#####1、设计模式是什么？ 你知道哪些设计模式，并简要叙述?

设计模式是一种编码经验，就是用比较成熟的逻辑去处理某一种类型的事情。

1). MVC模式：Model View Control，把模型 视图 控制器 层进行解耦合编写。

2). MVVM模式：Model View ViewModel 把模型 视图 业务逻辑 层进行解耦和编写。

3). 单例模式：通过static关键词，声明全局变量。在整个进程运行期间只会被赋值一次。

4). 观察者模式：KVO是典型的通知模式，观察某个属性的状态，状态发生变化时通知观察者。

5). 委托模式：代理+协议的组合。实现1对1的反向传值操作。

6). 工厂模式：通过一个类方法，批量的根据已有模板生产对象。

#####2、MVC 和 MVVM 的区别?

1). MVVM是对胖模型进行的拆分，其本质是给控制器减负，将一些弱业务逻辑放到VM中去处理。

2). MVC是一切设计的基础，所有新的设计模式都是基于MVC进行的改进。

3、#import跟 #include 有什么区别，@class呢，#import<> 跟 #import“”有什么区别?

答:

1). #import是Objective-C导入头文件的关键字，#include是C/C++导入头文件的关键字，使用#import头文件会自动只导入一次，不会重复导入。

2). @class告诉编译器某个类的声明，当执行时，才去查看类的实现文件，可以解决头文件的相互包含。

3). #import<>用来包含系统的头文件，#import”“用来包含用户头文件。

#####4、frame 和 bounds 有什么不同?

frame指的是：该view在父view坐标系统中的位置和大小。(参照点是父view的坐标系统)

bounds指的是：该view在本身坐标系统中的位置和大小。(参照点是本身坐标系统)

#####5、Objective-C的类可以多重继承么？可以实现多个接口么？Category是什么？重写一个类的方式用继承好还是分类好？为什么？

答：Objective-C的类不可以多重继承；可以实现多个接口（协议）；Category是类别；一般情况用分类好，用Category去重写类的方法，仅对本Category有效，不会影响到其他类与原有类的关系。

#####6、@property 的本质是什么？ivar、getter、setter 是如何生成并添加到这个类中的？

@property 的本质是什么？

@property = ivar + getter + setter;

“属性”（property)有两大概念：ivar（实例变量）、getter+setter（存取方法）

“属性”（property)作为 Objective-C 的一项特性，主要的作用就在于封装对象中的数据。Objective-C 对象通常会把其所需要的数据保存为各种实例变量。实例变量一般通过“存取方法”(access method)来访问。其中，“获取方法”（getter)用于读取变量值，而“设置方法”(setter)用于写入变量值。

#####7、@property中有哪些属性关键字？/ @property 后面可以有哪些修饰符？

属性可以拥有的特质分为四类:

1.原子性--- nonatomic 特质

2.读/写权限---readwrite(读写)、readonly (只读)

3.内存管理语义---assign、strong、 weak、unsafe_unretained、copy

4.方法名---getter=<name> 、setter=<name>

5.不常用的：nonnull,null_resettable,nullable

#####8、属性关键字 readwrite，readonly，assign，retain，copy，nonatomic 各是什么作用，在那种情况下用？

答：

1). readwrite 是可读可写特性。需要生成getter方法和setter方法。

2). readonly 是只读特性。只会生成getter方法，不会生成setter方法，不希望属性在类外改变。

3). assign 是赋值特性。setter方法将传入参数赋值给实例变量;仅设置变量时,assign用于基本数据类型。

4). retain(MRC)/strong(ARC) 表示持有特性。setter方法将传入参数先保留，再赋值，传入参数的retaincount会+1。

5). copy 表示拷贝特性。setter方法将传入对象复制一份，需要完全一份新的变量时。

6). nonatomic 非原子操作。决定编译器生成的setter和getter方法是否是原子操作，atomic表示多线程安全，一般使用nonatomic，效率高。

#####9、什么情况使用 weak 关键字，相比 assign 有什么不同?

1.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。

2.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 IBOutlet 控件属性一般也使用 weak；当然，也可以使用strong。

IBOutlet连出来的视图属性为什么可以被设置成weak?

因为父控件的subViews数组已经对它有一个强引用。

不同点：

assign 可以用非 OC 对象，而 weak 必须用于 OC 对象。

weak 表明该属性定义了一种“非拥有关系”。在属性所指的对象销毁时，属性值会自动清空(nil)。

#####10、怎么用 copy 关键字?

用途：

1. 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；

2. block 也经常使用 copy 关键字。

说明：

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，但写上 copy 也无伤大雅，还能时刻提醒我们：编译器自动对 block 进行了 copy 操作。如果不写 copy ，该类的调用者有可能会忘记或者根本不知道“编译器会自动对 block 进行了 copy 操作”，他们有可能会在调用之前自行拷贝属性值。这种操作多余而低效。

#####11、用@property声明的 NSString / NSArray / NSDictionary 经常使用 copy 关键字，为什么？如果改用strong关键字，可能造成什么问题？

答：用 @property 声明 NSString、NSArray、NSDictionary 经常使用 copy 关键字，是因为他们有对应的可变类型：NSMutableString、NSMutableArray、NSMutableDictionary，他们之间可能进行赋值操作（就是把可变的赋值给不可变的），为确保对象中的字符串值不会无意间变动，应该在设置新属性值时拷贝一份。

1. 因为父类指针可以指向子类对象,使用 copy 的目的是为了让本对象的属性不受外界影响,使用 copy 无论给我传入是一个可变对象还是不可对象,我本身持有的就是一个不可变的副本。

2. 如果我们使用是 strong ,那么这个属性就有可能指向一个可变对象,如果这个可变对象在外部被修改了,那么会影响该属性。

//总结：使用copy的目的是，防止把可变类型的对象赋值给不可变类型的对象时，可变类型对象的值发送变化会无意间篡改不可变类型对象原来的值。

#####12、浅拷贝和深拷贝的区别？

答：

浅拷贝：只复制指向对象的指针，而不复制引用对象本身。

深拷贝：复制引用对象本身。内存中存在了两份独立对象本身，当修改A时，A_copy不变。

#####13、系统对象的 copy 与 mutableCopy 方法？

不管是集合类对象（NSArray、NSDictionary、NSSet ... 之类的对象），还是非集合类对象（NSString, NSNumber ... 之类的对象），接收到copy和mutableCopy消息时，都遵循以下准则：

1. copy 返回的是不可变对象（immutableObject）；如果用copy返回值调用mutable对象的方法就会crash。

2. mutableCopy 返回的是可变对象（mutableObject）。

一、非集合类对象的copy与mutableCopy

在非集合类对象中，对不可变对象进行copy操作，是指针复制，mutableCopy操作是内容复制；

对可变对象进行copy和mutableCopy都是内容复制。用代码简单表示如下：

NSString *str = @"hello word!";

NSString *strCopy = [str copy] // 指针复制，strCopy与str的地址一样

NSMutableString *strMCopy = [str mutableCopy] // 内容复制，strMCopy与str的地址不一样

NSMutableString *mutableStr = [NSMutableString stringWithString: @"hello word!"];

NSString *strCopy = [mutableStr copy] // 内容复制

NSMutableString *strMCopy = [mutableStr mutableCopy] // 内容复制

二、集合类对象的copy与mutableCopy (同上)

在集合类对象中，对不可变对象进行copy操作，是指针复制，mutableCopy操作是内容复制；

对可变对象进行copy和mutableCopy都是内容复制。但是：集合对象的内容复制仅限于对象本身，对集合内的对象元素仍然是指针复制。(即单层内容复制)

NSArray *arr = @[@[@"a", @"b"], @[@"c", @"d"];

NSArray *copyArr = [arr copy]; // 指针复制

NSMutableArray *mCopyArr = [arr mutableCopy]; //单层内容复制

NSMutableArray *array = [NSMutableArray arrayWithObjects:[NSMutableString stringWithString:@"a"],@"b",@"c",nil];

NSArray *copyArr = [mutableArr copy]; // 单层内容复制

NSMutableArray *mCopyArr = [mutableArr mutableCopy]; // 单层内容复制

【总结一句话】：

只有对不可变对象进行copy操作是指针复制（浅复制），其它情况都是内容复制（深复制）！

#####14、这个写法会出什么问题：@property (nonatomic, copy) NSMutableArray *arr;?

问题：添加,删除,修改数组内的元素的时候,程序会因为找不到对应的方法而崩溃。

//如：-[__NSArrayI removeObjectAtIndex:]: unrecognized selector sent to instance 0x7fcd1bc30460

// copy后返回的是不可变对象（即 arr 是 NSArray 类型，NSArray 类型对象不能调用 NSMutableArray 类型对象的方法）

原因：是因为 copy 就是复制一个不可变 NSArray 的对象，不能对 NSArray 对象进行添加/修改。

#####15、如何让自己的类用 copy 修饰符？如何重写带 copy 关键字的 setter?

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 NSCopying 与 NSMutableCopying 协议。

具体步骤：

1. 需声明该类遵从 NSCopying 协议

2. 实现 NSCopying 协议的方法。

// 该协议只有一个方法:

- (id)copyWithZone:(NSZone *)zone;

// 注意：使用 copy 修饰符，调用的是copy方法，其实真正需要实现的是 “copyWithZone” 方法。

#####15、写一个 setter 方法用于完成 @property (nonatomic, retain) NSString *name，写一个 setter 方法用于完成 @property (nonatomic, copy) NSString *name ?

答：

// retain

```

- (void)setName:(NSString *)str {
```

```
[str retain];

[_name release];

_name = str;

}
```

```
// copy
```

```
- (void)setName:(NSString *)str {

id t = [str copy];

[_name release];

_name = t;

}
```

#####16、@synthesize 和 @dynamic 分别有什么作用?

@property有两个对应的词，一个是@synthesize（合成实例变量），一个是@dynamic。

如果@synthesize和@dynamic都没有写，那么默认的就是 @synthesize var = _var;

// 在类的实现代码里通过 @synthesize 语法可以来指定实例变量的名字。(@synthesize var = _newVar;)

1. @synthesize 的语义是如果你没有手动实现setter方法和getter方法，那么编译器会自动为你加上这两个方法。

2. @dynamic 告诉编译器，属性的setter与getter方法由用户自己实现，不自动生成（如，@dynamic var）。

#####17、常见的 Objective-C 的数据类型有那些，和C的基本数据类型有什么区别？如：NSInteger和int

答:

Objective-C的数据类型有NSString，NSNumber，NSArray，NSMutableArray，NSData等等，这些都是class，创建后便是对象，而C语言的基本数据类型int，只是一定字节的内存空间，用于存放数值;NSInteger是基本数据类型，并不是NSNumber的子类，当然也不是NSObject的子类。

NSInteger是基本数据类型Int或者Long的别名(NSInteger的定义typedef long NSInteger)，它的区别在于，NSInteger会根据系统是32位还是64位来决定是本身是int还是long。

#####17、id 声明的对象有什么特性?

答：id 声明的对象具有运行时的特性，即可以指向任意类型的Objcetive-C的对象。

#####19、Objective-C 如何对内存管理的，说说你的看法和解决方法?

答：Objective-C的内存管理主要有三种方式ARC(自动内存计数)、手动内存计数、内存池。

1). 自动内存计数ARC：由Xcode自动在App编译阶段，在代码中添加内存管理代码。

2). 手动内存计数MRC：遵循内存谁申请、谁释放；谁添加，谁释放的原则。

3). 内存释放池Release Pool：把需要释放的内存统一放在一个池子中，当池子被抽干后(drain)，池子中所有的内存空间也被自动释放掉。内存池的释放操作分为自动和手动。自动释放受runloop机制影响。

#####20、Objective-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?

答：线程创建有三种方法：使用NSThread创建、使用GCD的dispatch、使用子类化的NSOperation,然后将其加入NSOperationQueue;在主线程执行代码，方法是performSelectorOnMainThread，如果想延时执行代码可以用performSelector:onThread:withObject:waitUntilDone:

#####21、Category（类别）、 Extension（扩展）和继承的区别

区别：

1. 分类有名字，类扩展没有分类名字，是一种特殊的分类。

2. 分类只能扩展方法（属性仅仅是声明，并没真正实现），类扩展可以扩展属性、成员变量和方法。

3. 继承可以增加，修改或者删除方法，并且可以增加属性。

#####22、我们说的OC是动态运行时语言是什么意思?

答：主要是将数据类型的确定由编译时，推迟到了运行时。简单来说, 运行时机制使我们直到运行时才去决定一个对象的类别,以及调用该类别对象指定方法。

#####23、为什么我们常见的delegate属性都用是week而不是retain/strong?

答：是为了防止delegate两端产生不必要的循环引用。

@property (nonatomic, weak) id<UITableViewDelegate> delegate;

#####24、什么时候用delete，什么时候用Notification?

Delegate(委托模式)：1对1的反向消息通知功能。

Notification(通知模式)：只想要把消息发送出去，告知某些状态的变化。但是并不关心谁想要知道这个。

#####25、什么是 KVO 和 KVC?

1). KVC(Key-Value-Coding)：键值编码 是一种通过字符串间接访问对象的方式（即给属性赋值）

举例说明：

stu.name = @"张三" // 点语法给属性赋值

[stu setValue:@"张三" forKey:@"name"]; // 通过字符串使用KVC方式给属性赋值

stu1.nameLabel.text = @"张三";

[stu1 setValue:@"张三" forKey:@"nameLabel.text"]; // 跨层赋值

2). KVO(key-Value-Observing)：键值观察机制 他提供了观察某一属性变化的方法，极大的简化了代码。

KVO只能被KVC触发，包括使用setValue:forKey:方法和点语法。

// 通过下方方法为属性添加KVO观察

```

- (void)addObserver:(NSObject *)observer
```

```
forKeyPath:(NSString *)keyPath
options:(NSKeyValueObservingOptions)options
context:(nullable void *)context;
// 当被观察的属性发送变化时，会自动触发下方方法
- (void)observeValueForKeyPath:(NSString *)keyPath
ofObject:(id)object
change:(NSDictionary *)change
context:(void *)context{};
```

KVC 和 KVO 的 keyPath 可以是属性、实例变量、成员变量。

```
```

#####26、KVC的底层实现?

当一个对象调用setValue方法时，方法内部会做以下操作：

1). 检查是否存在相应的key的set方法，如果存在，就调用set方法。

2). 如果set方法不存在，就会查找与key相同名称并且带下划线的成员变量，如果有，则直接给成员变量属性赋值。

3). 如果没有找到_key，就会查找相同名称的属性key，如果有就直接赋值。

4). 如果还没有找到，则调用valueForUndefinedKey:和setValue:forUndefinedKey:方法。

这些方法的默认实现都是抛出异常，我们可以根据需要重写它们。

#####27、KVO的底层实现?

KVO基于runtime机制实现。

#####28、ViewController生命周期

按照执行顺序排列：

1. initWithCoder：通过nib文件初始化时触发。

2. awakeFromNib：nib文件被加载的时候，会发生一个awakeFromNib的消息到nib文件中的每个对象。

3. loadView：开始加载视图控制器自带的view。

4. viewDidLoad：视图控制器的view被加载完成。

5. viewWillAppear：视图控制器的view将要显示在window上。

6. updateViewConstraints：视图控制器的view开始更新AutoLayout约束。

7. viewWillLayoutSubviews：视图控制器的view将要更新内容视图的位置。

8. viewDidLayoutSubviews：视图控制器的view已经更新视图的位置。

9. viewDidAppear：视图控制器的view已经展示到window上。

10. viewWillDisappear：视图控制器的view将要从window上消失。

11. viewDidDisappear：视图控制器的view已经从window上消失。

#####29、方法和选择器有何不同?

selector是一个方法的名字，方法是一个组合体，包含了名字和实现。

#####30、你是否接触过OC中的反射机制？简单聊一下概念和使用

1). class反射

通过类名的字符串形式实例化对象。

```
Class class = NSClassFromString(@"student");

Student *stu = [[class alloc] init];
```

将类名变为字符串。

```
Class class =[Student class];

NSString *className = NSStringFromClass(class);
```

2). SEL的反射

通过方法的字符串形式实例化方法。

```
SEL selector = NSSelectorFromString(@"setName");
```

[stu performSelector:selector withObject:@"Mike"];

将方法变成字符串。

NSStringFromSelector(@selector*(setName:));

#####31、调用方法有两种方式：

1). 直接通过方法名来调用。[person show];

2). 间接的通过SEL数据来调用 SEL aaa = @selector(show); [person performSelector:aaa];

#####32、如何对iOS设备进行性能测试?

答： Profile-> Instruments ->Time Profiler

#####33、开发项目时你是怎么检查内存泄露?

1). 静态分析 analyze。

2). instruments工具里面有个leak可以动态分析。

#####34、什么是懒加载?

答：懒加载就是只在用到的时候才去初始化。也可以理解成延时加载。

我觉得最好也最简单的一个例子就是tableView中图片的加载显示了, 一个延时加载, 避免内存过高, 一个异步加载,避免线程堵塞提高用户体验。

#####35、类变量的 @public，@protected，@private，@package 声明各有什么含义?

@public 任何地方都能访问;

@protected 该类和子类中访问,是默认的;

@private 只能在本类中访问;

@package 本包内使用,跨包不可以。

#####36、什么是谓词?

谓词就是通过NSPredicate给定的逻辑条件作为约束条件,完成对数据的筛选。

```
//定义谓词对象,谓词对象中包含了过滤条件(过滤条件比较多)
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"age<%d",30];
//使用谓词条件过滤数组中的元素,过滤之后返回查询的结果
NSArray *array = [persons filteredArrayUsingPredicate:predicate];
```

#####37、isa指针问题

isa：是一个Class 类型的指针. 每个实例对象有个isa的指针,他指向对象的类,而Class里也有个isa的指针, 指向meteClass(元类)。元类保存了类方法的列表。当类方法被调 用时,先会从本身查找类方法的实现,如果没有,元类会向他父类查找该方法。同时注意的是:元类(meteClass)也是类,它也是对象。元类也有isa指针,它的isa指针最终指向的是一个根元类(root meteClass)。根元类的isa指针指向本身,这样形成了一个封闭的内循环。

#####38、如何访问并修改一个类的私有属性?

1). 一种是通过KVC获取。

2). 通过runtime访问并修改私有属性。

#####39、一个objc对象的isa的指针指向什么？有什么作用?

答：指向他的类对象,从而可以找到对象上的方法。

#####40、下面的代码输出什么?

```
@implementation Son : Father
```

```
- (id)init {
if (self = [super init]) {
NSLog(@"%@", NSStringFromClass([self class])); // Son
NSLog(@"%@", NSStringFromClass([super class])); // Son
}
return self;
}
@end
```

// 解析：

self 是类的隐藏参数，指向当前调用方法的这个类的实例。

super是一个Magic Keyword，它本质是一个编译器标示符，和self是指向的同一个消息接收者。

不同的是：super会告诉编译器，调用class这个方法时，要去父类的方法，而不是本类里的。

上面的例子不管调用[self class]还是[super class]，接受消息的对象都是当前 Son *obj 这个对象。

#####41、写一个完整的代理，包括声明、实现

```
// 创建
@protocol MyDelagate
@required
-(void)eat:(NSString *)foodName;
@optional
-(void)run;
@end

// 声明 .h
@interface person: NSObject<MyDelagate>
```

```
@end

// 实现 .m
@implementation person
- (void)eat:(NSString *)foodName {
NSLog(@"吃:%@!", foodName);
}
- (void)run {
NSLog(@"run!");
}

@end
```

#####42、isKindOfClass、isMemberOfClass、selector作用分别是什么

isKindOfClass：作用是某个对象属于某个类型或者继承自某类型。

isMemberOfClass：某个对象确切属于某个类型。

selector：通过方法名，获取在内存中的函数的入口地址。

#####43、delegate 和 notification 的区别

1). 二者都用于传递消息，不同之处主要在于一个是一对一的，另一个是一对多的。

2). notification通过维护一个array，实现一对多消息的转发。

3). delegate需要两者之间必须建立联系，不然没法调用代理的方法；notification不需要两者之间有联系。

#####44、什么是block?

闭包（block）：闭包就是获取其它函数局部变量的匿名函数。

#####45、block反向传值

在控制器间传值可以使用代理或者block，使用block相对来说简洁。

在前一个控制器的touchesBegan:方法内实现如下代码。

```
// OneViewController.m
TwoViewController *twoVC = [[TwoViewController alloc] init];
twoVC.valueBlcok = ^(NSString *str) {
NSLog(@"OneViewController拿到值：%@", str);
};
[self presentViewController:twoVC animated:YES completion:nil];

// TwoViewController.h （在.h文件中声明一个block属性）
@property (nonatomic ,strong) void(^valueBlcok)(NSString *str);

// TwoViewController.m （在.m文件中实现方法）
- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
// 传值:调用block
if (_valueBlcok) {
_valueBlcok(@"123456");
}
}
```

#####46、block的注意点

1). 在block内部使用外部指针且会造成循环引用情况下，需要用__week修饰外部指针：

__weak typeof(self) weakSelf = self;

2). 在block内部如果调用了延时函数还使用弱指针会取不到该指针，因为已经被销毁了，需要在block内部再将弱指针重新强引用一下。

__strong typeof(self) strongSelf = weakSelf;

3). 如果需要在block内部改变外部栈区变量的话，需要在用__block修饰外部变量。

#####47、BAD_ACCESS在什么情况下出现?

答：这种问题在开发时经常遇到。原因是访问了野指针，比如访问已经释放对象的成员变量或者发消息、死循环等。

#####48、lldb（gdb）常用的控制台调试命令?

1). p 输出基本类型。是打印命令，需要指定类型。是print的简写

p (int)[[[self view] subviews] count]

2). po 打印对象，会调用对象description方法。是print-object的简写

po [self view]

3). expr 可以在调试时动态执行指定表达式，并将结果打印出来。常用于在调试过程中修改变量的值。

4). bt：打印调用堆栈，是thread backtrace的简写，加all可打印所有thread的堆栈

5). br l：是breakpoint list的简写

#####49、你一般是怎么用Instruments的?

Instruments里面工具很多，常用：

1). Time Profiler: 性能分析

2). Zombies：检查是否访问了僵尸对象，但是这个工具只能从上往下检查，不智能。

3). Allocations：用来检查内存，写算法的那批人也用这个来检查。

4). Leaks：检查内存，看是否有内存泄露。

#####50、iOS中常用的数据存储方式有哪些?

数据存储有四种方案：NSUserDefault、KeyChain、file、DB。

其中File有三种方式：plist、Archive（归档）

DB包括：SQLite、FMDB、CoreData

#####51、iOS的沙盒目录结构是怎样的?

沙盒结构:

1). Application：存放程序源文件，上架前经过数字签名，上架后不可修改。

2). Documents：常用目录，iCloud备份目录，存放数据。（这里不能存缓存文件，否则上架不被通过）

3). Library:

Caches：存放体积大又不需要备份的数据。(常用的缓存路径)

Preference：设置目录，iCloud会备份设置信息。

4). tmp：存放临时文件，不会被备份，而且这个文件下的数据有可能随时被清除的可能。

#####52、iOS多线程技术有哪几种方式?

答：pthread、NSThread、GCD、NSOperation

#####53、GCD 与 NSOperation 的区别:

GCD 和 NSOperation 都是用于实现多线程:

GCD 基于C语言的底层API，GCD主要与block结合使用，代码简洁高效。

NSOperation 属于Objective-C类，是基于GCD更高一层的封装。复杂任务一般用NSOperation实现。

#####54、写出使用GCD方式从子线程回到主线程的方法代码

答：dispatch_sync(dispatch_get_main_queue(), ^{ });

```

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）

// 使用Dispatch Group追加block到Global Group Queue,这些block如果全部执行完毕，就会执行Main Dispatch Queue中的结束处理的block。

// 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);

dispatch_group_async(group, queue, ^{ /*加载图片1 */ });

dispatch_group_async(group, queue, ^{ /*加载图片2 */ });

dispatch_group_async(group, queue, ^{ /*加载图片3 */ });

// 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, dispatch_get_main_queue(), ^{

// 合并图片

});

```

#####55、dispatch_barrier_async（栅栏函数）的作用是什么?

函数定义：dispatch_barrier_async(dispatch_queue_t queue, dispatch_block_t block);

作用：

1.在它前面的任务执行结束后它才执行，它后面的任务要等它执行完成后才会开始执行。

2.避免数据竞争

// 1.创建并发队列

```
```

```
dispatch_queue_t queue = dispatch_queue_create("myQueue", DISPATCH_QUEUE_CONCURRENT);

// 2.向队列中添加任务

dispatch_async(queue, ^{ // 1.2是并行的

NSLog(@"任务1, %@",[NSThread currentThread]);

});

dispatch_async(queue, ^{

NSLog(@"任务2, %@",[NSThread currentThread]);

});


dispatch_barrier_async(queue, ^{

NSLog(@"任务 barrier, %@", [NSThread currentThread]);

});


dispatch_async(queue, ^{ // 这两个是同时执行的

NSLog(@"任务3, %@",[NSThread currentThread]);

});

dispatch_async(queue, ^{

NSLog(@"任务4, %@",[NSThread currentThread]);

});
```

```
// 输出结果: 任务1 任务2 ——》 任务 barrier ——》任务3 任务4

// 其中的任务1与任务2，任务3与任务4 由于是并行处理先后顺序不定。


#####56、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");
}
// 只输出：1。（主线程死锁）
```

#####57、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，基本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。

主线程的run loop默认是启动的。iOS的应用程序里面，程序启动后会有一个如下的main()函数

```
int main(int argc, char * argv[]) {
@autoreleasepool {
return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
}
}
```

#####58、什么是 Runtime

Runtime又叫运行时，是一套底层的C语言API，其为iOS内部的核心之一，我们平时编写的OC代码，底层都是基于它来实现的。

#####59、Runtime实现的机制是什么，怎么用，一般用于干嘛?

1). 使用时需要导入的头文件 <objc/message.h> <objc/runtime.h>

2). Runtime 运行时机制，它是一套C语言库。

3). 实际上我们编写的所有OC代码，最终都是转成了runtime库的东西。

比如：

类转成了 Runtime 库里面的结构体等数据类型，

方法转成了 Runtime 库里面的C语言函数，

平时调方法都是转成了 objc_msgSend 函数（所以说OC有个消息发送机制）

// OC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)。

// [stu show]; 在objc动态编译时，会被转意为：objc_msgSend(stu, @selector(show));

4). 因此，可以说 Runtime 是OC的底层实现，是OC的幕后执行者。

#####60、有了Runtime库，能做什么事情呢?

Runtime库里面包含了跟类、成员变量、方法相关的API。

比如：

（1）获取类里面的所有成员变量。

（2）为类动态添加成员变量。

（3）动态改变类的方法实现。

（4）为类动态添加新的方法等。

因此，有了Runtime，想怎么改就怎么改。

#####61、什么是 Method Swizzle（黑魔法），什么情况下会使用?

1). 在没有一个类的实现源码的情况下，想改变其中一个方法的实现，除了继承它重写、和借助类别重名方法暴力抢先之外，还有更加灵活的方法 Method Swizzle。

2). Method Swizzle 指的是改变一个已存在的选择器对应的实现的过程。OC中方法的调用能够在运行时通过改变，通过改变类的调度表中选择器到最终函数间的映射关系。

3). 在OC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OC的动态特性，可以实现在运行时偷换selector对应的方法实现。

4). 每个类都有一个方法列表，存放着selector的名字和方法实现的映射关系。IMP有点类似函数指针，指向具体的方法实现。

5). 我们可以利用 method_exchangeImplementations 来交换2个方法中的IMP。

6). 我们可以利用 class_replaceMethod 来修改类。

7). 我们可以利用 method_setImplementation 来直接设置某个方法的IMP。

8). 归根结底，都是偷换了selector的IMP。

#####62、_objc_msgForward 函数是做什么的，直接调用它将会发生什么?

答：_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。

#####63、什么是 TCP / UDP ?

TCP：传输控制协议。

UDP：用户数据协议。

TCP 是面向连接的，建立连接需要经历三次握手，是可靠的传输层协议。

UDP 是面向无连接的，数据传输是不可靠的，它只管发，不管收不收得到。

简单的说，TCP注重数据安全，而UDP数据传输快点，但安全性一般。

#####64、通信底层原理（OSI七层模型）

OSI采用了分层的结构化技术，共分七层：

物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

#####65、介绍一下XMPP?

XMPP是一种以XML为基础的开放式实时通信协议。

简单的说，XMPP就是一种协议，一种规定。就是说，在网络上传东西，XMM就是规定你上传大小的格式。

#####66、OC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？

```
// 创建线程的方法
- [NSThread detachNewThreadSelector:nil toTarget:nil withObject:nil]
- [self performSelectorInBackground:nil withObject:nil];
- [[NSThread alloc] initWithTarget:nil selector:nil object:nil];
- dispatch_async(dispatch_get_global_queue(0, 0), ^{});
- [[NSOperationQueue new] addOperation:nil];

// 主线程中执行代码的方法
- [self performSelectorOnMainThread:nil withObject:nil waitUntilDone:YES];
- dispatch_async(dispatch_get_main_queue(), ^{});
- [[NSOperationQueue mainQueue] addOperation:nil];
```

#####67、tableView的重用机制?

答：UITableView 通过重用单元格来达到节省内存的目的: 通过为每个单元格指定一个重用标识符，即指定了单元格的种类,当屏幕上的单元格滑出屏幕时，系统会把这个单元格添加到重用队列中，等待被重用，当有新单元格从屏幕外滑入屏幕内时，从重用队列中找看有没有可以重用的单元格，如果有，就拿过来用，如果没有就创建一个来使用。

#####68、用伪代码写一个线程安全的单例模式

```
static id _instance;
+ (id)allocWithZone:(struct _NSZone *)zone {
static dispatch_once_t onceToken;
dispatch_once(&onceToken, ^{
_instance = [super allocWithZone:zone];
});
```

```
return _instance;
}

+ (instancetype)sharedData {
static dispatch_once_t onceToken;
dispatch_once(&onceToken, ^{
_instance = [[self alloc] init];
});
return _instance;
}

- (id)copyWithZone:(NSZone *)zone {
return _instance;
}
```

#####69、如何实现视图的变形?

答：通过修改view的 transform 属性即可。

#####70、在手势对象基础类UIGestureRecognizer的常用子类手势类型中哪两个手势发生后，响应只会执行一次?

答：UITapGestureRecognizer,UISwipeGestureRecognizer是一次性手势,手势发生后,响应只会执行一次。

#####71、字符串常用方法：

```
NSString *str = @"abc*123";
NSArray *arr = [str componentsSeparatedByString:@"*"]; //以目标字符串把原字符串分割成两部分，存到数组中。@[@"abc", @"123"];
```

#####72、如何高性能的给 UIImageView 加个圆角?

不好的解决方案：使用下面的方式会强制Core Animation提前渲染屏幕的离屏绘制, 而离屏绘制就会给性能带来负面影响，会有卡顿的现象出现。

```
self.view.layer.cornerRadius = 5.0f;
self.view.layer.masksToBounds = YES;
```

正确的解决方案：使用绘图技术

```
- (UIImage *)circleImage {
// NO代表透明
UIGraphicsBeginImageContextWithOptions(self.size, NO, 0.0);
// 获得上下文
CGContextRef ctx = UIGraphicsGetCurrentContext();
// 添加一个圆
CGRect rect = CGRectMake(0, 0, self.size.width, self.size.height);
CGContextAddEllipseInRect(ctx, rect);
// 裁剪
CGContextClip(ctx);
// 将图片画上去
[self drawInRect:rect];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
// 关闭上下文
UIGraphicsEndImageContext();
return image;
}
```

还有一种方案：使用了贝塞尔曲线"切割"个这个图片, 给UIImageView 添加了的圆角，其实也是通过绘图技术来实现的。

```

UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 100, 100)];

imageView.center = CGPointMake(200, 300);

UIImage *anotherImage = [UIImage imageNamed:@"image"];

UIGraphicsBeginImageContextWithOptions(imageView.bounds.size, NO, 1.0);

[[UIBezierPath bezierPathWithRoundedRect:imageView.bounds

cornerRadius:50] addClip];

[anotherImage drawInRect:imageView.bounds];

imageView.image = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

[self.view addSubview:imageView];
```

#####73、你是怎么封装一个view的

1）. 可以通过纯代码或者xib的方式来封装子控件

2）. 建立一个跟view相关的模型，然后将模型数据传给view，通过模型上的数据给view的子控件赋值

```

/**

* 纯代码初始化控件时一定会走这个方法

*/

- (instancetype)initWithFrame:(CGRect)frame {

if(self = [super initWithFrame:frame]) {

[self setupUI];

}

return self;

}
```

```
/**
* 通过xib初始化控件时一定会走这个方法
*/
- (id)initWithCoder:(NSCoder *)aDecoder {
if(self = [super initWithCoder:aDecoder]) {
[self setupUI];
}
return self;
}

- (void)setupUI {
// 初始化代码
}
```

#####74、HTTP协议中 POST 方法和 GET 方法有那些区别?

1. GET用于向服务器请求数据，POST用于提交数据

2. GET请求，请求参数拼接形式暴露在地址栏，而POST请求参数则放在请求体里面，因此GET请求不适合用于验证密码等操作

3. GET请求的URL有长度限制，POST请求不会有长度限制

#####75、请简单的介绍下APNS发送系统消息的机制?

APNS优势：杜绝了类似安卓那种为了接受通知不停在后台唤醒程序保持长连接的行为，由iOS系统和APNS进行长连接替代。

APNS的原理：

1). 应用在通知中心注册，由iOS系统向APNS请求返回设备令牌(device Token)

2). 应用程序接收到设备令牌并发送给自己的后台服务器

3). 服务器把要推送的内容和设备发送给APNS

4). APNS根据设备令牌找到设备，再由iOS根据APPID把推送内容展示