

# app性能测试

app性能测试

安大叔

2013-03-26

# IOS App性能测试

## 一些概念

**App**性能测试指的是什么？通俗指的是 **app**本身

**APP**由什么组成？

**APP**测试要测试什么？

**APP**性能测试又要测试什么？

# IOS App性能测试

## App接口测试

任意工具都行，只要能测通你的接口即可  
常见的就是jmeter，lr都行，上课演示过，略过

## HTTP协议

# IOS App主要内容

现有工具

**Shark**

**Instruments**

**Clang静态分析器**

**UI Recorder**

# IOS App主要内容

## Shark

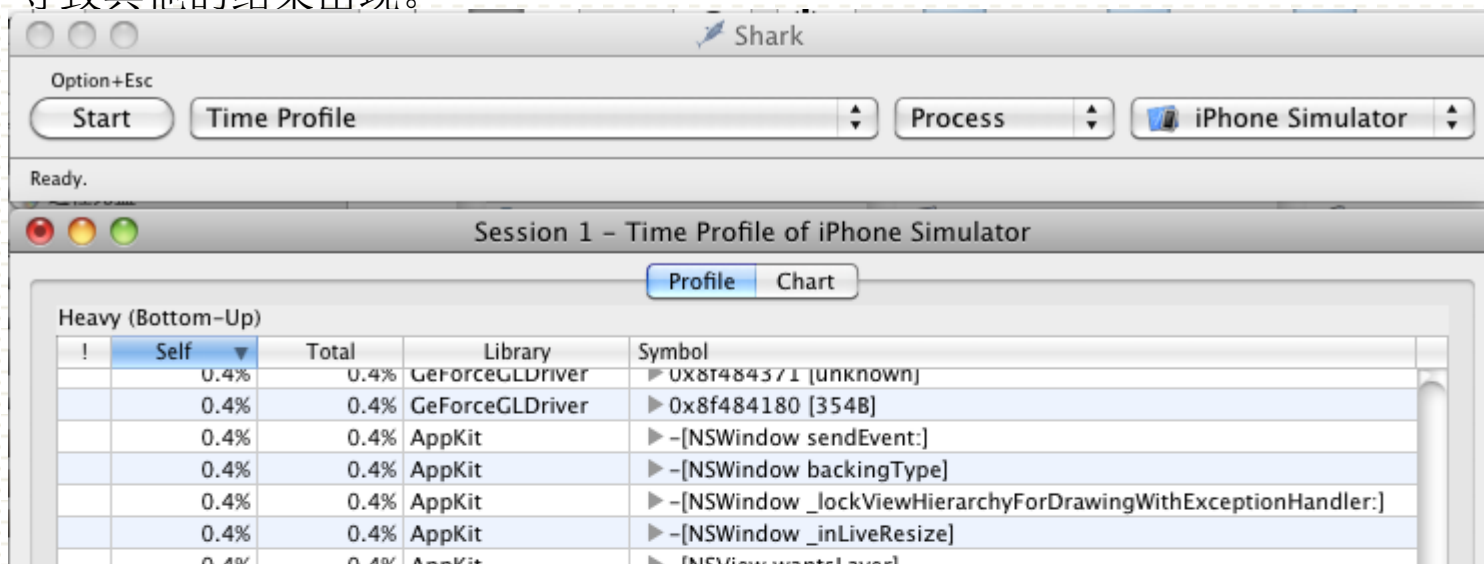
xcode包含许多性能分析工具，其中有一个叫shark，位于  
/Developer/Applications/Performance Tools下，他最早是用于处理运行mac的程序，现在对于运行在iphone上的程序也有很大帮助。

- 1. 在模拟器中构建运行程序，
- 2. 切换到shark设置，时间分析或其他方式，选择进程或者其他，选择指定的进程，
- 3. start，测试过程的操作，
- 4. stop，查看结果

# IOS App主要内容

## Shark

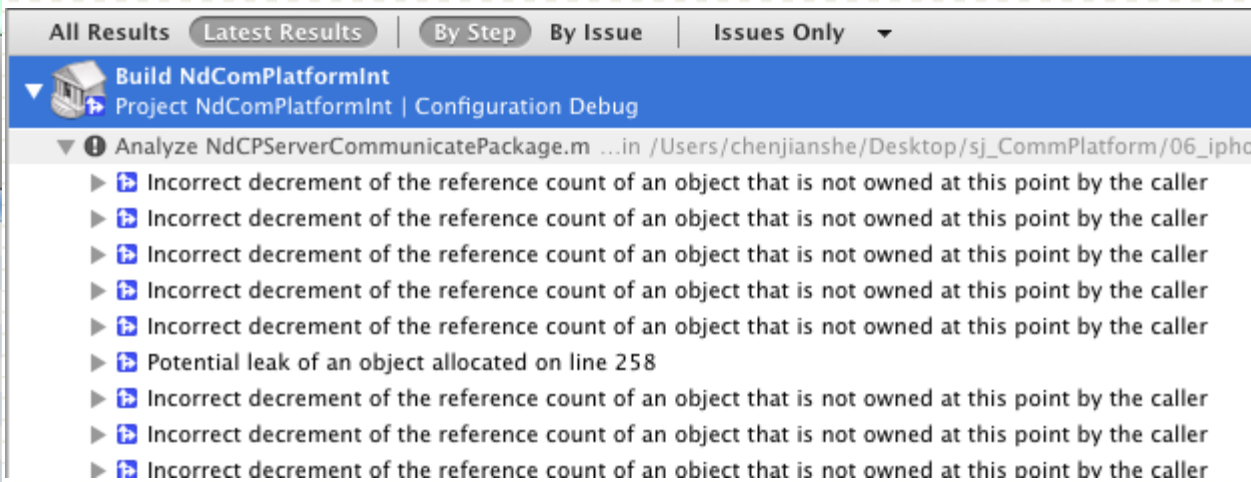
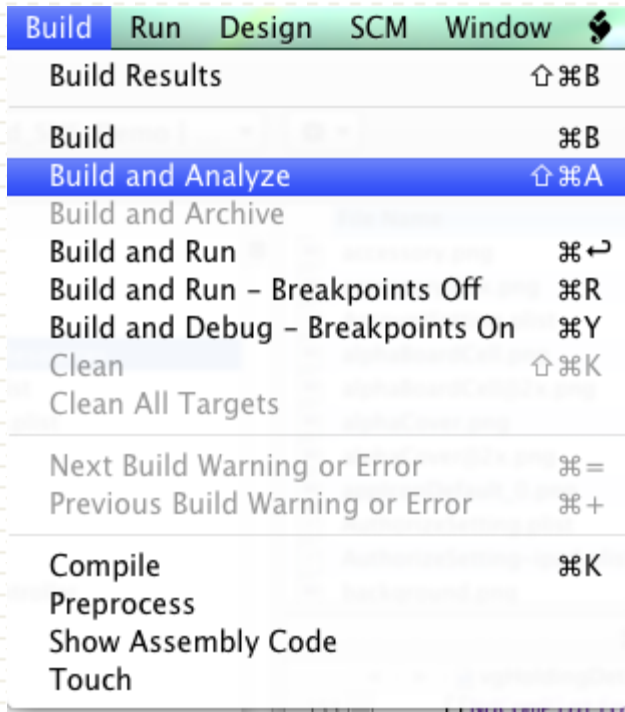
- 过分析，可能是消耗过多的GUI布局，或者是无谓的计算等，最终发现热点用更有效的方法来消除热点
- 在配置好代码签名的环境，也可以测试和mac相连的ios设备，选择其中相关远程调试选项即可开始，手头没有设备的环境，没有测试。
- shark有效但是还是有一定的局限性：它把在他的取样周期中的所有活动同等对待，无法捕捉一小段时间，而这段时间里也许有一些其他的外部事件，做了有趣的事情，会导致其他的结果出现。



# IOS App主要内容

## Clang静态分析器

- 这是一个LLVM的开源项目，更具体的信息可以通过其网站来了解。  
静态分析器不是在程序运行时检查代码，它是一个代码静态分析工具，通过自身的技术建模并试图发现那些易于识别的错误。
- Clang其实是集成在GCC编译器中的，并不一定需要xcode工具，也可以通过命令行配置来执行。
- 在xcode中使用非常简单，没错，只需要点击Build and analyze就坐等看结果了。



# IOS App主要内容

## Clang静态分析器

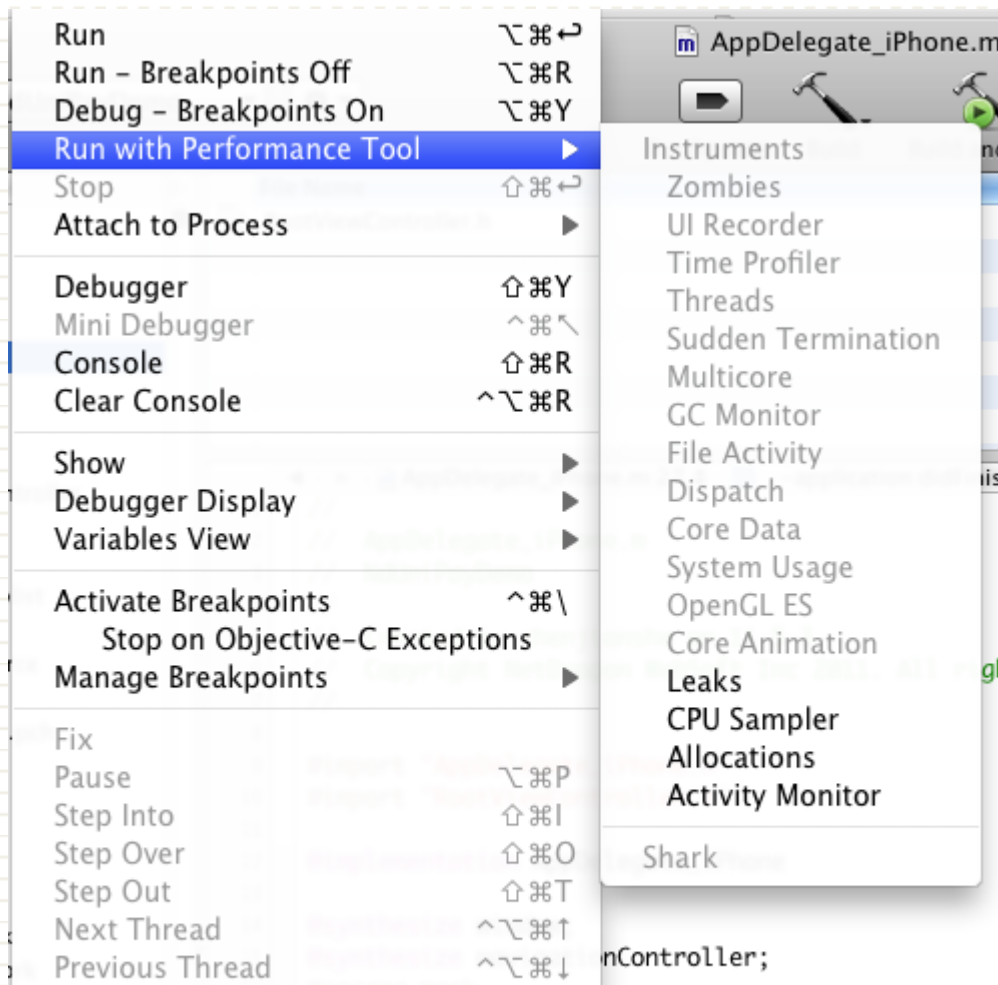
- Clang主要可以分析得出以下这些问题
- | 内存泄露
- | Retain和release的正确使用
- | 未初始化的变量
- | 未使用的变量
- | 无法到达的代码路径
- | 引用空指针
- | 除零
- | 死存储（赋值非从未使用的变量）
- | 类型不兼容的oc方法签名
- | 缺少dealloc
- 在应用程序的生命周期中，有很多的对象被创建，因此要发现一个较长时间内稳定但是泄露缓慢的对象的错误可能是比较困难的。Scan-build能捕捉到这样的引用技术错误。试图写一个已释放的变量会导致崩溃，静态分析器发现可以减少测试人员和用户的痛苦。
- 静态分析器可以遍历所有可能的路径并发现那些可能的会错过的错误



# IOS App主要内容

## UI Recorder

- Mac上存在的测试工具，有一个UI Recorder选项



# IOS App主要内容

## UI Recorder

- UI Recorder通过记录一些与GUI的交互操作，通过自动录制、检测和回放用户的应用操作。
- | 发现那些依赖于应用程序使用方式的的性能问题。就算只是通过简单的机械重放，通过可以发现bug的重现规律，可以发现可能需要和用户指定的交互动作才会出现的问题。
- | 可以自动进行需要的压力测试，通过以模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题。
- | 有效地帮助测试人员对复杂应用的不同发布版进行测试，提高测试人员的工作效率和质量，确保跨平台的、复杂的应用无故障发布及长期稳定运行

# IOS App Instruments



引出第一个工具“时间事件查看器”(自己杜撰的名字英文—Time Profiler),——他可以测量时间的间隔,中断程序执行,跟踪每个线程的堆栈.你可以想象下是xcode调试时按下暂停时的画面

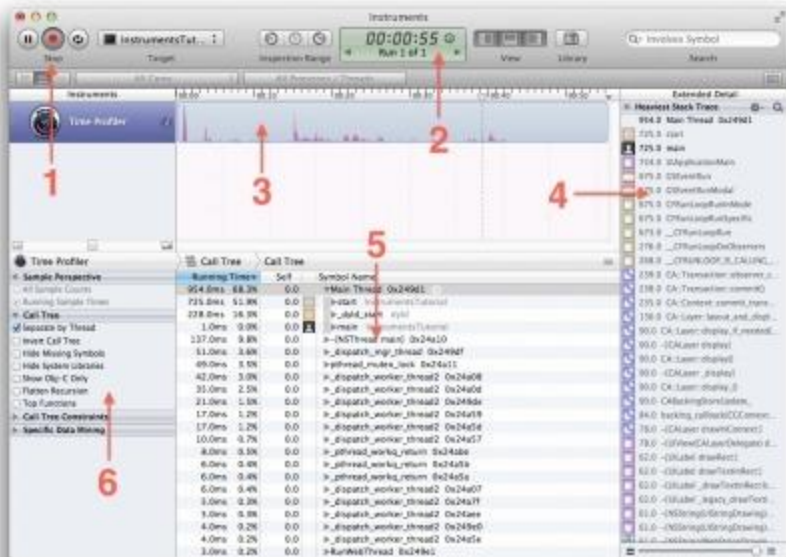


# IOS App主要内容

程序会启动  
Instruments,这时候  
你会看到一个选择  
窗口



# IOS App主要内容



- 1.录控按钮。中间的红色按钮将停止与启动它被点击时，应用程序目前正在分析。注意这实际上是停止和启动应用程序，而不是暂停它。
- 2.运行定时器和运行导航,定时器显示APP已经运行了多长时间,箭头之间是可以移动的。如果停止，然后使用录制按钮重新启动应用程序，这将开始一个新的运行。显示屏便会显示“run2 of 2”，你可以回到第一次运行的数据，首先你停止当前运行，然后按下左箭头回去。
3. 运行轨道。
4. 扩展面板,在时间探查仪器的情况下，它是用来跟踪显示堆栈
- 5.详细地面板。它显示了你正在使用的仪器的主要信息,这是使用频率最高的部门，可以从它这里看到cpu运行的时间



# IOS App主要内容

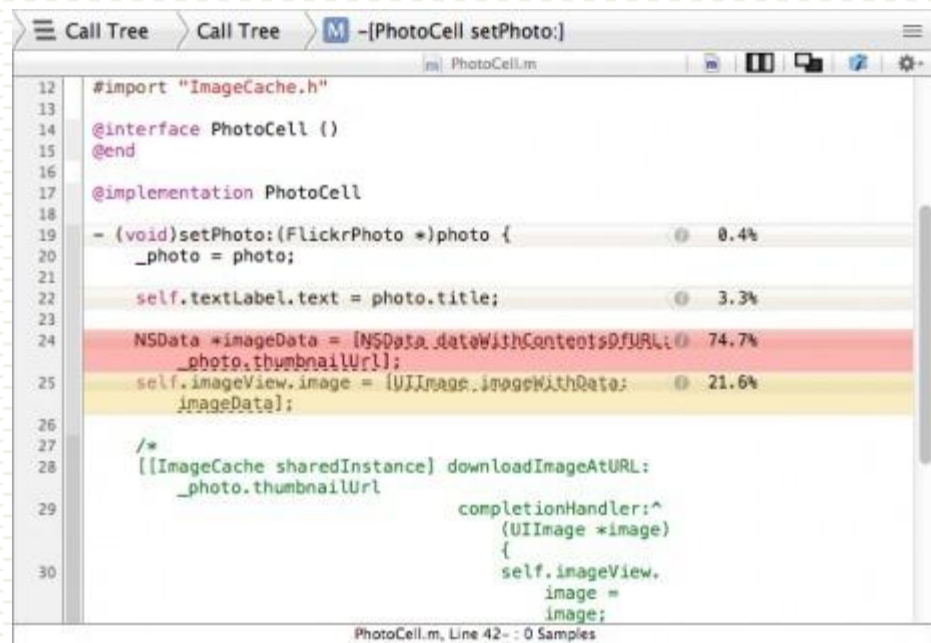


- **Separate by Thread:** 每个线程应该分开考虑。只有这样你才能揪出那些大量占用CPU的"重"线程
- **Invert Call Tree:** 从上倒下跟踪堆栈,这意味着你看到的表中的方法,将已从第0帧开始取样,这通常你是想要的,只有这样你才能看到CPU中话费时间最深的方法.也就是说FuncA{FunB{FunC}}勾选此项后堆栈以C->B-A把调用层级最深的C显示在最外面
- **Hide Missing Symbols:** 如果dSYM无法找到你的app或者系统框架的话,那么表中看不到方法名只能看到十六进制的数值,如果勾选此项可以隐藏这些符号,便于简化数据
- **Hide System Libraries:** 勾选此项你会显示你app的代码,这是非常有用的.因为通常你只关心cpu花在自己代码上的时间不是系统上的
- **Show Obj-C Only:** 只显示oc代码,如果你的程序是像OpenGL这样的程序,不要勾选侧向因为他有可能是C++的
- **Flatten Recursion:** 递归函数, 每个堆栈跟踪一个条目
- **Top Functions:** 一个函数花费的时间直接在该函数中的总和, 以及在函数调用该函数所花费的时间的总时间。因此, 如果函数A调用B, 那么A的时间报告在A花费的时间加上B.花费的时间,这非常真有用,因为它可以让你每次下到调用堆栈时挑最大的时间数字, 归零在你最耗时的方法。

# IOS App主要内容

Running Time	Self	Symbol Name
245.0ms 9.1%	0.0	▶-[PhotoCell setPhoto:] InstrumentsTutorial
11.0ms 0.4%	0.0	▶-[AppDelegate application:didFinishLaunchingWithOptions:] Instrum
5.0ms 0.1%	0.0	▶-[SearchResultsController tableView:cellForRowAtIndexPath:] In
5.0ms 0.1%	0.0	▶-[HomeController viewDidLoad] InstrumentsTutorial
3.0ms 0.1%	0.0	▶-[HomeController tableView:didSelectRowAtIndexPath:] Instrum
3.0ms 0.1%	0.0	▶-[HomeController searchBarSearchButtonClicked:] InstrumentsT
2.0ms 0.0%	0.0	▶-[ViewImageViewController didReceiveMemoryWarning] Instruments
1.0ms 0.0%	0.0	▶-[ViewImageViewController doneTapped:] InstrumentsTutorial
1.0ms 0.0%	0.0	▶-[HomeController tableView:cellForRowAtIndexPath:] Instrumer

大部分时间都花在更新表格照片了



```
12 #import "ImageCache.h"
13
14 @interface PhotoCell ()
15 @end
16
17 @implementation PhotoCell
18
19 - (void)setPhoto:(FlickrPhoto *)photo {
20     _photo = photo;
21
22     self.textLabel.text = photo.title;
23
24     NSData *imageData = [NSData dataWithContentsOfURL:
25         _photo.thumbnailUrl];
26     self.imageView.image = [UIImage imageWithData:
27         imageData];
28
29     /*
30     [[ImageCache sharedInstance] downloadImageAtURL:
31         _photo.thumbnailUrl
32         completionHandler:^(
33             UIImage *image)
34         {
35             self.imageView.
36                 image =
37                 image;
38         }];
39     */
40 }
```

几乎四分之三的时间花费在setPhoto: 方法都花在创造照片的图像数据!

现在可以看到的是什么问题,NSData's dataWithContentsOfURL 方法并不会立即返回,因为要从网上去数据,每次调用都需要长达几秒的时间返回,而此方法运行在主线程,可想而知会有什么结果了.

其实为了解决这个问题,类提供了一个ImageCache的后台异步下载的方法.

# IOS App主要内容

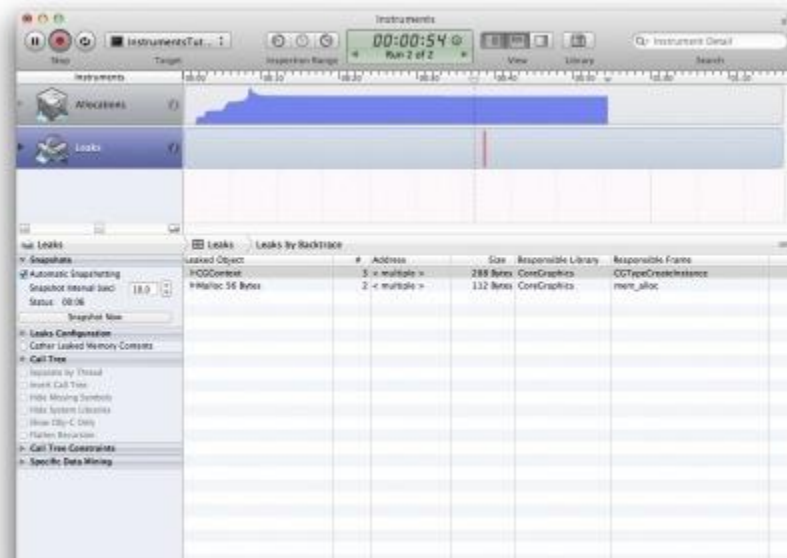
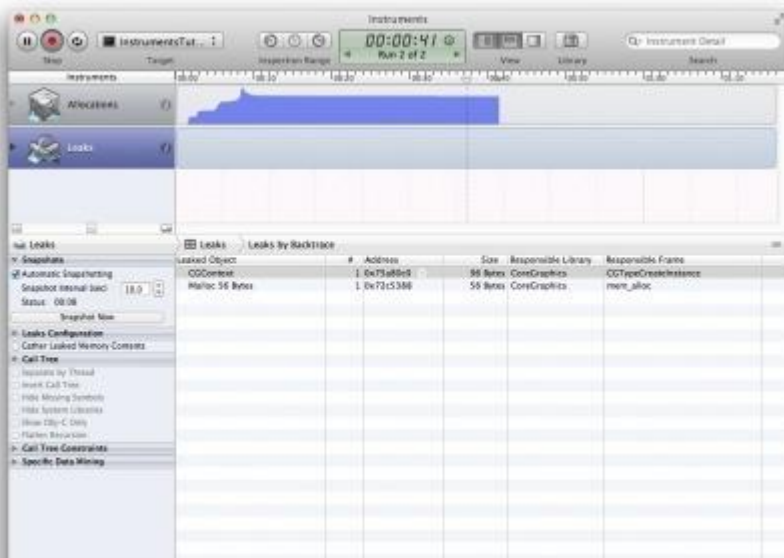
内存泄露





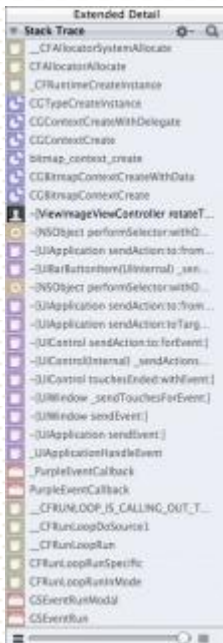
# IOS App主要内容

## 内存泄露



# IOS App主要内容

## 内存泄露



# Android App主要内容

- 监控工具Emmagee
- 监控工具GT
- JAVA内存管理
- 安卓内存管理
- 安卓内存泄露的检测方法
- 安卓内存泄露分析之DDMS和MAT
- 减少GC开销的措施

# 监控工具Emmagee简介

- Emmagee是网易杭州研究院QA团队开发的一个简单易上手的Android性能监测小工具，主要用于监控单个App的CPU，内存，流量，启动耗时，电量，电流等性能状态的变化，且用户可自定义配置监控的频率以及性能的实时显示，并最终生成一份性能统计文件。
- **Emmagee功能介绍**
  - 1、检测当前时间被测应用占用的CPU使用率以及总体CPU使用量
  - 2、检测当前时间被测应用占用的内存量，以及占用的总体内存百分比，剩余内存量
  - 3、检测应用从启动开始到当前时间消耗的流量数
  - 4、测试数据写入到CSV文件中，同时存储在手机中
  - 5、可以选择开启浮窗功能，浮窗中实时显示被测应用占用性能数据信息
  - 6、在浮窗中可以快速启动或者关闭手机的wifi网络

# Emmagee-用法



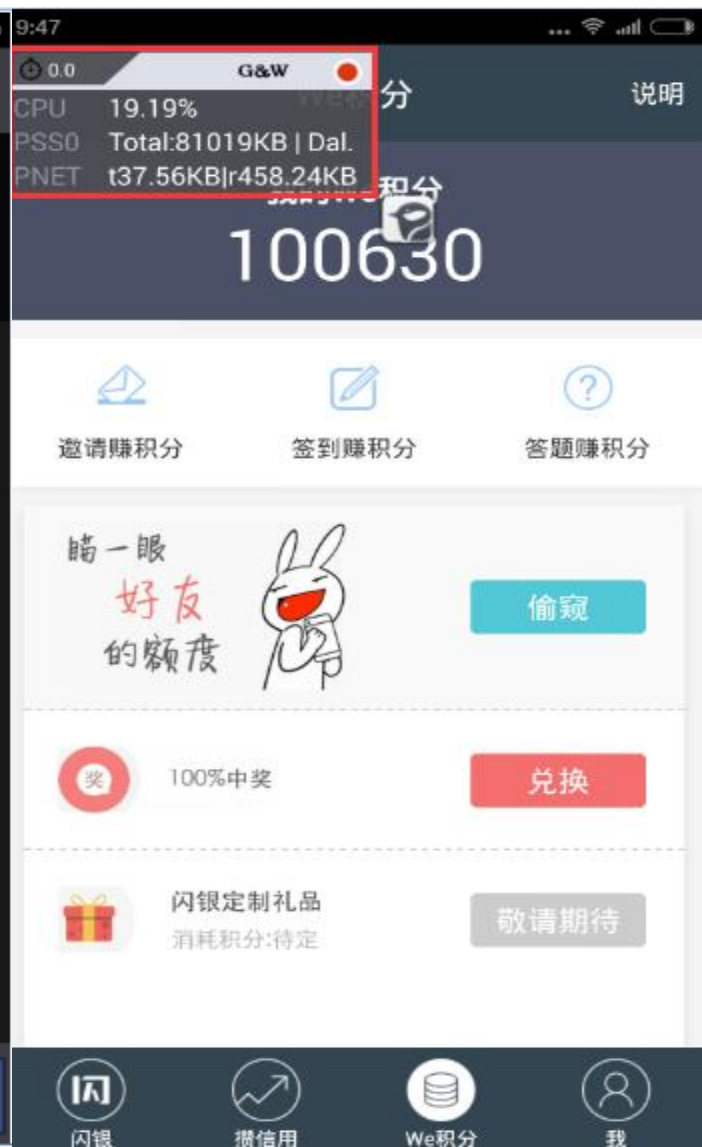
# Emmagee - 监控结果

应用包名	com.syqy.wecash																		
应用名称	闪银																		
应用PID	26368																		
机器内存大小(MB)	1967.83MB																		
机器CPU型号	ARMv7 Processor rev 2 (v7l)																		
Android系统版本	4.4.4																		
手机型号	MI 3																		
JID	10134																		
时间	栈顶Activ	应用占用内	应用占用内	机器剩余内	应用占用C	CPU总使用	cpu0总使用	cpu1总使用	cpu2总使用	cpu3总使用	流量(KB)	电量(%)	电流(mA)	温度(C)	电压(V)				
2015/7/16 14:41	Component	73.53	3.74	683.09	0	0	0	0	0	0	26	29	N/A	36.1	3.778				
2015/7/16 14:41	Component	73.75	3.75	690.55	11.1	43.57	43.53	42.06	44.55	0	28	29	N/A	36.1	3.778				
2015/7/16 14:41	Component	73.75	3.75	689.98	0.74	15	14.6	22.86	0	0	28	29	N/A	36.1	3.778				
2015/7/16 14:41	Component	73.75	3.75	663.8	0.71	33.85	30.04	42.79	0	0	28	29	N/A	36.1	3.778				
2015/7/16 14:41	Component	73.75	3.75	663.7	0.79	23.97	22.77	27.27	36	0	28	29	N/A	36.1	3.778				
2015/7/16 14:41	Component	73.94	3.76	662.98	2.48	12.6	11.35	43.48	0	0	30	29	N/A	36.1	3.778				
2015/7/16 14:41	Component	73.96	3.76	663.34	1.16	9.88	9.5	33.33	0	0	30	29	N/A	36.1	3.778				
2015/7/16 14:41	Component	74.09	3.77	663.08	1.92	10.77	10.85	14.29	0	0	31	29	N/A	36.3	3.85				
2015/7/16 14:41	Component	74.09	3.77	662.95	0.94	13.32	14	0	0	0	31	29	N/A	36.3	3.85				
2015/7/16 14:41	Component	74.36	3.78	662.49	1.55	13.98	14.23	0	0	0	31	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	74.19	3.77	630.07	0.62	57.76	51.47	74.1	52.33	80.85	31	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	114.68	5.83	613.59	30.08	49.58	48.42	66.87	36.73	34.85	240	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	114.89	5.84	587.96	2.63	23.03	19.06	44.44	28	14.29	240	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	114.89	5.84	587.84	0.59	8.45	8.63	0	0	0	240	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	142.2	7.23	608.68	26.99	46.57	45.54	42.63	58.04	46.32	458	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	158.48	8.05	616.5	29.31	45	43.97	52.94	0	0	561	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	158.6	8.06	590.95	5.86	16.85	14.9	45.71	0	0	561	29	N/A	36.3	3.85				
2015/7/16 14:42	Component	158.98	8.08	590.7	7.53	23.99	22.75	24	36.36	44.44	561	29	N/A	36.3	3.85				

# 监控工具GT简介

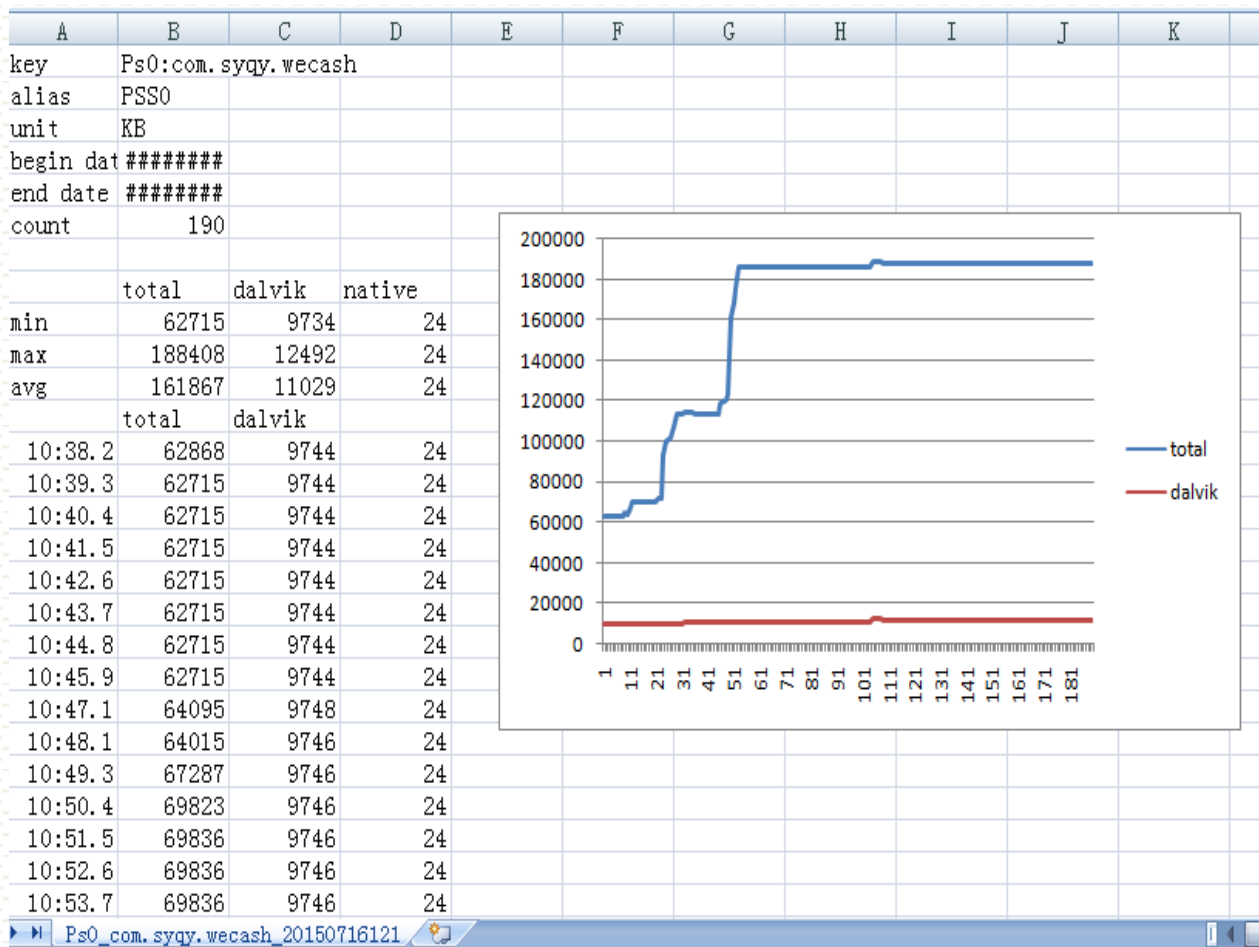
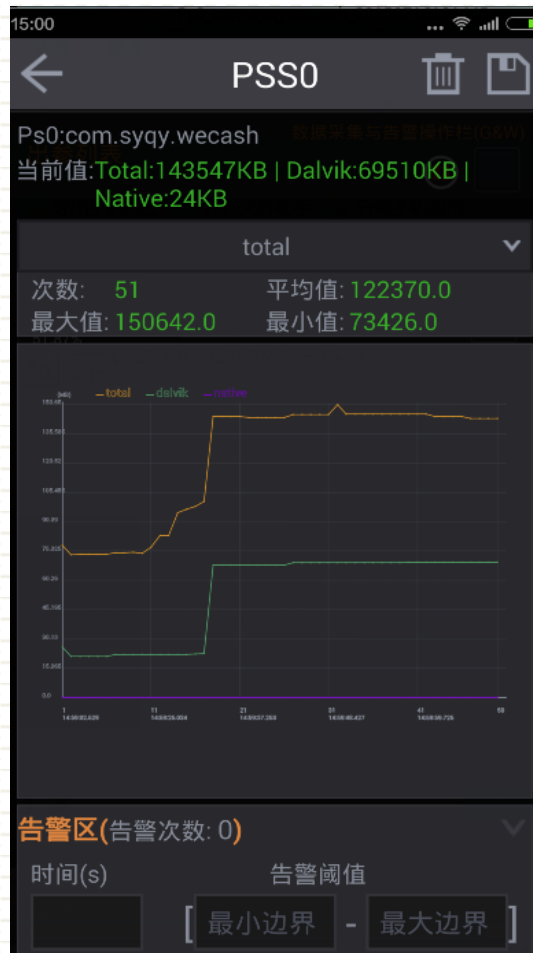
- **GT（随身调）**由腾讯移动评测实验室开发，仅凭一部手机，无需连接电脑，就可以对**APP**进行快速的性能测试(**CPU**、内存、流量、电量、帧率/流畅度等等)、开发日志的查看、**Crash**日志查看、网络数据包的抓取、**APP**内部参数的调试、真机代码耗时统计等等。
- **GT支持iOS和Android两个手机平台：**
  - **iOS版**是一个**Framework**包，必须嵌入**APP**工程，编译出带**GT**的**APP**才能使用；**iPhone**和**iPad**应用都能支持。
  - **Android版**由一个可直接安装的**GT控制台APP**和**GT SDK**组成，**GT控制台**可以独立安装使用，**SDK**需嵌入被调测的应用、并利用**GT控制台**进行信息展示和参数修改

# GT--使用





# GT—监控结果



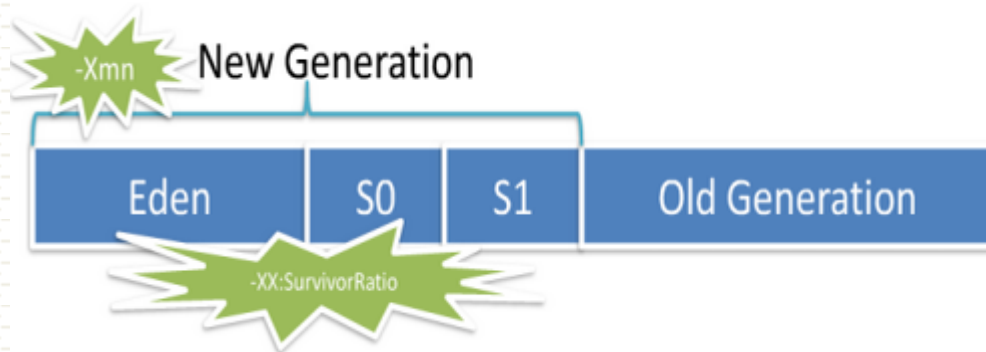
# JAVA内存管理--内存组成

**java内存组成：堆(Heap)和非堆(Non-heap)内存**

- 堆是JAVA运行时数据区域，所有类实例和数组的内存均从此处分配，在Java虚拟机启动时创建。简单来说堆就是Java代码可及的内存，是留给开发人员使用的；
- 在JVM中堆之外的内存称为非堆内存(Non-heap memory)，就是JVM留给自己用的，所有方法区、JVM内部处理或优化所需的内存(如JIT编译后的代码缓存)、每个类结构(如运行时常数池、字段和方法数据)以及方法和构造方法的代码都在非堆内存中。

# JAVA内存管理—堆内存分配

- JVM初始分配的内存由-Xms指定，最大分配的内存由-Xmx指定，在实际使用的过程中JVM会自动调整堆的大小，因此服务器一般设置-Xms、-Xmx相等以避免在每次GC后调整堆的大小。
- 年轻代= Eden space+2个survivor，年轻代用来存放新近创建的对象，年轻代的特点是对象更新速度快，在短时间内产生大量的“死亡对象”
- 年老代用于存放存活时间较长的对象



# JAVA内存管理--垃圾回收（GC）

- 在java开发过程中，是通过new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，在开发的过程中，不需要显式的去管理内存，java虚拟机会自动帮我们回收内存。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
- 一般情况下，当垃圾回收器在进行回收操作的时候，整个应用的执行是被暂时中止（stop-the-world）的。这是因为垃圾回收器需要更新应用中所有对象引用的实际内存地址。
- YoungGC：应用程序只能使用一个Eden区和一个survivor区，当其所使用的空间满了后将发生YoungGC，此时GC挂起程序，然后将Eden区和活动survivor区中的存活对象复制到另外一个非活动的survivor区中，然后一次性清除Eden区和活动survivor区，将原来的活动和非活动survivor区标记对调。
- FullGC：在指定次数回收后存活对象survivor区无法存放下时，将发生FullGC，此时将仍存在的对象移动到年老代中。

# JAVA内存管理--垃圾回收（GC）

- 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。
- 因此对于已经不再需要使用的对象，可以把它设置为null，这样当GC运行的时候，就好遍历到你这个对象已经没有引用，会自动把该对象占用的内存回收。我们没法像C++那样马上释放不需要的内存，但是我们可以主动告诉系统，哪些内存可以回收了。

# JAVA内存管理--内存泄露

造成OutOfMemoryError原因一般有2种：

- 1、内存泄露，对象已经死了，无法通过垃圾收集器进行自动回收，通过找出泄露的代码位置和原因，才好确定解决方案；
- 2、内存溢出，内存中的对象都还必须存活着，这说明Java堆分配空间不足，检查堆设置大小，检查代码是否存在对象生命周期太长、持有状态时间过长的情况。

何时发生内存泄露？

- 当应用线程在运行,并在运行过程中创建新对象,若这时内存空间不足,JVM就会强制地调用GC线程,以便回收内存用于新的分配。若GC一次之后仍不能满足内存分配的要求,JVM会再进行两次GC作进一步的尝试,若仍无法满足要求,则JVM将报“out of memory”的错误,Java应用将停止。



# 安卓内存管理

- 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，这也是安卓系统为什么比IOS慢的主要原因；
- 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，避免某一个应用崩溃而影响整个系统；
- JVM的内存管理是预分配的，重分配堆内存效率很低，且每个应用的JVM是独立的，可以多任务同时运行，因此不可能让每个任务都占用尽可能多的内存。如果堆很大，垃圾回收需要太多的时间，缓存的清理需要先暂停所有应用，然后进行清理、整理，所以就有了初始堆内存大小和最大堆内存大小相关的参数。

# Dalvik和Native

- android程序内存被分为2部分：native和dalvik，dalvik就是我们平常说的java堆，我们创建的对象是在这里面分配的，而bitmap是直接在native上分配的，对于内存的限制是 native+dalvik 不能超过最大限制。
- 用以下命令可以查看程序的内存使用情况：

```
shell@HM2014813:/ $ dumsys meminfo com.syqy.wecash
dumsys meminfo com.syqy.wecash
Applications Memory Usage (kB):
Uptime: 28315529 Realtime: 106650447

** MEMINFO in pid 4161 [com.syqy.wecash] **
```

	Pss Total	Private Dirty	Private Clean	Swapped Dirty	Heap Size	Heap Alloc	Heap Free
Native Heap	0	0	0	0	19824	15790	1025
Dalvik Heap	44873	44664	0	2588	49812	45411	4401
Dalvik Other	3158	3028	0	984			
Stack	560	560	0	100			
Ashmem	10348	10344	0	0			
Other dev	9865	9228	12	0			
.so mmap	16917	2308	13988	1740			
.apk mmap	126	0	56	0			
.ttf mmap	1386	0	872	0			
.dex mmap	4148	44	3036	68			
Other mmap	627	4	540	0			
Graphics	79632	79632	0	0			
GL	13664	13664	0	0			
Unknown	23630	23612	0	1636			
TOTAL	208934	187088	18504	7116	69636	61201	5426



# 安卓内存管理--Dumpsys 参数解释

- 其中Pss对应的TOTAL值为内存所实际占用的值
- Pss是占用的私有内存加上平分的共享内存。例如一块1M的共享内存被两个进程共享，那每个进程分500K。各进程的Pss相加基本等于实际被使用的物理内存，所以这个经常是最重要的参数。
- Private dirty，表示该进程私有的不跟disk数据一致的内存段。
- Private clean，包括该进程私有的干净的内存。包括该进程独自使用的so和进程的二进制代码段。
- 在dumpsys meminfo结果的其他值比较大的行：
  - ✓ .so表示映射的so库（vm area行的object名称包含.so字样），
  - ✓ .dex表示映射的.dex文件（dalvik的虚拟机二进制码），
  - ✓ Other dev表示映射其他的/dev（dalvik的heap也是映射到特殊的/dev上）

# 安卓内存管理--dalvik相关参数

和内存相关的参数在/system/build.prop文件中配置，手机不同其性能也不同，手机本身内存可能有大有小，所以针对每个应用的内存大小也不相同。

例如红米：

```
dalvik.vm.heapstartsize=8m  
dalvik.vm.heaptargetutilization=0.75  
dalvik.vm.heapminfree=2m  
dalvik.vm.heapmaxfree=8m  
dalvik.vm.heapgrowthlimit=96m  
dalvik.vm.heapsize=128m
```

米3：

```
dalvik.vm.heapstartsize=8m  
dalvik.vm.heapgrowthlimit=192m  
dalvik.vm.heapsize=512m  
dalvik.vm.heaptargetutilization=0.75  
dalvik.vm.heapminfree=512k  
dalvik.vm.heapmaxfree=8m
```

# 安卓内存管理-- dalvik参数含义

- **-dalvik.vm.heapstartsize**

堆分配的初始大小，调整这个值会影响到应用的流畅性和整体ram消耗。这个值越小，系统ram消耗越慢，但是由于初始值较小，一些较大的应用需要扩张这个堆，从而引发gc和堆调整的策略，会导致应用反应更慢。相反，这个值越大系统ram消耗越快，但是程序更流畅。

- **dalvik.vm.heapgrowthlimit**

受控情况下的极限堆（仅仅针对dalvik堆，不包括native堆）大小，dvm heap是可增长的，但是正常情况下dvm heap的大小是不会超过dalvik.vm.heapgrowthlimit的值。这个值控制那些受控应用的极限堆大小，如果受控的应用dvm heap size超过该值，则将引发oom（out of memory）

- **dalvik.vm.heapsize**

表示应用程序在任意时刻内可以使用的最大堆栈大小。

不受控情况下的极限堆大小，这个就是堆的最大值。不管它是不是受控的。这个值会影响非受控应用的dalvikheap size。一旦dalvik heap size超过这个值，直接引发oom。

# 内存泄露的检测方法

## 思路：

先记录当前的内存用量，然后在执行某种操作后，进行一次GC，如果内存没有明显的回落。此时即可以断定代码中可能存在内存泄漏。

## 检测方法：

使用DDMS中的Heap：

- 1) 打开DDMS并打开Devices视图和Heap视图
- 2) 点击选择要监控的进程
- 3) 选中Devices视图界面上的” update heap” 图标
- 4) 点击Heap视图中的” Cause GC” 按钮（相当于进行了一次GC的操作）
- 5) 执行被测业务
- 6) 点击Heap视图中的” Cause GC” 按钮

一般观察Data Object的Total值，正常情况下在每次GC后，这个值都会有明显的回落并会稳定在一个范围之内，说明代码中没有未被释放的内存；若这个值在每次GC后没有出现明显的回落，则说明代码中可能存在没有被释放的内存。

# DDMS工具的使用

The screenshot displays the Dalvik Debug Monitor (DDMS) interface, which is used for monitoring and debugging Android applications. The interface is divided into several sections:

- Top Bar:** Contains the title "Dalvik Debug Monitor" and standard window controls.
- Menu Bar:** Includes "File", "Edit", "Actions", and "Device".
- Left Panel:** Lists connected devices. The selected device is "xiaomi-hm\_note\_1s-7b", which is online and running Android 4.4.4. It has PID 4981 and TID 8601. The application "com.syqy.wecash" is also listed with PID 15373 and TID 8602 / 8700.
- Right Panel:** Displays heap information for the selected client. It includes tabs for "Info", "Threads", "VM Heap", "Allocation Tracker", "Sysinfo", "Network", "Emulator Control", and "Event Log". The "VM Heap" tab is active, showing heap updates after every garbage collection (GC).

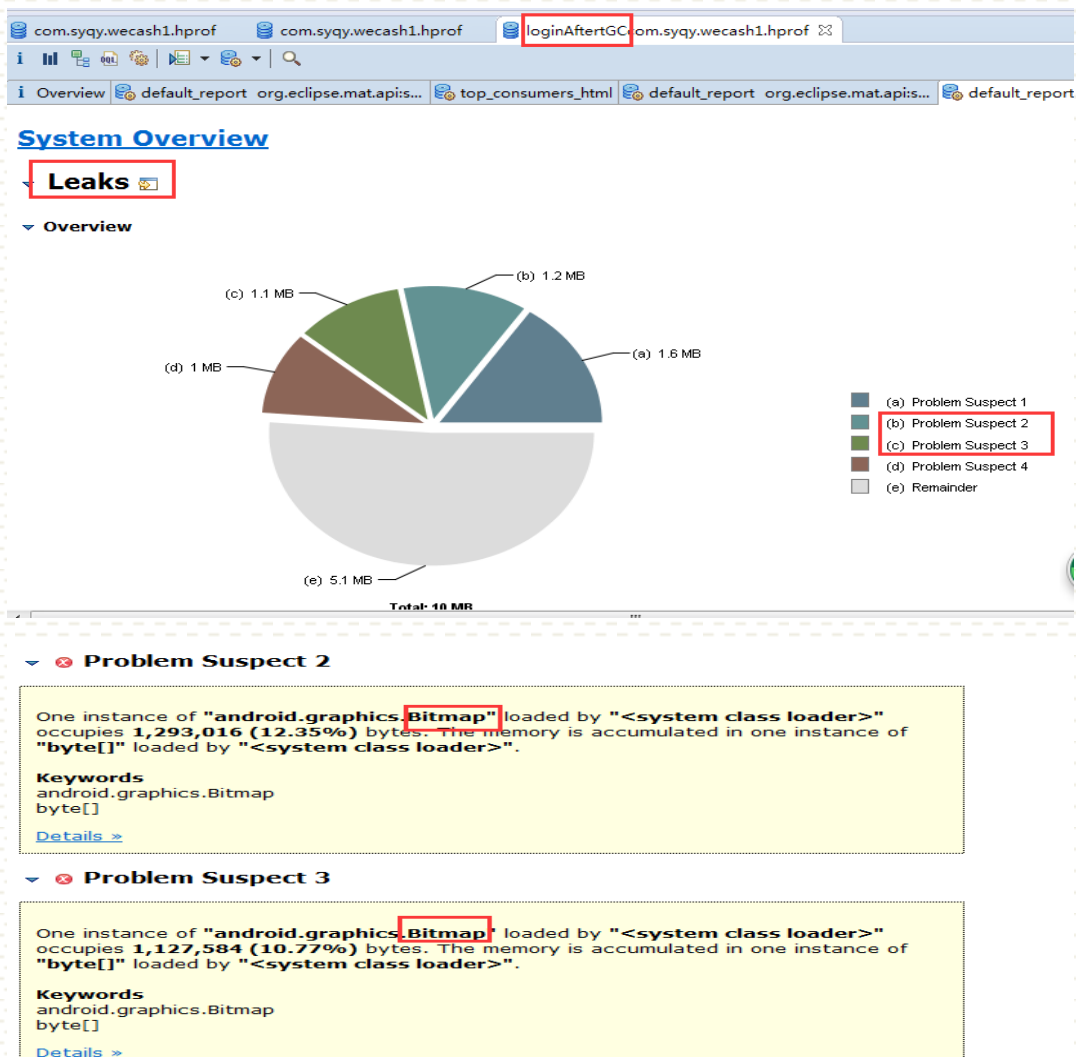
ID	Heap Size	Allocated	Free	% Used	# Objects
1	45,219 MB	40,991 MB	4,228 MB	90,65%	66,521

Below the heap information, there is a "Cause GC" button and a "Display:" dropdown set to "Stats". A table shows the distribution of heap objects:

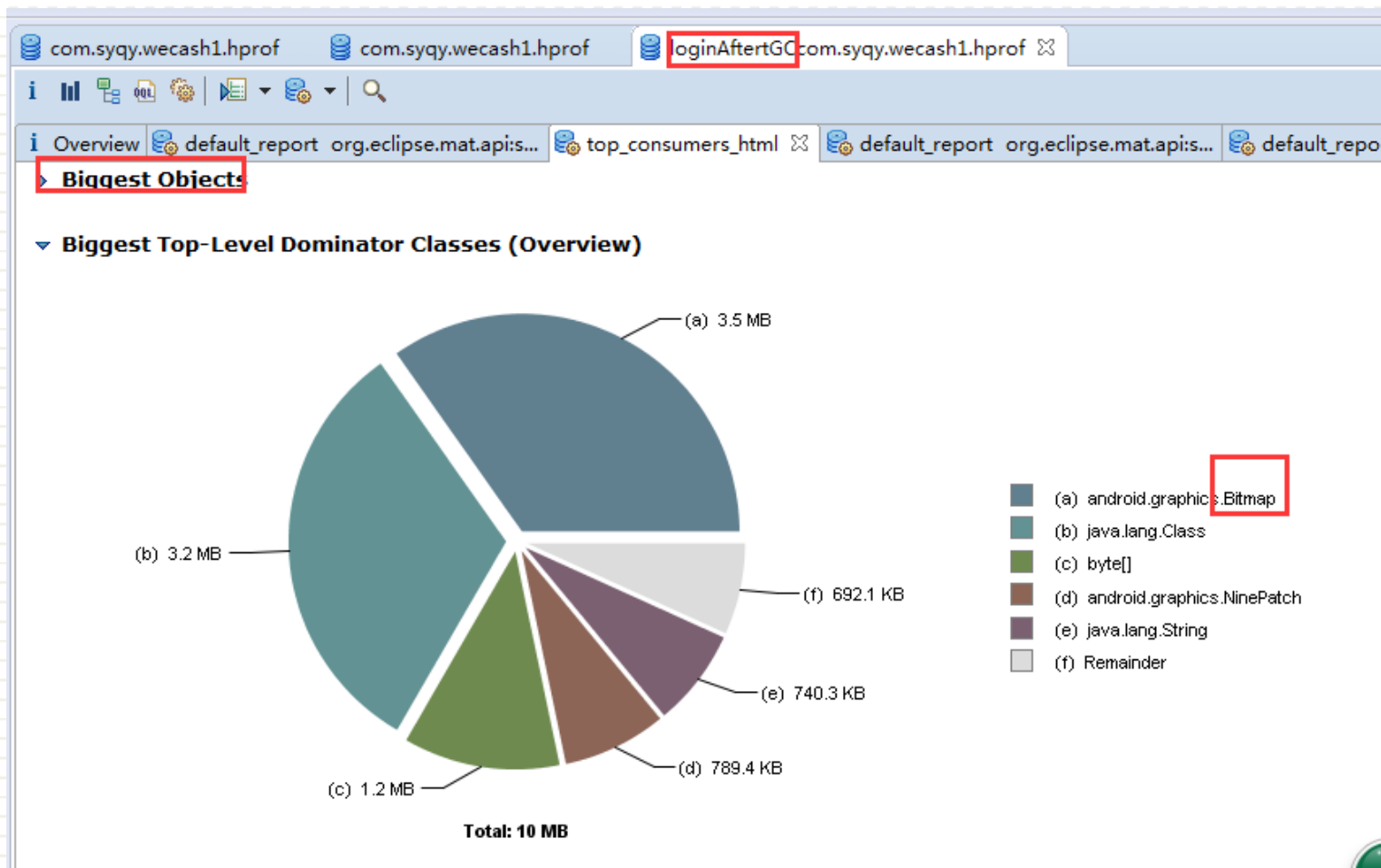
Type	Count	Total Size	Smallest	Largest	Media
free	2,843	4,151 MB	16 B	3,131 MB	80
data object	40,443	1,291 MB	16 B	744 B	32
class object	4,101	1,101 MB	168 B	42,398 KB	168
1-byte array (byte[], boolean[])	823	37,131 MB	24 B	4,930 MB	104
2-byte array (short[], char[])	15,397	1,059 MB	24 B	37,016 KB	56
4-byte array (object[], int[], float[])	5,685	407,641 KB	24 B	16,023 KB	40
8-byte array (long[], double[])	72	11,789 KB	24 B	4,008 KB	40
- Bottom Panel:** Contains a search bar for messages, a "verbose" dropdown, and a list of log messages. The messages are filtered to show all messages (no filters applied).

L...	Time	PID	TID	Application	Tag	Text
W	06-17 10:12:35.025	949	3771	MountService	MountService	getVolumeState(/mnt/sdcard/ext_sd)
W	06-17 10:12:35.025	949	1269	MountService	MountService	getVolumeState(/storage/extSdCard)
W	06-17 10:12:35.025	949	1277	MountService	MountService	getVolumeState(/mnt/sdcard/externa
W	06-17 10:12:35.025	949	1480	MountService	MountService	getVolumeState(/mnt/sdcard/externa
V	06-17 10:12:35.285	226	226	WLAN_PSA	WLAN_PSA	NL MSG, len[048], NL type[0x11] W
V	06-17 10:12:40.295	226	226	WLAN_PSA	WLAN_PSA	NL MSG, len[048], NL type[0x11] W

# MAT工具的使用1--分析内存泄露



# MAT工具的使用2--大对象分析



# 减少GC开销的措施

根据上述GC的机制,程序的运行会直接影响系统环境的变化,从而影响GC的触发。若不针对GC的特点进行设计和编码,就会出现内存驻留等一系列负面影响。为了避免这些影响,基本的原则就是尽可能地减少垃圾和减少GC过程中的开销。具体措施包括以下几个方面:

- 不要显式调用System.gc()
- 尽量减少临时对象的使用,临时对象在跳出函数调用后,会成为垃圾,少用临时变量就相当于减少了垃圾的产生
- 对象不用时最好显式置为Null,一般而言,为Null的对象都会被作为垃圾处理,所以将不用的对象显式地设为Null,有利于GC收集器判定垃圾,从而提高了GC的效率。
- 尽量使用StringBuffer,而不用String来累加字符串
- 能用基本类型如Int,Long,就不用Integer,Long对象,基本类型变量占用的内存资源比相应对象占用的少得多,如果没有必要,最好使用基本变量。
- 尽量少用静态对象变量,静态变量属于全局变量,不会被GC回收,它们会一直占用内存。
- 分散对象创建或删除的时间

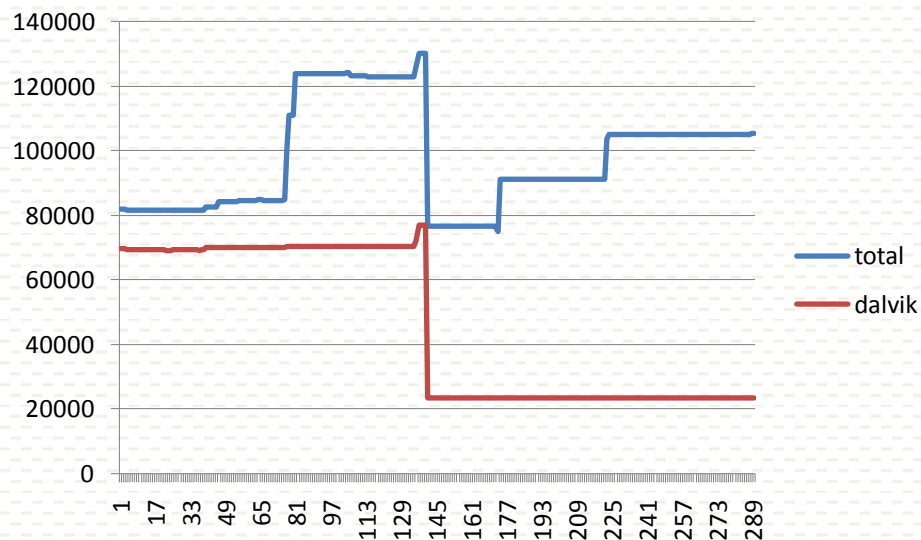


# wecash主要页面跳转内存使用情况

各页面内存使用情况（使用GT记录数据）：

- 启动：80M
- Login：122M
- 攒信用：130M—>76M （内存使用超过限制，发生了垃圾回收）
- WE积分：90M
- 我（用户中心）：105M

注：微信使用的过程中内存占用一般在80M左右



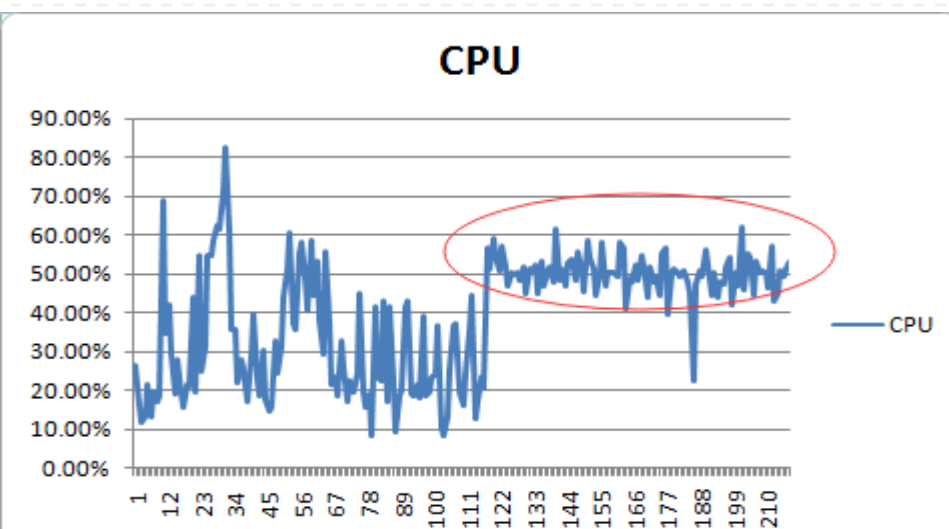
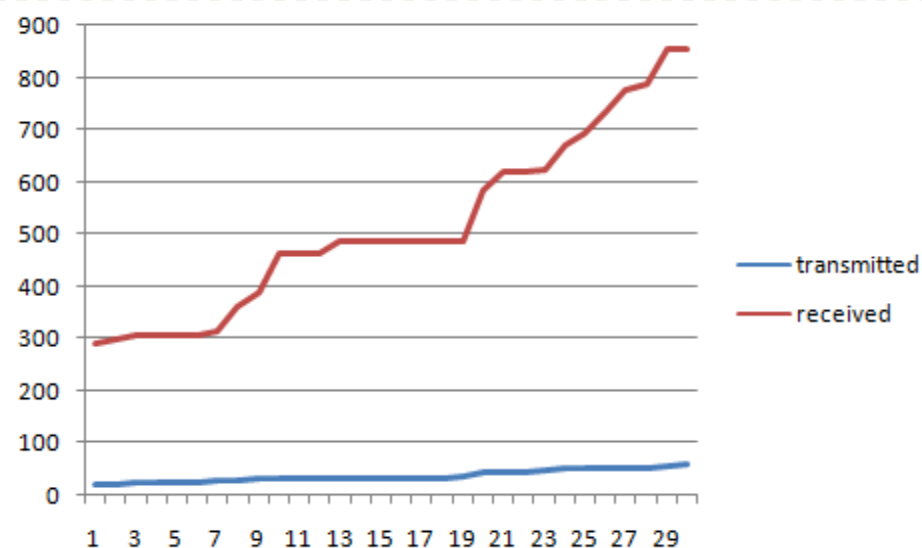
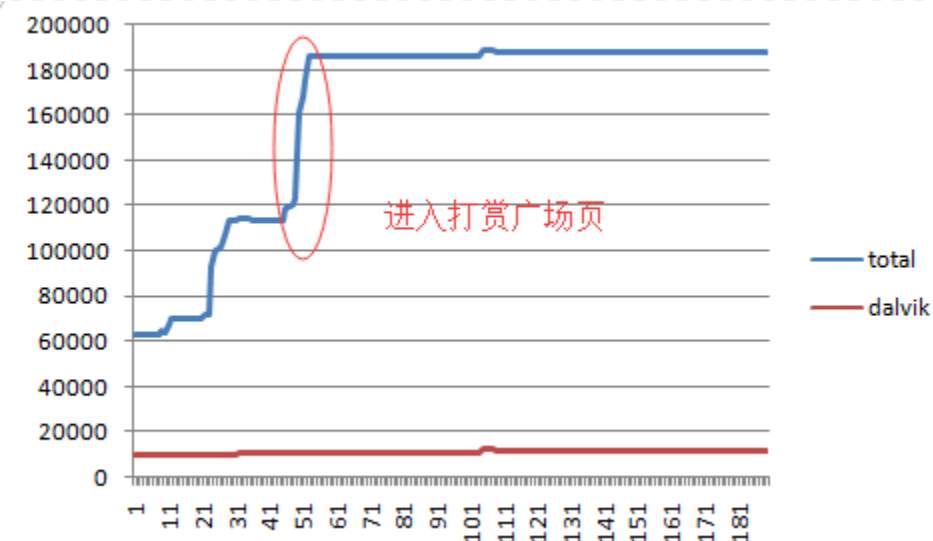
# Wecash 打赏

主要问题：

- 内存
  - 进入打赏广场页：内存上升了74M
- CPU
  - 停留在打赏页，CPU占用持续高达50%左右
- 流量

在进入打赏广场页时，接收到的数据量比较大，通过抓包发现主要是大的文件和图片。

# Wecash 打赏—监控记录



# Wecash征信报告

主要问题：

- 内存

- 进入征信引导页：内存上升了34M
- 查看“如何授权”提示页：内存上升了89M

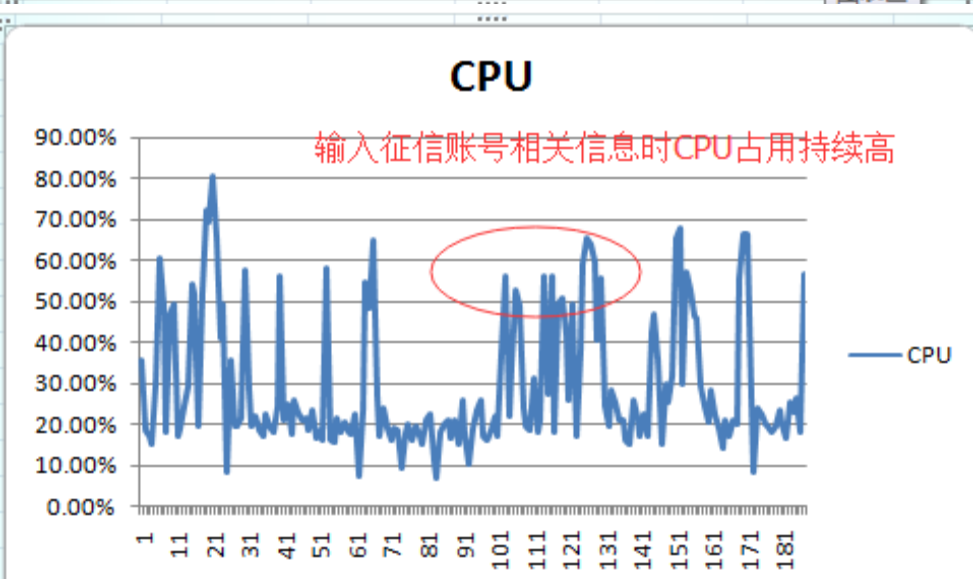
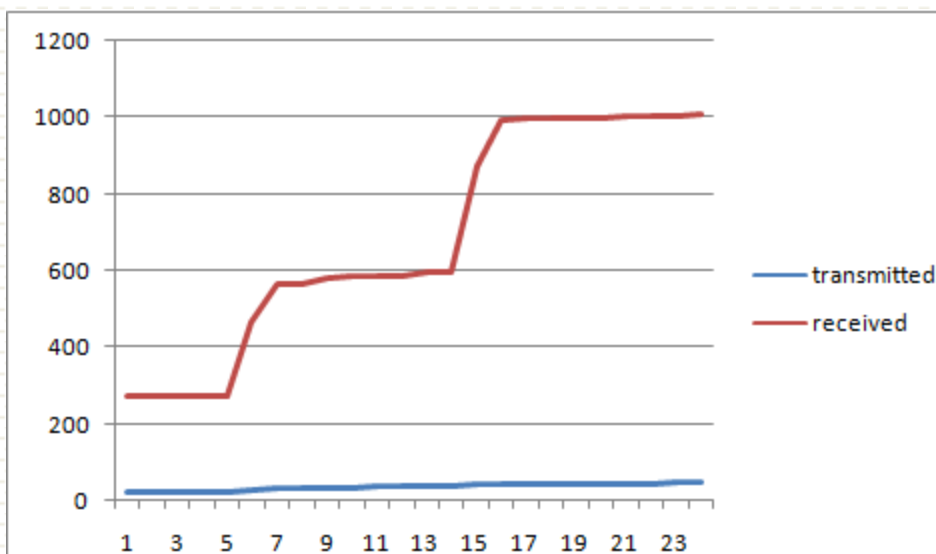
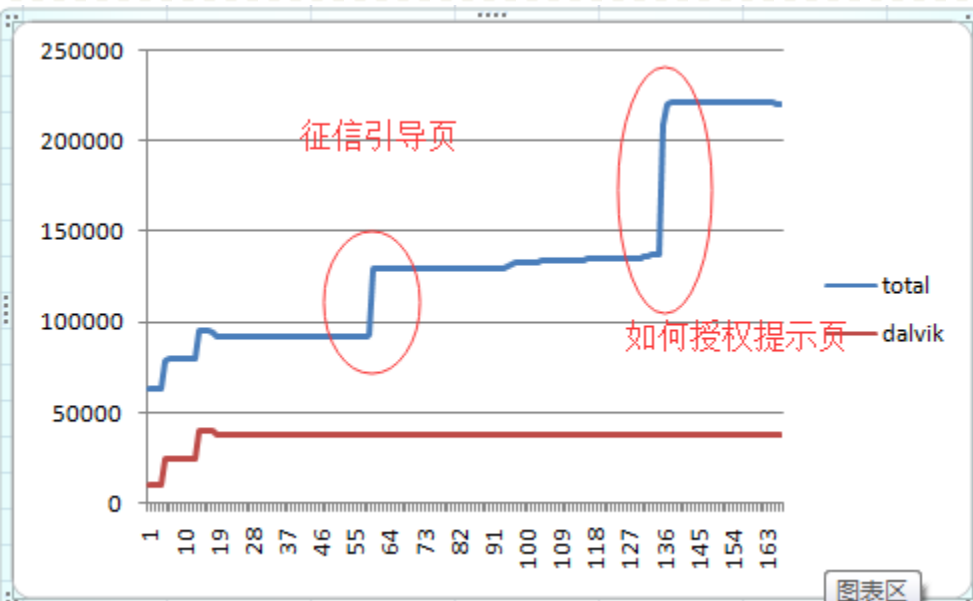
- CPU

- 在输入征信账号页面，CPU占用持续高达50%左右

- 流量

在进入征信引导页和如何授权提示页时，接收到的数据量比较大，通过抓包发现主要是大的文件和图片。

# 征信报告—监控记录



# 征信报告一抓包

auth.wecash.net:8611/credit\_invest\_how.html

Device: <Select model> Network: No throttling UA: Mozilla/5.0 (iPhone; CPU iPhone OS 8\_0 like Mac OS X; en-US; rv:4.0) AppleWebKit/537.51 (KHTML, like Gecko) Mobile Safari/537.51

1. 在“人民银行征信中心”页面申请信用信息地址: <https://ipcrs.pbccrc.org.cn> (建议电脑/手机浏览器中打开)

(1) 进入征信中心, 完成注册后立即登录

(2) 进入新手导航, 点击【下一步】

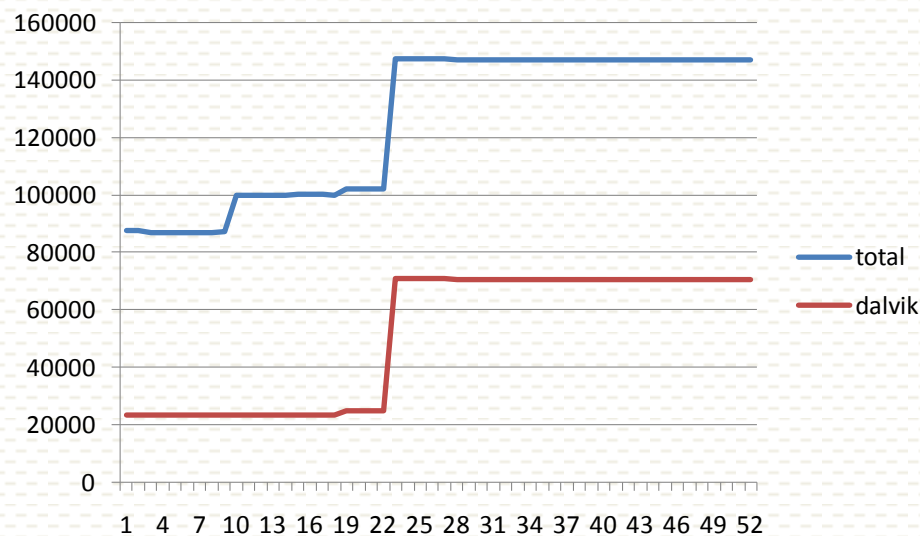
(3) 选择“问题验证”后, 点击“下一步”

Name	Path	Meth...	Status	Type	Initiator	Size	Time	Timeline
						Content	Latency	
1.2.png	/css/pbccrc	GET	200 OK	imag...	credit_inves...	84.5 KB	1.43 s	
1.5.png	/css/pbccrc	GET	200 OK	imag...	credit_inves...	84.2 KB	1.42 s	
1.3.png	/css/pbccrc	GET	200 OK	imag...	credit_inves...	82.3 KB	1.32 s	
1.4.png	/css/pbccrc	GET	200 OK	imag...	credit_inves...	81.7 KB	1.49 s	
1.1.png	/css/pbccrc	GET	200 OK	imag...	credit_inves...	81.3 KB	1.48 s	
base.css	/css	GET	200 OK	text/c...	credit_inves...	77.4 KB	1.04 s	
credit_invest_how.html		GET	200 OK	text/...	Parser	77.1 KB	1.04 s	
credit_invest_how.css	/css	GET	200 OK	text/c...	Parser	68.8 KB	626 ms	
bar.js	hgimnogjllphhkhlmebbmigja...	GET	200 OK	appli...	68.5 KB	624 ms		
bar.css	hgimnogjllphhkhlmebbmigja...	GET	200 OK	text/c...	credit_inves...	2.9 KB	83 ms	
bar.html	hgimnogjllphhkhlmebbmigja...	GET	200 OK	text/...	Parser	2.7 KB	80 ms	
credit_invest_how.html		GET	200 OK	text/...	Other	1.4 KB	63 ms	
credit_invest_how.css	/css	GET	200 OK	text/c...	Parser	2.5 KB	62 ms	
bar.js	hgimnogjllphhkhlmebbmigja...	GET	200 OK	appli...	bar.html:19	(from cache)	3 ms	
bar.css	hgimnogjllphhkhlmebbmigja...	GET	200 OK	text/c...	Parser	3 ms		
bar.html	hgimnogjllphhkhlmebbmigja...	GET	200 OK	text/...	bar.html:5	(from cache)	2 ms	
credit_invest_how.html		GET	200 OK	text/...	content.js:1...	(from cache)	2 ms	
credit_invest_how.css	/css	GET	200 OK	text/c...	Parser	1 ms		
bar.js	hgimnogjllphhkhlmebbmigja...	GET	200 OK	appli...	Script	1 ms		
bar.css	hgimnogjllphhkhlmebbmigja...	GET	200 OK	text/c...	Parser	1 ms		
bar.html	hgimnogjllphhkhlmebbmigja...	GET	200 OK	text/...	Script	1 ms		

11 requests | 400 KB transferred | 1.70 s (load: 1.71 s, DOMContentLoaded: 187 ms)

# wecash查看版本说明

- MI3查看版本说明操作，内存飙升40M(GT)，且切换到其他页面后，内存长时间不释放。（注：没有向服务器发送请求）





# 结束语

- 谢谢！