battery

- https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/index.html#//apple_ref/doc/uid/TP40015243-CH3-SW1
- https://help.apple.com/instruments/mac/9.3/index.html?localePath=en.lproj#/dev03a7149d
- https://download.developer.apple.com/iOS/iOS_Logs/Battery_Life_Logging_Instructions.pdf
- http://www.cocoachina.com/ios/20171205/21428.html

> app消耗的硬件资源的越多，系统工作越繁重，设备的温度就会逐渐上升。这时系统会通过一些措施降低设备温度。

- CPU

> CPU是电能消耗大户，高CPU使用量会迅速消耗掉用户的电池电量。app做的每件事几乎都需要用CPU，所以使用CPU要精打细算，真正有需要时通过分批、定时、有序地执行任务。

- 设备唤醒

> iOS设备通过睡眠来节能。只要设备被唤醒，屏幕和其他的硬件资源就必须通电，会产生很高的间接功耗。如非必须，app要尽量保持闲置，不要推送消息或用其他方式唤醒设备，特别是app在后台的时候。

- **网络操作**

> 大多数app都需要网络操作。网络通信时，蜂窝数据和Wi-Fi等元器件开始工作就会消耗电能。分批传输、减少传输、压缩数据、恰当地处理错误，你的app省电效果会很显著。

- **图像、动画、视频**

> app内容每次更新到屏幕上都需要消耗电能处理像素信息。动画和视频格外耗电。不经意的或者不必要的内容更新同样会消耗电能，所以UI不可见时，应该避免更新其内容。

- **位置**

> 很多app为了记录用户的活动或者提供基于位置的服务会进行定位。定位精度越高，定位时间越长，消耗电量也就越多。所以app应该尽量降低定位精度、缩短定位时间。不需要位置信息之后立即停止定位。

- **动作传感器**

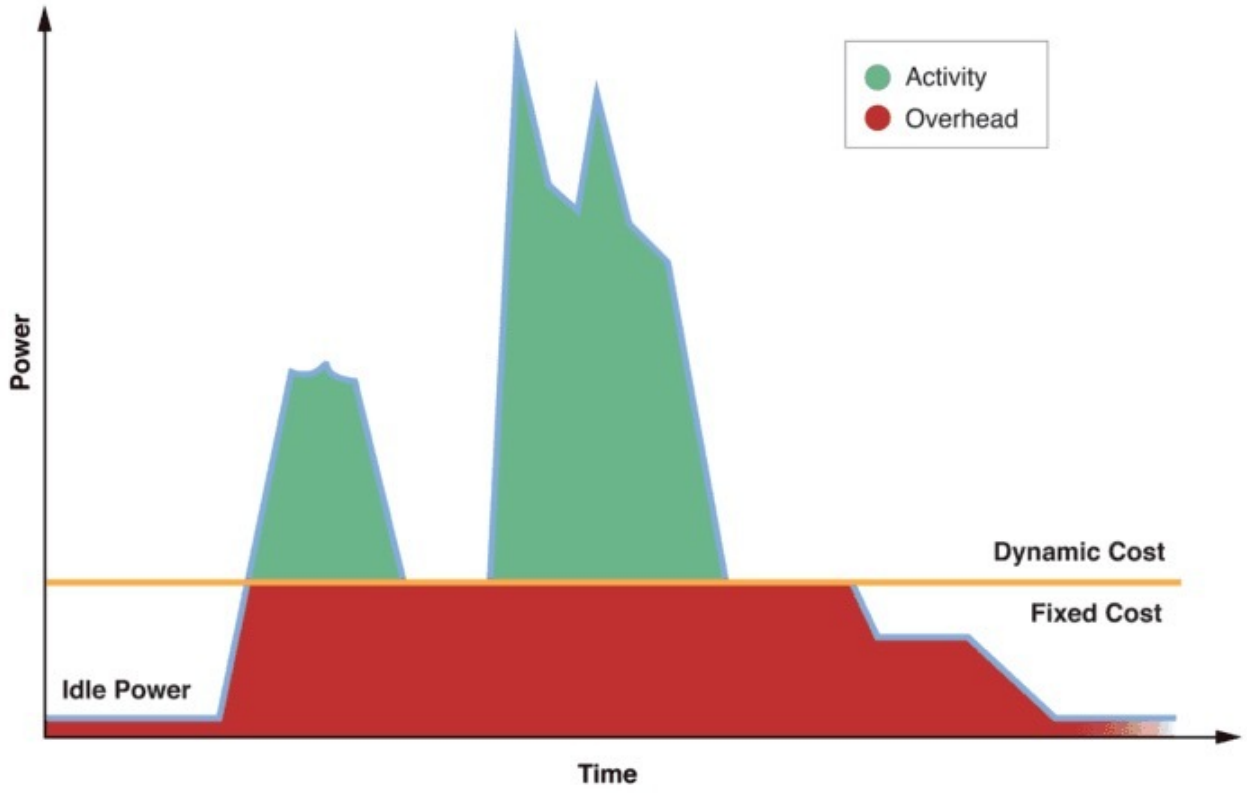
> 长时间用不上加速度计、陀螺仪、磁力计等设备的动作数据时，应该停止更新数据，不然也会浪费电能。应按需获取，用完即停。

- **蓝牙**

> 蓝牙活动频度太高会消耗电能，应该尽量分批、减少数据轮询等操作

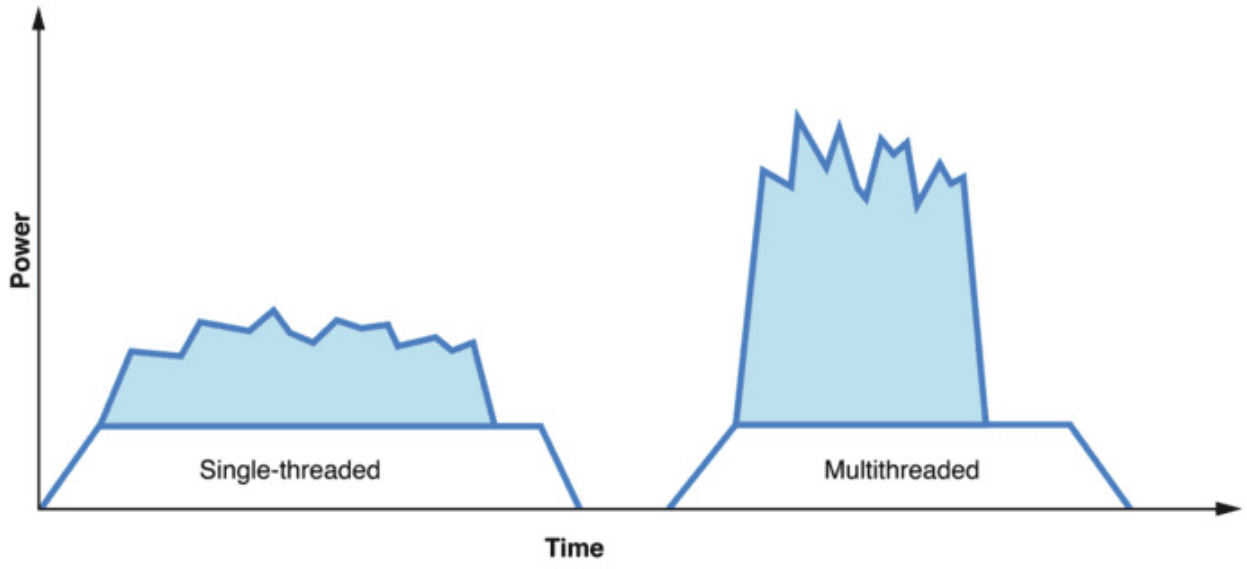
CPU

我们app执行任务有动态能耗(dynamic cost)和固定能耗(fixed cost)。



为减少固定能耗交换动态能耗

> 你的app可以通过分批执行、降低执行频率来避免产生零散的任务。例如，不采用同一个线程串行执行一系列任务，而是把任务同时放到多个线程，如图2所示。每次使用CPU，内存、缓存、总线等都得通电。通过分批执行，组件可以只上一次，使用时间也更短。



减少后台工作

> 实现UIApplicationDelegate中的方法，应用进入后台前做好暂停任务，保存数据等工作。如果确实需要完成用户执行的一些任务，应该调用UIApplicationDelegate中的beginBackgroundTaskWithExpirationHandler: 方法，这样后台任务可以继续执行几分钟。任务执行完毕后一定要调用endBackgroundTask:方法，不要等着系统强行挂起进程。

用QoS分级有序工作

> 多个app和众多操作需要共享CPU、缓存、网络等资源，为了保持高效，系统需要根据不同任务的优先级智能地管理这些工作。

少使用定时器

> app经常滥用定时器。想一下你app中的定时器，是否真的有必要存在。抛开具体场景不说，如果定时器触发太频繁，能耗影响是比较大的。

- 用事件通知代替定时器。

> 如果一定要用定时器，尽量高效地使用，可以参照下列指导方针：
>
> \1. 设置一个合适的超时时间。
>
> \2. 不再需要时及时关闭重复性定时器。
>
> \3. 设置触发公差。

优化I/O访问

> app每次执行I/O任务，比如写文件，会导致系统退出闲置模式。而且写入缓存格外耗电。通过下列方法可以提高能效、改善app性能。

> \1. 减小写入数据。数据有变化再写文件，尽量把多个更改攒到一起一次性写入。如果只有几个字节的数据改变，不要把整个文件重新写入一次。如果你的app经常要修改大文件里很少的内容，可以考虑用数据库存储这些数据。
>
> \2. 避免访问存储频度太高。如果app要存储状态信息，要等到状态信息有变化时再写入。尽量分批修改，不要频繁地写入这些小变动。
>
> \3. 尽量顺序读写数据。在文件中跳转位置会消耗一些时间。
>
> \4. 尽量从文件读写大数据块，一次读取太多数据可能会引发一些问题。比如，读取一个32M文件的全部内容可能会在读取完成前触发内容分页。
>
> \5. 读写大量重要数据时，考虑用dispatch_io，其提供了基于GCD的异步操作文件I/O的API。用dispatch_io系统会优化磁盘访问。
>
> \6. 如果你的数据由随机访问的结构化内容组成，建议将其存储在数据库中，可以使用SQLite或Core Data访问。特别是需要操作的内容可能增长到超过几兆的时候。
>
> \7. 了解系统如何缓存文件、如何优化缓存的使用。如果你不打算多次引用某些数据，不要自己缓存数据。

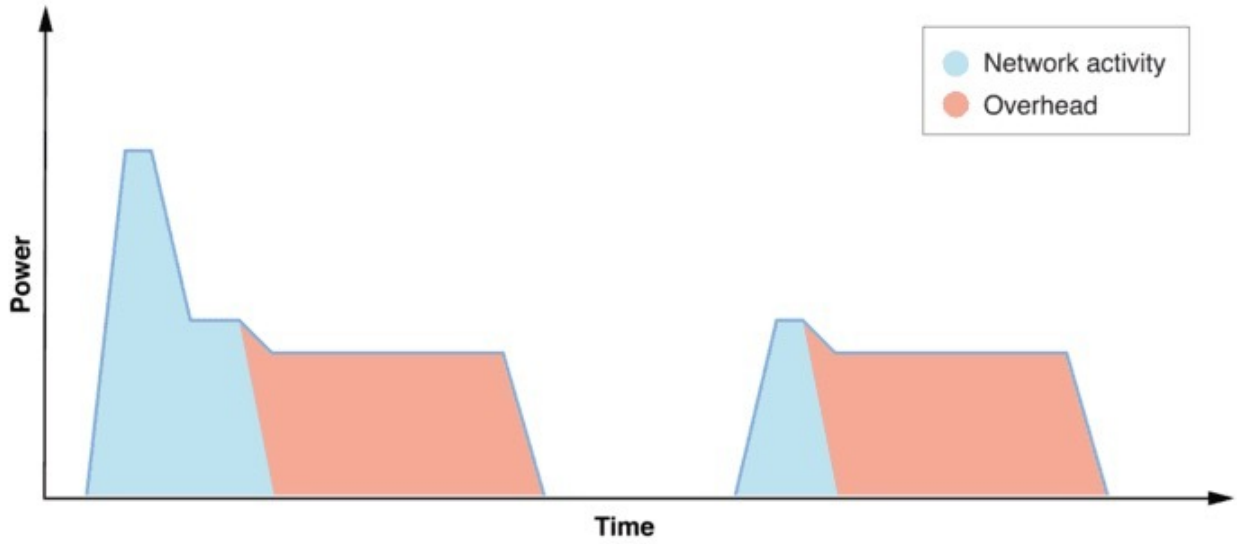
低电量模式

> iOS9之后，iPhone增加了低电量模式，用户如果希望延长iPhone电池的寿命，可以在设置 > 电池中开启该功能。开启该功能之后iOS会采取一些措施，比如：
>
> \1. 降低CPU和GPU性能
>
> \2. 暂停随意的和后台的活动，包括网络
>
> \3. 降低屏幕亮度
>
> \4. 缩短自动锁屏时间
>
> \5. 关闭邮件刷新
>
> \6. 关闭视角缩放
>
> \7. 关闭动态壁纸

> 你的app也应该做一些事情帮助系统节省电能，比如，可以减少动画、降低帧率、停止位置更新、关闭同步和备份功能等等。可以通过向NSNotificationCenter注册NSProcessInfoPowerStateDidChangeNotification通知监听低电量模式状态。

网络操作

> 只要app一执行网络操作，就会产生大量间接能耗(overhead cost)。网络硬件，比如蜂窝数据和Wi-Fi电路，为了省电默认是不通电的。为了执行网络操作，这些资源必须通电，之后为了等待接下来可能出现的任务，它们在操作完成后会继续保持一段时间的活跃。零散的网络传输会导致很高的间接能耗，迅速消耗电池电量，如图3所示



缩减网络请求

> \1. 减少、压缩网络数据。可以降低上传或下载的多媒体内容质量和尺寸等。
>
> \2. 使用缓存，不要重复下载相同的数据。
>
> \3. 使用断点续传，否则网络不稳定时可能多次传输相同的内容。
>
> \4. 网络不可用时不要尝试执行网络请求，尽量只在Wi-Fi情况下联网。

\5. 让用户可以取消长时间运行或者速度很慢的网络操作，设置合适的超时时间。

\6. 网络请求失败后用SCNetworkReachability的通知监测网络状态，网络可用后再重试

延迟联网

分批传输。比如，下载视频流时，不要传输很小的数据包，直接下载整个文件或者一大块一大块地下载。如果提供广告，一次性多下载一些，然后再慢慢展示。如果要从服务器下载电子邮件，一次下载多条，不要一条一条地下载。

网络操作能推迟就推迟。如果通过HTTP上传、下载数据，建议使用NSURLSession中的后台会话，这样系统可以针对整个设备所有的网络操作优化功耗。将可以推迟的操作尽量推迟到设备充电状态并且连接Wi-Fi时进行，比如同步和备份工作。

图像、动画、视频

以下列指导方针优化内容更新：

\1. 减少app使用的视图数量。

\2. 减少不透明视图的使用，比如视图上显示一个半透明模糊效果。如果要用不透明效果，避免用在内容频繁变化的地方。另外，由于内容变化后背景视图和半透明视图必须同时改变，这也会放大功耗。

\3. 避免绘制不可见的内容，比如app的内容被其他视图遮挡、被剪切(clipped)或者出画了。

\4. 动画尽可能用较低的帧率。比如，高帧率在玩游戏时有意义，但是菜单画面可能较低的帧率就够了。只有对用户体验有影响时才使用高帧率。

\5. 执行动画时不要修改帧率。比如，你的app帧率是60fps，整个动画就保持这个帧率不要变。

\6. 避免同时在屏幕上使用多种帧率。比如，你的游戏人物是60fps，天上的云彩移动又是30fps，不要出现这种状况，就算提高其中某一个的帧率，也要用相同的帧率。

\7. 开发游戏时使用推荐的framework。这些framework针对性能和功耗是做过优化的：2D游戏用SpriteKit、3D游戏用SceneKit、画面非常逼真的游戏用Mietal。

全屏播放视频时iOS可以通过高效管理各种资源来优化能耗，但是在视频上下额外添加图层会影响功耗优化效果。app尽量不要在全屏视频上添加额外的图层（即使是隐藏的图层）。如果用户有需要，可以通过比如单击这样的方式来显示播放控制之类的UI，不需要了以后应该把这些图层移除掉

优化定位和动作(Motion)

错误使用定位会阻碍设备进入睡眠模式，让定位硬件部分持续通电而消耗电池电量，这会使用户体验变的很差，下面来看一下如何针对功耗优化定位服务。

如果你的app只是需要快速确定一下用户的位置，最好用CLLocationManager的requestLocation(iOS9引入)方法。定位完成之后会自动让硬件断电。

除了导航，大多数app不需要一直实时更新位置。需要位置服务时开启一下定位，尽量多隔一些时间再进行下次位置更新，更新完了之后马上关掉定位。除非用户在移动的交通工具里，否则不频繁地更新位置一般没多大问题。

> 尽量降低定位精度。iOS设备默认采用最高精度定位，如果你的app不是确实需要米级的位置信息，不要用最高精度(kCLLocationAccuracyBest)或10米左右的精度(kCLLocationAccuracyNearestTenMeters)。一般来说Core Location提供的精度比你设置的要好，比如你设置为3公里左右的精度，可能会收到100米左右的精度信息。
>
> 如果定位精度一直达不到设置的精度时，停止更新位置，稍后再试。
>
> 需要后台更新位置时，尽量把pausesLocationUpdatesAutomatically设为YES，如果用户不太可能移动的时候系统会自动暂停位置更新。
>
> 后台定位时延时更新位置。如果要做一个健身类的软件追踪用户徒步的距离，可以等用户移动一段距离或者过一段时间之后再更新位置，这样可以让系统优化能耗。
>
> 合理使用访问监控(visit monitoring)。访问监控允许app接收用户频繁或长时间访问的场所的进出通知，比如在家、公司或者去喜欢的咖啡馆。
>
> 尽量不要用significant-change位置服务，优先考虑用region monitoring、 visit monitoring。
>
> 用户移动、摇晃、倾斜设备时，会产生动作(motion)事件，这些事件由加速度计、陀螺仪、磁力计等硬件检测。不需要监测设备方向时停止通知。比如用户进入一个只需要竖着显示的画面，及时把方向改变通知关掉。开启动作事件前设置一个比较大的更新间隔。

优化通知

> 尽量用本地通知(local notification)，如果你的app不依赖外部数据，而是需要基于时间的通知，应该用本地通知，可以让设备的网络硬件休息一下。
>
> 远程推送有两个级别，一个是立即推送，另一个是针对功耗优化过的延时推送。如果不是真的需要即时推送，尽量使用延时推送。

优化蓝牙通信

> \1. 没有必要的时候不要扫描蓝牙外设。
>
> \2. 扫描外设时一般不要用CBCentralManagerScanOptionAllowDuplicatesKey。
>
> \3. 只查找你需要的外设服务。外设可能提供很多服务和特性(characteristic)，查找外设的时候可以指定UUID。
>
> \4. 不要轮询设备特性值，用通知监测特征值的变化。
>
> \5. 特性值不再提供通知或者不再需要通信的时候就断开连接。

用Xcode测量功耗

> Cost 和 overhead。蓝色的是CPU执行任务消耗的电量，红色的是执行你的app消耗的其他系统资源电能。
>
> CPU。灰色方块表示你的app正在使用CPU执行任务。
>
> Network。灰色方块表示你的app正在进行网络操作。
>
> Location。灰色方块表示你的app正在使用位置服务。
>
> GPU。灰色方块表示你的app正在使用GPU执行图像相关操作，比如绘图或者播放动画。

> Background。灰色方块表示你的app处于后台状态，但是让系统仍然保持唤醒状态。
>
> 和用户交互时功耗应该比用户选择一个复杂的操作时低，不交互时不应该有功耗。
>
> 使用Instruments之前应该先考虑用Xcode中的仪表检查功耗问题。

用iOS设备直接记录功耗

> 在设备上进入设置 > 开发者 > Logging.
>
> 开启功耗记录

> 设备记录完成后返回图6所示页面，点击Stop Recording.
>
> 在Instruments中选择好设备，进入Energy Log.
>
> 选择File > Import Logged Data from Device。

iOS耗电量检测与优化

https://www.jianshu.com/p/4555704f9696

代码层面

1.合理使用NSDateFormatter 和 NSCalendar这种高开销对象

性能测试表明,NSDateFormatter的性能瓶颈是由于NSDate格式到NSString格式的转化,所以把NSDateFormatter创建单例意义不大.推荐的做法是,把最常用到的日期格式做缓存.

2.不要频繁的刷新页面,能刷新1行cell最好只刷新一行,尽量不要使用reloadData.

3.选择正确的集合

NSArray,使用index来查找很快(插入和删除很慢) 字典,使用键来查找很快 NSSets,是无序的,用键查找很快,插入/删除很快

4.少用运算获得圆角,不论view.maskToBounds还是layer.clipToBounds都会有很大的资源开销,必须要用圆角的话,不如把图片本身就做成圆角

5.懒加载,不要一次性创建所有的subview,而是需要时才创建.

6.重用

可以模仿UITableView和UICollectionView,不要一次性创建所有的subview,而是需要时才创建.完成了使命,把他放入到一个可重用集合中

7.图片处理

图片与imageView相同大小,避免多余运算 可以使用整副的图片,增加应用体积,但是节省CPU 可调大小的图片,可以省去一些不必要的空间 CALayer,CoreGraphics,甚至OpenGL来绘制,消耗CPU

8.cache,cache,cache(缓存所有需要的)

服务器相应结果的缓存(图片) 复杂计算结果的缓存(UITableView的行高) 9.尽量少用透明或半透明,会产生额外的运算.

10.使用ARC减少内存失误,dealloc需要重写并对属性置为nil

11.避免庞大的xib,storyBoard,尽量使用纯代码开发

CPU层面

1.Timer的时间间隔不宜太短,满足需求即可

2.线程适量,不宜过多,不要阻塞主线程

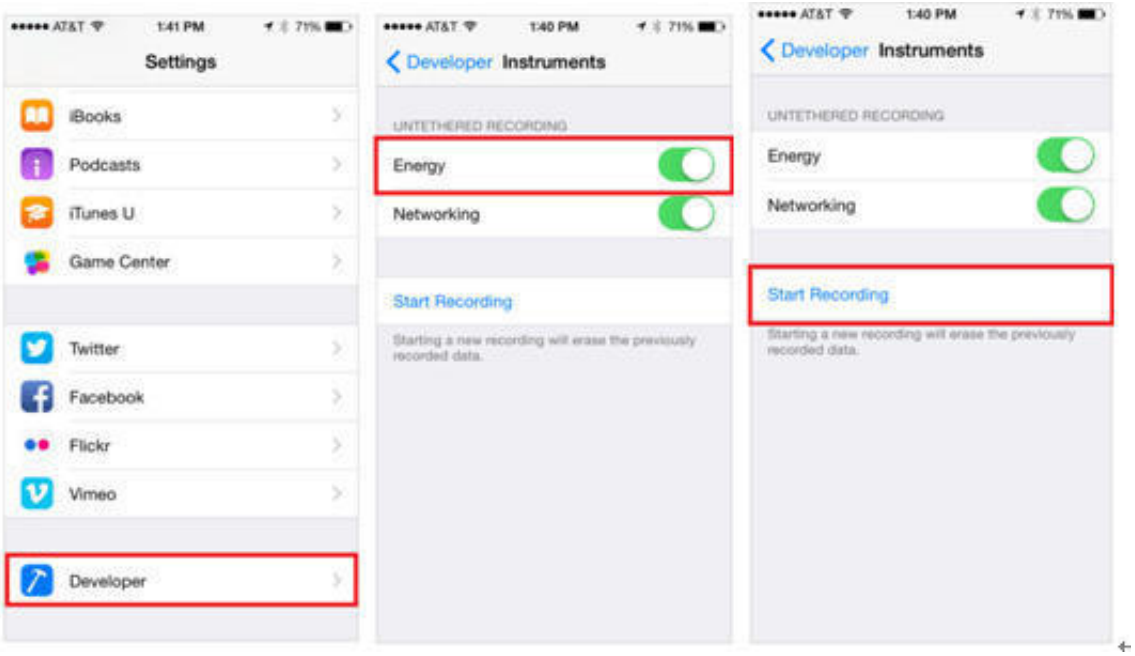
3.优化算法,减少循环次数

4.定位和蓝牙按需取用,定位之后要关闭或降低定位频率

iOS电量测试方法

https://www.jianshu.com/p/851b0acde9c3

1.iOS 设置选项 ->开发者选项－>logging －>start recording



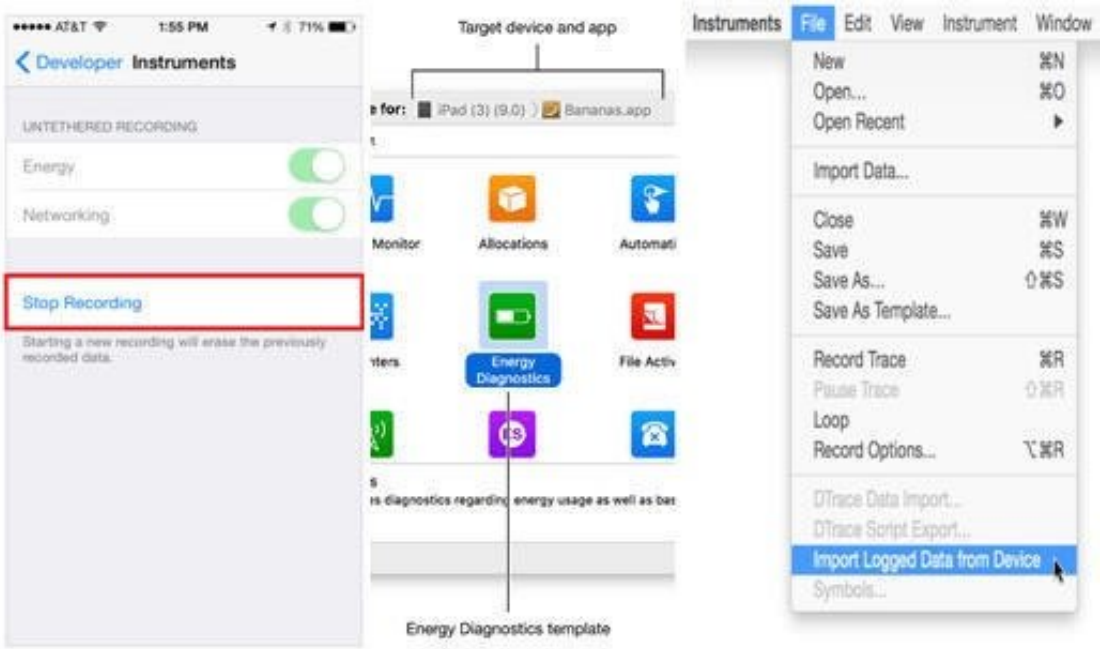
enter image description here

2.进行需要测试电量的场景操作后进入开发者选项点击stop recording

3.将iOS设备和Mac连接

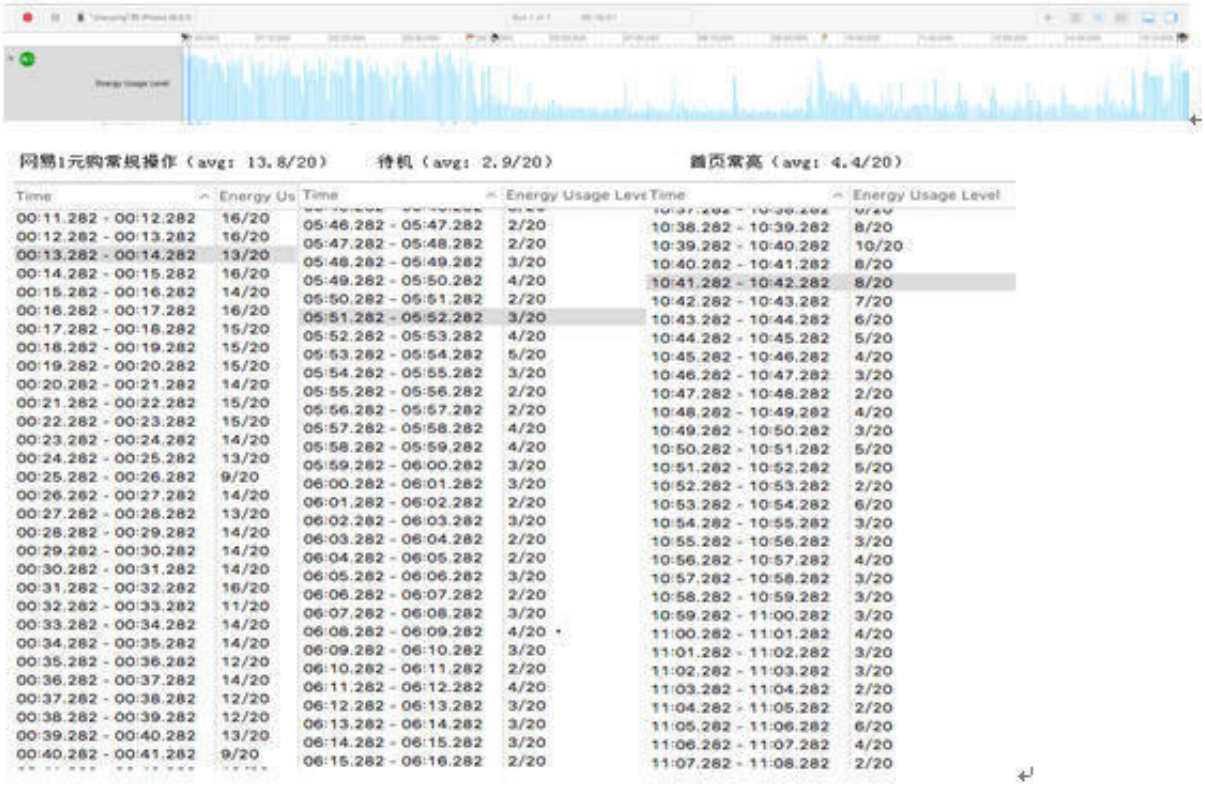
4.打开Instrument，选择Energy Diagnostics

5.选择 File > Import Logged Data from Device



enter image description here

6.保存的数据以时间轴输出到Instrument面板



enter image description here

其他

- 测试过程中要断开 iOS设备和电脑、电源的连接
- 电量使用level为0-20，1/20：表示运行该app，电池生命会有20个小时；20/20：表示运行该app，电池电量仅有1小时的生命
- 数据不能导出计算，只能手动计算平均值