
Table of Contents

Introduction	1.1
iOS	1.2
list	1.2.1
基础知识	1.2.1.1
原理知识	1.2.1.2
UI相关知识	1.2.1.3
网络知识	1.2.1.4
就业形式	1.2.1.5
知识结构图	1.2.1.6
工作指导	1.2.1.7
其他	1.2.1.8
我遇到的面试题	1.2.2

2018KEYWORD:空杯，进化，落地。

危险总是存在的，但恐惧只是一种选择。

To make complicated things simplified.（复杂问题简单化。） —— Jeff Dean

人生就像一场大火，我们每个人唯一可以做的就是从这场大火中多抢救一些东西出来。—— Bill Gates

大家游的速度其实不相上下，但是有的人更会减少阻力。

时间换钱，钱换时间，剥削永恒。

程序员薪资组成：公司职级，技术壁垒，面试结果，稀缺程度，Offer竞争。

温水煮青蛙。



本地使用或者启动：

```
gitbook serve
```

iOS基础篇

=====

block和 weak修饰符的区别?

- __block不管是ARC还是MRC模式下都可以使用，可以修饰对象，也可以修饰基本数据类型
- __weak 只能在ARC模式下使用，只能修饰对象（NSString），不能修饰基本数据类型
- block修饰的对像可以在block中被重新赋值，weak修饰的对像不可以

什么情况使用weak关键字，相比assign有什么不同？

- 首先明白什么情况使用weak关键字?
 - 在ARC中，在有可能出现循环引用的时候，往往要通过让其中一端使用weak来解决，比如：delegate代理属性，代理属性也可使用assign
 - 自身已经对它进行一次强引用，没有必要再强引用一次，此时也会使用weak，自定义IBOutle控件属性一般也使用weak；当然，也可以使用strong，但是建议使用weak
- weak 和assign的不同点
 - weak策略在属性所指的对象遭到摧毁时，系统会将weak修饰的属性对象的指针指向nil，在OC给nil发消息是不会有什么问题的；如果使用assign策略在属性所指的对象遭到摧毁时，属性对象指针还指向原来的对象，由于对象已经被销毁，这时候就产生了野指针，如果这时候在给此对象发送消息，很容易造成程序崩溃
 - assign 可以用于修饰非OC对象，而weak必须用于OC对象

引申点：weak 修饰的对像释放逻辑。

堆和栈的区别

- 从管理方式来讲
 - 对于栈来讲，是由编译器自动管理，无需我们手工控制；
 - 对于堆来说，释放工作由程序员控制，容易产生内存泄露(memory leak)
- 从申请大小方面讲
 - 栈空间比较小
 - 堆空间比较大
- 从数据存储方面来讲
 - 栈空间中一般存储基本类型，对象的地址
 - 堆空间一般存放对象本身，block的copy等

引申点：内存区的知识点（堆，栈，常量区，静态区）

风格纠错题

```

typedef enum{
    UserSex_Man,
    UserSex_Woman
}UserSex;

@interface UserModel :NSObject

@property(nonatomic, strong) NSString *name;
@property (assign, nonatomic) int age;
@property (nonatomic, assign) UserSex sex;

- (id)initWithUserName: (NSString*)name withAge:(int)age;

- (void)doLogIn;

@end

```

- 修改后的代码

```

```bash
typedef NS_ENUM(NSInteger, CYLSex) { CYLSexMan, CYLSexWoman };

@interface CYLUser : NSObject

@property (nonatomic, copy, readonly) NSString *name; @property (nonatomic, assign, readonly) NSUInteger age; @property (nonatomic, assign, readwrite) CYLSex sex;

- (instancetype)initWithName:(NSString *)name age:(NSUInteger)age sex:(CYLSex)sex;
- (instancetype)initWithName:(NSString *)name age:(NSUInteger)age;
- (instancetype)userWithName:(NSString *)name age:(NSUInteger)age sex:(CYLSex)sex;

@end
```

```

Objective-C使用什么机制管理对象内存？

- MRC 手动引用计数
- ARC 自动引用计数,现在通常ARC
- 通过retainCount 的机制来决定对象是否需要释放。每次runloop 的时候，都会检查对象的retainCount，如果retainCount 为0，说明该对象没有地方需要继续使用了，可以释放掉了

引申点：runloop知识点，或者autoreleasepool知识点

ARC通过什么方式帮助开发者管理内存？或者“ARC什么时机做了什么事情？”

- 通过编译器在编译的时候,插入类似内存管理的代码

ARC下还会存在内存泄露吗？

1. 循环引用会导致内存泄露
2. Objective-C对象与CoreFoundation对象进行桥接的时候如果管理不当也会造成内存泄露
3. CoreFoundation中的对象不受ARC管理，需要开发者手动释放

引申点：请举例 CoreFoundation 的几个类

@property 的本质是什么？@dynamic 与@synthesize 区别

- @property其实就是在编译阶段由编译器自动帮我们生成ivar成员变量，getter方法，setter方法
- @dynamic 手动设置setter和getter方法。如果不设置，就会crash
- @synthesize 编译自动设置setter和getter方法

@property后面可以有哪些修饰符？

- 基本数据类型，默认关键字为：
 - atomic
 - readonly
 - assign
- 对应对象类型，默认关键字为：
 - atomic
 - readonly
 - strong
- 方法名--
 - -getter=
 - setter=
- 其他关键词
 - 非原子操作 nonatomic
 - 读写权限readonly, readonly
 - 内存管理 assign, strong, weak, unsafe_unretained, copy
 - 方法名 getter,setter
 - 不常用的nonnull, null_resettable, nullable

引申点：atomic与nonatomic区别，什么时候用atomic。

使用atomic一定是线程安全的吗？

- 不是，atomic的本意是指属性的存取方法是线程安全的，并不保证整个对象是线程安全的。
- 举例：声明一个NSMutableArray的原子属性stuff，此时self.stuff和self.stuff = othersulf都是线程安全的。但是，使用[self.stuff objectAtIndex:index]就不是线程安全的，需要用互斥锁来保证线程安全性

怎么用copy关键字？

- NSString、NSArray、NSDictionary等等经常使用copy关键字，是因为他们有对应的可变类型：NSMutableString、NSMutableArray、NSMutableDictionary，为确保对象中的属性值不会无意间变动，应该在设置新属性值时拷贝一份，保护其封装性
- block也经常使用copy关键字

- block 使用copy 是从MRC 遗留下来的“传统”,在MRC 中,方法内部的block 是在栈区的,使用copy 可以把它放到堆区.
- 在ARC中写不写都行: 对于block使用copy还是strong效果是一样的, 但是建议写上copy, 因为这样显示告知调用者“编译器会自动对block进行了copy操作”

引申点: block的相关知识, copy内存的相关知识及使用场景。

copy知识点

- 浅复制(shallow copy): 在浅复制操作时, 对于被复制对象的每一层都是指针复制。
- 深复制(one-level-deepcopy): 在深复制操作时, 对于被复制对象, 至少有一层是深复制。
- 完全复制(real-deepcopy): 在完全复制操作时, 对于被复制对象的每一层都是对象复制。
- 非集合类对象的copy与mutableCopy
 - [不可变对象 copy] // 浅复制
 - [不可变对象 mutableCopy] //深复制
 - [可变对象 copy] //深复制
 - [可变对象 mutableCopy] //深复制
- 集合类对象的copy与mutableCopy
 - [不可变对象 copy] // 浅复制
 - [不可变对象 mutableCopy] //单层深复制
 - [可变对象 copy] //单层深复制
 - [可变对象 mutableCopy] //单层深复制
- 这里需要注意的是集合对象的内容复制仅限于对象本身, 对象元素仍然是指针复制

引申点: 给可变或者不可变数组, 字典, 等, 问内容是否两个内容是否一致

这个写法会出什么问题: @property (copy) NSMutableArray *array;

- 因为copy策略拷贝出来的是一个不可变对象, 然而却把它当成可变对象使用, 很容易造成程序奔溃
- 这里还有一个问题, 该属性使用了同步锁, 会在创建时生成一些额外的代码用于帮助编写多线程程序, 这会带来性能问题, 通过声明nonatomic可以节省这些虽然很小但是不必要额外开销, 在iOS开发中应该使用nonatomic替代atomic

如何让自定义类可以用copy 修饰符? 如何重写带copy 关键字的setter? 或者“如何使自己的对象具有copy属性”

- 若想令自己所写的对象具有拷贝功能, 则需实现NSCopying协议。如果自定义的对象分为可变版本与不可变版本, 那么就要同时实现NSCopyig与NSMutableCopying协议, 不过一般没什么必要, 实现NSCopying协议就够了

```
// 实现不可变版本拷贝
- (id)copyWithZone:(NSZone *)zone;

// 实现可变版本拷贝
- (id)mutableCopyWithZone:(NSZone *)zone;

// 重写带 copy 关键字的 setter
- (void)setName:(NSString *)name
```

```
{
    _name = [name copy];
}
```

引申点：如何使自己的对象具有归档功能？

+**(void)load;** +**(void)initialize;**有什么用处？

- **+*(void)load;***
 - 当类对象被引入项目时, runtime 会向每一个类对象发送load 消息
 - load方法会在每一个类甚至分类被引入时仅调用一次,调用的顺序：父类优先于子类,子类优先于分类
 - 由于load 方法会在类被import 时调用一次,而这时往往是改变类的行为的最佳时机, 在这里可以使用例如 method swizzling 来修改原有的方法
 - load方法不会被类自动继承
- **+*(void)initialize;***
 - 也是在第一次使用这个类的时候会调用这个方法, 也就是说initialize也是懒加载
- 总结：
 - 在Objective-C中, runtime会自动调用每个类的这两个方法
 - +load会在类初始加载时调用
 - +initialize会在第一次调用类的类方法或实例方法之前被调用
 - 这两个方法是可选的, 且只有在实现了它们时才会被调用
 - 两者的共同点：两个方法都只会被调用一次

Foundation对象与Core Foundation对象有什么区别

- Foundation框架是使用OC实现的, Core Foundation是使用C实现的
- Foundation对象和Core Foundation对象间的转换：俗称桥接
- ARC环境桥接关键字：

```
// 可用于Foundation对象 和 Core Foundation对象间的转换
__bridge

// 用于Foundation对象 转成 Core Foundation对象
__bridge_retained

// Core Foundation对象 转成 Foundation对象
__bridge_transfer
```

Foundation对象转成Core Foundation对象

- 使用__bridge桥接
- 如果使用bridge桥接,它仅仅是将strOC的地址给了strC,并没有转移对象的所有权, 也就是说,如果使用bridge桥接,那么如果strOC释放了,strC也不能用了
- 注意:在ARC条件下,如果是使用__bridge桥接,那么strC 可以不用主动释放,因为ARC会自动管理strOC和strC

```
NSString *strOC1 = [NSString stringWithFormat:@"abcdefg"];
CFStringRef strC1 = (__bridge CFStringRef)strOC1;
NSLog(@"%@", strOC1, strC1);
```

- 使用`__bridge_released`桥接

- 如果使用`__bridge_released`桥接, 它会将对象的所有权转移给`strC`, 也就是说, 即便`strOC`被释放了, `strC`也可以使用
- 注意: 在ARC条件下, 如果是使用`__bridge_released`桥接, 那么`strC`必须自己手动释放, 因为桥接的时候已经将对象的所有权转移给了`strC`, 而C语言的东西不是不归ARC管理的

```
NSString *strOC2 = [NSString stringWithFormat:@"abcdefg"];
// CFStringRef strC2 = (__bridge_released CFStringRef)strOC2;
CFStringRef strC2 = CFBridgingRetain(strOC2); // 这一句,
就等同于上一句
CFRelease(strC2);

Core Foundation对象转成Foundation对象
```

- 使用`__bridge`桥接

- 如果使用`__bridge`桥接, 它仅仅是将`strC`的地址给了`strOC`, 并没有转移对象的所有权
- 也就是说如果使用`__bridge`桥接, 那么如果`strC`释放了, `strOC`也不能用了

```
CFStringRef strC3 = CFStringCreateWithCString(CFAllocatorGetDefault(), "12345678", kCFStringEncodingASCII);
NSString *strOC3 = (__bridge NSString *)strC3;
CFRelease(strC3);
```

- 使用`__bridge_transfer`桥接

- 如果使用`__bridge_transfer`桥接, 它会将对象的所有权转移给`strOC`, 也就是说, 即便`strC`被释放了, `strOC`也可以使用
- 如果使用`__bridge_transfer`桥接, 他会自动释放`strC`, 也就是以后我们不用手动释放`strC`

```
CFStringRef strC4 = CFStringCreateWithCString(CFAllocatorGetDefault(), "12345678", kCFStringEncodingASCII);
// NSString *strOC = (__bridge_transfer NSString *)strC;
NSString *strOC4 = CFBridgingRelease(strC4); // 这一句,
```

就等同于上一句 MRC环境: 直接强转

```
-(void)bridgeInMRC
{
    // 将Foundation对象转换为Core Foundation对象, 直接强制类型转换即可
    NSString *strOC1 = [NSString stringWithFormat:@"xxxxxx"];
    CFStringRef strC1 = (CFStringRef)strOC1;
    NSLog(@"%@", strOC1, strC1);
    [strOC1 release];
    CFRelease(strC1);

    // 将Core Foundation对象转换为Foundation对象, 直接强制类型转换即可
    CFStringRef strC2 = CFStringCreateWithCString(CFAllocatorGetDefault(), "12345678", kCFStringEncodingASCII);
    NSString *strOC2 = (NSString *)strC2;
    NSLog(@"%@", strOC2, strC2);
    [strOC2 release];
    CFRelease(strC2);
}
```

addObserver:forKeyPath:options:context:各个参数的作用

用分别是什么， observer中需要实现哪个方法才能获得KVO回调？

```
// 注册要监听的对象
[self.person addObserver:self forKeyPath:@"name" options:NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld context:@"Person Name"];

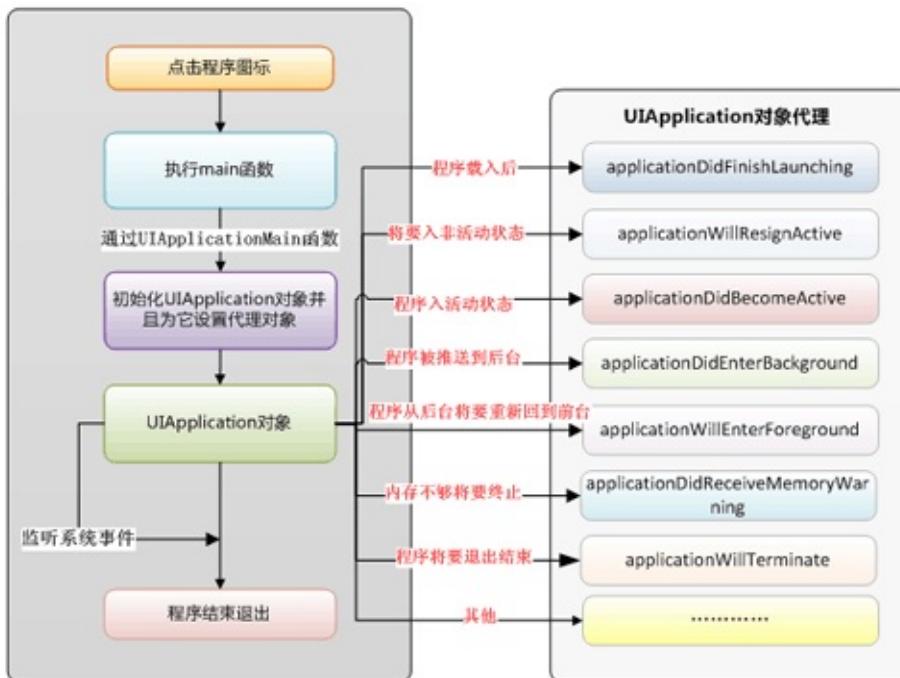
// 当监控的某个属性的值改变了就会调用
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void *)context{

}
```

NSCache优于NSDictionary的几点？

- 1.nscache 是可以自动释放内存的。
- 2.nscache是线程安全的， 我们可以在不同的线程中添加， 删除和查询缓存中的对象。
- 3.一个缓存对象不会拷贝key对象。

说一下AppDelegate的几个方法？从后台到前台调用了哪些方法？第一次启动调用了哪些方法？从前台到后台调用了哪些方法？



数据持久化的几个方案

- plist, 存储字典， 数组比较好用
- preference: 偏好设置， 实质也是plist

- NSKeyedArchiver: 归档, 可以存储对象
- sqlite: 数据库, 经常使用第三方来操作, 也就是fmdb
- coreData:也是数据库储存, 苹果官方的

程序性能的优化

1. 使用复用机制
2. 尽可能设置 View 为不透明
3. 避免臃肿的 XIB 文件
4. 不要阻塞主线程
5. 图片尺寸匹配 UIImageView
6. 选择合适的容器
7. 启用 GZIP 数据压缩
8. View 的复用和懒加载机制
9. 缓存服务器的响应信息 (response) 。图片。计算值。比如: UITableView 的 row heights。
10. 关于图形绘制
11. 处理 Memory Warnings
 - 在 AppDelegate 中实现 - [AppDelegate applicationDidReceiveMemoryWarning:] 代理方法。
 - 在 UIViewController 中重载 didReceiveMemoryWarning 方法。
 - 监听 UIApplicationDidReceiveMemoryWarningNotification 通知。
12. 复用高开销的对象
13. 减少离屏渲染(设置圆角和阴影的时候可以选用绘制的方法)
14. 优化 UITableView
 - 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。
15. 选择合适的数据存储方式
 - 在 iOS 中可以用来进行数据持久化的方案包括:
 - NSUserDefaults。只适合用来存小数据。
 - XML、JSON、Plist 等文件。JSON 和 XML 文件的差异在「选择正确的数据格式」已经说过了。
 - 使用 NSCoding 来存档。NSCoding 同样是对文件进行读写, 所以它也会面临必须加载整个文件才能继续的问题。
 - 使用 SQLite 数据库。可以配合 FMDB 使用。数据的相对文件来说还是好处很多的, 比如可以按需取数据、不用暴力查找等等。
 - 使用 CoreData。也是数据库技术, 跟 SQLite 的性能差异比较小。但是 CoreData 是一个对象图谱模型, 显得更面向对象; SQLite 就是常规的 DBMS。
16. 减少应用启动时间
 - 快速启动应用对于用户来说可以留下很好的印象。尤其是第一次使用时。
 - 保证应用快速启动的指导原则:
 - 尽量将启动过程中的处理分拆成各个异步处理流, 比如: 网络请求、数据库访问、数据解析等等。
 - 避免臃肿的 XIB 文件, 因为它们会在你的主线程中进行加载。重申: Storyboard 没这个问题, 放心使用。
 - 注意: 在测试程序启动性能的时候, 最好用与 Xcode 断开连接的设备进行测试。因为 watchdog 在使用

Xcode 进行调试的时候是不会启动的。

17. 使用 Autorelease Pool (内存释放池)
18. imageNamed 和 imageWithContentsOfFile

=====

Block 相关知识

使用block时什么情况会发生引用循环，如何解决？

在block内如何修改block外部变量？

对外部变量进行 `__block` 修饰

使用系统的某些block api（如UIView的block版本写动画时），是否也考虑循环引用问题？

- 系统的某些block api中， UIView的block版本写动画时不需要考虑，但也有一些api 需要考虑

以下这些使用方式不会引起循环引用的问题

```
[UIView animateWithDuration:duration animations:^{
    [self.superview layoutIfNeeded];
}];

[[NSOperationQueue mainQueue] addOperationWithBlock:^{
    self.someProperty = xyz;
}];

[[NSNotificationCenter defaultCenter] addObserverForName:@"someNotification"
                                             object:nil
                                             queue:[NSOperationQueue mainQueue]
                                             usingBlock:^(NSNotification * notification)
    { self.someProperty = xyz; }];
}
```

- 但如果方法中的一些参数是成员变量，那么可以造成循环引用，如GCD、NSNotificationCenter调用就要小心一点，比如GCD 内部如果引用了self，而且GCD 的参数是成员变量，则要考虑到循环引用，举例如下：

- GCD
 - 分析： `self->_operationsQueue-->block-->self`形成闭环，就造成了循环引用。修改为

```
__weak __typeof__(self) weakSelf = self;
dispatch_group_async(_operationsGroup, _operationsQueue, ^{
{
    [weakSelf doSomething];
    [weakSelf doSomethingElse];
} );
```

- NSNotificationCenter

- 分析： `self->_observer-->block-->self`形成闭环，就造成了循环引用

```
__weak __typeof__(self) weakSelf = self;
_observer = [[NSNotificationCenter defaultCenter]
addObserverForName:@"testKey"
object:nil
queue:nil
```

```
usingBlock:^(NSNotification *note){
    [weakSelf dismissModalViewControllerAnimated:YES];
});
```

GCD原理

- GCD的工作原理是：让程序平行排队的特定任务，根据可用的处理资源，安排他们在任何可用的处理器核心上执行任务。
- 一个任务可以是一个函数(function)或者是一个block。GCD的底层依然是用线程实现，不过这样可以让程序员不用关注实现的细节。
- GCD中的FIFO队列称为dispatch queue，它可以保证先进来的任务先得到执行。

GCD的队列 (`dispatch_queue_t`) 分哪两种类型？背后的线程模型是什么样的？

- 串行队列
- 并行队列
- `dispatch_global_queue()`;是全局并发队列
- `dispatch_main_queue()`;是一种特殊串行队列
- 背后的线程模型：自定义队列`dispatch_queue_t queue`;可以自定义是并行：`DISPATCH_QUEUE_CONCURRENT` 或者串行`DISPATCH_QUEUE_SERIAL`

如何用GCD同步若干个异步调用？（如根据若干个url异步加载多张图片，然后在都下载完成后合成一张整图）

- 必须是并发队列才起作用
- 需求分析
 - 首先，分别异步执行2个耗时的操作
 - 其次，等2个异步操作都执行完毕后，再回到主线程执行一些操作
- 使用队列组实现上面的需求

```
// 创建队列组
dispatch_group_t group = dispatch_group_create();

// 获取全局并发队列
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

// 往队列组中添加耗时操作
dispatch_group_async(group, queue, ^{
    // 执行耗时的异步操作1
});

// 往队列组中添加耗时操作
dispatch_group_async(group, queue, ^{
    // 执行耗时的异步操作2
});

// 当并发队列组中的任务执行完毕后才会执行这里的代码
dispatch_group_notify(group, queue, ^{
    // 如果这里还有基于上面两个任务的结果继续执行一些代码，建议还是放到子线程中，等代码执行完毕后在回到主线程
});
```

```
// 回到主线程
dispatch_async(group, dispatch_get_main_queue(), ^{
    // 执行相关代码...
});
});
```

dispatch_barrier_async的作用是什么？

barrier(屏障)

- 函数定义

```
dispatch_barrier_async(dispatch_queue_t queue, dispatch_block_t block);
```

- 必须是并发队列，要是串行队列，这个函数就没啥意义了
- 注意：这个函数的第一个参数queue 不能是全局的并发队列
- 作用：在它前面的任务执行结束后它才执行，在它后面的任务等它执行完成后才会执

```
-(void)barrier
{
    dispatch_queue_t queue = dispatch_queue_create("12342234", DISPATCH_QUEUE_CONCURRENT);

    dispatch_async(queue, ^{
        NSLog(@"----1----%@", [NSThread currentThread]);
    });
    dispatch_async(queue, ^{
        NSLog(@"----2----%@", [NSThread currentThread]);
    });

    // 在它前面的任务执行结束后它才执行，在它后面的任务等它执行完成后才会执行
    dispatch_barrier_async(queue, ^{
        NSLog(@"----barrier----%@", [NSThread currentThread]);
    });

    dispatch_async(queue, ^{
        NSLog(@"----3----%@", [NSThread currentThread]);
    });
    dispatch_async(queue, ^{
        NSLog(@"----4----%@", [NSThread currentThread]);
    });
}
```

GCD的一些常用的函数？ (group, barrier, 信号量, 线程同步)

- group:

- 我们使用队列组来开辟线程时，队列组中的队列任务是并发，当所有的队列组中的所有任务完成时候，才可以调用队列组完成任务。

```
/**创建自己的队列*/
dispatch_queue_t dispatchQueue = dispatch_queue_create("ted.queue.next", DISPATCH_QUEUE_CONCURRENT);
/**创建一个队列组*/
dispatch_group_t dispatchGroup = dispatch_group_create();
/**将队列任务添加到队列组中*/
dispatch_group_async(dispatchGroup, dispatchQueue, ^{
    NSLog(@"dispatch-1");
});
```

```

});
    /*将队列任务添加到队列组中*/
dispatch_group_async(dispatchGroup, dispatchQueue, ^{
    NSLog(@"dpatch-2");
});
    /*队列组完成调用函数*/
dispatch_group_notify(dispatchGroup, dispatch_get_main_queue(), ^{
    NSLog(@"end");
})

```

- **barrier:**

- 表示栅栏，当在并发队列里面使用栅栏时候，栅栏之前的并发任务开始并发执行，执行完毕后，执行栅栏内的任务，等栅栏任务执行完毕后，再并发执行栅栏后的任务。

```

dispatch_queue_t concurrentQueue = dispatch_queue_create("my.concurrent.queue", DISPATCH_QUEUE_CONCURRENT);
dispatch_async(concurrentQueue, ^{
    NSLog(@"dispatch-1");
});
dispatch_async(concurrentQueue, ^{
    NSLog(@"dispatch-2");
});
dispatch_barrier_async(concurrentQueue, ^{
    NSLog(@"dispatch-barrier");
});
dispatch_async(concurrentQueue, ^{
    NSLog(@"dispatch-3");
});
dispatch_async(concurrentQueue, ^{
    NSLog(@"dispatch-4");
});

```

- **信号量:**

- Semaphore是通过'计数'的方式来标识线程是否是等待或继续执行的。信号量

```

dispatch_semaphore_create(int) // 创建一个信号，并初始化信号的计数大小
/* 等待信号，并且判断信号量，如果信号量计数大于等于你创建时候的信号量的计数，就可以通过，继续执行，并且将你传入的信号量减1,
 * 如果传入的信号量计数小于你创建的计数，就表示等待，等待信号量的变化
 * 如果等待的时间超过你传入的时间，也会继续下面操作
 * 第一个参数：semaphore 表示信号量
 * 第二个参数：表示等待的时间
 * 返回int 如果传入的信号量计数大于等于你创建信号的计数时候，返回0. 反之，返回的不等于0
 */
int result = dispatch_semaphore_wait(dispatch_semaphore_t semaphore, time outTime); // 表示等待，也是阻碍线程
// 表示将信号量+1
dispatch_semaphore_signal(dispatch_semaphore_t semaphore);

- (void)semaphoreDemo1 {
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(10);
    for (int i = 0; i < 100; i++) {
        // 因为是异步执行的，所以每次循环Block里面的dispatch_semaphore_signal根本还没有执行就会执行dispatch_semaphore_wait,
        // 从而semaphore-1. 当循环10次后，semaphore等于0，则会阻塞线程，直到执行了Block的dispatch_semaphore_signal 才会继续执行
        NSLog(@"i %zd", i);
        // 执行十次之后阻塞当前线程
        dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER);
    }
}

- (void)semaphoreDemo2 {
    dispatch_semaphore_t goOnSemaphore = dispatch_semaphore_create(0);
    NSLog(@"ready");
    [self network:^(id result) {
        NSLog(@"net return:%@", result);
        dispatch_semaphore_signal(goOnSemaphore);
    }];
}

```

```

    }];
    dispatch_semaphore_wait(goOnSemaphore, DISPATCH_TIME_FOREVER);
    NSLog(@"go on");
}
- (void)network:(void(^)(id result))block {
    sleep(2.0);
    block(@(arc4random_uniform(2)));
}

```

以下代码运行结果如何？

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    NSLog(@"1");
    dispatch_sync(dispatch_get_main_queue(), ^{
        NSLog(@"2");
    });
    NSLog(@"3");
}

```

- 答案：主线程死锁

iOS 程序 main 函数之前发生了什么

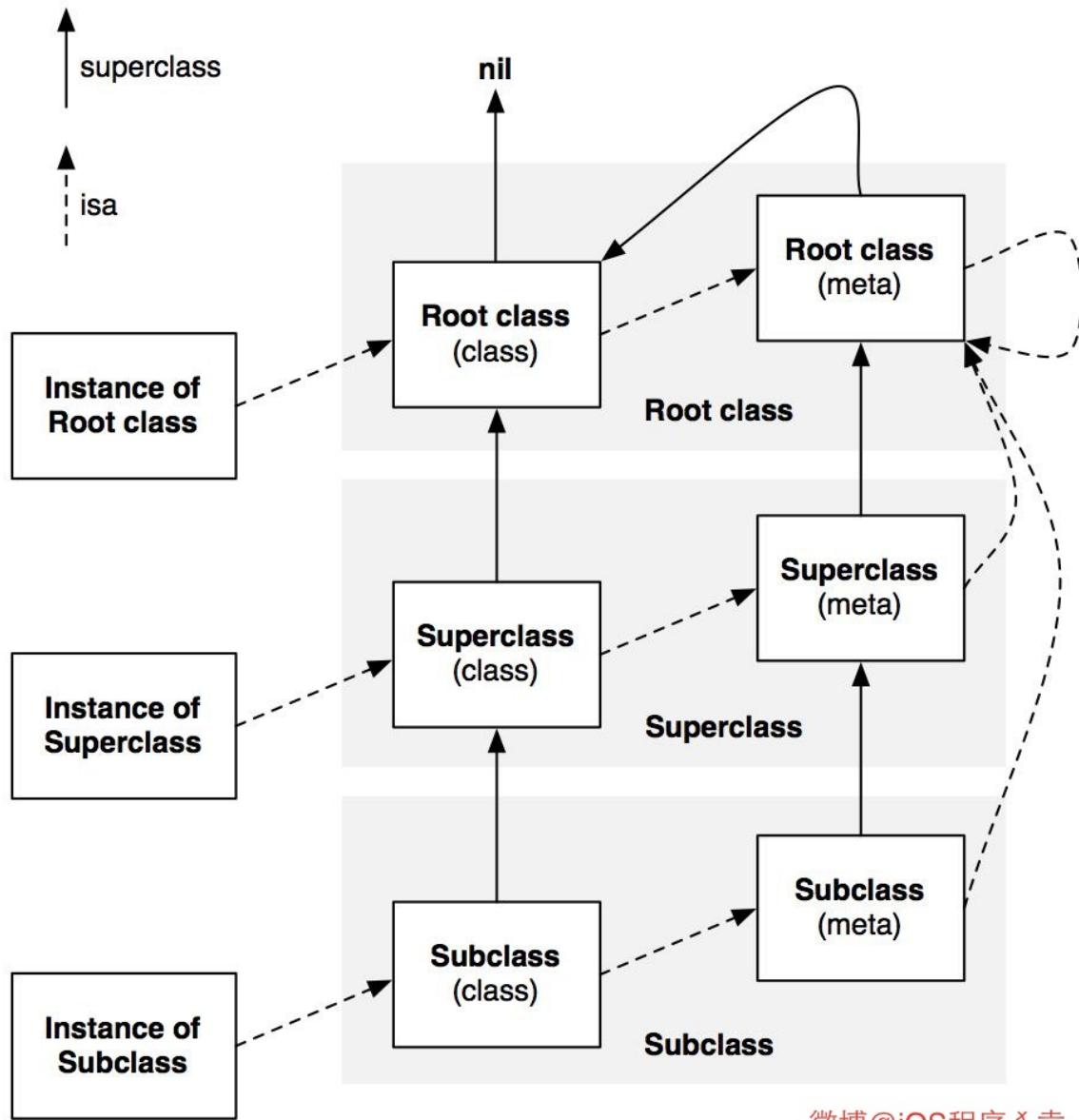
- 1.加载动态链接库(dyld)
- 2.加载图片资源符号(ImageLoader)
- 3.runtime+load

一个Objective-C对象如何进行内存布局？（考虑有父类的情况）

- 所有父类的成员变量和自己的成员变量都会存放在该对象所对应的存储空间中
- 父类的方法和自己的方法都会缓存在类对象的方法缓存中，类方法是缓存在元类对象中
- 每一个对象内部都有一个isa指针,指向他的类对象,类对象中存放着本对象的如下信息
 - 对象方法列表
 - 成员变量的列表
 - 属性列表
- 每个Objective-C 对象都有相同的结构，如下图所示

Objective-C 对象的结构图 | --- | ISA指针| 根类(NSObject)的实例变量| 倒数第二层父类的实例变量| ... | 父类的实例变量 | 类的实例变量 |

 - 根类对象就是NSObject, 它的super class指针指向nil
 - 类对象既然称为对象，那它也是一个实例。类对象中也有一个isa指针指向它的元类(meta class)，即类对象是元类的实例。元类内部存放的是类方法列表，根元类的isa指针指向自己，superclass指针指向NSObject类



微博@iOS程序猿袁

一个objc对象的isa的指针指向什么？有什么作用？

- 每一个对象内部都有一个isa指针，这个指针是指向它的真实类型
- 根据这个指针就能知道将来调用哪个类的方法

objc中的类方法和实例方法有什么本质区别和联系

- 类方法：
 - 类方法是属于类对象的
 - 类方法只能通过类对象调用
 - 类方法中的self是类对象
 - 类方法可以调用其他的类方法
 - 类方法中不能访问成员变量
 - 类方法中不能直接调用对象方法

- 类方法是存储在元类对象的方法缓存中
 - 实例方法：
 - 实例方法是属于实例对象的
 - 实例方法只能通过实例对象调用
 - 实例方法中的self是实例对象
 - 实例方法中可以访问成员变量
 - 实例方法中直接调用实例方法
 - 实例方法中可以调用类方法(通过类名)
 - 实例方法是存放在类对象的方法缓存中
-

是否可以把比较耗时的操作放在NSNotificationCenter中

- 首先必须明确通知在哪个线程中发出，那么处理接受到通知的方法也在这个线程中调用
- 如果在异步线程发的通知，那么可以执行比较耗时的操作；
- 如果在主线程发的通知，那么就不可以执行比较耗时的操作

NSNotificationCenter原理

- 消息通知的原理简单来说就是有一个通知中心，它有一些属性，来存放那些注册了的观察者的信息，然后当通过通知中心发送通知的时候会去根据name, object去查找符合条件的观察者，将通知发送给它们。
-

runtime 如何实现weak 属性

- 那么runtime如何实现weak变量的自动置nil？
- runtime对注册的类，会进行布局，会将 weak 对象放入一个 hash 表中。
- 用 weak 指向的对象内存地址作为 key，当此对象的引用计数为0的时候会调用对象的 dealloc 方法，
- 假设 weak 指向的对象内存地址是a，那么就会以a为key，在这个 weak hash表中搜索，找到所有以a为key的 weak 对象，从而设置为 nil。

weak属性需要在dealloc中置nil么

- 在ARC环境无论是强指针还是弱指针都无需在dealloc 设置为nil， ARC 会自动帮我们处理
- 即便是编译器不帮我们做这些， weak也不需要在dealloc中置nil
- 在属性所指的对象遭到摧毁时，属性值也会清空

runtime怎么添加属性、方法等

- ivar表示成员变量
- class_addIvar
- class_addMethod
- class_addProperty
- class_addProtocol
- class_replaceProperty

runtime如何通过selector找到对应的IMP地址？（分别考虑类方法和实例方法）

- 每一个类对象中都有一个对象方法列表（对象方法缓存）
- 类方法列表是存放在类对象中isa指针指向的元类对象中（类方法缓存）
- 方法列表中每个方法结构体中记录着方法的名称、方法实现以及参数类型，其实selector本质就是方法名称，通过这个方法名称就可以在方法列表中找到对应的方法实现。
- 当我们发送一个消息给一个NSObject对象时，这条消息会在对象的类对象方法列表里查找
- 当我们发送一个消息给一个类时，这条消息会在类的Meta Class对象的方法列表里查找

使用runtime Associate方法关联的对象，需要在主对象 dealloc的时候释放么？

- 无论在MRC下还是ARC下均不需要
- 被关联的对象在生命周期内要比对象本身释放的晚很多，它们会在被NSObject -dealloc调用的object_dispose()方法中释放
- 补充：对象的内存销毁时间表，分四个步骤

```

1. 调用 -release : 引用计数变为零
 * 对象正在被销毁，生命周期即将结束。
 * 不能再有新的 __weak 弱引用，否则将指向 nil。
 * 调用 [self dealloc]
2. 父类调用 -dealloc
 * 继承关系中最直接继承的父类再调用 -dealloc
 * 如果是 MRC 代码 则会手动释放实例变量们 (iVars)
 * 继承关系中每一层的父类 都再调用 -dealloc
3. NSObject 调 -dealloc
 * 只做一件事：调用 Objective-C runtime 中的 object_dispose() 方法
4. 调用 object_dispose()
 * 为 C++ 的实例变量们 (iVars) 调用 destructors
 * 为 ARC 状态下的 实例变量们 (iVars) 调用 -release
 * 解除所有使用 runtime Associate方法关联的对象
 * 解除所有 __weak 引用
 * 调用 free()

```

_objc_msgForward函数是做什么的？直接调用它将会发生什么？

- _objc_msgForward是IMP类型，用于消息转发的：当向一个对象发送一条消息，但它并没有实现的时候，_objc_msgForward会尝试做消息转发
- 直接调用_objc_msgForward是非常危险的事，这是把双刃刀，如果用不好会直接导致程序Crash，但是如果用得好，能做很多非常酷的事
- JSPatch就是直接调用_objc_msgForward来实现其核心功能的
- 详细解说参见这里的第一个问题解答

能否向编译后得到的类中增加实例变量？能否向运行时创建的类中添加实例变量？为什么？

- 不能，向编译后得到的类中增加实例变量；

- 能，向运行时创建的类中添加实例变量；
- 分析如下：
 - 因为编译后的类已经注册在runtime中，类结构体中的objc_ivar_list实例变量的链表和instance_size实例变量的内存大小已经确定，同时runtime会调用class_setIvarLayout或class_setWeakIvarLayout来处理strong weak引用，所以不能向存在的类中添加实例变量
 - 运行时创建的类是可以添加实例变量，调用class_addIvar函数，但是得在调用objc_allocateClassPair之后，objc_registerClassPair之前，原因同上。

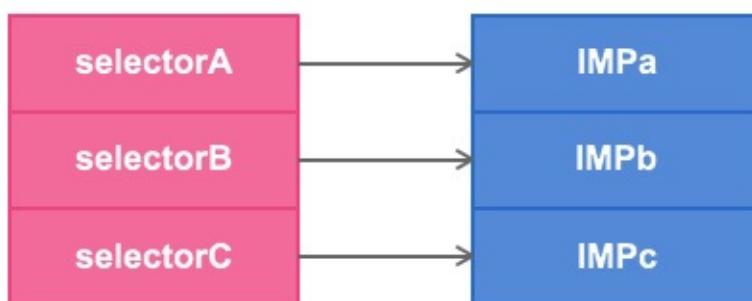
调用方法的过程 (runtime)

- Objective-C是动态语言，每个方法在运行时会被动态转为消息发送，即：objc_msgSend(receiver, selector)，整个过程介绍如下：
 - objc在向一个对象发送消息时，runtime库会根据对象的isa指针找到该对象实际所属的类
 - 然后在该类中的方法列表以及其父类方法列表中寻找方法运行
 - 如果，在最顶层的父类（一般也就NSObject）中依然找不到相应的方法时，程序在运行时会挂掉并抛出异常 unrecognized selector sent to XXX
 - 但是在这之前，objc的运行时会给出三次拯救程序崩溃的机会，这三次拯救程序崩溃的说明见问题《什么时候会报unrecognized selector的异常》中的说明
- 补充说明：Runtime 铸就了Objective-C 是动态语言的特性，使得C语言具备了面向对象的特性，在程序运行期创建，检查，修改类、对象及其对应的方法，这些操作都可以使用runtime中的对应方法实现。

引申点：三次拯救程序崩溃是什么？

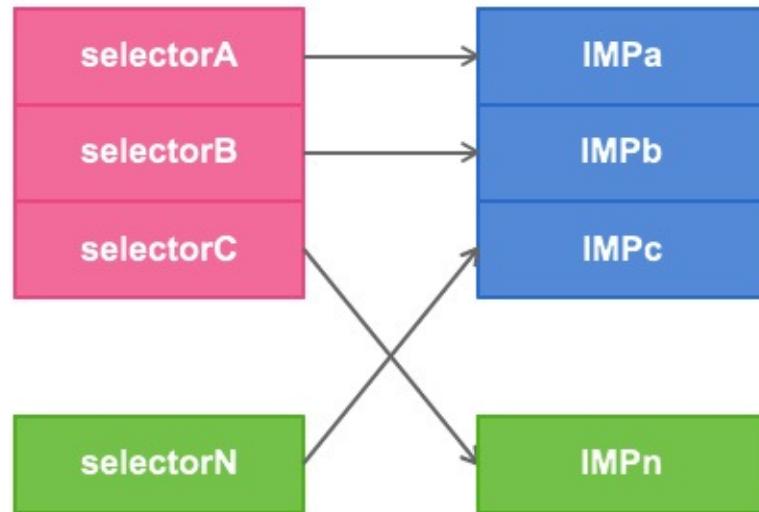
什么是method swizzling (俗称黑魔法)

- 简单说就是进行方法交换
- 在Objective-C中调用一个方法，其实是向一个对象发送消息，查找消息的唯一依据是selector的名字。利用Objective-C的动态特性，可以实现在运行时偷换selector对应的方法实现，达到给方法挂钩的目的
- 每个类都有一个方法列表，存放着方法的名字和方法实现的映射关系，selector的本质其实就是方法名，IMP有点类似函数指针，指向具体的Method实现，通过selector就可以找到对应的IMP



<http://blog.csdn.net/yiayaixuexi>

- 交换方法的几种实现方式
 - 利用method_exchangeImplementations 交换两个方法的实现
 - 利用class_replaceMethod 替换方法的实现
 - 利用method_setImplementation 来直接设置某个方法的IMP



<http://blog.csdn.net/yiyaaixueyi>

什么时候会报unrecognized selector的异常?

- 当调用该对象上某个方法,而该对象上没有实现这个方法的时候, 可以通过“消息转发”进行解决, 如果还是不行就会报unrecognized selector异常
- objc是动态语言, 每个方法在运行时会被动态转为消息发送, 即: `objc_msgSend(receiver, selector)`, 整个过程介绍如下:
 - objc在向一个对象发送消息时, runtime库会根据对象的isa指针找到该对象实际所属的类
 - 然后在该类中的方法列表以及其父类方法列表中寻找方法运行
 - 如果, 在最顶层的父类中依然找不到相应的方法时, 程序在运行时会挂掉并抛出异常`unrecognized selector sent to XXX`。但是在这之前, objc的运行时会给出三次拯救程序崩溃的机会
- 三次拯救程序崩溃的机会
 - Method resolution
 - objc运行时会调用`+resolveInstanceMethod:`或者`+resolveClassMethod:`, 让你有机会提供一个函数实现。
 - 如果你添加了函数并返回YES, 那运行时系统就会重新启动一次消息发送的过程
 - 如果`resolve`方法返回NO, 运行时就会移到下一步, 消息转发
 - Fast forwarding
 - 如果目标对象实现了`-forwardingTargetForSelector:`, Runtime这时就会调用这个方法, 给你把这个消息转发给其他对象的机会
 - 只要这个方法返回的不是nil和self, 整个消息发送的过程就会被重启, 当然发送的对象会变成你返回的那个对象。
 - 否则, 就会继续Normal Forwarding。
 - 这里叫Fast, 只是为了区别下一步的转发机制。因为这一步不会创建任何新的对象, 但Normal Forwarding转发会创建一个NSInvocation对象, 相对Normal Forwarding转发更快点, 所以这里叫Fast Forwarding
 - Normal forwarding
 - 这一步是Runtime最后一次给你挽救的机会。
 - 首先它会发送`-methodSignatureForSelector:`消息获得函数的参数和返回值类型。

- 如果-methodSignatureForSelector:返回nil, Runtime则会发出-doesNotRecognizeSelector:消息, 程序这时也就挂掉了。
- 如果返回了一个函数签名, Runtime就会创建一个NSInvocation对象并发送-forwardInvocation:消息给目标对象

有没有用过运行时, 用它都能做什么?

- 交换方法, 创建类, 给新创建的类增加方法, 改变isa指针
 - 交换方式: 一般写在类的+(void)load方法里面

```
Method originalM = class_getInstanceMethod([self class], @selector(originMethod));
Method exchangeM = class_getInstanceMethod([self class], @selector(exChangeMethod));
method_exchangeImplementations(originalM, exchangeM);
```

- 创建类:

```
/// 创建类
- (void)creatClassMethod {

    Class Person = objc_allocateClassPair([NSObject class], "Person", 0);
    //添加属性
    objc_property_attribute_t type = { "T", "@\"NSString\"" };
    objc_property_attribute_t ownership = { "C", "" }; // C = copy
    objc_property_attribute_t backingivar = { "V", "_privateName" };
    objc_property_attribute_t attrs[] = { type, ownership, backingivar };
    class_addProperty(Person, "name", attrs, 3);
    //添加方法
    class_addMethod(Person, @selector(name), (IMP)nameGetter, "@@:");
    class_addMethod(Person, @selector(setName:), (IMP)nameSetter, "v@:@");
    //注册该类
   objc_registerClassPair(Person);

    //获取实例
    id instance = [[Person alloc] init];
    NSLog(@"%@", instance);
    [instance setName:@"hxn"];
    NSLog(@"%@", [instance name]);
}

//get方法
NSString *nameGetter(id self, SEL _cmd) {
    Ivar ivar = class_getInstanceVariable([self class], "_privateName");
    return object_getIvar(self, ivar);
}

//set方法
void nameSetter(id self, SEL _cmd, NSString *newName) {
    Ivar ivar = class_getInstanceVariable([self class], "_privateName");
    id oldName = object_getIvar(self, ivar);
    if (oldName != newName) object_setIvar(self, ivar, [newName copy]);
}
```

- 添加方法

```
/**参数一、类名参数
二、SEL 添加的方法名字参数
三、IMP指针 (IMP就是Implementation的缩写, 它是指向一个方法实现的指针, 每一个方法都有一个对应的IMP)
参数四、其中types参数为"i:@@", 按顺序分别表示: 具体类型可参照[官方文档](https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Articles/ocrtTypeEncodings.html)i 返回值类型int, 若是v则表示void@
参数id(self): SEL(_cmd)@ id(str)
V@:表示返回值是void 带有SEL参数 (An object (whether statically typed or typed id))
*/
```

```
class_addMethod(Person, @selector(addMethodForMyClass:), (IMP)addMethodForMyClass, "V@:");
```

- 添加实例变量

```
/**参数一、类名参数
二、属性名称参数
三、开辟字节长度参数
四、对其方式参数
五、参数类型 “@” 官方解释 An object (whether statically typed or typed id) (对象 静态类型或者id类型) 具体类型可参照
[官方文档](https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Articles/ocrtTypeEncodings.html) return: BOOL 是否添加成功
*/
BOOL isSuccess = class_addIvar(Person, "name", sizeof(NSString *), 0, "@");
isSuccess?NSLog(@"添加变量成功"):NSLog(@"添加变量失败");
```

什么是 Runloop?

- 从字面上讲就是运行循环。
- 它内部就是do-while循环，在这个循环内部不断地处理各种任务。
- 一个线程对应一个RunLoop，主线程的RunLoop默认已经启动，子线程的RunLoop得手动启动（调用run方法）
- RunLoop只能选择一个Mode启动，如果当前Mode中没有任何Source(Sources0、Sources1)、Timer，那么就直接退出RunLoop
- 基本的作用就是保持程序的持续运行，处理app中的各种事件。通过runloop，有事运行，没事就休息，可以节省cpu资源，提高程序性能。

- Runloop对象

- iOS中有2套API来访问和使用RunLoop
- Foundation: NSRunLoop
- Core Foundation: CFRunLoopRef
- NSRunLoop和CFRunLoopRef都代表着RunLoop对象
- NSRunLoop是基于CFRunLoopRef的一层OC包装，所以要了解RunLoop内部结构，需要多研究CFRunLoopRef层面的API。

autorelease 对象在什么情况下会被释放？

- 分两种情况：手动干预释放和系统自动释放
- 手动干预释放就是指定autoreleasepool,当前作用域大括号结束就立即释放
- 系统自动去释放:不手动指定autoreleasepool,autorelease对象会在当前的 runloop 迭代结束时释放
- kCFRunLoopEntry(1):第一次进入会自动创建一个autorelease
- kCFRunLoopBeforeWaiting(32):进入休眠状态前会自动销毁一个autorelease,然后重新创建一个新的autorelease
- kCFRunLoopExit(128):退出runloop时会自动销毁最后一个创建的autorelease

苹果是如何实现autoreleasepool的？

- autoreleasepool以一个队列数组的形式实现,主要通过下列三个函数完成.
 - objc_autoreleasepoolPush
 - objc_autoreleasepoolPop

- objc_aurorelease
- 看函数名就可以知道，对autorelease分别执行push，和pop操作。销毁对象时执行release操作

runloop和线程有什么关系？

- 每条线程都有唯一的一个RunLoop对象与之对应的
- 主线程的RunLoop是自动创建并启动
- 子线程的RunLoop需要手动创建
- 子线程的RunLoop创建步骤如下：

为什么 UIScrollView 的滚动会导致 NSTimer 失效？

- NSTimer对象是在 NSDefaultRunLoopMode下面调用消息的，但是当我们滑动scrollview的时候，NSDefaultRunLoopMode模式就自动切换到UITrackingRunLoopMode模式下面，却不可以继续响应NSTimer发送的消息。所以如果想在滑动scrollview的情况下还调用NSTimer的消息，我们可以把NSRunLoop的模式更改为NSRunLoopCommonModes

NSTimer 准吗？怎么写一个准的timer

- 不准
- 原因：定时器被添加在主线程中，由于定时器在一个RunLoop中被检测一次，所以如果在这一次的RunLoop中做了耗时的操作，当前RunLoop持续的时间超过了定时器的间隔时间，那么下一次定时就被延后了。
- 利用 CADisplayLink
 - CADisplayLink是一个频率能达到屏幕刷新率的定时器类。iPhone屏幕刷新频率为60帧/秒，也就是说最小间隔可以达到1/60s。
- GCD定时器
 - 我们知道，RunLoop是dispatch_source_t实现的timer，所以理论上来说，GCD定时器的精度比NSTimer只高不低。

```

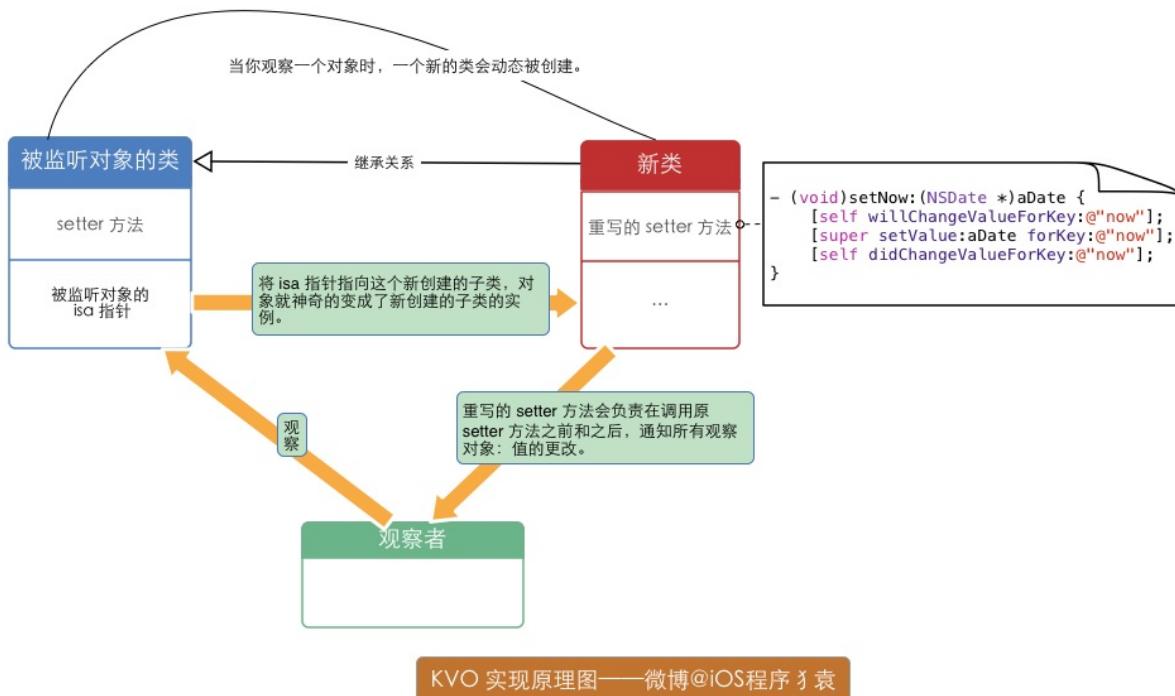
NSTimeInterval interval = 1.0;
_timer = dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0, dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0));
dispatch_source_set_timer(_timer, dispatch_walltime(NULL, 0), interval * NSEC_PER_SEC, 0);
dispatch_source_set_event_handler(_timer, ^{
    NSLog(@"GCD timer test");
});
dispatch_resume(_timer);

```

KVO内部实现原理

- KVO是基于runtime机制实现的
- 当某个类的属性对象第一次被观察时，系统就会在运行期动态地创建该类的一个派生类，在这个派生类中重写基类中任何被观察属性的setter方法。派生类在被重写的setter方法内实现真正的通知机制
- 如果原类为Person，那么生成的派生类名为NSKVONotifying_Person

- 每个类对象中都有一个isa指针指向当前类，当一个类对象的第一次被观察，那么系统会偷偷将isa指针指向动态生成的派生类，从而在给被监控属性赋值时执行的是派生类的setter方法
- 键值观察通知依赖于NSObject 的两个方法: willChangeValueForKey: 和didChangeValueForKey:；在一个被观察属性发生改变之前，willChangeValueForKey: 一定会被调用，这就会记录旧的值。而当改变发生后，didChangeValueForKey: 会被调用，继而observeValueForKey:ofObject:change:context: 也会被调用。
- 补充：KVO的这套实现机制中苹果还偷偷重写了class方法，让我们误认为还是使用的当前类，从而达到隐藏生成的派生类



如何手动触发一个value的KVO

自动触发的场景：在注册KVO之前设置一个初始值，注册之后，设置一个不一样的值，就可以触发了。想知道如何手动触发，必须知道自动触发KVO 的原理，见上面的描述 手动触发演示

```
@property (nonatomic, strong) NSDate *now;

- (void)viewDidLoad
{
    [super viewDidLoad];

    // “手动触发self.now的KVO”，必写。
    [self willChangeValueForKey:@"now"];

    // “手动触发self.now的KVO”，必写。
    [self didChangeValueForKey:@"now"];
}
```

若一个类有实例变量 NSString *_foo，调用 setValue:forKey: 时，是以 foo 还是 _foo 作为 key？

- 都可以

KVC的keyPath中的集合运算符如何使用？

- 必须用在集合对象上或普通对象的集合属性上
- 简单集合运算符有@avg, @count, @max, @min, @sum
- 格式@{@sum.age} 或{@"集合属性.@max.age"}? ? ?

KVC和KVO的keyPath一定是属性么？

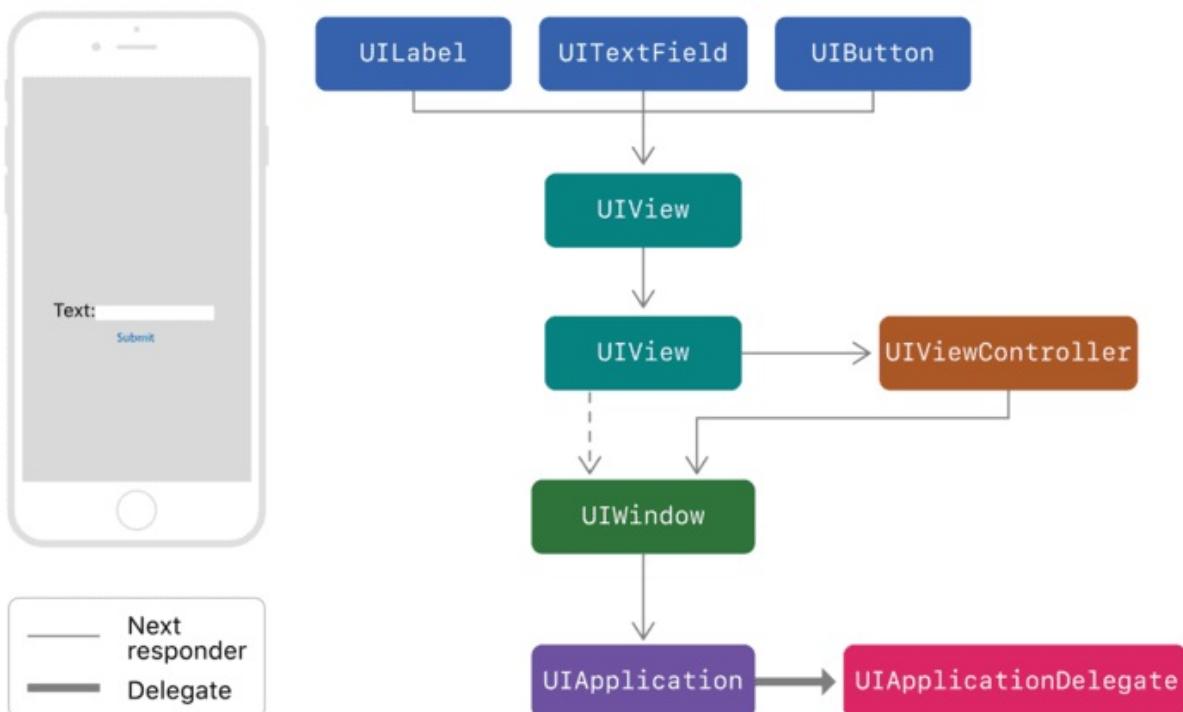
- 可以是成员变量

=====

为什么更新UI要在主线程？

- 1. UIKit不是线程安全的，假如在两个线程中设置了同一张背景图片，很有可能就会由于背景图片被释放两次，使得程序崩溃。或者某一个线程中遍历找寻某个subView，然而在另一个线程中删除了该subView，那么就会造成错误。
- 2. 子线程中对所有其他ui更新都要等到该子线程生命周期结束才进行，而对响应用户点击的Button的UI更新则是及时的。

iOS事件响应链



- 1、响应者链通常是由视图（`UIView`）构成的；
- 2、一个视图的下一个响应者是它视图控制器（`UIViewController`）（如果有的话），然后再转给它的父视图（`Super View`）；
- 3、视图控制器（如果有的话）的下一个响应者为其管理的视图的父视图；
- 4、单例的窗口（`UIWindow`）的内容视图将指向窗口本身作为它的下一个响应者 需要指出的是，Cocoa Touch应用不像Cocoa应用，它只有一个`UIWindow`对象，因此整个响应者链要简单一点；
- 5、单例的应用（`UIApplication`）是一个响应者链的终点，也可能到app delegate如果对象是`UIResponder`的实例并且不是响应链的一部分。它的下一个响应者指向nil，以结束整个循环。

UITableView的优化方法

缓存高度，异步绘制，减少层级，hide，避免离屏渲染

iOS面试- 网络

=====

说出几个http错误码

- 200 正常；请求已完成。
- 301 永久重定向，表示请求的资源已经永久的搬到了其他位置
- HTTP 400 - 请求无效
- 401 表示发送的请求需要有HTTP认证信息或者是认证失败了
- 403 表示对请求资源的访问被服务器拒绝了
- HTTP 404 - 无法找到文件
- HTTP 414 - 请求 - URI 太长
- Error 501 - 未实现
- HTTP 500 - 内部服务器错误
- 503 表示服务器超负载或正停机维护，无法处理请求

HTTP协议中POST方法和GET方法有那些区别？

1. GET提交，请求的数据会附在URL之后，会在地址栏中显示出来。

POST提交：把提交的数据放置在是HTTP包的包体中。

2. GET:特定浏览器和服务器对URL长度有限制。

POST:由于不是通过URL传值，理论上数据不受限。但实际各个WEB服务器会规定对post提交数据大小进行限制，Apache、IIS6都有各自的配置。

- 3.安全性：POST的安全性要比GET的安全性高。

比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存，(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击

https原理：证书传递、验证和数据加密、解密过程解析

- HTTPS其实是有两部分组成：HTTP + SSL / TLS，也就是在HTTP上又加了一层处理加密信息的模块。服务端和客户端的信息传输都会通过TLS进行加密，所以传输的数据都是加密后的数据
 - 1. 浏览器将自己支持的一套加密规则发送给网站。
 - 2. 网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。
 - 3. 浏览器获得网站证书之后浏览器要做以下工作：
 - a) 验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。
 - b) 如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。
 - c) 使用约定好的HASH算法计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。
 - 4. 网站接收浏览器发来的数据之后要做以下的操作：
 - a) 使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览

- 器发来的一致。
 - b) 使用密码加密一段握手消息，发送给浏览器。
 - 5. 浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。
- 这里浏览器与网站互相发送加密的握手消息并验证，目的是为了保证双方都获得了一致的密码，并且可以正常的加密解密数据，为后续真正数据的传输做一次测试。另外，HTTPS一般使用的加密与HASH算法如下：

* 非对称加密算法：RSA, DSA/DSS
 * 对称加密算法：AES, RC4, 3DES
 * HASH算法：MD5, SHA1, SHA256

说出几个加密方式

- DES加密算法
- AES加密算法
- RSA加密算法
- Base64加密算法
- MD5加密算法
- SHA1加密算法
- DES加密算法
 - DES加密算法是一种分组密码，以64位为分组对数据加密，它的密钥长度是56位，加密解密用同一算法。DES加密算法是对密钥进行保密，而公开算法，包括加密和解密算法。这样，只有掌握了和发送方相同密钥的人才能解读由DES加密算法加密的密文数据。因此，破译DES加密算法实际上就是搜索密钥的编码。对于56位长度的密钥来说，如果用穷举法来进行搜索的话，其运算次数为256。随着计算机系统能力的不断发展，DES的安全性比它刚出现时会弱得多，然而从非关键性质的实际出发，仍可以认为它是足够的。不过，DES现在仅用于旧系统的鉴定，而更多地选择新的加密标准。
- AES加密算法
 - AES加密算法是密码学中的高级加密标准，该加密算法采用对称分组密码体制，密钥长度的最少支持为128、192、256，分组长度128位，算法应易于各种硬件和软件实现。这种加密算法是美国联邦政府采用的区块加密标准，这个标准用来替代原先的DES，已经被多方分析且广为全世界所使用。AES加密算法被设计为支持128 / 192 / 256位 (/32=nb)数据块大小（即分组长度）；支持128 / 192 / 256位 (/32=nk)密码长度，，在10进制里，对应 34×1038 、 62×1057 、 1.1×1077 个密钥。
- RSA加密算法
 - RSA加密算法是目前最有影响力的公钥加密算法，并且被普遍认为是目前最优秀的公钥方案之一。RSA是第一个能同时用于加密和数字签名的算法，它能够抵抗到目前为止已知的所有密码攻击，已被ISO推荐为公钥数据加密标准。RSA加密算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要，但那时想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

加密流程

...

- 随意选择两个大的质数p和q，p不等于q，计算 $N=pq$ 。
- 根据欧拉函数，求得 $r = (p-1)(q-1)$
- 选择一个小于r的整数e，求得e关于模r的模反元素，命名为d。（模反元素存在，当且仅当e与r互质）
- 将p和q的记录销毁。 (N,e) 是公钥， (N,d) 是私钥。Alice将她的公钥 (N,e) 传给Bob，而将她的私钥 (N,d) 藏起来。
- ...
- Base64加密算法
 - Base64加密算法是网络上最常见的用于传输8bit字节代码的编码方式之一，Base64编码可用于在HTTP环境下

传递较长的标识信息。例如，在JAVAPERSISTENCE系统HIBEMATE中，采用了Base64来将一个较长的唯一标识符编码为一个字符串，用作HTTP表单和HTTPGETURL中的参数。在其他应用程序中，也常常需要把二进制数据编码为适合放在URL（包括隐藏表单域）中的形式。此时，采用Base64编码不仅比较简短，同时也具有不可读性，即所编码的数据不会被人用肉眼所直接看到。

- MD5加密算法

- MD5为计算机安全领域广泛使用的一种散列函数，用以提供消息的完整性保护。对MD5加密算法简要的叙述可以认为：MD5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。MD5被广泛用于各种软件的密码认证和钥匙识别上。MD5用的是哈希函数，它的典型应用是对一段信息产生信息摘要，以防止被篡改。MD5的典型应用是对一段Message产生fingerprint指纹，以防止被“篡改”。如果再有一个第三方的认证机构，用MD5还可以防止文件作者的“抵赖”，这就是所谓的数字签名应用。MD5还广泛用于操作系统的登陆认证上，如UNIX、各类BSD系统登录密码、数字签名等诸多方。

- SHA1加密算法

- SHA1是和MD5一样流行的消息摘要算法。SHA加密算法模仿MD4加密算法。SHA1设计为和数字签名算法（DSA）一起使用。SHA1主要适用于数字签名标准里面定义的数字签名算法。对于长度小于 2^{64} 位的消息，SHA1会产生一个160位的消息摘要。当接收到消息的时候，这个消息摘要可以用来验证数据的完整性。在传输的过程中，数据很可能发生变化，那么这时候就会产生不同的消息摘要。SHA1不可以从消息摘要中复原信息，而两个不同的消息不会产生同样的消息摘要。这样，SHA1就可以验证数据的完整性，所以说SHA1是为了保证文件完整性的技术。SHA1加密算法可以采用不超过264位的数据输入，并产生一个160位的摘要。输入被划分为512位的块，并单独处理。160位缓冲器用来保存散列函数的中间和最后结果。缓冲器可以由5个32位寄存器（A、B、C、D和E）来表示。SHA1是一种比MD5的安全性强的算法，理论上，凡是采取“消息摘要”方式的数字验证算法都是有“碰撞”的——也就是两个不同的东西算出的消息摘要相同，互通作弊图就是如此。但是安全性高的算法要找到指定数据的“碰撞”很困难，而利用公式来计算“碰撞”就更困难——目前为止通用安全算法中仅有MD5被破解。加密算法是密码技术的核心，以上这些加密算法是常用的加密算法，而这些算法有些已经遭到破译，有些安全度不高，有些强度不明，有些待进一步分析，有些需要深入研究，而神秘的加密算法世界，又会有新的成员加入，期待更安全的算法诞生。

引申点：对称加密，非对称加密等原理

2018 不同级别的iOS开发工程师的就业形势

- 3年以内软件开发经验、1年左右iOS平台开发经验的工程师一般位于菜鸟阶段，年薪在20w左右。一般职位要求精通Objective-C编程语言，良好的C、C++功底；精通常用数据结构与算法。在此基础上有很大的涨价空间，一般薪资涨幅达30%、甚至double。
- 拥有3年左右iOS平台开发经验的是高级iOS开发工程师，如在BAT，一般是阿里的P6，百度T5，腾讯T3.1，年薪在30w左右。不仅要求扎实的技术能力，一般还要求对终端产品的UI/UE有独到的见解与认识，追求良好的用户体验；对软件产品有强烈的责任心，具备良好的沟通能力和优秀的团队协作能力。
- 拥有4年以上iOS平台开发经验的工程师比较少。在BAT，一般是阿里的P7，百度T6，腾讯T3.2，年薪在50w左右；在创业团队，一般是iOS Tech Leader 研发主管，年薪根据所在公司的规模差别较大，一般在40w左右。不仅要求具备解决技术难题，带领技术团队的技术能力，还要求掌握前沿方向，把控业务的能力。
- 根据公司的地区规模、行业、业务、时间节点等方面原因，薪资会也很有所浮动，大概在20%~30%。

如何成为更高级别的iOS开发工程师？

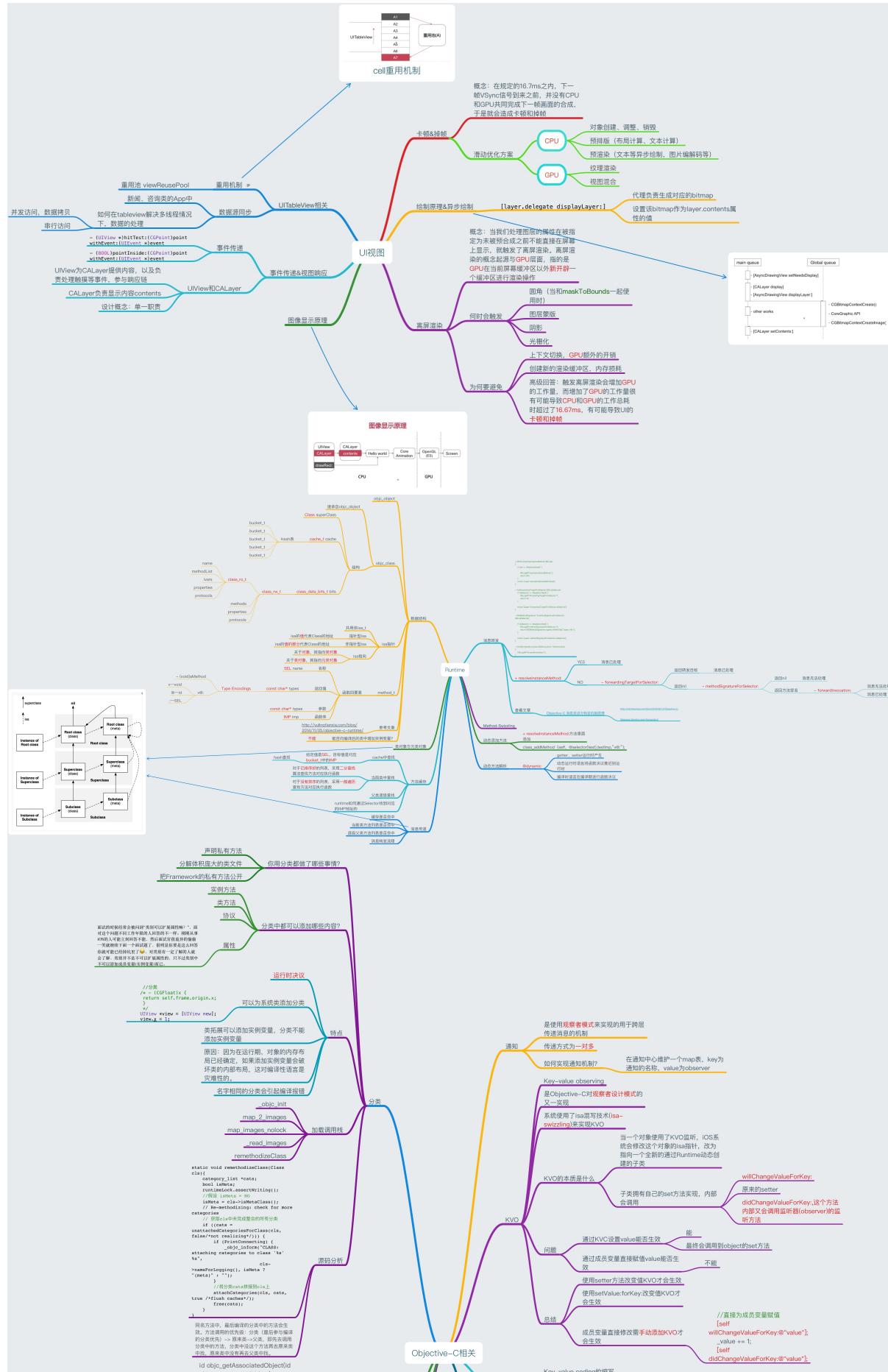
- 熟练掌握C/C++/Objective-C/Swift语言；
- 熟悉Cocoa Touch(Foundation, UIKit)、Objective-C中block, gcd, NSOperation等；
- 熟悉Object消息传递等机制，Objective-C Runtime，阅读源码；
- 熟练使用大部分iOS平台常用库，开源库（AFNetworking, SDWebImage, fmdb），开源控件（EGOTableViewPullRefresh, MRProgress）；
- 关注Github上iOS平台上开源项目最近趋势，尝试fork一些著名开源库；
- iOS App UI develop，熟练使用Interface Builder，理解ReactiveCocoa框架理念，阅读源码；
- 理解Restful Api概念，会使用Restkit，进行网络资源传输；
- 理解Beeframework类hybird框架结构原理，掌握HTML5, CSS, JavaScript等前端知识，掌握jQuery等常用库；
- 熟练使用各种工具debug，调试应用性能；
- 使用Git进行版本控制管理；
- 研究每年WWDC上推荐的最近方法技术，对代码进行重构升级；
- 阅读iOS开发书籍，开发者博客；
- 计算机基础知识扎实（计算机结构，数据结构，算法）。

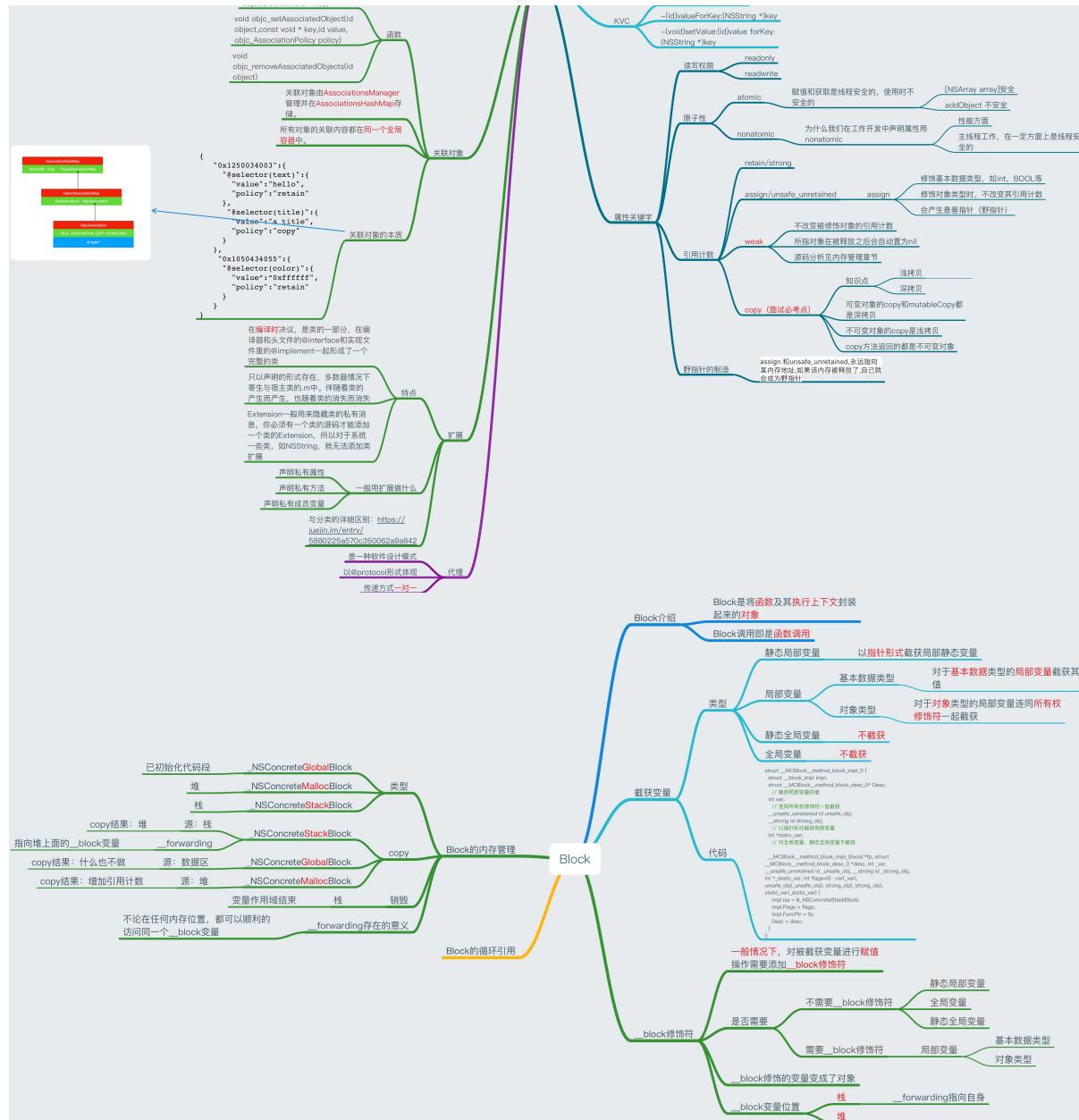
以上列举的这些你都能做到，必将走向大牛之路。

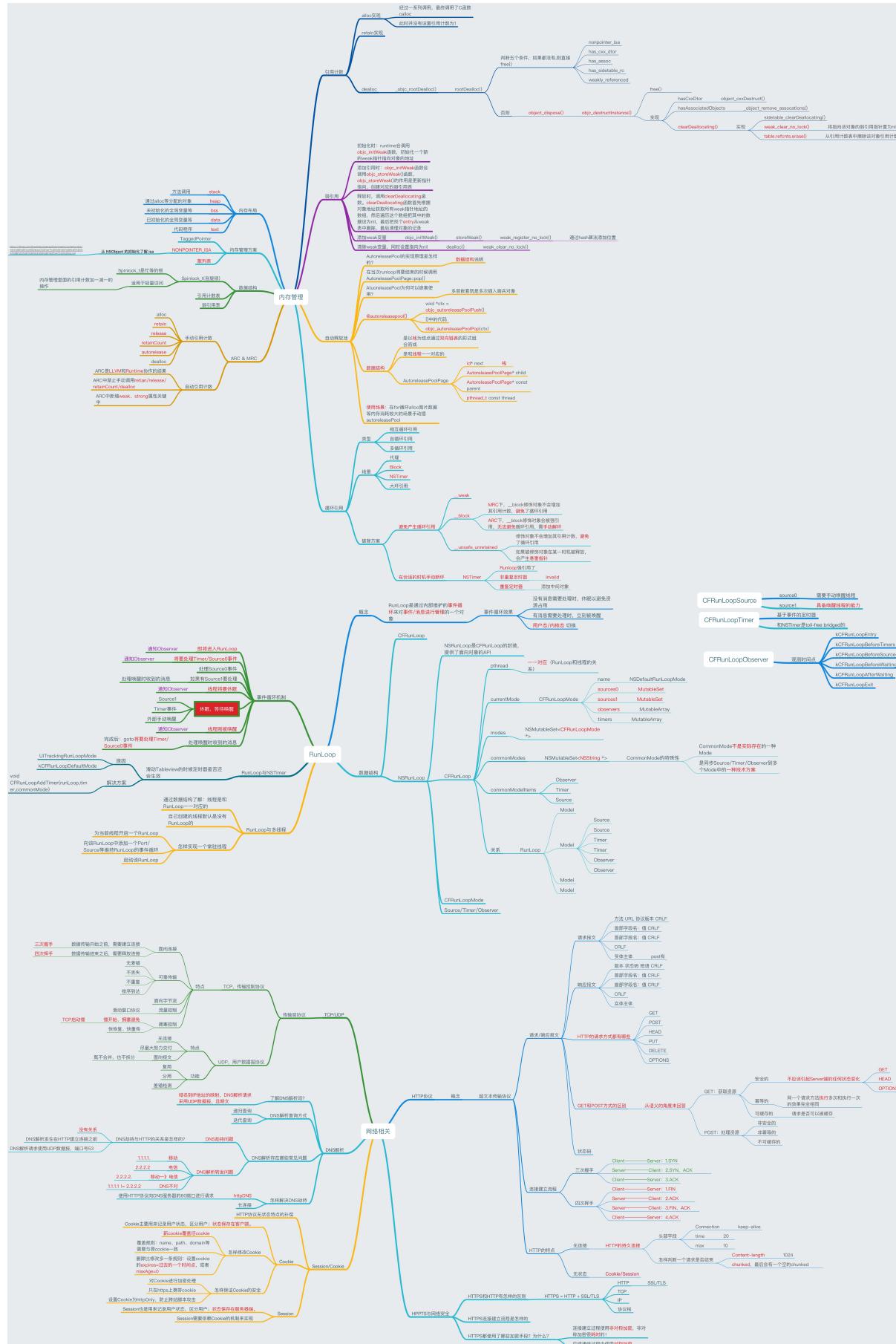
1.iOS Object-C 知识结构图

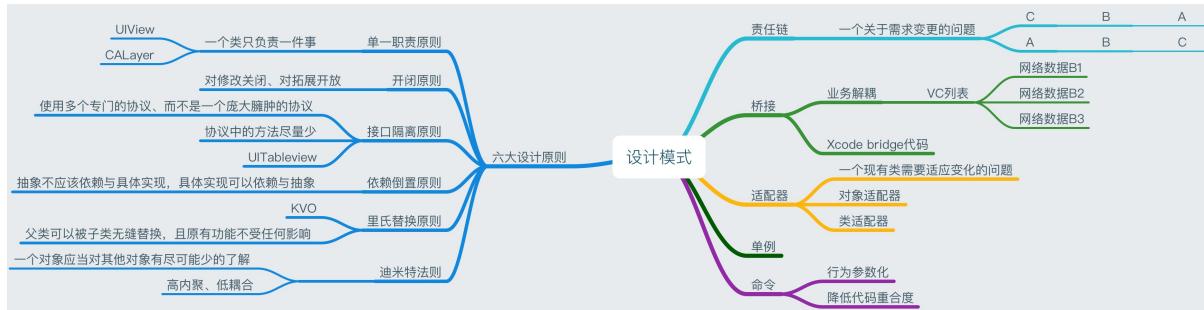


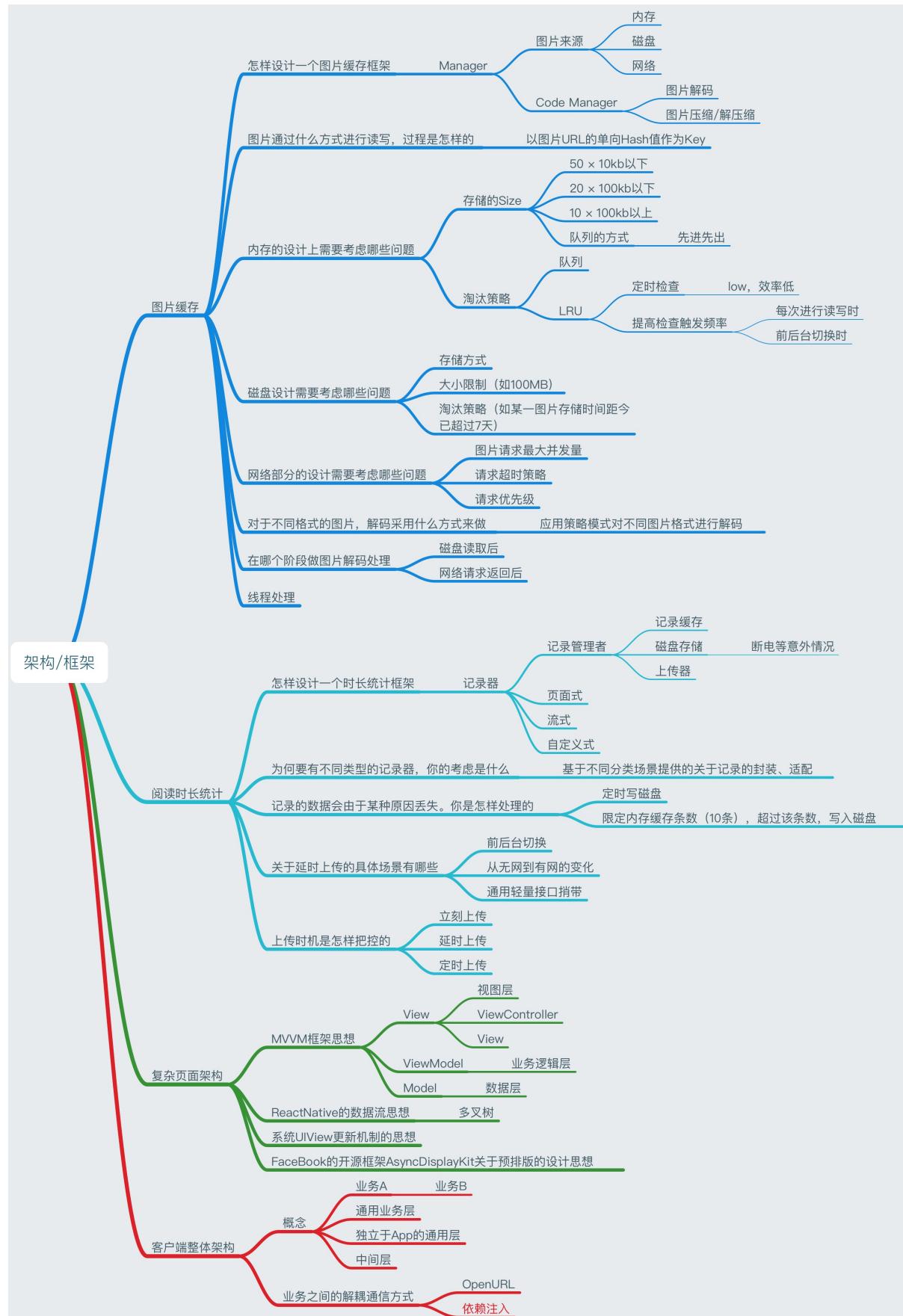
2.iOS Object-C 知识结构图2

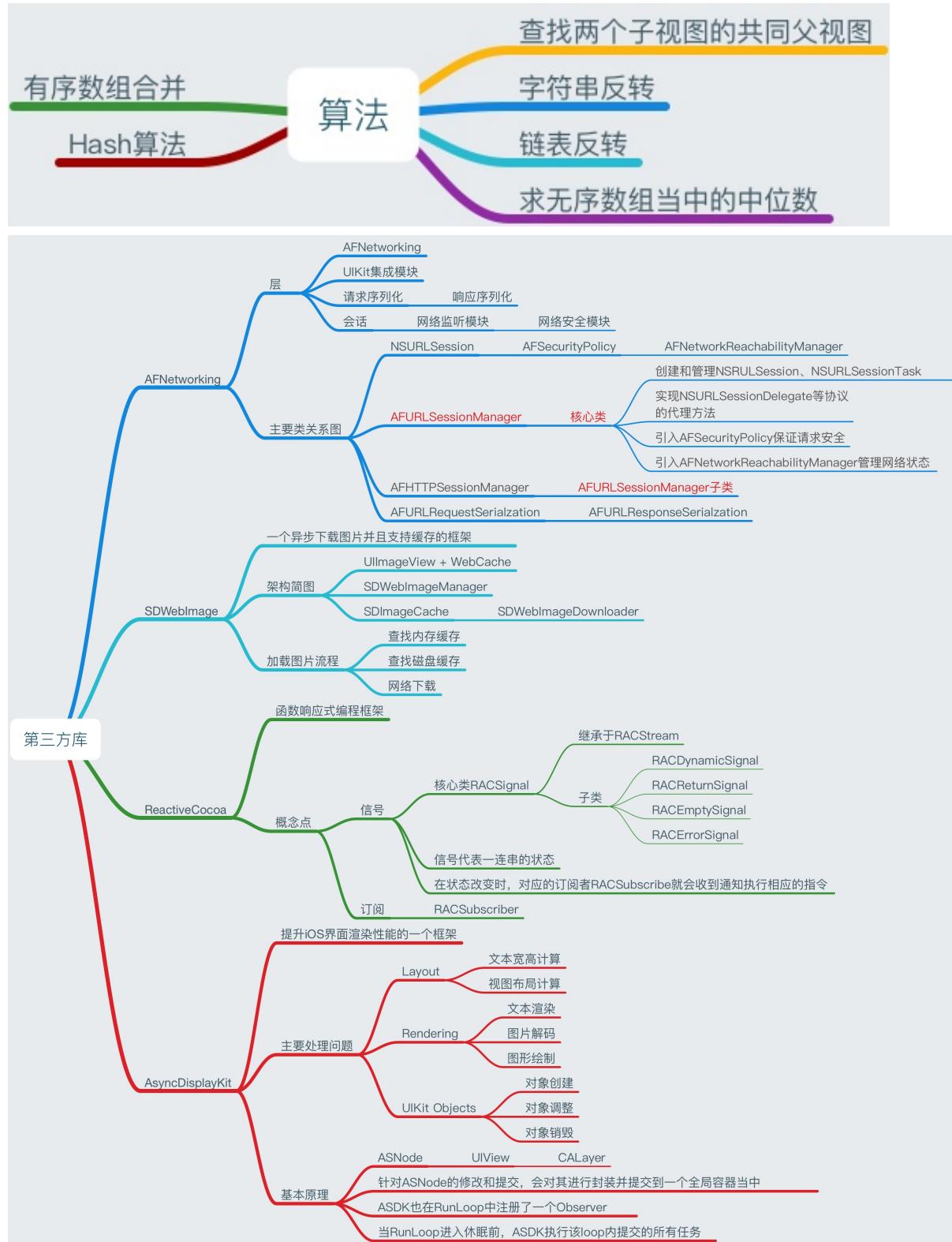




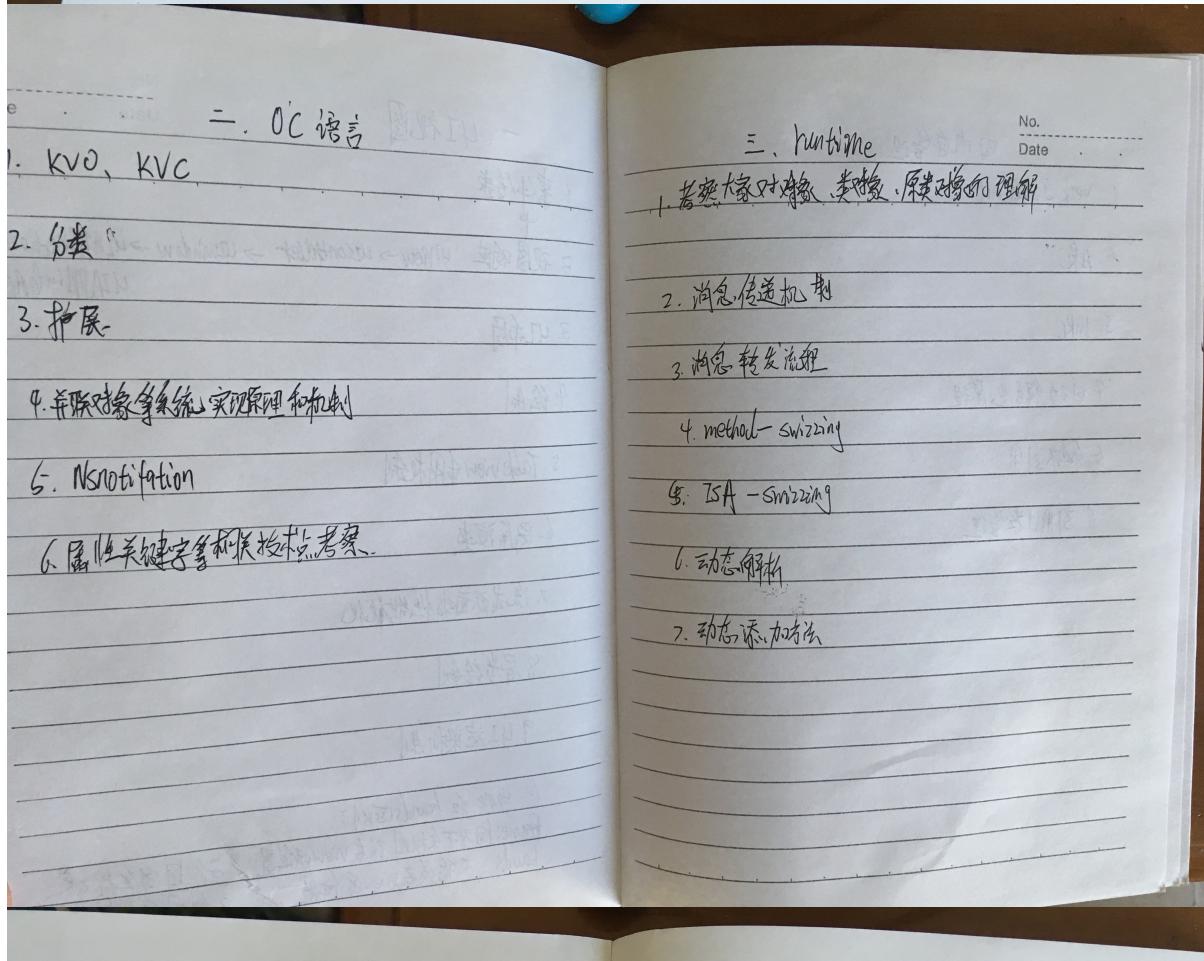
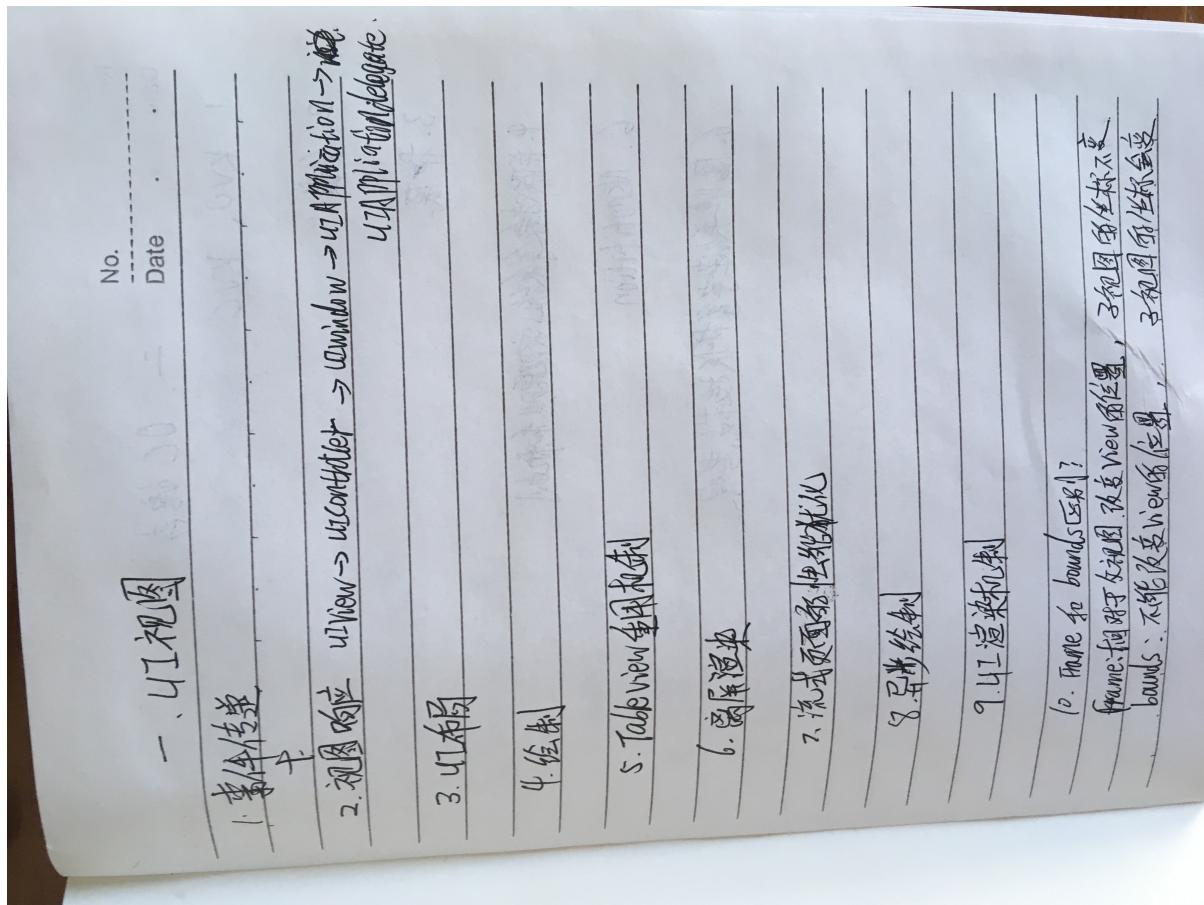




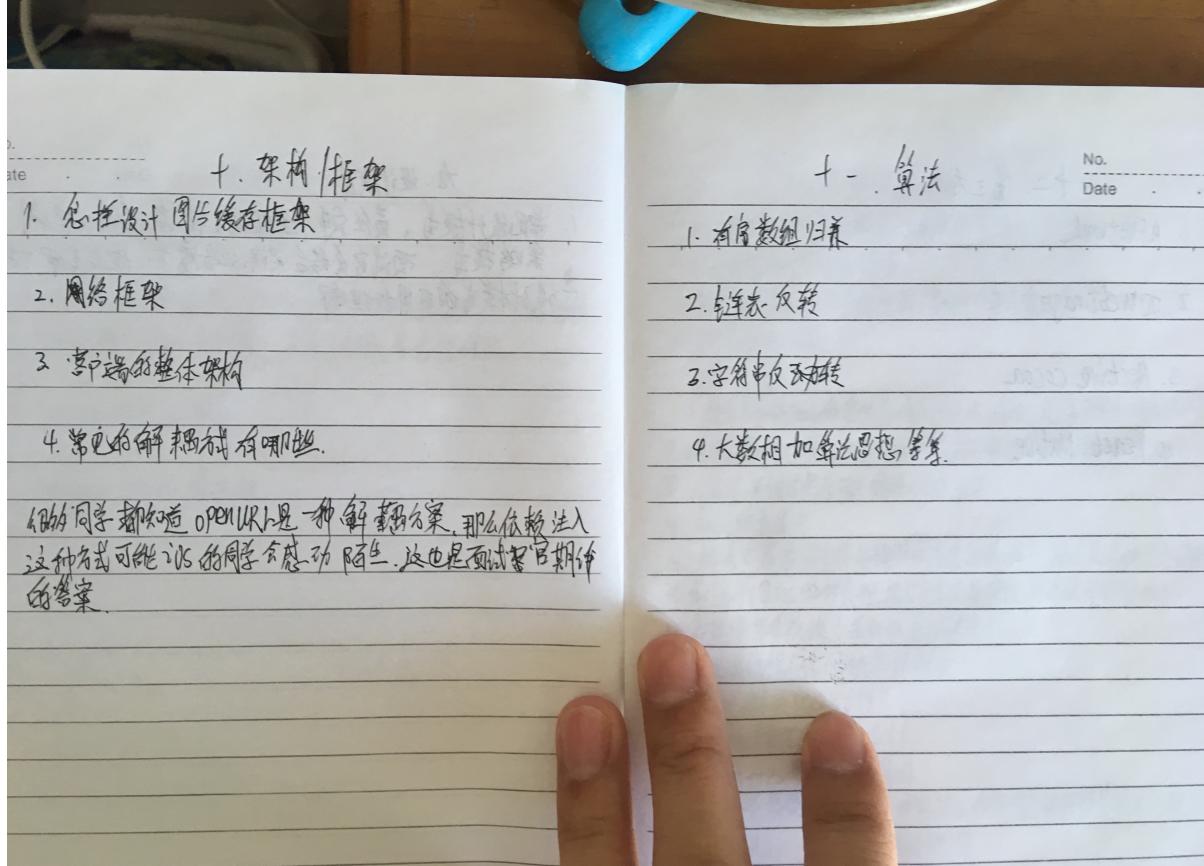
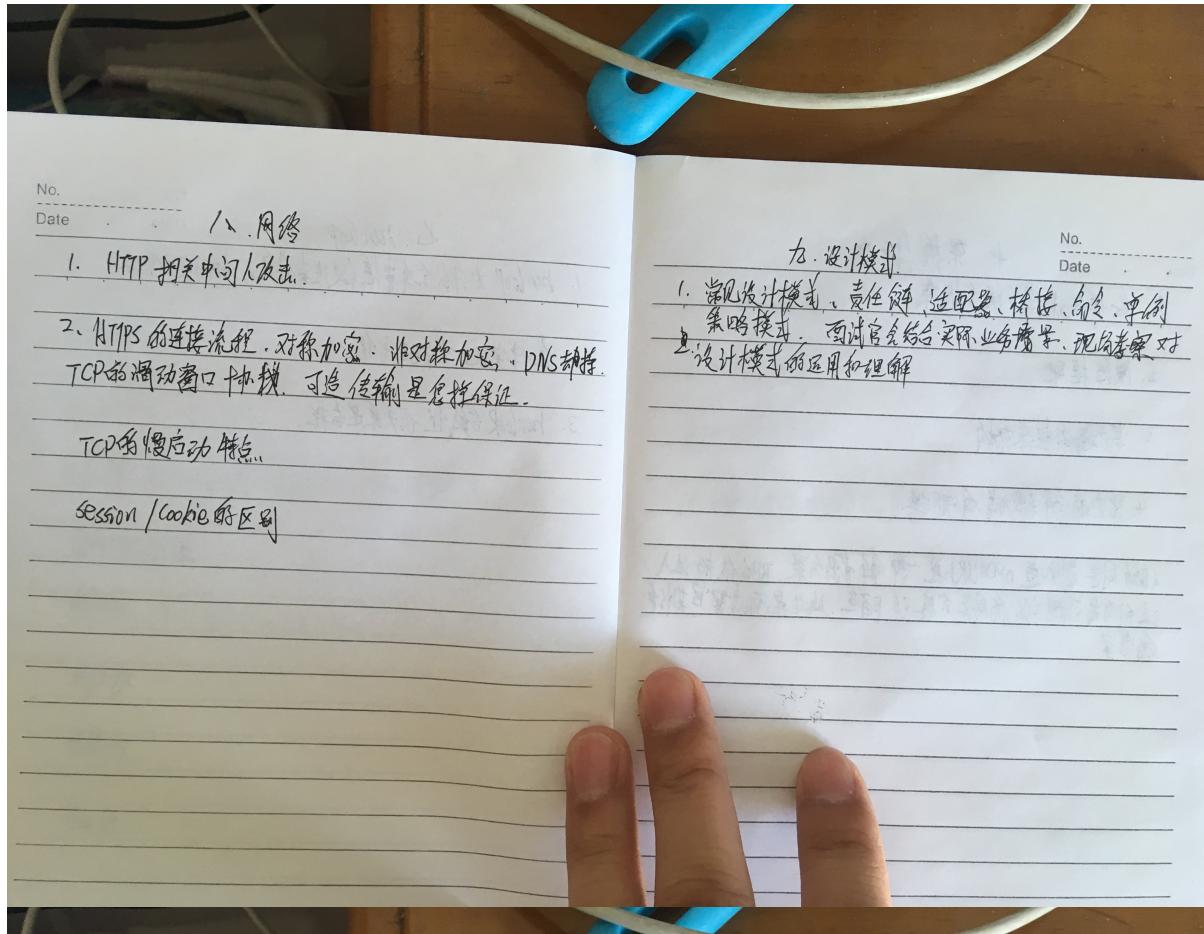


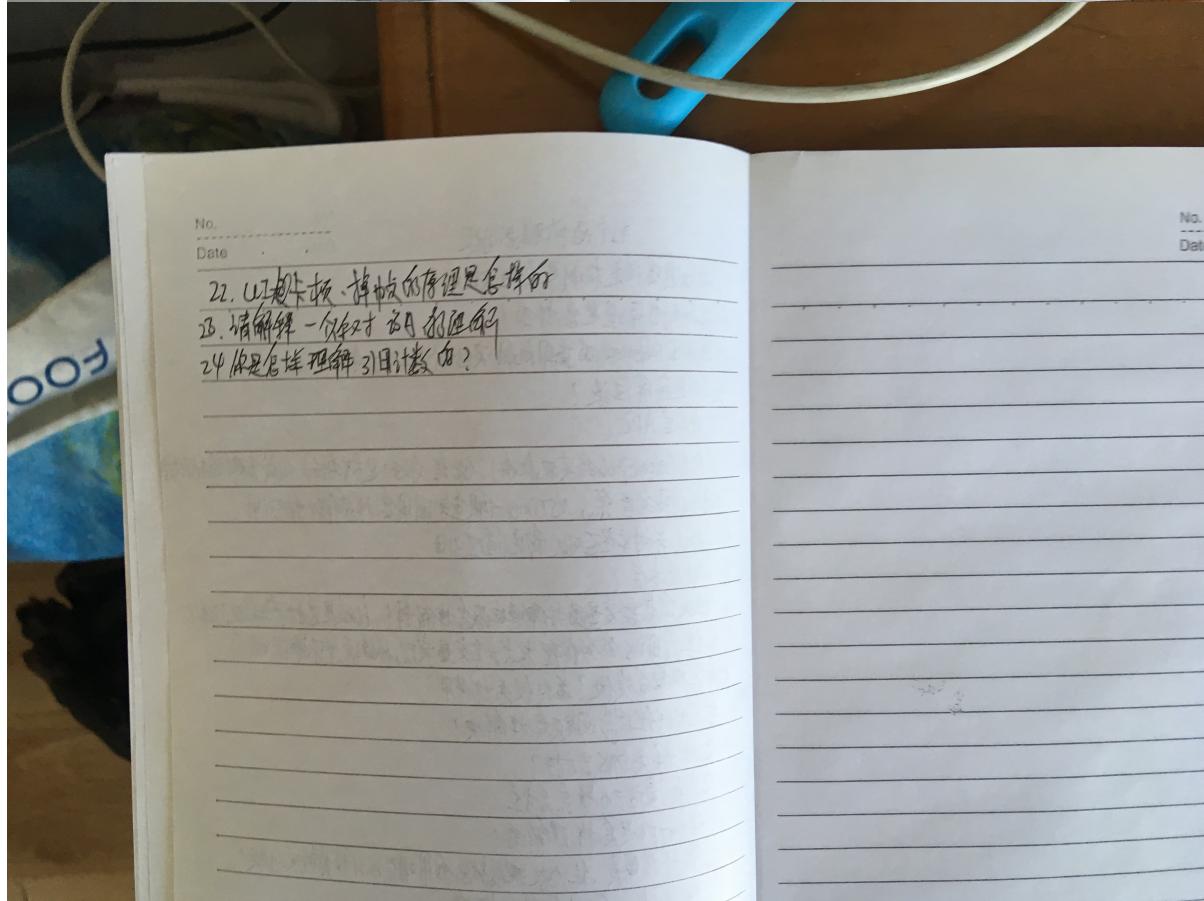
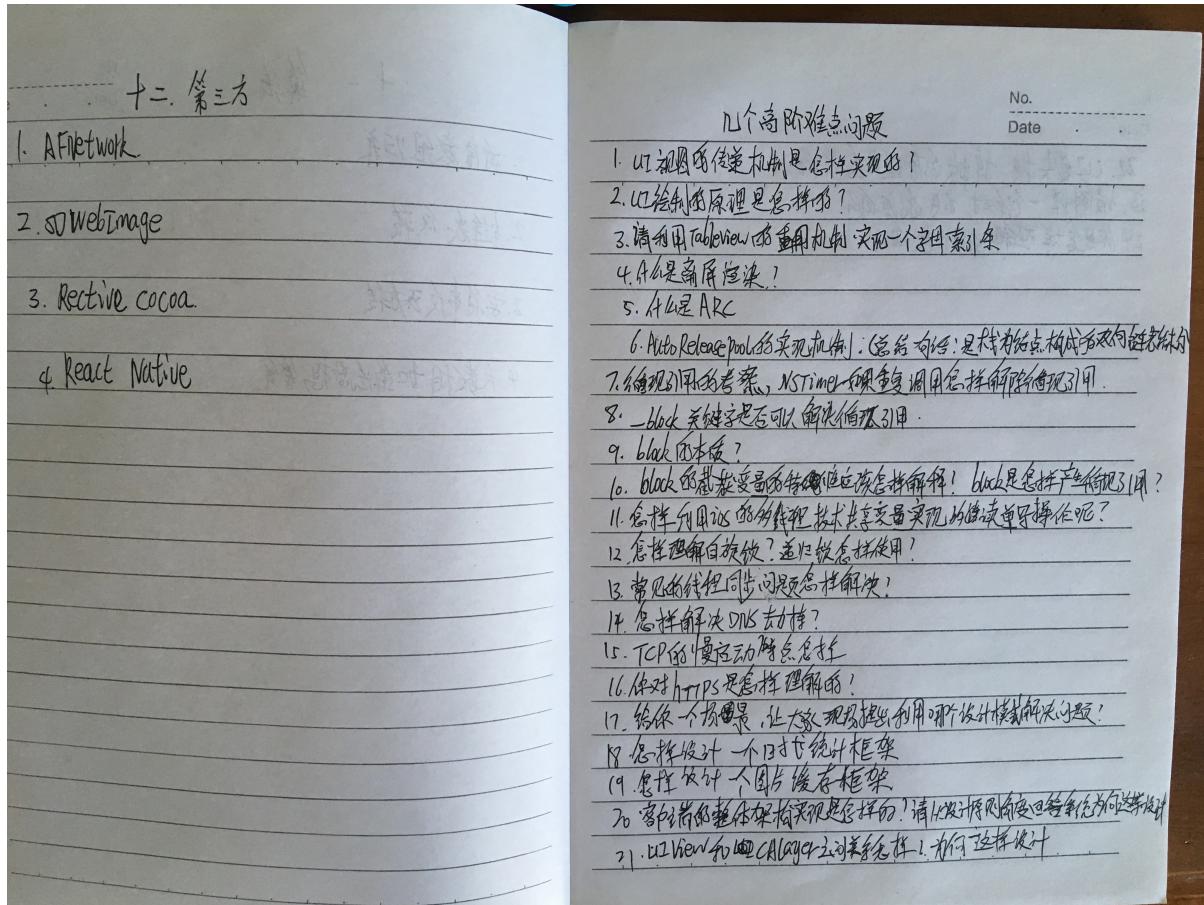


3.iOS Object-C 知识结构图3



| | |
|--|---|
| <p style="text-align: center;"><u>四. 内存管理</u></p> <ol style="list-style-type: none"> 1. weak自动置 nil 2. ARC 3. MRC 4. 自动释放池原理 5. 循环引用 6. 引用计数管理 | <p style="text-align: center;"><u>五. block</u></p> <p>No. _____ Date _____</p> <ol style="list-style-type: none"> 1. 萝获变量特性 2. -block关键字 3. block的本质 4. block的内存管理 5. 循环引用 <p>block的本质：是一个函数指针 加上一个对应上下文的结构体</p> |
| <p style="text-align: center;"><u>六. 的线程</u></p> <p>No. _____ Date _____</p> <ol style="list-style-type: none"> 1. NSOperation & NSOperationQueue 2. NSThread 3. GCD <p>往往给实际代码看对线程的掌握程度
常见锁的考察</p> <p>NSTlock</p> <p>递归锁</p> <p>自旋锁</p> <p>条件锁</p> | <p style="text-align: center;"><u>七. runloop</u></p> <p>No. _____ Date _____</p> <ol style="list-style-type: none"> 1. runloop为什么会有事情做没事休息，如何实现的 2. 怎样实现一个单线程 3. runloop与线程的关系是怎样 |







< Cocoa China >

编译信息写入辅助文件，创建文件架构 .app 文件

处理文件打包信息

执行 CocoaPod 编译前脚本，
checkPods Manifest.lock

编译.m文件，使用 CompileC 和
clang 命令

链接需要的 Framework

编译 xib

拷贝 xib，资源文件

编译 ImageAssets

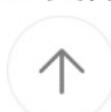
处理 info.plist

执行 CocoaPod 脚本

拷贝标准库

创建 .app 文件和签名

在 Xcode 中查看 clang 编译 .m 文件
的过程



面试指导

====

(一) 面试之前的准备

- 1.上该公司的网站，浏该公司的业务主营方向，目前的融资情况，历史，公司文化等，重中之重就是地址。（当然，这些在你投简历的时候就该了解，地址的话，面试通知里面也会写，不过确认一下总没坏处。）
- 2.洗个澡，换身干净的衣服，猿们注意刮胡子，媛们来个淡妆，千万不要喷香水，因为并不是所有人都喜欢的。
- 3.面试前要吃一些简单无味的东西，大蒜、洋葱等不要吃，注意不要太饱，不然脑子容易犯困。
- 4.提前查询好线路并规划出足够多的时间，稳重的去面试总比大汗淋漓的强得多。
- 5.不出意外你会提前了到达公司门，在附近转转，看看周围的环境，周围的饮食啊，交通，标志性建筑等，毕竟如果进公司，这些与你每天的工作是息息相关的。（面试时可以在最后把话题引向公司楼下的饮食或者一些设施，暗示你是个守时并且热爱生活的人。）
- 6.最好提前15分钟出现在公司的门口，跟前台的姐姐要礼貌，不要紧张，记得观察一下公司的硬件环境，公司员工的状态等，然后耐心的等待hr或者技术的面试。

(二) 介绍一下你自己

- 这是面试官100%会问的问题，一般人回答这个问题过于平常，只说姓名、年龄、爱好、所学专业等，如果你用一分钟来重复你的简历，那么，你的印象加分没有了！
- 不妨坦诚自信地展现自我，重点突出与应聘职位相吻合的优势。你的相关能力和素质是企业最感兴趣的信息。因为在许多情况下，在听取你的介绍时，面试官也会抓住他感兴趣的点深入询问。所以，在进行表述时，要力求以真实为基础，顾及表达的逻辑性和条理性，避免冗长而没有重点的叙述。一定要在最短的时间内激发起面试官对你的好感。并且企业很重视一个人的礼貌，求职者要尊重考官，在回答每个问题之后都说一句“谢谢”，企业喜欢有礼貌的求职者。

(三) 为何之前要离职 只谈发展不偏执

- 这个问题的回答很有技巧，这个问题决定了你是否能够愉快的融入一个团队。
- 回答这个问题时一定要小心，就算在前一个工作受到再大的委屈，对公司有多少的怨言，都千万不要表现出来，尤其要避免对公司本身主管的批评，避免面试官的负面情绪及印象。
- 建议此时最好的回答方式是将问题归咎在自己身上，例如觉得工作没有学习发展的空间，自己想在面试工作的相关产业中多加学习，或是前一份工作与自己的生涯规划不合等等，回答的答案最好是积极正面的。
 - 1)最重要的是：应聘者要使找招聘单位相信，应聘者在过往的单位的“离职原因”在此家招聘单位里不存在。
 - 2)避免把“离职原因”说得太详细、太具体。
 - 3)不能掺杂主观的负面感受，如“太辛苦”、“人际关系复杂”、“管理太混乱”、“公司不重视人才”、“公司排斥我们某某的员工”等。
 - 4)但也不能躲闪、回避，如“想换换环境”、“个人原因”等。
 - 5)不能涉及自己负面的人格特征，如不诚实、懒惰、缺乏责任感、不随和等。
 - 6)尽量使解释的理由为应聘者个人形象添彩。

(四)为什么来北京找工作？

- 面试官对异地求职者90%都会问的问题，主要考察你是否稳定，个人经验能力之外，排在第一位的就是稳定性，如

- 果不够稳定，那么其余都是空谈。
- 注意：不要说以前公司有多么不好。也不要说哪个哥们混的很不错，羡慕才来北京。因为企业招人想要的都是能够长期工作的人，可能哪个哥们哪天在别的地方又混的更好了，你是不是还要跳槽？所以，只要说来学习更多新技术和管理经验就够了。

(五) 不卑不亢不紧张

- 首先你要明白，面试是一个双选的过程。
- 1.不仅仅把面试当做面试官对自己的考练，对你来说，你同时也在面试这家公司。如果你对面试官的说法真的深有疑惑，大胆提出来，不要委曲求全，否则你未来的工作就会像你今天面试一样的状态在进行——你能坚持多久呢？今天你在工作面前妥协了，明天你就会为自己的妥协懊悔不已。
- 2.抱着“找一份工作，自己能做十年的工作”，这样的心态去面试。不谈面试官，不谈领导，单单是对一个求职人员，也要对自己负责。你想成为朝三暮四的人吗？你想每天都在担心着自己哪天会离开，渴望追求真正自我但却畏缩缩的情绪中度过吗？
- 3.面试官和面试者的关系并非敌对，相反，你应该让他感觉到一种朋友一般的相处方式和态度。没人愿意和敌人一起共事，却很愿意和朋友一起追求梦想。在这个基础上，你应该用微笑和语速以及条理来赢得印象分。
- 最后，也别太把自己当回事，毕竟自己是来谋得一份发挥才能的职位，没必要搞得别人来求着你去拯救世界一样。

(六) 工作生活两不误 薪资也要靠得住

- 薪资取决于你对于公司的价值。
- 如果你对薪酬的要求太低，那显然贬低自己的能力；如果你对薪酬的要求太高，那又会显得你分量过重，公司受用不起。一些雇主通常都事先对求职的职位定下开支预算，因而他们第一次提出的价钱往往是他们所能给予的最高价钱，他们问你只不过想证实一下这笔钱是否足以引起你对该工作的兴趣。
- 然后一定要记住要问清，薪资是税前还是税后，五险一金是多少，公司是否还有额外的商业保险或意外险等，公司的其他福利如何等等。

(七) 你五年之内的职业规划是什么

- 这是每一个应聘者都不希望被问到的问题，但是几乎每个人都会被问到，比较多的答案是“管理者”。但是近几年来，许多公司都已经建立了专门的技术途径。这些工作地位往往被称作“顾问”、“参议技师”或“高级软件工程师”等等。
- 当然，说出其他一些你感兴趣的职位也是可以的，比如产品销售部经理，生产部经理等一些与你的专业有相关背景的工作。要知道，考官总是喜欢有进取心的应聘者，此时如果说“不知道”，或许就会使你丧失一个好机会。最普通的回答应该是“我准备在技术领域有所作为”或“我希望能按照公司的管理思路发展”。
- 大部分面试官司都会问你是否有职业规划，这个问题的背后是了解你的求职动机和对自己中长期职业发展的思考。在回答这个问题之前，要对自己有个清晰的认识，知道自己想往哪个方向发展以及未来有什么计划，要给面试官一种积极向上，好学上进，有追求，有规划的感觉，面试官喜欢有规划的求职者。

(八) 情怀一定要伟大 但是不要说大话

- 这个问题，主要是看你长期的职业发展方向和公司符合不。另外可能考察你是否有激情。（有激情的人谈到自己的理想往往会展现出自我）
- 能问你这个问题，并且你感觉该公司的项目以及环境都是可以接受的，说明这个公司值得考虑。

- 要特别认真对待这个问题。每一个伟大的公司或者企业家都是怀揣着一个改变人类的梦想。俗话说的好，没有梦想，跟咸鱼有什么区别。但是回答这个问题的同时，也要说明自己不是一个仅有热血的无头苍蝇，需要表达出自己有一份敢打敢拼的心就好，要说出你肯为公司干十年八年革命的情怀，但也不要太浮夸。

(九) 你对加班怎么看

- 不用担心，实际上好多公司问这个问题，并不证明一定要加班。
- 只是有时候项目多了要加班，只是想测试你是否愿意为公司奉献。千万不要幼稚的反问面试官：“加班是否有加班费”？更何况加班费在如今的IT行业私企之中已经不复存在了，只有为数不多的公司还有。看到这里，肯定有人说，既然这样，那我就答应愿意加班，上次还有个学员在简历中居然写到：“喜欢加班”！太假了。你这样回答，如果你入职了，公司可能让你往死加班，谁让你当初愿意加班。所以说这个问题，要体现在你可以接受加班，但是你会尽量在正常的工作时间完成当天的任务，尽量避免加班的情况。

(十) 如果工作中与领导发生了分歧

- 想必只有傻子会说坚持自己或者说无条件服从领导
- 1)原则上我会尊重和服从领导的工作安排，同时私底下找机会以请教的口吻，婉转地表达自己的想法，看看领导是否能改变想法。
- 2)如果领导没有采纳我的建议，我也同样会按领导的要求认真地去完成这项工作。
- 3)还有一种情况，假如领导要求的方式违背原则，我会坚决提出反对意见，如领导仍固执己见，我会毫不犹豫地再向上级领导反映。

(十一) 你有最大的缺点

- 被面试官问的概率很大，也是HR的杀手锏和狠招，这个问题最难回答，通常面试官不希望听到求职直接回答的缺点是什么。
- 如果求职者说自己小心眼、脾气大、工作效率低，企业肯定不会录用你。不要自作聪明地回答“我最大的缺点就是过于追求完美”，有的人以为这样回答会显得自己比较出色，但事实上，他已经岌岌可危了。面试官喜欢求职者从自己的优点说起，中间加一些小缺点，最后再把问题转到优点上，突出优点的部分，面试官喜欢聪明的求职者。
- 这个问题举个例子：这个问题好难回答啊！我想……（亲和力表现，也缓解了自己的紧张情绪）我的缺点是比较执着，比如在技术方面比较爱钻研，有的时候会为一个技术问题加班到深夜。还有就是，工作比较按部就班，总是按照项目经理的要求完成任务。另外的缺点是，总在自己的工作范围内有创新意识，并没有扩展给其他同事。这些问题我想我可以进入公司后以最短的时间来解决，我的学习能力很强，我相信可以很快融入公司的企业文化，进入工作状态。我想就这些吧。

(十二) 你能为公司带来什么

- 企业很想知道未来的员工能为企业做什么，求职者除了要阐明自己的优势外，还应该说一些业务以外的事情
- 如：“就我的能力，我可以做一个优秀的员工在组织中发挥能力，给组织带来高效率和更多的收益”。企业喜欢求职者就申请的职位表明自己的能力，比如申请营销之类的职位，可以说：“我可以开发大量的新客户，同时，对老客户做更全面周到的服务，开发老客户的新需求和消费。”
- 除此之外，最好还要阐明自己希望找一份稳定而持久的工作。其实有的面试官就是想知道面试者对这份工作的热忱及理解度，并筛选因一时兴起而来应聘的人，如果你是在这个领域是无经验者，也可以强调“就算职种不同，也希望有机会发挥之前的经验”，所以你在面试时候一定要对该公司做好备课，往往公司不希望自己的公司成为别人的职业跳板。

(十三) 你对我们公司了解多少

- 首先去面试前先做功课，了解一下该公司的背景，让对方觉得你真的很有心想得到这份工作，而不仅仅是探探路。
- 对于这个问题，你要格外小心，如果你已经对该单位作了研究，你可以回答一些详细的原因，像“公司本身的高技术开发环境很吸引我。”、“我同公司出生在同样的时代，我希望能够进入一家与我共同成长的公司。”、“你们公司

一直都稳定发展，在近几年来在市场上很有竞争力。”、“我认为贵公司能够给我提供一个与众不同的发展道路。”这都显示出你已经做了一些调查，也说明你对自己的未来有了较为具体的远景规划。

(十四) 谈谈你对行业、技术发展趋势

- 企业对这个问题很感兴趣，只有有备而来的求职者能够过关。
- 求职者可以直接在网上查找对你所申请的行业部门的信息，只有深入了解才能产生独特的见解。企业认为最聪明的求职者是对所面试的公司预先了解很多，包括公司各个部门，发展情况，在面试回答问题的时候可以提到所了解的情况，企业欢迎进入企业的人是“知己”，而不是“盲人”。

(十五) 你有什么业余爱好

- 这个问题，据实回答即可，如若你没有，那就编一个吧，这个问题说明你是否热爱生活。
- 没有哪个企业希望找个生活中十分无趣的员工，因为这个会影响员工本人及同事们的工作心情及状态。如果没有，可以说一些自己喜欢音乐、电影等常规兴趣。如果有一些比较有趣的业余爱好，如果正好与面试官相同，也是一个很好的机会。
- 当然，如果你要说你的业余爱好是敲代码，写轮子，那就更加的好，不过，这也需要你说一下，你到底做过哪些轮子，所以该话术不要随意使用

(十六) 这几年工作中令你骄傲的事

- 这是考官给你的一个机会，让你展示自己把握命运的能力。
- 这会体现你潜在的领导能力以及你被提升的可能性。假如你应聘于一个服务性质的单位，你很可能会被邀请去午餐。记住：你的前途取决于你的知识、你的社交能力和综合表现。

(十七) 什么时候能入职

- 大多数企业会关心就职时间，最好是回答“如果被录用的话，到职日可按公司的规定上班”，如果还未辞去上一个工作，但上班时间又太近，似乎有些强人所难，因为交接至少要一个月的时间，应进一步说明原因，录取公司应该会通融的。

(十八) 最近关注过的新技术是什么

- 如果被hr问到这个问题的时候，不要紧张，因为hr并不是很懂技术，而且可以说是不懂，他只是会记录下来，汇总之后给技术总监看。
- 所以，回答这个问题不要紧张，如果之前有准备，那你就大谈特谈，直到他打断你为止。如果没有准备，那就说一些当下比较流行的新技术，技术面如果不是当天面试，就有了充足的时间，为你下次面试做准备。

(十九) 最近有看过哪些书

- 这个问题，面试官主要是想考察面试者是否是一个具有良好学习心态并且会看你是否是一个钻研技术的人。所以这个问题你要是说最近看了四大名著。。。恭喜你。再见。
- 所以，这个问题应该提前准备一些书的名字和内容简介，如果真的没准备，也想想之前有看过什么关于技术方面的书籍等，或者说你最近都在参加一些交流会啊等等。温故知新，有一者都可。

(二十) 你还有什么问题问我吗

- 这个问题看上去可有可无，其实很关键，面试官不喜欢说“没有问题”的人，没有问题就是自寻死路，没有问题传达出你对公司缺乏兴趣，而只是来寻找一笔薪水。

- 其实在面试过程中谦虚礼貌的问面试官怎么称呼，该部门工作中的信息，如项目情况，开发技术再或者说贵公司的晋升机制是什么样的等。表现出一种很积极主动的状态是非常讨巧的。也可以更多的了解到自己来的工作环境。企业很欢迎这样的求职者，因为体现出你对学习的热情和对公司的忠诚度以及你的上进心。

总结性的谈谈面试18家遇到的坑

首先一定要冷静

- 不论遇到什么水平的面试官,至少得把比装完。
- 基本上有一小半的情况是碰上非技术的面试官和你打交道，所以一定要旁敲侧击让其了解你确实很懂技术，比如他们想实现什么功能，你就说一个第三方框架或是思路来实现这个功能。
- 有时候也会遇到那种不是技术面，但面试官也并不怎么急着招iOS，过去就是瞎扯淡的情况，这种时候就抱着放松一下的心态好了。
- 当然，遇到的技术面试官也是五花八门。
- 有的明显技术水平一般般，不怎么问你一些深入的细节性问题(怕暴露他自己水平)。
- 基本都是问各个技术点让你描述，或者让你谈谈你擅长的技术点或是项目里面用到的技术(一定要有自己说的非常6的技术点 我个人是ui优化)，基本上碰到的技术面试官有6成是这种情况。
- 基本上这也是你最容易通过技术面试的情况了，并不需要把每个技术点具体用了什么方法给背下来，太多了你也记不住，至少得记住每个技术点是什么意思，想要实现什么目的，项目中什么地方用到了这个技术点，以及用到这个技术点我是用的哪个第三方框架，最好是把这个第三方框架的底层实现原理给了解一下(比如SDWebImage框架)。
- 这些只是只需要百度一下，看那些技术博客就行，我一般是在面试的路上去看这些的
- 剩下的就是些刁钻的面试官了，基本上是问各种细节性的问题，甚至会丧心病狂逮着你问服务器的问题。
- 这时候也尽量把自己知道的回答出来，至少得回答出用了什么框架这种最基本的，实在不知道也只能说用到这个技术的项目过去太久了，忘了。
- 总之这些面试官可以给自己面试之路的知识面有比较好的补充，哪方面问题之前没注意到的话面试结束后就赶紧去看面试宝典或是百度吧。
- 人事面的时候，一定要把自己的工作经历，离职原因，上家公司的待遇(包括工资构成等)，大学的专业(有必要再说，如果是一些完全用不到编程的专业还是轻描淡写带过去吧)，这些都需要事先想好怎么说，比较万能的离职原因无非就是上家公司项目太简单了，发挥不了自己真正实力云云。

某人拿到offer的面试的几个问题：

我： 你好，请问怎么称呼？

面试官：我是XX公司的项目经理

我：经理你好，我叫xxx，xx年毕业自xxx，专业是xxx，大学时候专业有用到c#做一些门户网站，那时候觉得编程挺有趣 于是毕业之后就转行做了当时比较火的ios 去了xxx公司 上家公司主要做外包项目 都是些电商类类型的项目 今年xx月离职 因为深圳整体互联网氛围更好 所以就来到深圳找工作

面试官：说说你对runtime的理解吧（可能是因为当时比较主动，所以面试的氛围也比较好，很轻松）

我：rumtime是运行时库 基于c语言的api接口 作用是动态的创建一个类 动态的添加属性和方法 遍历属性和方法名 动态修改属性和方法等等 （具体的忘记了，总之当时双方气氛比较融洽，就扯项目中用到的地方(字典转模型的时候遍历 nscoding归档解档的时候遍历 利用runtime拦截系统的init方法 kvo的底层原理是通过runtime动态生成一个子类然后去监听) 然后再扯一下class_copy开头的几个runtime方法(copyivarlist copymethod list)，扯了很久，现在想想可能是因为我把运行时瞎扯了很多到项目中的例子，让面试官觉得特别好评，其他三家拿到offer的也是这个情况）。

面试官：挺好 我刚刚看你的项目介绍说你用了tableView的优化 你是怎么做的？

我：（窃喜，这个问题都准备好几百次了）

- 1.cell重用
- 2.dequeueReusableCellWithIdentifier:forIndexPath:(会调用heightForRowAtIndexPath) 和 dequeueReusableCellWithIdentifier (后面这个不会再次调用heightForRowAtIndexPath)
- 3.tableView在cell显示之前会调用heightForRowAtIndexPath,有多少个cell就会调用多少次，算contentSize
- 4.使用了预估行高，并不会再显示之前去计算获取所有的行高，根据预估行高和实际行高来获取cell的行高，先根据预估行高计算好要先获取几个cell，如果计算的这几个cell高度确实够（高度能超出屏幕的高度就不计算了。如果不够还会计算），目的也是让contentSize大于屏幕，就能滚动，后面要显示，才来计算行高，会发现滚动条会跳
- 5.cellForRowAtIndexPath不要做耗时操作
 - 1.读取文件，写入文件，最好是放到子线程，或先读取好，在tableView去显示
 - 2.解压资源
 - 3.尽量少得计算计算，最好是先计算好，cellForRowAtIndexPath只做显示
 - 4.尽量不要去添加和移除view，现将会用到的控件懒加载，要就显示，不要就隐藏
- 6.tableView滚动的时候，不要去做动画
- 7.cell里面的控件，约束最好不要使用remake，动态添加约束是比较耗性能的
- 8.cell里面的控件，背景最好是不透明的（图层混合），view的背景颜色 clearColor 尽量少
- 9.图片圆角不要使用 layer.cornerRadius
- 10.图层最好不要使用阴影，阴影会导致离屏渲染
- 11.异步绘制
- 12.栅格化
- 13.AsyncDisplayKit -> 不使用UIKit (UIView) -> (Node)
- 14.借助工具来测试性能（当然这些是我自己整理出来的，当时我说的时候像第5、10、12都忘记说了，这里只是给出来自己整理的笔记，当然无论什么点都结合到了项目去说。总体也就是 XX地方XX用，遇到了XX问题，XX解决就大概这么个套路）

面试官：（上述问题我至少说了20分钟）恩，非常好，....

我：（接下来的就忘记了，都是一些瞎聊 聊公司 聊语言 聊兴趣爱好什么的 就是瞎聊 因为我前面已经把他给征服了 所以后面的瞎聊环节就自己都忘记了 面试那么多家，聊天这么个本事实在是见长，面多了也就那么回事，虽然我一开始也挺害怕的 面试10家之后也很想骂娘 一切都是坚持吧）

另外再贡献出来我对SDWebImage的一些见解吧 在另外一家公司聊的 我觉得也挺不错的

先是显示一个占位图 然后会根据url来处理图片 处理图片的大致流程是先从缓存中查找 然后去硬盘中查找(根据URLKey) 如果硬盘中有的话 就会先把图片添加到缓存中 然后显示这张图片 如果硬盘中也没有的话 就会根据url去由NSURLConnection异步下载 根据相关代理方法来判断图片下载中 下载完成以及下载失败几种情况 最后通过NSOperationQueue在子线程交给SDWebImageDecoder进行图片解码处理 最终通过相关的代理方法告知 SDWebImageManager图片下载完毕 然后会把图片同时保存在内存和硬盘中

iOS 面试重点目录

1.iOS基础

- 1.常见property属性 (readwrite, readonly, assign, weak, retain, copy, nonatomic, atomis等) , 及作用。
- 2.#import 跟#include、@class有什么区别? # import<> 跟 #import""又什么区别?
- 3.OC有多继承吗? 没有的话用什么代替?
- 4.Objective-C如何对内存管理的?内存管理的原则是?
- 5、Object C中创建线程的方法是什么?如果在主线程中执行代码, 方法是什么?如果想延时执行代码、方法又是什么?
- 6.浅复制和深复制的区别?
- 7、分类的作用? 分类和继承的区别?
- 8、frame和bounds有什么不同?
- 9、HTTP协议中, POST和GET的区别是什么?
- 10、视图控制器的生命周期方法调用顺序?
- 11.App的生命周期调用?
- 12.什么时候使用自动释放池。
- 13、iOS怎么做数据的持久化?
- 14、APNS的推送机制
- 16、UITableView的数据源方法和代理方法?
- 17.设计模式有哪些, Notification, 单例模式, KVO, KVC, 代理, target-action.
- - 1. block使用时的注意点
- 19.引用计数器 (ARC 和 MRC)
- 20.线程和进程
- 21.堆和栈的区别
- 22.UDP和TCP区别

原理知识

- 1.Block 相关知识
- 2.GCD原理
- 3.runtime
- 4.Runloop
- 5.AutoreleasePool
- KVO内部实现原理
- 加密方式及原理, MD5, RSA, DES, AES, Base64

UI相关知识

- 1.为什么更新UI要在主线程?
- 2.iOS事件响应链
- 3.UITableView的优化方法
- 4.UI如何布局
- 5.Frame和Bounds区别
- 6.UI绘制

网络知识

- 1.说出几个http错误码
- 2.HTTP和HTTPS区别
- 3.HTTPS连接方式
- 4.如何保证网络安全性

高级知识

- 1.组件化
- 2.MVC, MVVM, MVP
- 3.内存区的区别。
- 4.NSTimer为什么不准？如何设计更准
- 5.APP的性能优化
- 6.内存泄露的检查和循环引用
- 7.卡顿和掉帧原理，及如何优化。
- 8.锁与多线程
- 9.同步，异步，主线程，子线程各种使用。
- 10.崩溃检查处理
- 11.编译过程
- 12.离屏渲染

第三方源码

1. AFNetworking
2. SDWebImage
3. React Native

算法的考察

我遇到的面试题

1. 算法：16进制转10进制。 (美团出行)
2. 算法：单链表检查是否有环，未追问检查环位置 (贝壳||美团)
3. 算法：N叉数深度 (美团)
4. 算法：有15个瓶子，其中最多有一瓶有毒，现在有四只老鼠，喝了有毒的水之后，第二天就会死。如何在第二天就可以判断出哪个瓶子有毒 (滴滴客户端)
5. 算法：给300亿地图数据点，如何快速找到用户当前周边20个点。 (美团)
6. 算法：找出倒数第n个节点 (别人遇到的)
7. 算法：镜像二叉树 (别人遇到的)
8. 算法：写一个快速排序 (爱奇艺)
9. 算法：n个数里面查找中位数，例如 (5, 6, 2, 7, 1, 4, 3)，找出4。 (爱奇艺)
(算法一般只需要讲思路就行，即便让写)

此博客覆盖了不少题，可以看一下：

<https://cloud.tencent.com/developer/article/1165416>

- 1.git rebase 和 git merge的区别
- 2.block修饰， strong和copy区别
- 3.FPS指什么，怎么实现的
- 4.weak 处理
- 5.runtime 项目中的使用 (1.category 2.字典转模型 3.exchange 4.崩溃三次尝试拦截)
 - 1). 追问:+load方法调用时机及为什么在这种时机
 - 2). 追问: +load方法，一个class的4个category都有+load方法，调用顺序。为什么？
 - 3). 追问: 接2，如果每个category都转换了系统方法，例如viewDidAppear，则最终结果怎样，如何避免。
 - 4.) 追问: class的isa方法查找流程，实例方法和类方法
 - 5.) 追问: category内存处理机制
 - 6.) exchange方法交换 和 replace 方法替换 的区别
- 7.) 你知道有哪些情况会导致app崩溃，分别可以用什么方法拦截并化解
6. 应用从点击到启动时调用逻辑。
- 7.GCD相关知识，什么是GCD，信号量，group，栅栏等 1). 追问，这些使用时，如何选择 2). 追问，group是怎样使用的，(除了dispatch_group外，还需要知道enter和level的用法) 3). 追问，GCD的队列 (dispatch_queue_t) 怎么管理队列的。队列不是先进先出吗，这里怎么不是？
- 8.线程和进程的区别
- 9.category和extention的区别

10. 遇到过一次代码风格面试（爱奇艺）

```
typedef enum{
    UserSex_Man,
    UserSex_Woman
}UserSex;

@interface UserModel :NSObject

@property(nonatomic, strong) NSString *name;
@property (assign, nonatomic) int age;
@property (nonatomic, assign) UserSex sex;

-(id)initUserModelWithUserName: (NSString*)name withAge:(int)age;

-(void)doLogIn;

@end
```

11.以下打印结果是什么：（爱奇艺）

```
 NSLog(@"1");
dispatch_async(dispatch_get_main_queue(), ^{
    NSLog(@"2");

    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(0 * NSEC_PER_SEC)), dispatch_get_main_queue(), ^{
        NSLog(@"3");

        dispatch_async(dispatch_get_global_queue(0, 0), ^{
            NSLog(@"4");
        });
    });

    dispatch_async(dispatch_get_global_queue(0, 0), ^{
        NSLog(@"5");
    });
});

dispatch_async(dispatch_get_global_queue(0, 0), ^{
    NSLog(@"6");
});

NSLog(@"7");
```

打印结果如下：

```
2018-09-11 17:25:17.420211+0800 TestOCDemo[1293:7549260] 1
2018-09-11 17:25:17.420341+0800 TestOCDemo[1293:7549260] 7
2018-09-11 17:25:17.420350+0800 TestOCDemo[1293:7549296] 6
2018-09-11 17:25:17.425499+0800 TestOCDemo[1293:7549260] 2
2018-09-11 17:25:17.425643+0800 TestOCDemo[1293:7549296] 5
2018-09-11 17:25:17.426039+0800 TestOCDemo[1293:7549260] 3
2018-09-11 17:25:17.426153+0800 TestOCDemo[1293:7549296] 4
```

12.沙盒目录下有几个文件夹。NSUserDefaults存储格式，及在哪个文件夹下。

1).追问：版本升级会保留哪些文件夹

13.性能优化之类的

1.)如何进行的性能优化

2.) FPS静态是多少，滑动列表是多少

3.) 你的项目crash率多少

13.观察者的底层逻辑

14.CALayer和UIView

15.autoreleasepool的实现原理，考察autoreleasepoolPage。 1).追问2个@autoreleasepool{}会生成几个autoreleasepoolPage。

16.SDWebImage 流程及架构模式

17.https连接流程 (TLS/SSL) 1).非对称加密 2).说出一些常用的错误码及对应信息 3).什么是网络堵塞和流畅

18.timer 在主线程启动，在子线程关闭，会出现什么问题。

19.能否向编译后得到的类中增加实例变量？能否向运行时创建的类中添加实例变量？为什么？

20.ARC内存管理

1.) 追问什么时机增加autorelease

21.内存区（静态区，堆区，栈区，常量区）

22.自己写runtime解析属性时，比如一个integer属性，如何解析。

23.设计准的timer

1.)displaylink

2.)gcd_timer

24.面向对象编程，面向切面编程，面向接口编程，(彩琴遇到的，)

25.关联对象有什么应用，系统如何管理关联对象？其被释放的时候需要手动将其指针置空么？

26.离屏渲染原理

https://blog.ibireme.com/2015/11/12/smooth_user_interfaces_for_ios/

27.NSOperation的一些知识。例如（追问了， NSOperation能pause吗）

28.说出你所知道的设计模式。（当然不仅仅指的是当前那些常用的，而是那些不常用的，比如策略模式，抽象模式等，工厂模式等等）

1.)可能会追问

29.静态库和动态库区别

30.响应链和查找第一响应

1.)追问，如何扩大button的点击区域 (hitTest:event:)

2.)追问，在A View 上add B view，B view的frame 在A的外面，点击会如何。

其他：(weak被问的很多，可以按照此深度准备)

为什么 iOS 开发没人要了?

从历年 weak 看 iOS 面试:

2013年

面试官: 代理用 weak 还是 strong ?

我 : weak 。

面试官: 明天来上班吧

2014年

面试官: 代理为什么用 weak 不用 strong?

我 : 用 strong 会造成循环引用。

面试官: 明天来上班吧

2015年

面试官: weak 是怎么实现的?

我 : weak 其实是 系统通过一个 hash 表来实现对象的弱引用

面试官: 明天来上班吧

2016年

面试官: weak 是怎么实现的?

我 : runtime 维护了一个 weak 表, 用于存储指向某个对象的所有 weak 指针。weak 表其实是一个 hash (哈希) 表, key 是所指对象的地址, value 是 weak 指针的地址 (这个地址的值是所指对象指针的地址) 数组。

面试官: 明天来上班吧

2017年

面试官: weak 是怎么实现的?

我 :

初始化时: runtime 会调用 `objc_initWeak` 函数, 初始化一个新的 weak 指针指向对象的地址。

添加引用时: `objc_initWeak` 函数会调用 `storeWeak()` 函数, `storeWeak()` 的作用是更新指针指向, 创建对应的弱引用表。

释放时, 调用 `clearDeallocating` 函数。`clearDeallocating` 函数首先根据对象地址获取所有 weak 指针地址的数组, 然后遍历这个数组把其中的数据设为 nil, 最后把这个 entry 从 weak 表中删除, 最后清理对象的记录。

面试官: 明天来上班吧

2018年

面试官: weak 是怎么实现的?

我 : 跟 2017 年说的一样, 还详细补充了 `objc_initWeak`, `storeWeak`, `clearDeallocating` 的实现细节。

面试官: 小伙子基础不错。13k , 996 干不干? 干就明天来上班... 下一个

2019年

面试官: weak 是怎么实现的?

我 : 别说了, 拿纸来, 我动手实现一个。

写完后

面试官: 小伙子不错, 我考虑考虑, 你先回去吧么实现的?

我 : 别说了, 拿纸来, 我动手实现一个。

写完后

面试官: 小伙子不错, 我考虑考虑, 你先回去吧