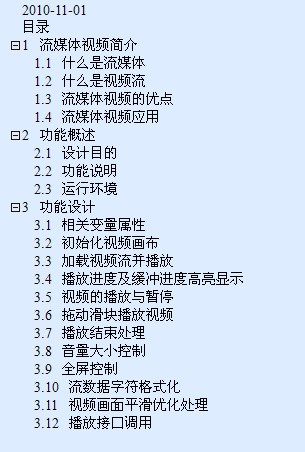
[**FLEX4.0开发流媒体视频播放器(总结篇)**](http://blog.csdn.net/bychentufeiyang/article/details/7910730)

原文连接 <http://www.cnblogs.com/peaceli/archive/2010/11/08/flv_videoplayer.html>

     视频播放器这个做开发的兄弟们应该都熟悉，现在的视听网站这么火热，流媒体技术也相当成熟，网上的介绍也很多。不过基本上流媒体播放器都是flash，大 多是写AactionScript脚本开发，利用FLEX开发也比较方便，和VS.NET的开发环境一样，看起来也熟悉，开发起来也有感觉些，我想至少应 该比写AS脚本要有感觉些，当然FLEX开发的应用程序最终也会被解析成AS脚本，生成SWF文件，供WEB页面嵌入调用。最近稍微空闲那么一点，从同事 那拷了个FLEX4.0，装上感觉下，之前也有意想做个流媒体播放器，正好这段时间有研究下。严格来说单纯的播放器不会难做，网上也有很多例子，只是流媒 体播放器只支持流格式的媒体文件，所有这里还有个媒体格式转换的问题，就是要把不同类型的视频格式转换成流格式的文件，即转换成视频流格式。我写的这篇暂 时只说说流媒体播放器的开发，不涉及转换的问题，这个有时间研究下再说说。

     利用FLEX开发视频播放器可以利用自带的组件VideoDisplay，也可以不用，最终两种方式开发出来的效果是可以一样的。我下面说的主要是不利用 组件的开发方式，利用组件开发的后面我也贴下代码片段和截图。主要以开发完后的开发文档内容来说下，写的内容不多，基本上要点应该还是写清楚了，先看下目 录，按目录顺序讲解：



# 1 流媒体视频简介

## 1.1  什么是流媒体

所谓流媒体是指采用流式传输方式在Internet上播放的媒体格式(扩展名一般为.flv，目前 Adobe公司为迎接高清时代又推出了.f4v格式)。流媒体又称流式媒体，是指用一个视频传输服务器把把节目当成数据包发出，传送到网络上，同过流媒体播放器进行画面还原显示给用户观看。

## 1.2 什么是视频流

视频流（Video Streaming）是指视频数据的传输，例如，它能够被作为一个稳定的和连续的流通过网络处理。因为流动，客户机浏览器或插件能够在整个文件被传输完成前显示多媒体数据。视频流技术基于 2 密钥技术，视频译码技术和可升级的视频分发技术发展。

## 1.3 流媒体视频的优点

流 媒体视频是边下载边播放边缓冲的，用户体验相比传统的下载播放好得多，传统的下载播放是用户等视频文件全部下载到缓存后再进行播放，用户等待时间比较长， 因此与单纯的下载播放方式相比，这种对多媒体文件边下载边播放的流式传输方式不仅使启动延时大幅度地缩短，而且对系统缓存容量的需求也大大降低。

## 1.4 流媒体视频应用

由于流媒体格式的数据传输速度快，因此被广泛应用于互联网上的大型视频点播网站，比较典型的视频点播网站又56视频网，优酷视频网、土豆视频网等。同时这种技术也可以用于在线视频教学系统进行点播学习或公司内部会议视频在线提供观看等。

补 充说明下那个高清格式的.f4v，因为测试的时候从土豆上拉了个高清格式的视频文件下来，发现后缀是.f4v的，利用开发的这个播放器播放不了，后缀名改 成flv是可以播放的，觉得奇怪，都是流格式应该是可以播放的才对，原来忘了在网站的http头文件的MIME类型添加扩展名,添加下.f4v的扩展名就 OK了，可以正常播放了，后来也看了下百度百科对这种高清格式是这么说的：

    作为一种更小更清晰，更利于在网络传播的格式，F4V已经逐渐取代了传统FLV，也已经被大多数主流播放器兼容播放，而不需要通过转换等复杂的方式。

　　F4V是Adobe公司为了迎接高清时代而推出继FLV格式后的支持H.264的F4V流媒体格式。

　　它和FLV主要的区别在于，FLV格式采用的是H263编码，而F4V则支持H.264编码的高清晰视频，码率最高可达50Mbps。

## 2 功能概述

## 2.1        设计目的

作为系统平台功能的扩充，开发此模块，现实一个功能相对全面的流媒体播放器，主要用于播放流媒体格式的视频，提供给特殊要求的客户用于外网在线播放新闻视频。

## 2.2        功能说明

1）  视频的暂停、播放

2）  视频拖动播放和定点播放

3）  音量的禁音和开启

4）  拖动滑块控制音量

5）  视频缓冲进度高亮显示

6）  视频全屏处理，点击按钮或点击视频画面实现全屏

## 2.3        运行环境

此模块采用Flex4.0开发，要求客户端安装flash 10.0.0 （含以上版本）

# 3 功能设计

## 3.1        相关变量属性

**private** **var** isPause:Boolean = **false**; //暂停状态

**private** **var** isSound:Boolean = **true**; //声音状态（是否禁音）

**private** **var** \_volume :Number = 0.6; //默认音量大小(最大值为1)

**private** **var** isFullScreen:Boolean = **false**; //是否是全屏

**private** **var** totalTime:Number; //播放总时间

**private** **var** playPosition:Number; //剪辑位置

**private** **var** videoUrl:String; //视频文件地址

**private** **var** videoWidth:Number; //视频宽度

**private** **var** videoHeight:Number; //视频高度

**private** **var** isAutoPlay:Boolean = **true**; //是否自动播放

## 3.2        初始化视频画布

点开视频播放页面后首先初始化视频播放的画面，根据接收的用户参数初始化视频画面的大小。

对象定义：

**import** mx.events.SliderEvent; //滑块事件命名空间引用

**private** **var** nc:NetConnection; //媒体连接对象

**private** **var** ns:NetStream; //网络流对象

**private** **var** metaDataObj:Object = {}; //媒体的元数据信息

**private** **var** video:Video ; //视频对象

初始化方法如下：

**private** **function** init():**void**

{

videoUrl = parameters.videoUrl;

videoWidth = parameters.videoWidth;

videoHeight = parameters.videoHeight

                video = **new** Video(videoWidth,videoHeight);

video.smoothing = **true**;//画面平滑处理，去掉全屏后的水纹以提高画面清晰度

uic.addChild(video); //将视频对象添加到页面

}

## 3.3        加载视频流并播放

当视频初始化完成后调用视频播放方法，将方法置于应用程序事件头里面。

<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955" minHeight="600" initialize="init()" creationComplete="startVideo()" >

**private** **function** startVideo():**void**

{

nc = **new** NetConnection();

nc.addEventListener(NetStatusEvent.NET\_STATUS,netStatusHandler); //添加播放连接监听事件

nc.connect(**null**);

}

当连接对象成功连接后，播放视频，上面的nc.connect(**null**);表示如果未使用 Flash Media Server,可使用null作为参数以便从本地文件系统或 Web 服务器中播放视频和 MP3 文件。

**private** **function** netStatusHandler(e:NetStatusEvent):**void**

{

ns = **new** NetStream(nc);

metaDataObj.onMetaData = **this**.onMetaData;

ns.client = metaDataObj;

video.attachNetStream(ns);

//ns.bufferTime = 5;

ns.play(videoUrl);

soundProcess.value = \_volume;

soundTrans.volume = \_volume

ns.soundTransform = soundTrans;

**this**.addEventListener(Event.ENTER\_FRAME,EnterFrameHandler); //添加播放过程中的监听事件

ns.addEventListener(NetStatusEvent.NET\_STATUS,NetStreamStatusHandler);//添加播放完毕(或其它状态)后的监听事件

**if**(!isAutoPlay) //客户端没有设置为自动播放时的处理

{

ns.pause();

btnPlay.source = playClass;

isPause = **true**;

}

**else**

{

btnPlay.source = pauseClass;

isPause = **false**;

}

}

//获取视频的元数据信息，这里的元数据信息包括视频编码，视频码率，音频编码，音频码率，音视频文件大小，流文件总大小，播放总时间等

**private** **function** onMetaData(obj:Object):**void**

{

totalTime = obj.duration;

fileSize = (obj.filesize/(1024\*1024)).toFixed(2).toString()+**"MB"**;//换算成兆字节并保留两位小数

}

**说明：**

**this**.addEventListener(Event.ENTER\_FRAME,EnterFrameHandler);用于监听播放过程中的事件处理，由于在播放过程中，播放进度和缓冲进度要实时显示以及播放到哪一个时间点了也都需要动态实时呈现给用户，因此视频进入加载画面时就需要实时不断监控该事件。

ns.addEventListener(NetStatusEvent.NET\_STATUS,NetStreamStatusHandler);用于监听在视频流播放完毕后的事件处理。

onMetaData是一个回调方法，当客户端加载到视频流时，通过异步获取该媒体的元数据信息，如媒体总大小、总的播放时间、采样率等等。

## 3.4        播放进度及缓冲进度高亮显示

//播放进度和缓冲进度处理

**private** **function** EnterFrameHandler(e:Event):**void**

{

**if** (totalTime>0)

{

playTime = ns.time;// ns.time为流媒体实时播放的时间

}

**if** (ns.bytesLoaded>0)

{

bufferRect.width = ns.bytesLoaded / ns.bytesTotal\*(playProcess.width-10);//计算缓冲方框的宽度(滑块本身也有一定的宽度,减去约10个像素宽度)

}

}

说明：

playTime作为播放进度条当前实时播放的时间点，视频的总时间作为播放显示进度条的最大值。

<mx:HSlider id="playProcess"minimum="0" value="{playTime}" maximum="{totalTime}"

ns.bytesLoaded为已缓冲好的流媒体字节大小（单位为byte），ns.bytesTotal为流媒体的总大小，通过比例计算（如上）可以得出缓冲进度在播放进度条上的位置，缓冲进度条其实是一个长方形框，以层的形式位于播放进度条下放，初始宽度为0,当缓冲达到100%时，即缓冲完毕时，缓冲条的长度和播放进度条等长（除去滑块宽度）。缓冲方框可以是一个BorderContainer，如下：

<s:BorderContainer x="14" y="411" width="0" height="4" id="bufferRect" buttonMode="true" borderColor="#70b2ee" backgroundColor="#70b2ee">

</s:BorderContainer>

页面所有的控件和标签如下：



<mx:Image source="{playClass}" click="play();" id="btnPlay" buttonMode="true" x="16" y="423"/>    
    <mx:UIComponent id="uic" height="400" width="640" click="play()" doubleClickEnabled="true" doubleClick="display()" buttonMode="true" y="0" x="0"/>   
    <s:BorderContainer x="14" y="411" width="0" height="4" id="bufferRect" buttonMode="true" borderColor="#70b2ee" backgroundColor="#70b2ee">  
    </s:BorderContainer>  
    <mx:Label text="{formatTime(playTime)}/{formatTime(totalTime)}                    {fileSize}" width="150" x="60" y="427"/>  
    <mx:HSlider id="playProcess"  minimum="0" value="{playTime}" maximum="{totalTime}" change="play\_onchange(event)" thumbPress="thumbPress();"  thumbRelease="thumbRelease();"  
                alpha="0.5"    dataTipFormatFunction="dataTipFormat" buttonMode="true" showTrackHighlight="true" x="9" y="398" width="630"/>  
      
    <mx:Image source="{sound1}" click="closeSound();" id="soundImg" buttonMode="true" x="364" y="421"/>  
    <mx:HSlider id="soundProcess" x="406" y="420"  minimum="0" maximum="1" change="sound\_thumbChanges(event)"    
                showTrackHighlight="true" dataTipFormatFunction="SoundTipFormat"   buttonMode="true"  width="135"/>    
    <mx:Button label="全屏" click="display();" buttonMode="true" cornerRadius="20" labelPlacement="right" paddingLeft="6" x="561" y="423"/>  
    <mx:SWFLoader id="load" x="16" y="486"/>



## 3.5        视频的播放与暂停

视频的暂停与播放调用视频流的pause()方法和resume()方法，通过是否暂停的状态变量判断控制，代码片段如下：

//播放、暂停设置

**private** **function** play():**void**

{

**if**(isPause)

{

ns.resume();

btnPlay.source = pauseClass; //设置按钮图标为点击暂停图标

isPause = **false**;

}

**else**

{

ns.pause();

btnPlay.source = playClass; //设置按钮图标为点击播放图标

isPause = **true**;

}

}

## 3.6        拖动滑块播放视频

拖动滑块播放视频文件，主要是判断和记录流的剪辑位置，找到最终的剪辑位置后可以调用视频流的seek(参数)方法，参数为当前的剪辑位置，如果不拖动直接点击任意剪辑位置定点播放，那么最终的剪辑位置应该是鼠标弹起的位置，这里定点点击实际上还是相当于触发了滑块移动的事件，只不过是滑块快速移动到你点击的位置而已，相关代码片段如下：

//拖动进度条时改变播放位置

**private** **function** play\_onchange(event:SliderEvent):**void**

{

**if**(ns.time == 0)

{

playProcess.value = 0;

**return**;

}

playPosition = playProcess.value; //保正播放进度統一

ns.seek(playPosition);

}

//进度条鼠标按下

**private** **function** thumbPress():**void**

{

ns.pause();

}

//进度条鼠标弹起,指拖动时滑块时鼠标弹起

**private** **function** thumbRelease():**void**

{

//ns.seek(playPosition);

btnPlay.source = pauseClass;

isPause = **false**;

ns.resume();

}

## 3.7        播放结束处理

一般视频播放完毕后，播放的指针头归零，即播放进度条上的滑块指向起始位置，同时播放按钮状态为准备就绪状态，视频流处于暂停状态。可以通过视频流的当前状态信息进行判断，如下面e.info.code状态值可以获得各种不同的状态，这里只取播放完毕停止后的状态值，代码片段如下：

//播放完毕处理

**private** **function** NetStreamStatusHandler(e:NetStatusEvent):**void**

{

**if**(e.info.code == **"NetStream.Play.Stop"**)

{

ns.seek(0);

btnPlay.source = playClass;

isPause = **true**;

ns.pause();

}

}

## 3.8        音量大小控制

视频声音控制通过SoundTransform 类操作，该类包含音量和平移的属性。如果禁音后运行调节滑块的话，需要再定义一个临时变量tmpSound，以便开启声音时为最终设置的音量。

//静音、开音控制

**private** **function** closeSound():**void**

{

**if**(isSound)

{

soundImg.source = sound;

tmpSound= ns.soundTransform;

soundTrans.volume = 0;

ns.soundTransform = soundTrans; // 这里禁音直接ns.soundTransform.volume = 0 这样不行，需要用对象赋值

isSound = **false**;

}**else**

{

soundImg.source = sound1;

ns.soundTransform = tmpSound;

isSound = **true**;

}

}

//通过滑块调整声音

**private** **function** sound\_thumbChanges(event:SliderEvent):**void**

{

tmpSound.volume = soundProcess.value;

ns.soundTransform = tmpSound;

}

## 3.9        全屏控制

全屏可以用flash画布舞台的stage属性值设置，不过要考虑下普通屏幕和宽屏的处理, 常见的显示器分辨率按其长宽比可分为为：4：3（1024×768）、5：4（1280×1024）、16：9、16：10 (这里暂以宽屏测试的，需要后续处理)，点击全屏按钮或点击视频画面都可以全屏，调用display()方法即可，代码片段如下：

//切換全屏显示

**private** **function** display():**void**

{

**if**(!isFullScreen)

{

stage.fullScreenSourceRect = **new** Rectangle(video.x, video.y,video.width,video.height);

stage.displayState =StageDisplayState.FULL\_SCREEN;

isFullScreen = **true**;

}**else**

{

stage.displayState = StageDisplayState.NORMAL;

isFullScreen = **false**;

}

}

## 3.10   流数据字符格式化

视频播放时当前时间和总时间是以秒为单位的，比如180s的文件，当前播放到一半显示90s，需要按时间格式来显示才友好，另外还有音量的值是介于0到1之间的某个值，也需要按百分比来显示。代码片段如下：

//格式化时间

**private** **function** formatTime(time:Number):String

{

**var** min:Number = Math.floor(time/60);

**var** sec:Number = Math.floor(time%60);

**var** timeResult:String = (min < 10 ? **"0"**+min.toString() : min.toString()) + **":"** + (sec == 10 ? **"0"**+sec.toString() : sec.toString());

**return** timeResult;

}

//声音slider格式化

**private** **function** SoundTipFormat(data:Number):String

{

**return** (data\*100).toFixed(0).toString()+**"%"**; //拖动声音进度条时显示百分比音量

}

## 3.11   视频画面平滑优化处理

一般视频画布全屏后，会产生文字图像有些失真的感觉，会产生水纹，对于这点的处理，Flex封装了一个简单有效的方法，只需设置一个属性即可，即video对象中有一个属性设置。如下：

video.smoothing = **true**;

把该属性设置为true，表是启用画面优化处理，加上这个设置能大大提高画面质量 。

## 3.12   播放接口调用

采用FLEX开放的播放器编译后最终生成的是一个.swf文件，需要通过页面去加载调用，可以是静态的html页面，也可以是动态的aspx页面，调用过程中引用Flex内置的一个JS文件——swfobject.js，

<script type="text/javascript" src="swfobject.js"></script>

<script type="text/javascript">

var swfVersionStr = "10.0.0";

var xiSwfUrlStr = "playerProductInstall.swf";

var flashvars = {};

flashvars.videoUrl = "S6000103.flv";

flashvars.videoWidth = "640";

flashvars.videoHeight = "400";

flashvars.videoVolumn = "0.6";

flashvars.isAutoPlay = "true";

var params = {};

params.quality = "high";

params.bgcolor = "#ffffff";

params.allowscriptaccess = "sameDomain";

params.allowfullscreen = "true";

var attributes = {};

attributes.id = "FLVPlayer";

attributes.name = "FLVPlayer";

attributes.align = "middle";

swfobject.embedSWF(

"FLVPlayer.swf", "flashContent",

"650", "450",

swfVersionStr, xiSwfUrlStr,

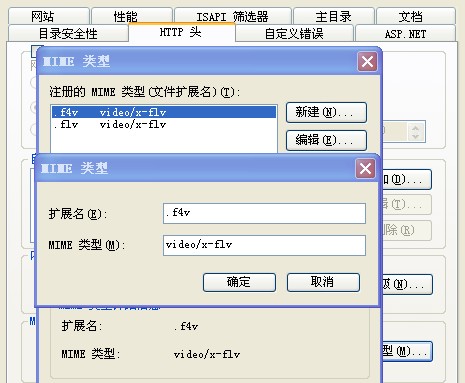
flashvars, params, attributes);

</script>

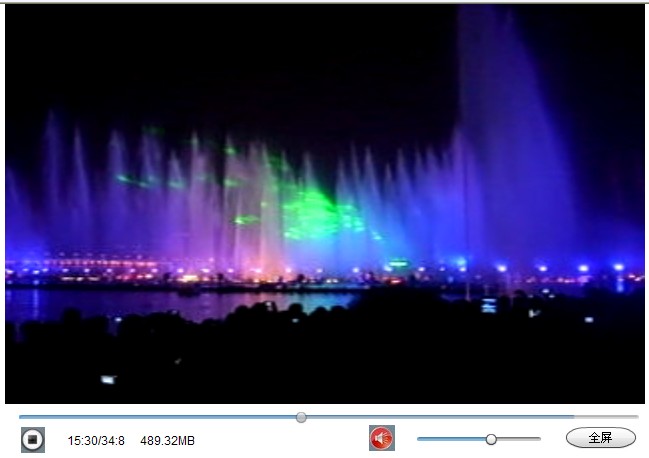
<div id="flashContent">

</div>

其中flashContent标签作为嵌入的swf播放器文件（FLVPlayer.swf）的容器，同时网站的http头进行如下设置（添加.flv和.f4v扩展名）：



下面贴几张播放的效果图：



让美工美化下，做一个像目前视频网类似效果的播放器还是相当简单的。

利用VideoDisplay组件开发的我贴下完整的代码片段，这里和上面对比少了缓冲进度的效果，另外没有画面优化的效果出来，如果要加这两个效果的话，要向上面那样定义连接，输入流对象，然后让这个组件加载实例化的video对象就可以了，代码如下：



<?xml version="1.0" encoding="utf-8"?>  
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"   
               xmlns:s="library://ns.adobe.com/flex/spark"   
               xmlns:mx="library://ns.adobe.com/flex/mx" fontSize="12" initialize="init();" backgroundAlpha="0.6" backgroundColor="#FFFFFF">  
  
    <fx:Declarations>  
        <!-- 将非可视元素（例如服务、值对象）放在此处 -->  
    </fx:Declarations>  
  
    <fx:Script>    
        <![CDATA[  
            import mx.controls.Alert;  
            import mx.events.MetadataEvent;  
            import mx.events.SliderEvent;  
            import mx.controls.sliderClasses.Slider;   
            import mx.events.VideoEvent;  
            import mx.utils.ObjectUtil;  
            import mx.controls.ProgressBarDirection;  
  
        [Embed(source="images/play.png")]    
        [Bindable]    
        private var playClass:Class;                //播放图标样式    
          
        [Embed(source="images/pause.png")]    
        [Bindable]    
        private var pauseClass:Class;               //暂停图标样式    
          
        [Embed(source="images/sound1.png")]    
        [Bindable]    
        private var sound1:Class;               //声音样式1    
          
        [Embed(source="images/sound.png")]    
        [Bindable]    
        private var sound:Class;                //声音样式2（静音）    
          
        [Bindable]    
        private var videoSource:String;         //媒体路径    
          
        private var isPause:Boolean = false;        //暂停状态    
        private var isSound:Boolean = true;         //声音状态    
        private var isFullScreen:Boolean = false;   //是否是全屏    
        private var tmpSound:Number = 0;            //临时声音大小    
              
        [Bindable]    
        private var fileSize:String;                 //视频文件大小  
              
        [Bindable]    
        private var playPosition:Number;            //播放进度     
          
        private function init():void{    
            videoSource = parameters.videosource;    
        }    
          
        private function playingMove(event:VideoEvent):void{    
            my\_hs.value = myVid.playheadTime;  //视频播放时同步进度条状态值  
              
        }    
  
        //拖动进度条时改变播放位置    
        private function hs\_onchange(event:SliderEvent):void{    
            if(myVid.playheadTime == -1){    
                my\_hs.value = 0;    
                return;    
            }    
            playPosition = my\_hs.value;             //保正播放进度統一    
            myVid.playheadTime = playPosition;   
        }    
          
        //进度条鼠标按下    
        private function thumbPress():void{    
            myVid.pause();    
        }    
        //进度条鼠标弹起    
        private function thumbRelease():void{    
            myVid.playheadTime = playPosition;    
            myVid.play();    
        }    
          
          
        //播放，暂停    
        private function playButton():void{    
            if(!isPause){    
                myVid.play();    
                playBtn.source = pauseClass;    
                isPause = true;    
            }else{    
                myVid.pause();    
                playBtn.source = playClass;    
                isPause = false;    
            }    
        }    
          
        //播放完成    
        private function vidComplete():void{    
            playBtn.source = playClass;    
            isPause = false;    
        }    
          
        //停止播放    
        private function stopButton():void{    
            myVid.stop();    
            playBtn.source = playClass;    
            isPause = false;    
        }    
          
        //切換全屏顯示    
        private function display():void{    
            if(!isFullScreen){    
                stage.fullScreenSourceRect = new Rectangle(myVid.x,myVid.y,myVid.width,myVid.height);  
                stage.displayState =StageDisplayState.FULL\_SCREEN;    
                isFullScreen = true;    
            }else{    
                stage.displayState = StageDisplayState.NORMAL;    
                isFullScreen = false;    
            }    
        }    
          
        //调整声音    
        private function sound\_thumbChanges(event:SliderEvent):void{    
            myVid.volume = hs\_sound.value;    
        }    
          
        //静音    
        private function closeSound():void{    
            if(isSound){    
                closeImg.source = sound;    
                tmpSound = myVid.volume;    
                myVid.volume = 0;    
                isSound = false;    
            }else{    
                closeImg.source = sound1;    
                myVid.volume = tmpSound;    
                isSound = true;    
            }    
        }    
          
  
        //格式化时间    
        private function formatTime(time:Number):String{    
            var min:Number = Math.floor(time/60);    
            var sec:Number = Math.floor(time%60);    
        var timeResult:String = (min < 10 ? "0"+min.toString() : min.toString()) + ":" + (sec == 10 ? "0"+sec.toString() : sec.toString());    
        return timeResult;    
        }    
        //slider格式化    
        private function dataTipFormat(data:Number):String{    
            return formatTime(data);    
        }    
               
              
        private function myVid\_metadataReceived(evt:MetadataEvent):void {  
  
            var meta:Object = evt.info; // 视频的元数据信息  
            fileSize = (meta.filesize/(1024\*1024)).toFixed(2).toString()+"MB";//换算成兆字节并保留两位小数   
  
            /\* 读取所有元数据属性和值  
            var i:int = 0;  
            var arr:Array = [];  
            var item:String;  
             var value:\*;  
            for (item in meta) {  
                if (ObjectUtil.isSimple(meta[item])) {  
                    if (meta[item] is Array) {  
                        value = "[Array]";  
                    } else {  
                        value = meta[item]  
                    }  
                      
                    Alert.show(item+":"+value);      
                }  
                  
            }\*/  
              
  
        }  
              
  
              
                 
     ]]>     
          
          
  
    </fx:Script>    
      
    <s:BorderContainer borderColor="#66ccff" y="0" cornerRadius="0" borderWeight="0" borderVisible="true" dropShadowVisible="false" x="0" width="660">  
          
        <mx:VideoDisplay id="myVid" y="10" height="400" width="640"  source="{videoSource}"  autoPlay="false"  buttonMode="true"  click="playButton();"  ready="myVid.visible = true;" metadataReceived="myVid\_metadataReceived(event);"  
          playheadUpdate="playingMove(event)"  complete="vidComplete();"  doubleClickEnabled="true"  doubleClick="display();" contentBackgroundAlpha="1.0" contentBackgroundColor="#000000" x="10"  
   
        />    
          
        <mx:HBox width="640" verticalAlign="middle" x="10" y="415" height="90">    
            <mx:Image source="{playClass}" click="playButton();" id="playBtn" buttonMode="true"/>    
            <mx:Label text="{formatTime(myVid.playheadTime)}/{formatTime(myVid.totalTime)}                    {fileSize}" width="150"/>  
            <mx:HRule height="0" width="200" buttonMode="true"/>   
            <mx:Image source="{sound1}" click="closeSound();" id="closeImg" buttonMode="true"/>    
            <mx:HSlider width="100" id="hs\_sound" minimum="0" maximum="1"      
                        change="sound\_thumbChanges(event)"    
                        value="{myVid.volume}" buttonMode="true" />    
            <mx:Button label="全屏" click="display();" buttonMode="true" cornerRadius="20" labelPlacement="right" paddingLeft="6"/>    
        </mx:HBox>    
        <mx:HSlider width="640" id="my\_hs" minimum="0" maximum="{myVid.totalTime}" height="10" showTrackHighlight="true" buttonMode="true" liveDragging="true"  
                    change="hs\_onchange(event)" thumbPress="thumbPress();"  thumbRelease="thumbRelease();" x="10" y="422"  dataTipFormatFunction="dataTipFormat" />   
  
  
          
    </s:BorderContainer>  
</s:Application>



效果图如下：



可以对比下平滑处理过的效果，明显文字更清晰些，尤其在全屏后效果更明显，平滑处理后基本不会有画面失真的现象。



说明：采用的是FLEX4.0开发的，其实里面的标签有还是3.0的，直接参考了网上的一些例子，也都是兼容的，就没改了。

主要参考了<http://www.cnblogs.com/yjmyzz/archive/2010/04/16/1713290.html>（杨过兄的这篇），基本上大同小异吧。

原 本我也想放个可以运行的demo上来，发现编译发布后还带有好几个swf的类库文件，没有的话播放不了，都是flex4.0的发布文件必须要的，想发布后 打包成一个单独的swf文件，也没有找到合适的方法，也就没有进一步处理了，如果大家有兴趣想玩下或研究下，可以装个FLEX4.0版本的，新建一个应用 程序，安照上面的步骤做个可以播放流媒体视频的播放器是没问题的。

**(2010-11-11 界面排了下版，排版后的效果如下)**



**（以上内容仅供学习参考，本人保留最终解释权）**