<https://blog.csdn.net/zhangxinrun/article/details/6721495>

这两天看csdn有一些关于socket粘包，socket缓冲区设置的问题，发现自己不是很清楚，所以查资料了解记录一下：

一 .两个简单概念长连接与短连接：  
1.长连接

    Client方与Server方先建立通讯连接，连接建立后不断开， 然后再进行报文发送和接收。

2.短连接

    Client方与Server每进行一次报文收发交易时才进行通讯连接，交易完毕后立即断开连接。此种方式常用于一点对多点   
通讯，比如多个Client连接一个Server.

二 .什么时候需要考虑粘包问题?

1:如果利用tcp每次发送数据，就与对方建立连接，然后双方发送完一段数据后，就关闭连接，这样就不会出现粘包问题（因为只有一种包结构,类似于http协议）。关闭连接主要要双方都发送close连接（参考tcp关闭协议）。如：A需要发送一段字符串给B，那么A与B建立连接，然后发送双方都默认好的协议字符如"hello give me sth abour yourself"，然后B收到报文后，就将缓冲区数据接收,然后关闭连接，这样粘包问题不用考虑到，因为大家都知道是发送一段字符。  
2：如果发送数据无结构，如文件传输，这样发送方只管发送，接收方只管接收存储就ok，也不用考虑粘包  
3：如果双方建立连接，需要在连接后一段时间内发送不同结构数据，如连接后，有好几种结构：  
 1)"hello give me sth abour yourself"   
 2)"Don't give me sth abour yourself"   
   那这样的话，如果发送方连续发送这个两个包出去，接收方一次接收可能会是"hello give me sth abour yourselfDon't give me sth abour yourself" 这样接收方就傻了，到底是要干嘛？不知道，因为协议没有规定这么诡异的字符串，所以要处理把它分包，怎么分也需要双方组织一个比较好的包结构，所以一般可能会在头加一个数据长度之类的包，以确保接收。

三 .粘包出现原因：在流传输中出现，UDP不会出现粘包，因为它有消息边界(参考Windows 网络编程)  
1 发送端需要等缓冲区满才发送出去，造成粘包  
2 接收方不及时接收缓冲区的包，造成多个包接收

解决办法：  
为了避免粘包现象，可采取以下几种措施。一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；二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；三是由接收方控制，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。

以上提到的三种措施，都有其不足之处。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。  
载自：http://blog.csdn.net/binghuazh/archive/2009/05/28/4222516.aspx  
====================================================================

网络通讯的封包和拆包 收藏

对于基于TCP开发的通讯程序,有个很重要的问题需要解决,就是封包和拆包.

一.为什么基于TCP的通讯程序需要进行封包和拆包.

TCP是个"流"协议,所谓流,就是没有界限的一串数据.大家可以想想河里的流水,是连成一片的,其间是没有分界线的.但一般通讯程序开发是需要定义一个个相互独立的数据包的,比如用于登陆的数据包,用于注销的数据包.由于TCP"流"的特性以及网络状况,在进行数据传输时会出现以下几种情况.  
假设我们连续调用两次send分别发送两段数据data1和data2,在接收端有以下几种接收情况(当然不止这几种情况,这里只列出了有代表性的情况).  
A.先接收到data1,然后接收到data2.  
B.先接收到data1的部分数据,然后接收到data1余下的部分以及data2的全部.  
C.先接收到了data1的全部数据和data2的部分数据,然后接收到了data2的余下的数据.  
D.一次性接收到了data1和data2的全部数据.

对于A这种情况正是我们需要的,不再做讨论.对于B,C,D的情况就是大家经常说的"粘包",就需要我们把接收到的数据进行拆包,拆成一个个独立的数据包.为了拆包就必须在发送端进行封包.

另:对于UDP来说就不存在拆包的问题,因为UDP是个"数据包"协议,也就是两段数据间是有界限的,在接收端要么接收不到数据要么就是接收一个完整的一段数据,不会少接收也不会多接收.

二.为什么会出现B.C.D的情况.  
"粘包"可发生在发送端也可发生在接收端.  
1.由Nagle算法造成的发送端的粘包:Nagle算法是一种改善网络传输效率的算法.简单的说,当我们提交一段数据给TCP发送时,TCP并不立刻发送此段数据,而是等待一小段时间,看看在等待期间是否还有要发送的数据,若有则会一次把这两段数据发送出去.这是对Nagle算法一个简单的解释,详细的请看相关书籍.象C和D的情况就有可能是Nagle算法造成的.  
2.接收端接收不及时造成的接收端粘包:TCP会把接收到的数据存在自己的缓冲区中,然后通知应用层取数据.当应用层由于某些原因不能及时的把TCP的数据取出来,就会造成TCP缓冲区中存放了几段数据.

三.怎样封包和拆包.  
   最初遇到"粘包"的问题时,我是通过在两次send之间调用sleep来休眠一小段时间来解决.这个解决方法的缺点是显而易见的,使传输效率大大降低,而且也并不可靠.后来就是通过应答的方式来解决,尽管在大多数时候是可行的,但是不能解决象B的那种情况,而且采用应答方式增加了通讯量,加重了网络负荷. 再后来就是对数据包进行封包和拆包的操作.  
    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.  
    对于拆包目前我最常用的是以下两种方式.  
    1.动态缓冲区暂存方式.之所以说缓冲区是动态的是因为当需要缓冲的数据长度超出缓冲区的长度时会增大缓冲区长度.  
    大概过程描述如下:  
    A,为每一个连接动态分配一个缓冲区,同时把此缓冲区和SOCKET关联,常用的是通过结构体关联.  
    B,当接收到数据时首先把此段数据存放在缓冲区中.  
    C,判断缓存区中的数据长度是否够一个包头的长度,如不够,则不进行拆包操作.  
    D,根据包头数据解析出里面代表包体长度的变量.  
    E,判断缓存区中除包头外的数据长度是否够一个包体的长度,如不够,则不进行拆包操作.  
    F,取出整个数据包.这里的"取"的意思是不光从缓冲区中拷贝出数据包,而且要把此数据包从缓存区中删除掉.删除的办法就是把此包后面的数据移动到缓冲区的起始地址.

这种方法有两个缺点.1.为每个连接动态分配一个缓冲区增大了内存的使用.2.有三个地方需要拷贝数据,一个地方是把数据存放在缓冲区,一个地方是把完整的数据包从缓冲区取出来,一个地方是把数据包从缓冲区中删除.第二种拆包的方法会解决和完善这些缺点.

前面提到过这种方法的缺点.下面给出一个改进办法, 即采用环形缓冲.但是这种改进方法还是不能解决第一个缺点以及第一个数据拷贝,只能解决第三个地方的数据拷贝(这个地方是拷贝数据最多的地方).第2种拆包方式会解决这两个问题.  
环形缓冲实现方案是定义两个指针,分别指向有效数据的头和尾.在存放数据和删除数据时只是进行头尾指针的移动.

2.利用底层的缓冲区来进行拆包  
由于TCP也维护了一个缓冲区,所以我们完全可以利用TCP的缓冲区来缓存我们的数据,这样一来就不需要为每一个连接分配一个缓冲区了.另一方面我们知道recv或者wsarecv都有一个参数,用来表示我们要接收多长长度的数据.利用这两个条件我们就可以对第一种方法进行优化.  
     对于阻塞SOCKET来说,我们可以利用一个循环来接收包头长度的数据,然后解析出代表包体长度的那个变量,再用一个循环来接收包体长度的数据.  
相关代码如下:  
    
char PackageHead[1024];  
char PackageContext[1024\*20];

int len;  
PACKAGE\_HEAD \*pPackageHead;  
while( m\_bClose == false )  
{  
memset(PackageHead,0,sizeof(PACKAGE\_HEAD));  
len = m\_TcpSock.ReceiveSize((char\*)PackageHead,sizeof(PACKAGE\_HEAD));  
if( len == SOCKET\_ERROR )  
{  
    break;  
}  
if(len == 0)  
{  
    break;  
}  
pPackageHead = (PACKAGE\_HEAD \*)PackageHead;  
memset(PackageContext,0,sizeof(PackageContext));  
if(pPackageHead->nDataLen>0)  
{  
len = m\_TcpSock.ReceiveSize((char\*)PackageContext,pPackageHead->nDataLen);  
}  
        }

m\_TcpSock是一个封装了SOCKET的类的变量,其中的ReceiveSize用于接收一定长度的数据,直到接收了一定长度的数据或者网络出错才返回.

int winSocket::ReceiveSize( char\* strData, int iLen )  
{  
if( strData == NULL )  
return ERR\_BADPARAM;  
char \*p = strData;  
int len = iLen;  
int ret = 0;  
int returnlen = 0;  
while( len > 0)  
{  
ret = recv( m\_hSocket, p+(iLen-len), iLen-returnlen, 0 );  
if ( ret == SOCKET\_ERROR || ret == 0 )  
{

return ret;  
}

len -= ret;  
returnlen += ret;  
}

return returnlen;  
}  
对于非阻塞的SOCKET,比如完成端口,我们可以提交接收包头长度的数据的请求,当 GetQueuedCompletionStatus返回时,我们判断接收的数据长度是否等于包头长度,若等于,则提交接收包体长度的数据的请求,若不等于则提交接收剩余数据的请求.当接收包体时,采用类似的方法.

载自： <http://blog.csdn.net/fjcailei/archive/2009/06/17/4276463.aspx>  
======================================================================  
几个问题：http://www.qqgb.com/Program/VC/VCJQ/Program\_200509.html  
这个问题产生于编程中遇到的几个问题：   
1、使用TCP的Socket发送数据的时候，会出现发送出错，WSAEWOULDBLOCK，在TCP中不是会保证发送的数据能够安全的到达接收端的吗？也有窗口机制去防止发送速度过快，为什么还会出错呢？   
  
2、TCP协议，在使用Socket发送数据的时候，每次发送一个包，接收端是完整的接受到一个包还是怎么样？如果是每发一个包，就接受一个包，为什么还会出现粘包问题，具体是怎么运行的？   
  
3、关于Send，是不是只有在非阻塞状态下才会出现实际发送的比指定发送的小？在阻塞状态下会不会出现实际发送的比指定发送的小，就是说只能出现要么全发送，要么不发送？在非阻塞状态下，如果之发送了一些数据，要怎么处理，调用了Send函数后，发现返回值比指定的要小，具体要怎么做？   
  
4、最后一个问题，就是TCP/IP协议和Socket是什么关系？是指具体的实现上，Socket是TCP/IP的实现？那么为什么会出现使用TCP协议的Socket会发送出错（又回到第一个问题了，汗一个）   
  
实在是有点晕了，如果我的问题有不清楚的地方，或者分数有问题，欢迎指出，谢谢

这个问题第1个回答：  
1 应该是你的缓冲区不够大,   
2 tcp是流,没有界限.也就所所谓的包.   
3 阻塞也会出现这种现象,出现后继续发送没发送出去的.   
4 tcp是协议,socket是一种接口,没必然联系.错误取决于你使用接口的问题,跟tcp没关系.

这个问题第2个回答：  
1 应该是你的缓冲区不够大,   
2 tcp是流,没有界限.也就无所谓包.   
3 阻塞也会出现这种现象,出现后继续发送没发送出去的.   
4 tcp是协议,socket是一种接口,没必然联系.错误取决于你使用接口的问题,跟tcp没关系.

这个问题第3个回答：  
1、应该不是缓冲区大小问题，我试过设置缓冲区大小，不过这里有个问题，就是就算我把缓冲区设置成几G，也返回成功，不过实际上怎么可能设置那么大、、、   
  
3、出现没发送完的时候要手动发送吧，有没有具体的代码实现？   
  
4、当选择TCP的Socket发送数据的时候，TCP中的窗口机制不是能防止发送速度过快的吗？为什么Socket在出现了WSAEWOULDBLOCK后没有处理？

这个问题第4个回答：  
1.在使用非阻塞模式的情况下，如果系统发送缓冲区已满，并示及时发送到对端，就会产生该错误，继续重试即可。   
3.如果没有发完就继续发送后续部分即可。

这个问题第5个回答：  
1、使用非阻塞模式时，如果当前操作不能立即完成则会返回失败，错误码是WSAEWOULDBLOCK，这是正常的，程序可以先执行其它任务，过一段时间后再重试该操作。   
2、发送与接收不是一一对应的，TCP会把各次发送的数据重新组合，可能合并也可能拆分，但发送次序是不变的。   
3、在各种情况下都要根据send的返回值来确定发送了多少数据，没有发送完就再接着发。   
4、socket是Windows提供网络编程接口，TCP/IP是网络传输协议，使用socket是可以使用多种协议，其中包括TCP/IP。

这个问题第6个回答：  
up

这个问题第7个回答：  
发送的过程是：发送到缓冲区和从缓冲区发送到网络上   
WSAEWOULDBLOCK和粘包都是出现在发送到缓冲区这个过程的