# IOS 知识点

**1. 类别(分类)、类扩展和继承实现区别**

分类：可以给类添加方法和实现，但是无法添加成员变量或属性(必要时可以利用runtime机制动态添加属性)；分类的修改是向下有效的，即改分类可以影响该类所有子类，因此如果如果目标类有很多子类，不想影响其他子类时用继承反之用分类；当方法名和类一致时覆盖父类方法；用来扩展系统类比较方便。

继承：子类继承父类的所有方法和属性，可以添加新的属性和方法，当方法名和父类一致时覆盖父类方法；

类扩展：可以添加新的方法和属性，但是只能类内部使用，外部无法调用；

**2. #import、#include、@class的区别**

#include是c语言中引入头文件的方式，其中#include””表示导入自定义文件，#include《》表示导入外部库文件；#include不能防止重复引用头文件的问题；

可以使用#ifndef #define #endif的方式防止重复导入；

#import 是oc语言中引入头文件的方式，可以防止重复引入头文件的问题；

@class表示告诉编译器引入了某个类，编译时并不会去检查该类的头文件，而使用#import引入头文件，编译时要去检查该类的头文件中的方法和属性，因此，在不调用该类的方法或属性时，使用@class能增加编译速度，还可以防止循环引用的问题；

**3. tcp和udp传输的特点和优缺点**

TCP是面向连接的，可靠的传输层协议，tcp传输要经过三次握手建立连接，然后进行传输，可以保证数据正确性和数据顺序，用于传输大量数据，建立连接耗费更多系统资源，可重发处理丢包；

UDP是面向无连接的，传输不可靠，用于少量数据传输，速度快，但不保证传输数据正确性，也不管接收对象是谁，只负责传输，不安全，不能处理丢包。

4. 说说tcp传输的三次握手和四次挥手

第一次握手：由客户端发送SYN包到服务器，并进入SYN——SEND状态，然后等待服务器回应。

第二次握手：服务器收到SYN包，并确认客户端SYN，同时服务器也发送一个SYN包，即SYN+ACK到客户端，并进入SUN——RECV状态，等待回应。

第三次握手：客户端接收到SYN+ACK包，并向服务器发送确认包ACK，完成三次握手。

客户端和服务器都可以发起关闭TCP链接，过程需要四次挥手

第一次挥手：客户端发送一个FIN报文给服务器，等待服务器响应。

第二次挥手：服务器收到FIN报文后，确认报文是客户端发起，同时发送ACK报文到客户端，客户端收到报文后进入FIN\_WAIT2状态，等待服务器发送FIN报文。

第三次挥手：服务器发送FIN终止报文，进入LAST\_ACK状态，等待最后的确认报文段。

第四次挥手：客户端收到终止报文段后，向服务端发送一个确认报文段，同时客户端断开链接，释放资源，服务端收到报文段，断开链接释放资源。

**5. Socket和Http的区别**

网络七层下到上分别为：物理层、数据链路层、网络层、传输层、回话层、表示层和应用层。

http对应应用层、tcp对应传输层、ip对应网络层。

Socket是基于TCP传输的长链接，建立连接要三次握手，一旦建立连接不会自动断开，需要客户端或服务端主动断开链接

http是超文本传输协议，特点是由客户端主动发起请求，服务器响应，响应结束自动断开链接，这个过程称为一次链接，是短链接,

**6. runtime理解及应用**

oc语言是一门动态语言，方法调用消息发送在运行时进行，意思当我们调用一个未实现的方法时，在编译阶段并不会报错，在运行时才会报错。具体过程是，当运行时执行一个方法调用时，编译器会将其转化成消息发送objc\_msgSend()（两个参数为实例名和方法名）去查找这个函数，首先通过实例isa指针找到该类，然后查找cache缓存池，如果没有找到该方法，去类的列表去查找，如果没有再逐级去父类查找，知道找到该方法，执行它的方法实现，并将该方法以名为key以实现为value加入缓存，以方便下次快速调用。如果最终也没有找到，这时程序还不会crash，运行时会去调用resolveInstanceMethod方法，我们可以在这个方法中动态添加实现，如果这个方法不做处理，我们还可以实现forwardingtargetforselector方法，设置备用接收者实现该方法，如果这个方法也没有处理，程序抛出异常。

Runtime的使用，动态添加方法，减少方法列表数量，给分类动态关联属性，遍历属性字典转模型（遍历model属性，如果属性在json中有对应值，则负值）。

KVO实现原理：当其对象属性被改变时能通知当前对象的机制，当观察某对象时，kvo机制动态创建该对象当前类的子类，并为这个子类重写被观察对象的settter方法，该方法负责在原类setter方法调用前后[super setValue:forKey]，调用[self willChangeValueForKey]和[self didChangeValueForKey]来通知属性变化。

KVC 实现原理：可以允许开发者通过key名直接访问对象的属性，或给对象的属性赋值，从而在运行时动态访问和修改对象的属性。所有继承了NSObject的类型，都支持KVC。如果没有找到setKey方法，会按照\_key,\_isKey,key,isKey的顺序来搜索成员并赋值，如果都没找到调用setValue：forUndefineKey方法抛出异常，所以我们可以重写此方法空实现来避免crash。

禁用KVC可以重写accessInstanceVariablesDirectly方法返回NO，

**7. runLoop理解及应用**

Runloop是通过内部维护的do-while事件循环来不停的处理事件的一个对象。它和线程是息息相关的，正常情况下我们用线程来执行一个或多个任务，执行完毕线程就会退出，因此，我们需要一种方式来让线程能够不断的处理任务，就有了runloop。

runloop会在循环中不停检测输入源和定时源，如果检测到唤醒线程处理。如果没有输入源则通知线程休眠。

核心是保证线程在没有消息处理时，让线程休眠，以节省cpu资源，提高性能，在有事件处理时唤醒线程处理事件。

Runloop5个相关类：CFRunLoopRef：代表Runloop对象；CFRunLoopModeRef：代表Runloop的运行模式；\*\*SourceRef：代表输入源；\*\*TimerRef：代表定时源；\*\*ObserverRef：代表观察者监听runloop状态。

RunloopModel相关

K CFRunloopDefaultModel：App的默认运行模式，主线程在这个模式下运行。

UITrackingRunloopModel：跟踪用户交互事件，用于ScrollView追踪触摸滑动，保证界面滑动时不受其他model影响。

KCFRunloopCommonModels：不是一种运行模式，是种标记位。

定时器和Scrollview冲突原因及解决方案：

原因是当我们不做任何操作时，当前runloop处于默认模式下运行，当我们拖动Scrollview的时候，runloop会结束默认模式，切换到UITrackingRunloopModel模式下，而定时器被添加在默认模式下，因此定时器停止工作。

解决办法：我们将定时器添加在标记位NSRunloopCommonModels模式下，因为默认模式和TrackingModel都是被标记的模式，这样定时器在两个模式下都可以工作了。

Runloop实际应用：

1. NSTimer的使用

2. ImageView推迟显示，当界面有tableview时，cell中都有图片，当我们滚动tableview时如果有很多图片需要显示，就会造成卡顿现象，就需要推迟显示。

1>监听Scrollview代理，在滚动结束时设置图片。

2>利用PerformSelector方法设置当前线程的runloop运行模式为默认模式[self.imageView perFormSelector:@selector(setImage:) withObject:[UIImage imageNamed:@”tupian”] afterDelay:4.0 inModes:NSDefaultTunLoopMdoe]

3. 后台常驻线程

**8. 内存管理篇**

程序内存空间分布：

栈区：由编译器自动管理内存分配释放，存放函数参数值和局部变量值。

堆区：由程序员分配释放。

全局区或静态区：存放全局变量和静态变量，由系统释放。

文字常量区：存放常量字符串，由系统释放。

程序代码区：存放代码。

iOS 5.0之前是由程序猿使用引用计数手动管理对象生命周期集内存，即当对象被以alloc new copy mutablecopy的方式被创建时，其引用计数会被+1，当对象被放弃持有即release autorelease时，内存计数-1，当引用计数为0时，对象内存释放。

5.0之后引入自动引用计数。引入\_strong \_weak。

内存管理原则：谁创建，谁释放，谁引用，谁管理。

僵尸对象：一个OC对象引用计数为0，对象内存已被标记回收，可是对象数值还存在，还没有被新数据覆盖，则该对象称为僵尸对象，该对象不能被访问，会crash。

空指针：空指针是有效指针，值为nil，可以给空指针发送消息。

野指针：指针指向了一个已经被销毁的对象的内存地址，向野指针发消息会crash，报EXC\_BAD\_ACCESS。

Runtime维护了一个weak表，用于存储指向某个对象的所有指针，key是该对象的内存地址value是所有weak指针地址的数组。

直接创建的block是存放在栈区的，不需要我们管理内存，当block作为返回值或属性被copy时，block会被拷贝到堆区，需要我们管理，主要注意循环引用。

数组enumerate遍历方法block里的对象和数组、字典、字符串的类方法创建，即非init和new生成的对象，被放在autorelease管理

nsstring内存管理

不可变字符串以format方式创建的，存放在堆区，引用计数+1，其他方式均存放在栈区不引起引用计数改变。可变字符串都是存放在堆区，引起计数+1。

**9. 多线程篇**

进程：是指在系统中正在运行的具有一定独立功能的应用程序，可以简单看成手机上一个app。

线程：进程执行任务的最小单元，一个进程可以有多个线程，至少要有一个线程，所有线程共享进程资源。

多线程：同一时间，cpu只能处理1条线程，多线程并发，实际上是cpu快速的在多条线程之间切换，若果切换的足够快，就会造成多线程并发的假象。

优点：能适当提高程序的执行效率，提高cpu利用率。

缺点：开启线程需要占用一定的内存空间，如果大量开启，会降低程序性能，cpu开销大。

NSThread：比较轻量级的技术，需要手动开辟子线程手动开启线程，如果使用构造器方式会自动开启。需要管理线程生命周期，线程同步和加锁问题。

GCD是c语言API封装成的，NSOperationQueue是对GCD的封装，是GCD的更高抽象。

二者有缺点：

1. GCD执行效率更高，使用方便。

2. GCD只支持FIFO（先进先出）的队列，而NSO可以设置最大并发量，设置优先级，调整执行顺序。

3.NSO支持KVO，可以监测NSO是否正在执行，是否结束，是否取消。

4.一般不需要考虑复杂的线程关系时，比如顺序，依赖关系时，建议使用运行快速的GCD。反之使用NSO。

Dispatch\_async：异步执行；

Dispatch\_sync：同步执行；

Dispatch\_group\_async和Dispatch\_group\_notify：在几个耗时任务完成后，再去执行接下来的任务，比如，多个网络请求后刷新UI。

Dispatch\_after：延时执行，确切说是延时加入队列。

Dispatch\_once：单次执行函数，实现单例。

**10. 线程锁篇**

最简单的死锁同步线程在主队列里执行就会造成死锁。

处理死锁的方法：破坏死锁产生的条件来解决死锁，通过算法合理分配资源来避免死锁发生。

一、什么是线程死锁

是指两个或两个以上的线程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。

二、线程死锁怎么发生

　发生死锁的情况一般是两个对象的锁相互等待造成的。

三、死锁发生的条件

1、互斥条件：所谓互斥就是进程在某一时间内独占资源。

2、请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

3、不剥夺条件:进程已获得资源，在末使用完之前，不能强行剥夺。

4、循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

死锁通常是一个线程锁定了一个资源A，而又想去锁定资源B；在另一个线程中，锁定了资源B，而又想去锁定资源A以完成自身的操作，两个线程都想得到对方的资源，而不愿释放自己的资源，造成两个线程都在相互等待，造成了无法执行的情况。

三、怎么解决线程死锁

当几个线程都要访问共享资源A、B、C时，保证使每个线程都按照同样的顺序去访问它们，比如都先访问A，在访问B和C。

四、自旋锁和互斥锁

相同点：

都能保证同一时间只有一个线程访问共享资源。都能保证线程安全。

不同点：

互斥锁：如果共享数据已经有其他线程加锁了，线程会进入休眠状态等待锁。一旦被访问的资源被解锁，则等待资源的线程会被唤醒。  
自旋锁：如果共享数据已经有其他线程加锁了，线程会以死循环的方式等待锁，一旦被访问的资源被解锁，则等待资源的线程会立即执行。

自旋锁的效率高于互斥锁。

条件锁基本使用NSCondition和NSConditionLock就足够使用。

自旋锁：OSSpinLock 在ios中已经不是线程安全的了，如果共享数据已经有其他线程加锁了，线程会以死循环的方式等待锁，一旦被访问的资源被解锁，则等待资源的线程会立即执行。（效率最高，如果一直等不到锁会较占用cpu资源）

信号锁：dispatch\_semaphore是gcd中通过信号量来实现共享数据的数据安全。（效率第二）

互斥锁：pthread\_mutex ，nslock ，synchronized都是互斥锁。如果共享数据已经有其他线程加锁了，线程会进入休眠状态等待锁。一旦被访问的资源被解锁，则等待资源的线程会被唤醒。（synchronized效率最低）

递归锁：pthread\_mutex(recursive)与NSRecursiveLock ， 多次调用不会阻塞已获取该锁的线程。

条件锁：nsconditionlock 满足一定的条件的加锁和解锁，可以实现依赖关系。nscondition条件锁，也是通过信号量来解锁，主要用来实现生产者消费者模式。

**11. Block篇**

Block:带有自动变量的匿名函数，是c语言的一种扩充。是将函数及其执行上下文封装起来的对象，具有参数和返回值。关于“带有自动变量（局部变量）”的含义，这是因为Block拥有捕获外部变量的功能。在Block中访问一个外部的局部变量，Block会持用它的临时状态，自动捕获变量值，外部局部变量的变化不会影响它的的状态。

**block的定义方式：**

//typedef简化block声明 声明一类block

typedef NSString\* (^TypeBlock)(NSString \*str);

//不带参数和返回值

@property (nonatomic, copy) void (^voidBlock)(void);

//带参数和返回值

@property (nonatomic, copy) NSString\* (^returnBlock)(NSString \*str);

//typedef简化block的声明

@property (nonatomic, copy) TypeBlock tBlock;

//以方法参数形式使用

- (void)testBlockOne:(TypeBlock)block;

-(void)testBlockTwo:(NSString\* (^)(NSString \*str))testBlock;

注意：static静态变量只会被初始化一次，可以修改值；const修饰变量不可修改；

1. 局部变量截获，是值的截获。

2. 局部静态变量截获，是指针的截获。

3. 全局变量，静态全局变量截获，不截获，直接取值。

4. 不使用外部变量的block是全局block。

5. 使用外部变量并且未进行copy操作的是栈block。

6. 对栈blcok进行copy操作，就是堆block，需要管理内存。

**12. 事件响应链篇**

响应者：能响应事件的UIResponder子类对象，如按钮和UIView；

相应链：由多个响应者组合起来的链条，就叫响应者链。实际应用中当用户点击屏幕后，我们把从第一响应者到UIApplication对象的一系列响应者即为响应者链。

事件传递：获得响应链后，将事件由第一响应者往UIApplication传递的过程称为事件传递。

整个流程：当IOS程序发生触摸事件后，系统会利用runloop将事件加入到UIApplication的任务队列中，然后分发到UIWindow，然后UIWindow的view调用hitTest：withEvent：方法看看自己是否有处理事件的能力，并调用当前视图的pointInside：withEvent方法，判断触摸点是否在view上，返回yes在，返回no不在。若返回yes，则遍历其子视图并对其子视图发送hitTest方法，判断触摸点是否在某个子视图上，一次类推，直到找到第一响应者。然后判断第一响应者能否响应触摸事件，如能响应链终止，如不能则向其父视图传递并判断能否响应，一次类推，直到上报到UIApplication也无法响应，则上报给appdelegate，若最终事件依旧无法响应，则被系统抛弃。

响应者对事件是否能处理，是在事件响应过程中判断的，查找第一响应者，只是判断是否有响应事件的能力。

视图不响应检查：hidden = yes视图被隐藏；userInteractionEnabled = NO不接受事件响应；透明视图不接受响应事件；子视图超出父视图范围；响应视图被其他视图覆盖；重写了hitTest；重写了pointInside方法；

应用：修改按钮点击范围。

**13. HTML5与原生交互**

与UIWebView交互：

1、OC调用JS

　 （1）OC调用代码

[self.webView stringByEvaluatingJavaScriptFromString:@"ocCallJSFunction({'name':'xiaoxiao'})"];

　 （2）JS代码

</script>  
　　　function ocCallJSFunction(data) {  
　　　　var obj = eval(data);  
　　　　alert(obj.name);  
　　　}  
　　<script>

2、JS调用OC

（1）OC代码

　　- (void)webViewDidFinishLoad:(UIWebView \*)webView {

    　　JSContext \*context = [webView valueForKeyPath:@"documentView.webView.mainFrame.javaScriptContext"];

    　　context[@"jsCallOCFunction"] = ^() {

       　　 NSLog(@"JS调用OC成功");

       　　 NSArray \*args = [JSContext currentArguments];

        　　for (JSValue \*jsVal in args) {

            　　NSLog(@"%@", jsVal.toDictionary);

      　　  }

  　　  };

　　}

　（2）JS调用代码

　　</script>

　　var message = {  
　　　　'method' : 'hello',  
　　　　'param1' : 'dada',  
　　　};  
　　jsCallOCFunction(message);

<script>

WKWebView交互：

1、OC调用JS

　 （1）OC调用代码

　　　　[self.wkWebView evaluateJavaScript:@"ocCallJSFunction({'name':'xiaoxiao'})" completionHandler:^(id \_Nullable obj, NSError \* \_Nullable error) {

       　　　　 NSLog(@"evaluateJavaScript, obj = %@, error = %@", obj, error);

   　　 }];

　（2）JS代码

　　</script>  
　　　function ocCallJSFunction(data) {  
　　　　var obj = eval(data);  
　　　　alert(obj.name);  
　　　}  
　　<script>

2、JS调用OC

（1）OC代码

　　// 初始化WKWebView

　　WKWebViewConfiguration \*config = [[WKWebViewConfiguration alloc] init];

   　 [config.userContentController addScriptMessageHandler:self name:@"jsCallOCFunction"];

      WKWebView \*wkWebView = [[WKWebView alloc] initWithFrame:CGRectZero configuration:config];

　　#pragma mark - WKScriptMessageHandler

　　- (void)userContentController:(WKUserContentController \*)userContentController didReceiveScriptMessage:(WKScriptMessage \*)message {

    　　NSLog(@"name = %@",message.name);

    　　NSLog(@"body = %@",message.body);

   　　 NSLog(@"frameInfo = %@",message.frameInfo);

　　}

　（2）JS调用代码

　　</script>

　　var message = {  
　　　　'method' : 'hello',  
　　　　'param1' : 'dada',  
　　　};  
　　window.webkit.messageHandlers.jsCallOCFunction.postMessage(message);

　　<script>

三、交互（WebViewJavascriptBridge）

功能总结：

OC调JS调用成功，JS可以回调数据到OC

JS调OC调用成功，OC可以回调数据到JS

1、OC调用JS

　 （1）OC调用代码

　　\_bridge = [WebViewJavascriptBridge bridgeForWebView:wkWebView];

　　id data = @{@"name" : @"xiaoxiao"};

   　 [\_bridge callHandler:@"ocCallJSFunction" data:data responseCallback:^(id response) {

       　　 NSLog(@"这里是OC调用JS成功后，JS回调的参数:%@", response);

   　 }];

　（2）JS代码

　　</script>

　　// 注册JS方法供OC调用  
　　bridge.registerHandler('ocCallJSFunction', function(data, responseCallback) {  
　　　　var obj = eval(data);  
　　　　alert(obj.userId);

　　　　var responseData = { 'code':'200' }  
　　　　responseCallback(responseData)  
　　})

　　<script>

2、JS调用OC

（1）OC代码

　　[\_bridge registerHandler:@"jsCallOCFunction" handler:^(id data, WVJBResponseCallback responseCallback) {

        NSLog(@"JS传过来的参数: %@", data);

        responseCallback(@"这是JS调用OC成功后，OC回调的参数");

    }];

　(2）JS调用代码

　　// JS调用OC注册的方法  
　　bridge.callHandler('jsCallOCFunction', {'name':'dada'}, function responseCallback(responseData) {

　　　　alert(responseData);  
　　})

**14.设计模式**

常用设计模式：单例模式，代理模式，mvc，mvvm，观察者模式。

**代理模式**：当一个类的某些功能需要由别的类来实现，但又不确定具体要谁来实现时，使用代理模式。包括协议，委托对象，代理对象。

协议：用来指定双方做什么，即声明方法；

委托对象：调用协议中的方法，并声明协议类型的delegate (如：@property (nonatomic,weak)id<SDCycleScrollViewDelegate>delegate) ；

代理对象：实现协议中的方法，并设置delegate的实现对象 (\*\*.delegate = self)；

使用实例：tableview。

单例模式：在整个应用程序中，这个类只会被初始化一次，只有一个实例，可以节约系统资源。

当需要整个应用程序共享一份资源时，或某个控制器或方法或数据需要多次使用(比如登陆接口里拿到的数据，可能是整个程序随处使用的数据)，可以考虑使用单例模式，如网络请求工具类。

优点：保证系统只有一个实例，且易于外界访问，节约系统资源。

缺点：单例一旦建立，对象指针保存在静态区，单利对象在堆区分配的空间要在程序终止后才会释放。单例无法继承，很难扩展。

实例：UIApplication Appdelegate

观察者模式：kvo

mvc模式：

**15.算法篇**

**16.存储方式篇**

沙盒路径结构：1.Document：用于存储用户创建的文档文件的目录，可通过配置实现itunes共享文件。2.Library：该目录下默认有两个子目录cache，Preferences。Cache目录下用作缓存数据使用，用户磁盘空间使用完毕时系统会删除该文件夹下的文件，在app使用期间不会，app没有运行时可能会。Preferences用于存储NSUserdefaults保存的数据。3.tmp用于保存临时使用的数据。

综上所述：永久保存文件可以保存在Document文件夹下或自己创建的目录，临时缓存的文件可以存储到tmp或cache目录下。

iOS常用五种存储方式

1. NSUserDefault 适合存储轻量级的本地数据，能够快速通过键值对存取值，使用方便。支持数据类型NSNumber NSString NSArray NSDictionary BOOL NSData。不能存储自定义数据，取出的数据都是不可变的。存放路径在Library/preference下，格式为Plist。存放登陆返回常用数据。

2. plist存储：创建.plist文件路径，字典加入数据，使用该方法[dic writeToFile:路径 atomically：yes]存储，取值dic = [NSDictionary dictionaryWithContentsOfFile:路经]。

3. 归档：创建person类，遵守NSCoding协议

@interface Person ： NSObject<NSCoding>

@property(nonatomic，strong)NSString \*name；

@end

@implementation Person

- (instancetype)initWithCoder：(NSCoder \*)coder

{

self = [super init];

if(self){

\_name = [coder decodeObjectForKey:@”name”];

}

return self;

}

- (void)encodeWithCoder：(NSCoder \*)coder

{

[coder encodeObject:self.name forKey:@”name”];

}

@end

归档

Person \*per = [[Person alloc] init];

Per.name = @”liu”;

NSData \*data = [NSKeyedArchiver archiveDataWithRootObject:per];

然后加入数组写入本地路径或

[NSKeyedArchiver archiveRootObject:per toFile:路径]

解档

Person \*per = [NSKeyedUnarchiver unarchiveObjectWithFile:路径];

4.coreData用于大量模型数据持久化存储 5.sqlite。

**17.weak和assign的区别**

两种修饰符修饰的指针变量，都不会改变引用对象的引用计数，但是在一个对象释放后，weak指针会自动置为nil，assign则不会，会造成野指针crash。

**18.传值方式篇**

1.block传值；2.代理；3.通知中心；4.属性传值； 5.单例传值；5.本地存储。

**19.性能优化篇**

1.tableview的重用机制，cell上尽可能减少子view，view尽量不要用透明度和圆角，view使用图片应和图片尺寸一致，缓存cell高度，delegate里直接返回cell高，不做其他逻辑处理。

2.对不会发生改变，并多次使用的资源做缓存，比如图片，cell行高。

3.在代理里监听内存警告，移除一些缓存或不可见的view。

4.对于重大开销对象，不要多次创建，比如NSDateFormatter和NSCalender初始化很慢。使用懒加载或者单例创建，这就要权衡内存使用和卡顿性能了。

5.延时操作在异步线程操作。

**20.推送APNS原理篇**

APNS即苹果远程推送服务器，是个推送中转站，我们的后台服务器或者第三方推送服务器推送给客户端的消息会先统一发送到苹果APNS服务器，然后由其根据设备唯一识别码发送到指定手机。

例子：

首先我们的为app申请推送的开发证书和发布证书，导入钥匙串，同时xcode设置勾选允许推送功能。然后在app的delegate的didFinishLaunching方法中注册远程推送，此时app会默认和APNS建立长链接，并切把设备UDID和Bundle ID发送给APNS服务器，APNS会根据二者生成deviceToken返回给ios设备。然后我们实现推送相关的几个代理，把deviceToken发送给服务器，服务器就可以根据deviceToken向APNS发送推送信息了。APNS接到信息会根据deviceToken查找映射表找到对应的UDID和Bundle ID，然后把消息推送过去。

21.地图使用篇

22.socket长链接篇

23.动画使用篇

24.get和post的区别

URL：包括协议部分Http：；域名[www.xxx.com](http://www.xxx.com)；端口：8080；参数部分

Get：可以从服务器上获取数据，但访问时把参数数据直接添加在访问路径中容易被外界看到，安全性差。

Post：可以从服务器获取数据，但访问数据参数请求body体中，安全性较高，不易被外界获取。

Get方式主要用来查询服务器数据；post用来做数据添加，修改或删除。