主要问题

- 1、说一下哪个项目的技术点比较难。
- 2、在项目里都做了哪些卡顿的优化？（理论被断掉了，直接说自己做了哪些处理？）
- 3、卡顿是怎么做的检测？为什么还要自己写检测工具？那检测的过程中是怎么定位到具体的方法的?
- 4、主线程的卡顿是怎么检测到的？为什么要单独写一个CADisplayLink检测主线程上的帧率，如果是主线程已经卡顿的情况下岂不是会加重卡顿?
- 5、为什么要使用RunLoop做tableView卡顿优化？一开始加载数据的时候岂不是没法使用？（简历里的坑）
- 6、在项目里是如何检测Crash的，降低崩溃率都做了哪些事情?
- 7、Bugly的检测原理是什么?
- 8、是否了解直播的一些底层原理？还是说只停留在SDK的使用过程?

1、可能是因为一开始紧张，有点语无伦次，所以理论被直接断掉了。然后说实践的时候，自己都觉得这些点很微不足道，没有底气。 2、面试官估计能力很强，他一直就觉得我使用的是系统API之类的，都应该是基础的东西。没什么难的。而且做的一些实践应该是必须要做的。(总之，被鄙视了)

不过最主要的是自己对知识点里的细节没有把握清楚。主要就是CADisplayLink检测主线程、runloop为什么用来优化tableview，是怎么做的。木有说清楚。

回答

1、原则：所有这些抽象类的问题，都应该尽可能引导到自己熟悉的知识点上。其次切忌模棱两可回答一些不必要的东西。

比如说：最难的项目应该是我最新的那个项目吧，因为在那个项目上，我做了很多列表流畅度优化、卡顿检测、崩溃检测等相关的尝试。

2、原则：永远优先直接回答问题，如果是自己比较熟悉的知识点，可以在回答的过程中，稍微带过。其次再看提问者会不会追问一些更深入的东西。

预排版：一般就是将**布局计算、文本计算、文字排版等放到后台线程中先异步生成**。比如： 刚获取到JSON数据时，将Cell中要显示的控件的布局数据都在后台线程中计算并封装在一个FeedLayout里。FeedLayout可以包含的内容包括：**cell中每个控件控件CGRect、整体Cell的高度、甚至CoreText的排版结果**等。这样cell里的所有布局就都提前计算好了。但是这个方式需要考虑如果第一页数据非常多的话，可能会导致预排版的时间过长，所以可以结合滑动减速或RunLoop等一起使用。

```
//预排版类
@interface FCFFeedLayout : NSObject
@property (nonatomic, strong) FCFFeed *feed; //对应cell Model
//其他数据全是对应的CGRect
@property (nonatomic, assign) CGRect iconRect;
@property (nonatomic, assign) CGRect nameRect;
@property (nonatomic, assign) CGRect sexRect;
@property (nonatomic, assign) CGRect contentRect;
... ...
@property (nonatomic, assign) CGRect seperatorViewRect;
//cell整体高度
@property (nonatomic, assign) CGFloat height;
@end
```

预渲染：这里主要是指图片的提前异步解码。预渲染也是可以将将要显示的内容提前在后台线程中生成好，然后使用的时候直接赋值就好了。

- 同样是在TableView加载之前，就在后台线程将图片这种资源先渲染出来，尤其是要处理圆角等容易容易导致离屏渲染的属性，尽量使用Core Graphics的API代替，同时缓存到内存中，基本的实现步骤就类似SDWebImage了。
- 预渲染和预排版实际就是提前准备好文本、图片相关的绘制内容。和异步绘制类似，只是时机不一样而已。

异步绘制：预渲染是在一开始数据回来的时候做的处理。而在列表滑动过程中，对于文本和图像的绘制，则可以使用异步绘制来进行。异步绘制的实现原理就是实现CALayer的displayer方法，然后在函数里，把文本绘制或图片绘制的操作放到后台线程中进行，最后将结果返回主线程显示。 **文本绘制主要使用CoreText进行排版绘制，最后包装到Core Graphics的上下文中。图片则直接使用Core Graphics的API进行解码绘制**。

```
- (void)display{
    dispatch_async(backgroundQueue, ^{
        CGContextRef ctx = CGBitmapContextCreate(...);
        // draw in context...
        CGImageRef img = CGBitmapContextCreateImage(ctx);
        CFRelease(ctx);
        dispatch_async(mainQueue, ^{
            layer.contents = img;
        });
    });
}
```

这一套东西下来，基本能够保证一个复杂的列表达到接近50~60帧的效果了。

其次，除了列表的这些步骤，我们平时写代码的过程中，也需要注意一下其他的操作（主要是从CPU和GPU的角度）：

- 对象创建：对象创建会分配内存、调整属性、甚至读取文件，比较消耗CPU。策略就是**尽量用轻量级的对象创建**，比如CALayer代替无需操作的UIView。又比如说YYKit，单张图片时，直接给CALayer添加了一个setImageWithURL的方法，将图片直接赋值给layer.contents。其次就是**尽量**

推迟创建对象的时间，比如使用懒加载。而对于可以复用的对象，**尽可能放到缓存池复用**，比如Cell。

- 对象调整：主要是CALayer的一些属性的调整，比如frame、bounds等，它实际上是通过运行时resolveInstanceMethod为对象临时添加一个方法，然后将对应属性保存到Dictionary里，同时通知Delegate、创建动画。 其次改变CALayer的一些**可动画属性值**时，会对模型层数据做动画，最后显示在展示层上，也多了一份数据的拷贝。所以对象(尤其是视图里的对象)的调整应尽量避免，尤其是视图层次，以及频繁地添加和移除视图。
- 布局计算和渲染：尽可能使用**预排版**的方式处理布局和排版。其次尽量少调整，对于复杂的视图来说，尽可能不使用Autolayout和storyboard。
- 纹理的渲染：实际就是指图片的渲染。iOS中几乎所有的UI视图最终都是绘制成Bitmap，包括文本、图片、栅格化的内容。所以可以尽量减少短时间内大量图片的显示，尽可能将多张图片合成到一张图片中。其次对于过大的图片，比如超过GPU的最大纹理尺寸（4096*4096）,则需要CPU先预处理。所以尽量不要让图片大小超过这个值。
- 视图混合：多个视图重叠时，GPU，所以应该尽量减少视图数量与层次。不透明的视图表面opaque属性，避免无用的Alpha通道合成。
- 避免离屏渲染。
- 线程数量的控制等。

3、卡顿检测： 首先之所以写卡顿的检测工具，主要是用于测试及产品等部门使用。在线的卡顿检测则是更依赖于Bugly来做的。 首先，我的工具里对于卡顿的检测，只做了两方面：

一是FPS帧率检测。PFS是通过CADisplayerLink来计算的。主要就是在主线程创建一个DisplayerLink，然后在回调函数里计算每秒回调的次数。CADIsplayerLink是RunLoop的回调帧率，也就是每一帧会回调一次。如果列表滑动的帧率保持在50到60帧的样子，就可以算是很流畅了。如果低于24帧，则基本有肉眼可见的卡顿。但是FPS只能显示出是否卡顿，但是没法定位到具体的函数。

二就是重写一个NSThread的子类，设置一个时间间隔和最大Watch Dog时间阈值。然后重写她的main函数，在函数里开启一个while循环，然后在时间间隔内查看主线程是否能够及时处理事件，如果主线程能及时处理消息，则说明不卡顿。如果不能处理，则获取主线程的线程堆栈，然后在主线程操作信号量发出之后或者超过最大Watch Dog时间阈值，则将获取到的线程堆栈及卡顿时长进行记录，视为一次卡顿，如果超过一个watch dog阈值则可以视为一次卡顿崩溃，先暂时存起来，如果后续有主线程的信号过来，则把假定的Watch Dog崩溃日志去除。

卡顿定位：刚才说过，FPS是没法定位到具体函数的。具体方法：一是使用Instrument的Timer Profile来分析代码的执行时间，基本可以定位到具体的函数。二就是通过内核线程信息获取线程堆栈，找到对应的函数。

4、回答如3题的第二种检测方案。只能说主线程放置CADisplayerLink是一种择中的方案。 5、使用RunLoop做卡顿优化，是指将一些耗时操作，比如图片解码，放到一个队里里，在监听到RunLoop是BeforeWaiting（即将进入休眠）的时候，再开始解码操作。

6、崩溃的捕获。

首先，根据是否可捕获，我们可以将崩溃记录大致分为三种：一是语言层面的Exception，二是mach异常，这两类是可以通过方法捕获到崩溃的，另外一种无法捕获到的崩溃就是后台查杀、内存被爆、主线程卡顿超时等导致的系统级别的查杀。

语言层面的Exception：可以通过注册uncaught_exception_handler，来截获崩溃，获取到Exception的callStackSymbols（异常堆栈列表）、reason、name；

mach异常则会发出Unix标准的Signal信号。所以可以通过注册SignalHandler来截获对应的线程，获取通过线程对应的backtrace_symbols获取堆栈信息。

另外一种则需要一些特殊的方式去避免或者监听崩溃的可能发生：

内存被爆和主线程卡顿，可以通过卡顿检测的最大时间阈值来模拟判断；

后台线程也可以通过监听后台线程运行时长来模拟判断。

7、Bugly的检测原理：略。

8、

其他看到的关系卡顿和Crash相关的面试问题

- 用户报卡顿，有哪些情况，该如何定位问题?
- 卡顿检测有哪些方案？如果要监控每个函数的耗时如何实现？页面停留时间的检测该如何实现?
- 页面直接卡死，导致无法获取FPS，怎么解决这个问题?
- 为什么有时候FPS很高，但还是感觉卡顿?
- 获取FPS后，怎么定位到具体函数?
- 怎么做图片、数组、字符串的无差别存储，key怎么确定，怎么删除数据。如何保证取出的数据顺序?
- 常见的Crash有哪些？如何对这些Crash堆栈进行收集？如何捕获Crash?
- 捕获的堆栈如何进行符号还原呢？UUID是什么？怎么获取?
- dysm文件是什么，有什么作用?
- 如果你的项目中既有Bugly的Crash监控系统，又有自己的监控系统，可能存在什么问题？怎么解决?
- 针对无堆栈的Crash，比如out of memory有什么定位思路?
- 常见的野指针问题有哪些?
- dyld是什么？动态链接和静态链接的区别？dyld是什么时候执行的？dyld如何把对应dylib中函数实现链接到另一个库中的?

- FPS相关：

  FPS就是根据CADisplayerLink来计算的。CADisplayerLink是RunLoop回调的帧率，也就是每一帧会回调一次。虽然CADisplayerLink尽可能保持帧率的连续，但是如果因为卡顿出现了丢帧的情况，CADisplayerLink自然就不会被回调。所以FPS只是一个相对流畅度的提现。（注意：未完成绘制的帧会被丢掉，但是runloop并不会丢弃操作，耗时的操作依旧会被执行）

  所以最科学地**检测卡顿的方式**：直接监听RunLoop的BefoeWaiting或Exit通知的时间间隔。当然也有人通过子线程定时地ping主线程，然后根据pong的返回间隔来确定是否卡顿问题。这两个方法的原理都是一样的，就是监听RunLoop本身事件循环的周期，当前最关键的问题是时间阈值的确定。卡顿的阈值一般设置为16.7ms * 40（也就是卡了40帧左右）、卡顿崩溃的阈值大概设为3min。 对于有时候FPS很高，但是仍然卡顿，没怎么明白原因。

  如何定位：

1、开发阶段：使用Time Profile来检测有卡顿的列表，找出耗时的函数； 2、自己开发的工具里：可以在已经确定卡顿的时候，使用thread_info获取对应的线程具体信息。然后查找到对应的堆栈。

质量保障体系相关体系：

- 内存泄漏检测
- 内存大图检测
- 图片主线程解压缩检测
- 卡顿检测
- 帧率检测
- 网络性能检测
- Crash检测
- Abort检测(jetsam杀死进程、watchdog杀死进程、后台崩溃)
- 内存消耗检测
- DNS解析检测
- 启动时间检测

想要处理列表等流程度，是肯定摆脱不了YYKit和AsyncDisplayKit这两个库的 iOS保持界面流程的技巧 AsyncDisplayKit详解 FPS及具体定位 // 了解和分享Crash Report iOS Crash日志堆栈解析

iOS CALayer及UI显示原理与优化