**软件复用讨论课**

**徐锦程 1353012**

**1.长连接与心跳机制**

长连接，指在一个连接上可以连续发送多个数据包，在连接保持期间，如果没有数据包发送，需要双方发链路检测包。短连接是指通讯双方有数据交互时，就建立一个连接，数据发送完成后，则断开此连接，即每次连接只完成一项业务的发送。

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连接都需要三步握手，这需要时间，如果每个操作都是先连接，再操作的话那么处理速度会降低很多，所以每个操作完后都不断开，下次处理时直接发送数据包就OK了，不用建立TCP连接。例如：数据库的连接用长连接，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。

而像WEB网站的http服务一般都用短链接，因为长连接对于服务端来说会耗费一定的资源，而像WEB网站这么频繁的成千上万甚至上亿客户端的连接用短连接会更省一些资源，如果用长连接，而且同时有成千上万的用户，如果每个用户都占用一个连接的话，那可想而知吧。所以并发量大，但每个用户无需频繁操作情况下需用短链接好。

心跳机制是定时发送一个自定义的结构体([心跳包](http://baike.so.com/doc/875746-925736.html))，让对方知道自己还活着，以确保连接的有效性的机制。

网络中的接收和发送数据都是使用操作系统中的SOCKET进行实现。但是如果此[套接字](http://baike.so.com/doc/383276-405861.html)已经断开，那发送数据和接收数据的时候就一定会有问题。可是如何判断这个[套接字](http://baike.so.com/doc/383276-405861.html)是否还可以使用呢?这个就需要在系统中创建心跳机制。其实TCP中已经为我们实现了一个叫做心跳的机制。如果你设置了心跳，那TCP就会在一定的时间(比如你设置的是3秒钟)内发送你设置的次数的心跳(比如说2次)，并且此信息不会影响你自己定义的协议。所谓"心跳"就是定时发送一个自定义的结构体([心跳包](http://baike.so.com/doc/875746-925736.html)或心跳帧)，让对方知道自己"在线"。 以确保链接的有效性。

所谓的[心跳包](http://baike.so.com/doc/875746-925736.html)就是客户端定时发送简单的信息给服务器端告诉它我还在而已。代码就是每隔几分钟发送一个固定信息给[服务端](http://baike.so.com/doc/5988965-6201932.html)，服务端收到后回复一个固定信息如果服务端几分钟内没有收到客户端信息则视客户端断开。比如有些通信软件长时间不使用，要想知道它的状态是在线还是离线就需要[心跳包](http://baike.so.com/doc/875746-925736.html)，定时发包收包。发包方:可以是客户也可以是[服务端](http://baike.so.com/doc/5988965-6201932.html)，看哪边实现方便合理。一般是客户端。服务器也可以定时轮询发心跳下去。[心跳包](http://baike.so.com/doc/875746-925736.html)之所以叫心跳包是因为:它像心跳一样每隔固定时间发一次，以此来告诉服务器，这个客户端还活着。事实上这是为了保持[长连接](http://baike.so.com/doc/7562182-7836275.html)，至于这个包的内容，是没有什么特别规定的，不过一般都是很小的包，或者只包含包头的一个空包。

**2.socket通信的讨论**

一个包没有固定长度，以太网限制在46－1500字节，1500就是以太网的MTU，超过这个量，TCP会为IP数据报设置偏移量进行分片传输，现在一般可允许应用层设置8k（NTFS系统）的缓冲区，8k的数据由底层分片，而应用层看来只是一次发送。  
        windows的缓冲区经验值是4k。  
        Socket本身分为两种，流(TCP)和数据报(UDP)，你的问题针对这两种不同使用而结论不一样。甚至还和你是用阻塞、还是非阻塞Socket来编程有关。  
        1、通信长度，这个是你自己决定的，没有系统强迫你要发多大的包，实际应该根据需求和网络状况来决定。对于TCP，这个长度可以大点，但要知道，Socket内部默认的收发缓冲区大小大概是8K，你可以用SetSockOpt来改变。但对于UDP，就不要太大，一般在1024至10K。注意一点，你无论发多大的包，IP层和链路层都会把你的包进行分片发送，一般局域网就是1500左右，广域网就只有几十字节。分片后的包将经过不同的路由到达接收方，对于UDP而言，要是其中一个分片丢失，那么接收方的IP层将把整个发送包丢弃，这就形成丢包。显然，要是一个UDP发包佷大，它被分片后，链路层丢失分片的几率就佷大，你这个UDP包，就佷容易丢失，但是太小又影响效率。最好可以配置这个值，以根据不同的环境来调整到最佳状态。  
        send()函数返回了实际发送的长度，在网络不断的情况下，它绝不会返回(发送失败的)错误，最多就是返回0。对于TCP你可以写一个循环发送。当send函数返回SOCKET\_ERROR时，才标志着有错误。但对于UDP，你不要写循环发送，否则将给你的接收带来极大的麻烦。所以UDP需要用SetSockOpt来改变Socket内部Buffer的大小，以能容纳你的发包。明确一点，TCP作为流，发包是不会整包到达的，而是源源不断的到，那接收方就必须组包。而UDP作为消息或数据报，它一定是整包到达接收方。  
        2、关于接收，一般的发包都有包边界，首要的就是你这个包的长度要让接收方知道，于是就有个包头信息，对于TCP，接收方先收这个包头信息，然后再收包数据。一次收齐整个包也可以，可要对结果是否收齐进行验证。这也就完成了组包过程。UDP，那你只能整包接收了。要是你提供的接收Buffer过小，TCP将返回实际接收的长度，余下的还可以收，而UDP不同的是，余下的数据被丢弃并返回WSAEMSGSIZE错误。注意TCP，要是你提供的Buffer佷大，那么可能收到的就是多个发包，你必须分离它们，还有就是当Buffer太小，而一次收不完Socket内部的数据，那么Socket接收事件(OnReceive)，可能不会再触发，使用事件方式进行接收时，密切注意这点。这些特性就是体现了流和数据包的区别。  
        补充一点，接收BuffSize >= 发送BuffSize >= 实际发送Size，对于内外部的Buffer都适用，上面讲的主要是Socket内部的Buffer大小关系。  
        3、TCP是有多少就收多少，如果没有当然阻塞Socket的recv就会等，直到有数据，非阻塞Socket不好等，而是返回WSAEWOULDBLOCK。UDP，如果没有数据，阻塞Socket就会等，非阻塞Socket也返回WSAEWOULDBLOCK。如果有数据，它是会等整个发包到齐，并接收到整个发包，才返回

**3.消息压缩**

用Java Socket来传输对象，在有些情况下比如网络环境不好或者对象比较大的情况下需要把数据对象进行压缩然后在传输，此时就需要压缩这些对象流，此时就可以GZIPInputStream和GZIPOutputStream来处理一下socket的InputStream和OutputStream。

此时仍然需要一个实现了java.io.Serializable接口的简单Java对象,在Server端使用，socket的InputStream首先被包装成GZIPInputStream，然后又被包装成ObjectInputStream，而socket的OutputStream首先被包装成GZIPOutputStream，然后又被包装成ObjectOutputStream.

Client也和Server端类似，同样要不socket的XXXStream包装成GZIPXXXStream，然后再包装成ObjectXXXStream

最后测试上面的代码，首先运行Server类，然后运行Client类，就可以分别在Server端和Client端控制台看到接收到的User对象实例了。

**4.socket保持长连接的讨论**

主要分为两部分：  
一，如何更好的检测TCP连接是否正常  
二，如何提取本机TCP连接状态  
  
一，如何更好的检测TCP连接是否正常

这方面问题，一般来说比较成熟的有两种方法：

1、是在应用层制定协议，发心跳包，这也是C#，JAVA等高级语言比较常用的方法。客户端和服务端制定一个通讯协议，每隔一定时间（一般15秒左右），由一方发起，向对方发送协议包；对方收到这个包后，按指定好的通讯协议回一个。若没收到回复，则判断网络出现问题，服务器可及时的断开连接，客户端也可以及时重连。  
 2、通过TCP协议层发送KeepAlive包。这个方法只需设置好你使用的TCP的KeepAlive项就好，其他的操作系统会帮你完成。操作系统会按时发送KeepAlive包，一发现网络异常，马上断开。  
  
 使用第二种方法的好处，是我们在应用层不需自己定协议，通信的两端，只要有一端设好这个值，两边都能及时检测出TCP连接情况。而且这些都是操作系统帮你自动完成的。像我们公司的服务端代码就是早写好的，很难改动。以前也没加入心跳机制，后面要改很麻烦，boss要求检测连接的工作尽量客户端单独完成....  
还有一个好处就是节省网络资源。KeepAlive包，只有很简单的一些TCP信息，无论如何也是比你自己设计的心跳包短小的。然后就是它的发送机制，在TCP空闲XXX秒后才开始发送。自己设计心跳机制的话，很难做到这一点。  
  
 这种方法也是有些缺陷的。比如某一时刻，网线松了，如果刚好被KeepAlive包检测到，它会马上断开TCP连接。但其实这时候TCP连接也算是established的，只要网线再插好，这个连接还是可以正常工作的。这种情况，大家有什么好办法处理吗？  
  
C#中设置KeepAlive的代码  
uint dummy = 0;  
byte[] inOptionValues = new byte[Marshal.SizeOf(dummy) \* 3];  
BitConverter.GetBytes((uint)1).CopyTo(inOptionValues, 0);  
BitConverter.GetBytes((uint)15000).CopyTo(inOptionValues, Marshal.SizeOf(dummy));  
BitConverter.GetBytes((uint)15000).CopyTo(inOptionValues, Marshal.SizeOf(dummy) \* 2);  
  
IPEndPoint iep = new IPEndPoint(this.\_IPadd, xxxx);  
this.\_socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  
this.\_socket.IOControl(IOControlCode.KeepAliveValues, inOptionValues, null);  
this.\_socket.Connect(iep);  
这里我设定TCP15秒钟空闲，就开始发送KeepAlive包，其实完全可是设定得长一点。  
  
  
二，如何提取本机TCP连接状态

设好了KeepAlive值，又遇到麻烦了，我没找到当网络异常时，它断开连接后怎么通知我...我搜了很久都没找到，要是哪位兄弟知道的话告诉我吧。我是使用笨办法的，找到所有本地TCP连接的信息，筛选出我需要的那个TCP。  
查看本机所有TCP连接信息，网上一般的方法，都是通过程序调用CMD命令里的netstat进行，然后再分析其内容。但在CMD窗口用过这个命令的都知道，悲剧的时候它显示完所有TCP信息需要15s，或者更长时间，这在我的程序中是不能忍受的。

有人提到用iphlpapi.dll。这是一个在win98以上操作系统目录System32都包含的库函数，功能异常强大，大家可以放心使用！但是使用起来比较麻烦，基本找不到C#现成使用的例子，就算有，也是很老版本的，完全不能用。  
下载了里面提到的项目，仔细结合自己体会进行修改，终于能用了。每隔一段时间，我的客户端就用这个方法扫描一遍本地TCP信息，若发现连接有问题，则断开重连。  
这个方法能瞬间得到本机所有TCP连接信息（如果你有兴趣可以扩充，它的功能真的是太强大了），没有CMD命令netstat那不能忍受的延迟，相当好用。