IOS客户端性能测试方案

版本 <1.0>

**目 录**

[1. 文档介绍 4](#_Toc403634221)

[1.1.性能测试目的 4](#_Toc403634222)

[1.2.读者对象 4](#_Toc403634223)

[2. 测试需求 4](#_Toc403634224)

[2.1. 性能测试功能点 4](#_Toc403634225)

[2.2. 性能需求 4](#_Toc403634226)

[2.3. 测试工具 5](#_Toc403634227)

[3. 性能测试执行策略及方法 6](#_Toc403634228)

[3.1. iPhone客户端CPU、内存性能测试 6](#_Toc403634229)

[3.1.1. windows环境下CPU、内存测试方法 6](#_Toc403634234)

[3.1.2. Mac环境下CPU、内存测试方法 9](#_Toc403634235)

[3.1.3. CPU、内存性能数据统计方法 11](#_Toc403634236)

[3.1.4. KTV线上包房唱歌实例分析 16](#_Toc403634237)

[3.1.5. iPhone客户端CPU、内存测试执行策略 17](#_Toc403634238)

[3.2. 电量、流量消耗性能测试 18](#_Toc403634239)

[3.2.1. 电量、流量消耗测试方法 18](#_Toc403634241)

[3.2.2. 电量、流量消耗测试方法 20](#_Toc403634242)

[3.3. 响应时间 20](#_Toc403634243)

[3.3.1. 响应时间测试方法 21](#_Toc403634245)

[3.3.2. 响应时间测试方法 25](#_Toc403634246)

[4. 附录.IOS客户端输出日志查看 26](#_Toc403634247)

[4.1. 日志打印方法 26](#_Toc403634248)

[4.2. 日志查看方法 28](#_Toc403634249)

[4.2.1. Tail命令实时监控文本日志 28](#_Toc403634253)

[4.2.2. Tail+grep命令实时筛选并查看文本日志 28](#_Toc403634254)

# 文档介绍

## 1.1.性能测试目的

App直接面向产品使用用户，客户端产品在CPU、内存占用异常情况可能导致客户端界面响应卡顿和手机发热异常，甚至可能因为内存溢出和CPU过高导致应用崩溃，直接影响用户使用体验；客户端产品在移动设备使用还要考虑到电量和流量的消耗，其中移动数据流量的消耗直接关系到用户使用客户端的成本。

鉴于上述几个可能存在的问题，客户端在功能测试稳定后需要引入性能测试，通过降低各项资源占用率以增强用户体验和提高产品的稳定性；

## 1.2.读者对象

本方案的预期读者：项目负责人、ios测试人员和其他相关人员。

# 测试需求

## 性能测试功能点

内存：客户端涉及到图片处理、较多复杂界面展现、音频数据处理

CPU：客户端涉及到音频播放、编解码、持续性数据网络发送和接收数据、动画展现

电量消耗：客户端涉及播放音频视频、长时间使用无线网络

流量：客户端涉及图片下载、在线音频播放

## 性能需求

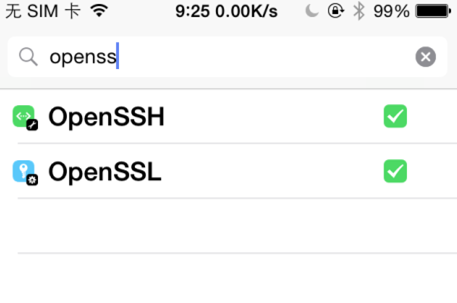
1. 内存：客户端长时间使用不会出现内存持续增加，相同步骤执行多次不会出现内存反复增加；
2. CPU：cpu使用率不能持续保持较高水平（cpu长时间保持较高水平会导致手机发热和电量消耗快），普通界面跳转不存在异常的cpu峰值过高情况；
3. 电量消耗：无线网络的使用时间、CPU和内存的占用等因素直接影响电量的消耗，尽可能减少接口情况、降低CPU和内存使用以保证电量消耗正常；
4. 流量：有效的处理图片和接口数据的缓存可以降低用户数据流量的使用，所以因尽可能避免相同图片重复下载，相同的接口数据反复情况的情况；

## 测试工具

1、PuTTY（内存及CPU监控软件）

简介：Putty是一个免费的、Windows 32平台下的telnet、rlogin和ssh[客户端](http://baike.baidu.com/view/930.htm" \t "_blank)，我们可以通过SSH远程登录iOS设备；

2、一部越狱的iPhone手机，越狱Cydia中添加<http://apt.178.com>源，在搜索中搜索下列插件并按照，需要在Cydia中下载安装如下插件有：openssh、top、adv-cmds、core utilities、apple file conduit、Datameter、appsync for ios\*；  


  
3、文档中提到的Camtasia Studio 8、itools（经典版）、iTunes、360wifi、Xshell等工具都可以在网上下载到；

# 性能测试执行策略及方法

## iPhone客户端CPU、内存性能测试



### windows环境下CPU、内存测试方法

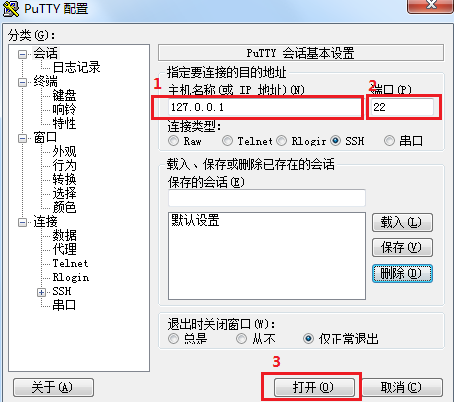
Windows平台下监控方法：使用Putty、Xshell等ssh远程访问工具通过shell脚本记录指定应用进程的cpu和内存数据，具体步骤如下；

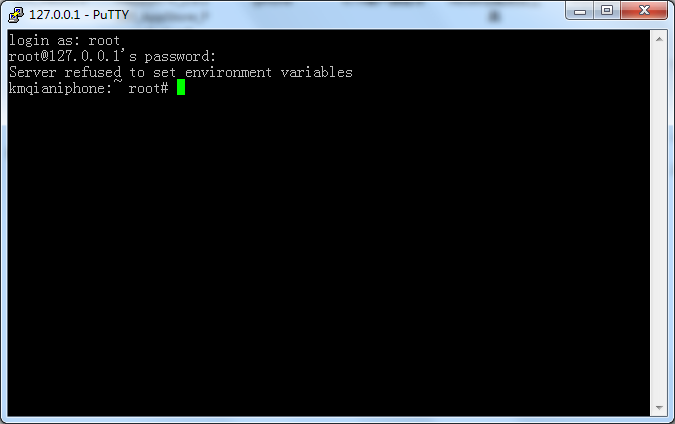
* **手机连接ssh并登录root账户**

1、越狱iPhone手机通过USB线连接电脑，再使用itools工具在高级设置中打开SSH隧道；



2、启动Putty工具配置ssh路径和端口号，命令行界面输入登录用户名root和默认密码alpine（Xshell工具的登录方法可以参考连接普通linux服务器的方式，地址就是127.0.0.1，端口号默认为22，用户名和密码默认为root和alpine）；





* **性能测试脚本执行**

测试脚本：for ((i=0; i<10;i++));do ps auxm|grep 'HappyKTV(进程名)'>>data.log;sleep 1 ;done

上述脚本的功能是间隔1s将指定进程的CPU和内存数值输出到data.log文件中，输出10次终止，其中i<10（指定输出次数），sleep 1（指定间隔时间 单位s）；日志文件 data.log默认在/var/root文件夹中（可以使用itools工具在文件系统->越狱系统->var-> root目录中找到）；

1、将脚本复制到上图窗口中（右击则粘贴）敲回车执行即可（执行过程中不可以关闭窗口，如果想停止执行使用Ctrl+C即可），如下图所示



脚本执行过程中想要查看具体数值可以再使用tail命令监控日志文件，即使用tail –f /var/root/data.log 命令实时监控输出性能数据日志数值（不能再性能脚本执行的窗口执行此操作，可以再新建一个窗口执行此操作，所以性能测试过程中同时也可以开多个窗口查看日志）；

执行完成后则如下图所示，可以将data.log文件拷贝出来分析，分析方法见下节：



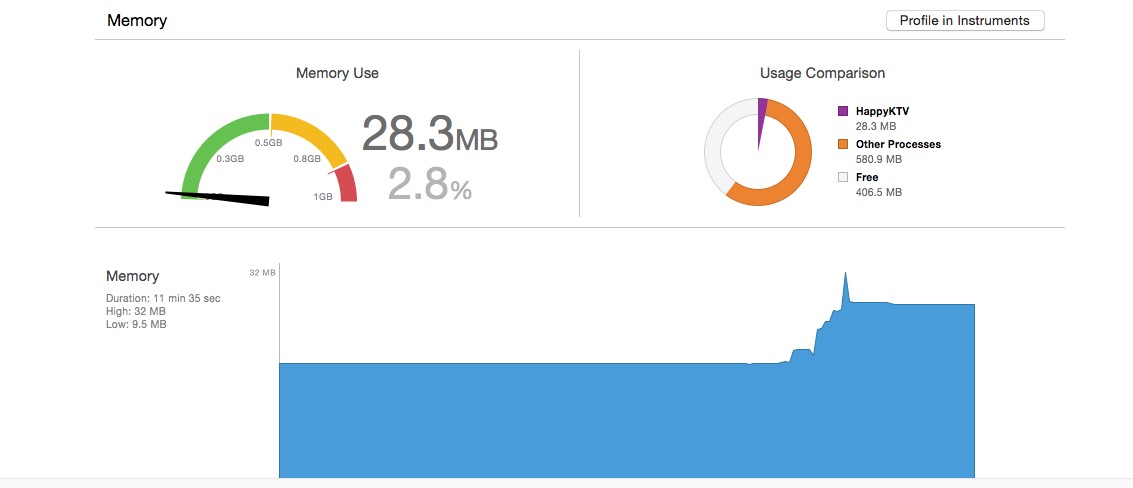
### Mac环境下CPU、内存测试方法

上述windows环境下使用ps命令测试内存的结果相较于xcode自带的内存检测工具计算出的内存要平均高15~20Mb左右（原因未知，估计是平台差异，但是内场增减的趋势是相同的不影响分析）；

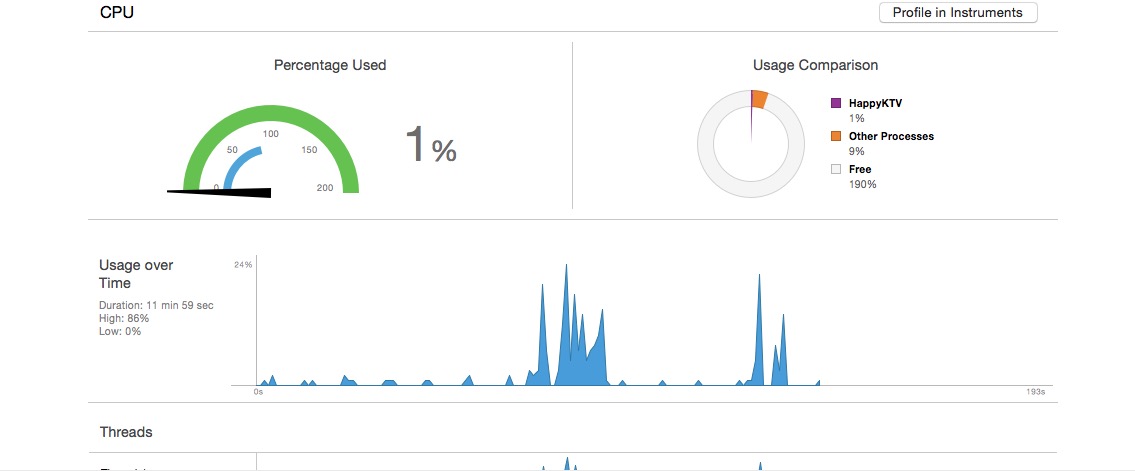
如果有条件使用mac电脑的话，苹果的Xcode工具中自带的性能检测工具在内存和CPU分析更为方便和简单，在ios8之后还增加了流量分析、端口ip检测和磁盘IO性能分析；此外，Xcode编译环境下进行的功能测试对于发现崩溃性问题定位原因方便准确多了，所以有条件的同事可以在Xcode编译源代码的情况下进行相关测试和查找问题；

* **Xcode自带的性能监控工具**

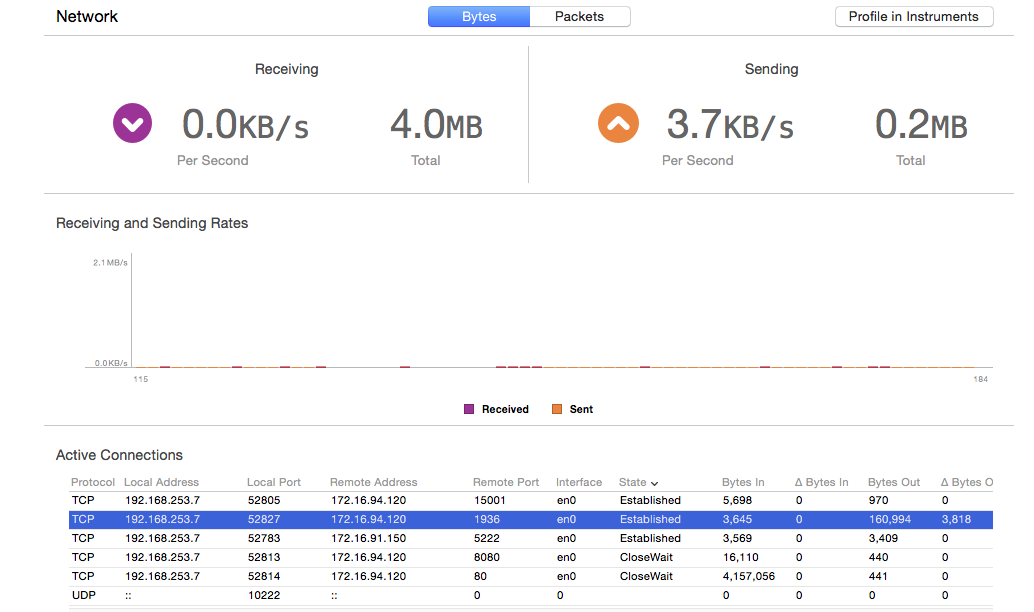
**内存分析插件：**



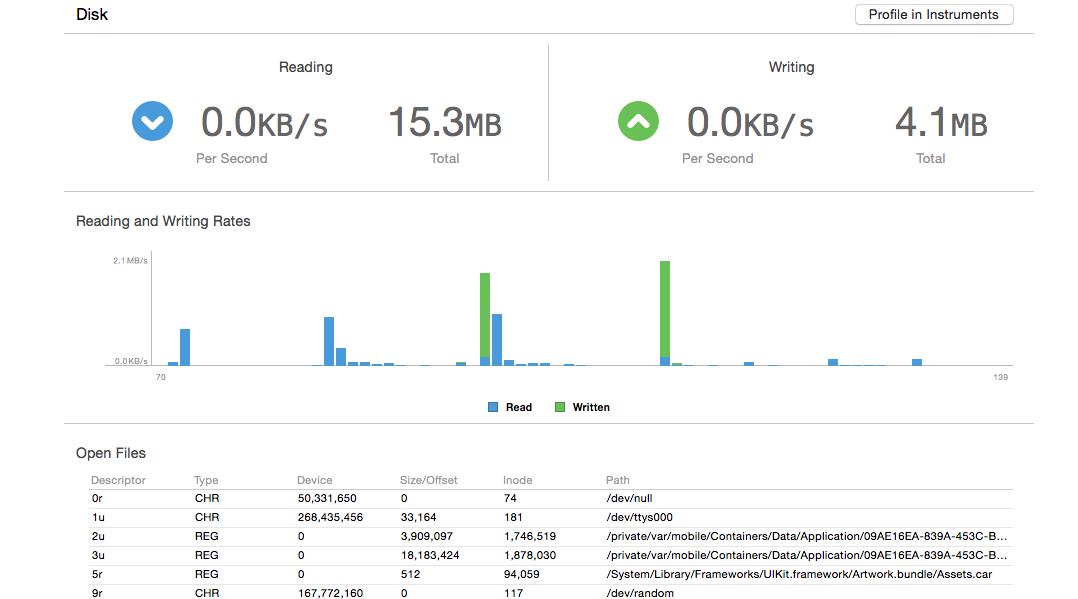
**CPU监控插件：**



**上下行流量及端口占用监控插件（ios8、xcode6之后版本支持）：**

****

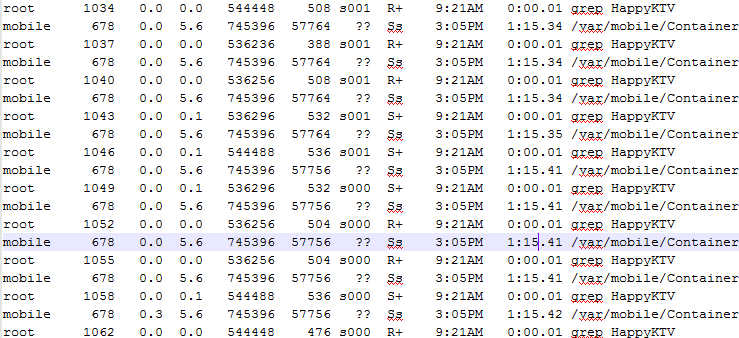
**磁盘IO读写性能（ios8、xcode6之后版本支持）：**

****

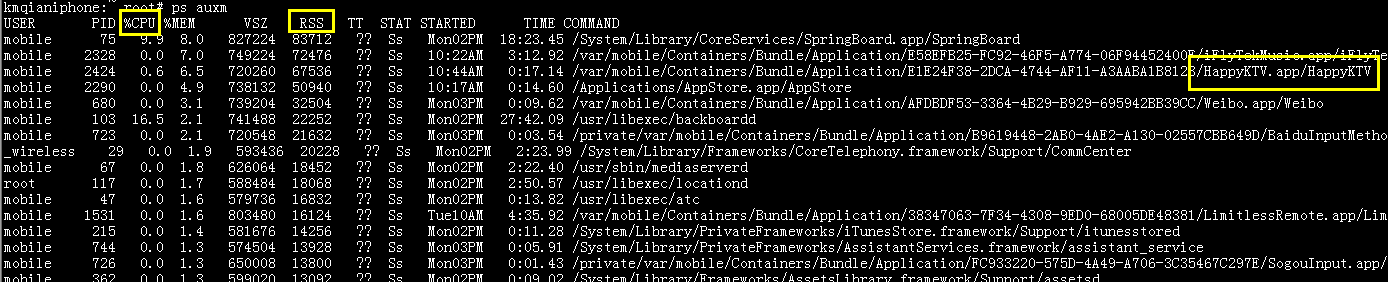
关于Xcode进行的性能测试操作比较简单便捷，所以这里就不做特别详细的描述，有条件的可以向相关开发人员学习怎么使用Xcode分析内存、cpu、流量监控和磁盘读写性能；

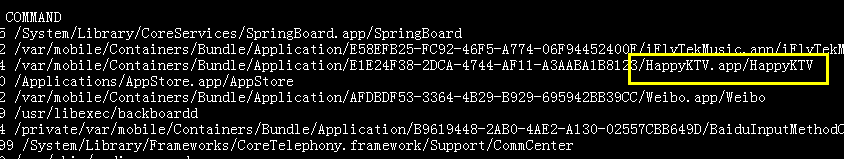
### CPU、内存性能数据统计方法

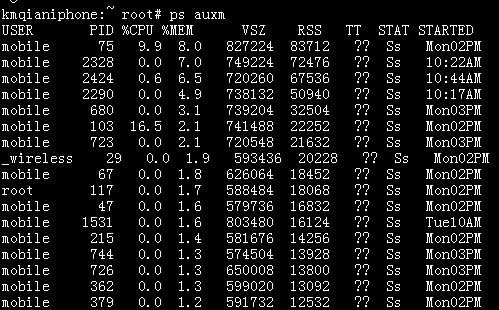
输出到data.log中的日志样式大概如下，其中每一列都对应一个字段，可参考下方图文说明相关字段解释；



SSH连接情况下直接使用ps auxm查看，可以看出统计脚本统计的字段名称和值，如下图：







部分字段名称含义：

USER 进程的属主；

PID   进程的ID；

PPID   父进程；

%CPU   进程占用的CPU百分比；

%MEM  占用内存的百分比；

VSZ     该进程使用的虚拟內存量（KB）；

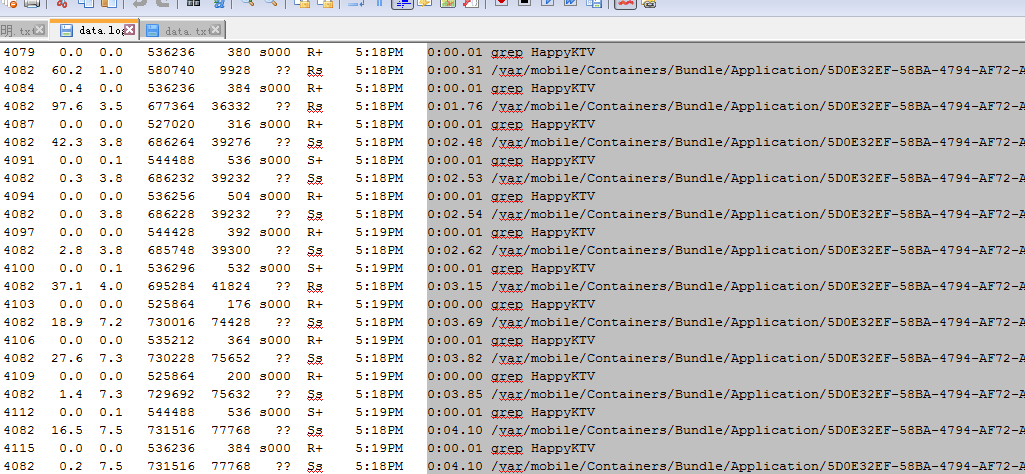
RSS     该进程占用的固定內存量（KB）（驻留中页的数量）；

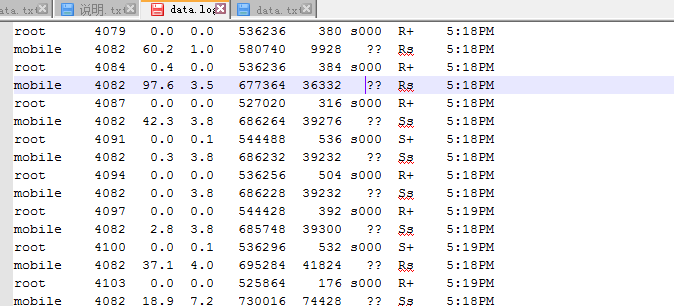
COMMAND   命令的名称和参数；（脚本就是根据这个进程名称筛选的，如上图中圈出的HappyKTV）

上述字段中，其中我们需要关心的就只有CPU%、RSS，分别指的是应用进程的cpu占用率和物理内存使用量；所以我们的目标就是筛选出这两个字段并导入到Excel中可以绘制成折线图分析趋势，可以通过notepa++工具进行列操作整理成如下样式；（因为grep命令进程的性能数据也会被脚本打印出来，所以需要通过搜索选项中的“在当前文件中查找”mobile行筛选出当前应用进程的性能数据；再用Notepad++列操作删除不需要的列，列操作只要按住Alt键后再选中相应不需要的列进行删除，对齐需要的列并复制到Excel中即可）

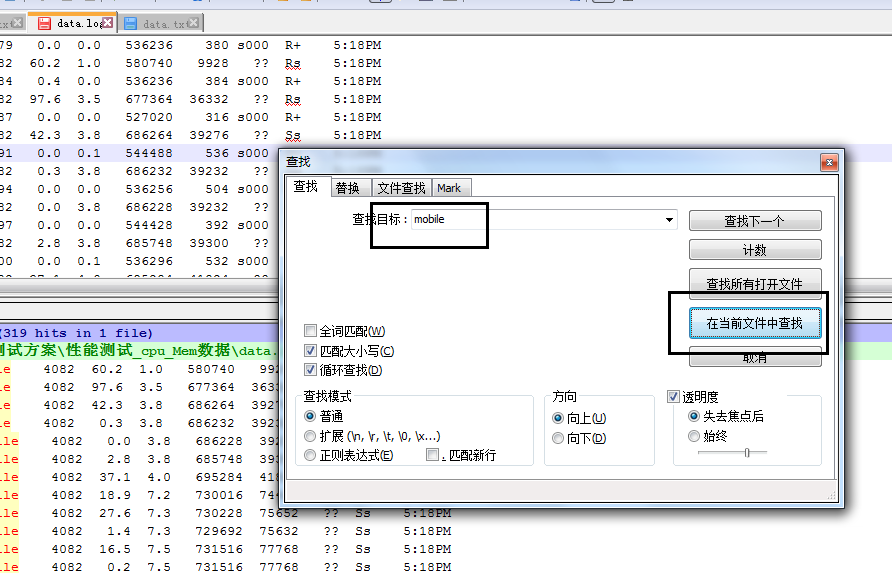
先通过列操作删除COMMAND再进行行筛选进程操作（这很重要，不然会有重复的数据），接下来我一步步用截图说明比较直观（这些都是notepad++工具的基本使用技巧，可以在网上找更多的教程学习）

1、通过列删除操作删除COMMAND字段的列值：

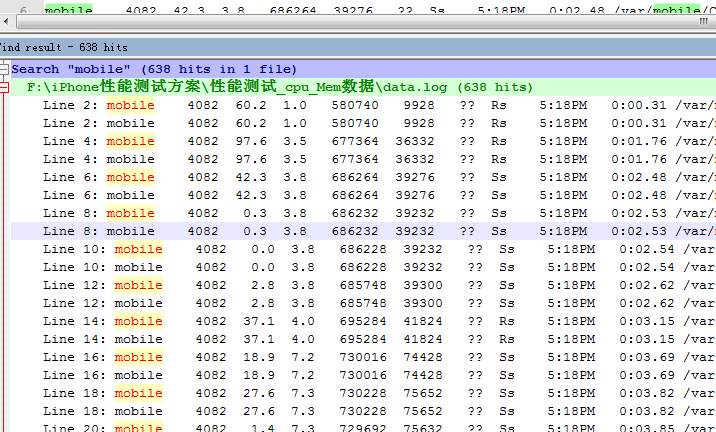




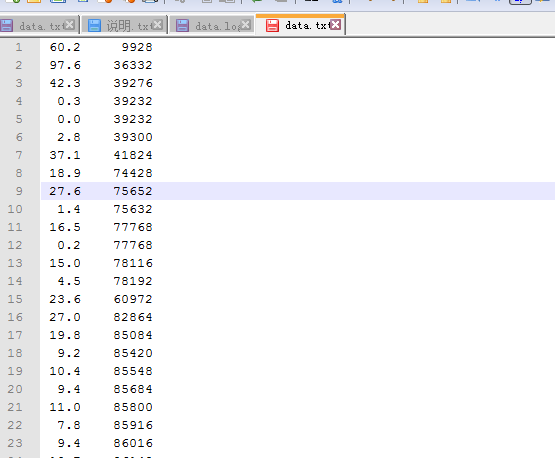
2、删除之后再使用mobile作为关键字在文件中筛选出Mobile的行：



3、再将查找的结果复制出来放到新的文件中进行一系列的列删除操作（这里就不详细介绍了）；



4、进行一系列的列删除操作，整理后的结果如下图所示（注意列值的对齐，不要多删除了字符）；



**将整理后的统计日志汇总至Excel中：**

上述脚本使用的时间间隔为1秒（sleep 1），所以在转制成Excel时可以添加一行时间列方便绘制以时间为维度的统计视图，如下为Excel和统计视图样式（见附件性能测试数据分析.xlsx）：

|  |  |  |
| --- | --- | --- |
| Time(s) | CPU% | RSS |
| 1 | 60.2 | 9.928 |
| 2 | 97.6 | 36.332 |
| 3 | 42.3 | 39.276 |
| 4 | 0.3 | 39.232 |
| 5 | 0 | 39.232 |
| 6 | 2.8 | 39.3 |
| 7 | 37.1 | 41.824 |
| 8 | 18.9 | 74.428 |
| 9 | 27.6 | 75.652 |
| 10 | 1.4 | 75.632 |
| 11 | 16.5 | 77.768 |
| 12 | 0.2 | 77.768 |
| 13 | 15 | 78.116 |
| 14 | 4.5 | 78.192 |
| 15 | 23.6 | 60.972 |
| 16 | 27 | 82.864 |
| 17 | 19.8 | 85.084 |

………..

可以将Excel绘制成图标更方便查看；

### KTV线上包房唱歌实例分析

**测试场景：客户端启动->进入包房中演唱歌曲->歌曲演唱过程中释放道具->结束演唱展示默认场景；**

**CPU和内存统计视图：**

如上图所示，红色线标出的是内存的占用趋势，从图中可以看出内存一直处于增长状态（此时客户端在唱歌过程中有持续的音频数据处理，内存随时持续间增长），内存在增长到最高峰是迅速下降（客户端唱歌结束释放了大部分内存资源），下降后有个迅速的上升（结束后切换默认场景，默认场景存在较多高清滚动图片导致需要分配更多内存），但是结束后内存下降至80Mb左右并保持平稳不变；重复唱歌会形成一个内存上升下降再上升的周期性变化趋势，总体保持平稳；

图中蓝色线标出的是cpu占用的趋势，图中cpu占用总体趋势在5%-20%之间（这个期间客户端进行了音频的编码和实时数据网络发送，所以持续性占用cpu），正常情况下cpu应该如264s之后（音频编码和网络数据发送结束）长期保持平稳态势，偶尔的网络请求或者界面动画导致cpu瞬间上升；cpu占用出现过四次高峰，第一次是客户端启动的时候，此时加载应用程序占用cpu会瞬间上升（瞬间的cpu高值一般无大影响）；后面三次是客户端展现道具动画（道具动画播放使用cpu非常高），但是维持的时间较短不影响性能，但是如果动画比较频繁可能需要考虑cpu方面的性能，长期的cpu占用高会导致电量消耗快和手机发热；

### iPhone客户端CPU、内存测试执行策略

执行性能测试时最好使用Release版本客户端（debug版本在内存和cpu使用上存在一定的限制），屏蔽不必要的日志输出，总的来讲就是保证客户端产品同最终用户使用的尽量一致；

* **内存性能测试执行策略**

因为客户端加载一个界面需要申请一定的内存，所以内存占用过高主要影响客户端加载界面响应速度；非一级界面在离开此页面时正常情况下需要释放所有申请的内存，即内存水平回到进入此界面之前的水平（实际情况会存在较少部分不能完全回收），如果内存完全没有下降则说明没有及时回收不必要的内存；进一步可以重复加载和离开此界面步骤查看内存是否重复增加，以此判断是否存在内存泄露，如果每次操作带来相同大小的持续增长则说明存在内存泄漏；

性能测试初期可以逐个界面采取上述步骤排查内存泄漏情况，性能测试稳定后可以考虑从用户最常用的使用场景进行组合式测试，测试过程中可不必观察内存情况，用户场景集成测试完成后可以将日志中的测试数据汇总生成Excel统计表格进行分析；

* **CPU性能测试执行策略**

CPU占用过高可能导致客户端发热和耗电量增加，cpu使用率在界面跳转、网络请求、音频播放时都会出现升高的情况，cpu占用率瞬间升高比较正常，一般来说cpu的占用率不能长期保持较高水平；如音频播放、定时器、网络数据请求和图片加载等都会增加cpu使用率，所以测试过程中可以侧重需要持续播放解码音频、请求网络、大量加载图片的功能点进行性能测试；没有持续消耗资源的功能点的客户端无需进行针对性测试，可以结合用户使用场景进行回归测试，测试完成后分析日志cpu的峰值和平均值，峰值无绝对要求，平均值依据不同客户端，一般不涉及持续视频音频播放、网络处理、编解码的客户端不应该超过10%；

## 电量、流量消耗性能测试



### 电量、流量消耗测试方法

* **电量消耗**：

电量消耗方面的计算没有什么特别好的方法和工具，比较直接的就是通过手机电池消耗的百分比来统计消耗情况(测试过程中注意排除屏幕亮度、功放大小、其他进程对电量测试的干扰)；统计电量较为方便的就是开发通过系统自带方法检测电池百分比占用并输出到日志中来监控电池消耗情况；

获取系统电量百分比的源码如下（源码来源于网络，欢乐KTV已经集成）：

1.注册电量的观察者，监控电量

 UIDevice \*device = [UIDevice currentDevice];

 device.batteryMonitoringEnabled = YES;

currentCharge.text = [NSString stringWithFormat:@"%.2f", device.batteryLevel];

  [device addObserver:self forKeyPath:@"batteryLevel" options:0x0 context:nil];

2.电量有变化时，就会得到调用，device.batteryLevel就是当前的电量

- (void)observeValueForKeyPath:(NSString \*)keyPath ofObject:(id)object change:(NSDictionary \*)change context:(void \*)context {

UIDevice \*device = [UIDevice currentDevice];

if ([object isEqual:device] && [keyPath isEqual:@"batteryLevel"]) {

currentCharge.text = [NSString stringWithFormat:@"%.2f", device.batteryLevel];

}

}

* **流量消耗**：

流量的消耗可以通过安装越狱插件Datameter（此软件是收费软件，可以在越狱应用商店下载破解版）来计算实时流量和流量消耗总计，如下图所示

Datameter插件安装：首先你的设备必须是已[越狱](http://iphone.91.com/jailbreak/" \t "_blank)的 (查看越狱教程)，先开启 Cydia，点击下方【管理】—【软件源】—右上角【编辑】—左上角【添加】—添加新源地址http://apt.feng.com/banxia—【添加源】，新源地址添加完后，在Cydia底部选【搜索】，然后在搜索框中输入Datameter ；

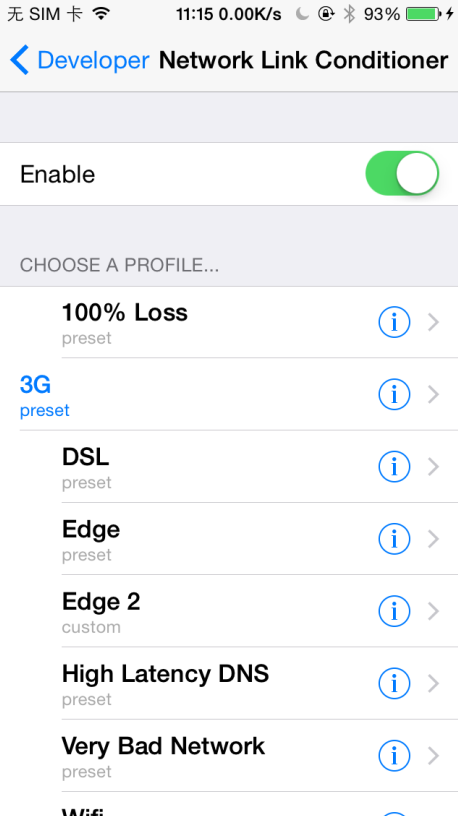
安装插件成功后再设置中会增加一栏DataMeter栏目，里面可以设置顶部状态栏显示实时流量数据（如下图顶部所示），wifi选择中会显示一段时间内使用的流量数据；



Ps：也可以使用360热点wifi等热点工具来统计使用的流量；

* **网络兼容性测试技巧**：

其中苹果为开发者提供了一种自带的网络环境模拟插件，在xcode中开启开发者模式即可在设置中看到，如下所示；



将Enable开关打开，选择不同的网络配置即可模拟上传和下载速度，以及网络丢包率等网络参数，此插件还支持自定义网络带宽和丢包率参数配置；

### 电量、流量消耗测试方法

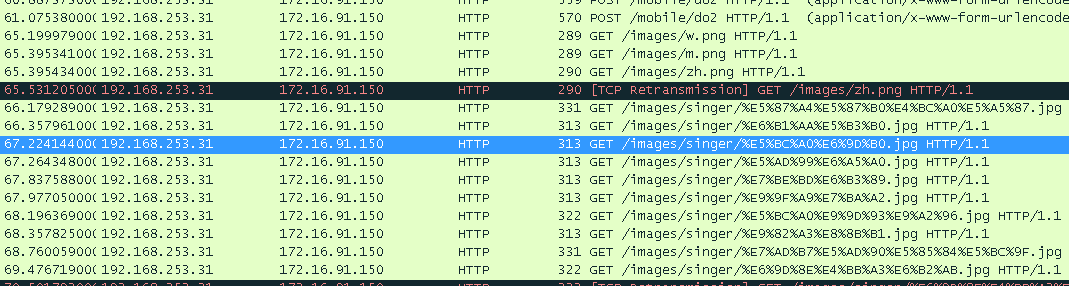
* **电量执行策略**：

一般来说电量的消耗主要来源于网络、图像展示、cpu耗电，所以耗电只是这些参数的直观展示，在做好上述参数性能测试基础上其实没有必要过于关心耗电量的测试，某个功能点的耗电测试没有什么意义，只要按照用户正常的使用场景通过日志统计耗电即可；

* **流量执行策略**：

尽管现在wifi已经较为普遍，但是我们客户端还是不能放弃那些使用数据流量的用户；数据流量都是按量收费的，所以客户端应该最大可能的节省流量；流量的消耗源头主要有接口请求、图片加载、在线音频播放；其中接口请求往往是不可减少的，但是需要做到减少不必要的接口请求（比如个人用户数据、一些一级界面榜单数据等），不但可以减少流量消耗还可以减轻接口压力；测试是否重复请求了不必要的接口可以从客户端请求日志方面查看（比较简单），也可以通过wireshark抓包检查；

图片加载一般来讲是需要本地缓存的，相同链接的图片在缓存目录未满的情况下是不会再次请求的（判断一个图片是否重新下载可以使用wireshark抓包工具查看是否重复出现相同链接的图片http请求）



，图片缓存一般来讲都是使用第三方的插件，出现问题的可能性极小。音频缓存必要性不是非常大，一般根据客户端的产品需求定义；所以总的来讲流量消耗方面的测试和电量差不多，按照用户正常的使用场景进行测试，最终根据数据统计给出一个测试统计结果即可（流量一般可以优化的余地不多，主要可以从接口返回的图片尺寸大小、接口请求和返回报文长度方面做优化）；

## 响应时间

产品对客户端响应时间无明确要求的可不进行此项测试，此项测试比较繁琐且耗时，测试数据的误差和操作执行的细致程度及方法有很大关系（视频录制测试响应时间方面的测试目前只有移动互联测试输入法团队使用过），因为实际参与的客户端项目没有这方面的测试经验，所以这里只提供一个可行的比较便捷的测试方法。



### 响应时间测试方法

因为客户端界面展现和跳转几乎是毫秒级的，所以要计算客户端响应时间比较麻烦，目前常用的方法就是将手机上的操作录制成视频，然后使用专用的视频软件逐帧进行分析，苹果手机自带一种视频投影服务AirPlay，可以使用itools苹果录屏大师插件录制手机上的操作录像（录制视频响应比较及时），方法如下：

1、PC端（windows系统）安装itools（经典版）、iTunes、360无线热点，手机连接电脑发出的热点wifi（AirPlay仅支持内网，笔记本可以直接在网上下载360wifi热点软件，台式机需要使用360小热点工具）；

2、启动itools工具，工具箱中打开“苹果录屏大师”工具；



3、手机端从底部滑出快捷栏选择AirPlay指定的设备，再打开镜像选项点击完成即可；





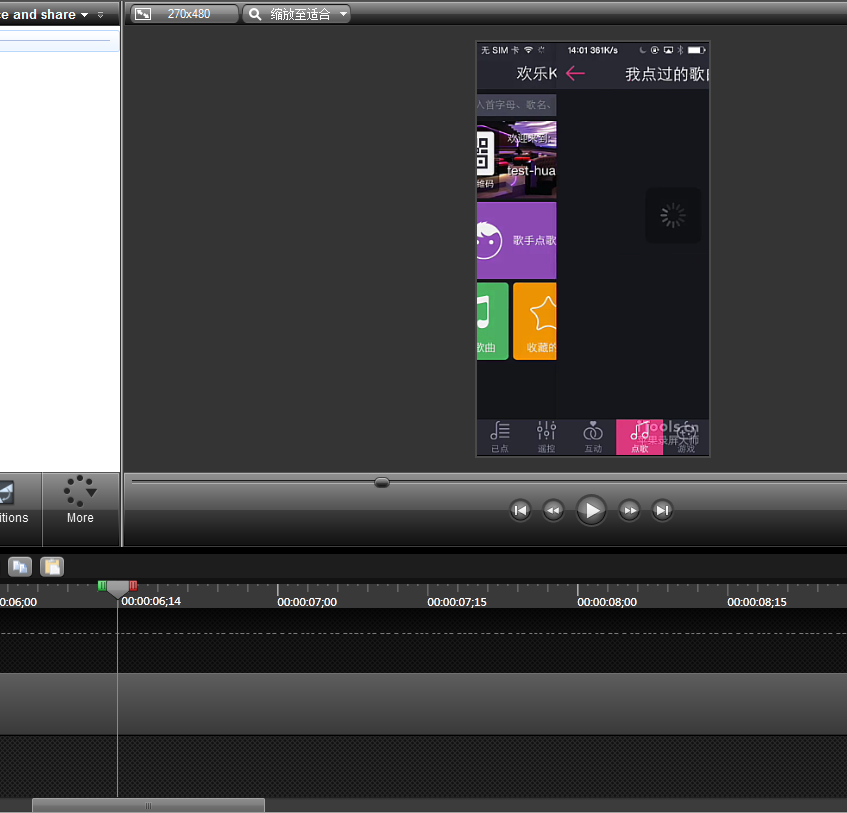
4、itools苹果录屏大师上启用录制（如下图圈出区域），录制完成后点击停止将自动保存录制视频；



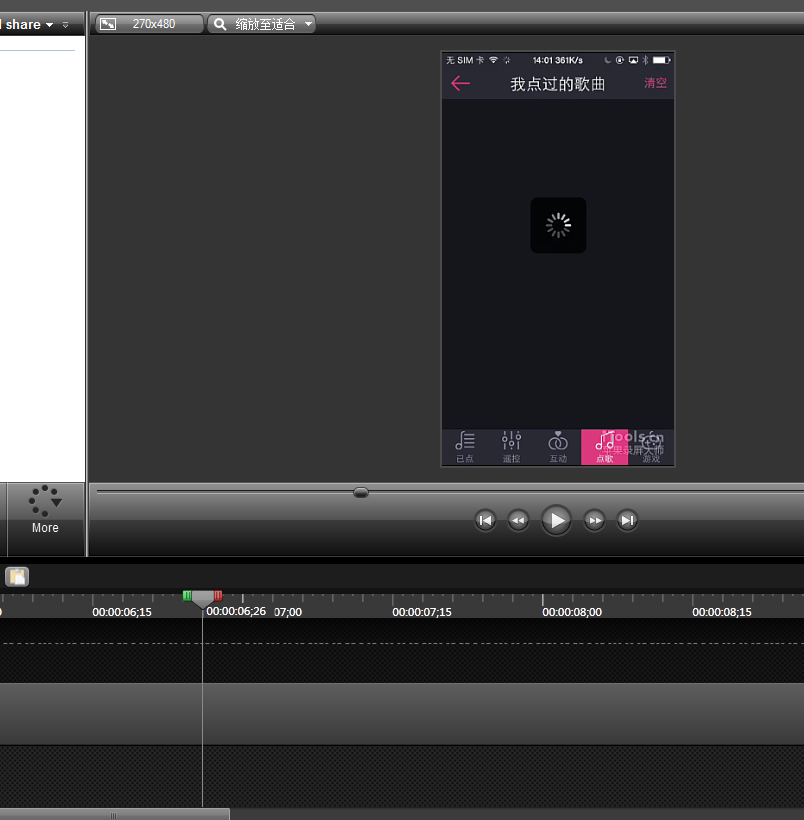
5、录制完成的视频再使用Camtasia Studio 8分析，将视频拖到轨道中并放大视频轨道的长度，然后轻微移动视频时间轴查看界面切换前后花费的时间，放大后的视频轨道将1秒分割成30个单位，所以最小单位为1/30s；

例如，如下图所示为一个界面从展现到结果加载的过程耗时计算：

界面切换：时间点为6:14

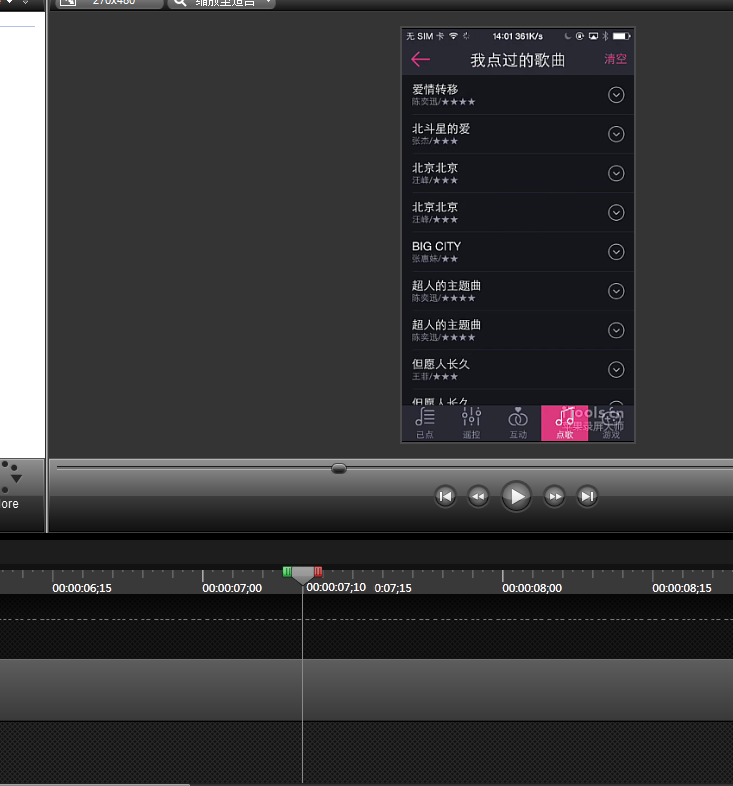


接口请求过程中：时间为6:26



操作方法简单说就是暂停视频播放，然后将视频轨道放大到最小单位，然后慢慢拖动指针观察视频画面切换，从一个画面开始为起点记录下指针位置时间，到下一个画面结束记录指针上时间，计算两个时间差即可得出响应时间；

数据加载完成：时间为7:10



所以此界面从加载到结束总耗时为（1s为30个刻度）T1=7:10-6:14=26\*1/30s≈0.87s，界面跳转时间T2=6:26-6:14=12\*1/30s≈0.4s,接口请求及加载时间T3≈0.47s；

### 响应时间测试方法

界面响应时间测试主要侧重于客户端层面复杂界面展现耗时计算，也可以用于集成测试接口响应速度（此方法较为繁琐，建议通过查看日志输出的方法计算）；客户端层面需要进行响应时间计算的功能点大多集中于动画展示、比较消耗cpu和内存的界面展现需要使用上述方法重点测试下响应时间，普通的界面不需要采取视频分析，人为感知界面响应时间即可，如果人为感觉较慢的界面再采取上述方法找到准确的数值以作修改的参考；

# 附录.IOS客户端输出日志查看

IOS平台没有像Android平台使用Eclipse查看日志那么方便，以前常用的itools查看实时日志的方式会产生很多无用的系统级日志，在日志输出比较频繁的情况不利于查看和检索；越狱的手机可以通过USB线连接手机，在使用itools工具打开手机的ssh隧道（步骤同上述性能测试连接方法一样）通过SSH工具登录root账户，再使用tail命令实时监控一个日志文件的输出，SSH工具查看日志的优势在于支持多样性的筛选和查看方法，且会屏蔽不需要的系统级日志，大大提高了测试的执行效率，推荐大家使用；

## 日志打印方法

开发只需要在工程中添加如下代码将Nslog日志追加到Document中的test.log中即可（如有疑问可以联系智能音箱和欢乐KTV相关开发询问）；

- (void)redirectNSLogToDocumentFolder{

    NSString \*fileName = @"test.log";

    NSString \*logFilePath = [DOCUMENT\_PATHstringByAppendingPathComponent:fileName];

    freopen([logFilePath cStringUsingEncoding:NSASCIIStringEncoding],"a+",stderr);

}

- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions

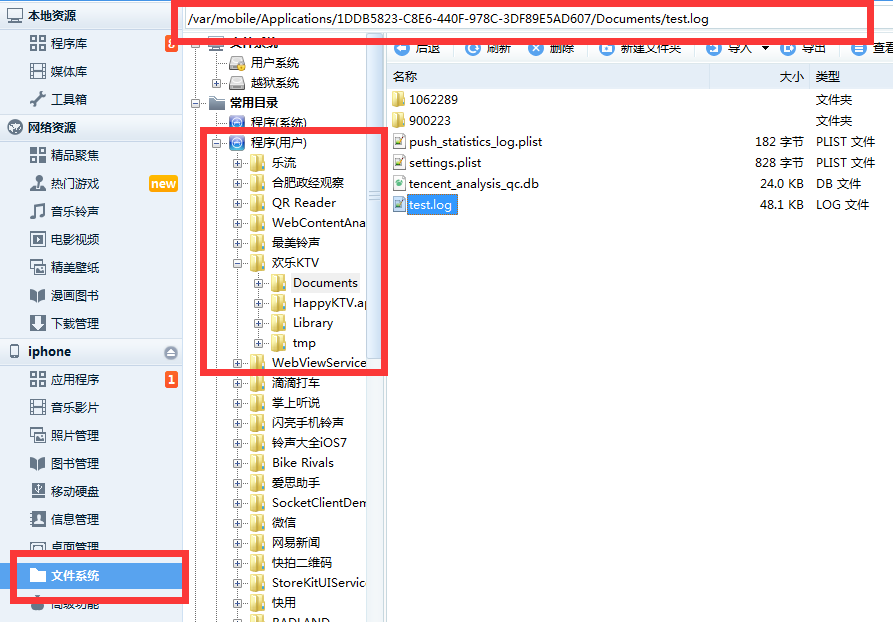
{

     //重定向日志输出

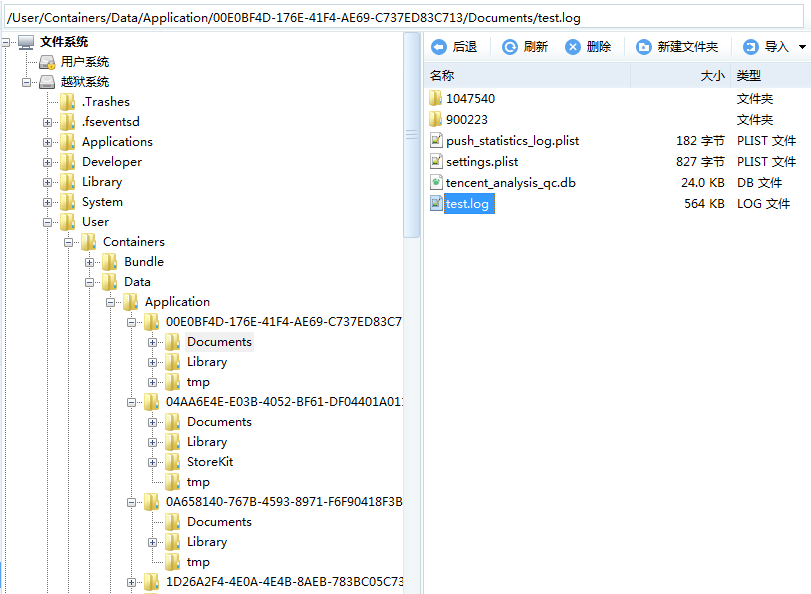
        [self redirectNSLogToDocumentFolder];

     }

使用itools工具查看日志文件存放的绝对路径，ios7及之前的系统版本可以使用如下方法在itools文件系统中找到日志文件的绝对路径，如下图



Ios8暂时需要在越狱系统/User/Containers/Data/Application/中找到相应的应用，可以按照应用安装时间查找相应的应用目录（itools越狱目录目前暂不支持ios8应用名称匹配）；



绝对路径：/var/mobile/Applications/1DDB5823-C8E6-440F-978C-3DF89E5AD6

07/Documents/test.log

## 日志查看方法

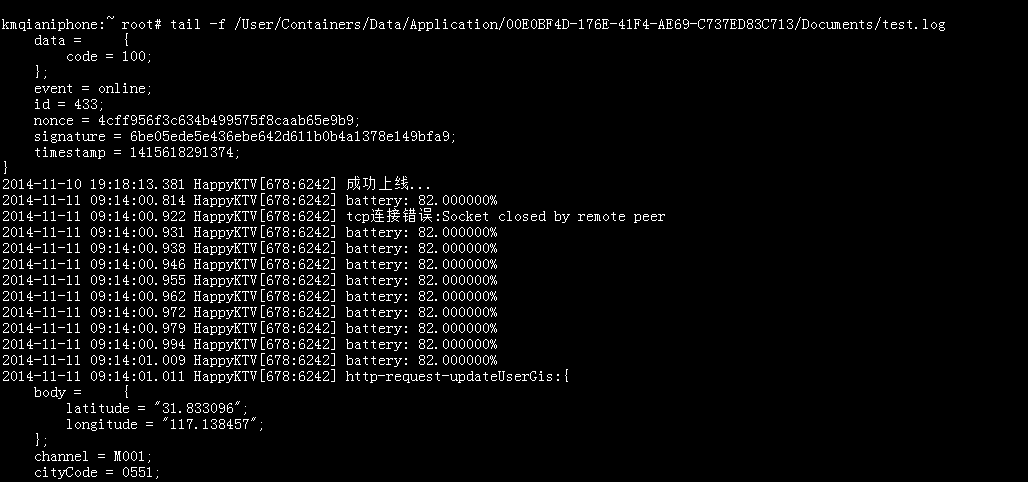


### Tail命令实时监控文本日志

使用putty或者xshell等ssh工具（建议使用Xshel工具，此工具查看比较方便，连接方法和连接一般的linux服务器一样）连接手机并登录root账户，使用tail命令可以实时监控日志输出，例如：

tail -f /var/mobile/Applications/1DDB5823-C8E6-440F-978C-3DF89E5AD60

7/Documents/test.log（日志路径）

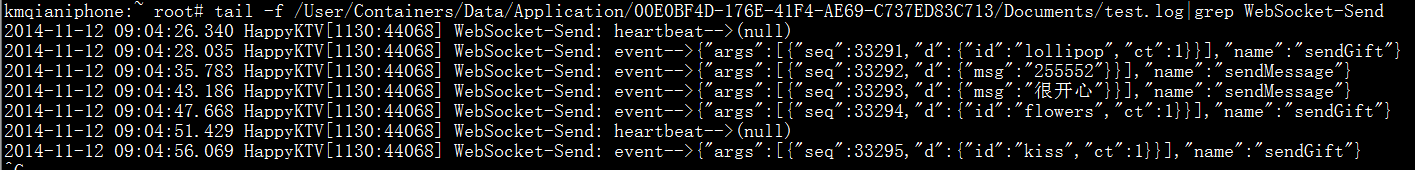


### Tail+grep命令实时筛选并查看文本日志

tail命令还可以配合grep使用更加方便（在日志输出比较频繁和定位指定问题时比较方便）,可以实时监控输出指定关键字日志，比如只想查看发送和接收消息相关的日志，例如：tail –f /var/mobile/Applications/1DDB58

23-C8E6-440F-978C-3DF89E5AD607/Documents/test.log|grep WebSocket-Send (发送消息的关键字)，如下图所示

如下为筛选node发送消息的日志（此处为了方便筛选可让开发在打印日志时指定不同的关键字开头，如Http>>可以指定为接口请求类型的关键字，开发打印日志时最好不换行输出）



下图为接收Node返回消息的日志（同一类型的接口请求和返回可以使用相同的关键字，方便筛选后查看）

