[**AS3 Socket 基础知识(很全面)**](http://blog.csdn.net/liminxin272/article/details/7021127)

在讲AS3 Socket之前我们先来搞清几个概念，这有助于我们更加了解网络通信！

# 1、网络中进程之间如何通信？

首要解决的问题是如何唯一标识一个进程，否则通信无从谈起！在本地可以通过进程PID来唯一标识一个进程，但是在网络中这是行不通的。其实TCP/IP协议族已经帮我们解决了这个问题，网络层的“**ip地址**”可以唯一标识网络中的主机，而传输层的“**协议+端口**”可以唯一标识主机中的应用程序（进程）。这样利用三元组（ip地址，协议，端口）就可以标识网络的进程了，网络中的进程通信就可以利用这个标志与其它进程进行交互。

使用 TCP/IP协议的应用程序通常采用应用编程接口：UNIX  BSD的套接字（socket）和UNIX System V的TLI（已经被淘汰），来实现网络进程之间的通信。就目前而言，几乎所有的应用程序都是采用socket，而现在又是网络时代，网络中进程通信是无处 不在，这就是我为什么说“一切皆socket”。

# 2、什么是Socket？

上面我们已经知道网络中的进程是通过 socket来通信的，那什么是socket呢？socket起源于Unix，而Unix/Linux基本哲学之一就是“一切皆文件”，都可以用“打开 open –> 读写write/read –> 关闭close”模式来操作。我的理解就是Socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对其进行的操作 （读/写IO、打开、关闭），这些函数我们在后面进行介绍。

### socket一词的起源:在 组网领域的首次使用是在1970年2月12日发布的文献IETF RFC33中发现的，撰写者为Stephen Carr、Steve Crocker和Vint Cerf。根据美国计算机历史博物馆的记载，Croker写道：“命名空间的元素都可称为套接字接口。一个套接字接口构成一个连接的一端，而一个连接可完 全由一对套接字接口规定。”计算机历史博物馆补充道：“这比BSD的套接字接口定义早了大约12年。”

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

好了，下面来讲讲AS3中的Socket编程。

我们在使用ActionScript3.0进行Socket编程的时候需要关注下面的问题,我们将在今后的学习中逐个对下面的问题进行讨论,并尽量逐渐的改进我们的程序.

1.与Socket服务器建立连接.   
2.向Socket服务器发送数据.   
3.从Socket服务器读数据.   
4.同Socket服务器进行握手,并确定收到了什么样的数据和如何处理这些数据.   
5.与Socket服务器断开,或者当服务器想与你断开的时候发消息给你.   
6.处理使用Sockets时候引发的错误.   
  
**1.与Socket服务器建立连接.**  
  
解决方法:   
     我们通过调用Socket.connect( )或者XMLSocket.connect( )方法并监听网络连接的事件消息.   
  
讨论:   
     连接一台Socket服务器你需要确定两个信息,一个是Socket服务器的域名或者IP地址,另一个是服务器监听的端口号.   
     无论你使用的是Socket还是XMLSocket类的实例,连接请求都是完全的一样的,两个类都是使用一个名叫connect()的方法,该方法有两个参数:   
host :   
     该参数为字符串类型,可以是一个域名,例如"www.example.com",也可以是一个IP地址,例如"192.168.1.101".如果Socket服务器与你该Flash影片发布的Web服务器是同一个,该参数为Null.  
port :   
     该参数为一个表示Socket服务器监听端口的int值.该值最小为1024.除非在服务器中有一个policy文件,用于指定允许端口号小于1024.   
     因为Flash Socket编程是一个异步的过程,connect()方法不会等到一个连接完成后再执行下一行代码的执行.如果你想在一个连接完全执行完之前与一个Socket完全绑定,那么你将会得到一个意想不到的结果,并且你当前的代码将不能工作.  
     在尝试一个新的Socket连接的时候我们最好先添加一个连接事件监听器.当一个连接建立成功,Socket或者XMLSocket会发出一个连接事件,这就可以让你知道交互已经准备好了.  
     下面举了一个Socket实例与本地Socket服务器的2900端口建立连接的例子:   
  
package {   
   import flash.display.Sprite;   
   import flash.events.\*;   
   import flash.net.Socket;   
  
   public class SocketExample extends Sprite {   
  
     private var socket:Socket;   
  
     public function SocketExample(   ) {   
       socket = new Socket(   );   
       
       // Add an event listener to be notified when the connection   
       // is made   
       socket.addEventListener( Event.CONNECT, onConnect );   
       
       // Connect to the server   
       socket.connect( "localhost", 2900 );   
     }   
     
     private function onConnect( event:Event ):void {   
       trace( "The socket is now connected..." );   
     }   
     
   }   
}   
  
     如果你想通过XMLSocket与服务器建立连接代码也是基本一样的.首先你创建了一个连接事件监听器,然后调用connect()方法.所不同的是Socket实例改为了XMLSocket:  
  
package {   
   import flash.display.Sprite;   
   import flash.events.\*;   
   import flash.net.XMLSocket;   
  
   public class SocketExample extends Sprite {   
  
     private var socket:XMLSocket;   
  
     public function SocketExample(   ) {   
       socket = new XMLSocket(   );   
       
       // Add an event listener to be notified when the connection is made   
       socket.addEventListener( Event.CONNECT, onConnect );   
       
       // Connect to the server   
       socket.connect( "localhost", 2900 );   
     }   
     
     private function onConnect( event:Event ):void {   
       trace( "The xml socket is now connected..." );   
     }   
     
   }   
}   
  
     如果连接失败,可能是下面两种原因的一种:一种是连接立即失败和运行时错误,另一种是如果无法完成连接从而产生一个ioError或者securityError事件.关于错误事件处理信息的描述,我们打算改日讨论.  
     请牢记,当与一个主机建立一个Socket连接时,Flash Player要遵守如下安全沙箱规则.   
     1.Flash的.swf文件和主机必须严格的在同一个域名,只有这样才可以成功建立连接.   
     2.一个从网上发布的.swf文件是不可以访问本地服务器的.   
     3.本地未通过认证的.swf文件是不可以访问任何网络资源的.   
     4.你想跨域访问或者连接低于1024的端口,必须使用一个跨域策略文件. 

     如果尝试连接未认证的域或者低端口服务,这样就违反了安全沙箱策略,同时会产生一个securityError事件.这些情况都可以通过使用一个跨域策 略 文件解决.无论是Socket对象还是XMLSocket对象的策略文件,都必须在连接之前通过使用 flash.system.Security.loadPolicyFile()方法载入策略文件.具体如下:  
     Security.loadPolicyFile("http://www.rightactionscript.com/crossdomain.xml");  
     获得的改策略文件不仅定义了允许的域名,还定义了端口号.如果你不设置端口号,那么Flash Player默认为80端口(HTTP协议默认端口).在<allow-access-from>标签中可以使用逗号隔开设置多个端口号.下 面这个例子就是允许访问80和110端口.  
  
<?xml version="1.0"?>   
  
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">  
<cross-domain-policy>   
   <allow-access-from domain="\*" to-ports="80,110" />   
</cross-domain-policy>   
  
**2.向Socket服务器发送数据.**

解决方法:   
     对于Socket对象来说,通过是用write方法(writeByte(),writeUTFBytes( )等方法.)先向缓存区写入数据,然后使用flush()方法发送数据.对于XMLSocket对象,使用send()方法.  
  
讨论:   
     Socket和XMLSocket类向Socket服务器发送数据的方法是不相同的.让我们首先看一下Socket类的方法.   
     当你使用Socket对象向服务器发送数据的时候,你首先要将数据写入到一个缓冲区中.Socket 类设置了一系列的方法来写数据.每一个方法都用于写不 同的数据类型的数据(或者不同的数据).这些方法分别是: writeBoolean( ), writeByte( ), writeBytes( ), writeDouble( ), writeFloat( ), writeInt( ), writeMultiByte( ), writeObject( ), writeShort( ), write- UnsignedInt( ), writeUTF(), 和writeUTFBytes( ). 这些方法大多数都只接受一个参数,该参数的类型同方法的名字相匹配.例如,writeBoolean()方法接受一个布尔值作为参数,而 writeByte( ), writeDouble( ), writeFloat( ), writeInt( ), writeShort( ), writeUnsignedInt( ) 方法接受一个数字型参数.writeObject()方法接受一个对象类型作为参数,但该对象必须序列化成为AMF格式.writeBytes( )方法允许你传一个ByteArray参数,并带有偏移量和长度两个参数.例如,下面这段代码,调用了一个writeBytes( )方法,该方法将ByteArray对象中的所有byt值都传出去了(偏移量为0,长度和ByteArray数组长度等长):  
  
socket.writeBytes(byteArray, 0, byteArray.length);   
  
     writeUTF( )和writeUTFBytes( ) 方法允许你的发送字符串类型的参数.每个一个方法只接受一个字符串作为参数.writeUTFBytes( )方法简单的将字符串作为Bytes发送.writeUTF( )方法在写入真正数据之前,先写入byts的数量.  
     writeMultiByte( )方法也允许字符串类型的参数,但是使用的为非默认字符集.该方法需要两个参数:字符串和字符集名称.在Flash和Flex的帮助文档中有一个自持所有 字符集的列表,该列表中的标签和描述符是一一对应的.使用标签值作为writeMultiByte( )作为字符集.例如下面的代码发送了一个编码为Unicode的字符串:  
  
socket.writeMultiByte("example", "unicode");   
     
     向一个Socket对象传数值的方法完全依赖于你所有数据的类型和服务所接受数据的类型.使用一个Socket对象,你完全可以使用 ActionScript写一个Telnet和POP mail客户端.这两种协议都支持ASCII字符指令.例如,在连接一个POP服务器之后,你可以通过使用USER指令指定一个用户.下面代码向一个 Socket对象发一条指令:  
  
// POP servers expect a newline (\n) to execute the preceding command.   
socket.writeUTFBytes("USER exampleUsername\n");   
     
     向一个Socket对象写入数据其实并没有将数据发送到Socket服务器.每调用一个write方法都向Socket对象添加一个数据.例如,下面代码向一个Socket对象添加了四个byte的数据,但是没有一个发出了.  
  
socket.writeByte(1);   
socket.writeByte(5);   
socket.writeByte(4);   
socket.writeByte(8);   
  
     当你想将这些累积的数据发送到Socket服务器需要调用flush()方法.flush()方法调用之后将把所有已经写入的数据发送出去,并清空缓冲区:   
  
socket.flush(   );   
  
     XMLSocket类是一个非常简单用于发送数据的API.写于发数据都是由send()这一个方法来完成的.send()方法可以接受任何数据类型的参 数.它可以将所有的参数都转换为一个字符串类型并发送到服务器.通常参数为一个XML对象或者一个包含数据结构类似XML数据的字符串:  
  
xmlSocket.send(xml);   
  
     然而,准确的格式完全依赖于服务器所能够接受的格式.如果服务器接受XML格式的数据,你必须发送XML格式的数据.如果服务器只接受URL编码的数据,你也必须发送URL编码的数据.  
  
  
**3.从Socket服务器读数据**  
  
解决方法:   
     对于Socket实例,先收到socketData事件,然后调用如下两个方法的一个,比如,readByte()或者readInt(),在事件控制器中确定不会去读过去的bytesAvailable.  
     对于XMLSocket实例,先收到data事件,然后解析从事件控制器内部装载的XML数据.   
  
讨论:   
     从一个socket连接接收的数据依赖于你使用的Socket的类型.socket和XMLSocket都可以从服务器接受到数据,但是它们处于不同重量级的技术.让我们在讨论XMLSocket之前先关注下Socket类.  
     我都知道socket在Flash中是一个异步的行为.因此,它就不能简单的创建一个Socket连接,然后就立刻尝试去读取数据.read方法不能等到 从服务器传过来数据之后在返回.换句话说,你只能在客户端从服务器载入所有数据之后才可以读取数据.在数据可用之前读数据会产生一个错误.  
     通过socketData事件广播到Socket实例,这样我们就可以知道什么时候数据可以被读取.那么我们要为socketData事件添加一个事件监 听器,任何时候只要有新的数据从一个socket服务器发送过来,都会触发事件控制器.在事件处理器的内部我们写入我们要执行的代码去读取和处理收到的数 据.  
     从一个前端服务器读取数据,Socket类为我们提供了许多不同的方法,这些方法依赖于你所读得数据类型.例如,你可以通过readByte()方法读一 个byte数据,或者通过一个使用readUnsignedInt()方法去读一个无符号整数.下面这个表列出来能够从服务器读取的数据类型,返回值,和 read方法每次读入的字节数.  
  
Table:Socket read methods for various datatypes   
方法:返回值类型         描述         字节数   
readBoolean( ):Boolean         从Socket读取一个Boolean值.         1   
readByte( ):int         从Socket读取一个byte值.         1   
readDouble( ):Number         从Socket读取一个IEEE 754双精度浮点数.         8   
readFloat( ):Number         从Socket读取一个IEEE 754单精度浮点数.         4   
readInt( ):int         从Socket读取一个有符号32-bit整数值.         4   
readObject( ):\*         从Socket读取一个AMF-encoded对象.         n   
readShort( ):int         从Socket读取一个有符号16-bit整数值.         2   
readUnsignedByte( ):uint         从Socket读取一个无符号字节.         1   
readUnsignedInt( ):uint         从Socket读取一个无符号32-bit整数         4   
readUnsignedShort( ):uint         从Socket读取一个无符号16-bit整数.         2   
readUTF( ):String         从Socket读取一个一个UTF8字符串.         n   
  
  
     有两个额外的方法没有在上面这个表中描述.它们分别是readBytes()和readUTFBytes().readBytes()方法只可以让socket读数据但不能返回一个值,并且该方法需要3个参数:  
  
bytes:   
     一个flash.util.ByteArray实例读取从socket中收到的数据.   
offset:   
     一个uint值,指定从什么位置开始读取socket中收到数据的偏移量.默认值为0.   
length:   
     一个uint值,用于指定读取bytes的数量.默认值为0,意思就是说将所有的可用的数据都放入ByteArray中.   
  
     另一个readUTFBytes()方法,只需要一个长度参数用于指定UTF-8字节的读入数量,并且该方法会将所有读入的字节码转换成为字符串类型.   
  
注意:在从一个Socket读数据之前,首先要判断bytesAvailable的属性.如果你不知道要读入的数据类型是什么就去读数据的话,将会产生一个错误(flash.errors.EOFError).  
  
     下面的例子代码连接了一个socket服务器,读取并显示每次从服务器发来的数据.   
  
package {   
   import flash.display.Sprite;   
   import flash.events.ProgressEvent;   
   import flash.net.Socket;   
  
   public class SocketExample extends Sprite {   
  
     private var socket:Socket;   
  
     public function SocketExample(   ) {   
       socket = new Socket(   );   
       
       // Listen for when data is received from the socket server   
       socket.addEventListener( ProgressEvent.SOCKET\_DATA, onSocketData );   
  
       // Connect to the server   
       socket.connect( "localhost", 2900 );   
     }   
  
     private function onSocketData( eventrogressEvent ):void {   
       trace( "Socket received " + socket.bytesAvailable + " byte(s) of data:" );  
  
       // Loop over all of the received data, and only read a byte if there   
       // is one available   
       while ( socket.bytesAvailable ) {   
         // Read a byte from the socket and display it   
         var data:int = socket.readByte(   );   
         trace( data );   
       }   
     }   
   }   
}   
  
     在上面的这个例子中,如果一个socket服务器发送回一个消息(例如"hello"),当一个客户段连入服务器就会返回并输出下面类似的文字:   
  
Socket received 5 byte(s) of data:   
72   
101   
108   
108   
111   
  
注意:一旦数据从socket读出,它就不能再次被读.例如,读一个字节之后,这个字节就不能再"放回来",只能读后边的字节.   
  
     当收到的数据为ASCII编码,你可以通过readUTFBytes()方法重新构建一个字符串.readUTFBytes()方法需要知道多少个字节需要转换为字符串.你可以使用bytesAvailable去读所有的字节数据:  
  
var string:String = socket.readUTFBytes(socket.bytesAvailable);   
  
     XMLSocket类的动作和Socket类相比在从服务器接受数据的风格相似.两者都是通过事件监听器来监听数据接收通知的,这主要取决于Flash异步的Socket实现.然而,在处理实际数据的时候有很大的不同.  
  
     有个XMLSocket实例在从服务器下载完数据后分发数据事件.通过flash.events.DataEvent.DATA常量定义的数据事件包含一个data属性,该属性包含了从服务器收到的信息.  
  
注意:使用XMLSocket从服务器返回的数据总是认为是一个字符串类型的数据.这样不用为任何数据类型的数据指定读取方法.   
  
     这些从服务器返回的数据是没有经过任何处理的原始数据.因此,你不能通过XMLSocket连接立即使用XML,你发送和接收的都是纯字符串数据.如果你期望XML,在你处理数据之前,你必须首先将这些数据转换为一个XML的实例.  
     下面的这段代码在初始化的时候通过XMLSocket连接到了本地服务器的2900端口.在连接成功之后,一个<test>消息会发送到服务 器.onData事件监听者控制从服务器返回的响应.在本例中返回字符串<response><test success='true'/></response>.你可以通过事件的data属性发现为字符串数据,然后XML类的构造函数将 字符串转换成为了XML实例.最后,通过使用E4X语法的XML实例的一部分信息.(关于通过使用E4X处理XML的更多详细信息,我们需要另外讨论.)  
  
package {   
   import flash.display.Sprite;   
   import flash.events.Event;   
   import flash.events.DataEvent;   
   import flash.net.XMLSocket;   
  
   public class SocketExample extends Sprite {   
  
     private var xmlSocket:XMLSocket;   
  
     public function SocketExample(   ) {   
       xmlSocket = new XMLSocket(   );   
       
       // Connect listener to send a message to the server   
       // after we make a successful connection   
       xmlSocket.addEventListener( Event.CONNECT, onConnect );   
       
       // Listen for when data is received from the socket server   
       xmlSocket.addEventListener( DataEvent.DATA, onData );   
  
       // Connect to the server   
       xmlSocket.connect( "localhost", 2900 );   
     }   
     
     private function onConnect( event:Event ):void {   
       xmlSocket.send( "<test/>" );   
     }   
  
     private function onData( eventataEvent ):void {   
       // The raw string returned from the server.   
       // It might look something like this:   
       // <response><test success='true'/></response>   
       trace( event.data );   
       
       // Convert the string into XML       
       var response:XML = new XML( event.data );   
       
       // Using E4X, access the success attribute of the "test"   
       // element node in the response.   
       // Output: true   
       trace( response.test.@success );   
     }   
   }   
}   
  
注意:在data事件分发数据之前,XMLSocket实例必须从服务器收到一个表示为空的byte('\\0').也就是说,从服务器仅仅只发送所需要的字符串是不够的,必须在结尾处加入一个表示为空的byte.

**4.同Socket服务器进行握手,并确定收到了什么样的数据和如何处理这些数据.**

解决方法:   
     创建不同的常量来声明协议的状态.使用这些常量将指定的处理函数映射到相应的状态.在一个socketData事件控制器中,通过状态映射调用这些函数的.   
  
讨论:   
     建立Socket连接通常要处理握手这个环节.尤其是在服务器初始化需要向客户端发送数据.然后客户端通过一种特殊的方式相应这些数据,接着服务器因此再次响应.整个处理过程直到握手完成并且建立起一个"正常的"连接为止.  
  
   处理服务器的不同响应是非难的,主要的原因是socketData事件控制器不能保存上下文的顺序.也就是说,服务器的响应不会告诉你"为什么"响应, 也不告诉你这些响应数据被那个处理程序来处理.要想知道如何处理这些从服务器返回的响应不能从响应的本身来获得,尤其在响应变化的时候.或许一个响应返回 了两个字节码,另一个返回了一个整数值还跟了一个双精度浮点数.这样看来让响应本身处理自己是一大难题.  
  
   我们通过创建一个状态量来标注不同的上下文,服务器通过这些上下文将数据发送到客户端.与这些状态量都有一个相关联的函数来处理该数据,这样你就可以很轻松的按照当前的协议状态去调用正确的处理函数.  
  
   当你要与一个Socket服务器建立连接需要考虑如下几个步骤:   
1.当与服务器连接的时候,服务器立刻返回一个标志服务器可以支持的最高协议版本号的整数值.   
2.客户端在响应的时候会返回一个实际使用协议的版本号.   
3.服务器返回一个8byte的鉴定码.   
4.然后客户端将这鉴定码返回到服务器.   
5.如果客户端的响应不是服务器端所期望的,或者,就在这个时候该协议变成了一个常规操作模式,于是握手结束.   
  
   实际上在第四步可以在鉴定码中包含更多的安全响应.你可以通过发送各种加密方法的密匙来代替逐个发送的鉴定码.这通常使用在客户端向用户索要密码的时候, 然后密码成为了加密过的8byte鉴定码.该加密过的鉴定码接着返回到服务器.如果响应的鉴定码匙服务器所期望的,客户端就知道该密码是正确的,然后同意 建立连接.  
  
   实现握手框架,你首先要为处理从服务器返回的不同类型的数据分别创建常量.首先,你要从步骤1确定版本号.然后从步骤3收取鉴定码.最后就是步骤5的常规操作模式.我们可以声明  
  
如下常量:   
  
public const DETERMINE\_VERSION:int = 0;   
public const RECEIVE\_CHALLENGE:int = 1;   
public const NORMAL:int = 2;   
  
   常量的值并不重要,重要的是这些值要是不同的值,两两之间不能有相同的整数值.   
  
   下一个步骤我们就要为不同的数据创建不同处理函数了.创建的这三个函数分别被命名为readVersion( ), readChallenge( ) 和 readNormalProtocol( ). 创建完这三个函数后,我们就必须将这三个函数分别映射到前面不同状态常量,从而分别处理在该状态中收到的数据.代码如下:  
  
stateMap = new Object(   );   
stateMap[ DETERMINE\_VERSION ] = readVersion;   
stateMap[ RECEIVE\_CHALLENGE ] = readChallenge;   
stateMap[ NORMAL             ] = readNormalProtocol;   
  
   最后一步是编写socketData事件处理控制器,只有通过这样的方式,建立在当前协议状态之上的正确的处理函数才可以被调用.首先需要创建一个 currentState的int变量.然后使用stateMap去查询与currentState相关联的函数,这样处理函数就可以被正确调用了.  
  
var processFunc:Function = stateMap[ currentState ];   
processFunc(   ); // Invoke the appropriate processing function   
  
   下面是一点与薄记相关的处理程序.在你的代码中更新currentState从而确保当前协议的状态.   
   前面我们所探讨的握手步骤的完整的代码如下:  
package {  
import flash.display.Sprite;  
import flash.events.ProgressEvent;  
import flash.net.Socket;  
import flash.utils.ByteArray;

public class SocketExample extends Sprite {

    // The state constants to describe the protocol  
    public const DETERMINE\_VERSION:int = 0;  
    public const RECEIVE\_CHALLENGE:int = 1;  
    public const NORMAL:int = 2;  
     
    // Maps a state to a processing function  
    private var stateMap:Object;  
     
    // Keeps track of the current protocol state  
    private var currentState:int;  
     
    private var socket:Socket;

    public function SocketExample( ) {  
      // Initialzes the states map  
      stateMap = new Object( );  
      stateMap[ DETERMINE\_VERSION ] = readVersion;  
      stateMap[ RECEIVE\_CHALLENGE ] = readChallenge;  
      stateMap[ NORMAL            ] = readNormalProtocol;  
       
      // Initialze the current state  
      currentState = DETERMINE\_VERSION;

      // Create and connect the socket  
      socket = new Socket( );  
      socket.addEventListener( ProgressEvent.SOCKET\_DATA, onSocketData );  
      socket.connect( "localhost", 2900 );  
    }  
     
    private function onSocketData( eventrogressEvent ):void {  
      // Look up the processing function based on the current state  
      var processFunc:Function = stateMap[ currentState ];  
      processFunc( );  
    }  
     
    private function readVersion( ):void {  
      // Step 1 - read the version from the server  
      var version:int = socket.readInt( );  
       
      // Once the version is read, the next state is receiving  
      // the challenge from the server  
      currentState = RECEIVE\_CHALLENGE;  
       
      // Step 2 - write the version back to the server  
      socket.writeInt( version );  
      socket.flush( );  
    }  
     
    private function readChallenge( ):void {  
      // Step 3 - read the 8 byte challenge into a byte array  
      var bytes:ByteArray = new ByteArray( );  
      socket.readBytes( bytes, 0, 8 );  
       
      // After the challenge is received, the next state is  
      // the normal protocol operation  
      currentState = NORMAL;  
       
      // Step 4 - write the bytes back to the server  
      socket.writeBytes( bytes );  
      socket.flush( );  
    }  
     
    private function readNormalProtocol( ):void {  
      // Step 5 - process the normal socket messages here now that  
      // that handshaking process is complete  
    }  
}  
}

**5.与Socket服务器断开,或者当服务器想与你断开的时候发消息给你.**

解决方法:  
    通过调用Socket.close( )或者XMLSocket.close( )方法显性的断开与服务器的连接.同时可以通过监听close事件获得服务器主动断开的消息.

讨论:  
    通常情况下我们需要对程序进行下清理工作.比如说,你创建了一个对象,当这个对象没有用的时候我们就要删除它.因此,无论我们什么时候连接一个 Socket服务器,都要在我们完成了必要的任务之后显性的断开连接. 一直留着无用的Socket连接浪费网络资源,应该尽量避免这种情况.如果你没有断开一个连接,那么这个服务器会继续保持着这个无用的连接.这样一来就很 快会超过了服务器最大Socket连接上线.

Socket和XMLSocket对象断开连接的方法是一样的.你只需要调用close()方法就可以了:

// Assume socket is a connected Socket instance  
socket.close( ); // Disconnect from the server   
简单心情,简单生活!   
回复 | 引用 | 举报 | 删除2楼 yanxuanlin   评论时间：2008年7月9日 0时34分8秒  
同样的,XMLSocket对象断开连接的方法一样:

// Assume xmlSocket is a connected XMLSocket instance  
xmlSocket.close( ); // Disconnect from the server

    close()方法用于通知服务器客户端想要断开连接.当服务器主动断开连接会发消息通知客户端.可以通过调用addEventListener()方法 注册一个close事件的一个监听器.Socket 和 XMLSocket都是使用Event.CLOSE作为"连接断开"事件类型的;例如:

// Add an event listener to be notified when the server disconnects  
// the client  
socket.addEventListener( Event.CLOSE, onClose );

注意:调用close()方法是不会触发close事件的,只用服务器主动发起断开才会触发.一旦一个Socket断开了,就无法读写数据了.如果你想要从新这个连接,你只能再建立个新的连接了.

**6.处理使用Sockets时候引发的错误.**

解决方法:  
    使用try/catch处理I/O和EOF(end of file)错误.

讨论:  
    Socket和XMLSocket类对错误的处理很类似.不如,当调用connect()方法的时候,在下面任何一个条件成立的情况下Socket和XMLSocket对象会抛出一个类型为SecurityError的错误.

\* 该.swf未通过本地安全认证.  
\* 端口号大于655535.

    当调用XMLSocket对象的send()或者Socket对象的flush()的时候,如果socket还没有连接这两个方法都会抛出一个类型为 IOError的错误.尽管你可以将send()或者flush()方法放入try/catch结构块中,你也不能依赖于try/catch结构块作为你 应用程序的逻辑.更好的办法是,在调用send()或者flush()方法之前使用一个if语句首先判断一下Socket对象的connected属性是 否为True.例如,下面的代码使用了if语句作为程序逻辑的一部分,当Socket对象当前不是连接状态就调用 connectToSocketServer()方法.但是我们依然需要将flush()方法放到try/catch语句块中.通过使用 try/catch语句块将flush()方法抛出的错误写入到日志中:

if ( socket.connected ) {  
    try {  
        socket.flush( );  
    }  
    catch( error:IOError ) {  
        logInstance.write( "socket.flush error\n" + error );  
    }  
}  
else {  
    connectToSocketServer( );  
}

    所有的Socket类的read方法都能够抛出EOFError和IOError类型的错误.当你试图读一个数据,但是没有任何可用数据将触发EOF错误.当你试图从一个已经关闭的Socket对象中对数据时将会抛出I/O错误.

    除了Socket和XMLSocket类的方法能够抛出的错误以外,这些类的对象还会分发错误事件.有两种基本的错误事件类型,他们分别由 socketIOError和securityError错误引起.IOError事件为IOErrorEvent类型,当数据发送或接收失败触发该事 件.SecurityError事件是SecurityErrorEvent类型,当一个Socket尝试连接一个服务器,但由于服务器不在安全沙箱范围 之内或者端口号小于1024的时候触发该错误事件.

注意:这两种安全策略引起的错误都可以通过跨域访问策略文件解决.