简说

1. JS-OC

先创建了WKWebView,然后配置，利用WKUserContentController的addScriptMessageHandle来添加处理JS调用OC的方法，有时网页适配兼容问题，利用WKUserScript来注入JS，OC调用JS时，会调用webView的evaluateJavaScript方法，completionHandler是回调处理，JS调用OC会调用didReceiveScriptMessage，根据WKUserContentController的addScriptMessageHandle添加的方法名来判断调用的是什么方法;

复杂一点的就是cookie问题，cookie丢失问题解决方式就是在request中手动帮其添加上，加载网页的时候，在请求头里面添加一次cookie但这只能解决第一次进入的cookie问题，如果页面内跳转（a标签等）还是取不到cookie，因此还要再加代码。 拼凑js字符串，执行js的 webView evaluateJavaScript:JSCookieString completionHandler:nil;

2、3DTouch

如果是应用图标添加，就配置[UIApplicationShortcutItems]这个数组（Title，Type，IconType，Icon，Subtitle，UserInfo） 也可以用纯代码配置

如果是UITableViewCell实现

if (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable) {

NSLog(@"3D Touch 可用!");

//给cell注册3DTouch的peek（预览）和pop功能 ，注册(在哪个页面上使用该功能就注册在哪个页面上)

[self registerForPreviewingWithDelegate:self sourceView:cell];

} else {

NSLog(@"3D Touch 无效");

}

再实现代理方法[previewingContext：viewControllerForLocation]

3、图片加水印

// 给图片添加文字水印

-(UIImage \*)jx\_WaterImageWithImage:(UIImage \*)image text:(NSString \*)text textPoint:(CGPoint)point attributedString:(NSDictionary \* )attributed{

    //1.开启上下文

    UIGraphicsBeginImageContextWithOptions(image.size, NO, 0);

    //2.绘制图片

    [image drawInRect:CGRectMake(0, 0, image.size.width, image.size.height)];

    //添加水印文字

    [text drawAtPoint:point withAttributes:attributed];

    //3.从上下文中获取新图片

    UIImage \* newImage = UIGraphicsGetImageFromCurrentImageContext();

    //4.关闭图形上下文

    UIGraphicsEndImageContext();

    //返回图片

    return newImage;

}

// 给图片添加图片水印

+ (instancetype)waterMarkWithImageName:(NSString \*)backgroundImage andMarkImageName:(NSString \*)markName{

UIImage \*bgImage = [UIImage imageNamed:backgroundImage];

UIGraphicsBeginImageContextWithOptions(bgImage.size, NO, 0.0);

[bgImage drawInRect:CGRectMake(0, 0, bgImage.size.width, bgImage.size.height)];

UIImage \*waterImage = [UIImage imageNamed:markName];

CGFloat scale = 0.3;

CGFloat margin = 5;

CGFloat waterW = waterImage.size.width \* scale;

CGFloat waterH = waterImage.size.height \* scale;

CGFloat waterX = bgImage.size.width - waterW - margin;

CGFloat waterY = bgImage.size.height - waterH - margin;

[waterImage drawInRect:CGRectMake(waterX, waterY, waterW, waterH)];

UIImage \*newImage = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

return newImage;

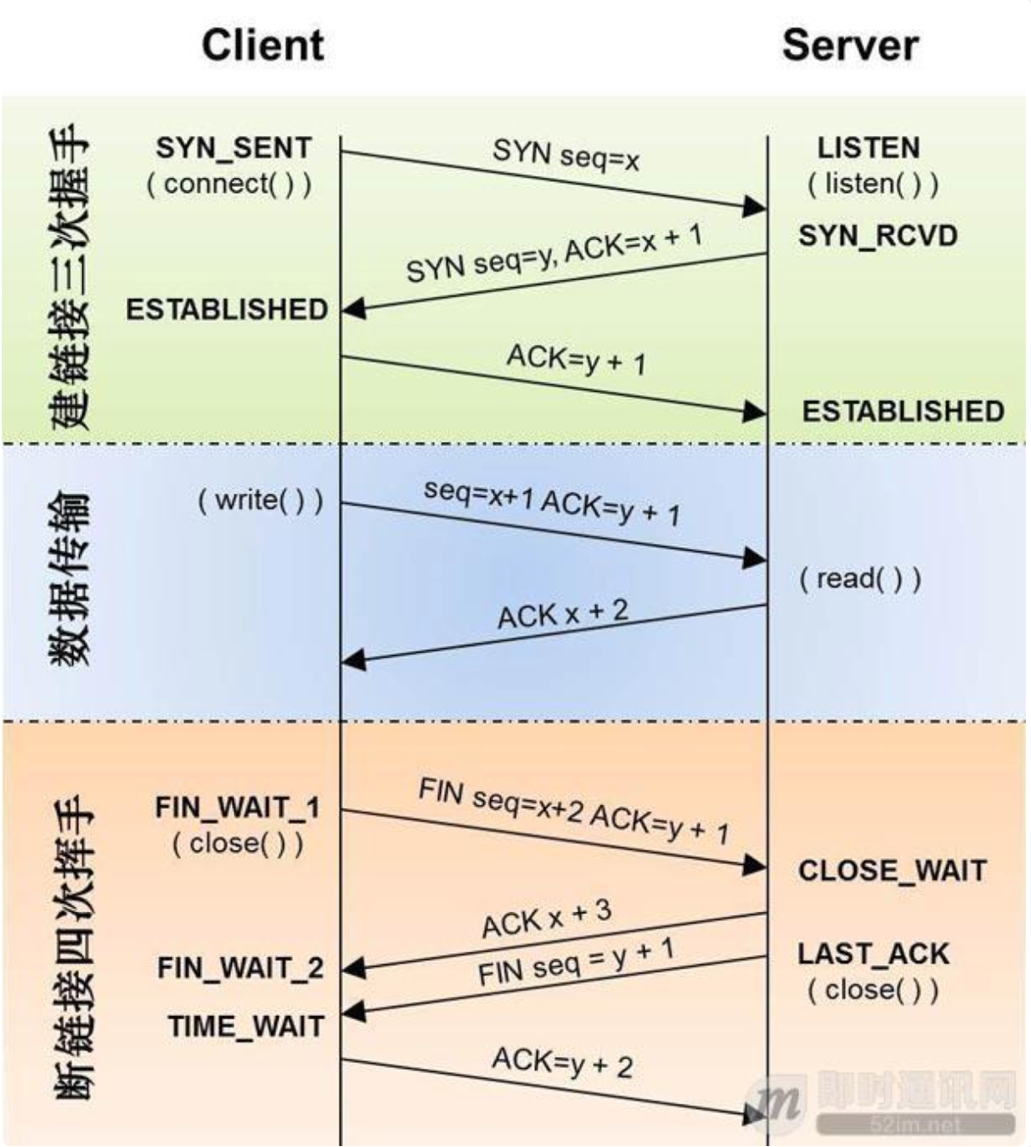
}

4、TCP / UDP

都属于传输层协议

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)

TCP 通过“三次握手”来建立连接，再通过“四次挥手”断开一个连接。



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。

就好比发短信一样，UDP 只需要知道对方的 ip 地址，将数据报一份一份的发送过去就可以了，其他的作为发送方，都不需要关心。

**数据发送方式**

Tcp发送数据

由于 TCP 是建立在两端连接之上的协议，所以理论上发送的数据流不存在大小的限制。但是由于缓冲区有大小限制，所以你如果用 TCP 发送一段很大的数据，可能会截断成好几段，接收方依次的接收。

Udp发送数据

由于 UDP 本身发送的就是一份一份的数据报，所以自然而然的就有一个上限的大小。

UDP协议本身，UDP协议中有16位的UDP报文长度，那么UDP报文长度不能超过2^16=65536；

**数据有序性**

tcp:

对于 TCP 来说，本身 TCP 有着超时重传、错误重传、还有等等一系列复杂的算法保证了 TCP 的数据是有序的，假设你发送了数据 1、2、3，则只要发送端和接收端保持连接时，接收端收到的数据始终都是 1、2、3。

Udp:

而 UDP 协议则要奔放的多，无论 server 端无论缓冲池的大小有多大，接收 client 端发来的消息总是一个一个的接收。并且由于 UDP 本身的不可靠性以及无序性，如果 client 发送了 1、2、3 这三个数据报过来，server 端接收到的可能是任意顺序、任意个数三个数据报的排列组合。

5、socket

网络上两个程序通过一个双向通信连接实现数据交互，这种双向通信的连接叫做Socket（套接字）。

Socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），通过Socket，我们才能使用TCP/IP协议

Socket可以支持不同的传输协议（TCP或UDP），当使用TCP协议进行连接时，该Socket连接就是一个TCP连接；同理，当使用UDP协议进行连接时，该Socket连接就是一个UDP连接。  
多个TCP连接或多个应用程序进程可能需要通过同一个TCP协议端口传输数据。为了区别不同的应用程序进程和连接，计算机操作系统为应用程序与TCP/IP协议交互提供了套接字（Socket）接口。应用层可以和传输层通过Socket接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

**Step 1 服务器监听**

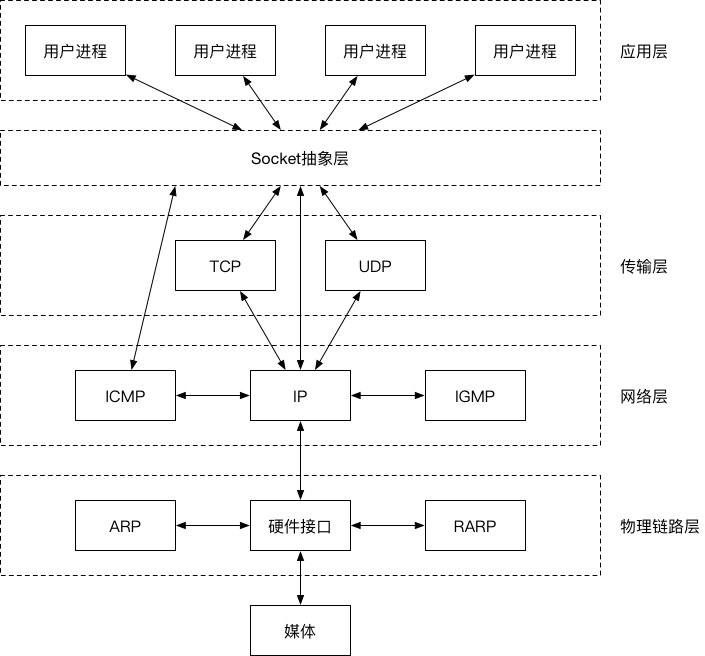
服务端Socket始终处于等待连接状态，实时监听是否有客户端请求连接。

**Step 2 客户端请求**

客户端Socket提出连接请求，指定服务端Socket的地址和端口号，这时就可以向对应的服务端提出Socket连接请求。

**Step 3 连接确认**

当服务端Socket监听到客户端Socket提出的连接请求时作出响应，建立一个新的进程，把服务端Socket的描述发送给客户端，该描述得到客户端确认后就可建立起Socket连接。而服务端Socket则继续处于监听状态，继续接收其他客户端Socket的请求。



## 基于第三方开源库[CocoaAsyncSocket](https://link.jianshu.com/?t=https%3A%2F%2Fgithub.com%2Frobbiehanson%2FCocoaAsyncSocket)

\_socket = [[GCDAsyncSocket alloc] initWithDelegate:self delegateQueue:dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0)];

[\_socket setUserData:command];

NSError \*error = nil;

[\_socket connectToHost:WIFI\_DIRECT\_HOST onPort:WIFI\_DIRECT\_PORT error:&error];

- (void)socket:(GCDAsyncSocket \*)sock didConnectToHost:(NSString \*)host port:(uint16\_t)port{

NSLog(@"Socket连接成功：%s",\_\_func\_\_);

}

6、即时通讯

**心跳机制**

**心跳就是用来检测TCP连接的双方是否可用。**

**们需要说明的是TCP的KeepAlive机制只能保证连接的存在，但是并不能保证客户端以及服务端的可用性；**

某台服务器因为某些原因导致负载超高，CPU 100%，无法响应任何业务请求，但是使用 TCP 探针则仍旧能够确定连接状态，这就是典型的连接活着但业务提供方已死的状态

 我们客户端发起心跳Ping（一般都是客户端），假如设置在10秒后如果没有收到回调，那么说明服务器或者客户端某一方出现问题，这时候我们需要主动断开连接。

 服务端也是一样，会维护一个socket的心跳间隔，当约定时间内，没有收到客户端发来的心跳，我们会知道该连接已经失效，然后主动断开连接。

**PingPong机制:**

当服务端发出一个Ping，客户端没有在约定的时间内返回响应的ack，则认为客户端已经不在线，这时我们Server端会主动断开Scoket连接，并且改由APNS推送的方式发送消息。

同样的是，当客户端去发送一个消息，因为我们迟迟无法收到服务端的响应ack包，则表明客户端或者服务端已不在线，我们也会显示消息发送失败，并且断开Scoket连接。

**重连机制:**

理论上，我们自己主动去断开的Scoket连接（例如退出账号，APP退出到后台等等），不需要重连。其他的连接断开，我们都需要进行断线重连。  
一般解决方案是尝试重连几次，如果仍旧无法重连成功，那么不再进行重连。