面试集合一

来源：iOS 程序犭袁 大神的技术清谈微信群

进度：11/161

版本：draft

解答：pmst

最后更新：2020/06/09

⚠️:欢迎转载，转载请备注出处。

1. webView 和 wkWebView 的区别

2. hybrid 的原理，怎么调用 js 方法的

2.1 Webview URL 拦截

- oc 调用 js 代码，借助 UIWebview 和 WKWebview 类开放的两个接口：

```
// UIWebview
- (nullable NSString *)stringByEvaluatingJavaScriptFromString:(NSSt
ring *)script;

// WKWebview
- (void)evaluateJavaScript:(NSString *)javaScriptString completionH
andler:(void (^ _Nullable)(_Nullable id, NSError * _Nullable error))
completionHandler;
```

- js 调用 oc 方法，需要自定义一套规则，比如 `client://action?param1=xxx&param2=yyy` 等，网页端通过 `location.href` 或构造 `iframe` 发送请求，客户端拦截请求、解析、派发给指定业务模块执行:

```
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NS
URLRequest *)request navigationType:(UIWebViewNavigationType)naviga
tionType{
    if ([request.URL.absoluteString hasPrefix:@"client://mail"]) {
       // 发送邮件
       return NO;
    }
    return YES;
}
```

2.2 WebViewJavascriptBridge

WebViewJavascriptBridge 之前学习源码的一些笔记

2.3 WKScriptMessageHandler

html 向客户端发起请求，首先客户端先 addScriptMessageHandler 一个名为 method 方法，并且实现代理方法， 网页端 javascript 通过 postMessage 的方式向客户端发送消息。

```
// 客户端
- (void)setupWKWebView{
    WKWebViewConfiguration *configuration = [[WKWebViewConfiguration al
loc] init];
    configuration.userContentController = [[WKUserContentController all
oc] init];
    [configuration.userContentController addScriptMessageHandler:self n
ame:@"method"];

    WKWebView *webView = [[WKWebView alloc] initWithFrame:self.view.fra
me configuration:configuration];
    webView.UIDelegate = self;
}

// 实现代理方法
- (void)userContentController:(WKUserContentController *)userContentCon
troller didReceiveScriptMessage:(WKScriptMessage *)message{
    if ([message.name isEqualToString:@"method"]) {
        // 解析
     }
}

// HTML
window.webkit.messageHandlers.method.postMessage()
```

2.4 JavaScriptCore(UIWebview)

深入浅出 JavaScriptCore

2.5 React Native 通信

5. 输入网址打开一个页面的流程，从浏览器输入 url 到返回页面经历了什么

- 从输入 URL 到页面加载的过程？如何由一道题完善自己的前端知识体系！
- 前端经典面试题: 从输入 URL 到页面加载发生了什么？
- 从输入 URL 到页面展示，你想知道些什么？
- 从 URL 输入到页面展现的过程

6. 堆的数据结构

数据结构：堆（Heap）

7. 二叉搜索树的作用

8. mmap 的原理

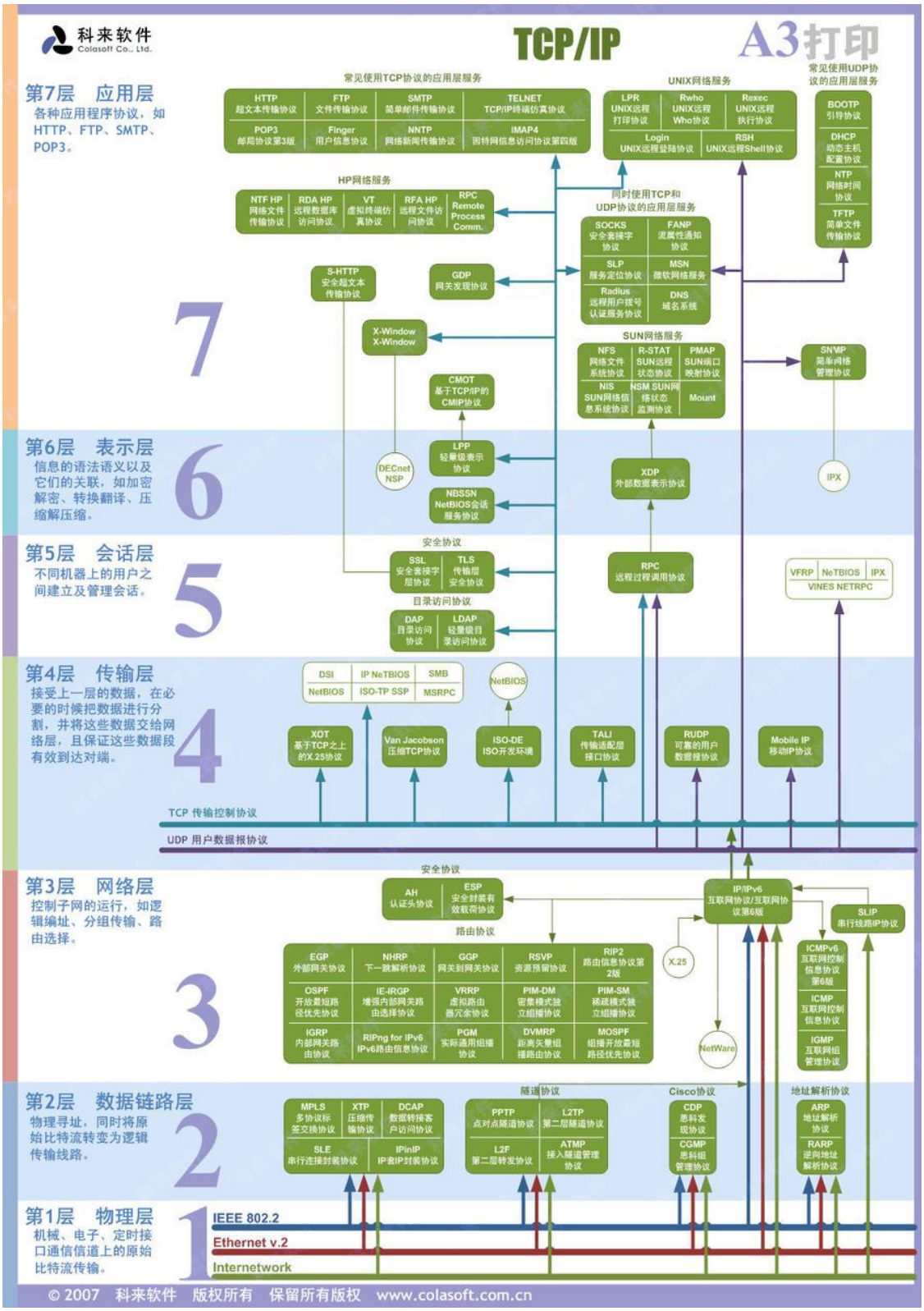
4. swift 跟 oc 怎么混合开发

5. SSL 里面怎么确保数据交换的过程、SSL 用了什么加密算法, 分别加密的是哪一个部分、为什么这么做 (提示: 对称加密、非对称加密都用到了, 对称加密传什么、非对称加密传什么东西)

6. WebSocket 底层协议是哪一层?

Websocket 是应用层协议，全双工通讯技术，基于 TCP 传输协议，并复用 HTTP 的握手通道。

1. 浏览器、服务器建立 TCP 连接，三次握手；
2. TCP 连接成功后，浏览器通过 HTTP 协议向服务器传送 WebSocket 支持的版本号等信息，向服务端发送 websocket 的握手请求，复用了上面 HTTP 请求时 TCP 用到的握手通道；(这里客户端：申请协议升级，服务端响应协议升级，都是采用了标准的 HTTP 报文格式，且只支持 GET)
3. 服务器收到客户端的握手请求后，同样采用 HTTP 协议回馈数据;
4. 当收到了连接成功的消息后，通过 TCP 通道进行传输通信；



图源：科来软件。

服务端代码：

```
var app = require('express')();
var server = require('http').Server(app);
var WebSocket = require('ws');

var wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', function connection(ws) {
    console.log('server: receive connection.');

    ws.on('message', function incoming(message) {
        console.log('server: received: %s', message);
    });

    ws.send('world');
});

app.get('/', function (req, res) {
  res.sendfile(__dirname + '/index.html');
});

app.listen(3000);
```

客户端:

```
<script>
  var ws = new WebSocket('ws://localhost:8080');
  ws.onopen = function () {
    console.log('ws onopen');
    ws.send('from client: hello');
  };
  ws.onmessage = function (e) {
    console.log('ws onmessage');
    console.log('from server: ' + e.data);
  };
</script>
```

- WebSocket 于 HTTP 、WebSocket 与 Socket 的区别
- WebSocket 协议：5 分钟从入门到精通

7. group 如何实现 barrier 类似的功能?

8. gcd queue 的区别

9. gcd、NSOperation 区别, 功能方法区别.

10. HTTP、HTTPS 区别?

11. GET、POST 请求的 cache 怎么做,几级缓存? 着重讲本地缓存? 缓存有效期怎么做的?内部缓存机制的优化机制?如何防止内存、磁盘的缓存爆掉?

13. 什么情况下会发生 H5 白屏的情况?

14. 层序遍历也叫什么遍历，怎么实现

15. isa 指针里面都存了什么，32 和 64 位分别讲一下

- 32 位平台的 isa 指针指向其类对象；
- 64 位平台的 isa 指针为 non-pointer isa，不再”单纯“。

```
# if __arm64__
#   define ISA_MASK        0x0000000ffffffff8ULL
#   define ISA_MAGIC_MASK  0x000003f000000001ULL
#   define ISA_MAGIC_VALUE 0x000001a000000001ULL
#   define ISA_BITFIELD
          \
      uintptr_t nonpointer        : 1;
          \
      uintptr_t has_assoc         : 1;
          \
      uintptr_t has_cxx_dtor      : 1;
          \
      uintptr_t shiftcls          : 33; /*MACH_VM_MAX_ADDRESS 0x100
0000000*/ \
      uintptr_t magic             : 6;
          \
      uintptr_t weakly_referenced : 1;
          \
      uintptr_t deallocating      : 1;
          \
      uintptr_t has_sidetable_rc  : 1;
          \
      uintptr_t extra_rc          : 19
#   define RC_ONE   (1ULL<<45)
```

```
#   define RC_HALF  (1ULL<<18)

# elif __x86_64__
#   define ISA_MASK        0x00007ffffffffff8ULL
#   define ISA_MAGIC_MASK  0x001f800000000001ULL
#   define ISA_MAGIC_VALUE 0x001d800000000001ULL
#   define ISA_BITFIELD
            \
      uintptr_t nonpointer        : 1;
            \
      uintptr_t has_assoc         : 1;
            \
      uintptr_t has_cxx_dtor      : 1;
            \
      uintptr_t shiftcls          : 44; /*MACH_VM_MAX_ADDRESS 0x7ff
fffe00000*/ \
      uintptr_t magic             : 6;
            \
      uintptr_t weakly_referenced : 1;
            \
      uintptr_t deallocating      : 1;
            \
      uintptr_t has_sidetable_rc  : 1;
            \
      uintptr_t extra_rc          : 8
#   define RC_ONE   (1ULL<<56)
#   define RC_HALF  (1ULL<<7)

# else
#   error unknown architecture for packed isa
# endif
```

低 3 位标识是否为 nonpointer，是否有关联对象，以及是否有 c++ 的析构方法，紧接着 ARM 是 33 位就是类对象地址，X86 下是 44 位；后面还记录了是否有 weak 对象指向当前对象，是否正在 deallocing，另外就是 retaincount 会记录在 extra_rc， 若引用计数将要溢出时，extra_rc 降为一半，然后将 has_sidetable_rc 标识为 true。

```
ALWAYS_INLINE id
objc_object::rootRetain(bool tryRetain, bool handleOverflow)
{
    if (isTaggedPointer()) return (id)this;

    bool sideTableLocked = false;
    bool transcribeToSideTable = false;

    isa_t oldisa;
    isa_t newisa;
```

```
    do {
        transcribeToSideTable = false;
        oldisa = LoadExclusive(&isa.bits);
        newisa = oldisa;
        if (slowpath(!newisa.nonpointer)) {
            ClearExclusive(&isa.bits);
            if (!tryRetain && sideTableLocked) sidetable_unlock();
            if (tryRetain) return sidetable_tryRetain() ? (id)this :
 nil;
            else return sidetable_retain();
        }
        // don't check newisa.fast_rr; we already called any RR ove
rrides
        if (slowpath(tryRetain && newisa.deallocating)) {
            ClearExclusive(&isa.bits);
            if (!tryRetain && sideTableLocked) sidetable_unlock();
            return nil;
        }
        uintptr_t carry;
        newisa.bits = addc(newisa.bits, RC_ONE, 0, &carry);  // ext
ra_rc++

        if (slowpath(carry)) {
            // newisa.extra_rc++ overflowed
            if (!handleOverflow) {
                ClearExclusive(&isa.bits);
                return rootRetain_overflow(tryRetain);
            }
            // Leave half of the retain counts inline and
            // prepare to copy the other half to the side table.
            if (!tryRetain && !sideTableLocked) sidetable_lock();
            sideTableLocked = true;
            transcribeToSideTable = true;
            newisa.extra_rc = RC_HALF; // ⚠️⚠️ <<<<<<<< 这里降为一
半
            newisa.has_sidetable_rc = true; // ⚠️⚠️ <<<<<<< 打上标
记
        }
    } while (slowpath(!StoreExclusive(&isa.bits, oldisa.bits, newis
a.bits)));

    if (slowpath(transcribeToSideTable)) {
          //  ⚠️⚠️ 然后 sidetable 会加上 extra_rc/2 的值
        sidetable_addExtraRC_nolock(RC_HALF);
    }

    if (slowpath(!tryRetain && sideTableLocked)) sidetable_unlock();
    return (id)this;
}
```

16. ARC 下哪些情况会造成内存泄漏

- NSTimer、CADisplayLink 定时器内存泄露

17. AVFoundation 介绍

19. TCP 流量控制

29. PerformSelector & NSInvocation 优劣对比* ## 30. HTTPS 的握手过程

密钥协商过程：

1. 客户端将 TLS 版本，支持的加密算法，ClientHello random C 发给服务端【客户端->服务端】
2. 服务端从加密算法中 pick 一个加密算法， ServerHello random S，server 证书返回给客户端；【服务端->客户单】
3. 客户端验证 server 证书【客户端】
4. 客户端生成一个 48 字节的预备主密钥，其中前 2 个字节是 Protocol Version，后 46 个字节是随机数，客户端用证书中的公钥对预备主密钥进行非对称加密后通过 client key exchange 子消息发给服务端【客户端->服务端】
5. 服务端用私钥解密得到预备主密钥；【服务端】
6. 服务端和客户端都可以通过预备主密钥、ClientHello random C 和 ServerHello random S 通过 PRF 函数生成主密钥；会话密钥由主密钥、SecurityParameters.server_random 和 SecurityParameters.client_random 数通过 PRF 函数来生成会话密钥里面包含对称加密密钥、消息认证和 CBC 模式的初始化向量，对于非 CBC 模式的加密算法来说，就没有用到这个初始化向量。

Session ID 缓存和 Session Ticket 里面保存的也是主密钥，而不是会话密钥，这样每次会话复用的时候再用双方的随机数和主密钥导出会话密钥，从而实现每次加密通信的会话密钥不一样，即使一个会话的主密钥泄露了或者被破解了也不会影响到另一个会话。

31. 简述下 match-o 文件结构

32. CTMediator

33. 图片是什么时候解码的，如何优化

34. 隐式动画 & 显示动画区别

35. drawrect & layoutsubviews 调用时机

36. NSTimer、CADisplayLink、dispatch_source_t 的优劣

37. app 如何接收到触摸事件的

54. 读过 AFN 的源码吗？读过，那么有一个 security pin mode 是什么意思

55. 如何分析 linkmap

56. 勾住 C 语言的方法 

fishhook

57. 反编译

58. 音频降燥、视频合成

59. 一个项目中很多方法，把耗时的前几个方法找出来

60. 二叉树中增加节点

61. 检测主线程卡顿

62. 8 种锁，然后问的很细，为什么这个锁性能差，那个锁性能好

63. HTTPS 与 HTTP 的区别? 非对称加密、对称加密都是在哪一个步骤?

64. 如何实现一个网络监控? –追问–> 如果拦截 NSURLSession 具体会怎么做? 涉及哪些方法,方法名字说下?如何保证线程安全(并发场景下, 监控的线程和正常业务方的线程, 如何保证正常回调?) (考 N otification 的回调逻辑)

65. Swift、OC 如何相互调用? Swift–>OC 、OC –>Swift? 我开发一个 Swift 的 SDK,(API 都是 Swift 的), 内部需要调用到 OC 的库, 要怎么做?.

66. 动态库和静态库区别, 优缺点辨析? –追问–>包大小的差异的原因?为什么会有差异?

67. 如何解决静态库符号冲突的问题?

68. CoreAnimation 的实现原理?

69. 如何让 CoreAnimation 变得可交互? 比如让动画播放一半, 点击让他停止? 让他播放到 50%就停止播放? –追问–>基于你的方案, 请分析 CoreAnimation 内部相关接口实现原理(写出伪代码).

70. 我取消一个 CoreAnimation 动画? 到 50% 后, 我让他不再播放动画(不是暂停)?分析内部如何实现的?

71. 执行一个 NSThread 任务, 如何在执行过程中让他终止?

72. iOS NSOperation 是如何终止/取消任务的?

73. 解释一下离屏渲染, 什么场景下会出现?优化点在哪里?

74. 介绍下 Swizzle 的步骤? 具体到方法名.

75. Swizzle 时, 我不想替换父类, 只想替换子类,怎么办?

76. Swizzle 的优缺点? 缺点会导致什么问题?

77. 响应链: 如果 Swizzle 了 父 View 的 touchBegin 的方法, 会对子 View 造成什么影响?

78. GCD group 如何实现同步的? (还能用什么实现?)

79. 线程、队列的关系? 一个线程是否可能存在于两个队列?

80. 队列一定会创建线程吗?

81. 队列是否可以无限制创建?

82. 内存泄漏如何检测? 要求能具体到循环引用链条, 你用到的工具(比如:https://github.com/facebook/FBRetainCycleDetector)实现原理怎么做?

83. 讲一下组件化/SDK 中的接口设计规范有哪些?(比如: API break change 升级常见、预留字段、)

84. 公司有多个项目启动, 如何让接入的组件效率提升? (不局限于 iOS, 要讲前后端配合的方案, 偏重量级的方案)

85. WKWebView 如何解决 cookie 、跨域、方法拦截、等问题?

86. 自己做 SDK 如何解决与接入方的 SDK 版本冲突?(考虑包大小、研发人力、维护成本)

87. 组件化/SDK 怎么分层的、怎么封装的?如何协调不通部门的人去共享你的组件/SDK?(参数上下文、接口设计高度抽象)

88. 你们 hybrid APP 技术方案跨端方案(weex、RN、flutter)选择? 与 h5 hybrid 方案相比优缺点, 哪些场景需要用到 native、哪些页面 h5、哪些页面用跨端方案(weex、RN、flutter) ?

89. 如果你做的是一个超级 APP (微信、淘宝), 里面有一个引擎可以运行不同的小程序, 你如何设计保证小程序之间的安全性?

90. 介绍一下你们 APP 的架构设计

91. H5 与 跨端方案(weex、RN、flutter) 如何进行代码复用、适配?

92. OC 是否支持重载? 为什么?

93. 堆和栈的区别是什么?

94. 栈、堆分别是否会被线程所共享?

95. 内存空间中除了堆和栈还有什么内容?

96. HTTP 请求方法种类有哪些?(别忘记 HEAD)

97. RN、weex 前端页面 怎么渲染到 native 页面上的? (答案提示 node 节点树)

98. DNS、工作在什么层、默认端口?

99. SSL 里面怎么确保数据交换的过程、SSL 用了什么加密算法, 分别加密的是哪一个部分、为什么这么做 (提示: 对称加密、非对称加密都用到了, 对称加密传什么、非对称加密传什么东西)

100. 方法如果写了多个分类、会执行哪一个?执行逻辑是什么样?

101. IMP、SEL Method 都表示什么意思? 与 _cmd 相关 ## 102. weak 如何把 对象重制为 nil ## 103. assign、strong 区别, 是否能用 assign 修饰 NSObject? ## 104. 删除单链表的倒数第 K 个节点 ## 105. 方法交换和分类同时去 hook 同一个方法, 结果会怎么样? 具体交换的是什么? 交换时是如何处理传参数? 如果使用 NSInvocation 的话, 是否能处理方法有返回值的场景?具体怎么处理的? ## 106. 介绍 Weex/RN 的渲染原理? 与 Flutter 渲染的区别? 目录树的步骤是在哪里的? iOS/android 渲染出来的树是否一致? ## 107. Weex JS 运行环境是多个还是一个? ## 108. Weex/RN 与 H5、native 相比的优缺点? ## 109. Weex/RN 与 H5 白屏的原因? ## 110. 你认为 c++、与大前端相关的语言, 比如 objc、swfit、js 相比它的优缺点? ## 111. mutablearray 是怎么实现的，mutablearray 申请内存空间干什么用，做增删操作的时候内存空间是怎么改变的，可以用别的方法实现吗？ ## 112. [[NSMutableArray alloc] init] 和 [NSMutableArray array] 有什么区别

前者返回的数组实例对象是由调用者负责释放，后者是一个 authorelease 对象，最近的一个 autoreleasepool 作用域结束后会被释放

113. https 为什么能被抓包？抓到 https 的原理是什么

变种问题就是 charles 如何抓包，中间人攻击原理。

客户端和服务端通信，加入中间人这个第三者后，简单阐述几个知识点：

1. 中间人对于客户端来说就是一个”服务端“；而对于服务端则就是”客户端“角色；
2. 客户端将请求发送给中间人，中间人原封不动的把请求递交给服务端；服务端 response 给中间人，中间人在原封不动地将数据回给客户端，此刻中间人不过是个代理，仅仅只是充当了一个递交、转发的角色；
3. 对于 HTTP 明文传输，中间人自然可以查看，顺便说一句 DNS 查询时候，自然也是可以截获修改的，TCP/UDP 这种协议无法保证安全性；而对于 HTTPS 中间人就无法查看加密数据，因为客户端和服务端通信时候数据都是加密的（对称加密），密钥协商阶段是非对称加密；

4. 像 Charles/Fiddler 抓包原理实际上就是客户端要信任它们的证书（**这个是自签的根证书，有兴趣可以看下自建 CA 为服务器部署 https**），现在 **Charles 会为客户端访问的每个域名都用上面的根证书颁发一个证书**，当客户端和 Charles 通信时，先 TCP 三次握手建立连接，然后 SSL 四次握手阶段密钥协商，因为 Charles 自签的根证书已经被信任，所以它颁发的那些域名证书自然也是被信任的（有兴趣自己打开 chrome 的证书管理，确实 Charles 根证书在抓包时为每个域名都颁发了一个证书），所以在密钥协商阶段中的证书校验也是 OK 的，客户端和 Charles 实际上也是通信加密了，但是由于对称密钥就是 Charles 和客户端协商得到的，Charles 自己做加密、解密操作玩罢了。

114. 结构体中为什么不能使用 oc 对象

115. 我们在开发中使用文件的.mm 是基于什么原因?

.m 文件引用了 C++ 的文件。

116. 我们使用__bridge 来进行修饰是为了做什么？ ## 117. person 有个+test 方法，实现输出 persion test，student 继承 persion，头文件定义-test 方法，但没实现，student *obj=new student [obj test]结果是啥 ## 118. string 和 NSString 的区别 ## 119. dynamic 在 swift 与 OC 中的作用 ## 120. protobuf 的原理 ## 121. rn 与 flutter 的区别，flutter 组件渲染规则 ## 122. gcd 的使用，能不能取消？

iOS8 之后可以调用 dispatch_block_cancel 来取消（需要注意必须用 dispatch_block_create 创建 dispatch_block_t）

```
- (void)gcdBlockCancel{

    dispatch_queue_t queue = dispatch_queue_create("com.gcdtest.www", D
ISPATCH_QUEUE_CONCURRENT);

    dispatch_block_t block1 = dispatch_block_create(0, ^{
        sleep(5);
        NSLog(@"block1 %@",[NSThread currentThread]);
    });

    dispatch_block_t block2 = dispatch_block_create(0, ^{
        NSLog(@"block2 %@",[NSThread currentThread]);
    });

    dispatch_block_t block3 = dispatch_block_create(0, ^{
        NSLog(@"block3 %@",[NSThread currentThread]);
```

```
    });

    dispatch_async(queue, block1);
    dispatch_async(queue, block2);
    dispatch_block_cancel(block3);
}

// 输出日志
# block2 <NSThread: 0x600001bb4180>{number = 6, name = (null)}
# block1 <NSThread: 0x600001ba44c0>{number = 5, name = (null)}
```

自定义变量，然后在 block 中疯狂打标记，这里以 YYAsyncLayer 源码中取消上一次绘制为例：

```
if (async) {
        if (task.willDisplay) task.willDisplay(self);
        YYSentinel *sentinel = _sentinel;
        int32_t value = sentinel.value;
        BOOL (^isCancelled)(void) = ^BOOL() {
            return value != sentinel.value;
        };
                // .... 省略 ...
        dispatch_async(YYAsyncLayerGetDisplayQueue(), ^{
            if (isCancelled()) {
                CGColorRelease(backgroundColor);
                return;
            }
            UIGraphicsBeginImageContextWithOptions(size, opaque, scale);
            CGContextRef context = UIGraphicsGetCurrentContext();
            if (opaque) {
               // .... 省略 ...
            }
            task.display(context, size, isCancelled);
            if (isCancelled()) { // <---- 判断是否取消
                UIGraphicsEndImageContext();
                dispatch_async(dispatch_get_main_queue(), ^{
                    if (task.didDisplay) task.didDisplay(self, NO);
                });
                return;
            }
            UIImage *image = UIGraphicsGetImageFromCurrentImageContext
();
            UIGraphicsEndImageContext();
            if (isCancelled()) { // <---- 判断是否取消
                dispatch_async(dispatch_get_main_queue(), ^{
                    if (task.didDisplay) task.didDisplay(self, NO);
                });
                return;
            }
```

```
            dispatch_async(dispatch_get_main_queue(), ^{
                if (isCancelled()) { // <---- 判断是否取消
                    if (task.didDisplay) task.didDisplay(self, NO);
                } else {
                    self.contents = (__bridge id)(image.CGImage);
                    if (task.didDisplay) task.didDisplay(self, YES);
                }
            });
        });
    } else {
        // .... 省略 ...
    }
}
```

123. APP 架构师对什么指标比较关心?

124. 32 位系统和 64 位系统的本质区别是什么？

125. 动态库和静态库区别, 优缺点辨析? –追问–>包大小的差异的原因?为什么会有差异?

126. Swift、OC 如何相互调用? Swift–>OC 、OC –>Swift? 我开发一个 Swift 的 SDK,(API 都是 Swift 的), 内部需要调用到 OC 的库, 要怎么做?.

127. 什么情况下会发生 H5 白屏的情况?

128. OOM (Out Of Memory) 类型的 crash 介绍下, 怎么检测, 怎么处理?

129. websocket 协议 与 MQTT 协议的区别? MQTT 是否支持搭配 websocket 实现聊天功能?

130. 弱网情况下 IM 即时通讯的优化? 比如: 网络抖动.

131. 什么是 SNI? SNI 是什么的缩写? iOS 上想设置 SNI , 怎么设?

算法 / Coding

9. 堆排序、归并排序、快排原理，优缺点

堆排序：

归并排序:

快排：

12. 请手写一份 LRU 实现 (要求先介绍 LRU 实现, LRU 如何 key-value 的数据进行处理)

```
type LRUCache struct {
    size,capacity int
  cache map[int] *LinkedNode
  head,tail *LinkedNode
}
type LinkedNode struct {
  key, value int
  prev, next *LinkedNode
}

func initLinkedNode(key, value int)*LinkedNode {
  return &LinkedNode {
    key:key,
    value:value,
  }
}

func Construct(capacity int)LRUCache {
  l := LRUCache {
    size: 0,
    capacity:capacity,
    cache:map[int]*LinkedNode {},
    head:initLinkedNode(0,0),
    tail:initLinkedNode(0,0)
  }
  l.head.next = l.tail
  l.tail.prev = l.head
  return l
}

func (this *LRUCache)removeNode(node *LinkedNode){
  node.prev.next = node.next
  node.next.prev = node.prev
}

func (this *LRUCache)addTohead(node *LinkedNode){
  node.prev = this.head
  node.next = this.head.next
  this.head.next.prev = node
  this.head.next = node
}

func (this *LRUCache)moveTohead(node *LinkedNode){
  this.removeNode(node)
  this.addTohead(node)
}
```

```go
func (this *LRUCache) Get(ket int)int {
  if _,ok := this.cache[key]; !ok {
    return -1;
  }
  node := this.cache[key]
  this.moveTohead(node)
  return node.value
}

func (this *LRUCache) removeTail() *LinkedNode{
    node := this.tail.prev
    this.removeNode(node)
    return node
}

func (this *LRUCache) put(key, value int){
  if _,ok := this.cache[key]; !ok{
    node := initLinkedNode(key,value);
    this.cache[key] = node
    this.addTohead(node)
    this.size++
    if this.size > this.capacity {
      removed := this.removeTail()
      delete(this.cache, remove.key)
      this.size--
    }
  } else {
    node := this.cache[key]
    node.value = value
    this.moveTohead(node)
  }
}
```

18. 给定一字符串只包含数字，请写一个算法，找出该字符串中的最长不重复子串的长度（不重复是指子串中每一元素不同于子串中其他元素）如：“120135435”最长不重复子串

22. 给你一个数组和一个目标值，从数组中找到三个值，使其和最接近目标值。

23. 给你一万亿个数据，让你取出前一百万个数据

24. KMP 算法

25. 如果最高效的计算 17 * 2？

26. 找到链表的倒数第 k 个结点？

27. 10 亿个数中找最大的 1000 个数

28. 合并有序链表

38. 两个无限长度链表（也就是可能有环） 判断有没有交点

39. 二维有序数组查找数字

40. 把 “www.zhidao.baidu.com” 这样的字符串改成 “com/baidu/zhidao/www”。——老题目了，剑指 offer 的，两次逆序排列即可。

41. 两个链表中间交于某个节点，求这个结点。

42. 找出一个字符串中只出现一次且是第一个的字符

43. 最长公共前缀

44. 数据流中的第 K 大元素

45. 滑动窗口最大值

46. 前 K 个高频单词

47. 两数之和

48. 有效的字母异位词

49. 找链表的倒数第 k 个结点

50. 把一个链表比某个值大的放在左边，比它小的放在右边

51. 反转链表

52. 环形链表

53. 反转字符串

132. 排序算法, 字母和数字排序, 字母优先级高于数字: abc123.

面试集合二

1. 结构体的字节对齐和 OC 对象的字节对齐？

2. instance（实例对象）、class（类对象）、meta-class（元类对象）分别储存了什么信息？为什么要设计元类？

3. KVO 的具体实现流程？访问成员变量（类似 self->age）会触发 KVO 嘛？KVC 会触发 KVO 嘛？KVO 的两个核心调用方法是？

4. KVC 的原理？getter 和 setter 的搜索策略是什么？KVC 有什么实际的应用？

5. Catagory 和 extension 分别的使用场合和特点是什么？

6. Catagory 的实现原理是什么？Catagory 有哪些用处？Catagory 有什么局限？

7. Class 和 他的 Catagory 同名方法的调用顺序是什么？Catagory A 和 Catagory B 同名方法的调用顺序是如何？如果想要不按照系统顺序执行要怎么做？

8. +load 和 +initialize 的调用时机和顺序？两者区别是什么？

9. Catagory 有 +load 方法么？+load 是什么时候调用的？能继承么？会覆盖 Class 的 +load 么？

10. Catagory 关联对象（AssociateObject）的底层实现是什么？

11. block 的本质是什么？block 的底层实现是怎样的？block 的变量捕获是什么原理？block 的类型有哪些？什么情况下会把栈上的 block 赋值到堆上？

（block 我真的好多都没记住啊…）

12. isa 指针是什么？里面有哪些特殊的位数？什么是 TaggedPointer 的优化？

13. class 的底层结构是什么样的？

14. method_t 里包含什么？

15. super 的本质是什么？

16. OC 的消息机制有几步？

17. 如何防止类似 unrecognized selector 的错误？_objc_msgForward 能干什么？ ### 18. runtime 有哪些应用？方法替换（method - Swizzling）有什么缺点？如何安全的进行方法替换？ ### 19. RunLoop 的本质是什么？ ### 20. Runloop 和线程是什么关系？ ### 21. Runloop 的底层数据结构是什么样的？有几种 运行模式（mode）？每个运行模式下面的 CFRunloopMode 是哪些？他们分别是什么职责？ ### 22. Runloop 的监听状态有哪几种？ ### 23. Runloop 的工作流程大概是什么样的？ ### 24. Runloop 有哪些应用？ ### 25. 多线程，异步执行（async）一个 performSelector 会执行么？如果加上 afterDelay 呢？ ### 26. 你知道 iOS 有哪些锁？性能分别怎么样？ ### 27. 自旋锁和互斥锁怎么选择？ ### 28. 引用计数怎么实现的？weak 怎么实现的？sideTable 的 底层结构是怎么样的？ weak 指针做了什么操作？ ### 29. AutoReleasePool（自动释放池） 的底层实现是什么？他怎么实现及时释放的？子线程的释放时机是怎么样的？ ### 30. 对象的 release 是怎么处理的？