**1 关于Http协议介绍**

http属于应用层的面向对象的协议。

其特点是简洁，快速，适用于分布式超媒体信息系统。

详细描述是：

1 支持客户端/服务器模式；

2简单快速；客户向服务器发送请求时，只需传送请求方法和路径，请求常用的方法是 get post 少数用head，每种方法规定了客户与服务器联系的类型不同，由于http协议简单，使得http服务器器程序规模小，通信速度快。

3 灵活，http允许传输任意类型数据对象。正在传输的类型由Content-Type加以标记。

4无连接；无连接的含义是限制每次连接只处理一个请求。服务器处理完客户端的请求，并收到客户端的应答后，即断开连接，采用这种方式可以节省传输的时间。

5 无状态 http协议是无状态协议，无状态协议是指协议对事物的处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重新连接，这样可能导致每次连接传输的数据量增大。另一方面，服务器不需要先前信息时它的应答速度比较快。

6 http请求由三部分组成，分别是：请求行、消息报头、请求正文

状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：  
1xx：指示信息--表示请求已接收，继续处理  
2xx：成功--表示请求已被成功接收、理解、接受  
3xx：重定向--要完成请求必须进行更进一步的操作  
4xx：客户端错误--请求有语法错误或请求无法实现  
5xx：服务器端错误--服务器未能实现合法的请求

常见状态代码、状态描述、说明：  
200 OK      //客户端请求成功  
400 Bad Request  //客户端请求有语法错误，不能被服务器所理解  
401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用   
403 Forbidden  //服务器收到请求，但是拒绝提供服务  
404 Not Found  //请求资源不存在，eg：输入了错误的URL  
500 Internal Server Error //服务器发生不可预期的错误  
503 Server Unavailable  //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

**2消息推送**

消息推送最简单的方法就是使用第三方的，比如现在使用比较多的是极光推送，机关推送的技术原理是：移动无线网络长连接

移动互联网络的现状：

因为手机平台本身、电量、网络流量的限制，移动互联网应用在设计上跟传统PC 上的应用很大不一样，需要根据手机本身的特点，尽量的节省电量和流量，同时又要尽可能的保证数据能及时到达客户端。为了解决数据同步的问题，在手机平台上，常用的方法有

2种。一种是定时去服务器上查询数据，也叫Polling，还有一种手机跟服务器之间维护一个TCP 长连接，当服务器有数据时，实时推送到客户端，也就是我们说的Push。从耗费的电量、流量和数据送达的及时性来说，Push 都会有明显的优势，但Push 的实现和维护成本相对较高。在移动无线网络下维护长连接，相对也有一些技术上的难度

移动无线网络的特点：

因为IP v4 的IP 量有限，运营商分配给手机终端的IP 是运营商内网的IP，手机要连接Internet，就需要通过运营商的网关做一个网络地址转换（Network Address Translation，NAT）。简单的说运营商的网关需要维护一个外网IP、端口到内网IP、端口的对应关系，以确保内网的手机可以跟Internet 的服务器通讯。

对于大部分移动无线网络运营商都在链路一段时间没有数据通讯时，会淘汰NAT 表中的对应项，造成链路中断。

Android 平台上长连接的实现

为了不让NAT 表失效，我们需要定时的发心跳，以刷新NAT 表项，避免被淘汰。Android 上定时运行任务常用的方法有2种，一种方法用Timer，另一种是AlarmManager。

Timer

Android 的Timer 类可以用来计划需要循环执行的任务，Timer 的问题是它需要用WakeLock 让CPU 保持唤醒状态，这样会大量消耗手机电量，大大减短手机待机时间。这种方式不能满足我们的需求。

AlarmManager

AlarmManager 是Android 系统封装的用于管理RTC 的模块，RTC (Real Time Clock) 是一个独立的硬件时钟，可以在CPU 休眠时正常运行，在预设的时间到达时，通过中断唤醒CPU。这意味着，如果我们用AlarmManager 来定时执行任务，CPU 可以正常的休眠，只有在需要运行任务时醒来一段很短的时间。极光推送的Android SDK 就是基于这种技术实现的。

服务器设计

当有大量的手机终端需要与服务器维持长连接时，对服务器的设计会是一个很大的挑战。假设一台服务器维护10万个长连接，当有1000万用户量时，需要有多达100台的服务器来维护这些用户的长连接，这里还不算用于做备份的服务器，这将会是一个巨大的成本问题。那就需要我们尽可能提高单台服务器接入用户的量，也就是业界已经讨论很久了的C10K 问题。

上面只是针对极光推送来说，下面是具体的消息推送的一般有的方式：

1. 通过SMS进行服务器端和客户端的交流通信

在Android平台上，你可以通过拦截SMS消息并且解析消息内容来了解服务器的意图，可以实现完全的实时操作。但是问题是这个方案的成本相对比较高，且依赖于运营商

1. 循环主动定时获取

这种方法需要客户端来做一个定时或者周期性的访问服务器端接口，以获得最新的消息。轮询的频率太慢可能导致某些消息的延迟，太快则会大量消耗网络带宽和电池

1. 持久连接

这个方案可以解决由轮询带来的性能问题，但是还是会消耗手机的电池。我们需要开一个服务来保持和服务器端的持久连接（苹果就和谷歌的C2DM是这种机制）。但是对于Android系统，当系统可用资源较低，系统会强制关闭我们的服务或者是应用，这种情况下连接会强制中断。（Apple的推送服务之所以工作的很好，是因为每一台手机仅仅保持一个与服务器之间的连接，事实上C2DM也是这么工作的。即所有的推送服务都是经由一个代理服务器完成的，这种情况下只需要和一台服务器保持持久连接即可。C2DM=*Cloud to Device Messaging*）。

　相比之下第三种还是最可行的。为软件编写系统服务或开机启动功能；或者如果系统资源较低，服务被关闭后可以在onDestroy ()方法里面再重启该服务，进而实现持久连接的方式。

　C2DM内置于Android的2.2系统上，无法兼容老的1.6到2.1系统；且依赖于Google官方提供的C2DM服务器，由于国内的网络环境，这个服务经常不可用。

　建立在TCP协议之上的XMPP协议，不仅可提供可这种持久连接的功能，能实现服务器和客户机的双工通信，还能不依赖与系统版本和google服务器的限制，提供了比较好的解决方案。

当别人要求你讲解详细的xmpp协议的时候才需要讲解详细

下面讲到的是**XMPP协议的一些介绍：**

XMPP全称Extensible Messaging and Presence Protocol，前身是Jabber项目，是一种以XML为基础的开放式即时通讯协议。XMPP因为被Google Talk和网易泡泡应用而被广大网民所接触。XMPP的关键特色是，分散式的即时通讯系统，以及使用XML串流。XMPP目前被IETF国际标准组织完成了标准化工作。

**Android push notification**(androidpn) 是一个基于XMPP协议的java开源实现，它包含了完整的客户端和服务器端。该服务器端基本是在另外一个开源工程openfire基础上修改实现的。

　　androidpn客户端需要用到一个基于java的开源XMPP协议包asmack，这个包同样也是基于openfire下的另外一个开源项目smack，不过我们不需要自己编译，可以直接把androidpn客户端里面的asmack.jar拿来使用。客户端利用asmack中提供的XMPPConnection类与服务器建立持久连接，并通过该连接进行用户注册和登录认证，同样也是通过这条连接，接收服务器发送的通知。

　　androidpn服务器端也是java语言实现的，基于openfire开源工程，不过它的Web部分采用的是spring框架，这一点与openfire是不同的。Androidpn服务器包含两个部分，一个是侦听在5222端口上的XMPP服务，负责与客户端的XMPPConnection类进行通信，作用是用户注册和身份认证，并发送推送通知消息。另外一部分是Web服务器，采用一个轻量级的HTTP服务器，负责接收用户的Web请求。服务器的这两方式，意义非凡：当相应的TCP端口被防火墙封闭，可以使用轮询的方式进行访问，因此又有助于通过防火墙。

下面是xmpp详解

深入学习XMPP议

1. XMPP（协议简介）XMPP协议（Extensible Messaging and PresenceProtocol消息处理现场协议）是一种基于XML的协议，目的是为了解决及时通信标准而提出来的，最早是在Jabber上实现的。它继承了在XM环境中灵活的发展性。因此，基于XMPP的应用具有超强的可扩展性。并且XML很易穿过防火墙，所以用XMPP构建的应用不易受到防火墙的阻碍。利用XMPP作为通用的传输机制，不同组织内的不同应用都可以进行有效的通信。二．IM（即时通讯软件简介）Instant Messenger，及时通信软件，就是大家使用的QQ、MSN Messenger和Gtalk等等。其中Gtalk 就是基于XMPP 协议的一个实现，其他的则不是。当前IM 几乎作为每个上网者必然使用的工具，在国外的大型企业中有一些企业级的IM应用，但是其商业价值还没完全发挥出来。设想既然XMPP 协议是一个公开的协议，那么每个企业都可以利用它来开发适合本身企业工作，提高自身生产效率的IM；甚至，你还可以在网络游戏中集成这种通信软件，不但让你可以边游戏边聊天，也可以开发出适合游戏本身的IM 应用，比如说一些游戏关键场景提醒功能，团队语音交流等等都可以基于IM来实现。三．本文主要内容本文主要讲解在android使用xmpp协议进行即时通信，所涉及3个主要的东西，它们是openfire、smack和spark，这个三个东东结合起来就是完整的xmpp IM实现，这里简单介绍一下这3个东东在下文的作用：openfire主要是作为服务器，负责管理客户端的通信连接，以及提供客户端一些通信信息和连接信息。Smack主要是xmpp协议的实现，提供了一套很好的api，所以下面操作xmpp都是通过使用smack的api来实现，当然因为是在android里，所以使用的是asmack这个包，里面方法跟smack包差不多。Spark 是IM客户端的实现，其实就是使用了smack 的api实现的。数据通讯具体实现的流程：四．开发具体配置环境：1.配置openfire服务器Openfire是一个强大的即时消息（IM）和聊天服务器，它实现了XMPP协议,可以使用它轻易的构建高效率的即时通信服务器. 其安装和部署都十分简单，并利用Web进行管理。单台服务器可支持上万并发用户，由于是采用开放的XMPP协议，可以使用各种支持XMPP协议的IM客户端软件登陆服务。安装前准备工作：一：Java运行环境（已安装则可跳过此步骤）官方下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index.html 二：数据库（之前已安装则可跳过此步骤）数据库可以使用DB2,Oracle,MySQL,PostgreSQL,SQL Server等其中任意一种安装前准备工作完成后：一：建立数据库a.建立数据库，记下数据库名，如OpenfireServer。b.在数据库管理系统下建立新用户，如OpenfireUser。选择SQL server身份验证，把强制密码过期的勾去掉，只勾上强制实施密码策略。在下面的数据库下拉列表中选择OpenfireServer，然后确定即可。c.然后在OpenfireServer数据库目录下，在其安全性中，新建一个用户，用户名任取。登录名选择上面创建的OpenfireUser，架构选择db\_owner,数据库角色成员身份选择db\_owner, 二：安装Openfire a.到官方下载最新版的Openfire安装包官方网址为：http://www.igniterealtime.org/projects/openfire/ b.双击Openfire安装包开始安装安装完毕后，会有提示框提示是否登录，此时先放着不动三：执行数据库脚本a.到Openfire安装目录下：如D:\Program Files\Openfire\resources\database 下面有几个数据库的SQL脚本，直接拖动到SQL Server Studio中执行，执行前确保可用数据库下拉列表选中的是OpenfireServer。b.接着连接对象资源管理器，使用SQL Server身份验证方式，输入用户名OpenfireUser，密码尝试进行登录操作。若成功登陆，则证明数据库建立成功。四：首次设置Openfire a.点击Openfire提示框中Launch Admin 按钮，进入首次设置页面b.语言选择简体中文数据库选择标准数据库连接选择相应的数据库驱动类型；URL一栏中，把[hostname]改成当前的主机名；把[database]改成数据库名OpenfireServer；接着分号后面的Appname全部去掉；点击continue ，设置密码——>>结束五：重启Openfire 然后登录管理页面，进行Openfire管理。至此，恭喜您，Openfire部署安装完成。配置成功如果以后ip地址变了，那肯定又是开不了，解决办法请移步：http://blog.csdn.net/HappySheepherder/article/details/4707124 配置成功后，在服务器创建一个简单的用户来测试，然后安装spark，设置好服务器的ip与端口，使用刚才创建的用户登录，登录OK说明服务器成功搭建。2.客户端配置要求（1）android 2.2平台及以上。（2）asmack-jse.jar开发必须包。（3）客户端Eclispse，服务端Myeclipse IDE开发环境。五．具体实例展示1.先建一个Android项目。2.导入项目必须XMPP协议jar包。4通过XMPP协议规则，进行数据通讯。最后我们就可以利用connection进行连接，登录，注册最后如有错误请包容，学习资料如有不够深入，请选择其它专业资料。

**3关于大图片的问题：**

张泽华老师讲过

**4关于内存优化：**

　不少人认为JAVA程序，因为有垃圾回收机制，应该没有内存泄露。其实如果我们一个程序中，已经不再使用某个对象，但是因为仍然有引用指向它，垃圾回收器就无法回收它，当然该对象占用的内存就无法被使用，这就造成了内存泄露。如果我们的java运行很久,而这种内存泄露不断的发生，最后就没内存可用了。当然java的，内存泄漏和C/C++是不一样的。如果java程序完全结束后，它所有的对象就都不可达了，系统就可以对他们进行垃圾回收，它的内存泄露仅仅限于它本身，而不会影响整个系统的。C/C++的内存泄露就比较糟糕了，它的内存泄露是系统级，即使该C/C++程序退出，它的泄露的内存也无法被系统回收，永远不可用了，除非重启机器。  
　　Android的一个应用程序的内存泄露对别的应用程序影响不大。为了能够使得Android应用程序安全且快速的运行，Android的每个应用程序都会使用一个专有的Dalvik虚拟机实例来运行，它是由Zygote服务进程孵化出来的，也就是说每个应用程序都是在属于自己的进程中运行的。Android为不同类型的进程分配了不同的内存使用上限，如果程序在运行过程中出现了内存泄漏的而造成应用进程使用的内存超过了这个上限，则会被系统视为内存泄漏，从而被kill掉，这使得仅仅自己的进程被kill掉，而不会影响其他进程（如果是system\_process等系统进程出问题的话，则会引起系统重启）。  
一、引用没释放造成的内存泄露  
1.1注册没取消造成的内存泄露  
　　这种Android的内存泄露比纯java的内存泄露还要严重，因为其他一些Android程序可能引用我们的Anroid程序的对象（比如注册机制）。即使我们的Android程序已经结束了，但是别的引用程序仍然还有对我们的Android程序的某个对象的引用，泄露的内存依然不能被垃圾回收。  
比如示例1:  
　　假设我们希望在锁屏界面(LockScreen)中，监听系统中的电话服务以获取一些信息(如信号强度等)，则可以在LockScreen中定义一个PhoneStateListener的对象，同时将它注册到TelephonyManager服务中。对于LockScreen对象，当需要显示锁屏界面的时候就会创建一个LockScreen对象，而当锁屏界面消失的时候LockScreen对象就会被释放掉。  
　　但是如果在释放LockScreen对象的时候忘记取消我们之前注册的PhoneStateListener对象，则会导致LockScreen无法被垃圾回收。如果不断的使锁屏界面显示和消失，则最终会由于大量的LockScreen对象没有办法被回收而引起OutOfMemory,使得system\_process进程挂掉。  
　　虽然有些系统程序，它本身好像是可以自动取消注册的（当然不及时），但是我们还是应该在我们的程序中明确的取消注册，程序结束时应该把所有的注册都取消掉。  
1.2集合中对象没清理造成的内存泄露  
　　我们通常把一些对象的引用加入到了集合中，当我们不需要该对象时，并没有把它的引用从集合中清理掉，这样这个集合就会越来越大。如果这个集合是static的话，那情况就更严重了。  
二、资源对象没关闭造成的内存泄露  
　　资源性对象比如（Cursor，File文件等）往往都用了一些缓冲，我们在不使用的时候，应该及时关闭它们，以便它们的缓冲及时回收内存。它们的缓冲不仅存在于java虚拟机内，还存在于java虚拟机外。如果我们仅仅是把它的引用设置为null,而不关闭它们，往往会造成内存泄露。因为有些资源性对象，比如SQLiteCursor（在析构函数finalize（）,如果我们没有关闭它，它自己会调close()关闭），如果我们没有关闭它，系统在回收它时也会关闭它，但是这样的效率太低了。因此对于资源性对象在不使用的时候，应该调用它的close()函数，将其关闭掉，然后才置为null.在我们的程序退出时一定要确保我们的资源性对象已经关闭。  
　　程序中经常会进行查询数据库的操作，但是经常会有使用完毕Cursor后没有关闭的情况。如果我们的查询结果集比较小，对内存的消耗不容易被发现，只有在常时间大量操作的情况下才会复现内存问题，这样就会给以后的测试和问题排查带来困难和风险。  
三、一些不良代码成内存压力  
　　有些代码并不造成内存泄露，但是它们，或是对没使用的内存没进行有效及时的释放，或是没有有效的利用已有的对象而是频繁的申请新内存，对内存的回收和分配造成很大影响的，容易迫使虚拟机不得不给该应用进程分配更多的内存，造成不必要的内存开支。  
3.1，Bitmap没调用recycle()  
　　Bitmap对象在不使用时,我们应该先调用recycle()释放内存，然后才它设置为null.虽然recycle()从源码上看，调用它应该能立即释放Bitmap的主要内存，但是测试结果显示它并没能立即释放内存。但是我它应该还是能大大的加速Bitmap的主要内存的释放。  
3.2，构造Adapter时，没有使用缓存的 convertView  
　　以构造ListView的BaseAdapter为例，在BaseAdapter中提共了方法：  
public View getView(int position, View convertView, ViewGroup parent)来向ListView提供每一个item所需要的view对象。初始时ListView会从BaseAdapter中根据当前的屏幕布局实例化一定数量的view对象，同时ListView会将这些view对象缓存起来。当向上滚动ListView时，原先位于最上面的list item的view对象会被回收，然后被用来构造新出现的最下面的list item。这个构造过程就是由getView()方法完成的，getView()的第二个形参 View convertView就是被缓存起来的list item的view对象(初始化时缓存中没有view对象则convertView是null)。  
　　由此可以看出，如果我们不去使用convertView，而是每次都在getView()中重新实例化一个View对象的话，即浪费时间，也造成内存垃圾，给垃圾回收增加压力，如果垃圾回收来不及的话，虚拟机将不得不给该应用进程分配更多的内存，造成不必要的内存开支。ListView回收list item的view对象的过程可以查看:  
view plaincopy to clipboardprint?  
android.widget.AbsListView.java --> void addScrapView(View scrap) 方法。  
  
示例代码：

[复制代码](javascript:void(0);)

1 public View getView(int position, View convertView, ViewGroup parent) {

2

3 View view = new Xxx(...);

4

5 ... ...

6

7 return view;

8

9 }

[复制代码](javascript:void(0);)

修正示例代码：

Android内存管理

[复制代码](javascript:void(0);)

1 public View getView(int position, View convertView, ViewGroup parent) {

2

3 View view = null;

4

5 if (convertView != null) {

6

7 view = convertView;

8

9 populate(view, getItem(position));

10

11 ...

12

13 } else {

14

15 view = new Xxx(...);

16

17 ...

18

19 }

20

21 return view;

22

23 }

[复制代码](javascript:void(0);)

概述：  
　　在android的开发中，要时刻主要内存的分配和垃圾回收，因为系统为每一个dalvik虚拟机分配的内存是有限的，在google的G1中，分配的最大堆大小只有16M，后来的机器一般都为24M，实在是少的可怜。这样就需要我们在开发过程中要时刻注意。不要因为自己的代码问题而造成OOM错误。  
JAVA的内存管理：  
　　大家都知道，android应用层是由java开发的，android的davlik虚拟机与jvm也类似，只不过它是基于寄存器的。因此要了解android的内存管理就必须得了解java的内存分配和垃圾回收机制。  
　　在java中，是通过new关键字来为对象分配内存的，而内存的释放是由垃圾收集器（GC）来回收的，工程师在开发的过程中，不需要显式的去管理内存。但是这样有可能在不知不觉中就会浪费了很多内存，最终导致java虚拟机花费很多时间去进行垃圾回收，更严重的是造成JVM的OOM。因此，java工程师还是有必要了解JAVA的内存分配和垃圾回收机制。

|  |
| --- |
| 内存结构  file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-6926.png  上面这张图是JVM的结构图，它主要四个部分组成：Class Loader子系统和执行引擎，运行时方法区和本地方法区，我们主要来看下RUNTIME DATA AREA区，也就是我们常说的JVM内存。从图中可以看出，RUNTIMEDATA AREA区主要由5个部分组成：  · Method Area:被装载的class的元信息存储在Method Area中，它是线程共享的  · Heap(堆)：一个java虚拟机实例中只存在一个堆空间，存放一些对象信息，它是线程共享的  · Java栈： java虚拟机直接对java栈进行两种操作，以帧为单位的压栈和出栈（非线程共享）  · 程序计数器（非线程共享）  · 本地方法栈（非线程共享）    JVM的垃圾回收（GC）  file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-12485.png  JVM的垃圾原理是这样的，它把对象分为年轻代（Young）、年老代（Tenured）、持久代（Perm），对不同生命周期的对象使用不同的垃圾回收算法。  · 年轻代(Young)  年轻代分为三个区，一个eden区，两个Survivor区。程序中生成的大部分新的对象都在Eden区中，当Eden区满时，还存活的对象将被复制到其中一个Survivor区，当此Survivor区的对象占用空间满了时，此区存活的对象又被复制到另外一个Survivor区，当这个Survivor区也满了的时候，从第一个Survivor区复制过来的并且此时还存活的对象，将被复制到年老代。  · 年老代（Tenured）  年老代存放的是上面年轻代复制过来的对象，也就是在年轻代中还存活的对象，并且区满了复制过来的。一般来说，年老代中的对象生命周期都比较长。  · 持久代（Perm）  用于存放静态的类和方法，持久代对垃圾回收没有显著的影响。  Android中内存泄露监测  在了解了JVM的内存管理后，我们再回过头来看看，在android中应该怎样来监测内存，从而看在应用中是否存在内存分配和垃圾回收问题而造成内存泄露情况。  在android中，有一个相对来说还不错的工具，可以用来监测内存是否存在泄露情况：DDMS—Heap  file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-22715.png  使用方法比较简单：  · 选择DDMS视图，并打开Devices视图和Heap视图  · 点击选择要监控的进程，比如：上图中我选择的是system\_process  · 选中Devices视图界面上的"update heap" 图标  · 点击Heap视图中的"Cause GC" 按钮（相当于向虚拟机发送了一次GC请求的操作）  在Heap视图中选择想要监控的Type，一般我们会观察dataobject的 total size的变化，正常情况下total size的值会稳定在一个有限的范围内，也就说程序中的代码良好，没有造成程序中的对象不被回收的情况。如果代码中存在没有释放对象引用的情况，那么data object的total size在每次GC之后都不会有明显的回落，随着操作次数的增加而total size也在不断的增加。（说明：选择好data object后，不断的操作应用，这样才可以看出total size的变化）。如果totalsize确实是在不断增加而没有回落，说明程序中有没有被释放的资源引用。那么我们应该怎么来定位呢？  Android中内存泄露定位  Mat(memory analyzer tools)是我们常用的用来定位内存泄露的工具，如果你使用ADT，并且安装了MAT的eclipse插件，你需要做的是进入DDMS视图的Devices视图：  file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-2165.png  点击"dump HPROF file"按钮，然后使用MAT分析下载下来的文件。  file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-6565.png |

 下面列出了存在的问题，点击detail进去，会列出详细的，可能会存在问题的代码：  
file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-32625.png  
file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-21158.png  
关于MAT的使用可以参考：[http://www.blogjava.net/rosen/ ... .html](http://www.blogjava.net/rosen/archive/2010/06/13/323522.html)  
这位兄弟写的比较详细。

总结

不管是java还是android，都应该了解内存分配和垃圾回收机制，工程师要做到写的代码中没有bad code很难，关键是在出现问题的时候该怎么去排查Android内存优化  
一、 Android的内存机制  
　　Android的程序由Java语言编写，所以Android的内存管理与Java的内存管理相似。程序员通过new为对象分配内存，所有对象在java堆内分配空间；然而对象的释放是由垃圾回收器来完成的。C／C++中的内存机制是“谁污染，谁治理”，java的就比较人性化了，给我们请了一个专门的清洁工（GC）。  
　　那么GC怎么能够确认某一个对象是不是已经被废弃了呢？Java采用了有向图的原理。Java将引用关系考虑为图的有向边，有向边从引用者指向引用对象。线程对象可以作为有向图的起始顶点，该图就是从起始顶点开始的一棵树，根顶点可以到达的对象都是有效对象，GC不会回收这些对象。如果某个对象 (连通子图)与这个根顶点不可达(注意，该图为有向图)，那么我们认为这个(这些)对象不再被引用，可以被GC回收。  
二、Android的内存溢出  
Android的内存溢出是如何发生的?  
　　Android的虚拟机是基于寄存器的Dalvik，它的最大堆大小一般是16M，有的机器为24M。因此我们所能利用的内存空间是有限的。如果我们的内存占用超过了一定的水平就会出现OutOfMemory的错误。  
为什么会出现内存不够用的情况呢？我想原因主要有两个：

由于我们程序的失误，长期保持某些资源（如Context）的引用，造成内存泄露，资源造成得不到释放。

保存了多个耗用内存过大的对象（如Bitmap），造成内存超出限制。

三、万恶的static  
　　static是Java中的一个关键字，当用它来修饰成员变量时，那么该变量就属于该类，而不是该类的实例。所以用static修饰的变量，它的生命周期是很长的，如果用它来引用一些资源耗费过多的实例（Context的情况最多），这时就要谨慎对待了。

[复制代码](javascript:void(0);)

1 public class ClassName {

2

3 private static Context mContext;

4

5 //省略

6

7 }

[复制代码](javascript:void(0);)

　　以上的代码是很危险的，如果将Activity赋值到么mContext的话。那么即使该Activity已经onDestroy，但是由于仍有对象保存它的引用，因此该Activity依然不会被释放。  
我们举Android文档中的一个例子。

[复制代码](javascript:void(0);)

private static Drawable sBackground;

@Override

protected void onCreate(Bundle state) {

super.onCreate(state);

TextView label = new TextView(this);

label.setText("Leaks are bad");

if (sBackground == null) {

sBackground = getDrawable(R.drawable.large\_bitmap);

}

label.setBackgroundDrawable(sBackground);

setContentView(label);

}

[复制代码](javascript:void(0);)

　　sBackground是一个静态的变量，但是我们发现，我们并没有显式的保存Contex的引用，但是，当Drawable与View连接之后，Drawable就将View设置为一个回调，由于View中是包含Context的引用的，所以，实际上我们依然保存了Context的引用。这个引用链如下：  
Drawable->TextView->Context  
　　所以，最终该Context也没有得到释放，发生了内存泄露。  
如何才能有效的避免这种引用的发生呢？

应该尽量避免static成员变量引用资源耗费过多的实例，比如Context。

Context尽量使用Application Context，因为Application的Context的生命周期比较长，引用它不会出现内存泄露的问题。

使用WeakReference代替强引用。比如可以使用WeakReference<Context> mContextRef;

该部分的详细内容也可以参考Android文档中Article部分。  
四、都是线程惹的祸  
　　线程也是造成内存泄露的一个重要的源头。线程产生内存泄露的主要原因在于线程生命周期的不可控。我们来考虑下面一段代码。

[复制代码](javascript:void(0);)

1 public class MyActivity extends Activity {

2

3 @Override

4

5 public void onCreate(Bundle savedInstanceState) {

6

7 super.onCreate(savedInstanceState);

8

9 setContentView(R.layout.main);

10

11 new MyThread().start();

12

13 }

14

15

16 private class MyThread extends Thread{

17

18 @Override

19

20 public void run() {

21

22 super.run();

23

24 //do somthing

25

26 }

27

28 }

29

30 }

[复制代码](javascript:void(0);)

　　这段代码很平常也很简单，是我们经常使用的形式。我们思考一个问题：假设MyThread的run函数是一个很费时的操作，当我们开启该线程后，将设备的横屏变为了竖屏，一般情况下当屏幕转换时会重新创建Activity，按照我们的想法，老的Activity应该会被销毁才对，然而事实上并非如此。  
　　由于我们的线程是Activity的内部类，所以MyThread中保存了Activity的一个引用，当MyThread的run函数没有结束时，MyThread是不会被销毁的，因此它所引用的老的Activity也不会被销毁，因此就出现了内存泄露的问题。  
file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ksohtml/wps\_clip\_image-6439.png  
　　有些人喜欢用Android提供的AsyncTask，但事实上AsyncTask的问题更加严重，Thread只有在run函数不结束时才出现这种内存泄露问题，然而AsyncTask内部的实现机制是运用了ThreadPoolExcutor,该类产生的Thread对象的生命周期是不确定的，是应用程序无法控制的，因此如果AsyncTask作为Activity的内部类，就更容易出现内存泄露的问题。  
这种线程导致的内存泄露问题应该如何解决呢？

将线程的内部类，改为静态内部类。

在线程内部采用弱引用保存Context引用。

解决的模型如下：

[复制代码](javascript:void(0);)

1 public abstract class WeakAsyncTask<Params, Progress, Result, WeakTarget> extends

2 AsyncTask<Params, Progress, Result> {

3 protected WeakReference<WeakTarget> mTarget;

4

5 public WeakAsyncTask(WeakTarget target) {

6 mTarget = new WeakReference<WeakTarget>(target);

7 }

8

9 /\*\* {@inheritDoc} \*/

10 @Override

11 protected final void onPreExecute() {

12 final WeakTarget target = mTarget.get();

13 if (target != null) {

14 this.onPreExecute(target);

15 }

16 }

17

18 /\*\* {@inheritDoc} \*/

19 @Override

20 protected final Result doInBackground(Params... params) {

21 final WeakTarget target = mTarget.get();

22 if (target != null) {

23 return this.doInBackground(target, params);

24 } else {

25 return null;

26 }

27 }

28

29 /\*\* {@inheritDoc} \*/

30 @Override

31 protected final void onPostExecute(Result result) {

32 final WeakTarget target = mTarget.get();

33 if (target != null) {

34 this.onPostExecute(target, result);

35 }

36 }

37

38 protected void onPreExecute(WeakTarget target) {

39 // No default action

40 }

41

42 protected abstract Result doInBackground(WeakTarget target, Params... params);

43

44 protected void onPostExecute(WeakTarget target, Result result) {

45 // No default action

46 }

47 }

[复制代码](javascript:void(0);)

事实上，线程的问题并不仅仅在于内存泄露，还会带来一些灾难性的问题。由于本文讨论的是内存问题，所以在此不做讨论。

由于51cto不让我一次传完，说我的字数太多了，所以分开传了。  
五、超级大胖子Bitmap  
　　可以说出现OutOfMemory问题的绝大多数人，都是因为Bitmap的问题。因为Bitmap占用的内存实在是太多了，它是一个“超级大胖子”，特别是分辨率大的图片，如果要显示多张那问题就更显著了。  
如何解决Bitmap带给我们的内存问题？

及时的销毁。

　　虽然，系统能够确认Bitmap分配的内存最终会被销毁，但是由于它占用的内存过多，所以很可能会超过java堆的限制。因此，在用完Bitmap时，要及时的recycle掉。recycle并不能确定立即就会将Bitmap释放掉，但是会给虚拟机一个暗示：“该图片可以释放了”。

设置一定的采样率。

　　有时候，我们要显示的区域很小，没有必要将整个图片都加载出来，而只需要记载一个缩小过的图片，这时候可以设置一定的采样率，那么就可以大大减小占用的内存。如下面的代码：

[复制代码](javascript:void(0);)

1 private ImageView preview;

2

3

4 BitmapFactory.Options options = new BitmapFactory.Options();

5

6

7 options.inSampleSize = 2;//图片宽高都为原来的二分之一，即图片为原来的四分之一

8

9 Bitmap bitmap = BitmapFactory.decodeStream(cr.openInputStream(uri), null, options);

10

11

12 preview.setImageBitmap(bitmap);

[复制代码](javascript:void(0);)

巧妙的运用软引用（SoftRefrence）

　　有些时候，我们使用Bitmap后没有保留对它的引用，因此就无法调用Recycle函数。这时候巧妙的运用软引用，可以使Bitmap在内存快不足时得到有效的释放。如下例：  
/\*\*本例子为博主随手一写，来说明用法，并未验证\*/

[复制代码](javascript:void(0);)

1 private class MyAdapter extends BaseAdapter {

2

3 private ArrayList<SoftReference<Bitmap>> mBitmapRefs = new ArrayList<SoftReference<Bitmap>>();

4 private ArrayList<Value> mValues;

5 private Context mContext;

6 private LayoutInflater mInflater;

7

8 MyAdapter(Context context, ArrayList<Value> values) {

9 mContext = context;

10 mValues = values;

11 mInflater = (LayoutInflater) context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

12 }

13 public int getCount() {

14 return mValues.size();

15 }

16 public Object getItem(int i) {

17 return mValues.get(i);

18 }

19

20 public long getItemId(int i) {

21 return i;

22 }

23

24 public View getView(int i, View view, ViewGroup viewGroup) {

25 View newView = null;

26 if(view != null) {

27 newView = view;

28 } else {

29 newView =(View)mInflater.inflate(R.layout.image\_view, false);

30 }

31

32 Bitmap bitmap = BitmapFactory.decodeFile(mValues.get(i).fileName);

33 mBitmapRefs.add(new SoftReference<Bitmap>(bitmap)); //此处加入ArrayList

34 ((ImageView)newView).setImageBitmap(bitmap);

35

36 return newView;

37 }

38 }

[复制代码](javascript:void(0);)

 六、行踪诡异的Cursor  
　　Cursor是Android查询数据后得到的一个管理数据集合的类，正常情况下，如果查询得到的数据量较小时不会有内存问题，而且虚拟机能够保证Cusor最终会被释放掉。  
　　然而如果Cursor的数据量特表大，特别是如果里面有Blob信息时，应该保证Cursor占用的内存被及时的释放掉，而不是等待GC来处理。并且Android明显是倾向于编程者手动的将Cursor close掉，因为在源代码中我们发现，如果等到垃圾回收器来回收时，会给用户以错误提示。  
所以我们使用Cursor的方式一般如下：

[复制代码](javascript:void(0);)

1 Cursor cursor = null;

2 try {

3 cursor = mContext.getContentResolver().query(uri,null, null,null,null);

4 if(cursor != null) {

5 cursor.moveToFirst();

6 //do something

7 }

8 } catch (Exception e) {

9 e.printStackTrace();

10 } finally {

11 if (cursor != null) {

12 cursor.close();

13 }

14 }

[复制代码](javascript:void(0);)

　　有一种情况下，我们不能直接将Cursor关闭掉，这就是在CursorAdapter中应用的情况，但是注意，CursorAdapter在Acivity结束时并没有自动的将Cursor关闭掉，因此，你需要在onDestroy函数中，手动关闭。

[复制代码](javascript:void(0);)

1 protected void onDestroy() {

2

3 if (mAdapter != null && mAdapter.getCurosr() != null) {

4

5 mAdapter.getCursor().close();

6

7 }

8

9 super.onDestroy();

10

11 }

[复制代码](javascript:void(0);)

　　CursorAdapter中的changeCursor函数，会将原来的Cursor释放掉，并替换为新的Cursor，所以你不用担心原来的Cursor没有被关闭。  
　　你可能会想到使用Activity的managedQuery来生成Cursor，这样Cursor就会与Acitivity的生命周期一致了，多么完美的解决方法！然而事实上managedQuery也有很大的局限性。  
　　managedQuery生成的Cursor必须确保不会被替换，因为可能很多程序事实上查询条件都是不确定的，因此我们经常会用新查询的Cursor来替换掉原先的Cursor。因此这种方法适用范围也是很小。  
七、其它要说的。  
　　其实，要减小内存的使用，其实还有很多方法和要求。比如不要使用整张整张的图，尽量使用9path图片。Adapter要使用convertView等等，好多细节都可以节省内存。这些都需要我们去挖掘，谁叫Android的内存不给力来着。

**5 dp、sp和px的区别：**

现在这里介绍一下，dp也就是dip。这个和sp基本类似。如果设置表示长度、高度等属性时可以使用dp或sp。但如果设置字体，需要使用sp。dp是与密度无关，sp除了与密度无关外，还与scale无关。如果屏幕密度为160，这时dp和sp和px是一样的。1dp=1sp=1px，但如果使用px作单位，如果屏幕大小不变（假设还是3.2寸），而屏幕密度变成了320。那么原来TextView的宽度设成160px，在密度为320的3.2寸屏幕里看要比在密度为160的3.2寸屏幕上看短了一半。但如果设置成160dp或160sp的话。系统会自动将width属性值设置成320px的。也就是160 \* 320 / 160。其中320 / 160可称为密度比例因子。也就是说，如果使用dp和sp，系统会根据屏幕密度的变化自动进行转换.

## 6Android Touch事件原理加实例分析