1、设计模式是什么？ 你知道哪些设计模式，并简要叙述？

设计模式是一套被 反复使用、多数人知晓、经过分类编目的、代码设计经验的总结。

单例模式：单例模式确保某一个类只有一个实例，并提供一个访问它的全剧访问点。具体的详情可点击进入查看

工厂模式：工厂父类负责定义创建产品对象的公共接口，而工厂子类则负责生产具体的产品对象，即通过不停的工厂子类来创建不同的产品对象。具体的详情可点击进入查看

代理模式 :为某个对象提供一个代理，并由这个代理对象控制对原对象的访问。具体的详情可点击进入查看

适配器模式： 将一个接口转换成客户希望的另一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。适配器模式的别名是包装器模式（Wrapper），是一种结构型设计模式。具体的详情可点击进入查看

装饰者模式： 不改变原有对象的前提下，动态地给一个对象增加一些额外的功能。具体的详情可点击进入查看

2、MVC 和 MVVM 的区别？

MVC

MVC（Model-View-Controller）模式结构图，可分为三部分：**模型**（Model）、**视图**（View）、**控制器**（Controller）。其在 MVC 模式中所扮演的角色分别为：

Model：模型管理应用程序的数据，响应有关其状态信息（通常来自 View）的请求，并响应指令以更改状态（通常来自 Controller）。

View：视图管理数据的展示。

Controller：控制器解释用户的输入，并通知模型、视图进行状态更新

所有通信都是单向的。
优点：对 Controller 进行瘦身，将 View 内部的细节封装起来了，外界不知道 View 内部的具体实现
缺点：View 和 Controller 依赖于 Model

MVVM

MVVM（Model View View-Model）就是为了解决过于臃肿的问题。MVVM 的思想是将 Controller 中 UI 控制逻辑与业务逻辑进行分离，并抽离出一个 View-Model 来完成 UI 控制的逻辑。而 Controller 只需要负责业务逻辑即可

唯一的区别是，View-Model 可以调用 Model 定义的方法，从 Model 中获取数据以用于 View，并对数据进行预处理，使 View 可以直接使用。View 又可以向 View-Model 发出用户的操作命令，从而更改 Model。MVVM 实现了一种双向绑定机制。

优点：降低了 View 和 Model 之间的耦合；分离了业务逻辑和视图逻辑。
缺点：View 和 Model 双向绑定导致 bug 难以定位，两者中的任何一方出现问题，另一方也会出现问题；增加了胶水代码。

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

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：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：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？没有的话用什么代替？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？

OC 不可以多继承，OC 是单继承。有时可以用分类和协议来代替多继承
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:@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以及作用？

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，一般使用 nonatomic，效率高。
atomic：多线程安全，但是性能低
strong：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。
copy ：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。
assign：用于基本数据类型
readwrite：可读可写特性。需要生成 getter 方法和 setter 方法
readonly：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。
retain：相当于 ARC 中的 strong

8、delegate 和 notification 的区别

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
notification：不需要两者之间有联系,实现一对多消息的转发
delegate：需要两者之间必须建立联系，不然没法调用代理的方法

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（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、self.跟 self->什么区别？

1>. self.是调用 get 方法或者 set 放
2>. self 是当前本身，是一个指向当前对象的指针
3>. self->是直接访问成员变量

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？

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？

浅拷贝：对一个对象地址的拷贝。源对象和副本对象是同一对象
深拷贝：对一个对象的拷贝。源对象和副本对象是不同的两个对象

* * *

13、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。

```
-[__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： 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。

14、一个 objc 对象的 isa 的指针指向什么？有什么作用？

指向他的类对象,从而可以找到对象上的方法

15、Objective-C 如何对内存管理的，说说你的看法和解决方法？

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。

16、iOS UIViewController 的完整生命周期?

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。

17、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,因为 viewDidLoad 方法默认开了一条主线程，然后又执行 dispatch_sync(dispatch_get_main_queue(), ^{...});会导致你等我我等你，结果导致死锁。

18、Object-C 有私有方法吗？私有变量呢？

1>.OC 没有类似@private 的修饰词来修饰方法，只要写在.h 文件中，就是公共方法
2>. 如果你不在.h 文件中声明，只在.m 文件中实现，或在.m 文件的 Class Extension 里声明，那么基本上和私有方法差不多，可以使用类扩展（Extension）来增加私有方法和私有变量
3>. 使用 private 修饰的全局变量是私有变量

19、关键字 const 什么含义？

```
const int a;

int const a;

const int *a;

int const *a;

int * const a;

int const * const a;
```

1>. 前两个的作用是一样：a 是一个常整型数
2>. 第三、四个意味着 a 是一个指向常整型数的指针(整型数是不可修改的，但指针可以)
3>. 第五个的意思：a 是一个指向整型数的常指针(指针指向的整型数是可以修改的，但指针是不可修改的)
4>. 最后一个意味着：a 是一个指向常整型数的常指针(指针指向的整型数是不可修改的，同时指针也是不可修改的)

20、用伪代码写一个线程安全的单例模式

```
static XXManager * instance = nil;

+ (instancetype)shareInstance {

      static dispatch_once_t onceToken;

      dispatch_once(&onceToken, ^{

        instance = [[self alloc] init];

});

return instance;}

+ (id)allocWithZone:(struct _NSZone *)zone {



      static dispatch_once_t onceToken;

      dispatch_once(&onceToken, ^{

        instance = [super allocWithZone:zone];

});

return instance;

}

- (id)copyWithZone:(NSZone *)zone {return instance;

}
```

21、category(类别) 和 extension(扩展) 的区别

1>. 类别有名字，类扩展没有分类名字，是一种特殊的分类。
2>. 类别只能扩展方法（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、tableView 的重用机制？

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

23、iOS 内存的使用和优化的注意事项?

重用问题：如 UITableViewCells、UICollectionViewCells、UITableViewHeaderFooterViews。设置正确的 reuseIdentifier，充分重用
1>不要使用太复杂的 XIB/Storyboard：载入时就会将 XIB/storyboard 需要的所有资源，包括图片全部载入内存。
尽量把 views 设置为不透明：当 opque 为 NO 的时候，图层的半透明取决于图片和其本身合成的图层为结果，可提高性能
选择正确的数据结构：学会选择对业务场景最合适的数组结构是写出高效代码的基础。
gzip/zip 压缩：当从服务端下载相关附件时，可以通过 gzip/zip 压缩后再下载，使得内存更小，下载速度也更快。
延迟加载：对于不应该使用的数据，使用延迟加载方式。对于不需要马上显示的视图，使用延迟加载方式。比如，网络请求失败时显示的提示界面，可能一直都不会使用到，因此应该使用延迟加载。
数据缓存：对于 cell 的行高要缓存起来，使得 reload 数据时，效率也极高。
而对于那些网络数据，不需要每次都请求的，应该缓存起来。可以写入数据库，也可以通过 plist 文件存储
处理内存警告：一般在基类统一处理内存警告，将相关不用资源立即释放掉

24、iOS 你在项目中是怎么优化内存的？

这个问题有时候笔试中也有，有时候有些面试官会在面试中问你这个问题

1>.避免庞大的 Xib(Xib 比 frame 消耗更多的 CPU 资源)
2>.不要阻塞主线程，尽量把耗时的操作放到子线程
3>.重用和延迟加载
4>.尽量减少视图数量和层次
5>.优化 TableView,为了使 TableVIew 有更好的滚动性能可采取以下措施：

* 正确使用 ruseIdentifier 来重用 cells
* 采用懒加载即延迟加载的方式加载 cell 上的控件
* 当 TableView 滑动的时候不加载
* 缓存 cell 的高度。在呈现 cell 前，把 cell 的高度计算好缓存起来，避免每次加载 cell 的时候都要计算
* 尽量使用不透明的 UI 控件

25、写一个完整的代理，包括声明、实现

```
// 创建

@protocol PersonDelagate

@required

-(void)eat:(NSString *)foodName;

@optional

-(void)run;

@end



// 声明 .h

@interface Person: NSObject<PersonDelagate>

@end

// 实现 .m

@implementation Person

- (void)eat:(NSString *)foodName {NSLog(@"吃:%@", foodName);

}

- (void)run {

   NSLog(@"run");

}

@end
```

26、iOS 你在项目中用过 GCD 吗？举个例子

用过。比如 网络请求数据成功之后刷新列表

```
  [HTTPRequest POST:kbazaarUrl parameter:nil success:^(id resposeObject)
{

      /**网络请求成功之后处理数据*/

       if (Success) {

             //处理数据 ...

          }

          dispatch_async(dispatch_get_main_queue(), ^{

              [weakSelf.tableView reloadData];

          });

       }else{

          [weakSelf failEndRefreshStatus:0];

          [MBProgressHUD LY_ShowError:resposeObject[@"msg"] time:2.0];

       }

   } failure:^(NSError *error) {

       [MBProgressHUD LY_ShowError:kNoNetworkTips time:2.0];

   }];
```

27、GCD 与 NSOperation 的区别

NSOperation:相对于 GCD 来说，更加强大。可以给 operation 之间添加依赖关系、取消一个正在执行的 operation、暂停和恢复 operationQueue 等

GCD: 是一种更轻量级的，以 FIFO(先进先出，后进后出)的顺序执行并发任务。使用 GCD 我们并不用关心任务的调度情况，而是系统会自动帮我们处理。但是 GCD 的短板也是非常明

显的，比如我们想要给任务之间添加依赖关系、取消或者暂停一个正在执行的任务时就会变得束手无策。

28、写出使用 GCD 方式从子线程回到主线程的方法代码

```
dispatch_sync(dispatch_get_main_queue(), ^{ });
```

29、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:YE
S];

- dispatch_async(dispatch_get_main_queue(), ^{});

- [[NSOperationQueue mainQueue] addOperation:nil];
```

30、你是怎么封装一个 view 的

笔者就曾遇到过这个问题。
当时这样，原题目是：**怎么用纯代码或者 xib 实现一个 View 的组件化**。

看到这个题目的时候我就懵逼了，就好像考试的过程中碰到不会的题目一样，这里直接是 0 分。后来面试的过程中，我就用面试官此题怎解。霹雳啪啦给我说了一大堆，最后我问是不是封装一个 View?他回答说:是的！封装任意一个 View.之后就问我实现思路要实现什么方法。然后我就说实现`initWIthFrame:`然后创建需要的控件，最后直接通过添加个类方法 show 出来即可。因为我也不知道这个任意的 View 到底是个什么 View，就将 TA 假象成一个弹框好了。结果面试官继续追问过好几次还要实现什么方法呢？？？ 我沉思了好久好一会（明明就差不多是这样子左右，为啥他还继续追问呢）。。。面试官可能感受到了空中传来一阵尴尬气氛之后，说不知道没关系。。。之后又问了一个类似 QQ 消息点击 tabBar 上面的 item 然后列表的小红点有类似西红柿爆炸的效果。经过上一个`骚问题`,我已经知道面试已经凉凉了(而且听说接手的项目是之前废弃半年的棋牌项目，现在重新启动) 我直接说不知道。对于此面试官来说，我可能是个`菜鸡`吧。我觉得身心受到了严重的打击。可能还是不够努力吧！

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

```
- (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}
```