架构是死的，而最合适的才是最好的。
> 但是有一点，面试也是一种学习，至少它能让你知道你的薄弱点在哪。

12 条评论 3 条新浪微博 最新 最早 最热

枯枝 Chrome|46.0.2490.80 Windows 8.1

回复 梦幻halo: 人家也没说啥.....

38分钟前 回复 顶 转发

kym张涛 (http://www.kymjs.com/) 博主 超级土豪浏览器 无敌SymbianOS

回复 梦幻halo: 好的

17小时前 回复 顶 转发

(http://www.kymjs.com/)

梦幻halo (http://www.baidu.com/p/梦幻halo) Chrome|48.0.2564.116 Mac OS X

(http://www.baidu.com/p/梦幻halo)

你好，我是春林。

我要先说明下：任何一个使用第三方图片加载框架的同学都会知道，图片加载过程中，是可以压缩的。我们的图片加载，Volley，UIL，Picasso，包括Fresco，在各个不同业务线的APP中都是有用到的。

我当时想去讨论的点是：“图片不压缩会导致程序崩溃”。这个真的是一个真命题吗？更深层次的只是，包括堆栈的分配，GC内存回收，Fresco突破内存限制等相关的延伸才是我更关注的。查看代码，只是想问你volley两次执行decode，但是不消耗内存。其实你只需要告诉我第一次decode中设置options就可以了。

另外，想声明的一点是：我并没有否认APP框架上对图片加载过程中做压缩，但是我想表达的意思是，我们并不依赖这个，因为我们在获取图片时，会附上view的w和h值。压缩这个步骤放在服务端，从根源上就不需要再关注APP层面的压缩。这并不是什么很深或者很新的技术，只是从另一个方向上解决了问题而已。

祝好，顺利找到好工作

3月16日 回复 顶(1) 转发

cwr941012 Chrome|49.0.2623.75 Mac OS X

同请问博主是校招还是社招

3月16日 回复 顶(1) 转发

巨芋 (http://weibo.com/juyujuyu) Chrome|46.0.2490.86 Windows 8.1

(http://weibo.com/juyujuyu)

请问博主是社招吗？

3月14日 回复 顶(1) 转发

tedy Chrome|37.0.0.0 Android

写了那么多开源项目，依然被一些基础的东西面试住了

3月13日 回复 顶 转发

fighter Chrome|45.0.2454.101 Windows 8.1

支持

3月13日 回复 顶 转发

kym张涛 (http://www.kymjs.com/) 博主 超级土豪浏览器 无敌SymbianOS

回复 hexi_tiantong99: 谢谢，已改正。

3月12日 回复 顶(1) 转发

hexi_tiantong99 (http://weibo.com/3048522175) Chrome|48.0.2564.116 Mac OS X

ThreadLocal的目的不是线程共享数据，是为了线程不共享数据，每个线程都有一份本地的副本，从而从根本上解决现线程安全问题，也就没有同步的问题，你要是看过Realm的代码，Realm.getInstance(Context)就是通过ThreadLocal实现的，保证不同线程得到的Realm的实例都不同。

3月11日 回复 顶 转发

徐昊Xiho (http://weibo.com/xuxiho) Chrome|48.0.2564.116 Windows 10

感谢分享。。我会好好分析

3月11日 回复 顶 转发

代世文 Apple Safari|601.3.9 Mac OS X

感谢分享，最近也打算出去面试。加油~

3月11日 回复 顶 转发 举报

谢权1991 (http://weibo.com/xiequan1991) Chrome|48.0.2564.116 Mac OS X

666666666666666

3月9日 回复 顶 转发