**WebScoket-SocketRocket框架的学习总结文档**

**一、概念**

1、Scoket

Socket 其实并不是一个协议。它工作在 OSI 模型会话层（第5层），是为了方便大家直接使用更底层协议（一般是TCP 或 UDP ）而存在的一个抽象层。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)。

Socket通常也称作”套接字”，用于描述IP地址和端口，是一个通信链的句柄。网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个Socket，一个Socket由一个IP地址和一个端口号唯一确定。应用程序通常通过”套接字”向网络发出请求或者应答网络请求。

Socket在通讯过程中，服务端监听某个端口是否有连接请求，客户端向服务端发送连接请求，服务端收到连接请求向客户端发出接收消息，这样一个连接就建立起来了。客户端和服务端也都可以相互发送消息与对方进行通讯，直到双方连接断开。

2、WebSocket

WebSocket协议是基于TCP的一种新的HTML5网络协议。它实现了浏览器与服务器全双工(即双方可同时向对方发送消息)、单套接字(TCP用主机的IP地址加上主机上的端口号作为TCP连接的端点)通信——允许服务器主动发送信息给客户端。

WebSocket 是为了满足基于 Web 的日益增长的实时通信需求而产生的。在传统的 Web 中，要实现实时通信（比如网页的QQ，聊天系统等），通用的方式是采用 HTTP 协议不断发送请求（按照以往的技术能力通常是采用轮询、Comet技术解决）。

HTTP协议是非持久化的，单向的网络协议，在建立连接后只允许浏览器向服务器发出请求后，服务器才能返回相应的数据。当需要即时通讯时，通过轮询在特定的时间间隔（如1秒），由浏览器向服务器发送Request请求，然后将最新的数据返回给浏览器。这样的方法最明显的缺点就是需要不断的发送请求，而且通常HTTP request的Header是非常长的，为了传输一个很小的数据 需要付出巨大的代价，是很不合算的，占用了很多的宽带。这种方式即浪费带宽（HTTP HEAD 是比较大的），又消耗服务器 CPU 占用（没有信息也要接受请求）。

然而WebSocket的出现可以弥补这一缺点。在WebSocket中，只需要服务端和浏览器通过HTTP协议进行一个握手的动作，然后单独建立一条TCP的通信通道进行数据传输。

**原理**

WebSocket同HTTP一样，也是应用层的协议，但是他是一种双向通信协议，是建立在TCP之上的。

**连接过程 —— 握手过程**

（1）浏览器和服务器建立TCP连接，这是基础；

（2）TCP连接成功后，浏览器通过HTTP协议向服务器发送WebSocket支持的版本号等信息。

（3）服务器收到客户端的握手请求后，同样采用HTTP协议

回馈数据。

（4）当收到连接成功的消息后，通过TCP通道进行数据传输。

**WebSocket与HTTP的关系**

相同点：

（1）都是基于TCP的，都是可靠性传输协议；

（2）都是应用层协议

不同点：

（1）WebSocket是双向通信协议，模拟Socket协议，可以双向发送或接收信息。HTTP是单向的。

（2）WebSocket是需要握手进行建立连接的。

注意：WebSocket在握手时，数据是通过HTTP协议传输的。但是建立连接后，真正传输数据时，是不需要HTTP协议的。

**Socket与WebSocket关系：**

相同点：

都能实现双向通信；

不同点：

（1）WebSocket是一种协议，而Socket是一组接口；

（2）WebSocket是应用层协议，而Socket是位于传输层与应用层之间的抽象层；

（3）WebSocket在传输数据之前需要进行握手操作，而Socket不需要。

**二、SocketRocket框架**

SocketRocket是对WebSocket的封装，利用Object-C编写，使用ARC模式，采用了并行架构。SocketRocket遵循最新的WebSocket规范[RFC 6455](http://tools.ietf.org/html/rfc6455)。支持TLS (wss)。使用NSStream(流)/CFNetworking大部分的工作由后端的工作队列（worker queues）完成。基于委托编程。SocketRocket支持iOS 4.x系统，不需要任何的UI包依赖。

1、导入框架

该框架并不开源，我们只能看到对外暴露出来的方法。使用cocopod可以快速导入该框架到项目中。

2、使用框架框架

1) 引入框架头文件#import <SocketRocket/SRWebSocket.h>

2）初始化框架并建立连接

self.socket = [[SRWebSocket alloc] initWithURLRequest: [NSURLRequest requestWithURL:[NSURL URLWithString:@"ws:xxxxxxxxxxx"]]];//这里填写你服务器的地址

3）遵循并指定代理

 self.socket.delegate = self;  //SRWebSocketDelegate 协议

4）打开连接加载请求，一个对象只能调用一次该方法，如果再调用open，就会报警告

 [self.socket open];    //开始连接

5）发送消息

调用-(void)sendData:(id)data 方法来发送消息。这个data可以是一个UTF8的字符串或者NSData对象。

6) 通过代理方法来获取接收到的消息

当接收到信息时，会通过- (void)webSocket:(SRWebSocket \*)webSocket didReceiveMessage:(id)message代理方法回调。

7）关闭连接

 [self.socket close];    //关闭连接

8）其他回调

//连接成功回调

-(void)webSocketDidOpen:(SRWebSocket \*)webSocket;

//连接失败回调

-(void)webSocket:(SRWebSocket \*)webSocket didFailWithError:(NSError \*)error;

//连接中断回调

-(void)webSocket:(SRWebSocket \*)webSocket didCloseWithCode:(NSInteger)code reason:(NSString \*)reason wasClean:(BOOL)wasClean;

-(void)webSocket:(SRWebSocket \*)webSocket didReceivePong:(NSData \*)pongPayload;

-(BOOL)webSocketShouldConvertTextFrameToString:(SRWebSocket \*)webSocket;

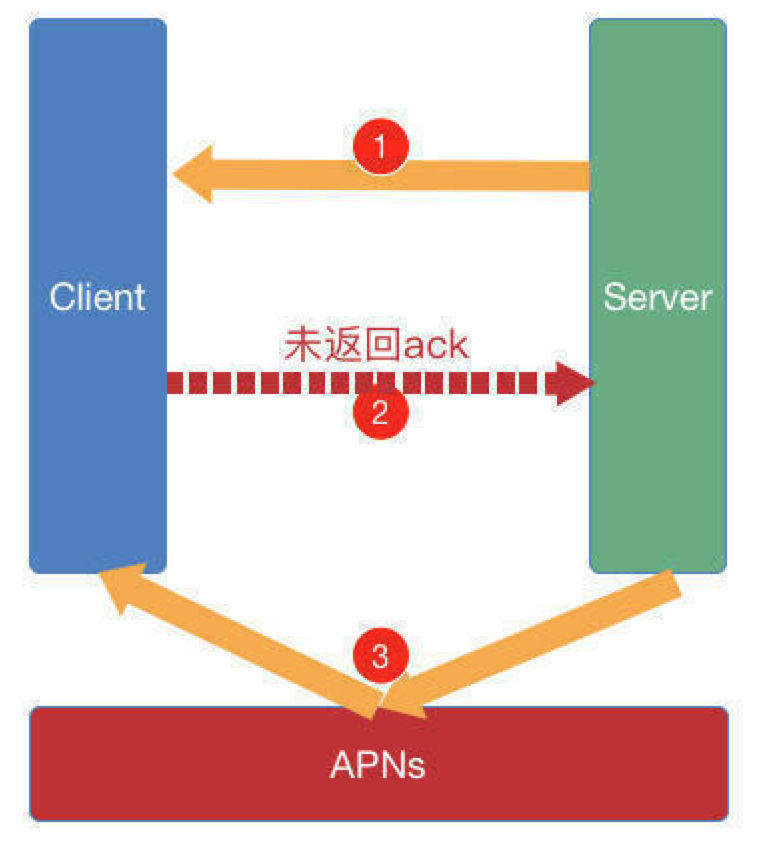
2、心跳的概念

简单的来说，心跳就是用来检测TCP连接的双方是否可用。那又会有人要问了，TCP不是本身就自带一个KeepAlive机制吗？  
 这里我们需要说明的是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.  
 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。  
 国内移动无线网络运营商在链路上一段时间内没有数据通讯后, 会淘汰NAT表中的对应项, 造成链路中断。而国内的运营商一般NAT超时的时间为5分钟，所以通常我们心跳设置的时间间隔为3-5分钟。

3、pingpong机制

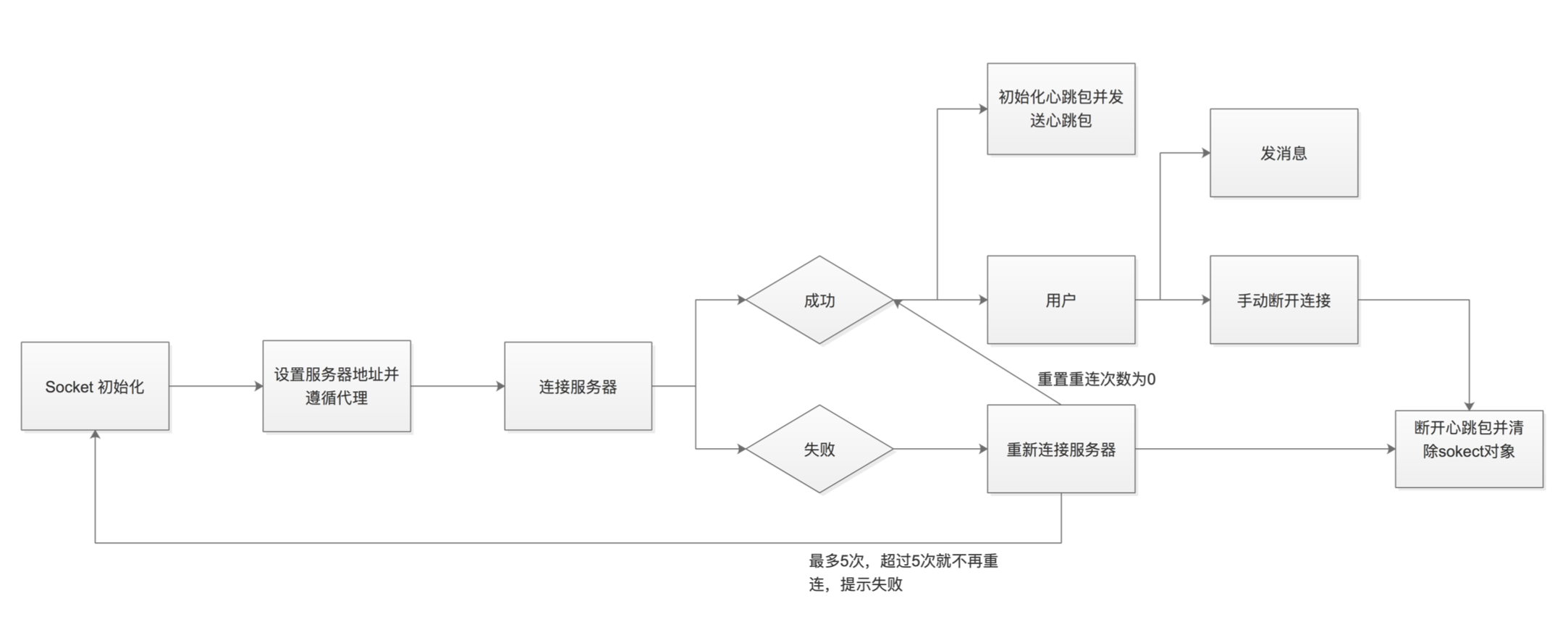
当服务端发出一个Ping，客户端没有在约定的时间内返回响应的ack，则认为客户端已经不在线，这时我们Server端会主动断开Scoket连接，并且改由APNS推送的方式发送消息。

  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。



4、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。



**5、粘包**

1）.概念

数据包的内容粘在了一起，顺序不对

2）出现原因：在流传输中出现，UDP不会出现粘包；发送端需要等缓冲区满才发送出去，造成粘包;接收方不及时接收缓冲区的包，造成多个包接收。

3）解决办法及问题

1.是对于发送方引起的粘包现象，用户可通过编程设置来避免，TCP提供了强制数据立即传送的操作指令push，TCP软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；

2.是对于接收方引起的粘包，则可通过优化程序设计、精简接收进程工作量、提高接收进程优先级等措施，使其及时接收数据，从而尽量避免出现粘包现象；

3.是由接收方控制，将一包数据按结构字段，人为控制分多次接收，然后合并，通过这种手段来避免粘包。

不足之处：

第一种编程设置方法虽然可以避免发送方引起的粘包，但它关闭了优化算法，降低了网络发送效率，影响应用程序的性能，一般不建议使用。

第二种方法只能减少出现粘包的可能性，但并不能完全避免粘包，当发送频率较高时，或由于网络突发可能使某个时间段数据包到达接收方较快，接收方还是有可能来不及接收，从而导致粘包。

第三种方法虽然避免了粘包，但应用程序的效率较低，对实时应用的场合不适合。

**6、封包**

封包就是给一段数据加上包头，这样一来数据包就分为包头和包体两部分内容了（以后讲过滤非法包时封包会加入"包尾"内容）。

包头其实上是个大小固定的结构体，其中有个结构体成员变量表示包体的长度，这是个很重要的变量，其他的结构体成员可根据需要自己定义。根据包头长度固定以及包头中含有包体长度的变量就能正确的拆分出一个完整的数据包。

**7、拆包**

在上面说到我们给每个数据包添加头部，头部中包含数据包的长度，这样接收到数据后，通过读取头部的长度字段，便知道每一个数据包的实际长度了，再根据长度去读取指定长度的数据便能获取到正确的数据了。

**8、断包**

我们为了防止粘包与断包,将每一条TCP数据都已\n结尾,这样当我们拿到\n的时候,就说明这是一条完整的数据.

如果一次接受的数据中包含两个或两个以上的\n的时候,则以\n拆分成两条或两条以上的完整数据进行解析,即粘包处理.

如果一次接受的数据中没有\n,则将这条数据与下次接受的数据合并起来,知道接受到\n为止,即断包处理.