一、atomic 和nonatomic的区别?

1、atomic 使用同步锁，原子性:

在该属性在调用getter和setter方法时，会加上自旋锁（osspinlock）

即在属性在调用getter和setter方法时，保证同一时刻只能有一个线程调用读写方法。

2、nonatomic：不使用同步锁，非原子性:

在该属性在调用setter和setter方法时，不会加上自旋锁，也就是线程并不安全。

二、id和instanceType有什么区别?

1、相同点:

instancetype和id都是万能指针，指向对象。

2、不同点:

①id在编译的时候不能判断对象的真实类型，instancetype在编译的时候可以判断对象的真实类型。

②id可以用来定义变量，可以作为返回值类型，可以作为形参类型，instancetype只能作为返回值类型。

三、self和super的区别

1、self调用自己的方法，super调用父类方法

2、self是类，super是预编译指令

3、[self class]和[super class]输出是一样的

4、self和super的底层实现原理

当使用self调用方法时，先从当前类的方法列表中开始查找，如果没有，就从父类中再找。

当使用super时，则从父类的方法列表中开始找，然后调用父类的方法

当使用self调用时，会使用objc_msgSend函数，id objc_msgSend (id theReceiver, SEL theSelector,...)第一个参数是消息接受者，第二个参数是调用具体类方法的selector,后面是selector方法的可变参数，以[self setName:]为例，编译器会替换成调用objc_msgSend的函数调用，其中theReceiver是self,theSelector是@selector(setName:)，这个selector是从当前self的class的方法列表开始找的setName，当找到后把对应的selector传递过去。

当使用super调用时，会使用objc_msgSendSuper函数， id objc_msgSendSuper(struct objc_super *super, SEL op...)，第一个参数是objc_supe的结构体，第二个参数还是类似上面的类方法的selector

```
struct objc_super{
    id receiver;
    Class superClass;
}
```

四、UIResponder的理解

UIResponder类是专门用来响应用户的操作处理各种事件的，包括触摸事件、运动事件、远程控制事件，UIApplication、UIView、UIViewController这几个类是直接继承自UIResponder,所以这些类都可以响应事件。

五、keywindow 和delegate的window有何区别

1、delegate.window程序启动时设置的window对象

2、keywindow这个属性保存了[windows]数组中的[UIWindow]对象，该对象最近被发送了[makeKeyAndVisible]消息

一般情况下delegate.window和keyWindow是同一个对象，但不能保证keyWindow就是delegate.window,因为keyWindow会因为makeKeyAndVisible而变化，例如程序中添加了一个悬浮窗口，这个时候keyWindow就会变化。

六、@autoreleasePool的数据结构?

简单说是双向链表，每张链表头尾相接，有parent、child指针，每创建一个池子，会在首部创建一个哨兵对象作为标记，最外层池子的顶端会有一个next指针，当链表容量满了，就回在链表的顶端并指向下一张表

七、循环引用

循环引用的实质：多个对象相互之间有强引用，不能释放让系统回收

如何解决循环因引用?

1、避免产生循环引用，通常是将strong引用改为weak引用，比如在修饰属性时用weak在block内调用对象方时，使用其弱引用

八、KVO-键值观察原理(kvo)

1、当给一个A类添加KVO的时候，runtime会动态生成了一个子类NSKVONotifing_A,让A类的isa指针指

向NSKVONotifing_A类，重写class方法，隐藏对象真实类信息。

2、重写监听属性的setter方法，在setter方法内部调用了Foundation的_NSSetObjectValueAndNotify函数

3、_NSSetObjectValueAndNotify函数内部

首先会调用willChangeValueForKey

然后给属性赋值

最后调用didChangeValueForKey

最后调用observe的observeValueForKeyPath去告诉监听器属性值发生了改变

4、重写dealloc做一些KVO内存释放

九、为什么Block用copy关键字

Block在没有使用外部变量时，内存存在全局区，当Block在使用外部变量的时候，内存是存在于栈区，当Block copy之后，是存在堆区的，存在于栈区的特点是对象随时有可能被销毁，一旦销毁在调用的时候，就会造成系统崩溃.所以block要用copy关键字。

十、网络七层协议

应用层、表示层、会话层、传输层、网络层、数据链路层、物理层

十一、HTTPS连接建立流程

HTTPS为了兼顾安全与效率，同时使用了对称加密和非对称加密，在传输的过程中会涉及到三个密钥:

服务端的公钥和私钥，用来进行非对称加密
客户端生成的随机密钥，用来进行对称加密

1、客户端访问HTTPS连接。

客户端会会把安全协议版本号、客户端支持的加密算法列表，随机数C发给服务端。

2、服务端发送证书给客户端

服务端接收密钥算法配件后，会和自己支持的加密算法列表进行比对，如果不符合，则断开连接，否则服务端会在该算法列表中，选择一种对称算法、一种公钥算法（具有特定密钥长度的RSA）和一种MAC算法发给客户端。在发送算法的同时还会把数字证书和随机数发送给客户端

3、客户端验证server证书

会对server公钥进行检查，验证其合法性，如果发现公钥有问题，那么HTTPS传输就无法继续。

4、客户端组装会话密钥

如果公钥合格，那么客户端会用服务器公钥来生成一个前主密钥(PMS),并通过该前主密钥和随机数C、S来组装会话密钥

5、客户端将前主秘钥加密发送给服务端。

是通过服务端的公钥来对前主密钥进行对称加密，发送给服务端。

6、服务端通过私钥解密得到主密钥

7、服务端组装会话密钥

8、数据传输

十二、TCP和UDP的区别

TCP:面向连接、传输可靠（保证数据正确性，保证数据顺序）用于传输大量数据（流模式）、速度慢，建立连接需要开销较多（时间、系统资源）

UDP:面向非连接、传输不可靠，用于传输少量数据（数据包模式）、速度快

十三、iOS开发中数据持久性有哪几种?

iOS本地数据保存有多种方式，比如NSUserDefaults、归档、文件保存、数据库、CoreData、KeyChain（钥匙串）等多种方式、其中KeyChain是保存在沙盒范围以外的地方，也就是与沙盒无关。

十四、进程和线程

进程:

1、进程是一个具有一定独立功能的程序关于某次数据集合的一次运行活动，它是操作系统分配资源的基本单元。

2、进程是指在系统中正在运行的一个应用程序，就是一段程序执行过程，我们可以理解为手机上的app. 3、每个进程之间是独立，每个进程均运行在其专用且受保护的内存空间内，拥有独立运行所需的全部资源。

线程:

1、程序执行流的最小单元，线程是进程中的一个实体。

2、一个进程想要执行任务，必须至少有一条线程，应用程序启动的时候，系统会默认开启一条线程也就是主线程。

进程和线程的关系

1、线程是进程的执行单元，进程所有的任务都在线程中执行

2、一个程序可以对应多个进程（多进程），一个进程中可有多个线程，但至少要有一条线程。

3、同一个进程内的线程共享进程资源。

什么是多线程?

多线程的实现原理：事实上，同一时间内单核CPU只能执行一个线程，多线程是CUP快速的在多个线程之间切换，造成了多个线程同时执行的假象。

如果是多核CPU就真的可以同时处理多个线程了。

多线程的目的是为了同步完成多项任务，通过提高系统的资源利用率来提高系统的效率。

多线程的优点和缺点?

优点:

能适当提高程序的执行效率

能适当提高资源利用率（CPU、内存利用率）

缺点:

1、开启线程会占用一定的内存空间（默认情况下，主线程占用1M,子线程512KB）,如果开启大量的线程，会占用大量的内存空间，降低程序的性能。

2、线程越多，CPU在调度线程上的开销就越大

多线程的并行和并发有什么区别?

并行：充分利用计算机的多核，在多个线程上同步进行

并发：在一条线程上通过快速切换，让人感觉在同步执行

多线程中的死锁

死锁是由于多个线程（进程）在执行过程中，因为争夺资源而造成的互相等待的现象。

GCD的执行原理

GCD底层有一个线程池，这个池中存放的是一个个线程，”池“中的线程可以重用，当一段时间后这个线程没有被调用，这个线程就会被销毁，注意，开多少个线程是由线程池决定的，池是系统自动维护的，不需要程序员维护，程序员只需要向队列中添加任务，队列调度即可。

如果队列中存放的是同步任务，则任务出队后，底层线程池会提供一条线程供这个任务执行，任务执行完毕后这条线程再回到线程池，这样队列中的任务反复调度，因为是同步的，所以我们用currentThread打印的时候，是同一个线程。

如果队列中的任务存放的是异步任务，当任务出队后，底层线程池会提供一个线程供任务执行，因为是异步执行，队列中的任务不需要等待当前任务执行完毕就可以调度下一个任务，这时底层线程池中会再次提供一个线程供第二个任务执行，执行完毕后再回到底层线程池中。

这样就对一个线程完成一个复用，而不需要每一个任务执行都开启新线程，也就节约了系统的开销，提高了效率。iOS7的时候，使用GCD系统通常只能开5-8条线程，iOS8以后，系统可以开多个线程，但是在实际开发中，建议开启的线程数为3-5条。

十五、ARC下如何解决block循环引用问题?

三种方式：__weak、__unsafe_unretained、__block

__weak: 不会产生强引用，指向对象销毁时，会自动让指针置为nil

__unsafe_unreatined:不会产生强引用，不安全，指向的对象销毁时，指针储存的地址不变。

__block必须把引用对象置为nil,并且要调用该block

十六、 对称加密和非对称加密的区别

对称加密又称为公开密钥加密，加密和解密都会用到同一个密钥，如果密钥被攻击者攻破，此时的加密就失去了意义。常见的对称加密算法：DES、 3DES、AES

非对称加密称为共享密钥加密，使用一对非对称的密钥，一个是私有密钥，一个是公有密钥，公钥加密只能用私钥解密，私钥加密只能用公钥来解密，常见的公钥加密算法有：RSA、背包算法、椭圆曲线加密算法。

十七、什么是isa指针

isa指针，等价于is kind of实例对象的isa指向类对象，类对象的isa指向元类对象，元类对象的isa指针指向根元类对象。isa有两种类型，纯指针，指向内存地址，NON_POINTER_ISA,除了内存地址，还存有一些其他信息。

十八、runtime的具体应用

1、利用关联对象（AssociatedObject）给分类添加属性

2、遍历类的所有成员变量

3、交换方法实现

4、利用消息转发机制解决方法找不到的问题。

5、KVC字典模型转换

十九、什么是多线程

1、同一时间，cpu只能处理一条线程，只有一条线程在执行，多线程并发执行，其实是cpu快速的在多条线程之间调度切换，如果cpu调度线程的时间足够快，就造成了多线程并发执行的假象。2、如果线程非常非常多，cpu会在N多线程之间调度，消耗大量的cup资源，每条线程被调度执行的频次降低。3、多线程的优点：能适当提高程序的执行效率，能适当提高资源的利用率（cpu/内存利用率）4、多线程的缺点开启线程需要占用一定的内存空间，默认情况下，主线程占用1M，子线程占用512KB,如果开启大量的线程，会占用大量的内存空间，降低程序的性能。线程越多，CPU在调度线程上的开销就越大。程序设计更加复杂：比如线程之间的通信、多线程的数据共享。

任务

执行的操作，也就是在线程中执行的那段代码，在GCD中是放在block中的，执行任务有两种方式：同步执行sync和异步执行async

二十、SDWebImage实现原理

1、根据给定的url生成一个唯一的key,之后利用这个key到缓存中查找对应的图片缓存。

2、读取图片缓存，根据key先从内存中读取图片缓存，若没有命中内存缓存则读取磁盘缓存，如果磁盘缓存命中，那么将磁盘缓存读到内存中成为内存缓存，如果都没有命中缓存的话，那么就在执行的doneBlock中开始下载图片。

3、图片下载操作完成后将图片对应的数据通过completed Block进行回调。（在图片下载方法中，调用了一个方法用于添加创建和下载过程中的各类Block回调downloadImageWithURL: 如果URL是第一次加载的话，那么就会执行createCallback这个回调Block,然后在createCallback里面开始构建网络请求，在下载过程中执行各类进度Block回调，当图片下载完成之后会回到done的Block回调中做图片转换处理和缓存操作）

二一、setNeedsDisplay 和 layoutifNeeded两者是什么关系?

UIView的setNeedsDisplay和setNeedsLayout两个方法都是异步执行的，而setNeedsDisplay会自动调用drawRect方法，这样可以拿到UIGraphicsGetCurrentContext进行绘制，而setNeedsLayout会默认调用layoutSubviews,给当前的视图做了标记，layoutifNeeded查找是否有标记，如果有标记会立刻刷新。只有setNeedsLayout和layoutIfNeeded这二者结合起来使用，才会起到立刻

刷新的效果。

二二、NSArray与NSSet的区别?
NSArray内存中存储地址连续，而NSSet不连续
NSSet效率高，内部使用hash查找，NSArray需要遍历查找。
NSSet通过anyObject访问元素，NSArray通过下标访问。

二三、NSHashTable与NSMapTable?
NSHashTable是NSSet的通用版本，对元素若引用，可变类型，可以再访问成员时copy
NSMapTable是NSDictiony的通用版本，对元素弱引用，可变类型;可以在访问成员时copy
（注：NSHashTable与NSSet的区别：NSHashTable可以通过option设置元素弱引用/copyin，只有可变类型。但是添加对象的时候NSHashTable耗费时间是NSSet的两倍。NSMapTable与NSDictionary的区别：同上）

二四、weak属性如何自动置为nil的?
Runtime会对weak属性进行内存布局，构建hash表，以weak属性对象内存地址为key,weak属性（weak自身地址）为value,当对象引用计数为0的时候dealloc时，会将weak属性自动置nil.

二十五、category的加载是发生在运行时，加载category的过程
1、把category的实例方法、协议以及属性添加到类

上。
2、把category的类方法和协议添加到类的metaclass上。

二十六、runtime中，SEL和IMP的区别?
每个类都有一个方法列表，方法列表储存方法名，方法实现，参数类型，SEL是方法名（编号）、IMP是指向方法实现的首地址。

二十七、autoreleasepool的原理和使用场景?
若干个autoreleasepoolpage组成的双向链表的栈结构，objc_autoreleasepoolpush、objc_autoreleasepoolpop、objc_autorelease
使用场景：多次创建临时变量导致内存上涨时，需要延迟释放
autoreleasepoolpage的内存结构：4k储存大小

二十八、NSOperation和GCD的区别
GCD底层使用C语言编写高效、NSOperation是对GCD的面向对象的封装，对于特殊需求，如取消任务、设置任务优先级、任务状态监听，NSOperation更加方便。
NSOperation可以设置依赖关系，而GCD只能通过dispatch_barrier_async实现
NSOperation可以通过KVO观察当前operation执行状态（执行/取消）。
NSOperation可以设置自身优先级（queuePriority）。GCD只能设置队列的优先级（DISPATCH_QUEUE_PRIORITY_DEFAULT）,无法在执行的block中设置优先级

NSOperation可以定义operation如NSInvationOperation/NSBlockOperation,而GCD执行任务可以自定义封装但没有那么高的代码复用度。

GCD高效、NSOperation开销相对较高。

二十九、struct和class的区别

class可以继承，struct不可以

class是引用类型，struct是值类型

struct在function里修改property时需要mutating关键字修饰

三十、try、try? 与 try!

try:手动捕获异常

try?:系统帮我们处理，出现异常返回nil;没有异常返回对应的对象

try!:直接告诉系统，该方法没有异常。如果出现异常程序会crash

三十一、GCD详解

GCD中两个核心的概念：任务和队列

任务：执行操作的意思，就是在线程中执行的那段代码。在GCD中是放在block中的，执行任务有两种方式：同步执行 和 异步执行。两者的主要区别是：是否等待队列的任务执行结束，以及是否具备开启新线程的能力。

同步执行(sync):

同步添加任务到指定的队列中，在添加的任务执行结束之前，会一直等待，直到队列里面的任务完成之后再继续执行。只能在当前线程中执行任务，不具备开启新线程的能力。

异步执行（aync）：

异步添加任务到执行的队列中，它不会做任何等待，可以继续执行任务。可以在新的线程中执行任务，具备开启新线程的能力。注意：异步执行async 虽然具有开启新线程的能力，但是不一定开启新线程。这跟任务所指定的队列类型有关。

队列（Dispatch Queue）:队列是指执行任务的等待队列，即用来存放任务的队列。队列是一种特殊的线性表，采用FIFO先进先出的原则，即新任务总是被插入到队列的末尾，而读取任务的时候总是从队列的 头部开始读取，每读取一个任务，则从队列中释放一个任务。

GCD中有两种队列：串行队列和并发队列。两者都符合FIFO先进先出原则，两者的主要区别是：执行顺序不同，以及开启的线程数不同。

串行队列（Serial Dispatch Queue）：

每次只有一个任务被执行。让任务一个接一个的执行。只开启一个线程，一个任务执行完毕后，在执行下一个任务。

并发队列（Concurrent Dispath Queue）：

可以让多个任务同时执行。可以开启多个线程，并且同时执行任务。并发队列的并发功能只有在异步（dispatch_async）方法下才有效。

GCD的使用步骤

1、创建一个队列。

2、将任务追加到任务的等待队列中年，然后系统会根据任务类型执行（同步执行或异步执行）。

---

* 队列创建的方法/获取方法

可以使用dispatch_queue_create方法来创建队列。该方法需要传入两个参数：

第一个参数表示队列的唯一标识，用于DEBUG,可为空。队列的名称推荐使用应用程序ID这种逆序全称域名。

第二个参数用来识别是串行队列还是并发队列。DISPATCH_QUEUE_SERIAL表示串行队列，DISPATCH_QUEUE_CONCURRENT表示并发队列。

串行队列的创建方法

```
dispatch_queue_t queue = dispatch_queue_create("net.baixing.com", DISPATCH_QUEUE_SERIAL);
dispatch_queue_t queue = dispatch_queue_create("net.baixing.com",DISPATCH_QUEUE_CONCURRENT);
```

对于串行队列，GCD默认提供了：主队列（Main Dispatch Queue）

所有放在主队列中的任务，都会放在主线程中执行。

可使用dispatch_get_main_queue() 方法获得主队列。

```
dispatch_queue_t queue = dispatch_get_main_queue();
```

对于并发队列，GCD默认提供了全局并发队列
可以使用dispatch_get_global_queue方法来获取全局并发队列。需要传入两个参数。第一个参数表示队列的优先级，一般用DISPATCH_QUEUE_PRIORITY_DEFAULT。第二个参数暂时没用，用0即可。

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);
```

---

* 任务创建的方法
GCD提供了同步执行任务的创建方法dispatch_sync和异步执行任务创建方法dispatch_async

三十二、创建定时器的三种方式
1、NSTimer
特性:
*存在延迟:不管是一次性还是周期性的timer的实际触发事件的时间,都会与所加入的RunLoop和RunLoopMode有关，如果RunLoop正在执行一个连续的运算
,timer就会被延迟触发，重复性的timer遇到这种情况,如果延迟超过一个周期,则会和后面的触发进行合并，即在一个周期内只会触发一次。但是不管该timer的触发时间延迟的有多离谱，他后面的timer的触发时间总是倍数于第一次添加timer的间隙。
*必须加入RunLoop:使用上面的创建方式,会自动把

timer加入MainRunloop的NSDefaultRunLoopMode中.如果使用以下方式创建定时器,就必须手动加入Runloop;

2、CADisplayLink

3、GCD方式 GCD定时器不受Runloop约束，比NSTimer更加准时。

三十三、宏定义与const的区别
编译时刻：宏是预编译（编译之前处理），const是编译阶段。
编译检查：宏不做检查，不会报编译错误，只是替换，const会编译检查，会报编译错误。
宏的好处：宏能定义一些函数，方法。const不能
宏的坏处：使用大量宏，容易造成编译时间久，每次都需要重新替换。

三十四、为什么要设计metaclass?

先说结论: 为了更好的复用传递消息.metaclass只是需要实现复用消息传递为目的工具.而Objective-C所有的类默认都是同一个MetaClass(通过isa指针最终指向metaclass). 因为Objective-C的特性基本上是照搬的Smalltalk,Smalltalk中的MetaClass的设计是Smalltalk-80加入的.所以Objective-C也就有了metaclass的设计.

实例的实例方法函数存在类结构体中

类方法函数存在metaclass结构体中

三十五、class_copyIvarList() & class_copyPropertyList()区别

class_copyIvarList() 能获取到所有的成员变量,包括花括号内的变量(.h和.m都包括)

class_copyIvarList()获取默认是带下划线的变量

class_copyPropertyList() 只能获取到 以@property关键字声明的中属性(.h和.m都包括)

class_copyPropertyList()获取默认是不带下划线的变量名称.

但是以上两个方法都只能获取到当前类的属性和变量

(也就是说获取不到父类的属性和变量)

三十六、SEL与selector

SEL是selector在object-c中的表示类型,selector是方法选择器，可以理解为区分方法的ID，而这个ID的数据结构是SEL;

三十七、分类category的实现原理

Category就是对装饰模式的一种具体实现，它的主要作用是在不改变原有类的前提下，动态地给这个类添加一些方法。实例方法、属性和协议

好处：可以减少单个文件的体积，可以把不同的功能组织到不同的category里，可以由多个开发者共同完成一个类，可以按需加载想要的category等等。

所有的OC类和对象，在runtime层都是用struct表示的，category在runtime层，用category_t

------在本类和分类有相同的方法时，本类的方法会被覆盖，优先调用分类的方法------

三十八、autoreleasepool

1、什么是autoreleasepool？ Autoreleasepool自动释放池是OC中的一种内存自动回收机制，它可以延迟加入AutoreleasePool中的变量release的时机，在正常情况下，创建的变量会超出其作用域的时候release,但是如果将变量加入AutoreleasePool,那么release将延迟执行。

2、AutoreleasePool的创建和释放

-----创建-----

创建：App启动后，苹果在主线程Runloop里注册了两个Observer,其回调都是_warpRunLoopWithAutoreleasePoolHandler()。

第一个Observer监听的事件是Entry(即进入Loop),其回调内会调用_objc_autoreleasePoolPush()创建自动释放池。优先级最高，保证创建释放池发生在其他回调之前。

-----释放-----

第二个Observer监视了两个事件：BeforeWaiting(准备进入休眠)时调用_objc_autoreleasePoolPop()和_objc_autoreleasePoolPush()释放旧的池并创建新池。Exit(即将推出loop)时调用了_objc_autoreleasePoolPop()来释放自动释放池。优先级最低，保证其释放池子发生在其他所有回调之后。

也就是说AutoreleasePool创建是在一个runloop事件开始之前（push）,AutoreleasePool释放是在Runloop事件即将结束之前（pop）。
AutoreleasePool里的Autorelease对象的加入是在Runloop事件中，AutoreleasePool里的Autorelease对象的释放是在AutoreleasePool释放时。

三十九、load方法和initialize方法的异同
load和initialize都会在实例化对象之前调用，以main函数为分水岭，前者是在main函数之前，后者是在main函数之后。
load和initialize方法都不会显示的调用父类的方法而是自动调用，即使子类没有initialize方法也会调用父类的方法，load方法不会调用父类。
load和initialize方法内部使用了锁，因此他们是线程安全的，实现时要尽可能简单，避免线程阻塞，不要再次使用锁。
load方法常用来method swizzle,initialize尝尝用于初始化全局变量和静态变量。

四十、weak原理实现步骤
weak实现原理可概括三步：
1、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。
2、添加引用时：objc_initWeak函数会调用objc_storeWeak()函数，objc_storeWeak()的作用是更新指针指向，创建对应的弱引用表。

3、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,然后把这个entry从weak表中删除，然后清理对象的记录。

四十一、App从点击屏幕（硬件）到完全渲染的过程。

1、首先一个视图由CPU进行Frame布局，准备视图和图层的层级关系。

2、CPU会将处理视图和图层的层级关系打包，通过IPC(内部处理通信)通道提交给渲染服务，渲染服务由OpenGL ES和GPU组成。

3、渲染服务首先将图层数据交给OpenGL ES进行纹理生成和着色。生成前后帧缓存，再设备的VSync信号和CADisplayLink为标准，进行前后帧缓存的切换。

4、将最终要显示在画面上的后帧缓存交给GPU,进行采集图片和形状，运行变换，应用纹理和混合，最终显示在屏幕上。

四十二、AFNetworking是如何玩转RunLoop的

```
//为了不让runloop run起来没事干导致消失
//所以给runloop加了一个NSMachPort,给它一个mode去监听
//实际上port什么也没干，就是让runloop一直在等，目的就是让runloop一直活着
//这是一个创建常驻服务线程的好方法
NSRunloop *runloop = [NSRunLoop currentRunLoop];
[runloop addPort:[NSMachPort port]
```

```
forMode:NSDefaultRunLoopMode];
 [runloop run];
```

四十三、MVC/MVVM/MVP

MVC:

1、model和view永远不能互相通信、只能通过controller传递。

2、Controller 可以直接与 Model 对话（读写调用 Model），Model 通过 Notification 和 KVO 机制与 Controller 间接通信。

3、Controller 可以直接与 View 对话，通过 outlet，直接操作 View，outlet 直接对应到 View 中的控件，View 通过 action 向 Controller 报告事件的发生(如用户 Touch 我了)。Controller 是 View 的直接数据源（数据很可能是 Controller 从 Model 中取得并经过加工了）。Controller 是 View 的代理（delegate），以同步 View 与 Controller

MVP:

任务均摊 -- 我们将最主要的任务划分到 Presenter 和 Model，而 View 的功能较少；

可测试性 -- 非常好，由于一个功能简单的 View 层，所以测试大多数业务逻辑也变得简单；

易用性 -- 代码量比 MVC 模式的大，但同时 MVP 的概念却非常清晰。

模型与视图完全分离，我们可以修改视图而不影响。

可以更高效地使用模型，因为所以的交互都发生在一个

地方——Presenter内部。
我们可以将一个Presener用于多个视图，而不需要改变Presenter的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁。
如果我们把逻辑放在Presenter中，那么我们就可以脱离用户接口来测试这些逻辑（单元测试）

MVVM:
任务均摊 —— MVVM 的 View 要比 MVP 中的 View 承担的责任多。因为前者通过 ViewModel 的设置绑定来更新状态，而后者只监听 Presenter 的事件但并不会对自己有什么更新。
可测试性 —— ViewModel 不知道关于 View 的任何事情，这允许我们可以轻易的测试 ViewModel。同时 View 也可以被测试，但是由于属于 UIKit 的范畴，对他们的测试通常会被忽略。
易用性 —— 在实际开发中必须把 View 中的事件指向 Presenter 并且手动的来更新 View，如果使用绑定的话，MVVM 代码量将会小的多。

四十四、三次握手
详细的：
第一次握手：客户端向服务起发出连接请求报文，这时报文首部中的同部位SYN=1,同时随机生成初始序列号seq=x,此时，客户端进入了SYN-SENT状态，等待服务器的确认。
第二次握手：服务器收到请求报文后，如果同意连接，则发出确认报文。确认报文中应该ACK=1,SYN=1,确认号是ack=x+1,同时也要为自己随机初始化一个序列号

seq=y,此时，服务器进程进入了SYN-RCVD状态，询问客户端是否做好准备。
第三次握手：客户端进程收到确认后，还要向服务器给出确认。确认报文的ACK=1,ack=y+1,此时建立连接，客户端进入ESTABLISHED状态，服务器端也进入ESTABLISHED状态。

简单的：
第一次握手：客户端给服务器发送一个SYN报文。
第二次握手：服务器收到SYN报文之后，会应答一个SYN+ACK报文。
第三次握手：客户端收到SYN+ACK报文之后，会应答一个ACK报文。

三次握手的作用
1、确认双方的接收能力，发送能力是否正常。
2、指定自己的初始化序列号，为后面的可靠传送准备。
3、如果是https协议的话，三次握手这个过程，还会进行数字证书的验证以及加密密钥的生成。

四次挥手：
1、第一次挥手：客户端发送一个FIN报文，报文中会指定一个序列号。此时客户端处于CLOSED_WAIT1状态。
2、第二次握手：服务端收到FIN之后，会发送ACK报文，且把客户端的序列号值+1作为ACK报文的序列号值，表明已经收到客户端的报文了，此时服务端处于CLOSE_WAIT2状态。
3、第三次挥手：如果服务端也想断开连接了，和客户

端的第一次挥手一样，发给FIN报文，且指定一个序列号。此时服务端处于LAST_ACK的状态。

4、第四次挥手：客户端收到FIN之后，一样发送一个ACK报文作为应答，且把服务端的序列号值+1作为自己ACK报文的序列号值，此时客户端处于TIME_WAIT状态。需要过一阵子以确保服务端收到自己的ACK报文之后才会进入CLOSED状态。

5、服务端收到ACK报文之后，就处于关闭连接了，处于CLOSED状态。

```
//选择排序
void selectSort (int *arr , int length){
    for (int i = 0; i < length - 1; i++)
{
        for (int j = i + 1; j < length;
j++) {
            if (arr[i] > arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

//冒泡排序
void bubblingSort (int *arr,int length){
    for (int i = 0; i < length - 1; i++)
{
        for (int j = 0; j < length - i -
```

```
1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```