面试总结

根据个人简历所描述信息来进行提问.
问题列表如下:

1. 如何进行性能优化,都针对那些方面使用了什么方法去解决.

我一直都是觉得App 性能优化是一个很重要的课题, 性能对于一个 App 体验来说简直是至关重要的,我把性能优化划分成几个点去针对性的进行优化,首先是 UI层面上的优化,其次是数据读写,还有内存泄漏等问题,涉及性能的知识点很多。

2. 如何作的应用安全防范,都涉及到了那几方面,如何去实现,这样做的优缺点.

- 网络安全
  使用 HTTPS 协议, 网络方面的话 建议要使用 HTTPS 协议, 其相对 HTTP 传输协议多了一层 SSL 的安全防范, HTTP 传输协议都是明文传输,相对来说很不安全,假如传输

过程中被拦截下来,三方可以直接从传输的数据中获取到相关信息, 无法保障用户隐私安全. 网景公司设计了一层 SSL ,对传输数据进行加密,也就是我们进行的 HTTPS 传输协议, 多了一份对数据进行加密, 以及通过证书认证身份的网络协议, 要比 HTTP安全.
HTTPS 的主要思想是在不安全的网络环境下, 通过SSL 对数据进行适当加密以及证书被验证且被信任来实现安全的传输通道,对窃取和中间人攻击提供合理的防护。

- 中间人攻击
- 打包证书校验
  主要解决方法是通过 AFNetworking 来进行证书校验

  服务器返回的证书与本地证书公钥进行校验,如果正确则继续.
  通过字节常量数据来读取证书,因为将证书直接保存到项目中, Bundle的证书文件可能会被人替换而实现中间人攻击.
  设置证书数据,
  创建 AFHTTPS管理类,
  设置安全策略。

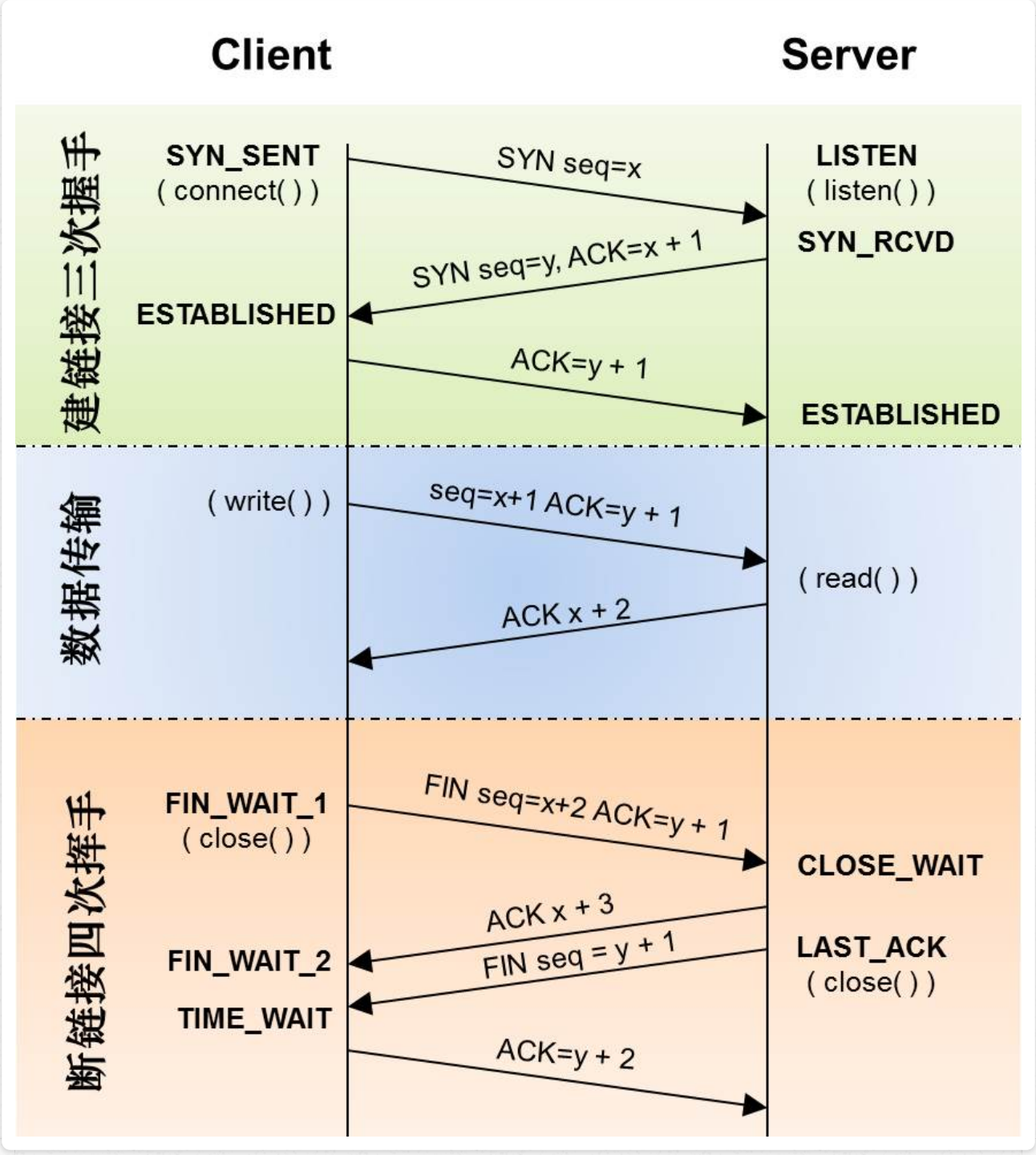
```
    // 服务端返回的证书与本地保存的证书中的publicKey 部分进行校验,如果正确则继续,不校验证书有效期。
    AFSecurityPolicy *policy = [AFSecurityPolicy policyWithPinningMode:AFSSLPinningModePublicKey];
    // 从字节数据加载证书数据,从常量加载能防止因 bundle 的 cer文件被人替换而实现中间人攻击,提高App 安全性.
    NSData *cerData = [[NSData alloc] initWithBytes:YMAPICerBytes length:sizeof(YMAPICerBytes)/sizeof(YMAPICerBytes[0])];
//设置证书数据, 不设置pinnedCertificates时默认搜索bundle的cer文件自动设置证书数据
policy.pinnedCertificates = [NSSet setWithArray:@[cerData]];
     AFHTTPSessionManager *manager = [AFHTTPSessionManager manager];
//设置HTTPS安全策略
manager.securityPolicy = policy;
```

- 防止参数篡改
- 防止重放性攻击
- 检测是否是越狱设备
- 代码混淆
- 包名检测
- 安全键盘

- 用户敏感信息采用加密传输,尽量不要存储在本地
- Apple 安全文档

3. 网络传输协议的实现原理. HTTPS HTTP TCP/IP 等

参考: HTTP/TCP
位码即tcp标志位,有6种标示:SYN(synchronous建立联机) ACK(acknowledgement 确认) PSH(push传送) FIN(finish结束) RST(reset重置) URG(urgent紧急)
Sequence number(顺序号码) Acknowledge number(确认号码)
三次握手



1. 第一次握手：建立连接。客户端发送连接请求报文段，将SYN位置为1，Sequence Number为x；然后，客户端进入SYN_SEND状态，等待服务器的确认；
2. 第二次握手：服务器收到SYN报文段。服务器收到客户端的SYN报文段，需要对这个SYN报文段进行确认，设置Acknowledgment Number为x+1(Sequence Number+1)；同时，自己自己还要发送SYN请求信息，将SYN位置为1，Sequence Number为y；服务器端将上述所有信息放到一个报文段（即SYN+ACK报文段）中，一并发送给客户端，此时服务器进入SYN_RECV状态；
3. 第三次握手：客户端收到服务器的SYN+ACK报文段。然后将Acknowledgment Number设置为y+1，向服务器发送ACK报文段，这个报文段发送完毕以后，客户端和服务器端都进入ESTABLISHED状态，完成TCP三次握手。完成了三次握手，客户端和服务器端就可以开始传送数据。以上就是TCP三次握手的总体介绍。

四次挥手

1. 第一次分手：主机1（可以使客户端，也可以是服务器端），设置Sequence Number和Acknowledgment Number，向主机2发送一个FIN报文段；此时，主机1进入FIN_WAIT_1状态；这表示主机1没有数据要发送给主机2了；
2. 第二次分手：主机2收到了主机1发送的FIN报文段，向主机1回一个ACK报文段，Acknowledgment Number为Sequence Number加1；主机1进入FIN_WAIT_2状态；主机2告诉主机1，我“同意”你的关闭请求；
3. 第三次分手：主机2向主机1发送FIN报文段，请求关闭连接，同时主机2进入LAST_ACK状态；
4. 第四次分手：主机1收到主机2发送的FIN报文段，向主机2发送ACK报文段，然后主机1进入TIME_WAIT状态；主机2收到主机1的ACK报文段以后，就关闭连接；此时，主机1等待2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，主机1也可以关闭连接了。

至此，TCP的四次分手就这么愉快的完成了。当你看到这里，你的脑子里会有很多的疑问，很多的不懂，感觉很凌乱；没事，我们继续总结。

HTTP 是一种超文本传输协议, 属于明文传输, 相对来说是不安全的, 当今主要采用 HTTPS 安全传输协议来进行网络通讯, 其相对于 HTTP 多了一个 SSL 层, 通过 SSL 来做一个证书校验 受信任和 加密传输的方式, 使服务器和客户端建立一条安全的传输通道, SSL 证书通过知名的证书颁发机构颁发.

HTTPS 传输原理其实还是不变的, 抛开 S 层来看的话, HTTP 是建立在 TCP 协议连接之上的, 通过 TCP 连接之后, 客户端通过 HTTP 传送数据到服务器, 收到服务器相应。

在 HTTP 1.0 的时候, 客户端与服务器建立一次连接完成之后就立即断开. 需要重新发起回话必须重新建立连接。
到了 HTTP 1.1 则可以在建立连接之后发送多条请求, 请求可以异步进行, 不会发生阻碍关系。

HTTP 传输流程一般是:
1. 客户端发送 Requeste Header + Requeste Body: 请求头和请求体. 其中请求头主要包含内容为: 具体请求方法, HTTP 协议版本

4. 如何进行的加密, 你了解的加密算法都有那些, 安全吗

资料: 安全加密

- 对账号密码采用 MD5 进行加密.
- BASE 64 加密
- AES

5. 如何使用路由实现页面之间的跳转.

6. 说一下中间人攻击,重放等.

在目前网络上, 很多都是在使用相对安全的 HTTPS 协议, 其采用明文加密以及证书信任校验来建立一条安全的通讯通道,但是这样做一样就真的安全了吗。黑客三方恶意软件等可以在证书信任校验这一步进行入手,具体实现方法是其伪造一个可信任的证书,此证书一旦安装到设备上,且

7. 理解的MVC MVVC. MVVC 里的双向绑定

8. 如何使用的 WKWebView 做 OC 与 JS 交互.

9. 说一下你封装的常用工具库.

10. 如何继承 IM 云客服框架.

参考 IM 云客服文档, 调用其接口,初始化项目, 注册相应的 ID.

11. 单个工程如何实现多 Target. 应该注意什么问题.

12. 说一下转场动画, 隐式动画等.

13. 代码规范.

我认为一个好的项目不仅在框架设计方面和应用不同需求的解决能力方面体现。我觉得代码规范也是衡量一个项目好与差的标准。 如果从项目的初期开始就严格使用一套统一的代码规范,并且外加上框架设计的足够好,一定能减少后期很多问题,使其走的更远。
代码规范不仅提高代码可读性,新人接入之后可快速上手, 提高平时修复 Bug,版本迭代效率。 后面的这几点通过合理的代码规范来控制,能起到很大的作用。
我在项目中代码规范几个重要的点分别是:
* 对经常要用到的控制器使用一套方法调用模板.这个可以通过 Xcode 来设置. 一般在 Viewdidload: 进行 导航栏设置, 视图设置, 网络请求, 添加通知. 然后还有 tableView 代理方法等. 基本每个类都是以这样的顺序来执行.
* 统一使用 .语法, 懒加载, Setter getter 方法写在类最后面.
* 驼峰命名法. 第一个字母小写,后面间隔的每一个大写, 类名开头是大写。
* 每个方法结束之后 一个空格接括号。
* 在一段长的代码里面 一个逻辑相当于一块整体, 每个整体一个回车分割开。
* 项目内一些类命名,以公司简称 三个英文字母开头. 由于Apple 都是以两个字母开头, 防止那一天Apple 更新出一个与我们同名的类就尴尬了。
* 对外提供的常量使用 K 开头.
* 多使用静态变量串声明常量,而非宏定义, 静态变量可以标注类型, 一目了然。

14. 如何实现数据统计埋点与业务分离.

这里主要涉及到两个三方库,一个是友盟和Bugly,友盟用来上报统计结果, Bugly用来上报 app 崩溃分析, 关于数据统计埋点与业务分离的实现方式主要是使用 Runtime 的消息转发 和 方法交换来实现的, 主要是对 UIControl 和 UITableView 以及 手势 添加一个分类, 然后在 load 方法里面进行方法交转, 因为对于 UIControl 类触发消息机制,都会调用到 sendAction:to:forEvent 方法, 然后对其进行一个方法交换, 将, 然后在原有方法里面通过 拼接 当前页面控制器 + 按钮名 成统计ID; 统一进行上报.
针对 UITableView 则是通过拦截 setdelegate 方法, 获取到当前 delegate, 然后在通过 delegate 进行对其 tableView: didselectrow 进行方法交换, 进行页面统计, 统计 ID 也是类似的拼接方式完成。以及一些主要控制器的停留时间统计一般实现方法是在 ViewWillapper 进行统计,然后在到 ViewWilldisapper 进行结束的统计.

15. 说一下 SDWebImage 实现原理图片加载原理, 假如客户端正在下载很多张图片,突然切换到后台, 这些数据如何处理.

客户端加载一张网络图片, 首先会先去检测 URL 是否为空,如果不是则先查看 内存, 如果内存有则直接从内存读取,假如没有则在到磁盘中查找, 磁盘中如果有则从磁盘读取,否则直接通过 URL 网络下载,下载完之后依次将图片缓存到内存和磁盘中。

16. 如何采用中间人思想进行数据传输. 解耦.

以控制器跳转为例,一般一个控制器想要跳转到另一个控制器,正常需要 A控制器引入 B控制器,然后在 A控制器里面实现生成 B 控制器实例, 在进行跳转. 这样 A 控制器会对 B 控制器有一层引用. 在一个 App 中页面跳转关系要更加的复杂. 如果每次跳转都是采用相应的逻辑, 可能 A 控制器里面要引入好都个控制器, 很显然这样做 A 控制器相对其他控制器来说耦合度太高了, 并且跳转代码看起来也不够优雅. 后期维护管理起来也不方便.

这里采用中间人思想引入 模块化的管理方式来进行页面跳转以及数据传输, 主要实现思路是创建一个 中间层, 中间层通过 runtime 的方法来通过 类名字符串 生成相应 控制器, 中间层也不需要知道要跳转的控制器是那一个, 这样主要的控制器生成代码都移动到中间层来实现即可. 相当于 每个需要跳转的控制器 对 中间层有一个依赖关系。而对其他控制器完全无耦合.

然后这里关于控制器的参数传递,我们可以在引入一个类, 每个控制器有一个专门对应的类, 来负责初始化,以及参数赋值, 参数通过 字典的方式传递。 否则把所有不同的控制器初始化都写在 中间层也会有很大程度的代码冗余

17. 说一下你理解的 delegate. block. 使用Block 注意的点

两种都是属于一种回调机制, 对于 delegate 一般是在对外提供接口, 提供的方法比较多的时候,一般使用它来实现比较方便. block 的话一般是一块简短的代码块的时候会采用它, 因为使用其相对 delegate 来说代码量相对会少很多.
两种优缺点: delegate 相对 block 相对会有一点儿优势, 因为 delegate 只是多创建了一个类, 而block 的由于其系统机制, 相对来说性能开销会更大一点, 而且有时候会发生内存泄漏. 一般系统使用 block 进行回调都是针对 类方法使用, 而对实例方法很少.

18. 你对多线程的理解, GCD. NSOperation. NSThread.

- GCD 信号量 信号量就是控制访问资源的一个指标, 因为在多线程中, 系统提供访问的线程资源是有限的, 有时候如果我同时开启多个子线程, 但是系统可能没有这么多对应的资源供应我们访问, 所以此时 我们最好通过信号量来对可访问的子线程进行控制管理.

19. 对Runtime. RunLoop 的理解. 以及应用。

Runtime: 南峰子
App Runtime
RunLoop 理解: RunLoop 其是与没一条线程一一对应的, 当项目运行执行到 Main 函数生成 application 的时候, 也同时开启 一个默认模式的 RunLoop 对应 主线程工作, 它的工作原理是 系统休眠的时候停止运行, 等系统开始运行的时候

五个模型

- 顾名思义其表示不断的循环运行. 它实现的思路大概是以 do while 完成, 判断条件是直到 当前线程退出, 才会结束循环. 其实它是与 线程一一对应的, 可以说起是为了线程而生,没有线程, 就没有 RunLoop 的必要, 其是线程的基本框架部分, Cocoa 和 CoreFundation 都提供了对 runloop 对象提供配置 和管理的接口.
- 当项目启动执行到 Main 函数 的 UIApplicationMain 函数的时候, 这个方法会同时为主线程生成一个 runloop 对象, 使其与主线程同时运行, 来管理应用事件响应等, 所以我们应用才能再 无人响应的时候可以休息, 当需要它的时候 可以立即开始干活起来.
- 对于其他线程, runloop 是默认不会自动启动的, 当我们想在其他子线程运行 runloop 时 需要手动调用. 比如在解决 计时器 在滚动页面 的问题, 将定时器添加到其他线程, 之后还需要将 runloop run 启动才可以。

- NSRunLoop *runloop = [NSRunLoop currentRunLoop]; 可以通过此代码来获取任意线程的 runloop;

  runloop 详解
  YYKit RunLoop

20. 如何减少 App 崩溃率.

BayMax

- unrecognized selector crash 消息转发机制导致. 一个对象调用一个方法, 相当于这个向这个对象发送一条消息. 重新 methodSignatureForSelector: 和forwardInvocation: 判断如果在本类中找不到 NSMethodSignature. 则将消息转发到自定义的一个单利对

象里面, 然后会调用invokeWithTarget: 把这个单利对象设置成 Target, 这样单利收到消息, 我们在通过重写 resolveInstanceMethod 方法, 通过Runtime 使用 SEL 参数 来动态为这个单利添加上这个方法. 实际上这个方法什么也不做, 并不会影响到项目.

- KVO Crash KVO 崩溃的原因主要是: 我们添加观察者和删除观察者必须是成对的方式进行, 否则在系统通过字典删除,找不到这个 key, 导致崩溃. 我们通过重写 添加观察者和移除观察者 的实现, 然后在分类里面添加一个 全局的 Map, 每次添加的时候先把观察者和观察Key, 添加到 Map 字典上。在调用原有的方法, 然后删除的时候, 我们可以先来查询这个 Map 里面是否有这个 Key. 如果没有则直接跳过, 防止系统调用导致异常。

  *** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '+[NSMutableDictionary dictionaryWithObjectsAndKeys:]: second object of each pair must be non-nil. Or, did you forget to nil-terminate your parameter list?'
  *** Terminating app due to uncaught exception 'NSRangeException', reason: 'Cannot remove an observer for the key path "schoolNameww" from because it is not registered as an observer.'

- NSNotification Crash 在 iOS9 之后得到系统改善, 应该是在 dealloc 添加了 移除添加统治者的代码. 因为我们对一个消息添加通知, 这个类为接收者. 在这个类销毁之后, 没有将这个通知从接收者移除掉, 然后发送消息那边又发送了一条消息, 这条消息会直接发送给 接收者, 但是此时接收者 可能是一个野指针 或者 nil。 假如是一块野指针地址, 那么这个消息发送到野指针上, 会直接导致异常 崩溃, 野指针错误. 在 OC 中禁止向被回收的内存地址发送消息。

- NSTimer Crash

- Container crash（数组越界，插nil等） 这里主要针对 NSArray,NSDictionary,NSMutableArray,NSMutableDictionary. 常用的读取和插入方法进行 方法交换, 在读取数组的时候, 先做一个逻辑判断是否越界. 在插入字典的时候判断插入值是否为 nil 等 在去调用原生方法。

- NSString crash （字符串操作的crash）NSString 和 NSMutableString 导致异常崩溃, 和容器类导致异常原理类似. 比如对字符串截取, 越界. 对字符串转换。

- Bad Access crash （野指针）野指针错误在 应用崩溃中占比率最高。 这里防护是一个重点.野指针类型crash的表现为：Exception Type:SIGSEGV，Exception Codes: SEGV_ACCERR 或者

- UI not on Main Thread Crash (非主线程刷UI (机制待改善)) 解决思路: 对UIView 写一个分类,使用 Runtime 来动态更换 系统方法, UIView 涉及视图更新的有四个方法,setNeedsLayout,setNeedsDisplay,setNeedsDisplayInRect,setNeedsUpdateConstraint

对这是个方法进行Hook 方法交换,然后在自定义的方法里面判断当前是否在主线程,如果在继续调用此方法, 否则通过 GCD 切回主线程在调用. 用到的线程判断方法是使用 C语言实现的 String.h 的 strcmp 这是一个比较字符串的方法,你

- 按钮或者导航控制器被多次点击,事件多次触发导致 Crash. 这里针对 pushViewController 做方法交换, 然后判断当前控制器是否就是要推出的控制,直接 retrun. 还有判断

- # 21. 埋点统计

22. App 启动流程优化

我将 App 启动流程主要分成二部分, 第一部分是 Per-Main, 第二部分为: Main 之后.
一个 App的启动 从点击 Icon 开始, 系统要去动态加载,链接,绑定. 主要做了 三件事。

- Mach-O 格式
- 虚拟内存基础
- Mach-O 二进制加载

  主要进行的优化是针对 Main 之后的部分进行.

  Main 之前

1. 尽量减少使用 +Load 方法. 如果可以的话尽量使用 initialize。一旦项目里有过多个 Load 需要加载, 每个方法占用 几毫秒的事件, 全部加起来也是相当耗时的, 主要原因是 + Load 方法会在项目启动之前系统会将每个类实现的 +Load 方法执行. 而 initialize 是当此类收到消息之后才会去调用, 如果此类没有实现此方法,就会调用其父类。 类似懒加载的机制。
2. 检查项目里一些 不需要使用的静态变量, 常量。类和方法等, 还有一些功能基本一致, 方法名不同的, 尽量都清理掉, 减少系统加载时间. 使用 AppCode 扫描未使用的值, 参数,方法等.
3. 减少引入不必要的Framework.
4. Main 之后
   a. 主要启动页面的 UI, 都不要使用 Xib 或者 故事版去构建, 都使用纯代码来完成, 因为 Xib 故事版都还需要进行一次解析成代码, 来完成绘制。
   b. 在 ApplicationdidFinishLaunching 中进行主要优化, 将一些涉及 数据统计, 推送服务,项目缓存设置,初始化分享配置等功能, 按照实际业务需求具体划分, 尽量能延后加载就延后加载. 如果可以放到后台加载的也放到后台来进行加载。
   c. 主UI框架tabBarController的viewDidLoad函数里， 去掉一些不必要的函数调用。

d. 首次启动页渲染的页面优化使其更快展示;

23. 免密登录实现原理. 优缺点等

主要是在本地存储用户账户, 密码不进行保存, 然后在 App 启动的时候, 从缓存中读取上一次账号, 用账号和 设备号作为参数, 发送到后台实现免密登录。
这样做虽然避免了用户密码保存在本地出现的安全问题, 但是也同时引起了另一个漏洞, 假如黑客通过逆向分析得到我们是以这种方式来进行免密登录, 可以通过得到用户手机号 + 设备号 来作为参数 Pin 我们的这个登录接口,来完成登录的操作。

参考: 免密登录,我以为有多难
参考: app的登录认证与安全

在登录的时候如何保护用户密码防止泄露:
在关于密码验证,涉及隐私 敏感的信息,一般对其使用 MD5 加密,进行传输。 并且一律使用 POST 请求. GET 请求的话会把参数拼接到 URL 上,不太安全。

方案二:
客户端向服务器发送请求. 不传任何参数.
服务器利用 RSA 算法生成一对公钥和私钥.将公钥返回到客户端.
客户端通过公钥对 敏感数据进行加密,在传输到服务器。
服务器使用私钥解开加密 进行验证。

引入SessionId 和Token 方案三:
客户端向服务器发送请求. 不传任何参数.
服务器利用 RSA 算法生成一对公钥和私钥.将公钥返回到客户端.
客户端通过公钥对 敏感数据进行加密, 然后利用 RSA 算法生成一对公钥和私钥. 将公钥和密文传输到服务器。
服务器使用私钥解开加密,完成验证, 在利用公钥对 Token进行加密, 将 SessionId 和 加密的Token返回到客户端。
客户端利用自己生成的私钥对token密文解密， 得到真正的token。

引入 SessionId 每次请求时, 将其添加到请求参数说,来验证是否来自客户端请求. 但是为了防止 SessionId 被拦截获取到之后, 黑客利用其来伪造客户端进行请求, 又添加了一个Token的验证。

24. 性能优化都做了什么? 当发现某个页面突然内存暴增, 卡顿异常的情况,应该如何处理。有什么好的处理内存泄漏, 性能优化的方案。

WeRead 性能优化总结

25. weak 是如何实现的。 延展 copy strong, 底层实现机制。 TODO;

由 Runtime 维护的一个 SideTable 来进行管理, 在 NSObject.mm 内实现, 其是一个结构体, 里面包含 原子操作锁, 引用计数 hash 表, weak 引用全局 hash 表. 对于 weak 修饰的对象, 当我们对其赋值时, clang 将其编译成 runtime 的方法, object_initWeak(&p, p) 的形式转换. 此过程主要传入 被弱引用对象的地址,和其本身。 在方法内部通过 Hash 表以对象地址作为 Key 的形式来存储弱引用对象. 直到弱引用对象 释放的时候, 在通过 这个 Hash 表来查找引用其的对象. 将其置为 nil。

```
    struct SideTable {
     // 保证原子操作的自旋锁
     spinlock_t slock;
     // 引用计数的 hash 表
     RefcountMap refcnts;
     // weak 引用全局 hash 表
     weak_table_t weak_table;
}
```

参考 weak 冬瓜

weak

26. 有没有遇到过野指针,野指针处理方法。

在项目非上线的时候, 一般看报错类型可以看判断出来是野指针错误, 野指针类型crash 的表现为： Exception Type:SIGSEGV， Exception Codes: SEGV_ACCERR. 异常访问。

打开 Xcode 提供的 zombie 机制. 这里会提示指定类访问导致异常。

27. SDWebImage 实现原理, 如果在一个页面同时下载多个图片, 当突然断开, 其对已经下载好的图片如何做存储。

28. 循环引用解决方法。

循环引用经常出现在 delegate, block, NSTimer 上. 主要原因是 A 持有 B, B 又同时持有 A。 相互持有 导致其一方无法正常释放. 一般做法是将其中一方改为使用 weak 来修饰。 不会对其引用计数器 +1.
比如在 delegate 下, 一般 delegate 属性使用 weak 来进行修饰, 而非 strong。
在 block 下, 比如在一个类里面实现 block, 由于 block 是一块代码段, 其先声明好但是何时对其调用自己并不知道, 所以系统 使 block 对当前类 以及 声明的 变量等, 都做了一层引用。 这是 block 的一个特性. 我们在 block 外部 使用 weak 来对 self 进行修饰。
NSTimer, 启动定时器要需要赋值其一个 Target, 以及 SEL, 时间间隔, 是否重复等. 然后定时器 加入到 RunLoop 当中执行循环. 此时 NSTimer 对 对象是持有关系, 假如我们想释放掉对象, 则必须先销毁掉这个定时器, 将其置空即可 正常释放。

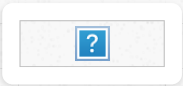
29. 消息转发机制。

根据Apple定义: 将一条消息发送到一个不能处理该消息的对象上是一个错误的,为了防止直接崩溃,Apple 在宣布崩溃之前,使用运行时系统为接收对象提供了第二次接收消息的机会。

OC 里面的所有方法调用都是基于给这个对象发送消息的原理实现。
发送消息:
1. 先通过 isa 指针找到 Class.
2. 从 Class 的 cacheMothlds 中遍历查找方法。
3. 从 Class 的 Mothlds 中遍历查找方法。
4. 查找 Class 的 superClass。
发送消息最后进行到 Class 的根类 还是没有找到相应方法时, 系统会进行消息转发机制, 来让开发者进行补救。
消息转发: 如果一个对象无法响应一个消息, 走完发送消息流程之后,Apple 给一个补救机会,提供第二次接收此消息的机会.

1. 询问是否要重写该方法. 动态方法解析, +(Bool)resolveInstanceMethod::(SEL)sel resolveClassMethod 我们可以在类里面重写此方法, 动态添加一个其他方法来代替, 结束消息转发流程。
2. 询问是否要将该方法转发到其他类上。 - (id)forwardingTargetForSelector: 动态方法转换, 我们可以通过重写此方法来, 将此方法转发到其他类去完成。
3. - (void)forwardInvocation: 最后一步。 NSInvocation 参数 包含一个整条方法信息。
如果一个对象无法响应一个方法，那么就会进入消息转发机制
第一步，我们可以动态创建一个方法实现去响应这个消息，消息转发结束
第一步未处理，第二步我们可以选择一个备选的消息接收者去处理这个消息
第二步未实现，最后一步第三步，启动完整的消息转发机制，处理方法签名，实现NSInvocation对象转发。

最好在第一步处理，次之第二步，如果第三步只是使用备选接收者处理，还不如直接第二步快速处理结束，毕竟第三步需要方法签名和封装NSInvocation。



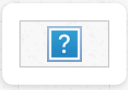
30. 响应链循环转发机制。

31.iOS 与 JS 交互细节, 直接在代理加载页面完成在注入,导致注入过晚, 有什么好的解决方法。

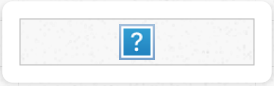
32. GCO block 实现机制。

33.远程推送实现原理。

iOS 系统中推送主要分两种, 一种是本地推送, 一种是远程推送。 远程推送是系统级别的, 没有开启应用,也可以接受到. 本地推送的话, 应用关闭之后是无法接受到的.
APNS推送机制:



Provider: 推送消息的服务器, 负责将消息发送到 Apple 的 APNS 服务器上.
APNS: Apple 的一个负责转发通知的推送服务器. 它可以通过查找 Token 找到已经注册开启通知功能的 客户端.
iphone: 接收到推送消息之后,把相应消息发送给指定的应用程序, 弹出通知. 点击则会进行 client-app 的 回调事件。



1. 一般在 ApplicationdidFinishLaunch 进行注册, 注册要提供 设备 UIID, 推送证书文件, 应用 ID. 数据发送到 APNS 服务器。
2. APNS 返回注册成功的 APP Token。
3. 将 App Token 保存到 服务器中. 每个 Token 代表注册通知的每台设备.
4. 服务器使用 Token 为参数 发送通知到 APNS 服务器.
5. APNS 接收到相关通知, 在通过 Token 找到 相应设备, 将 Notification 发送到指定的设备里.

关于证书: 证书制作

34. 组件化工具

35. 导航控制器,Tabbar控制器,视图控制器的 Push, Present 相应的视图层级结构关系。

36. @property 相关关键字,以及其原理

内存管理: Strong copy weak assign 还有一个是负责 OC 对象内存管理的,与Weak类似,只是其释放时不会置 nil

37. HTTP中GET和POST有什么区别,还有使用其他请求方式吗?

38. TableView 的复用机制,如何进行优化提高流畅, 有没有对其进行过重写。

复用机制: 一般来说 TableView 是 结合 HeaderView Cell 等布满屏幕, 其 HeaderView Cell 都可以使用复用机制, 除了初始屏幕上展示出来的 Cell, 都是直接创建的方式来生成外, 每次我们向上滑动向下滑动, 一般会有一个 Cell 向外滚出, 一个新的 Cell 新加入. 这时会调用 tableView 相应生成 Cell 代理方法, TableView 有一个对 Cell 的缓存池, 当屏幕上有一个 Cell 滚出时, 会自动将其加入到缓存池内去, 然后在开发中 我们可以在代理方法通过 tableView 调用缓存池查找的方式, 来判断缓存池中是否有类似的 Cell, 如果有则直接拿出来使用, 否则的话就手动创建. 这样在一个表格上面 即使展示了 上千个同类型 Cell, 总共需要生成一个屏幕上所展示的 + 1个 两个就足够了。

方法一:

1. 针对自定义 TableViewCell 的方式, 这里一般会先进行一个注册, 我们在初始 TableView 的时候进行一个注册, 将自定义 Cell 类型 和 相应标识符传入 注册接口的参数中。
2. 在 tableView 数据源方法中进行复用, 调用 **dequeueReusableCellWithIdentifier: forIndexPath:** 然后传入相应标识符和当前 indexPath。来获取 cell, 这里主要是从缓存池中去找是否有这种 Cell, 如果没有会自动帮我们创建。

方法二:

1. 针对未自定义 TableViewCell 的方式, 不需要提前进行注册, 只需要相应标识符即可, 只

需要在给相应 TableView 添加一个标识符, 然后在通过
dequeueReusableCellWithIdentifier: 传入标识符, 然后到缓存池里面找是否有相应类型的 Cell。
2. 如果有找到直接复用, 没有的话, 我们自己通过 初始化方法来进行创建即可。

复用需要注意的问题.
1. 设置 Cell 的存在差异性的那些属性（包括样式和内容）时，有了 if 最好就要有 else，要显式的覆盖所有可能性。
设置 Cell 的存在差异性的那些属性时，代码要放在初始化代码块的外部。

TableView 优化
• 通过正确的设置 reuseIdentifier 来重用 Cell。
• 尽量减少不必要的透明 View。
• 尽量避免渐变效果、图片拉伸和离屏渲染。
• 当不同的行的高度不一样时，尽量缓存它们的高度值。
• 如果Cell 展示的内容来自网络，确保用异步加载的方式来获取数据，并且缓存服务器的 response。
• 使用 shadowPath 来设置阴影效果。
• 尽量减少 subview 的数量，对于 subview 较多并且样式多变的 Cell，可以考虑用异步绘制或重写drawRect。
• 尽量优化 - [UITableView tableView:cellForRowAtIndexPath:]
方法中的处理逻辑，如果确实要做一些处理，可以考虑做一次，缓存结果。
• 选择合适的数据结构来承载数据，不同的数据结构对不同操作的开销是存在差异的。
• 对于 rowHeight、sectionFooterHeight、sectionHeaderHeight 尽量使用常量。

39. 常用的设计模式, 例如工厂模式,单例模式,代理模式等等. 例子。

40. 协议 @protocal, 分类 Category, 延展 Extension 匿名, 说一下其原理,应用等等。

@protocal: 其是一个只有声明,没有实现的 .h 文件, 其主要作用是, 用于实现委托代理协议。 在@protocal 中写 @property 时, 只有相应的 set get 方法声明,没有相应的实现, 并不能完成相应的 @property 效果. 有时候可以应用 @protocal 来完成 类似 C++ 的多继承 机制。 让一个类遵守多个协议, 然后去完成相应的实现。

@Category: 其是 OC 2.0 添加的一个新特性, Apple 推荐其主要使用场景是一个复杂的类可以通过类别的方式抽取出不同的主要功能通过模块化的方式进行完成, 减少一个文

件体积, 方便多人共同开发. 然后我们从分类 加载的机制中, 还可以利用其机制来替换掉系统原有方法等等. 对原有类进行一个扩展。
注意点:
1. 分类不可以直接添加属性, 如果需要则使用 runtime 通过关联对象的方式来添加。 在分类的 set get 方法中使用 runtime Api 关联对象添加属性。
2. 分类的加载 是在 运行期间 决定的, 是在原有类已经确定下结构下来之后才进行, 所以想要对原有 类的属性进行添加, 原有类的 内存字节已经计算确定下来, 如果通过正常的方式进行添加则会报错.则会破坏掉原有类内存结构布局等。 要借助 runtime。 原有类的 method 添加方式是在 原 Class 的 methodlist 中进行插入的方式, 会把分类中方法插在最前面, 所以起到覆盖原有方法的效果, 实际上并没有把原有方法替换掉。
3. 分类中不会加载 +load 方法

OC 是一门动态运行的语言: 主要基于其很多决策都放在 runtime 运行时来判断。

Extension: 很像一个 匿名的分类, 一般用于声明私有属性, extension在编译期决议，它就是类的一部分，在编译期和头文件里的@interface以及实现文件里的@implement一起形成一个完整的类，它伴随类的产生而产生，亦随之一起消亡。extension一般用来隐藏类的私有信息，你必须有一个类的源码才能为一个类添加extension，所以你无法为系统的类比如NSString添加extension。

41. 持久化方案. 都是用了什么方式实现等等。

```
[数据持久化](https://www.jianshu.com/p/7616cbd72845)
* plist文件（属性列表）
* preference（偏好设置）
* NSKeyedArchiver（归档）
* SQLite 3(数据库)
* CoreData（数据库）
```

沙盒

其是 iOS 的一种保护文件安全的机制, 它其实是一个针对单个项目的文件夹, 每个项目只能访问属于自己的这个文件夹
应用程序包: NSString* path = [[NSBundle mainBundle] bundlePath]; 这里主要包含 应用程序源文件 xxx.app 资源(图片, info.plist文件),代码签名, 可执行文件等等.
* documents: 最常用的目录, 一般 使用备份手机软件都是会对这个文件夹进行备份. 适合存储重要数据。NSString *path = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES).firstObject;
* Library -> Cache. iTunes不会同步此文件夹，适合存储体积大，不需要备份的非重要数

据。NSString *path = NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask, YES).firstObject;

Preference: iTunes同步该应用时会同步此文件夹中的内容，通常保存应用的设置信息。
* Tmp： iTunes不会同步此文件夹，系统可能在应用没运行时就删除该目录下的文件，所以此目录适合保存应用中的一些临时文件，用完就删除
NSString *path = NSTemporaryDirectory();

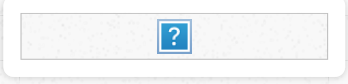
42. 深拷贝与浅拷贝

其主要针对 对象 和 容器类。
深拷贝: 对一个对象内容进行拷贝, 生成新的指针,用于保存. 得到一份与原有对象内容相同, 内存地址完全不同的对象。 拷贝容器本身,返回一个对象,指向不同的内存地址.
浅拷贝: 对一个对象内容与地址进行拷贝, 生成一个与原有对象内容指针一直的对象. 指向同一个地址。简单说 拷贝容器本身,返回一个对象,指向相同的内存地址.

43. GCD,NSOperation,NSThred, 创建方式,都有什么区别特性等。

答: 一般在项目中都会用到多线程技术, 目前有三种实现方案,GCD会用的比较多,有时候也会用 NSOperation, NSThred的话很少有到。然后先说一下这三种类型的特点和基本使用方式, NSThread, 它提供三种创建方式 可以通过 动态显示创建 alloc initWithTarget 添加相应参数即可 然后可以设置其优先级,静态显示创建 detachNew,隐式创建就是 我们平时会用到的Self PerformSelect 然后可以选择在子线程运行或当前线程去执行. 然后到 NSOperation, 其引进了 队列来装载线程的方式实现多线程技术, 队列分别有串行,并行, 表示的是队列
GCD是基于C封装的函数,具备非常高的效率,在ARC环境下,无须主动管理内存,无须dispatch_retain和dispatch_release,可以将重点关注在业务逻辑上。

[谈iOS多线程(NSThread、NSOperation、GCD)编程]



- 进程： 一个具有一定独立功能的程序关于某个数据集合的一次运行活动。可以理解成一个运行中的应用程序。
- 线程： 程序执行流的最小单元，线程是进程中的一个实体。
- 同步： 只能在当前线程按先后顺序依次执行，不开启新线程。
- 异步： 可以在当前线程开启多个新线程执行，可不按顺序执行。
- 队列： 装载线程任务的队形结构。

- 并发： 线程执行可以同时一起进行执行。
- 串行： 线程执行只能依次逐一先后有序的执行。

  注意:
  • 一个进程可有多个线程。
  • 一个进程可有多个队列。
  • 队列可分并发队列和串行队列。

44. AFNeterworking 基于什么实现的网络请求。

45. 对三方库的源码阅读。

46. 两炷香,开关判断灯光,红黄蓝,空瓶子换啤酒。

47. OC 语言的三大特征体现

封装、继承、多态

OC 三大特征
封装:
继承:
多态:

对公司要什么需要问的

- 公司产品相对竞品有什么优势, 未来想要通过什么方式来跟竞品做竞争。
- 技术部会不会经常组织一些技术分享等。
- 当前iOS 项目主要都使用到什么技术框架.
- 如果我有幸被公司选中, 我需要提前做什么准备 来更好适应这个岗位。
-
- 公司组织架构,发展方向,技术部架构.企业文化。