iOS 基于GCDAsyncSocket快速开发Socket通信

2016年1月17日

GCDAsyncSocket是CocoaAsyncSocket第三方库中的其中一个类，本文介绍的就是基于这一个类来做快速的socket通信开发，而且该库已经支持IPv4和IPv6

我们对GCDAsyncSocket做了一层封装调用，它包含了建连、断开、重连、心跳、自定义请求

首先，介绍一下CocoaAsyncSocket第三方库的用途

> CocoaAsyncSocket provides easy-to-use and powerful asynchronous socket libraries for Mac and iOS.

翻译成：

> CocoaAsyncSocket为Mac和iOS提供了易于使用且强大的异步通信库

在Podfile文件中，只要加上这句话就可以使用了

```
pod 'CocoaAsyncSocket', '7.4.1'
```

简单的Socket通信包括了建连、断开连接、发送socket业务请求、重连这四个基本功能

下面，我就按照这个四个基本功能来讲一下，怎么来使用CocoaAsyncSocket中GCDAsyncSocket这个类来开发Socket通信

首先，Socket在第一步时，需要建连才能开始通信

在GCDAsyncSocket中提供了四种初始化的方法

```
- (id)init;
- (id)initWithSocketQueue:(dispatch_queue_t)sq;
- (id)initWithDelegate:(id)aDelegate delegateQueue:(dispatch_queue_t)dq;
- (id)initWithDelegate:(id)aDelegate delegateQueue:(dispatch_queue_t)dq socketQueue

@property (atomic, weak, readwrite) id delegate;
#if OS_OBJECT_USE_OBJC
@property (atomic, strong, readwrite) dispatch_queue_t delegateQueue;
#else
@property (atomic, assign, readwrite) dispatch_queue_t delegateQueue;
#endif
```

sq是socket的线程，这个是可选的设置，如果你写null，GCDAsyncSocket内部会帮你创建一个它自己的socket线程，如果你要自己提供一个socket线程的话，千万不要提供一个并发线程，在频繁socket通信过程中，可能会阻塞掉，个人建议是不用创建

aDelegate就是socket的代理

dq是delegate的线程

必须要需要设置socket的代理以及代理的线程，否则socket的回调你压根儿就不知道了，

比如：

```
self.socket =[[GCDAsyncSocket alloc] initWithDelegate:delegate delegateQueue:dispat
```

接着，在设置代理之后，你需要尝试连接到相应的地址来确定你的socket是否能连通了

```
- (BOOL)connectToHost:(NSString *)host
              onPort:(uint16_t)port
         withTimeout:(NSTimeInterval)timeout
               error:(NSError **)errPtr;
```

host是主机地址，port是端口号

如果建连成功之后，会收到socket成功的回调，在成功里面你可以做你需要做的一些事情，我这边的话，是做了心跳的处理

```
- (void)socket:(GCDAsyncSocket *)sock didConnectToHost:(NSString *)host port:(uint
```

如果建连失败了，会收到失败的回调，我这边在失败里面做了重连的操作

```
- (void)socketDidDisconnect:(GCDAsyncSocket *)sock withError:(NSError *)err;
```

重连操作其实比较简单，只需要再调用一次建连请求，我这边设置的重连规则是重连次数为5次，每次的时间间隔为2的n次方，超过次数之后，就不再去重连了

```
- (void)socketDidDisconnect:(GCDAsyncSocket*)sock withError:(NSError*)err {
 self.status= -1;
 if(self.reconnection_time>=0 && self.reconnection_time <= kMaxReconnection_
   [self.timer invalidate];
    self.timer=nil;
  int time =pow(2,self.reconnection_time);
  self.timer= [NSTimer scheduledTimerWithTimeInterval:time target:selfsele
   self.reconnection_time++;
  NSLog(@"socket did reconnection,after %ds try again",time);
  } else {
   self.reconnection_time=0;
  NSLog(@"socketDidDisconnect:%p withError: %@", sock, err);
        }
}
```

这里我用status来标记socket的连接状态

那么socket已经建连了，该怎么发起socket通信呢？

你需要和后端开发人员商定好socket协议格式，比如：

```
[NSString stringWithFormat:@"{\"version\":%d,\"reqType\":%d,\"body\":\"%@\"}\r\n",PROT
```

中间应该大家都能看得懂，那为什么后面需要加上\r\n呢？

这个\r\n是socket消息的边界符，是为了防止发生消息黏连，没有\r\n的话，可能由于某种原因，后端会收到两条socket请求，但是后端不知道怎么拆分这两个请求

同理，在收到socket请求回调时，也会根据这个边界符去拆分

那为什么要用\r\n呢？

而且GCDAsyncSocket不支持自定义边界符，它提供了四种边界符供你使用\r\n、\r、\n、空字符串

在拼装好socket请求之后，你需要调用GCDAsyncSocket的写方法，来发送请求，然后在写完成之后你会收到写的回调

```
[self.socket writeData:requestData withTimeout:-1 tag:0];
```

timeout是超时时间，这个根据实际的需要去设置

这个是写的回调

```
- (void)socket:(GCDAsyncSocket*)sock didWriteDataWithTag:(long)tag;
```

在写之后，需要再调用读方法，这样才能收到你发出请求后从服务器那边收到的数据

```
[self.socketreadDataToData:[GCDAsyncSocket CRLFData] withTimeout:10 maxLength:5000
```

[GCDAsyncSocket CRLFData]这里是设置边界符，maxLength是设置你收到的请求数据内容的最大值

在读回调里面，你可以根据不同业务来执行不同的操作

```
- (void)socket:(GCDAsyncSocket*)sock didReadData:(NSData*)data withTag:(long)tag;
```

最后一个则是断开连接，这个只需要调用

```
[self.socket disconnect];
```

ok，这样的话，最简单基础的socket通信，你已经大致能完成了~

2016.4.26更新

在网络环境以及其他因素下，很有可能会造成客户端或者后端没有接收到回调或者请求，那该怎么办？

我们需要加上消息回执的处理

客户端发出请求的时候，可以将该请求放到存到数组里面，等到后端的相应回调在移除掉，如果该请求超时或者在一段时间内没有收到确认返回，说明后端没有接收到我们的请求，我们可以将该请求重新发送

客户端接收请求的时候，后端将数据发给客户端，客户端需要增加回执处理，告诉后端，客户端接收到数据了，如果后端没接收到，也重新推一遍数据，客户端和后端双向保护来解决丢失问题

2016.8.5更新

有些时候，不能定位是否是后端问题还是客户端/SDK问题的时候，可以用命令行抓一下socket包看看（用Charles只能抓http和https包）

命令行如下：

```
sudo tcpdump -i any -n -X port 7070
```

Tip：7070端口号请根据实际的调试端口号修改

效果如下：

红色部分就是socket包的内容了

PREV NEXT