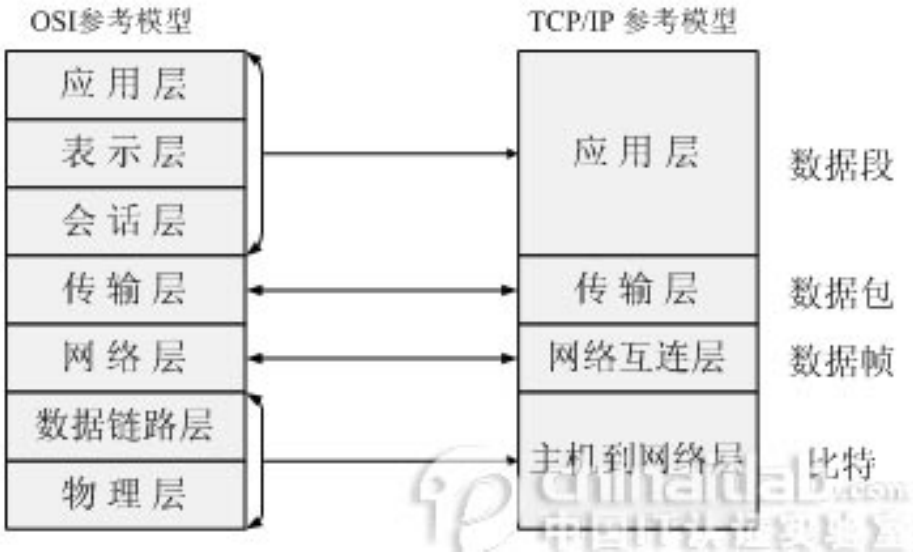
一.TCP/IP协议

1.TCP/IP四层网络模型



1.TCP/IP，UDP的基本概念

TCP/IP(Transmission Control Protocol/Internet Protocol)即传输控制协议/网间协议，它是一种流模式的协议。

作用:TCP 面向连接，传输可靠(保证数据正确性，保证数据顺序)。

TCP是面向连接的，也就是说，在连续持续的过程中，socket中收到的数据都是由同一台主机发出的(劫持什么的不考虑),因此，知道保证数据是有序的到达即可，至于每次读取多少数据，可以设置。

应用场景:

效率要求较低，但对准确性要求较高的场景。因为数据中需要对数据确认，重发，排序等操作，效率没有UDP效率高。举例:文件传输，收发邮件，远程登录

UDP(User Data Protocl)，即用户数据报协议，是与TCP相对应的协议，它属于TCP/IP协议族中的一部分。

作用: 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。

UDP是无连接的协议，也就是传输数据之间不用建立连接，不管对方是否真正收到数据，就连续发送数据包，所以会造成丢包，收到的数据包顺序不一致的情况。

应用场景:

效率要求相对较高，对准确性要求相对低的场景。举例：QQ聊天，在线视频，网络语音通话，广播通信(广播，多播)。

2.全双工，半双工，单工通信

全双工（Full Duplex）是指在发送数据的同时也能够接收数据，两者同步进行，这好像我们平时打电话一样，说话的同时也能够听到对方的声音。目前的网卡一般都支持全双工。

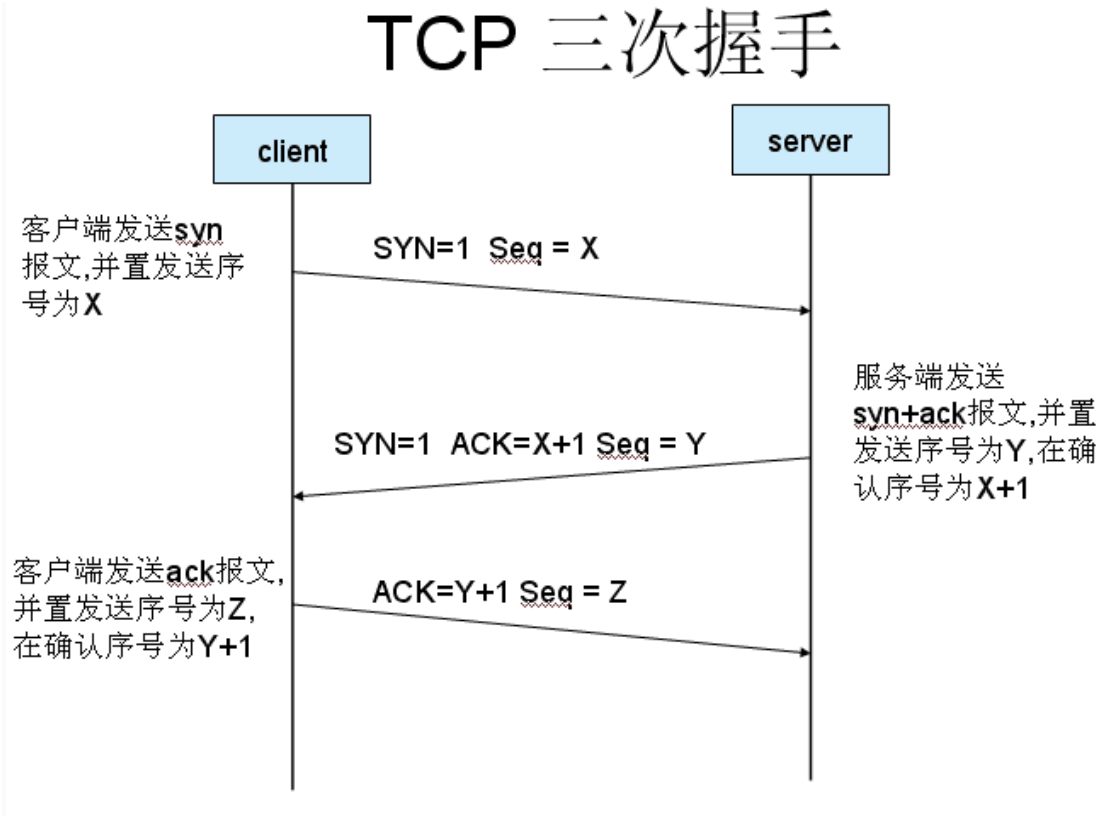
半双工（Half Duplex），所谓半双工就是指一个时间段内只有一个动作发生，举个简单例子，一条窄窄的马路，同时只能有一辆车通过，当目前有两量车对开，这种情况下就只能一辆先过，等到头儿后另一辆再开，这个例子就形象的说明了半双工的原理。早期的对讲机、以及早期集线器等设备都是基于半双工的产品。随着技术的不断进步，半双工会逐渐退出历史舞台.

单工通信是指通信线路上的数据按单一方向传送.

3.TCP三次握手

所谓三次握手(Three-way Handshake)，是指建立一个TCP连接时，需要客户端和服务器总共发送3个包。

三次握手的目的是连接服务器指定端口，建立TCP连接,并同步连接双方的序列号和确认号并交换 TCP 窗口大小信息.在socket编程中，客户端执行connect()时。将触发三次握手。



首先了解一下几个标志，SYN（synchronous），同步标志，ACK (Acknowledgement），即确认标志，seq应该是Sequence Number，序列号的意思，另外还有四次握手的fin，应该是final，表示结束标志。

第一次握手：客户端发送一个TCP的SYN标志位置1的包指明客户打算连接的服务器的端口，以及初始序号X,保存在包头的序列号(Sequence Number)字段里。

第二次握手：服务器发回确认包(ACK)应答。即SYN标志位和ACK标志位均为1同时，将确认序号(Acknowledgement Number)设置为客户的序列号加1以，即X+1。

第三次握手：客户端再次发送确认包(ACK) SYN标志位为0，ACK标志位为1。并且把服务器发来ACK的序号字段+1，放在确定字段中发送给对方.并且在数据段放写序列号的+1。

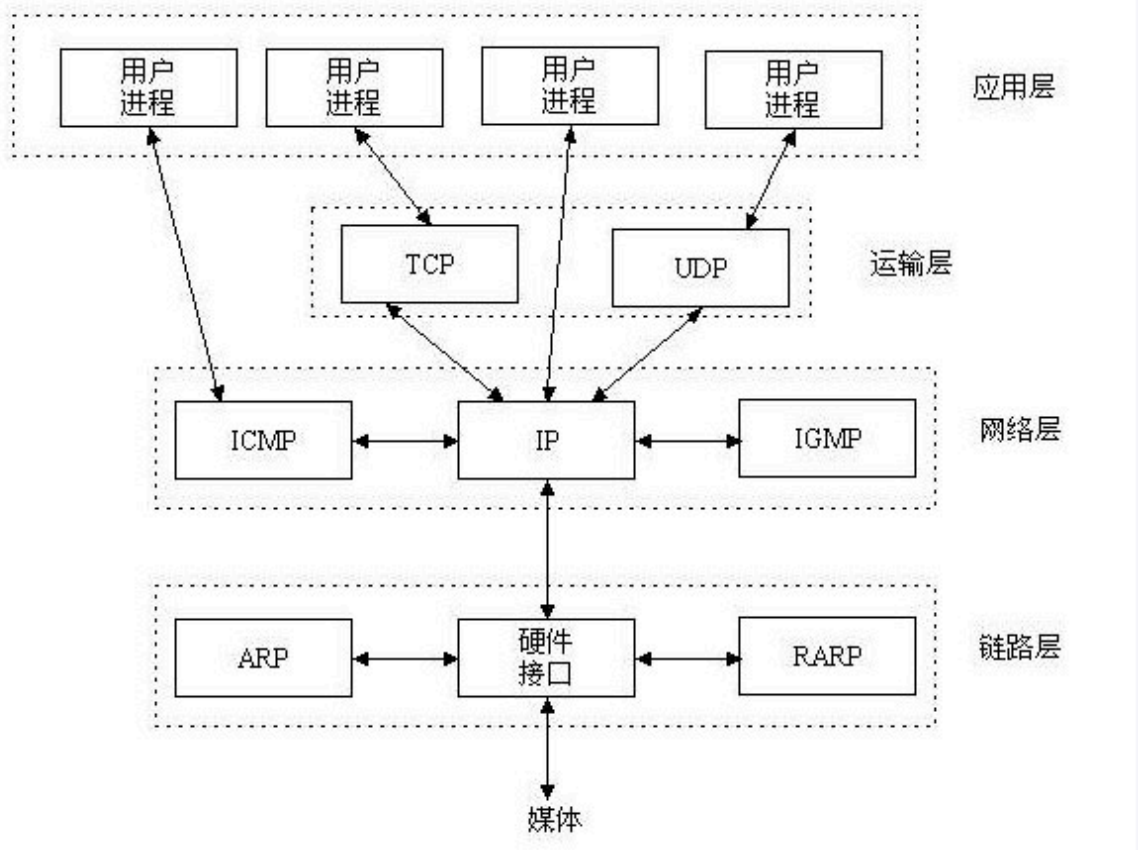
3.数据传输过程:发送端发送数据,然后进入等等ACK确定信号状态,随后接收端收到数据，发送ACK确认信号，发送端接收到ACK后才发送下一组数据，同时发送端有一个定时器，定时时间

到了没有接收到ACK,就认为发送失败，进行重新发送。因为发送端发送完数据后处于等待状态，因此，为了提供效率，引入”滑动窗口”概念，就是发送的时候一次发送多组数据，相当于窗口的大小，然后当接收到第一个ack后，就将窗口向后移动一个数据，就形成了滑动窗口的情况。

4.TCP四次挥手

TCP的的连接拆除(断开连接)需要发送四个数据包

5.各协议间的关系



6.tcp/ip的其他概念(重要)

(1)长连接和短连接

a.长连接:指在一个tcp连接上可以连续发送多个数据包，如果没有数据包发送，需要双方检测包以维持此连接，一般需要自己做在线维持。

操作过程:

连接–>数据传输–>保持连接(心跳包)–>数据传输–>保持连接(心跳)–>….–>关闭连接

应用场景:

长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况。

每个TCP连接都需要三部握手，这需要时间，如果每个操作都是先连接，

再操作的话，处理速度会降低很多，所以每个操作完成后不断开，下次处理时之间发送数据包就OK，不用建立TCP连接.

例如:数据库的连接用长连接

b.短连接:指通信双方有数据交互时，就建立以个tcp连接，数据发送完成后，就断开此tcp连接。

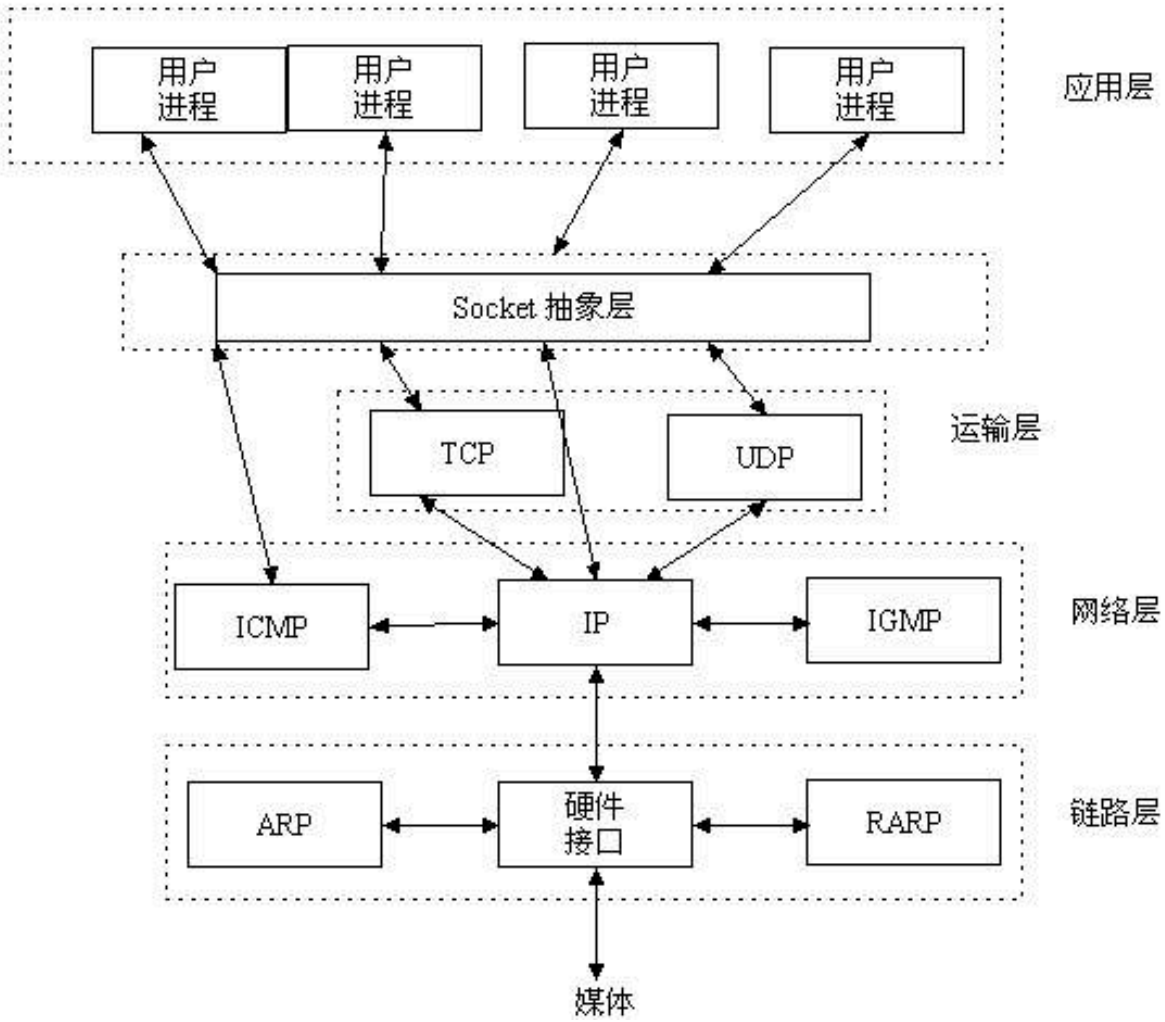
操作过程:

连接–>数据传输–>关闭连接

应用场景:

如果用短连接频繁的通信会造成socket错误，而且频繁的socket创建也是对资源的浪费，所以短连接用的地方比较少，常用的是长连接。

2.Socket的基本概念



(1).什么是socket?

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，他是一组接口，它把

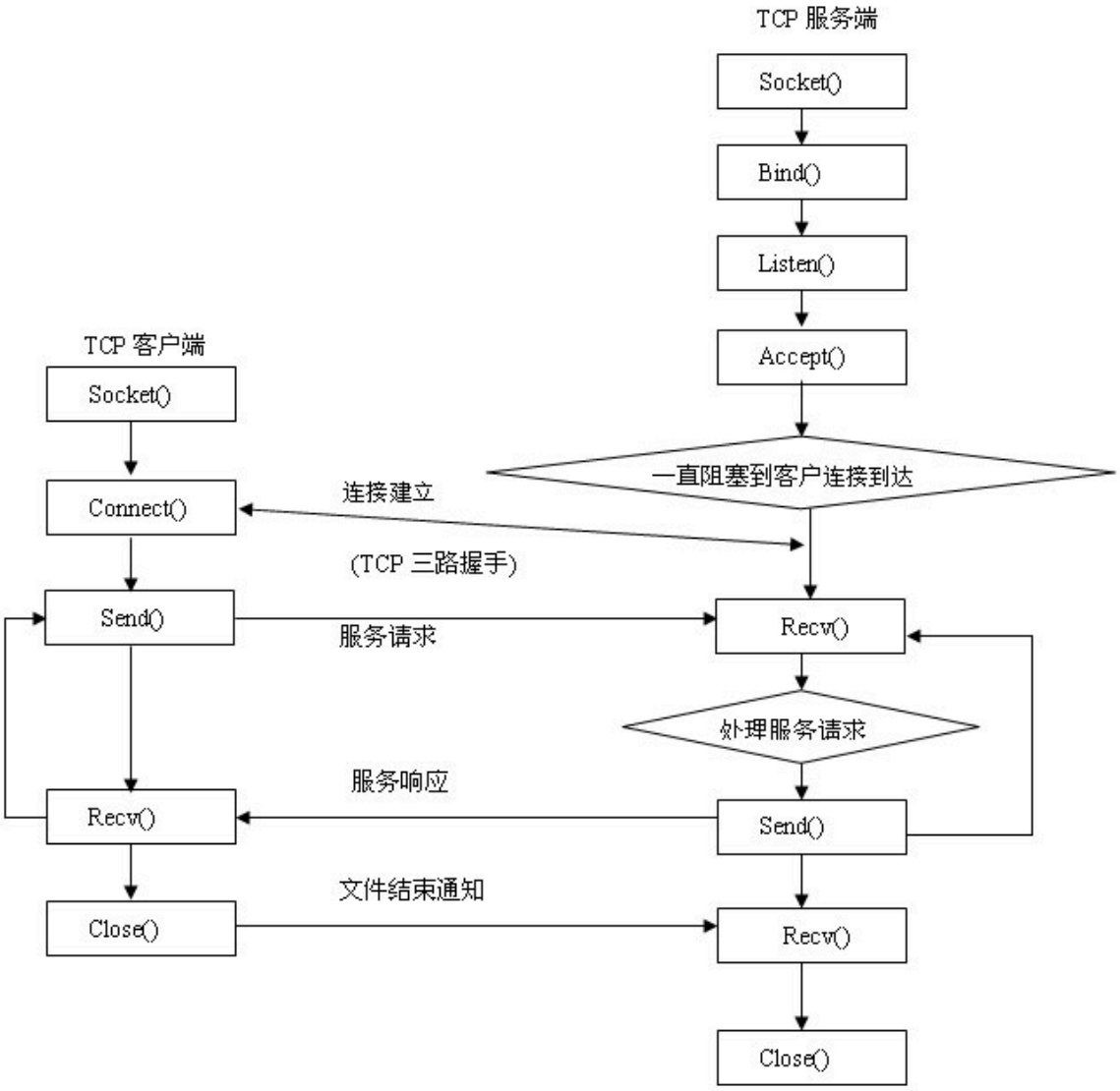
复杂的TCP/IP协议隐藏在Socket接口的后面，让Socket去组织数据，以符合指定的协议。

(2).Socket的使用(tcp/udp)

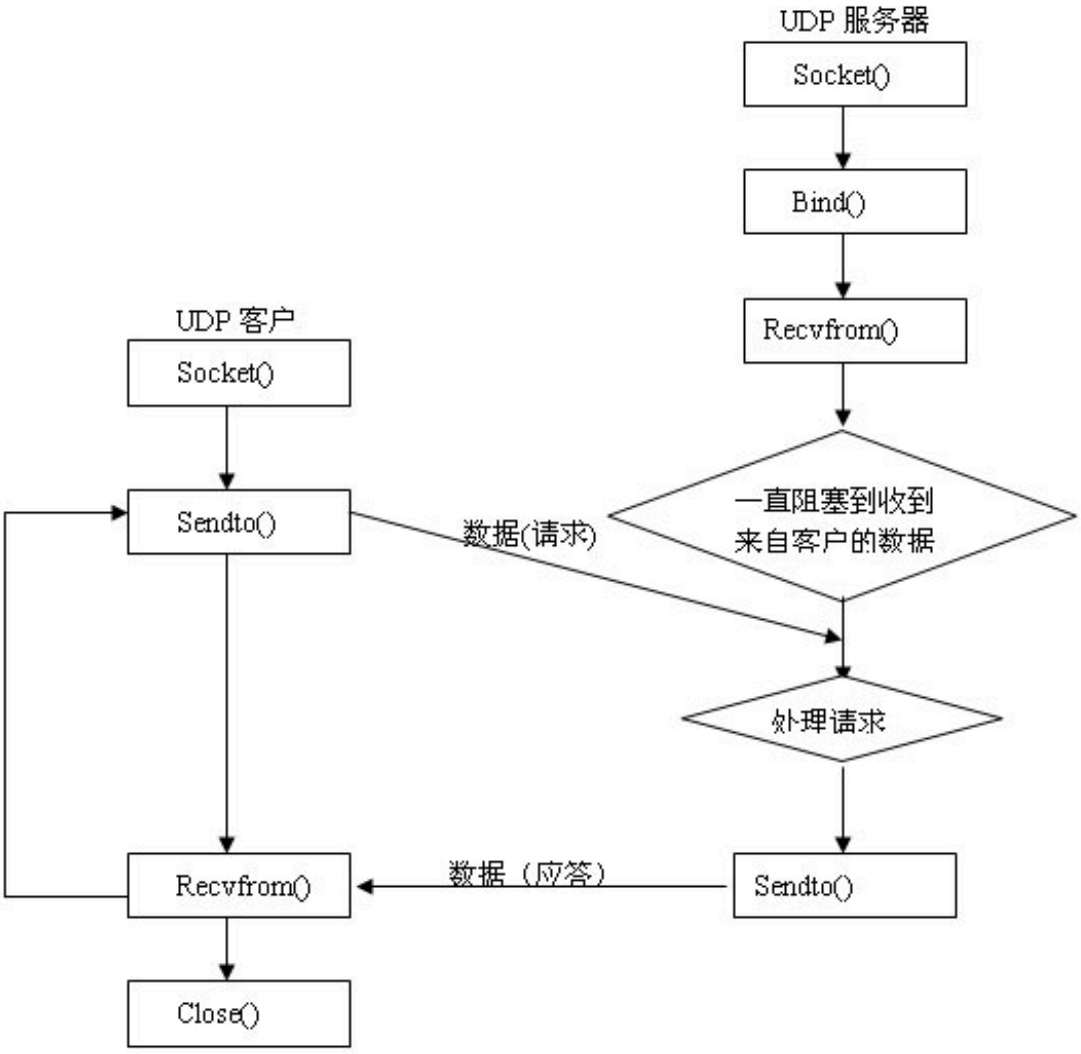
举例:要打电话给一个同学，听到手机铃声后就提取电话，这时你和你的同学就建立了一个连接，可以讲话了，等沟通交流几分钟后结束，挂断电话结束本次通话。这个生活场景的例子就解释了TCP/IP的工作原理和流程

tcp和udp的socket是有区别的，这里给出这两种的设计框架

基于TCP-服务器程序设计基本框架



基本UDP客户一服务器程序设计基本框架



常用的Socket类型有两种：流式Socket（SOCK_STREAM）和数据报式Socket（SOCK_DGRAM）。流式是一种面向连接的Socket，针对于面向连接的TCP服务应用；数据报式Socket是一种无连接的Socket，对应于无连接的UDP服务应用。

3.CocoaAsyncSocket(第三方框架) 实现 Socket

1.socket服务器:

检测服务器端口通讯:

tcp: `nc -lk 6666(端口号)`

`udp:` `nc -ul 6665(端口号)`

2.tcp/udp客户端代码:

pod 'CocoaAsyncSocket'

tcp:


```
#import "GCDAsyncSocket.h" //
for TCP

@interface ViewController :
UIViewController<GCDAsyncSocket
Delegate>
{
    IBOutlet UITextField
*_ipTF;
    IBOutlet UITextField
*_portTF;
    IBOutlet UITextField
*_contentTF;
```

```
    GCDAsyncSocket
*_syncSocket;//创建一socket
}
-(IBAction) connectBtn:
(id)sender;
-(IBAction) sendBtn:(id)
sender;
@end


//连接
-(IBAction) connectBtn:
(id)sender{
    NSString *strIp=_ipTF.text;
    NSInteger
nPort=[_portTF.text intValue];


    _syncSocket =
[[GCDAsyncSocket alloc]
initWithDelegate:self
delegateQueue:dispatch_get_main
_queue()];
```

```
    NSError *err = nil;
    if (![_syncSocket
connectToHost:strIp
onPort:nPort error:&err]) //
Asynchronous!
    {
        // If there was an
error, it's likely something
like "already connected" or "no
delegate set"
        NSLog(@"I goofed: %@",
err);
    }
}
//发送数据
-(IBAction) sendBtn:(id)
sender{

    // At this point the socket
is NOT connected.
    // But I can start writing
to it anyway!
    // The library will queue
all my write operations,
    // and after the socket
```

```
connects, it will automatically
start executing my writes!
    NSData
*dData=[_contentTF.text
dataUsingEncoding:NSUTF8StringE
ncoding];
    [_syncSocket
writeData:dData withTimeout:-1
tag:1];
}
//已连接到主机
- (void)socket:(GCDAsyncSocket
*)sender didConnectToHost:
(NSString *)host port:
(UInt16)port
{
    NSLog(@"连接成功");
}
//数据发送到服务器成功
- (void)socket:(GCDAsyncSocket
*)sock didWriteDataWithTag:
(long)tag
{
    NSLog(@"数据已发送至服务
```

```
器%ld",tag);
}
//读取从服务器发来的数据
- (void)socket:(GCDAsyncSocket
*)sender didReadData:(NSData
*)data withTag:(long)tag
{
    NSLog(@"从服务器读取数据");
    NSString* message =
[[NSString alloc]
initWithData:data
encoding:NSUTF8StringEncoding];
    NSLog(@"message is: \n
%@",message);

}
```

2. UDP

```
    #import
"GCDAsyncUdpSocket.h" // for
UDP

@interface ViewController :
UIViewController<GCDAsyncUdpSoc
```

```
ketDelegate>
{
    IBOutlet UITextField
*_ipTF;
    IBOutlet UITextField
*_portTF;
    IBOutlet UITextField
*_contentTF;

    GCDAsyncUdpSocket
*_syncSocket;
}
-(IBAction) connectBtn:
(id)sender;
-(IBAction) sendBtn:(id)
sender;
@end

//连接
-(IBAction) connectBtn:
(id)sender{
    NSString *strIp=_ipTF.text;
    NSInteger
nPort=[_portTF.text intValue];
```

```
    _syncSocket =
[[GCDAsyncUdpSocket alloc]
initWithDelegate:self
delegateQueue:dispatch_get_main
_queue()];

    NSError *err = nil;
    if (![_syncSocket
connectToHost:strIp
onPort:nPort error:&err]) //
Asynchronous!
    {
        // If there was an
error, it's likely something
like "already connected" or "no
delegate set"
        NSLog(@"I goofed: %@",
err);
    }
}
//发送数据
-(IBAction) sendBtn:(id)
sender{
```

```
    // At this point the socket
is NOT connected.
    // But I can start writing
to it anyway!
    // The library will queue
all my write operations,
    // and after the socket
connects, it will automatically
start executing my writes!
    NSData
*dData=[_contentTF.text
dataUsingEncoding:NSUTF8StringE
ncoding];
//    [_syncSocket
writeData:dData withTimeout:-1
tag:1];


    [_syncSocket sendData:dData
withTimeout:-1 tag:1];
}

- (void)socket:
(GCDAsyncUdpSocket *)sender
didConnectToHost:(NSString
```

```
*)host port:(UInt16)port
{
    NSLog(@"连接成功");
}

- (void)socket:
(GCDAsyncUdpSocket *)sock
didWriteDataWithTag:(long)tag
{
    NSLog(@"数据已发送至服务
器%ld",tag);
}
- (void)socket:
(GCDAsyncUdpSocket *)sender
didReadData:(NSData *)data
withTag:(long)tag
{
    NSLog(@"从服务器读取数据");
    NSString* message =
[[NSString alloc]
initWithData:data
encoding:NSUTF8StringEncoding];
    NSLog(@"message is: \n
%@",message);
```

```
}
```

扩展知识:
蓝牙: BabyBluetooth　https://github.com/coolnameismy/BabyBluetooth