很高兴进入第三期的 flash 杂志，这一次我会和浪迹天涯分别整理写这些笔记文章，这些都是我们的兴趣和爱好。没有为什么，只是因为我们对 Flash 有一份独特喜爱之情。

----夏天的树人

这次和夏天的树人一起整理这些笔记，因为我和他有共同的爱好和共同的追求。希望能给大家一些帮助！！

---浪迹天涯

感谢插画提供：By lin

Ps 封面设计：lonmy

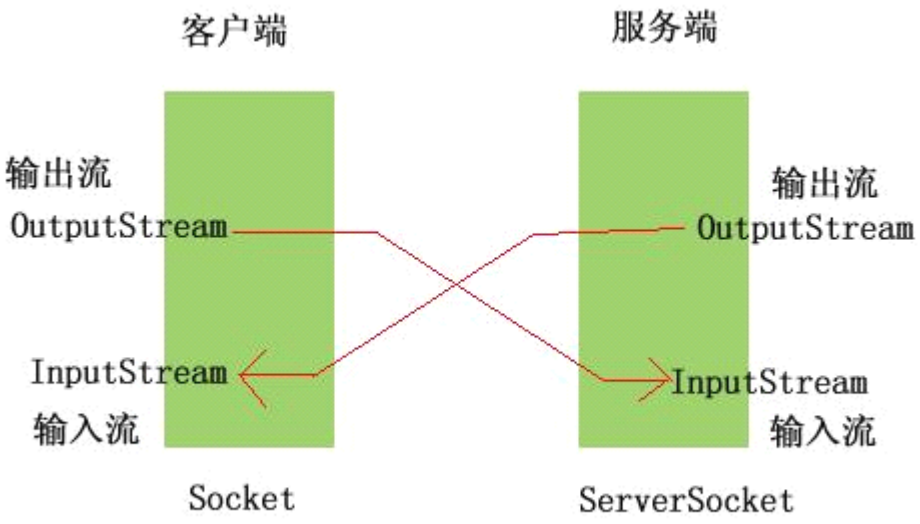
目录

一、Flash Sockt 编程

Flash 在8.0的版本时候已经有了关于socket 通信一些功能，但是功能还是很局限。Flash 升级到cs3版本后，引进As3.0 编程，关于socket编程有了一定提升，它可以发送一些二进制的数据，通信能力得到一定提高。在说socket编程之前，我们还是不能忘记，flash在服务器领域还是"小学生"，不过在客户端领域，它就是一个"研究生"；

首先我们看一些利用java做客户端，java做服务器来搭建一个最简单的通信。所用到的知识，包括输入输出流，还有java一些Socket编程。

原理： 通信的过程，是一个双工的过程，也就是说，客户端既然 发送端 也是接收端 。同样服务器也是一样



从这里可以看出，客户端的输入流 就是服务器的输出流所发送来的信息。而服务器的输入流同样是客户端所发送来的

接下来，我们尝试使用java写一个简单的Tcp通信。这里只是一个小小程序，程序上还有设计思想上还需要扩展。

1.1、java客户端

```
import java.net.*;
import java.io.*;

public class TcpClient {

	public static void main(String[] args) {
		String host="127.0.0.1";//本地回环地址
		int port =8001;//端口
		try {
			Socket s = new Socket(host,port);
			InputStream ips =s.getInputStream();//获取输入流
			OutputStream ops=s.getOutputStream();//获取输出流
			BufferedReader brNet = new BufferedReader(
					new InputStreamReader(ips));//输入流

			PrintWriter pw = new PrintWriter(ops,true);
			BufferedReader brKeyBoard = new BufferedReader(
					new InputStreamReader(System.in));//键盘缓冲
			while(true){
				String strWord =brKeyBoard.readLine();
				pw.println("Jim:"+strWord+"\n");
				System.out.println(brNet.readLine());

			}
		} catch (UnknownHostException e) {
			// TODO 自动生成 catch 块
			e.printStackTrace();
		} catch (IOException e) {
			// TODO 自动生成 catch 块
			e.printStackTrace();
		}
	}

}
```

程序解析：

程序的入口：是 main 函数。

在 main 函数里面创建一个 socket 对象： Socket s = new Socket(host,port);

并写上两个参数，一个是主机名，也是 ip ，而 port 就是端口，指定一个端口。（这个端口要在允许的范围填写）

接下来，创建两个输入输出流：

InputStream ips =s.getInputStream();

OutputStream ops=s.getOutputStream();

通过 Socket 类的 getInputStream 和 getOutputStream 返回输入输出流信息

为了使效率提高，在客户端引入缓冲机制。BufferedReader 类用于缓冲使用。

BufferedReader brNet = new BufferedReader(new InputStreamReader(ips));//输入流

BufferedReader brKeyBoard = new BufferedReader(new InputStreamReader(System.in));//键盘缓冲

下一步 接收键盘输入的信息 以及 发送信息到服务器端里面去：

```
while(true){
    String strWord =brKeyBoard.readLine();
    pw.println("Jim:"+strWord+"\n");
    System.out.println(brNet.readLine());

  }
```

Socket 会抛出 OException 异常，所以要进行捕捉。try{}catch（）处理机制

1.2、服务器端

```
1.  package com;
2.  import java.net.*;
3.  import java.io.*;
4.  public class Server {
5.
6.
7.          public static void main(String[] args) {
8.          PrintWriter os=null;
9.              try{
10.             ServerSocket ss=new ServerSocket(8001);
11.             System.out.println("等待连接....");
12.             Socket sk=ss.accept();
13.             System.out.println("连接成功....");
14.
    BufferedReaderbr=newBufferedReader(newInputStreamReader(sk.getInputStrea
    m()));
15.             System.out.println("获取里面的内容.....");
16.             //等待接受信息
17.           os = new PrintWriter(sk.getOutputStream());
18.           String word=null ;
19.         //发送信息
20.         while((word=br.readLine())!=null)
21.         {
22.           os.println("你好我是服务器");
23.           os.flush();
24.           System.out.println("内容:" + word);
25.
26.         }
27.       }
28.             catch(IOException ex)
29.             {
30.                   ex.printStackTrace();
32.             }
33.       }
34.
35. }
```

设计问题：

服务器搭建其实也很简单

首先创建一个 ServerSocket ss=new ServerSocket(8001);

通过 ServerSocket 对象调用 accept()方法 让其一直阻塞直到返回 Socket 对象这样我们就可以像在客户端一样进行输入流和输出流的处理。

输出结果：将客户端发送的字符串进行输出

```
while((word=br.readLine())!=null)
 {
  os.println("你好我是服务器");
  os.flush();
  System.out.println("内容:" + word);

 }
```

但问题其实还是有的，在客户端发送的时候，由于没有过滤掉回车符所以在服务器里面会输出两个结果

等待连接....

连接成功....

获取里面的内容.....

内容:Jim:dd

内容:

注意一个是有内容，一个是没有内容导致这个 Tcp 的通信出现问题。

接下来，我们尝试搭建 flash 客户端

1.3、Flash 客户端的制作

```
package
{
   import flash.net.Socket;
   import flash.events.*;
   import flash.display.Sprite;
   import flash.errors.*;
   import flash.display.SimpleButton;
   public class Client extends Sprite
   {
      private var mysocket:Socket;
      private var host:String="localhost";    //ip 地址
      private var port:int=8001;    //端口
      public function Client()
      {
         btn.addEventListener(MouseEvent.CLICK,SendData);
         mysocket=new Socket();    //建立 socket 对象
         mysocket.addEventListener(Event.CONNECT,OnConnect);
         mysocket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
         mysocket.addEventListener(ProgressEvent.SOCKET_DATA , receivedata);
         //mysocket.addEventListener(SecurityError
         mysocket.connect(host,port);
      }
      private function OnConnect(e:Event):void
      {
         trace("连接成功");
         mysocket.writeUTFBytes("Test successful2\n");
         mysocket.flush();//发送数据

      }
      private function ioErrorHandler(e:IOErrorEvent):void
      {
         trace("连接失败");
      }
      private function receivedata(e:ProgressEvent):void
      {
         trace("收到的字节数"+mysocket.bytesAvailable);
         var msg:String;
         while (mysocket.bytesAvailable)
         {
            msg+=mysocket.readMultiByte(mysocket.bytesAvailable,"utf8");
            trace(msg);
         }
```

```
        }
        private function SendData(e:MouseEvent):void
        {
            trace("发送");
            mysocket.writeUTFBytes("i am flash\n");
            mysocket.flush();//发送数据


        }
    }
}
```

我们首先解决第一个问题：（其实在 cookbook 那里已经有说的）这里简单演示一下

1.3.1.Flash 连接服务器：

```
mysocket=new Socket();
mysocket.addEventListener(Event.CONNECT,OnConnect);
mysocket.connect(host,port);//连接服务器
```

host ：是连接的 ip 端，这里连接本地所以使用 localhost 或者 127.0.0.1 就 ok

port：端口号只有大于 1024 的时候基本上可以顺利，小于 1024 还要进行一些配置

监听连接是否成功：如果成功就发送信息给客户端

```
private function OnConnect(e:Event):void
  {
  trace("连接成功");
  mysocket.writeUTFBytes("Test successful2\n");//发送消息到客户端
  mysocket.flush();//发送数据
}
```

1.3.2 客户端发送信息

连接成功后，我们可以发送一个简单信息给服务器：如下

```
mysocket.writeUTFBytes("Test successful2\n");
mysocket.flush();//发送数据
```

注意，有个\n 没有这个回车符，看看会有一些什么发生呢。

使用 writeUTFBytes 是不可能立刻发送数据给服务器的，还要使用 flush 方法才能发送给服务器。

1.3.3.处理错误：

Socket 会引发 IO 错误，所以为了完善程序，会加人一些错误处理检测

```
mysocket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
private function ioErrorHandler(e:IOErrorEvent):void
{
    trace("连接失败");
}
```

1.3.4.接收服务器返回的信息

问题:我们希望监听服务器返回的信息,因此 socket 对其 ProgressEvent 事件进行监听。

```
mysocket.addEventListener(ProgressEvent.SOCKET_DATA, receivedata);
private function receivedata(e:ProgressEvent):void
 {
  trace("收到的字节数"+mysocket.bytesAvailable);//返回收到的字节
  var msg:String="";
  while (mysocket.bytesAvailable)
  {
   msg+=mysocket.readMultiByte(mysocket.bytesAvailable,"utf8");
   trace(msg);
  }
 }
```

这样我们可以接收到服务器发送来的信息，但是存在一个问题，接收数据会存在一个回车符合的问题，需要我们去解决。

参考：

bytesAvailable : uint [read-only] 输入缓冲区中可读取的数据的字节数。

readMultiByte(length:uint, charSet:String):String 使用指定的字符集，从该字节流读取一个多字节字符串。

1.3.5.发送信息给服务器

问题：我们希望发送信息给服务器

解决：

使用 writeUTFBytes 方法

```
private function SendData(e:MouseEvent):void
{
 trace("发送");
 mysocket.writeUTFBytes("i am flash\n");
 mysocket.flush();//发送数据

}
```

在场景上放一个按钮上去，通过点击按钮事件，发送信息给服务器 socket 端口。

语法参考：

writeUTFBytes(value:String):void 将一个 UTF-8 字符串写入套接字。

1.4、java服务器

这里写一个简单服务器程序，使用一个做简单的线程，不过线程并不完善，有缺点，但不影响程序演示：

```
package com;
import java.net.*;
import java.io.*;
public class Server implements Runnable {

  public void run()
  {
      PrintWriter os=null;
        try{
        ServerSocket ss=new ServerSocket(8001);
        System.out.println("等待连接....");
        Socket sk=ss.accept();  //连接阻塞直到有 socket 回来
        System.out.println("连接成功....");
        BufferedReader br = new BufferedReader(new InputStreamReader(sk.getInputStr
eam(),"utf-8"));   //缓冲输入流
        System.out.println("获取里面的内容.....");
        //等待接受信息
        os = new PrintWriter(sk.getOutputStream()); //输出数据
        String word=null ;
       //发送信息
       while((word=br.readLine())!=null)
       {
```

```
            System.out.println("内容:" + word);
            os.println("你好我是服务器");
            os.flush();
          }
        }
            catch(IOException  ex)
            {
                ex.printStackTrace();

            }

    }

    public static void main(String[] args) {
      new Thread(new Server()).start();    //启动线程
    }

}
```

接收结果：
等待连接....
连接成功....
获取里面的内容.....
内容:Test successful2
内容:i am flash
内容:i am flash
内容:i am flash
内容:i am flash
内容:i am flash
为了让程序更加完善，我们可以将程序修改得更加完善
客户端发送的内容：
连接成功
收到的字节数 16
null 你好我是服务器
注意：出现 null 这个结果，也许在服务出现一些小毛病，但基本发送数据的和接收数据都成功了
程序需要完善的地方：
包括客户端和服务器
处理回车符合问题
还有多线程的显示列表问题。这是我们所要去完善程序的
最后发现原来在 var msg:String=""; 写上这样 null 就不会出现了

二、AS 杂志之热点难点专题

作者：浪迹天涯 整理
Socket 编程专题

2.1.ActionScript3.0 Socket 编程(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容器.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容器,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,可那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>
```

2.2.ActionScript3.0 Socket 编程(2)

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");
```

相一个 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.

2.3.ActionScript3.0 Socket编程(3)

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容器,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容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.

2.4.ActionScript3.0 Socket 编程(4)

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
        }
    }
}
```

2.5.ActionScript3.0 Socket 编程(5)

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
```

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 事件的一个监听容器.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

2.6.ActionScript3.0 Socket 编程(6)

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

三、Flash 与 javaScript 通信：

ExternalInterface 类是外部 API，在 ActionScript 和 Flash Player 的容器之间实现直接通讯的应用程序编程接口，例如，含有 JavaScript 的 HTML 页。 推荐对所有 JavaScript 与 ActionScript 之间的通信使用 ExternalInterface。

3.1.As3.0 发生参数给Javascript

As3.0 可以调用网页中 html 文件里面 javascript 函数，同样也可以传递参数给 javascript 通过实现 ExternalInterface.call 方法来达到通信的目的。

代码实现：

场景中有一个按钮，属性为 btn，当点击按钮的时候，进行发送参数并调用 javascript

指定文档类：Example .as

```
package
{
   import flash.display.MovieClip;
   import flash.external.ExternalInterface;
   import flash.events.*;
   import flash.display.SimpleButton;

   public class Example extends MovieClip
   {
      public function Example()
      {
         btn.addEventListener(MouseEvent.CLICK,onclick);
          //ExternalInterface.call("aa");
      }
      private function onclick(e:MouseEvent):void
      {
       ExternalInterface.call("myname","hello"); //传递 hello 参数去 javascript 里面去


      }

   }
}
```

As 代码完成后，发布 Html 格式，和 swf 文件，接下来可以通过 dreamweaver 或者文本编辑器打开网页添加如下的 javascript 函数。

在 html 文档里面写上一个函数名为 myname 的函数

```
<script language="javascript">
function myname(title)
{
 window.alert(title);
 document.write("ddd");
  location.href="http://www.baidu.com";
}
</script>
```

实现效果解析：
上面的代码是：在场景有一个按钮，当点击了按钮的时候 发生以下的 javascript 代码。第一发出警告信息
第二写上一个文本“ddd” 第三是一个超链接
这样我们可以尝试做一些 flash 按钮，然后尝试调用 javacript 来做一些超链接 或者其他事情。这样就减少我们很多工作。

应用拓展：flash 广告的应用，通过点击 flash 文件发送给 javascript 文件，跳转统计广告点击率的页面，通过判断是否重复的 ip，如果不是统计数据，发送信息给数据库。

四、Flash+XML 应用：

问题：
我希望可以写一个外部加载 xml 的类，并能够返回 xml 的信息

方法：
仿效 flex 的 httpserver 类，可以返回一个 xml 的信息；

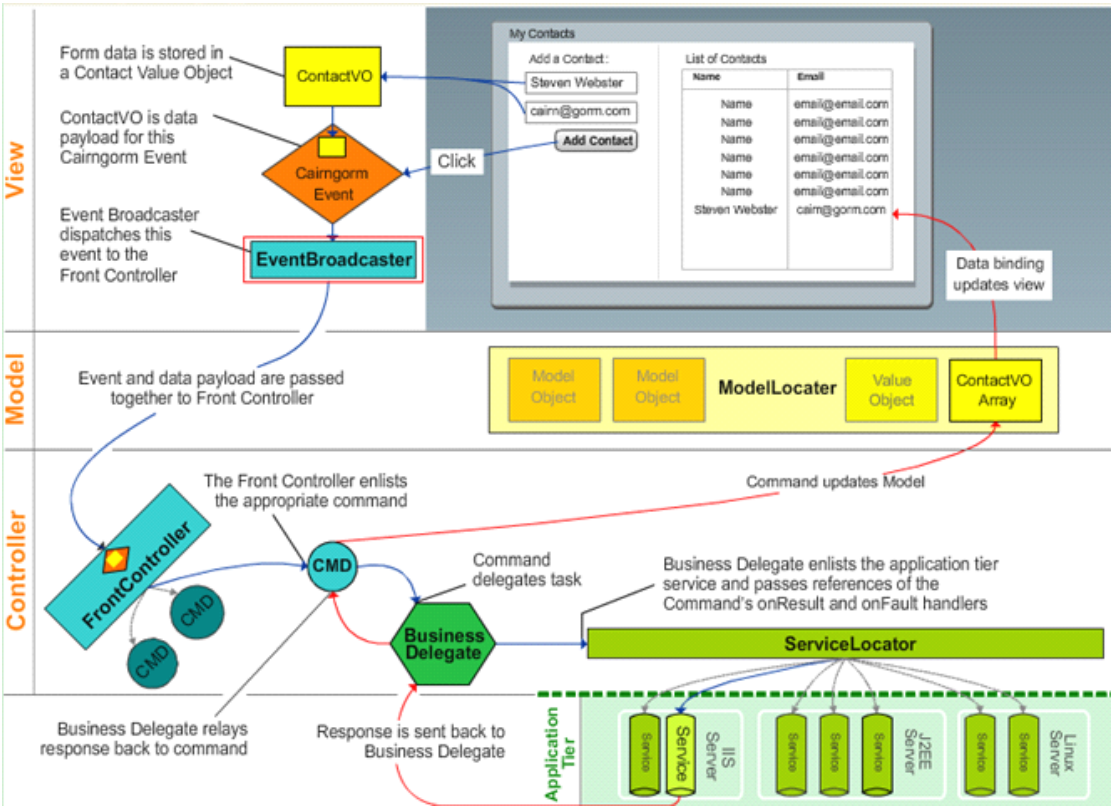
自定义事件类：
在 flash 里面的，首先我们先自定义一个事件类，用于加载 xml 的时候返回一个参数

```
package
{
   import flash.events.Event;
   public class LoadEvent extends Event
   {
      public static  const LOADXML:String="loadxml";
      public var result:XML;
      public function LoadEvent(type:String,bubbles:Boolean=false,cancelable:Boolean=f
alse)
      {
         super(type,false,false);
      }
      override public  function clone():Event
      {
         return new LoadEvent(result);
      }
   }
}
```

代码解析：

自定义个 LoadEvent 事件，这个事件带一个参数， result 类型是 xml ，通过分发事件能够通过事件来传递一些参数。在 MVC 的模式下，这种技术很实用的。

如 ：



自定义一个事件类型：LOADXML

如果有兴趣的可以参考 Cairngorm 框架 网址 http://www.cairngormdocs.org/

2.定义一个加载xml的类，这个类用于管理xml 的加载，XMLManager类派生EventDispatcher类用于分派自定义的事件

```
package
{
   import flash.net.*;
   import flash.events.*;
   public class XMLManager extends  EventDispatcher
   {
      public function XMLManager()
      {

      }
      public function LoadXML(pach:String):void
      {
         var ld:URLLoader=new URLLoader();
         ld.load(new URLRequest(pach));
         ld.addEventListener(Event.COMPLETE,resultHandle);
         ld.addEventListener(IOErrorEvent .IO_ERROR,errorhandle);
      }

      private function resultHandle(e:Event):void
      {
         var myxml:XML=XML(e.target.data );
         var sendevent:LoadEvent=new LoadEvent(LoadEvent.LOADXML);
         sendevent.result=myxml;
         this.dispatchEvent(sendevent);

      }
      private function errorhandle(e:Event):void
      {
         throw new Error("加载失败");

      }

   }
}
```

方法体：LoadXML(路径)； 加载完后，就调用我们的事件

```
var sendevent:LoadEvent=new LoadEvent(LoadEvent.LOADXML);
sendevent.result=myxml;
this.dispatchEvent(sendevent);//分派我们的事件
```

由于我们的 public class XMLManager extends EventDispatcher XMLManager 继承了 EventDispatcher 类 ，而它是 DisplayObject 类的基类 因此我们可以调用他的事件分派 EventDispatcher 类允许显示列表上的任何对象都是一个事件目标，同样允许使用 IEventDispatcher 接口的方法。

在文档类调试：

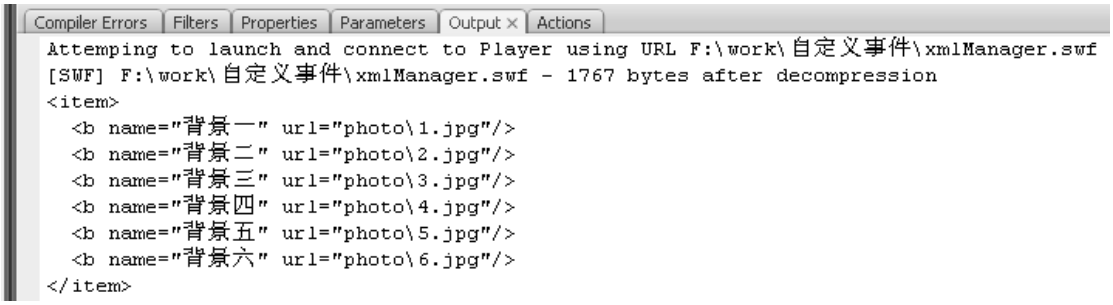
```
package
{
   import flash.display.Sprite;
   import flash.events.*;
   public class main extends Sprite
   {
      public function main()
      {
         var ld:XMLManager=new XMLManager();
         ld.LoadXML("picture.XML");
         ld.addEventListener(LoadEvent.LOADXML ,resulthander);

      }
      private function resulthander(e:LoadEvent):void
      {
         trace(e.result);
      }
   }
}
```

picture.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<item>
<b    name="背景一" url="photo\1.jpg"/>
<b    name="背景二" url="photo\2.jpg" />
<b    name="背景三" url="photo\3.jpg"/>
<b    name="背景四" url="photo\4.jpg"/>
<b    name="背景五" url="photo\5.jpg" />
<b    name="背景六" url="photo\6.jpg"/>
</item>
```

显示结果：



这样我们就获取了外部 XML 的信息，接下来，我们开始读取这些数据。

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    public class main extends Sprite
    {
        private var myxml:XML;
        public function main()
        {
            var ld:XMLManager=new XMLManager();
            ld.LoadXML("picture.XML");
            ld.addEventListener(LoadEvent.LOADXML ,resulthander);
        }
        private function resulthander(e:LoadEvent):void
        {
            trace(e.result);
            myxml=e.result;
            trace(myxml.b[0].@name);//读取第一 b 标签的 name 属性
            trace(myxml.b[0].@url);//读取第一 b 标签的 url 属性
        }
    }
}
```

这样我们可以访问到 xml 里面的信息。通过@为访问属性的符号，name 和 url 都叫属性。
<b name="背景一" url="photo\1.jpg"/> 而 b[0]代表着第一个标签 b，因为这里有六个相同的标签 b 所以我们可以把他看着数组的形式来存放。
至于代码如何去使用，我们需要去动手一下吧。这里不多说了。

五、熟悉 AS3 的 package，以及多个 package 之间的相互通信

说明：一个很简单的 demo，有 4 个按钮，当鼠标划过和移开时会呈现出不同的状态，单击后变成 disabled，再点击其他的按钮时，之前 disabled 的按钮恢复，被点击的按钮失效。
演示：http://www.live-my-life-with-yuyi.com/as3_cases/communicating/
准备工作：打开源文件 communicating_final.fla，点击属性里的发布设置，点击 Actionscript3 旁边的设置，在最下面的 classpath 里，引入 classes 的文件夹的路径，然后点击确定，前期工作就准备完了。
代码：源文件里的代码很简单：

```
import todd.interactive.ButtonSet;
var buttons:ButtonSet = new ButtonSet();
buttons.addButtons([one_mc,two_mc,three_mc,four_mc]);
addChild(buttons);
```

导入 ButtonSet 类，实例化，然后调用里面的一个方法，最后将它放到舞台上。classes 文件夹下面的 todd->interactive 文件夹里有两个 as 文件，其中一个就是刚刚调用的 ButtonSet，来看看 ButtonSet 的源码：

```
package todd.interactive
{
    //其实只需载入 display 和 events 就可以了，不过多载入几个并不影响文件大小和效率
    import flash.display.*;
    import flash.events.*;
    import flash.filters.*;
    import flash.net.*;
    import flash.geom.*;
    import flash.ui.*;
    import flash.utils.*;
    import fl.transitions.*;
    import fl.transitions.easing.*;

    public class ButtonSet extends MovieClip
    {
        public var buttons:Array;

        public function ButtonSet()
        {

        }

        public function addButtons(buttonSet:Array):void
```

```
        {
            buttons = buttonSet;
            for(var i:int = 0; i < buttons.length; i++)
            {
                addChild(buttons[i]);
            }
        }
    }
}
```

一个类应该被放在一个 package 里面，就像钱应该被放到钱包里一样。package 后面定义的是该类的路径。然后一系列常用的类。定义了一个全局变量 buttons，在变量前面加一个 public 就可以了。这样就能在整个类中被访问到。类名应该和文件名一样（区分大小写），然后定义一个同名函数，这个函数会在类被初始化时调用，就像 php4 的类一样。这里只是搭了个架子，没有具体内容。然后定义了一个函数 addButtons，它的作用就是将一些 mc 或者 sprites 放到自己的 container 里（addChild）。还有一个类：DisablingButton，也是位于 todd->interactive 文件夹下，这也是这个案例的核心。对了，之前已经将 RectButton 的 linkage 里的 baseClass 设置为 todd.interactive.DisablingButton。代码稍微有点长，且听我细细道来:

```
package todd.interactive{
    import flash.display.*;
    import flash.events.*;
    import todd.interactive.ButtonSet;
    public class DisablingButton extends MovieClip {
        var labels:Array;
        var thisParent:*;
        var thisIndex:int;
        public function DisablingButton() {
            labels = this.currentLabels;
            this.addEventListener(MouseEvent.CLICK, disableButton);
            this.addEventListener(MouseEvent.ROLL_OVER, over);
            this.addEventListener(MouseEvent.ROLL_OUT, out);
            this.addEventListener(Event.ADDED,setParent);
        }
        function disableButton(event:MouseEvent):void {
            for (var i:int = 0; i < labels.length; i++) {
                if (labels[i].name == "disable") {
                    this.gotoAndPlay("disable");
                }
            }
            this.removeEventListener(MouseEvent.CLICK, disableButton);
            this.removeEventListener(MouseEvent.ROLL_OVER, over);
            this.removeEventListener(MouseEvent.ROLL_OUT, out);
            enableOthers();
```

```
        }
        function enableButton():void {
            this.addEventListener(MouseEvent.CLICK, disableButton);
            this.addEventListener(MouseEvent.ROLL_OVER, over);
            this.addEventListener(MouseEvent.ROLL_OUT, out);
            this.gotoAndStop(1);
        }
        function over(event:MouseEvent):void {
            for (var j:int = 0; j < labels.length; j++) {
                if (labels[j].name == "over") {
                    this.gotoAndPlay("over");
                }
            }
        }
        function out(event:MouseEvent):void {
            for (var k:int = 0; k < labels.length; k++) {
                if (labels[k].name == "out") {
                    this.gotoAndPlay("out");
                }
            }
        }
        function setParent(event:Event):void {
            if (this.parent is ButtonSet) {
                thisParent=this.parent;
                for (var w:int=0; w < thisParent.buttons.length; w++) {
                    if (this == thisParent.buttons[w]) {
                        thisIndex=w;
                    }
                }
            }
        }
        function enableOthers():void {
            for (var z:int=0; z < thisParent.buttons.length; z++) {
                if (z != thisIndex) {
                    thisParent.buttons[z].enableButton();
                }
            }
        }
    }
}
```

载入了两个常用类后，又载入了刚刚定义的 ButtonSet 类，这样我们就能使用 ButtonSet 的一些方法了。
注意：这个类必须继承 Movieclip 类，因为该类的对象是一个 mc。然后定义了一些全局变量（默认均为 public）。创建析构函数 DisablingButton，labels = this.currentLabels; 这句话的意思是取得当前 mc 的 label 属性，以 array 的形式返回，包含了 label.frame，label.name 等等的属性。然后监听自己的鼠标点击、移入、移出事件。

this.addEventListener(Event.ADDED,setParent);这句话的意思是当自己被添加进一个容器时调用 setParent 函数。disableButton 这个函数作用是，将当前 mc 的状态变成 disabled，然后取消监听事件，同时激活其他的按钮。 enableButton 函数的作用就是激活自己的监听事件，并初始化自己的状态。over 和 out 函数很简单，就是设置自己当前的状态。setParent 函数的最终目的是捕获点击事件发生在哪个 mc 上（gotoAndPlay 方法将触发 EVENT.ADDED，所以 over 和 out 函数都将触发 setParent 函数，这也是一个待改进的地方）。enableOthers 函数顾名思义，激活其他的按钮。因为 setParent 已经记住了，最后的点击事件发生在哪个 mc 上，所以只要遍历一下 buttons，然后激活其他的 mc 就可以了。

六、AS3 中的拖动及碰撞检测

没有 press 和 release 事件 hitTest()被分尸了

```
var check_mc=new Sprite();
this.addChild(check_mc) ;
// check_mc.addEventListener(Event.ENTER_FRAME,checkFunc)
function checkFunc(evt:Event)
{
//检测对象 /*
if(mc1.hitTestObject(mc2))
{
   trace(“true“); };*/
   //检测坐标
   if(mc1.hitTestPoint(this.mouseX,this.mouseY,false))
   { trace(“true“) ;}
}
addList(mc1);
addList(mc2) ;
//增加事件侦听器
function addList(mc)
{
   mc.addEventListener(“mouseDown“,drag) ;
   mc.addEventListener(“mouseUp“,drag) ;
}
//拖动
function drag(evt:MouseEvent)
{
   var obj=evt.target ;
   var evtType=evt.type ;
   switch(evtType)
   {
      case “mouseDown“:
         obj.startDrag() ;
         break ;
      case “mouseUp“:
      obj.stopDrag() ;
      break ;
   }
}
```

七、如何用 AS 代码隐藏 Flash 的右键菜单

经常看到 Flash 的右键菜单中只显示了很少的几个菜单项，其实实现这个效果并不难，只要几句 AS 就能搞定。

第一种写法:

```
var my_cm:ContextMenu = new ContextMenu();//新建一个菜单对象
my_cm.hideBuiltInItems();//新建一个菜单对象的内容隐藏
my_cm.builtInItems.print = true;//如果你想要某个或某几个出现,可以这样设置,这里设置的仅显示"打印"
this.menu = my_cm;//将菜单附加到对象
```

第二种写法:

```
var my_cm:ContextMenu = new ContextMenu();//新建一个菜单对象
my_cm.builtInItems.print = false;//要删除的菜单项设置为 false
my_cm.builtInItems.quality=false;
my_cm.builtInItems.zoom=false;
my_cm.builtInItems.forward_back=false;
this.menu = my_cm;//将菜单附加到对象
```

下面是可控制的菜单项目:

zoom:缩放
quality:显示质量
play:播放
loop:循环
rewind:后退原文链接

forward_back:快进/返回
print:打印