
Online
GAME
www.TTNo1.com

Macromedia Flash Media Server 2

帮助 中文版-译本

说明：因为翻译制作的时间较短，所以里面有部份图片没放上，还有些格式没调整，但很多 Flash 开发者们早就迫切的想要这份中文版，所以先提早放出了，而不完善的地方会在今后完善，请大家关注，在之后还会制作成帮助文档便于大家使用，现在就先用这个应付一下吧！谢谢您的支持和谅解！

云南 TT 概念游戏公司整理制作

——全国第一的 Flash 网络游戏专家——

www.TTNo1.com

特别感谢：

Macromedia Flash Media Server 2 开发小组的英文原版
以及在翻译及整理过程中给于帮助所有 Flash 开发者们

特别通告：

TTGame 诚邀各位 Flash 爱好者、开发者们的加入！
希望你与我们一起开创一片 Flash 的新天地！
为不断增强团队实力，实现更大更远的目标！

特招聘：

Flash 游戏开发者
Flash 应用开发者
Flash 动画高手
Flash 艺术设计
及 Flash 各路高手

希望你积极与我们取得联系
不论你是想要加入我们、还是想要与我们交流、
还是因为有共同的目标、还是因为想与我们合作
还是、还是、还是、、、、、、、
都希望你通过以下联系方式联系我们

联系方式：
Email：ynttgame@163.com
QQ: 50955577

《服务器端ActionScript语言参考》

服务器端通讯ActionScript是一种服务器上的脚本语言，它可以让你开发高效和可伸缩的客户机/服务器Macromedia Flash Media Server应用程序。例如，你可以使用服务器端ActionScript来控制登录过程，在连接的Macromedia Flash应用程序中控制事件，决定用户可以在他们的Flash应用程序中看到什么，以及与其他服务器进行通讯。你也可以使用服务器端脚本来允许和禁止用户访问多种服务器端应用程序资源，并允许用户更新和共享信息。

服务器端ActionScript是基于ECMA-262规范的（ECMAScript 1.5），该规范是源自JavaScript，并可以让你访问核心JavaScript服务器对象模型。服务器端ActionScript为开发通讯应用程序提供了全局方法和类，并展示了一个丰富的对象模型。你也可以创建你自己的类、属性，以及方法。这个字典提供了有关类，及其属性、方法和事件的详细信息。

客户机端通讯ActionScript是基于ECMA-262规范的，但在其执行时有一些不同。但不管怎么说，服务器端ActionScript，没有偏离ECMA-262规范。

使用服务器端ActionScript

要在一个Flash Media Server应用程序中使用服务器端ActionScript，你需要编写代码，把脚本拷贝到适当的服务器目录中，并运行那个SWF文件以连接到服务器。

创建服务器端ActionScript并将其命名为main.asc。所有那些被植入这个脚本文件但并未处于某个函数体中的ActionScript代码都会在这个应用程序被装载但application.onAppStart事件还未被调用之前被执行一次。

注意：你也可以把你的服务器端脚本文件命名为app_name，在这里，app_name是你的应用程序的目录的名字，并以.asc或.js作为文件扩展名来保存它。同样，服务器端ActionScript文件中的任何双字节字符（包括所有亚洲语言的字符）必须以UTF-8进行编码。

注意：对于大规模的应用程序，你可能需要使用多个服务器端脚本文件。你可以使用Flash Media Server文档编译器实用工具来把这些文件作为来自一单一位置的包进行部署。要获得更多信息，参看《开发媒体应用程序》中的“归档和编译服务器端脚本文件”。

注意：服务器端脚本也有一个安全装载阶段，关键性的代码可以在主应用程序装载阶段之前利用这个安全装载阶段被装载。服务器端脚本也可以让你创建受保护对象，这些对象的方法和数据不能被检查和操作。这两种特性可以让你实现系统调载。服务器端脚本也可以让你创建受保护对象，这些对象的方法和数据不能被检查和操作。这两种特性可以让你实现系统调用，保护重要的数据和函数。要获得更多信息，参看《开发媒体应用程序》中的“实现安全系统对象”。

要安装和测试服务器端ActionScript文件，完成下列步骤：

1. 定位到Macromedia Flash Media Server/applications目录。

   默认的/applications目录的位置是在Macromedia Flash Media Server产品的安装目录下。

   注意：如果你没有接受默认的安装位置，并且你也无法确定这个applications目录位于哪儿，你可以在Vhost.xml文件的

AppsDir标签中找到这个目录的位置，这个Vhost.xml文件位于\Flash Media Server\conf_defaultRoot__defaultVhost_\中。你

的SWF和HTML文件应该被发布到一个Web服务器目录下，而你的服务器端ASC文件，你的音频/视频

FLV文件，以及你的

ActionScript FLA源文件都不应该被某个浏览你的网站的用户访问到。

2. 你的服务器端脚本文件必须被命名为main.asc、main.js、registered_app_name.asc，或registered_app_name.js。

3. 在/applications目录下创建一个子目录，将其命名为appName，在这里，appName是一个你选作你的Flash Media Server应

用程序的文件名的名字。你必须在客户机端ActionScript中把这个名字作为一个参数传递给NetConnection.connect方

法。

4. 把main.asc文件放置到这个appName目录中，或是appName目录中的一个名为scripts的子目录中。

5. 在一个浏览器或是在一个独立的Flash Player中打开这个Flash应用程序（这个SWF文件）。

这个SWF文件必须包含把appName传递给NetConnection类的connect方法的ActionScript代码，就像下面这样：

```
nc = new NetConnection();
nc.connect("rtmp://flashcomsvr.mydomain.com/myFlashComAppName");
```

注意：你可以使用管理控制台来检查这个应用程序是否成功的装载了。

使用命名约定

当你编写服务器端ActionScript代码时，在你命名你的应用程序、方法、属性，以及变量时，有一些命名约定你必须遵守。
这些规则可以让你有条理的识别对象，以便使你的代码能够正常的执行。

命名应用程序

Flash Media Server应用程序的命名必须遵守《统一资源标识符RFC2396》的约定。该约定支持一个层级命名系统，在这个
系统中，一个前斜杠（/）用来分隔层级体系中的元素。第一个元素指定了这个应用程序的名字，跟在应用程序名后的元素指定
了应用程序实例的名字。应用程序的每一个实例都有它自己的脚本环境。

指定实例

通过在一个应用程序名后指定一个唯一的应用程序实例名，你可以运行同一应用程序的多个实例。例如，
rtmp:/support/session215指定了一个名为“support”的客户支持应用程序，并引用了这个应用程序的一个名为“session215”的特定会话。通过引用同一个流或共享对象，连接到同一实例名的所有用户之间都可以进行通讯。

使用JavaScript语法

你必须遵循所有的JavaScript语法。例如，JavaScript是大小写敏感的，并且不允许在名字中出现除下划线（_）和美元符号（$）以外的其他标点符号。你可以在名字中使用数字，但名字不能以一个数字打头。

避免被保留的命令

Flash Media Server有一些保留命令，你不能在脚本中使用这些命令。这些命令或者是属于客户机端NetConnection类的方法，或者是属于服务器端Client类的方法。这意味着假如在客户机端你有一个NetConnection对象的话，则你就不能进行下面这样的调用：

nc.call("reservedCmd", ...);

在这个例子中，“reservedCmd”是下列这些命令：closeStream、connect、createStream、deleteStream、onStatus、pause、play、publish、receiveAudio、receiveVideo，或seek。它也不能是任何服务器端Client类的方法：getBandwidthLimit、setBandwidthLimit、getStats，以及ping。

ActionScript类

下表列出了服务器端ActionScript语言参考中的所有类。

ActionScript类	描述
Application类	Application类包含有关一个Flash Media Server应用程序实例的信息，它会一直维持这些信息直到这个应用程序实例被卸载。
Client类	Client类让你处理连接到一个Flash Media Server应用程序实例的每一个用户或说client。

File类	File类让应用程序写入服务器的文件系统。
LoadVars类	LoadVars类可以让你从某个远端或本地把变量装入一个服务器端脚本。
NetConnection类	服务器端NetConnection类可以让你在一个Flash Media Server应用程序实例和一个应用程序服务器、另一个Flash Media Server，或是同一台服务器上的另一个Flash Media Server应用程序实例之间创建一个双路连接。
SharedObject类	SharedObject类可以让你在多个客户应用程序间实时地共享数据。
SOAPCall类	SOAPCall类是从所有Web服务（Web Service）调用返回的对象类型。
SOAPFault类	SOAPFault类是返回到WebService.onFault和SOAPCall.onFault函数的错误对象的对象类型。
Stream类	Stream类可以让你处理一个Flash Media Server应用程序中的每一个流。
WebService类	WebService类可以让你创建和访问一个WSDL/SOAP Web服务。
XML类	XML类可以让你装载、解析、发送、建立，以及操作XML文档。
XMLSocket类	XMLSocket类实现客户机套接字，这可以让Flash Media Server和一个由一个IP地址或域名识别的服务器进行通讯。
XMLStream类	XMLStream类是一个XMLSocket类的变种。它拥有与XMLSocket类完全相同的方法、属性和事件，但它是以片段来传输和接收数据的。

下表列出了服务器端ActionScript语言参考中的所有全局函数。

全局函数	描述
clearInterval()	取消一个由对setInterval()方法的调用而设置的超时。
getGlobal()	当secure.asc文件被装载时，提供对这个全局对象的访问。
load()	在main.asc文件中装载一个ActionScript文件。
protectObject()	将用户定义或内建对象保护在一个C包装对象的后面。
setAttributes()	让你保护某些方法和属性避免被计算、写入和删除。
setInterval()	按照一个指定的时间间隔不断的调用一个函数或方法，直到clearInterval()方法被调用。
protectObject()	将用户定义或内建对象保护在一个C包装对象的后面。
setAttributes()	让你保护某些方法和属性避免被计算、写入和删除。
setInterval()	按照一个指定的时间间隔不断的调用一个函数或方法，直到clearInterval()方法被调用。
trace()	对一个表达式求值并显示这个值。

ActionScript元素

本文档中的条目是按类名、方法、属性或事件处理器的名称的字母顺序排序的。下表以字母顺序分别列出了所有的类、方法、属性和事件处理器。

ActionScript元素	查看条目
acceptConnection	Application.acceptConnection()
addHeader	NetConnection.addHeader()
addRequestHeader	LoadVars.addRequestHeader(), XML.addRequestHeader()
agent	Client.agent
allowDebug	Application.allowDebug
appendChild	XML.appendChild()
attributes	XML.attributes
autoCommit	SharedObject.autoCommit
broadcastMsg	Application.broadcastMsg()
bufferTime	Stream.bufferTime
call	Client.call(), NetConnection.call()
canAppend	File.canAppend
canRead	File.canRead
canReplace	File.canReplace
canWrite	File.canWrite
childNodes	XML.childNodes
clear	SharedObject.clear(), Stream.clear()
clearInterval	clearInterval()
clearSharedObject	Application.clearSharedObjects()
clearStreams	Application.clearStreams()
clients	Application.clients
cloneNode	XML.cloneNode()
close	NetConnection.close(), File.close(), SharedObject.close(), XMLSocket.close()
"commandName"	Client."commandName"
commit	SharedObject.commit()

config	Application.config
connect	NetConnection.connect(), XMLSocket.connect()
contentType	LoadVars.contentType, XML.contentType
copyTo	File.copyTo()
createElement	XML.createElement()
createTextNode	XML.createTextNode()
creationTime	File.creationTime
decode	LoadVars.decode()
detail	SOAPFault.detail
disconnect	Application.disconnect()
docTypeDecl	XML.docTypeDecl
eof	File.eof()
exists	File.exists
faultactor	SOAPFault.faultactor
faultcode	SOAPFault.faultcode
faultstring	SOAPFault.faultstring
firstChild	XML.firstChild
flush	File.flush(), SharedObject.flush(), Stream.flush()
gc	Application.gc()
get	SharedObject.get(), Stream.get()
getBandwidthLimit	Client.getBandwidthLimit()
getBytesLoaded	LoadVars.getBytesLoaded(), XML.getBytesLoaded()
getBytesTotal	LoadVars.getBytesTotal(), XML.getBytesTotal()
getNamespaceForPrefix	XML.getNamespaceForPrefix()
getPrefixForNamespace	XML.getPrefixForNamespace()

getProperty	SharedObject.getProperty()
getPropertyNames	SharedObject.getPropertyNames()
getStats	Application.getStats(), Client.getStats()
handlerName	SharedObject.handlerName
hasChildNodes	XML.hasChildNodes()

hostName	Application.hostname
ignoreWhite	XML.ignoreWhite
insertBefore	XML.insertBefore()
ip	Client.ip
isConnected	NetConnection.isConnected
isDirectory	File.isDirectory
isDirty	SharedObject.isDirty
isFile	File.isFile
isOpen	File.isOpen
lastChild	XML.lastChild
lastModified	File.lastModified
length	File.length, Stream.length()
list	File.list()
load	load(), LoadVars.load(), XML.load()
loaded	LoadVars.loaded, XML.loaded
localName	XML.localName
lock	SharedObject.lock()
mark	SharedObject.mark()
mkdir	File.mkdir()
mode	File.mode
name	Application.name, File.name, SharedObject.name, Stream.name
namespaceURI	XML.namespaceURI
nextSibling	XML.nextSibling
nodeName	XML.nodeName
nodeType	XML.nodeType
nodeValue	XML.nodeValue
onAppStart	Application.onAppStart
onAppStop	Application.onAppStop
onClose	XMLSocket.onClose

onConnect	Application.onConnect, XMLSocket.onConnect
onConnectAccept	Application.onConnectAccept
onConnectReject	Application.onConnectReject
onData	LoadVars.onData, XML.onData, XMLSocket.onData
onDisconnect	Application.onDisconnect
onFault	SOAPCall.onFault, WebService.onFault
onHTTPStatus	LoadVars.onHTTPStatus, XML.onHTTPStatus
onLoad	LoadVars.onLoad, WebService.onLoad, XML.onLoad
onLog	Log.onLog
onResult	SOAPCall.onResult
onStatus	Application.onStatus, NetConnection.onStatus, SharedObject.onStatus, Stream.onStatus
onSync	SharedObject.onSync
onXML	XMLSocket.onXML
open	File.open()
parentNode	XML.parentNode
parseXML	XML.parseXML()
ping	Client.readAccess
play	Stream.play()
position	File.position
prefix	XML.prefix
previousSibling	XML.previousSibling
protocol	Client.protocol
purge	SharedObject.purge()
read	File.read()
readAccess	Client.readAccess
readAll	File.readAll()
readByte	File.readByte()
readln	File.readln()
record	Stream.record()

referrer	Client.referrer
registerClass	Application.registerClass()
registerProxy	Application.registerProxy()
rejectConnection	Application.rejectConnection()
remove	File.remove()
removeNode	XML.removeNode()
renameTo	File.renameTo()
request	SOAPCall.request
__resolve	Client.__resolve
response	SOAPCall.response
resyncDepth	SharedObject.resyncDepth
secure	Client.secure
seek	getGlobal()
send	LoadVars.send(), SharedObject.send(), Stream.send(), XML.send(), XMLSocket.send()
sendAndLoad	LoadVars.sendAndLoad(), XML.sendAndLoad()
server	Application.server
setAttributes	setAttributes()
setBandwidthLimit	Client.setBandwidthLimit()
setBufferTime	Stream.setBufferTime()
setInterval	setInterval()
setProperty	SharedObject.setProperty()
setVirtualPath	Stream.setVirtualPath()
shutdown	Application.shutdown()
size	SharedObject.size(), Stream.size()
server	Application.server
status	XML.status
syncWrite	Stream.syncWrite
toString	File.toString(), LoadVars.toString(), XML.toString()
trace	trace()
type	File.type
unlock	SharedObject.unlock()
uri	NetConnection.uri, Client.uri
version	SharedObject.version
virtualKey	Client.virtualKey
write	File.write()
writeAccess	Client.writeAccess
writeAll	File.writeAll()
writeByte	File.writeByte()
writeln	File.writeln()
xmlDecl	XML.xmlDecl

Application类

可用性

Flash Communication Server 1

Application类包含了有关一个Flash Media Server应用程序实例的信息，这些信息会一直维持直至这个应用程序实例被卸载。

一个Flash Media Server应用程序是一个流对象、共享对象，以及客户机（连接的用户）的集合。每一个应用程序都有一个唯一的名字，而你可以使用在“使用命名约定”中描述的命名方案来创建一个应用程序的多个实例。

一个Flash Media Server应用程序的每一个实例都有一个application对象，这是Application类的一个单一实例。你不需要使用构造器函数来创建这个application对象；当一个应用程序由服务器例示时，application对象会被自动创建。使用下面的语法来调用Application类的方法、属性和事件处理器：

application.methodPropertyOrHandler

使用application对象来接受或拒绝客户机的连接尝试，注册和注销类和代理，以及创建当一个应用程序启动或停止，或当一个客户机连接或断开连接时要调用的函数。

除了Application类的内建属性外，你还可以创建其他具有任何合法的ActionScript类型的属性，包括对其他ActionScript对象的引用。例如，下面的代码行创建了一个array类型的新属性和一个number类型的新属性：

```
application.myarray = new Array();
application.num_requests = 1;
```

Application类的方法汇总

方法	描述
Application.acceptConnection()	接受一个来自客户机的至一个应用程序的连接。
Application.broadcastMsg()	向所有连接的客户机广播一条消息。
Application.clearSharedObjects()	清理与当前实例相关的所有共享对象。
Application.clearStreams()	清理与当前实例相关的所有流对象。
Application.disconnect()	从服务器断开一个客户机的连接。
Application.gc()	调用垃圾收集器来回收该应用程序实例未使用的任何资源。
Application.getStats()	返回这个应用程序实例的网络状态。
Application.registerClass()	注册或注销一个构造器，这个构造器是在对象的反序列化期间被调用的。
Application.registerProxy()	注册一个NetConnection或Client对象以完成一个方法请求。
Application.rejectConnection()	拒绝至一个应用程序的连接。
Application.shutdown()	卸载应用程序实例。

Application类的属性汇总

属性	描述
Application.allowDebug	一个布尔值，可以允许管理者使用approveDebugSession()服务器管理ActionScript方法来访问你的Flash Media Server应用程序（true），或是不允许（false）。
Application.clients	只读；一个对象，该对象包含了当前连接到这个应用程序的所有客户的一个列表。
Application.config	允许你访问Application.xml配置文件的ApplicationObject标签的属性。
Application.name	只读；一个应用程序实例的名字。
Application.server	只读；服务器的平台和版本。

Application类的事件处理器汇总

属性	描述
Application.onAppStart	当这个应用程序被服务器装载时调用。
Application.onAppStop	当这个应用程序被服务器卸载时调用。
Application.onConnect	当一个客户机连接到这个应用程序时调用。
Application.onAppStop	当这个应用程序被服务器卸载时调用。
Application.onConnect	当一个客户机连接到这个应用程序时调用。
Application.onConnectAccept	当一个客户机成功的连接到这个应用程序时调用；仅用于通讯组件。
Application.onConnectReject	当一个客户机连接到这个应用程序失败时调用；仅用于通讯组件。
Application.onDisconnect	当一个客户机从这个应用程序断开连接时调用。
Application.onStatus	当一个脚本产生一个错误时调用。

Application.acceptConnection()

可用性
Flash Communication Server MX 1.0
用法
application.acceptConnection(clientObj)
参数
clientObj 一个客户机对象；要接受的一个客户机。
返回

无

描述

方法。接受自一个客户机到这个服务器的连接调用。当NetConnection.connect自客户机端被调用时，application.onConnect事件处理器会在服务器端被调用以通告一个脚本。你可以在一个application.onConnect事件处理器中使用application.acceptConnection方法来接受来自一个客户机的连接。你可以在一个application.onConnect事件处理器的外部使用application.acceptConnection方法来接受一个已经处于一种未决状态（例如，要验证一个用户名和密码）的客户机连接。

当你使用组件，并且你的代码包含一个对application.acceptConnection或application.rejectConnection的明确的调用的话，则onConnect方法的最后一行（按执行顺序算）应该是application.acceptConnection或application.rejectConnection中的一个。同样，任何跟在显式的acceptConnection或rejectConnection语句后面的逻辑都必须位于application.onConnectAccept和application.onConnectReject语句中，否则它将被忽略。这个必要条件只有当你使用组件时才是需要的。

例子

下面的例子使用application.acceptConnection方法来接受来自client1的连接：

```
application.onConnect = function (client1){
  //在这里插入代码
  application.acceptConnection(client1);
  client1.call("welcome");
};
```

注意：上面的例子展示了来自一个不使用组件的应用程序的代码。

Application.allowDebug()

Flash Media Server 2

用法

application.allowDebug

描述

属性；一个布尔值，允许（true）或禁止（false）管理员使用approveDebugSession()服务器管理API来访问你的应用程序。

一个调试连接会显示有关共享对象和流的信息。

Application.broadcastMsg()

可用性

Flash Media Server 2

用法

application.broadcastMsg(cmd [, p1, p2, ..., pN])

参数

cmd 一个字符串；要广播的一条消息。

pl 一个字符串；额外的信息。

返回

无。

描述

方法；把一条消息广播到所有连接的客户机。

这个方法相当于循环遍历Application.clients数组并在每一个独立的客户机上调用Client.call()，但这个方法的效率更高（尤其是当连接的客户机数量很大时）。唯一的不同是当你调用broadcastMsg()时你不能指定一个响应对象，除此以外，两种语法是一
样的。

共享对象可以用SharedObject.handlerName属性来处理广播消息。

例子

下面的服务器端代码把一条消息发送到客户机：

```
application.broadcastMsg("handlerName", "Hello World");
```

下面的客户机端代码捕获一条消息并把它打印到输出面板中：

```
nc = new NetConnection();
nc.handlerName = function(msg) { trace(msg); }
// traces out "Hello World"
```

Application.clearSharedObjects()

可用性

Flash Communication Server MX

用法

```
application.clearSharedObjects(soPath);
```

参数

soPath 一个字符串，它指出了一个共享对象的URI。

返回

一个布尔值。如果位于指定路径的共享对象被删除了，则返回true；否则，返回false。如果使用通配符来删除多个流文件，则只有当匹配这个通配符模式的所有共享对象都被成功删除了，这个方法才会返回true；否则，它将返回false。

描述

方法。删除由soPath参数指定的永久性共享对象，并从活动的共享对象（永久性的和非永久性的）上清除所有的属性。

soPath参数指定了一个共享对象的名字，路径中可以包含斜杠（/）作为目录间的分隔符。路径中最后的元素可以包含通配符模式（例如，一个问号[?]和一个星号[*]）或是一个共享对象名。
application.clearSharedObjects方法会沿着指定的路径在共享对象层级体系中穿行并清理所有的共享对象。指定一个斜杠会清理与一个应用程序实例有关的所有共享对象。

下面是soPath参数的可能的值：

/ 清除与实例关联的所有本地和永久性共享对象。

/foo/bar 清除共享对象/foo/bar；如果bar是一个目录名的话，则没有共享对象会被删除。

/foo/bar/* 清除存储在实例目录/foo/bar下的所有共享对象。如果没有永久性共享对象在这个命名空间中被使用的话，则

这个bar目录也会被删除。

/foo/bar/XX?? 清除所有以XX开始且后跟任意两个字符的共享对象。如果一个目录名匹配这个规则，则在这个目录中的

如果你调用clearSharedObjects方法，并且指定的路径匹配一个当前正活动着的共享对象，则其所有属性都会被删除，并且
一个“clear”事件会被发送到这个共享对象的所有订阅者。如果它是一个永久性共享对象的话，则这个永久性存储也会被清
除。

例子

下面的例子清除一个实例的所有共享对象：

```
function onApplicationStop(){
function onApplicationStop(){
   application.clearSharedObjects("/");
}
```

Application.clearStreams()

可用性

Flash Communication Server MX

用法

application.clearStreams(streamPath);

参数

streamPath 一个字符串，指出一个流的URI。

返回

一个布尔值。如果位于指定路径的流被删除了，则返回true；否则，返回false。如果使用通配符来删除多个流文件，则只有当匹配这个通配符模式的所有流都被成功删除了，这个方法才会返回true；否则，它将返回false。

描述

方法。清除与这个应用程序实例关联的记录的流文件（FLV）和MP3文件。你可以使用这个方法来清除单个流，与这个应用程序实例关联的所有流，仅那些位于这个应用程序实例的一个特定的子目录中的流，或是仅那些其名字匹配一个特定的通配
符模式的流。

streamPath参数指定了一个流相对于这个应用程序的实例目录的位置和名字。你可以在路径中包含斜杠（/）来作为目录间
的分隔符。路径中最后的元素可以包含通配符模式（例如，一个问号[?]和一个星号[*]）或是一个流名。
application.clearStreams
方法会沿着指定的路径在流层级体系中穿行并清理所有的记录流。指定一个斜杠会清理与一个应用程序实例有关的所有流。

要清除与这个应用程序实例关联的MP3文件，用mp3:作为至这个流的路径的前导（例如，
mp3:/streamPath）。默认情况
下，application.clearStreams方法只清除记录的FLV流。你也可以通过在流路径前放置flv:来明确只清除FLV流（例如，
flv:/streamPath）。

下面是streamPath参数的一些可能的值：

/report 从应用程序实例目录中清除流文件清除与实例关联的所有的记录流（report01.flvFLV 。

/presentations/intro 从应用程序实例的/presentations子目录中清除记录的流文件intro.flv；如果intro是一个目录名的话，

则没有流会被删除。

/presentations/* 从应用程序实例的/presentations子目录中清除所有的记录流文件。如果没有流在这个命名空间中被使用

的话，则这个/presentations子目录也会被删除。

mp3:/ 清除与实例关联的所有MP3文件。

mp3:/ 清除与实例关联的所有MP3文件。 子目录中清除名为requiem.mp3的MP3文件。

mp3:/mozart/requiem 从应用程序实例的/mozart子目录中清除名为requiem.mp3的MP3文件。

mp3:/mozart/* 从应用程序实例的/mozart子目录中清除所有的MP3文件。

/presentations/report?? 清除所有以report开头，后跟任意两个字符的所有记录流文件（FLV）。如果在给定的目录列表

中有目录的话，则这些目录会被清除任何匹配report??的流。

如果一个application.clearStreams方法在一个当前正记录的流上被调用的话，则被记录的文件将被设置为长度是（被清
除），并且内部的缓存数据也会被清除。

注意：你也可以使用Server Management ActionScript API removeApp方法来删除单个实例的所有资源。

例子

下面的例子清除所有的记录流：

```
function onApplicationStop(){
  application.clearStreams("/");
}
```

下面的例子从应用程序实例的/disco子目录中清除所有的MP3文件：

```
function onApplicationStop(){
  application.clearStreams("mp3:/disco/*");
}
```

Application.clients

可用性

Flash Communication Server MX

用法

application.clients

描述

属性（只读）。一个对象，该对象包含了当前连接到的这个应用程序的所有的Flash客户机或其他的Flash Media Server。这
个对象是一个定制对象，就像一个数组，但只有一个属性-length。在这个对象中的每一个元素都是对一个Client对象实例的引
用，并且你可以使用length属性来确定连接到这个应用程序的用户数。你可以对application.clients属性应用数组访问运算符（[]）
来访问这个对象中的元素。

这个对象是用于clients属性的，而clients属性不是一个数组，但其行为是相同的，只除了一点：你不能使用下面这样的语法
来遍历这个对象：

```
for(var i in application.clients) {
  //在这里插入代码
}
```

代替之，你应该使用下面的代码来遍历一个clients对象中的每一个元素：

```
for (var i = 0; i < application.clients.length; i++) {
  //在这里插入代码
}
```

例子

这个例子使用一个for循环来application.clients数组中的每一个元素，并在每一个客户机上调用serverUpdate方法：

```
for (i = 0; i < application.clients.length; i++){
  application.clients[i].call("serverUpdate");
}
```

Application.config()

可用性

Flash Media Server 2

描述用法

方法；允许你访问Application.xml配置文件的ApplicationObject标签。

例子

为了使用这个范例，在Application.xml文件中使用下面的范例<ApplicationObject>标签：

```
<Application>
  <JSEngine>
    <ApplicationObject>
      <config>
        <user_name>jdoe</user_name>
        <dept_name>engineering</dept_name> <config>
        <user_name>jdoe</user_name>        </ApplicationObject>
        <dept_name>engineering</dept_name> <config>
    </ApplicationObject>
  </JSEngine>
</Application>
```

下列代码行中的任何一个都访问前面的Application.xml文件范例中定义的同一个Application属性：

```
trace("I am " + application.config.user + " and I work in the " + application.config.dept_name + " department.");

trace("I am " + application.config["user"] + " and I work in the " + application.config["dept_name"] + " department.");
```

下面的代码被发送到应用程序的日志文件和Application检查器：

I am jdoe and I work in the engineering department.

Application.disconnect()

可用性
Flash Communication Server MX
用法
application.disconnect(clientObj)
参数
clientObj 要断开连接的客户机。这个对象必须是一个来自application.clients数组的Client对象。
返回
一个布尔值。如果断开是成功的，则返回true；否则，返回false。
描述
方法。导致服务器终止一个客户机至这个应用程序的连接。当这个方法被调用时，在客户机端，NetConnection.onStatus会
以一个NetConnection.Connection.Closed状态消息被调用。application.onDisconnect方法也会被调用。
例子
这个例子调用application.disconnect方法来断开一个应用程序实例的所有的用户连接：

```
function disconnectAll(){
   for (i=0; i < application.clients.length; i++){
      application.disconnect(application.clients[i]);
   }
}
```

Application.gc()

可用性
Flash Media Server 2
用法
application.gc()
参数
无。
返回
无。
描述
方法；调用垃圾收集器来回收该应用程序实例未使用的任何资源。

Application.getStats()

可用性
Flash Communication Server MX
用法
application.getStats()

返回

一个带有多个属性的ActionScript对象，其中的每一个属性代表一个返回的状态。

描述

方法。返回应用程序实例的状态，包括发送和接收的总的字节数、发送和接收的RTMP消息的数量、被丢弃的消息的数
量、连接到这个应用程序实例的客户机数量，以及已经从这个应用程序实例断开连接的客户机数量。

例子

下面的trace例子使用Application.getStats来把这个应用程序实例的状态输出到输出窗口：

```
stats = application.getStats();
trace("Total bytes received: " + stats.bytes_in);
trace("Total bytes sent : " + stats.bytes_out);
trace("RTMP messages received : " + stats.msg_in);
trace("RTMP messages sent : " + stats.msg_out);
trace("RTMP messages dropped : " + stats.msg_dropped);
trace("Total clients connected : " + stats.total_connects);
trace("Total clients disconnected : " + stats.total_disconnects);
```

Application.hostname

可用性

Flash Communication Server MX 1.5

用法

application.hostname

描述

属性（只读）。对于默认虚拟主机，包含服务器的主机名；对于非默认虚拟主机，包含虚拟主机名。

例子

下面的例子把运行当前应用程序的服务器的名字输出至输出窗口。

```
trace( application.hostname )
```

Application.name

可用性

Flash Communication Server MX

用法

application.name

描述

属性（只读）。包含Flash Media Server应用程序实例的名字。

例子

下面的例子在其执行某些代码前依赖一个特定的字符串来检查name属性：

```
if (application.name == "videomail/work"){
  //在这里插入代码
}
```

Application.onAppStart

可用性

Flash Communication Server MX

用法

application.onAppStart = function (){}

参数

无

返回

无

描述

事件处理器。当服务器第一次装载这个应用程序实例时调用。你使用这个处理器来初始化一个应用程序的状态。你可以使
用application.onAppStart和application.onAppStop来初始化和清除一个应用程序中的全局变量，因为这些事件在一个应用程序实例
的生命期中都只被调用一次。

例子

下面的例子为application.onAppStart处理器定义了一个匿名函数，用来发送一个trace消息：

```
application.onAppStart = function () {
  trace ("onAppStart called");
};
```

Application.onAppStop

可用性

Flash Communication Server MX

用法

application.onAppStop = function (info){}

参数

info 一个信息对象，它解释了为什么这个应用程序会被停止运行。参看“服务器端信息对象”。

返回

值通过你定义的函数被返回，如果有的话；否则，将返回null。要拒绝卸载这个应用程序，返回false；要卸载这个应用程
序，返回true或任何非false值。

描述

事件处理器。当这个应用程序就要被服务器卸载时调用。你可以定义一个当这个事件处理器被调用时要执行的函数。如果
函数返回true，则这个应用程序被卸载。如果函数返回false，则这个应用程序没有被卸载。如果你没有为这个事件处理器定义一
个函数，或是如果返回值不是一个布尔值，则当这个事件被调用时这个应用程序会被卸载。

Flash Media Server应用程序会把一个信息对象传递给application.onAppStop事件。你可以使用服务器端

ActionScript来查看这
个信息对象以决定在你定义的函数中要做什么事情。你也应该定义application.onAppStop事件以便在停止系统运行前通告用户。

如果你使用Administration Console或是Server Management ActionScript API来卸载一个Flash Communication Server应用程序，
则application.onAppStop不会被调用。因此，你不能使用application.onAppStop事件来告诉用户这个应用程序正在退出。

例子

这个范例定义了一个函数来在应用程序上执行停止系统运行的操作。这个函数被指派给这个事件处理器，以便当这个处理
器被调用时可以执行这个函数。

```
function onMyApplicationEnd(info){
  //在这里执行所有与这个特定的应用程序的停止系统运行有关的逻辑。
}

application.onAppStop = onMyApplicationEnd;
```

Application.onConnect

可用性

Flash Communication Server MX

用法

application.onConnect = function (clientObj [, p1, ..., pN]){}

参数

clientObj 连接到这个应用程序的客户机。

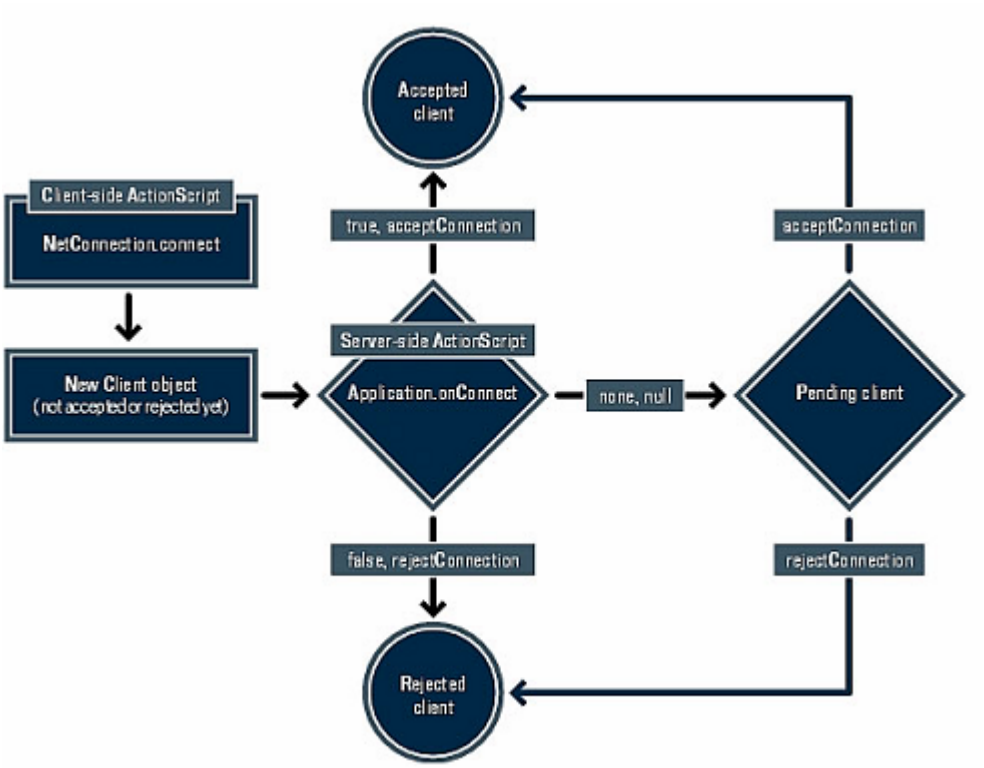
p1 ..., pN 传递给application.onConnect方法的可选的参数。当一个客户机连接到这个应用程序时，这些参数自客户机端的
NetConnection.connect方法传递而来。

返回

返回你提供的值。如果你返回一个布尔值true，则服务器就接受这个连接；如果值是false，则服务器就拒绝这个连接。如果
你返回null或没有返回值，则服务器就会把客户机置于一种未决状态，并且客户机不能接收或发送消息。如果客户机被置于一种
未决状态，则你必须在以后的某个时间调用application.acceptConnection或application.rejectConnection来接受或拒绝这个连接。例
如，通过在你的application.onConnect事件处理器中作一个对一个应用程序服务器的NetConnection调用，并依赖于答复处理器接
收到的信息，让答复处理器调用application.acceptConnection或application.rejectConnection来执行一个外部的身份验证。

你也可以在application.onConnect事件处理器中调用application.acceptConnection或application.rejectConnection。如果你这么做
的话，则由这个函数返回的任何值都会被忽略。

注意：返回或与返回true或false并不一样。值和被视为与其他任何整数一样，而不能接受或拒绝一个连接。



如何使用application.onConnect来接受、拒绝，或把一个客户机置于一种未决状态。

描述

事件处理器。当NetConnection.connect从客户机端被调用，且一个客户机试图连接到一个应用程序实例时，这个事件处理器
会在服务器端被调用。你可以为application.onConnect事件处理器定义一个函数。如果你没有定义一个函数，则默认情况下连接
会被接受。如果服务器接受这个新连接，则application.clients对象会被更新。

你可以在服务器端脚本中使用application.onConnect事件来执行身份验证。身分验证所需的所有信息都应该由客户机作为你
定义的参数（p1 ..., pN）发送到服务器。除了身份验证外，这个脚本也可以设置对所有服务器端对象的访问权限，通过设置
Client.readAccess和Client.writeAccess属性，这个客户机可以修改这些服务器端对象。

如果有多个并发连接请求同一个应用程序，则服务器会序列化这些请求，以便同一时刻只有一个application.onConnect处理
器执行。为application.onConnect函数编写执行迅速的代码以避免客户机长时间的连接等待是个好主意。

注意：如果你正在使用组件架构（也就是说，你正在你的服务器端脚本文件中装载components.asc文件），则你必须使用
Application.onConnectAccept方法来接受客户机的连接。

例子

下面的范例展示了onConnect处理器的三种基本用法：

（用法一）

```
application.onConnect = function (clientObj [, p1, ..., pN]){
  //在这里插入代码以调用完成身份验证工作的方法
  //返回null以把客户机置于一种未决状态
  return null;
};
```

（用法二）

```
application.onConnect = function (clientObj [, p1, ..., pN]){
  //在这里插入代码以调用完成身份验证工作的方法
  //接受这个连接
  application.acceptConnection(clientObj);
};
```

（用法三）

```
application.onConnect = function (clientObj [, p1, ..., pN])
{
  //在这里插入代码以调用完成身份验证工作的方法
  //通过返回true来接受这个连接
  return true;
};
```

这个范例验证用户已经发送的密码“XXXX”。如果密码被发送了，则这个用户的访问权限会被修改，并且这个用户可以
完成这个连接。在这种情况下，这个用户可以在他自己的目录中创建或编写流和共享对象，并可以读取或查看这个应用程序实
例中的任何共享对象或流。

```
//这个代码应该被放置在全局范围内
application.onConnect = function (newClient, userName, password){
  //完成所有这个应用程序专有的逻辑
  if (password == "XXXX"){
    newClient.writeAccess = "/" + userName;
    this.acceptConnection(newClient);
  } else {
    var err = new Object();
    err.message = "Invalid password";
    this.rejectConnection(newClient, err);
  }
};
```

如果密码是错误的，则用户会被拒绝，并且一个其message属性被设置为“Invalid password”的信息对象会被返回到客户机
端。这个对象被指派给infoObject.application。要访问这个message属性，在客户机端使用下面的代码：

```
ClientCom.onStatus = function (info){
  trace(info.application.message);
  //它将在客户机端的输出窗口中打印"Invalid password"
};
```

Application.onConnectAccept

可用性

Flash Media Server MX（仅用于通讯组件）

用法

```
application.onConnectAccept = function (clientObj [,p1, ..., pN]){}
```

参数

clientObj 一个客户机对象；该客户机连接到这个应用程序。

p1...pN 传递给application.onConnectAccept方法的可选的参数。当一个客户机连接到这个应用程序时，这些参数自客户机端
的NetConnection.connect方法传递而来。

返回

无。

描述

事件处理器。只有当通讯组件被使用时（也就是说，当components.asc脚本被装入你的服务器端应用程序脚本中时）才会
被调用。当NetConnection.connect从客户机端被调用，且一个客户机成功地连接到一个应用程序实例时，application.onConnectAccept才会在服务器端被调用。

使用onConnectAccept来处理包含组件的应用程序中一个被接受连接的结果。

如果你不是使用Flash Media Server组件框架，则你可以在接受或拒绝这个连接后执行application.onConnect处理器中的代
码。但不管怎么说，当你使用组件架构时，任何你想要在连接被接受或拒绝后执行的代码都必须被放置在事件处理器
application.onConnectAccept和application.onConnectReject中。这种体系结构可以让所有的组件决定一个连接是被接受还是被拒
绝。

例子

下面的范例是你可以用于一个应用程序的客户机端代码：

```
nc = new NetConnection();
nc.connect("rtmp:/test","jlopes");

nc.onStatus = function(info) {
    trace(info.code);
};

nc.doSomething = function(){
    trace("doSomething called!");
}
```

下面的范例是你可以包含在main.asc文件中的服务器端代码：

```
//当使用组件时，总是要装载components.asc
load("components.asc");

application.onConnect = function(client, username){
    trace("onConnect called");
     gFrameworkFC.getClientGlobals(client).username = username;
    if (username == "hacker") {
        application.rejectConnection(client);
    }
    else {
        application.acceptConnection(client);
    }
}
```

```
//由于组件被使用，因此，代码应该位于onConnectAccept和onConnectReject语句中。
application.onConnectAccept = function(client, username){
    trace("Connection accepted for "+username);
    client.call("doSomething",null);
}

application.onConnectReject = function(client, username){
    trace("Connection rejected for "+username);
}
```

Application.onConnectReject

可用性

Flash Media Server（仅用于通讯组件）

用法

```
application.onConnectReject = function (clientObj [,p1, ..., pN]){}
```

参数

clientObj 连接到这个应用程序的客户机。

p1...pN 传递给application.onConnectReject方法的可选的参数。当一个客户机连接到这个应用程序时，这些参数自客户机端的NetConnection.connect方法传递而来。

返回

无

描述

事件处理器。只有当通讯组件被使用时（也就是说，当components.asc脚本被装入你的服务器端脚本中时），当NetConnection.connect()从客户机端被调用，且一个客户机连接到一个应用程序实例失败时，application.onConnectReject才会在服务器端被调用。

使用onConnectReject来处理包含组件的应用程序中一个被拒绝连接的结果。

如果你不是使用Flash Media Server组件框架，则你可以在接受或拒绝这个连接后执行application.onConnect处理器中的代码。但不管怎么说，当你使用组件架构时，任何你想要在连接被接受或拒绝后执行的代码都必须被放置在事件处理器application.onConnectAccept和application.onConnectReject中。这种体系结构可以让所有的组件决定一个连接是被接受还是被拒绝。

例子

下面的范例是你可以用于一个应用程序的客户机端代码：

```
nc = new NetConnection();
nc.connect("rtmp:/test","jlopes");

nc.onStatus = function(info) {
    trace(info.code);
```

```
};

nc.doSomething = function(){
    trace("doSomething called!");
}
```

下面的范例是你可以包含在main.asc文件中的服务器端代码：

```
//当使用组件时，总是要装载components.asc
load( "components.asc" );

application.onConnect = function(client, username){
    trace("onConnect called");
     gFrameworkFC.getClientGlobals(client).username = username;
    if (username == "hacker") {
        application.rejectConnection(client);
    }
    else {
        application.acceptConnection(client);
    }
}

application.onConnectAccept = function(client, username){
    trace("Connection accepted for "+username);
    client.call("doSomething",null);
}

application.onConnectReject = function(client, username){
    trace("Connection rejected for "+username);
}
```

Application.onDisconnect

可用性

Flash Communication Server MX

用法

```
application.onDisconnect = function (clientObj){}
```

参数

clientObj 一个客户机对象；一个断开与应用程序的连接的客户机。

返回

服务器忽略任何返回值。

描述

事件处理器。当一个客户机从应用程序断开连接时调用。你可以使用这个事件处理器来清洗任何客户机状态信息，或是通
告这个事件的其他用户。这个事件处理器是可选的。

例子

下面的范例使用一个匿名函数并将其指派到application.onDisconnect事件处理器：

```
//这个代码应该被放置在全局范围内。
application.onDisconnect = function (client){
  //完成所有这个客户机特有的断开连接的逻辑
  //在这里插入代码
  trace("user disconnected");
};
```

Application.onStatus

可用性

Flash Communication Server MX

用法

```
application.onStatus = function (infoObject){}
```

参数参数

infoObject 一个对象，它包含错误级别、代码，有时还有一个描述。

返回

回调函数返回的任何值。

描述

事件处理器。当服务器在处理一个瞄准这个应用程序实例的消息时遭遇了一个错误，则这个事件处理器就会被调
用。application.onStatus事件处理器支持任何不能找到处理器的Stream.onStatus或NetConnection.onStatus消息。同样，有几个状态
调用只能到达application.onStatus。这个事件处理器可以被用于调试产生错误的消息。

例子例子

下面的范例定义了一个函数，任何时候当下面的范例定义了一个函数，任何时候当application.onStatusapplication.onStatus方法被调用，这个函数就会发送一个方法被调用，这个函数就会发送一个tracetrace语句。你也可以定义语句。你也可以定义

一个函数，赋予用户有关发生的错误的类型的反馈信息。

```
appInstance.onStatus = function(infoObject){
  trace("An application error occurred");
};
```

Application.registerClass()

可用性

Flash Communication Server MX

用法

```
application.registerClass(className, constructor)
```

参数

className 一个ActionScript类的名字。

constructor 一个构造器函数，用来在对象的反编序期间创建一个特定类类型的一个对象。构造器函数的名字必须与
className一样。在对象的编序期间，构造器函数的名字被串行化为对象的类型。要注销这个类，传递值null作为constructor参
数。编序是把一个对象变成某些你可以经由网络发送到另一台计算机的东西的过程。

返回

无

描述

方法。注册一个构造器函数，当反编序一个某一类类型的对象时会用到这个构造器函数。如果一个类的构造器没有被注
册，则你就不能调用这个被反编序的对象的方法。这个方法也用于注销一个类的构造器。这是一个服务器的高级使用，并且只
有当在一个客户机和一个服务器之间发送ActionScript对象时才会用到。

客户机和服务器经由一个网络连接进行通讯。因此，如果你使用类型化的对象，则每一端都必须有它们共同使用的同一对
象的原型。换句话说，客户机端和服务器端ActionScript都必须定义和声明它们共享的数据的类型，以便在客户机的一个对象、
方法或属性及其服务器上的对应元素间有一个清晰的相互关系。你可以使用application.registerClass在服务器端注册对象的类类
型，以便你可以使用定义在这个类中的方法。

构造器函数应该被用于初始化属性和方法；它们不应该被用于执行服务器代码。当消息从客户机被接收并且需要是“安全
的”（万一它们被一个恶毒的客户机执行的话）的时，构造器函数会被自动调用。你不应该定义会导致消极情况的程序，比如
填满了硬盘或是极大的消耗了CPU。

构造器函数是在对象的属性被设置前调用的。一个类可以定义一个onInitialize方法，这个方法会在对象连同其所有的属性都
已经被初始化之后被调用。在一个对象被反编序之后，你可以使用这个方法来处理数据。
的话，则当一个客户机端类的一个实例从客户机被传递到服务器时，application.registerClass将不能适当的识别这个类，并会返
回一个错误。

例子

下面的范例定义了一个Color构造器函数，其具有属性和方法。在这个应用程序连接后，registerClass方法被调用以为类型
Color的对象注册一个类。当一个类型化对象从客户机被发送到服务器时，这个类会被调用来创建服务器端对象。在这个应用程
序停止之后，registerClass方法会被再次调用，并传递值null以注销这个类。

```
function Color(){
  this.red = 255;
```

序停止之后，registerClass方法会被再次调用，并传递值null以注销这个类。

```
  this.blue = 0;function Color(){
  this.red = 255;
  this.green = 0;
  this.blue = 0;
}
```

```
Color.prototype.getRed = function(){
   return this.red;
}
Color.prototype.getGreen = function(){
   return this.green;
}
Color.prototype.getBlue = function(){
   return this.blue;
}
Color.prototype.setRed = function(value){
   this.red = value;
}
Color.prototype.setGreen = function(value){
   this.green = value;
}
Color.prototype.setBlue = function(value){
   this.blue = value;
}
application.onAppStart = function(){
   application.registerClass("Color", Color);
};
application.onAppStop = function(){
   application.registerClass("Color", null);
};
```

下面的范例展示了如何结合prototype属性来使用application.registerClass方法：

```
function A(){}
function B(){}

B.prototype = new A();
//把构造器设置回B的构造器。
B.prototype.constructor = B;
//在这里插入代码
application.registerClass("B", B);
```

Application.registerProxy()

可用性

Flash Communication Server MX

用法

application.registerProxy(methodName, proxyConnection [, proxyMethodName])

参数

methodName 一个方法名。所有要在这个应用程序实例上执行methodName的请求都会被转寄到proxyConnection对象。

proxyConnection 一个Client或NetConnection对象。所有要执行由methodName指定的远端方法的请求都

会被发送到Client或
NetConnection对象，这个对象是在proxyConnection参数中指定的。返回的任何结果都被发送回调用的始发者。要注销，或删除
这个代理，为这个参数提供一个值null。

proxyMethodName 一个可选的参数。如果proxyMethodName与由methodName参数指定的方法不同的话，则服务器会在由
proxyConnection参数指定的对象上调用这个方法。

返回

一个值，它被发送回进行调用的客户机。

描述

方法。把一个方法调用映射到另一个函数。你可以使用这个方法在位于同一个Flash Media Server（或不同的Flash Media
Server）上的不同的应用程序实例间进行通讯。客户机可以执行它们连接到的任何应用程序实例的服务器端方法。服务器端脚
本可以使用这个方法来注册那些将要被代理到位于同一服务器或一个不同服务器上的其他应用程序实例的方法。你可以通过调
用这个方法并为proxyConnection参数传递null来删除或注销这个代理，这会产生与从未注册过这个方法同样的效果。

例子

在下面的例子中，当应用程序启动时，application.registerProxy方法在application.onAppStart事件处理器的一个函数中被调用
并执行。在函数块内部，一个新的名为myProxy的NetConnection对象被创建并被连接。之后，application.registerProxy方法被调
用以把getXyz方法指派给myProxy对象。

```
application.onAppStart = function(){
  var myProxy = new NetConnection();myProxy.connect("rtmp://xyz.com/myApp");
  application.registerProxy("getXyz", myProxy);
```

Application.rejectConnection()

可用性

Flash Communication Server MX

用法

application.rejectConnection(clientObj, errObj)

参数

clientObj 一个要拒绝的客户机。

errObj 参数 一个任意类型的对象，它被发送到客户机，解释了拒绝的原因。当连接被拒绝时，errObj对象是作为传递给

clientObj 一个要拒绝的客户机。

errObj 一个任意类型的对象，它被发送到客户机，解释了拒绝的原因。当连接被拒绝时，errObj对象是作为传递给
application.onStatus调用的信息对象的application属性可用的。

返回

无

描述

方法。拒绝从一个客户机至服务器的连接呼叫。当一个新的客户机进行连接时，application.onConnect事件处理器会通告一
个脚本。在指派给application.onConnect的函数中，你可以接受或拒绝这个连接。你也可以为application.onConnect定义一个函
数，在这个函数中调用一个应用程序服务器进行身份验证。在这种情况中，一个应用程序可以根据这个应用程序服务器的响应
回调来调用application.rejectConnection以断开客户机至服务器的连接。

当你使用组件，并且你的代码包含一个对application.acceptConnection或application.rejectConnection的明确的调用的话，则
onConnect方法的最后一行（按执行顺序算）应该是application.acceptConnection或application.rejectConnection中的一个。同样，任
何跟在显式的acceptConnection或rejectConnection语句后面的逻辑都必须位于application.onConnectAccept和application.onConnectReject语句中，否则它将被忽略。这个必要条件只有当你使用组件时才是需要的。

例子

在下面的例子中，由client1指定的客户机被拒绝，并被提供包含在err.message中的错误消息。消息“Too many
connections”会出现在客户机端。

```
function onConnect(client1){
    // 在这里插入代码。
    var err = new Object();
    err.message = "Too many connections";
    application.rejectConnection(client1, err);
}
```

下面的代码应该出现在客户机端：

```
clientConn.onStatus = function (info){
  if (info.code == "NetConnection.Connect.Rejected"){
    trace(info.application.message);
    //在客户机端这会发送消息"Too many connections"至输出窗口。
  }
};
```

Application.server

可用性

Flash Communication Server MX

用法

application.server

描述

属性（只读）。包含了平台和服务器版本信息。

例子

下面的范例在执行if语句中的代码前，依赖于一个字符串来检查server属性：

```
if (application.server == "Flash Media Server-Windows/1.0"){
  //在这里插入代码
}
```

Application.shutdown()

可用性
Flash Media Server 2
用法
application.shutdown()
参数
无。
返回
一个布尔值，指出成功（true）或失败（false）。
描述
方法；卸载应用程序实例。
如果应用程序正运行于vhost（虚拟主机）或应用程序级范围，则仅这个应用程序实例被卸载，而核心进程仍然会继续运
行。如果应用程序正运行于实例范围，则这个应用程序会被卸载，且核心进程也会终止。这个过程是异步完成的；实例是在卸
载序列开始时被卸载，而不是在shutdown()调用返回时卸载。
例子
在下面的范例中，由client1指定的客户机被拒绝，并提供了包含在err.message中的错误信息。消息“Too many
connections”出现在服务器端。

```
function onConnect(client1){
    // 在这里插入代码。
    var err = new Object();err.message = "Too many connections";
    application.rejectConnection(client1, err);
}
```

下面的代码将出现在的客户机端：

```
clientConn.onStatus = function (info){
    if (info.code == "NetConnection.Connect.Rejected"){
        trace(info.application.message);
        // 在客户机端发送"Too many connections"消息至输出面板。
    }
};
```

clearInterval()

可用性
Flash Communication Server MX
用法
clearInterval(intervalID)

参数

intervalID 由一个先前对setInterval()方法的调用返回的一个唯一的ID。

返回

无

描述

方法（全局）。清除一个暂停时间，这个暂停时间是由一个对setInterval()方法的调用设置的。

例子

下面的范例创建了一个名为callback的函数并把它传递给setInterval()方法，这个setInterval()方法会每毫秒被调用一次并
发送消息“interval called”至输出窗口。setInterval()方法返回一个唯一的标识符，这个标识符被指派给变量intervalID。这个标识
符允许你清除一个特定的setInterval()调用。在代码的最后一行，intervalID变量被传递给clearInterval()方法以清除这个setInterval
()调用：

```
var intervalID;
intervalID = setInterval(callback, 1000);
//一段时间之后
clearInterval(intervalID);
```

Client类

Client类

可用性

Flash Communication Server 1

Client类可以让你处理连接到一个Flash Media Server应用程序实例的每一个用户，或称客户机。当一个用户连接到一个应用
程序时，服务器会自动创建一个Client对象；当这个用户从这个应用程序断开连接时，这个对象会被销毁。用户对于他们连接到
的每一个应用程序都有一个唯一的Client对象。同一时刻可以有数千个Client对象处于活动状态。

你可以使用Client类的属性来确定每一个客户机的版本、平台，以及IP地址。你也可以对不同的应用程序资源（比如一个
Stream对象和共享对象）单独设置读写权限。使用Client类的方法来设置带宽限制及调用客户机端脚本中的方法。

当你从一个客户机端ActionScript脚本中调用NetConnection.call时，在服务器端脚本中执行的方法必须是一个Client类的方
法。在你的服务器端脚本中，你必须定义任何你想要从客户机端脚本中调用的方法。你也可以直接从服务器端脚本的Client类实
例中调用你在服务器端脚本定义的任何方法。

如果Client类的所有实例（一个应用程序中的每一个客户机）都需要相同的方法或属性，则你可以把这些方法和属性添加到
类自身中，而不是把它们添加到一个类的每一个实例中。这一过程被称为扩展一个类。你可以扩展任何服务器端或客户机端

ActionScript类。要扩展一个类，代替在类的构造器函数中定义方法或是把它们指派给类的个别的实例，你需要把方法指派给类
的构造器函数的prototype属性。当你把方法和属性指派给prototype属性时，这些方法会自动地对类的所有实例都可用。

扩展一个类可以让你个别的定义一个类的方法，这会使它们在脚本中更加可读。并且，更为重要的原因是，把方法添加到
prototype是更为有效率的方式，否则的话，每次一个实例被创建，这些方法就会被重新解释一遍。

下面的代码展示了如何把方法和属性指派给一个类的一个实例。在application.onConnect期间，一个类实例clientObj作为一个
参数被传递到服务器端脚本。然后，你可以把一个属性和方法指派给这个客户机实例：

```
application.onConnect = function(clientObj){
  clientObj.birthday = myBDay;
  clientObj.calculateDaysUntilBirthday = function(){
    //在这里插入代码
  }
};
```

前面的例子可以工作，但每次一个客户机连接时都必须被执行。如果你希望同样的方法和属性对于application.clients数组中
所有的客户机而言都可用，且不需要每次都定义它们的话，那么，你必须把它们指派给Client类的prototype属性。要利用
prototype方法来扩展一个内建类需要两个步骤。你可以在你的脚本中以任何顺序来写这些步骤。下面的例子扩展了内建的Client
类，因此，第一步就是编写你将指派给prototype属性的函数：

```
//第一步：编写函数


function Client_getWritePermission(){
//"writeAccess"属性已经内建于客户机类
return this.writeAccess;
}

function Client_createUniqueID(){
  var ipStr = this.ip;
//"ip"属性已经内建于客户机类
  var uniqueID = "re123mn"
//你将需要在上面的行中编写代码，它为每一个客户机实例创建一个唯一的ID。
  return uniqueID;
}

//第二步：把prototype方法指派给函数


Client.prototype.getWritePermission = Client_getWritePermission;
Client.prototype.createUniqueID = Client_createUniqueID;

//一个好命名习惯是以类名后跟一个下划线来开始所有的类方法名
```

你也可以把属性添加到prototype，就像下面的范例展示的这样：

```
Client.prototype.company = "Macromedia";
```

这些方法对于任何实例都是可用的，因此，在application.onConnect（它被传递以一个clientObj参数）中，你可以编写下面的
代码：

```
application.onConnect = function(clientObj){
   var clientID = clientObj.createUniqueID();
   var clientWritePerm = clientObj.getWritePermission();
};
```

Client类的方法汇总

方法	描述
Client.call()	在Flash客户机上异步的执行一个方法，并把值从Flash客户机返回到服务器。
Client.getBandwidthLimit()	返回客户机或服务器可以为这个连接尝试使用的最大带宽。
Client.getStats()	返回客户机的状态。
Client.readAccess()	发送一个“ping”消息至客户机。如果客户机响应了，则这个方法返回true；否则，返回false。
Client.__resolveClient.setBandwidthLimit()	设置连接的最大带宽。为未定义过的属性提供值。

Client类的属性汇总

属性	描述
属性	描述
Client.agent	只读；Flash客户机的版本和平台。
Client.ip	只读；Flash客户机的IP地址。
Client.protocol	只读；客户机用来连接到服务器的协议。
Client.readAccess	客户机对其拥有读访问权的一个访问级列表。
Client.referrer	只读；发起这个连接的SWF文件或服务器的URL。
Client.secure	只读；一个布尔值，指出一个Internet连接是安全的（true）还是不安全的（false）。
Client.uri	只读；由要连接到这个应用程序实例的客户机所确定的URI。
Client.virtualKey	客户机的用户代理类型（代表性的就是Flash Player的版本），但其可以

被设置成任何合法的
键值。

Client.writeAccess　　客户机对其拥有写访问权的一个访问级列表。

Client类的事件处理器汇总

事件处理器	描述
Client."commandName"	当NetConnection.call（commandName）在一个客户机端脚本中被调用时被调用。

Client.agent

可用性
Flash Communication Server MX
用法
clientObject.agent
描述
属性（只读）。包含Flash客户机的版本和平台信息。
例子
下面的例子依赖字符串“WIN”来检查agent属性，并根据其是否匹配来执行不同的代码。这个代码被写在一个onConnect函
数中。

```
function onConnect(newClient, name){
   if (newClient.agent.indexOf("WIN") > -1){
      trace ("Window user");
   }   else {
      trace ("non Window user.agent is" + newClient.agent);
   }
}
```

Client.call()

可用性
Flash Communication Server MX
用法
clientObject.call(methodName, [resultObj, [p1, ..., pN]])
参数
methodName 一个以[objectPath/]method这种形式指定的方法。例如，命令someObj/doSomething告诉客

户机要在客户机上调
用NetConnection.someObj.doSomething方法。

resultObj 一个可选的参数，当发送者期待一个来自客户机的返回值时需要这个参数。如果参数被传递但没有返回值被期待
的话，则传递值null。结果对象可以是你定义的任何对象，并且，为了有用起见，这个结果对象应该有两个方法-onResult和
onStatus，这些方法会在结果到达时被调用。如果远端方法的调用是成功的，则resultObj.onResult事件会被触发；否则，
resultObj.onStatus事件将被触发。

p1, ..., pN 可选的参数，它们可以是任何ActionScript类型，包括对另一个ActionScript对象的引用。当这个方法在Flash客户
机上执行时，这些参数会被传递给methodName参数。如果你使用这些可选的参数，则你必须为resultObject传递一些值；如果你
不希望返回值的话，传递null。

返回

如果一个对methodName的调用在客户机上是成功的，则返回一个布尔值true；否则，返回false。

描述

方法。在起源Flash客户机或在另一个服务器上执行一个方法。远端方法可以可选的返回数据，这些数据将作为resultObj参
数的结果被返回，假如提供了resultObj参数的话。典型的远端对象是一个连接到服务器的Flash客户机，但这个远端对象也可以
是另一个服务器。无论这个远端代理是一个Flash客户机还是另一个服务器，这个方法都是在远端代理的NetConnection对象上被
调用的。

例子

下面的例子展示了一个客户机端脚本，这个客户机端脚本定义了一个名为random的函数，这个函数会生成一个随机数字：

```
nc = new NetConnection();
nc.connect ("rtmp://someserver/someApp/someInst");
nc.random = function(){
   return (Math.random());
};
```

下面的服务器端脚本在application.onConnect处理器中使用Client.call方法来调用定义于客户机端的random方法。这个服务器
端脚本还定义了一个名为randHander的函数，这个函数作为resultObj参数被用于Client.call方法中。

```
application.onConnect = function(clientObj){
   trace("we are connected");
   application.acceptConnection(clientObj);
   clientObj.call("random", new randHandler());
};
randHandler = function(){
   this.onResult = function(res){
      trace("random num: " + res);
   }
   this.onStatus = function(info){
      trace("failed and got:" + info.code);
```

```
    }
};
```

Client."commandName"

可用性

Flash Communication Server MX

用法

function clientCommand([p1, ..., pN]){}

参数

p1, ..., pN 可选的参数，它们被传递给命令消息处理器，如果这个消息包含了你定义的参数的话。这些参数是被传递给
NetConnection.call()方法的ActionScript对象。

返回

一个你定义的ActionScript对象。只有当这个命令消息被提供了一个响应处理器时，这个对象才会作为对这个命令的一个响
应被编序并被发送到客户机。

描述

的NetConnection.call事件处理器。当一个，这会导致Flash Flash Communication ServerNetConnection.call在服务器端搜索这个Client对象，寻找一个匹配这个命令名参数的方
法。如果这个参数被发现了，则这个方法就会被调用，并且返回值会被发送回客户机。

例子

这个范例创建了一个名为这个范例创建了一个名为sumsum的方法，这个方法是作为服务器端的的方法，这个方法是作为服务器端的ClientClient对象对象newClientnewClient的一个属性存在的：的一个属性存在的：

```
newClient.sum = function sum(op1, op2){
    return op1 + op2;
};
```

然后，这个sum方法可以在Flash客户机端的NetConnection.call中进行调用，就像下面的范例这样：

然后，这个sum方法可以在Flash客户机端的NetConnection.call中进行调用，就像下面的范例这样：

```
nc = new NetConnection();
nc.connect("rtmp://myServer/myApp");
nc.call("sum", new result(), 20, 50);
function result(){
    this.onResult = function (retVal){
        output += "sum is " + retVal;
    };
    this.onStatus = function(errorVal){
        output += errorVal.code + " error occurred";
    };
}
```

这个sum方法也可在服务器端调用，就像这样：

```
newClient.sum();
```

下面的范例创建了两个函数，你可以从一个客户机端或服务器端脚本中调用它们：

```
application.onConnect = function(clientObj) {
  //添加一个名为“foo”的可调用函数；foo会返回数字
  clientObj.foo = function() {return 8;};
  //添加一个远端函数，这个函数没有定义在onConnect调用中
  clientObj.bar = application.barFunction;
};
//这个bar函数添加了它被提供的两个值
application.barFunction = function(v1,v2) {
  return (v1 + v2);
};
```

通过在一个客户机端脚本中使用下面的代码，你可以调用前面范例中定义的两个函数中的任何一个：

```
c = new NetConnection();
c.call("foo");
c.call("bar", null, 1, 1);
```

通过在一个服务器端脚本中使用下面的代码，你可以调用前面范例中定义的两个函数中的任何一个：

```
c = new NetConnection();
c.onStatus = function(info) {
  if(info.code == "NetConnection.Connect.Success") {
    c.call("foo");
    c.call("bar", null, 2, 2);
  }
};
```

Client.getBandwidthLimit()

可用性

Flash Communication Server MX

用法

clientObject.getBandwidthLimit(iDirection)

参数

iDirection 一个整数，指定了连接的方向：指示一个客户机到服务器的方向，指示一个服务器到客户机的方向。

返回

一个整数，指示每秒的字节。

描述

方法。返回客户机或服务器可用于这个连接的最大带宽。使用iDirection参数来获取这个连接的每一个方向的值。返回的值
指出了每秒的字节，并可以用Client.setBandwidthLimit()来改变。每个应用程序的用于一个连接的默认值被设置在Application.xml
文件中。

例子

下面的范例使用Client.getBandwidthLimit()结合iDirection参数来设置两个变量-clientToServer和

```
serverToClient：
    application.onConnect = function(newClient){
      var clientToServer= newClient.getBandwidthLimit(0);
      var serverToClient= newClient.getBandwidthLimit(1);
    };
```

Client.getStats()

可用性

Flash Communication Server MX

用法

clientObject.getStats()

返回

一个对象，其不同的属性对应着返回的每一个状态。

描述

方法。返回客户机的状态，包括发送和接收的总字节数、发送和接收的RTMP消息数、丢弃的RTMP消息数，以及客户机作
出对一个ping消息的响应需要花费多少时间。

例子

下面的范例使用Client.getStats()来把一个新客户机的状态输出到输出窗口：

```
application.onConnect( newClient ){
  stats = newClient.getStats();
  trace("Total bytes received : " + stats.bytes_in);
  trace("Total bytes sent : " + stats.bytes_out);
  trace("RTMP messages received : " + stats.msg_in);
  trace("RTMP messages sent: " + stats.msg_out);
  trace("RTMP messages dropped : " + stats.msg_dropped);
  trace("Ping roundtrip time: " + stats.ping_rtt);
}
```

Client.ip

可用性

Flash Communication Server MX

用法

Client.ip

描述

属性（只读）。包含Flash客户机的IP地址。

例子

下面的范例使用Client.ip属性来验证一个新客户机是否有一个特定的IP地址。结果将决定要执行哪个代码块。

```
application.onConnect = function(newClient, name){
   if (newClient.ip == "127.0.0.1"){
      //在这里插入代码
   } else {
      //在这里插入代码
   }
};
```

Client.ping()

可用性

Flash Communication Server MX

用法

clientObject.ping()

描述

方法；发送一个“ping”消息到一个客户机并等待一个响应。如果这个客户机响应了，则这个方法返回true；否则的话，返
回false。使用这个方法来确定客户机的连接是否依然是活动的。

例子

下面的onConnect函数ping连接的客户机并trace该方法的结果：

```
application.onConnect(newClient) {
      if (newClient.ping()){
            trace("ping successful");
      }
      else {
            trace("ping failed");
      }
}
```

Client.protocol

可用性

Flash Communication Server MX

用法

Client.protocol

描述

属性（只读）。包含一个字符串，这个字符串指出了客户机用来连接到服务器的协议。这个字符串可以是下列值中的一
个：

rtmp 在永久性套接字连接上的RTMP（Real Time Message Protocol--实时消息协议）

rtmpt 利用HTTP协议的RTMP隧道

rtmps 在一个SSL（Security Socket Layer--安全套接层）连接上的RTMP

要获得有关Flash Media Server中的HTTP隧道功能的更多信息，请看Flash Media Server 2的《客户机端ActionScript语言参
考》中的NetConnection.connect()条目。

例子

下面的例子检查了当一个客户机连接到这个应用程序时，这个客户机所使用的连接协议。

```
application.onConnect(clientObj) {
  if(clientObj.protocol == "rtmp") {
    trace("Client connected over a persistent connection");
  } else if(clientOjb.protocol == "rtmpt") {
    trace("Client connected over an HTTP tunneling connection");
  }
}
```

Client.readAccess

可用性

Flash Communication Server MX

用法

clientObject.readAccess

描述

属性。为这个客户机提供对包含着应用程序资源（共享对象和流）的目录的读访问权。要赋予一个客户机对包含着应用程
序资源的目录的读访问权，只需把这些目录列出在一个字符串中，并用分号来分隔每个目录。

默认情况下，所有的客户机都有完全的读权限，并且readAccess属性是被设置为斜杠（/）的。要赋予一个客户机读权限，
指定一个访问级列表（以URI格式），用分号分隔。位于一个指定的URI中的任何文件和目录也被认为是可访问的。例如，如果
myMedia被指定为一个访问级，那么，在myMedia目录中的任何文件和目录也都是可访问的（例如，myMedia/mp3s）。类似
的，在myMedia/mp3s目录中的任何文件和目录也都是可访问的，以此类推。

具有对一个包含流的目录的读访问权的客户机可以在这个特定的访问级播放流。具有对一个包含共享对象的目录的读访问
权的客户机可以在这个特定的访问级订阅共享对象并接收共享对象的改变通告。

对于流，readAccess控制着这个连接可以播放的流。

对于共享对象，readAccess控制着这个连接是否可以侦听到共享对象的改变。

技巧：尽管你不能使用这个属性来控制对某个单独文件的访问，但假如你想控制对它的访问的话，你可以单独为这个文件
创建一个目录。

例子

下面的onConnect函数赋予了一个客户机对myMedia/mp3s和myData/notes，及其子目录的读访问权：

```
application.onConnect = function(newClient, name){
  newClient.readAccess = "myMedia/mp3s;myData/notes";
};
```

Client.referrer

可用性

Flash Communication Server MX

用法

clientObject.referrer

描述

属性（只读）。一个字符串对象，该字符串的值被设置为发起这个连接的SWF文件或服务器的URL。

例子

下面的范例定义了一个onConnect回调函数，当客户机连接到这个应用程序时，这个函数会发送一个trace至输出窗口，指出
这个新客户机的起源：

```
application.onConnect = function(newClient, name){
   trace("New user connected to server from" + newClient.referrer);
};
```

Client.__resolve

可用性

Flash Communication Server MX

用法

Client.__resolve = function(propName){}

参数

propName 一个未定义属性的名字

参数

propName 一个未定义属性的名字

返回

这个未定义属性的值，这个未定义属性是由propName参数指定的。

描述

事件处理器。为未定义属性提供值。当一个Client对象的一个未定义属性被服务器端ActionScript代码引用时，那个对象会被
它们已经被定义过了。 __resolve方法的话，则这个__resolve方法就会被调用，并被传递以这个未定义属性的

例子

下面的范例定义了一个函数，任何时候一个未定义属性被引用，这个函数就会被调用：

```
Client.prototype.__resolve = function (name) {
   return "Hello, world!";
};
function onConnect(newClient){
//打印"Hello World"
   trace (newClient.property1);
//打印"Hello World"
   trace (newClient.property1);
}
```

Client.secure

Flash Media Server 2
用法
clientObject.secure
描述
属性（只读）；一个布尔值，指出一个Internet连接是安全的（true）还是不安全的（false）。

Client.setBandwidthLimit()

可用性

可用性
Flash Communication Server MX
用法
clientObject.setBandwidthLimit(iServerToClient, iClientToServer)
参数
iServerToClient 从服务器到客户机的带宽，以字节/秒为单位。如果你不想改变当前设置，使用。
iClientToServer 从客户机到服务器的带宽，以字节/秒为单位。如果你不想改变当前设置，使用。
返回
无
描述
方法。为这个客户机设置从客户机到服务器，从服务器到客户机，或是二者，的最大带宽。每个应用程序的一个连接的默
认值被设置在Application.xml文件中。指定的值不能超越在Application.xml文件中指定的带宽上限。
例子
下面的范例基于传递给onConnect函数的值，为每一个方向设置了带宽限制：

```
application.onConnect = function(newClient, serverToClient, clientToServer){
  newClient.setBandwidthLimit(serverToClient, clientToServer);
  application.acceptConnection(newClient);
}
```

Client.uri

可用性
Flash Media Server 2
用法
clientObject.uri
描述

属性（只读）；要连接到这个应用程序实例的客户机确定的URI。

例子

下面的范例定义了一个onConnect回调函数，它发送一条消息指出新客户机用于连接到这个应用程序的URI（Uniform
Resource Identifier--统一资源标识符）：

```
application.onConnect = function(newClient, name){
    trace("New user requested to connect to " + newClient.uri);
};
```

Client.virtualKey

可用性

Flash Media Server 2

用法

clientObject.virtualKey

描述

属性；客户机的用户代理类型，代表性的就是Flash Player版本，但它可以被设置成任何合法的键值。一个合法的键值可以
包含除“*”和“:”以外的任何字符构成的字符串。

Client.writeAccess

可用性

Flash Communication Server MX

用法

clientObject.writeAccess

描述

属性。为这个客户机提供对包含着应用程序资源（共享对象和流）的目录的写访问权。要赋予一个客户机对包含着应用程
序资源的目录的写访问权，只需把这些目录列出在一个字符串中，并用分号来分隔每个目录。

默认情况下，所有的客户机都有完全的写权限，并且writeAccess属性是被设置为斜杠（/）的。要赋予一个客户机写权限，
指定一个访问级列表（以URI格式），用分号分隔。位于一个指定的URI中的任何文件和目录也被认为是可访问的。例如，如果
myMedia被指定为一个访问级，那么，在myMedia目录中的任何文件和目录也都是可访问的（例如，myMedia/mp3s）。类似
的，在myMedia/mp3s目录中的任何文件和目录也都是可访问的，以此类推。

具有对一个包含流的目录的写访问权的客户机可以在这个特定的访问级播放流。具有对一个包含共享对象的目录的写访问
权的客户机可以在这个特定的访问级订阅共享对象并接收共享对象的改变通告。

对于流，writeAccess控制着谁可以发布和记录流。

对于共享对象，writeAccess控制着谁可以创建和更新这些共享对象。

技巧：尽管你不能使用这个属性来控制对某个单独文件的访问，但假如你想控制对它的访问的话，你可以单独为这个文件
创建一个目录。

技巧：不要在客户机端使用前导斜杠来前缀流路径。

例子

下面的范例提供了对/myMedia/myStreams和myData/notes目录的写访问权。

```
application.onConnect = function(newClient, name){
  newClient.writeAccess = "/myMedia/myStreams;myData/notes";
  application.acceptConnection();
};
```

File类

可用性

Flash Media Server 2

File类可以让应用程序写入服务器的文件系统。这对于在不使用数据库服务器的情况下存储信息、为进行调试而创建日志文
件，或是跟踪使用都是很有用的。同样的，目录列表对于在不使用Flash Remoting的情况下建立流和共享对象的内容列表也是很
有用的。

File对象允许访问服务器文件系统。为了避免被滥用，Flash Media Server允许访问为应用程序实例正运行于其上的虚拟主机
而指定的沙箱中的文件。

提示：沙箱是一种安全机制，用以决定一个应用程序可以怎样与本地文件系统、网络，或是同时与本地文件系统和网络进
行交互。约束一个文件可以怎样与本地文件系统货网络进行交互有助于保护你的计算机和文件的安全。

服务器管理员可以设置用于一个虚拟主机中所有应用程序的沙箱，并可以为个别的应用程序提供额外的控制，假如需要的
话。下面的规则是由服务器强制实施的：

File对象不能利用文件路径规范进行创建。

默认情况下，一个脚本只能访问其宿主应用程序的application目录中的文件和目录。

可以建立一个虚拟目录映射来访问application目录外部的文件。

斜杠（/）被用作路径分隔符。

File对象路径遵循URI约定。

如果一个路径中包含的一个反斜杠（\）或一个点（.）或两个点（..）是路径分隔符间被找到的唯一的字符构成的话，
访问会被拒绝。

Root对象不能被重命名或删除。

例如，如果一个利用斜杠（/）的路径被用于创建一个File对象，则application文件夹会被映射。

通过为File对象路径指定虚拟目录映射，应用程序可以被允许访问额外的目录。

这是通过在Application.xml文件的JSEngine部分中指定一个FileObject标签来实现的，就像下面的范例中展示的这样：

<JSEngine>
<FileObject>

</FileObject> <VirtualDirectory>/videos;C:\myvideos</VirtualDirectory>

</JSEngine> <VirtualDirectory>/fcsapps;C:\Program Files\fcs\applications</

这个范例在默认的application目录外还指定了两个额外的目录映射。任何由/videos/xyz/vacation.flv指定的路径都将被映

射到c:/myvideos/xyz/vaction.flv。类似的，/fcapps/conference被映射到c:/Program Files/fcs/applications/conference。任何不匹

配这个映射的路径都将被解析到默认的application文件夹。例如，如果c:/myapps/filetest是application目录的话，

则/streams/hello.flv会被映射到c:/myapps/filetest/streams/hello.flv。

当下列File类的方法失败时，File对象调用Application.onStatus事件处理器来报告错误：

File.copyTo()

当下列File类的方法失败时，File对象调用Application.onStatus事件处理器来报告错误：

File.copyTo()
File.flush()
File.list()
File.mkdir()
File.open()
File.read()
getGlobal()
File.remove()
File.renameTo()
File.write()

File类的方法汇总

方法	描述
File.close()	关闭这个文件。
File.copyTo()	把一个文件拷贝到一个不同的位置或是使用一个不同的文件名拷贝到相同的位置。
File.eof()	返回一个布尔值，指出文件指针是（true）否（false）位于文件的末尾。
File.flush()	清出一个文件的输出缓冲区。
File.list()	如果这个文件是一个目录，则返回一个数组，数组中的每一个元素对应着这个目录中的每一个文件。
File.mkdir()	试图在这个文件目录中创建一个目录。
File.open()	打开一个文件以便你可以从中进行读取或对其进行写入。

File.read()	从一个文件中读取指定数量的字符并返回一个字符串。
File.readAll()	读取这个文件并返回一个数组，数组中的每一个元素对应着文件中的每一行。
File.readByte()	从这个文件中读取下一个字节并返回下一个字节的数字值，如果操作失败则返回-1。
File.readln()	从这个文件中读取下一行并作为一个字符串返回。
File.remove()	移除由File对象所指向的文件或目录。
File.renameTo()	移动或重命名一个文件。
File.write()	把数据写入到一个文件。
File.writeAll()	获取一个数组作为参数并在数据中的每一个元素上调用File.writeln()方法。
File.writeByte()	把一个字节写入到一个文件。
File.writeln()	把数据写入到一个文件并在输出最后一个参数后添加一个与平台有关的行尾字符。
getGlobal()	跳过指定数量的字节并返回新的文件位置。
File.toString()	返回一个字符串，指出File对象的名字。

File类的属性汇总

属性	描述
File.canAppend	只读；一个布尔值，指出一个文件是（true）否（false）可以为追加目的而打开。
File.canRead	只读；一个布尔值，指出一个文件是（true）否（false）可以被读取。
File.canReplace	只读；一个布尔值，指出一个文件是（true）否（false）以替换标记启用模式被打开。
File.canWrite	只读；一个布尔值，指出你是（true）否（false）可以写入到一个文件。
File.creationTime	只读；一个数据对象，包含这个文件被创建的时间。
File.exists	只读；一个布尔值，指出这个文件或目录是（true）否（false）存在。
File.isDirectory	只读；一个布尔值，指出这个文件是（true）否（false）是一个目录。
File.isFile	只读；一个布尔值，指出这个文件是（true）否（false）是一个常规数据文件。
File.isOpen	只读；一个布尔值，指出这个文件是（true）否（false）是打开的。
File.lastModified	只读；一个日期对象，包含文件最近一次被修改的时间。
File.length	只读；对于一个目录而言，是这个目录中文件的数量，当前目录和父目录条目不计算在内；对于一个文件而言，是文件中的字节数。
File.mode	只读；一个打开文件的模式。
File.name	只读；一个字符串，指出这个文件的名字。
File.position	这个文件中当前的偏移量。
File.type	只读；一个字符串，指定当一个文件被打开时所使用的编码或数据的类型。

File类的构造器

可用性

Flash Media Server 2

用法

fileObject = new File(name)

参数

name 指定文件或目录的名字。这个名字只可以包含UTF-8编码的字符；高字节值可以使用URI字符编码方案进行编码。利
用Application.xml文件中指定的映射，这个被指定的名字会被映射到一个系统路径。如果这个路径是无效的，则这个对象的name
属性就会被设置成一个空字符串，且没有文件操作可以被执行。

返回

如果成功，返回一个File对象；否则，返回null。

描述

例子 File类的一个实例，使用对象创建的标准语法。

下面的代码创建File类的一个实例：

var errorLog = new File("/logs/error.txt");

File.canAppend

可用性

可用性

Flash Media Server 2

用法

fileObject.canAppend

描述

属性（只读）；一个布尔值，指出一个文件是（true）否（false）可以为追加目的而打开。对于关闭的文件，这个属性是
undefined的。

File.canRead

可用性

Flash Media Server 2

用法

fileObject.canRead

描述

属性（只读）；一个布尔值，指出一个文件是（true）否（false）可以被读取。

File.canReplace

可用性
Flash Media Server 2
用法
fileObject.canReplace
描述
属性（只读）；一个布尔值，指出一个文件是（true）否（false）以创建模式被打开。对于关闭的文件，这个属性是
undefined的。

File.canWrite

可用性
Flash Media Server 2
用法
fileObject.canWrite
描述
属性（只读）；一个布尔值，一个布尔值，指出你是（true）否（false）可以写入到一个文件。
注意：如果File.open()被调用以打开文件，则这个被打开文件中的模式是必须被遵守的。例如，如果这个文件是以读取模式
被打开的，则你可以从这个文件中读取，但你不能写入到这个文件中。

File.close()

可用性
Flash Media Server 2
用法
fileObject.close()
参数
无。
返回
一个布尔值，指出这个文件是（true）否（false）被成功的关闭。
描述
方法；关闭这个文件。如果这个文件不是打开的，则返回false。当对象超出其作用域时，这个方法会自动在一个打开的File
对象上被调用。
例子

下面的代码关闭/path/file.txt文件：

```
if (x.open("/path/file.txt", "read") ){
    // 在这里做某些事情。
    x.close();
```

File.copyTo()

可用性

Flash Media Server 2

用法

Flash Media Server 2

用法

fileObject.copyTo(name)

参数

name 指定目的文件的名字。这个名字只可以包含UTF-8字符；高字节值可以使用URI字符编码方案进行编码。利用
Application.xml文件中指定的映射，这个被指定的名字会被映射到一个系统路径。如果这个路径是无效的或者目的文件不存在，
则这个操作会失败，且这个方法会返回false。

描述

方法；把一个文件拷贝一个不同的位置，或是使用一个不同的文件名将其拷贝到相同的位置。如果源文件不存在或者如果
源文件是一个目录的话，这个方法会返回false。

例子

下面的代码把由myFileObj文件对象设置的这个文件拷贝到由参数提供的位置：

```
if (myFileObj.copyTo( "/logs/backup/hello.log")){
    // 在这里做某些事情。
}if (myFileObj.copyTo( "/logs/backup/hello.log")){
    // 在这里做某些事情。
}
```

File.creationTime

可用性

fileObject.creationTimeFlash Media Server 2

描述用法

方法（只读）；一个Date对象，包含这个文件被创建的时间。

方法（只读）；一个Date对象，包含这个文件被创建的时间。

File.eof()

File.eof()

可用性
Flash Media Server 2
用法
fileObject.eof()
参数
无。
返回
一个布尔值。
描述
方法（只读）；返回一个布尔值，指出文件指针是（true）否（false）指向文件的末尾。如果文件被关闭，则这个方法返回
true。
例子
下面的while语句可以让你插入代码，这些代码会一直执行直到文件的指针到达文件的末尾：

```
while (!myFileObj.eof()){
// 在这里做某些事情。
}
```

File.exists

可用性
Flash Media Server 2
用法
fileObject.exists
描述
属性（只读）；一个布尔值，指出这个文件或目录是（true）否（false）是存在的。

File.flush()

可用性
Flash Media Server 2
用法
fileObject.flush()
参数

无。

返回

一个布尔值；返回一个布尔值，指出清出操作是（true）否（false）成功。

描述

方法；清出一个文件的输出缓冲区。如果这个文件被关闭，则这个操作会失败。

例子

下面的if语句可以让你插入代码，如果myFileObj清出成功，则这些代码就会执行：

```
if (myFileObj.flush()){
    // 在这里做某些事情。
}
```

File.isDirectory

可用性

Flash Media Server 2

用法

fileObject.isDirectory

描述

属性（只读）；一个布尔值，指出这个文件是（true）否（false）是一个目录。

例子

一个描述目录的File对象具有一些用来描述该目录中所包含的文件的属性。这些属性具有与该目录中的文件相同的名字，就
像下面的例子中展示的这样：

```
myDir = new File("/some/directory");
myFileInDir = myDir.foo;
```

下面的例子使用命名的属性来查找引用没有合法属性名的文件：

```
mySameFileInDir = myDir["foo"];
myOtherFile = myDir["some long filename with spaces and such"];
```

File.isFile

可用性

Flash Media Server 2

用法

fileObject.isFile

描述

属性（只读）；一个布尔值，指出一个文件是（true）否（false）是一个数据文件。

File.isOpen

可用性
Flash Media Server 2
用法
fileObject.isOpen
描述
属性（只读）；一个布尔值，指出一个文件是（true）否（false）是打开的。
注意：目录不需要被打开。

File.lastModified

可用性
Flash Media Server 2
用法
fileObject.lastModified
描述
属性（只读）；一个Date对象，包含文件最近一次被修改的时间。

File.length可用性

Flash Media Server 2
用法
fileObject.length
描述
属性（只读）；对于一个目录而言，是这个目录中文件的数量，当前目录和父目录条目不计算在内；对于一个文件而言，
是文件中的字节数。

属性（只读）；对于一个目录而言，是这个目录中文件的数量，当前目录和父目录条目不计算在内；对于一个文件而言，
是文件中的字节数。

File.list()

可用性
Flash Media Server 2
用法
fileObject.list(filter)
参数
filter 一个Function对象，用来决定返回的数组中的文件。当一个文件的名字被作为参数传递给它时，

如果这个函数返回
true，则这个文件会被添加到由File.list()返回的数组中。这个参数是可选的，并允许你过滤此调用的结果。

返回

一个Array对象，包含目录中所有的文件对象。

描述

方法；如果这个文件是一个目录，则返回一个数组，其中的每一个元素对应这个目录中的每一个文件。

例子

下面的例子返回当前目录中其名字由三个字符构成的文件：

```
var a = x.currentDir.list(function(name){return name.length==3;});
```

File.mkdir()

可用性

Flash Media Server 2

用法

```
fileObject.mkdir(newDir)
```

参数

newDir 一个字符串，指出新目录的名字。这个名字是与当前File对象实例相关的。

返回

一个布尔值；在这个文件目录创建一个目录。

例子

下面的例子在myFileObject实例中创建一个logs目录：

```
if (myFileObject.mkdir("logs")){
    // 如果一个logs目录被成功创建就做一些事情。
}
```

File.mode

可用性

Flash Media Server 2

用法

```
fileObject.mode
```

描述

属性（只读）；一个打开文件的模式。它可以与为打开这个文件而传递的mode参数不同，假如你使用了重复的属性（例
如，"read, read"）或是如果某些属性被省略了。如果这个文件被关闭，则这个属性是undefined。

File.name

可用性

Flash Media Server 2

用法

fileObject.name

描述

属性（只读）；一个字符串，指出这个文件的名字。如果这个File对象是用一个非法的路径被创建的，则这个值会是一个空
字符串。

File.open()

可用性

Flash Media Server 2

用法

fileObject.open("type","mode")

参数

type 一个字符串，指出这个文件的编码类型。下面的类型是被支持的：

"text" 使用默认的文件编码为文本访问的目的打开这个文件。

"binary" 为二进制访问打开这个文件。

"utf8" 为UTF-8访问打开这个文件。

mode 一个字符串，指出打开这个文件的模式。下面的模式是合法的且可以被组合（模式是大小写敏感的，多个模式之间必
需用逗号分隔）：

"read" 指出这个文件是为读取而打开的。

"write" 指出这个文件是为写入而打开的。

"readWrite" 指出这个文件是为读取和写入而打开的。

"append" 为写入打开这个文件。在你试图写入这个文件时，这个文件的指针会被定位到文件的末尾。

"create" 如果这个文件不存在就创建一个新文件。如果这个文件存在，它的内容会被破坏。

注意：如果"read"和"write"都被设置了，则"readWrite"会被自动设置。

返回

一个布尔值，指出这个文件是（true）否（false）被成功打开。

描述

open()open()方法。没有默认值用于type和

例子 type和modemode参数-这些值必须被指定。参数-这些值必须被指定。

下面的if语句可以让你插入代码，当一个文本文件以读取模式被打开时，这些代码会被执行：

```
if (myFileObject.open("text", "read") ){
    // 在这里做某些事情。
}
```

File.position

可用性

Flash Media Server 2

用法

fileObject.position

描述

属性；文件中当前的偏移量。这是唯一一个可以被设置的属性。设置这个属性会在这个文件中执行一个搜索操作。这个属
性对于关闭的文件是undefined。

File.read()

可用性

Flash Media Server 2

用法

fileObject.read(numChars)

参数

numChars 一个整数，指定了要读取的字符数。如果numChars指定了比文件中剩余的字符更多的字符数，则这个方法会读
取到文件的末尾。numChars 一个整数，指定了要读取的字符数。如果numChars指定了比文件中剩余的字符更多的字符数，则这个方法会读
取到文件的末尾。

返回

一个字符串。

描述

方法；从一个文件中读取指定数量的字符数并返回一个字符串。如果这个文件是以二进制模式打开的，则这个操作将会失
败。

例子

下面的代码以读取模式打开一个文本文件并以打头的个字符、一行或一个字节来设置变量：

```
if (myFileObject.open( "text", "read") ){
    strVal = myFileObject.read(100);
    strLine = myFileObject.readln();
    strChar = myFileObject.readByte();
}
```

File.readAll()

可用性

Flash Media Server 2

用法

fileObject.readAll()

参数

无。

返回

一个数组。

描述

方法；从文件指针的位置后读取文件并返回一个数组，数组中的每一个元素对应文件中的每一行。如果这个文件是以二进
制被打开的，则这个操作会失败。

File.readByte()

可用性

Flash Media Server 2

用法

fileObject.readByte()

参数

无。

返回

一个数字；不是一个正整数就是-1。

描述

方法；从这个文件中读取下一个字节并返回下一个字节的数字值或-1，假如这个操作失败的话。如果这个文件不是以二进
制打开的，则这个操作会失败。

File.readln()

可用性

Flash Media Server 2

用法

fileObject.readln()

参数

无。

返回

一个字符串。

描述

方法；从这个文件中读取下一行并作为一个字符串返回。行分隔符字符（Windows上的\r\n或Linux上

的\n）不包含在这个
字符串中。字符\r会被挑过；\n决定行的实际末尾。如果这个文件是以二进制模式打开的，则这个操作会失败。

File.remove()

可用性
Flash Media Server 2
用法
fileObject.remove()
参数
无。
返回
一个布尔值，指出这个文件或目录是（true）否（false）被成功的移除。
描述
方法；移除由这个File对象指定的文件或目录。如果这个文件是打开的、路径指向一个根文件夹，或者目录不是空的，则这
个方法会返回false。
例子
下面的if语句可以让你在myFileObject被移除时执行代码：

```
if (myFileObject.remove()){
    // 在这里做某些事情。
}
```

File.renameTo()

可用性
Flash Media Server 2
用法
fileObject.renameTo(name)
参数
name 这个文件或目录的新名字。这个名字只可以包含UTF-8编码的字符；高字节值可以使用URI字符编码方案进行编码。
利用Application.xml文件中指定的映射，这个被指定的名字会被映射到一个系统路径。如果这个路径是无效的或这个目的文件不
存在的话，则这个操作将会失败。
返回
一个布尔值，指出这个文件是（true）否（false）被成功的重命名或移动。
描述
方法；移动或重命名一个文件。如果这个文件是打开的或这个目录指向根目录的话，则这个操作将失

败。

例子

下面的代码可以让你在myFileObject被重命名时执行代码：

```
if (myFileObject.renameTo( "/logs/hello.log.old")){
    // 在这里做某些事情。
}
```

File.seek()

可用性

Flash Media Server 2

用法

fileObject.seek(numBytes)

参数

numBytes 一个整数，指出要把文件指针从当前位置移动的字节数。

返回

如果这个操作是成功的，则返回文件中的当前位置；否则的话，返回-1。如果文件是被关闭的，则这个操作会失败且会报
告一个警告。如果这个操作是在一个目录上调用的，则会返回undefined。

描述

方法；跳过一个指定的字节数并返回新的文件位置。这个方法既可以接受正数也可以接受负数作为参数。

例子

下面的范例中，如果这个seek调用是成功的，则if语句中的代码就会运行：

```
if (fileObj.seek(10) != -1){
// 在这里做某些事情。
}
```

File.toString()

可用性

Flash Media Server 2

用法

fileObject.toString()

参数

无。

返回

一个字符串。

描述

方法；返回至File对象的路径。

例子

下面的范例输出File对象myFileObject的路径：

```
trace(myFileObject.toString());
```

File.type

可用性

Flash Media Server 2

用法

fileObject.type

描述

属性（只读）；一个字符串，指定当一个文件被打开时所使用的数据或编码的类型。支持的字符串包（Byte Order Mark"text"、"utf8"、-字节顺序标记）被检测到，则这个"binary"。这个属性对于目录和被关闭的文件是type属性会被设置为undefined"utf8"。 "text"模式打开且UTF-8 BOM

File.write

可用性

Flash Media Server 2

用法

fileObject.write(param0, param1,...paramN)

fileObject.write(param0, param1,...paramN)

参数

param0, param1,...paramN 要写入文件的参数。

返回

一个布尔值，指出写入操作是（true）否（false）是成功的。

描述

方法；把数据写入一个文件。write()方法把每一个参数转换成一个字符串，然后在不使用分隔符的情况下把它写入文件。

文件内容会在内部进行缓冲。File.flush()方法会把缓冲区中的内容写入到磁盘上的文件。

例子

下面的范例在myFileObject文本文件的末尾写入“Hello world”：

```
if (myFileObject.open( "text", "append") ) {
    myFileObject.write("Hello world");
}
```

File.writeAll()

可用性

Flash Media Server 2
用法
fileObject.writeAll(array)
参数
array 一个Array对象，包含了要写入文件的所有元素。
返回
一个布尔值，指出写入操作是（true）否（false）是成功的。
描述
方法；获取一个数组作为参数，并在数组中的每一个元素上调用File.writeln()方法。
文件内容会在内部进行缓冲。File.flush()方法会把缓冲区中的内容写入到磁盘上的文件。

File.writeByte()

可用性
Flash Media Server 2
用法
fileObject.writeByte(number)
参数
number 要写的数字值。
返回
一个布尔值，指出写入操作是（true）否（false）是成功的。
描述
方法；把一个字节写入一个文件。
文件内容会在内部进行缓冲。File.flush()方法会把缓冲区中的内容写入到磁盘上的文件。
例子
下面的范例把字节写入到myFileObject文件的末尾：

```
if (myFileObject.open("text","append")) {
    myFileObject.writeByte(65);
}
```

File.writeByte()

可用性
Flash Media Server 2
用法
fileObject.writeByte(number)
参数
number 要写的数字值。
返回
一个布尔值，指出写入操作是（true）否（false）是成功的。
描述

方法；把一个字节写入一个文件。
文件内容会在内部进行缓冲。File.flush()方法会把缓冲区中的内容写入到磁盘上的文件。
例子
下面的范例把字节写入到myFileObject文件的末尾：

```
if (myFileObject.open("text","append")) {
    myFileObject.writeByte(65);
}
```

File.writeln()

可用性
Flash Media Server 2
用法

```
fileObject.writeln(param0, param1,...paramN)
```

参数
param0, param1,...paramN 要写入文件的字符串。
描述
方法；把数据写入到一个文件并在最后一个参数的后面添加一个与平台有关的行尾字符。
文件内容会在内部进行缓冲。File.flush()方法会把缓冲区中的内容写入到磁盘上的文件。
例子
下面的范例为写入操作打开一个文本文件并写入一行：

```
if (fileObj.open( "text", "append") ) {
    fileObj.writeln("This is a line!");
}

}
```

getGlobal()

可用性
getGlobal()
参数
无。
返回
无。
描述
无。 secure.asc文件被装载时提供对全局对象的访问。使用这个全局对象来操作内建的全局函数。为了避免无
描述
方法（全局）；在secure.asc文件被装载时提供对全局对象的访问。使用这个全局对象来操作内建的全局函数。为了避免无
意中对全局对象的访问，应该总是把对它的引用保存在一个临时变量（以var声明的）中；不要把它的引用

保存在一个成员变量
或全局变量中。这个函数只可以在secure.asc文件中使用。

Flash Media Server 2有两个脚本执行模式：安全和普通。在安全模式中，只有secure.asc文件（假如它存在的话）被装载并
计算–没有其他的应用程序脚本被装载。同样的，全局getGlobal()和protectObject()函数仅在安全模式是可用的。它们是非常强大
的，因为它们提供了对脚本执行环境的完全访问，并可以让你创建系统对象。一旦secure.asc文件被装载，这个服务器就会切换
到普通脚本执行模式，直到这个应用程序被卸载。

使用getGlobal()函数来创建受保护的系统调用。要获得更多信息，参看《开发媒体应用程序》中的“实现安全系统对象”。

例子

下面的代码获得对全局对象的引用：

```
var global = getGlobal();
```

load()

可用性

Flash Communication Server MX

用法

```
load(filename);
```

参数

filename 从main.asc文件至一个ActionScript文件的相对路径。

返回

无

描述

方法（全局）。在main.asc文件中装载一个ActionScript文件。只有在这个ActionScript文件首次被装载时，这个方法才会被
执行。被装载的文件会在main.asc文件被成功装载、编译并执行后，在application.onAppStart()被执行前，被编译和执行。这个被
指定文件的路径是相对于main.asc被解析的。这个方法用于装载ActionScript库。

例子

下面的范例装载myLoadedFile.as文件：

```
load("myLoadedFile.as");
```

LoadVars类

可用性

Flash Media Server 2

LoadVars类可以让你把一个对象中的所有变量发送到一个指定的URL并把一个指定的URL处的所有变量装载到一个对象。

它也可以让你发送特定的变量，而不是所有的变量，这可以使你的应用程序变得更加的有效。你可以使用LoadVars.onLoad处理
器来确保你的应用程序在数据被装载后运行，而不是在那之前就运行。

LoadVars类工作起来就像XML类；它使用load()、send()和sendAndLoad()方法与服务器进行通讯。LoadVars类和XML类之间
的主要差别在于LoadVars传输ActionScript的名称和值对，而不是XML对象中存储的XML文档对象模型（DOM）树。LoadVars类
与XML类遵循相同的安全限制。

LoadVars类的方法汇总

方法	描述
LoadVars.addRequestHeader()	添加或更改与POST动作一起发送的HTTP请求标题（如Content-Type或SOAPAction）。
LoadVars.decode()	把查询字符串转换为指定的LoadVars对象的属性。
LoadVars.getBytesLoaded()	返回最近一次或当前LoadVars.load()或LoadVars.sendAndLoad()方法调用所装载的字节数。
LoadVars.getBytesTotal()	返回在所有的LoadVars.load()或LoadVars.sendAndLoad()方法调用期间所装载的总字节数。
LoadVars.load()	从指定的URL下载变量，解析这个变量的数据，然后将结果变量放置在一个LoadVars对象中。
LoadVars.send()	把myLoadVars对象中的变量发送到指定的URL。
LoadVars.sendAndLoad()	把myLoadVars对象中的变量发送到指定的URL，并可以把服务器响应的数据装载到一个LoadVars对象中。
LoadVars.toString()	以MIME内容编码格式application/x-www-urlform-encoded返回一个包含了myLoadVars中所有可枚举变量的字符串。

LoadVars类的属性汇总

属性	描述
LoadVars.contentType	在您调用LoadVars.send()或LoadVars.sendAndLoad()方法时发送到服务器的MIME类型。
LoadVars.loaded	一个布尔值，指示一个LoadVars.load()或LoadVars.sendAndLoad()操作是（true）否（false）已完成。

LoadVars类的事件汇总

事件处理器	描述
LoadVars.onData	当数据从服务器上完全下载时，或者当从服务器下载数据的过程中出现错误时调用。
LoadVars.onHTTPStatus	当Flash Media Server接收一个来自服务器的HTTP状态代码时调用。
LoadVars.onLoad	当LoadVars.load()或LoadVars.sendAndLoad()操作已完成时调用。

LoadVars类的构造器

可用性
Flash Media Server 2
用法
new LoadVars()
参数
无。
描述
构造器；创建一个LoadVars对象。你可以使用LoadVars对象的方法来发送和装载数据。
例子
下面的范例创建一个名为my_lv的LoadVars对象：

```
var my_lv = new LoadVars();
```

LoadVars.addRequestHeader()

可用性

Flash Media Server 2

用法

myLoadVars.addRequestHeader(header, headerValue)

参数

header 一个字符串或字符串数组，表示一个HTTP请求标题名称。

headerValue 一个字符串，表示与header关联的值。

返回

无。

描述

方法；添加或更改与POST动作一起发送的HTTP请求标题（如Content-Type或SOAPAction）。这个方法有两种可能的使用
情况：你可以传递两个字符串、header和headerValue，或者你可以传递一个字符串数组、替换的标题名称和标题值。

如果通过多次调用来设置相同的标题名称，则每个后继值将替换在上一次调用中设置的值。

下列标准HTTP标题不能用这个方法来添加或改变用这个方法来添加或改变：Accept-Ranges、Age、Allow、Allowed、
Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-
Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、
Vary、Via、Warning和WWW-Authenticate。

例子

下面的范例把一个定制的名为SOAPAction具有值Foo的HTTP标题名称添加到my_lv对象：

```
my_lv.addRequestHeader("SOAPAction", "'Foo'");
```

下面的范例创建了一个名为headers的数组，它包含两个可交替的HTTP标题和与其相关联的值。这个数组被作为参数传递
给addRequestHeader()方法。

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];
my_lv.addRequestHeader(headers);
```

下面的范例创建了一个新的LoadVars对象，添加了一个被称为FLASH-UUID的请求的标题。这个标题包含一个服务器可以
检查的变量。

```
var my_lv = new LoadVars();
my_lv.addRequestHeader("FLASH-UUID", "41472");
my_lv.name = "Mort";
my_lv.age = 26;
my_lv.send("http://flash-mx.com/mm/cgivars.cfm", "_blank", "POST");
```

LoadVars. contentType

可用性

Flash Media Server 2

用法

myLoadVars.contentType

描述

属性；在您调用LoadVars.send()或LoadVars.sendAndLoad()方法时发送到服务器的MIME类型。默认值为application/x-www-
rulform-encoded。

例子

下面的示例创建一个LoadVars对象，并显示发送到服务器的数据的默认内容类型。

```
var my_lv = new LoadVars();
trace(my_lv.contentType);

// 输出：application/x-www-form-urlencoded
```

LoadVars.decode()LoadVars.decode()

可用性

Flash Media Server 2

用法

myLoadVars.decode(queryString)

参数

queryString 一个包含名称/值对的URL编码的查询字符串。

返回

无。

描述

方法；将查询字符串转换为指定LoadVars对象的属性。此方法由LoadVars.onData事件处理函数内部使用。如果您覆盖
LoadVars.onData事件处理函数，则可以显式调用LoadVars.decode()来分析变量字符串。

例子

下面的示例跟踪三个变量：

```
// 创建一个新的LoadVars对象
var my_lv = new LoadVars();
//把变量字符串转换成属性
my_lv.decode("name=Mort&score=250000");
trace(my_lv.toString());
// 遍历my_lv中的属性
for (var prop in my_lv) {
      trace(prop+" -> "+my_lv[prop]);
```

LoadVars.getBytesLoaded()

可用性

Flash Media Server 2

用法Flash Media Server 2myLoadVars.getBytesLoaded()用法
myLoadVars.getBytesLoaded()
参数
无。
返回
一个Number对象。
描述
方法；返回最近一次或当前LoadVars.load()或LoadVars.sendAndLoad()方法调用所装载的字节数。
contentType属性的值不会影响getBytesLoaded()的值。

LoadVars.getBytesTotal()

可用性
Flash Media Server 2
用法
myLoadVars.getBytesTotal()
参数
无。
返回
一个Number对象。
返回
如果没有装载操作正在进行或者如果一个装载操作尚未开始，则getBytesTotal()方法将返回undefined。如果无法确定总字节
数-例如，如果下载已开始但服务器尚未传输一个HTTP Content-Length，则getBytesTotal()方法也将返回undefined。
描述
方法；返回在所有的LoadVars.load()或LoadVars.sendAndLoad()方法调用期间装载到一个对象中的总字节数。每次当一个对
load()或sendAndLoad()的调用被发出，getBytesLoaded()方法就会被重置，但getBytesTotal()方法会在原基础上继续增加。
contentType属性的值不会影响getBytesLoaded()的值。

LoadVars.load()

可用性
Flash Media Server 2
用法
myLoadVars.load(url)
参数
url 一个字符串，指出从其下载变量的URL。

返回

一个布尔值，指出成功（true）或失败（false）。

描述

方法；从指定的URL下载变量，解析变量数据，然后将结果变量放在一个LoadVars对象（myLoadVars）中。你可以从一个
远端URL或从一个本地文件系统的URL装载变量；二者应用的是同样的编码标准。

myLoadVars对象中任何具有与下载变量相同名字的属性将被重写。下载的数据必须是MIME内容类型application/x-www-
urlform-encoded。

LoadVars.load()方法调用是异步的。

例子

下面的代码定义了一个onLoad处理器函数，当数据被返回时，该处理器函数会发出信号：

```
var my_lv = new LoadVars();
my_lv.onLoad = function(success) {
    if (success) {
    trace(this.toString());
    } else {
    trace("Error loading/parsing LoadVars.");
    }
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

LoadVars.loaded

可用性

Flash Media Server 2

用法

myLoadVars.loaded

描述用法

myLoadVars.loaded

描述

属性；一个布尔值，指示一个LoadVars.load()或LoadVars.sendAndLoad()操作是（true）否（false）已经完成。

例子

下面的示例加载一个文本文件，并在操作完成时把信息写入日志文件：

```
var my_lv = new LoadVars();
my_lv.onLoad = function(success) {
    trace("LoadVars loaded successfully: "+this.loaded);
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

LoadVars.onData

可用性

Flash Media Server 2

用法

myLoadVars.onData(src){}

参数

src 一个字符串或undefined；来自LoadVars.load()或LoadVars.sendAndLoad()方法调用的原始（未解析）数据。

描述

事件处理器；当数据从服务器上完全下载时，或者当从服务器下载数据的过程中出现错误时调用。此处理函数在解析数据
之前调用，因此它可用于调用自定义解析例程，而不必调用Flash Player中的内置解析例程。对于分配给LoadVars.onData的函
数，传递给该函数的src参数的值可以是undefined，也可以是包含从服务器下载的URL编码名称/值对的字符串。如果src参数为
undefined，则从服务器下载数据时将出现错误。

LoadVars.onLoad的默认实现调用LoadVars.onData。您可以通过对LoadVars.onData分配自定义函数来覆盖此默认实现，但是
没有调用LoadVars.onLoad，除非在LoadVars.onData的实现中调用它。

例子

下面的范例装载一个文本文件并在操作完成时在一个名为content_ta的TextArea实例中显示内容。如果一个错误出现，则信
息会显示在输出面板中。如果一个错误出现，这个错误信息会写入到日志文件。

```
var my_lv = new LoadVars();
my_lv.onData = function(src) {
    if (src == undefined) {
        trace("Error loading content.");
        return;
    }
    content_ta.text = src;
};
my_lv.load("content.txt", my_lv, "GET");
```

LoadVars.onHTTPStatus

可用性

Flash Media Server 2

用法

myLoadVars.onHTTPStatus(httpStatus){}

参数

httpStatus 一个数字；由服务器返回的HTTP状态代码。例如，值为表示服务器尚未找到请求的URI的匹配项。在
ftp://ftp.isi.edu/in-notes/rfc2616.txt处的HTTP规范的.4和.5节中，可以找到HTTP状态代码。

描述

事件处理器；当Flash Media Server从服务器接收到一个HTTP状态代码时调用。这个处理器可以让你捕获HTTP状态代码并

根据该状态代码进行动作。

onHTTPStatus处理函数在onData前调用，它在加载失败时触发对值为undefined的onLoad的调用。触发onHTTPStatus后，不
管是否覆盖onHTTPStatus，始终触发onData。要充分地使用onHTTPStatus处理函数，应该编写一个函数来捕获onHTTPStatus调
用的结果；然后可以在onData和onLoad处理函数中使用该结果。如果未调用onHTTPStatus，则表示播放器没有尝试发出URL请
求。

如果Flash Media Server无法从服务器获取状态代码或无法与服务器进行通讯，则默认值会传递到你的ActionScript代码。

例子

下面的示例演示如何使用onHTTPStatus()帮助调试。此示例收集HTTP状态代码并将它们的值和类型分配给LoadVars对象的
实例。（请注意，此示例在运行时创建实例成员this.httpStatus和this.httpStatusType。）onData方法使用这些实例成员跟踪调试中
有用的HTTP响应的相关信息。

```
var myLoadVars = new LoadVars();

myLoadVars.onHTTPStatus = function(httpStatus) {
    this.httpStatus = httpStatus;
    if(httpStatus < 100) {
        this.httpStatusType = "flashError";
    }
    else if(httpStatus < 200) {
        this.httpStatusType = "informational";
    }
    else if(httpStatus < 300) {
        this.httpStatusType = "successful";
    }
    else if(httpStatus < 400) {
        this.httpStatusType = "redirection";
    }
    else if(httpStatus < 500) {
        this.httpStatusType = "clientError";
    }
    else if(httpStatus < 600) {
        this.httpStatusType = "serverError";
    }
}

myLoadVars.onData = function(src) {
    trace(">> " + this.httpStatusType + ": " + this.httpStatus);
    if(src != undefined) {
        this.decode(src);
        this.loaded = true;
        this.onLoad(true);
```

```
        }
        else {
            this.onLoad(false);
        }
    }

    myLoadVars.onLoad = function(success) {
    }

    myLoadVars.load("http://weblogs.macromedia.com/mxna/flashservices/getMostRecentPosts.cfm");
```

LoadVars.onLoad

可用性

Flash Media Server 2

用法

myLoadVars.onLoad(success){}

参数

success 一个布尔值，指示LoadVars.load()操作是以成功结束（true）还是以失败结束（false）。

描述

事件处理器；当LoadVars.load()或LoadVars.sendAndLoad()操作已完成时调用。如果变量的装载是成功的，则success参数为
true。如果变量没有被接收，或者如果在接收来自服务器的响应时发生了一个错误，则success参数为false。

如果success参数是true，则loadVarsObject会被由LoadVars.load()或LoadVars.sendAndLoad()操作所下载的变量填充，而这些
变量将在onLoad处理器被调用时变为可用。

默认情况下这个方法是未定义的，但你可以通过指派给它一个回调函数来定义它。

例子

下面的范例创建了一个新的LoadVars对象，试图从一个远端URL把变量装入它，并打印结果：

```
myLoadVars = new LoadVars();
myLoadVars.onLoad = function(result){
    trace("myLoadVars load success is " + result);
}
myLoadVars.load("http://www.someurl.com/somedata.txt");
```

LoadVars.send()

可用性

Flash Media Server 2

用法

myLoadVars.send(url [, target, method])

参数

url myLoadVars.send(url [, target, method])一个字符串；上载变量的目标URL。

参数

url 一个字符串；上载变量的目标URL。

target 一个File对象。如果你使用了这个可选的参数，则任何返回的数据都会被输出到这个指定的File对象。如果这个参数被
省略，则响应将被丢弃。

method 一个字符串，指出HTTP协议的GET或POST方法。默认值为POST。这个参数是可选的。

返回

方法；将myLoadVars对象中的变量发送到指定的URL。myLoadVars对象中的所有可枚举变量将连接成一个字符串，然后使
用HTTP POST方法将此字符串发送到URL。在HTTP请求标题中发送的MIME内容类型是LoadVars.contentType的值。

LoadVars.sendAndLoad()

可用性

可用性

Flash Media Server 2

用法

myLoadVars.sendAndLoad(url, target[, method])

参数

url 一个字符串；上载变量的目标URL。

method 一个字符串，指出HTTP协议的GET或POST方法。默认值为POST。这个参数是可选的。

返回

一个布尔值，指出成功（true）或失败（false）。

描述

方法；将myLoadVars对象中的变量发送到指定的URL。服务器的响应被下载，并将其作为变量数据进行解析，然后将结果
变量放在target对象中。变量发送的方式与LoadVars.send()相同。变量下载到target中的方式与LoadVars.load()相同。

LoadVars.toString()

可用性

Flash Media Server 2

用法

myLoadVars.toString()

参数

无。

返回

一个字符串。

描述

方法；以MIME内容编码格式application/x-www-form-urlencoded返回包含myLoadVars中所有可枚举变量的字符串。

例子

下面的示例实例化一个新LoadVars()对象并创建两个属性，然后使用toString()以URL编码的格式返回包含这两个属性的字符
串：

```
var my_lv = new LoadVars();
my_lv.name = "Gary";
my_lv.age = 26;
trace (my_lv.toString());
//输出: age=26&name=Gary
```

可用性

Flash Media Server 2

Log类可以让你创建一个Log对象，这个对象可以作为一个可选的参数传递给WebService类的构造器。

Flash Media Server 2

Log类可以让你创建一个Log对象，这个对象可以作为一个可选的参数传递给WebService类的构造器。

Log类的事件处理器汇总

事件处理器	描述
Log.onLog	当一个日志消息被发送到一个日志时被调用。

Log类的构造器

可用性

Flash Media Server 2

用法

new Log([logLevel][, logName])

参数

logLevel Level必须被设置为下列之一（如果没有显式设置的话，则level默认为Log.BRIEF）：

Log.BRIEF 接收重要的生存期事件和错误通知。

Log.VERBOSE 接收所有的生存期事件和错误通知。

Log.DEBUG 接收规格的细粒度的事件和错误。

logName 一个可选的参数，可以用来区别多个同时运行于同一输出的日志。

返回

一个Log对象。

描述

构造器；创建一个Log对象，它可以被作为一个可选的参数传递给WebService类的构造器。

例子

下面的范例创建了一个Log类的新实例：

newLog = new Log();

Log.onLog

可用性

Flash Media Server 2

用法

myLog.onLog(message){}

参数

message 一个日志消息。

返回

无。

描述

事件处理器；当一个日志消息被发送到一个日志时被调用。

NetConnection类

可用性

Flash Communication Server MX 1.0

服务器端NetConnection类可以让你在一个Flash Media Server应用程序实例和一个应用程序服务器、另一个Flash Media
Server，或同一台服务器上的另一个Flash Media Server应用程序实例间创建一个两路连接。你可以使用NetConnection对象来创建
强大的应用程序，例如，你可以从一个应用程序服务器得到天气信息，或是共享由其他Flash Media Server或应用程序实例装载
的一个应用程序。

你可以使用NetConnection.connect()方法利用标准协议（比如HTTP）连接到一个应用程序服务器进行服务器到服务器的交

互，或者利用Macromedia Real-Time Messaging Protocol（RTMP-实时消息协议）格式或SSL（RTMPS）连接到另一个Flash
Media Server以共享音频、视频和数据。Flash Media Server中的SSL使用一个被称为OpenSSL的第三方开源库。

NetConnection类的方法汇总

方法	描述
NetConnection.addHeader()	添加一个环境头。
NetConnection.call()	在一个远端服务器上调用一个方法或操作。
NetConnection.close()	关闭一个服务器连接。
NetConnection.connect()	连接到一个应用程序服务器或另一个Flash Media Server服务器。

NetConnection类的属性汇总

属性	描述
NetConnection.isConnected	只读；一个布尔值，指出一个连接是（true）否（false）已经做好了。
NetConnection.uri	只读；由NetConnection.connect()方法传递的URI。

NetConnection类的事件处理器汇总

NetConnection类的事件处理器汇总

事件处理器	描述
NetConnection.onStatus	当连接状态发生改变时被调用。

NetConnection类的构造器

可用性

Flash Communication Server MX 1.0

用法

new NetConnection()

参数

无。

返回

一个NetConnection对象。

描述

构造器；创建一个NetConnection类的新实例。

例子

下面的范例创建NetConnection类的一个新实例：

newNC = new NetConnection();

NetConnection.addHeader()

可用性

Flash Communication Server MX

用法

myNetConn.addHeader(name, mustUnderstand, object)

参数

name 一个字符串，用来识别头及其与之关联的ActionScript对象数据。

mustUnderstand 一个布尔值；true指示服务器在处理跟在头后面的任何东西或消息前，必须理解并处理这个头。

object 任何ActionScript对象。

返回

无

描述

方法。把一个环境头添加到AMF（Action Message Format-动作消息格式）包结构中。这个头会随每一个未来AMF包被发
送。如果你用相同的名字调用NetConnection.addHeader()，则新头会替换现有的头，并且新头会在NetConnection对象的持续期间
一直维持。你可以通过用要删除的头名和一个未定义的对象来调用NetConnection.addHeader()来删除一个头。

译者追加的额外解释：

为了把消息发送到远端服务和从远端服务接收消息，Flash Remoting MX使用了Action Message Format（AMF），这是一种
HTTP设计用于来回编解码数据类型。仿效ActionScript SOAP（Simple Object Access ProtocolAMF， -简单对象访问协议），可以在Flash AMF使用一个包格式来转播消息。一个AMF包由下列部分组成：

包含AMF版本信息的包头

环境头计数

环境头数组，该数组包含的信息描述了每一条AMF消息将于其中被处理的环境。

消息计数

消息数组

例子

下面的范例创建了一个新的NetConnection实例nc，并连接到一个位于Web服务器www.foo.com的应用程序，这个应用程序
正在端口进行侦听。这个应用程序发送服务/blag/SomeCoolService。代码的最后一行添加了一个名为foo的头：

```
nc=new NetConnection();
nc=new NetConnection();nc.connect("http://www.foo.com:1929/blag/SomeCoolService");
nc.connect("http://www.foo.com:1929/blag/SomeCoolService");nc.addHeader("foo", true, new Foo());
nc.addHeader("foo", true, new Foo());
```

NetConnection.call()

可用性

Flash Communication Server MX

用法

```
myNetConnection.call(methodName, [resultObj, p1, ..., pN])
```

参数

methodName 一个以[objectPath/]method这种形式指定的方法。例如，命令someObj/doSomething告诉远端服务器要用所有的
p1, ..., pN参数调用clientObj.someObj.doSomething方法。

resultObj 一个可选的参数，用于处理来自服务器的返回值。结果对象可以是你定义的任何对象，并且，可以有两个定义的
方法来处理返回的结果–onResult和onStatus。如果一个错误被作为结果返回了，则onStatus会被调用；否则，onResult会被调用。

p1, ..., pN 可选的参数，它们可以是任何ActionScript类型，包括对另一个ActionScript对象的引用。当这个方法在远端应用
程序服务器上被执行时，这些参数会被传递给上面指定的methodName。

返回

对于RTMP连接，如果一个对methodName的调用是被发送到客户机的，则返回一个布尔值true；否则，返回false。对于应用
程序服务器连接，它总是返回true。

描述

方法。在一个Flash Media Server或应用程序实例连接到的一个应用程序服务器上调用一个命令或方法。NetConnection.call方
法在服务器上的工作与其在客户机上的工作是一样的：它在一个远端服务器上调用一个命令。

注意：如果你想要从一个服务器上调用一个客户机上的方法，使用Client.call()方法。

例子

这个范例使用RTMP来执行一个从一个Flash Media Server到另一个Flash Communication Server的调用。这个代码作了一个至
server 2上的App1应用程序的连接，然后调用server 2上的方法Sum：

```
nc1.connect("rtmp://server2.mydomain.com/App1", "svr2",);
nc1.call("Sum", new Result(), 3, 6);
```

下面的服务器端ActionScript代码是在server 2上。当客户机连接时，这个代码会检查看其是否有一个参

数等于svr1。如果这
个客户机有这样一个参数，则Sum方法就会被定义，以便当这个方法自svr1被调用时，svr2可以用适当的方法进行响应。

```
application.onConnect = function(clientObj){
  if(arg1 == "svr1"){
    clientObj.Sum = function(p1, p2){
      return p1 + p2;
    }
  }
  return true;
};
```

下面的范例使用一个AMF请求来执行对一个应用程序的调用。这允许Flash Communication Server连接到一个应用程序服务
器，然后调用quote方法。Java适配器通过把点左边的标识符用作类名，把点右边的标识符用作类的一个方法，发出了这个调
用。

```
nc = new NetConnection;
nc.connect("http://www.xyz.com/java");
nc.call("myPackage.quote", new Result());
```

NetConnection.close()

可用性

Flash Communication Server MX

用法

myNetConnection.close()

参数

无

返回

无

返回

无

描述

方法。关闭与服务器的连接。在你关闭连接之后，你可以重用这个NetConnection实例并重新连接到一个老的应用程序或一
个新的应用程序。

注意：NetConnection.close不会对HTTP连接产生影响。

例子

下面的代码关闭了NetConnection实例myNetConn：

```
myNetConn.close();
```

NetConnection.connect()

可用性

Flash Communication Server MX

用法

myNetConnection.connect(URI, [p1, ..., pN])

myNetConnection.connect(URI, [p1, ..., pN])参数

参数URI 一个要连接到的URI。

URI 一个要连接到的URI。

p1, ..., pN 可选的参数，可以是任何ActionScript类型，包括对其他ActionScript对象的引用。这些参数被作为连接参数发送
给application.onConnect事件处理器用于RTMP连接。在AMF连接到应用程序服务器的情况中，任何RTMP参数都被会忽略。

返回

对于RTMP连接，如果成功，返回一个布尔值true，否则，返回false。对于AMF连接到应用程序服务器，则总是返回true。

描述

方法。连接到一个应用程序服务器或另一个Flash Media Server。主机URI具有下列格式：

[protocol://]host[:port]/appName[/instanceName]

例如，下面是合法的例如，下面是合法的URIURI：：

http://appServer.mydomain.com/webApphttp://appServer.mydomain.com/webApp

rtmp://rtserver.mydomain.com/realtimeApp

rtmps://rtserver.mydomain.com/secureApp

你可以使用NetConnection.connect()方法利用标准协议（比如HTTP）连接到一个应用程序服务器进行服务器到服务器的交
互，或者利用Macromedia Real-Time Messaging Protocol（RTMP-实时消息协议）格式或SSL（RTMPS）连接到另一个Flash
Media Server以共享音频、视频和数据。Flash Media Server中的SSL使用一个被称为OpenSSL的第三方开源库。

编写一个application.onStatus回调函数并为RTMP连接检查NetConnection.isConnected属性，以确定一个成功的连接是否已经
做成是一个好的编程习惯。对于 AMF（Action Message Format-动作消息格式）连接，检查NetConnection.onStatus。
做成是一个好的编程习惯。对于AMF（Action Message Format-动作消息格式）连接，检查NetConnection.onStatus。

关于安全连接

Flash Media Server在被指派的安全端口上只接受RTMPS连接。一个端口被标记为安全的是通过在Adaptor.xml文件的
HostPort标签中的端口前面指定一个负号，就像下面的代码所展示的这样：

<HostPort>:1935,80,-443</HostPort>

这个代码指定FMS可以在端口、和上侦听任何接口，在这里，被指派为一个安全端口，它只接受RTMPS

连
接。在端口或上尝试一个RTMPS连接将会失败，因为客户机试图执行一个服务器不能完成的SSL“握手”。同样的，一
个至端口的常规RTMP连接也会失败，因为服务器试图执行一个客户机不能完成的SSL握手。

通过检查服务器端的Client.secure属性，你可以确定一个至服务器的连接是否是经由一个安全通道的，就像下面的范例所展
示的这样：

```
application.onConnect = function(client){
    if (client.secure){
        trace("This client is connected over a secure connection.");
    }
}
```

创建一个调试连接

你可以把一个属性和关键字追加到一个Internet连接请求上以创建一个调试连接。一个调试连接赋予你对一个应用程序更大
的访问权；例如，你可以播放流并查看共享对象。

要创建一个调试连接：

调用NetConnection.connect()并追加一个位的请求数字，就像下面的范例中展示的这样：

nc.connect("rtmp://fmsaddress/appName/instanceName?_fcs_debugreq_=1234

出于安全性的考虑，这个Internet连接被置于一种未决状态。

调用服务器管理API approveDebugSession()并传递给它同样的调试请求数字。

调试连接试图连接到这个应用程序。

如果Application.allowDebug属性是true，则这个连接被批准。

要提供彻底的安全并阻止调试连接，在application.xml文件中把application.allowDebug设置为false。这个设置会覆盖在服
务器端代码中被批准的调试连接。

如果连接被批准，则Services.onDebugConnect会被调用。

假定你可以使用onConnect处理器来拒绝这个连接。

转义连接URI

把在任何用于连接到Flash Media Server的URI中的特殊字符转换（或说转义）成URL编码是非常重要的。如果你没有转义特
殊字符，则共享对象就不能经由一个Internet连接正常工作。当创建一个调试连接时转义URI尤其重要，因为调试属性中有一些
特殊字符必须被追加到URI上。使用下面的代码来转义一个URI：

```
function escapeURI(uri){
    index = uri.indexOf('?');
    if (index == -1){
        return uri;
    }

    prefix = uri.substring(0, index);
    uri = uri.substring(index);

    return prefix += escape(uri);
}
basicString = "rtmp://serverName/appName/instance?_fcsdebug_req=someNumber";
escapedString = escapeURI(BasicString);
nc.connect(EscapedString);
```

例子

下面的范例为NetConnection的myConn实例创建了一个至一个Flash Media Server的RTMP连接：

```
myConn = new NetConnection();
myConn.connect("rtmp://tc.foo.com/myApp/myConn");
```

下面的范例为NetConnection的myConn实例创建了一个至一个应用程序服务器的AMF连接：

```
myConn = new NetConnection();
myConn.connect("http://www.xyz.com/myApp/");
```

下面的范例创建了一个调试连接：

```
nc_admin = new NetConnection();
nc_admin.connect("rtmp://tc.foo.com/myApp/myConn?_fcs_debugreq_=1234");
nc_admin.call("approveDebugSession", null, "myApp/myConn", 1234);
```

NetConnection.isConnected

可用性

Flash Communication Server MX

用法

myNetConnection.isConnected

属性（只读）。一个布尔值，指出一个连接是否已经被做成。如果有一个至服务器的连接，则它被设置被描述 truet 回调函数中检查这个属性的值是一个好主意。对于AMF连接至应用程序服务器，这个属性总是true。 。

例子

这个范例在一个onStatus定义中使用NetConnection.isConnected来检查一个连接是否已经做成：

```
nc = new NetConnection();nc = new NetConnection();

nc.connect("rtmp://tc.foo.com/myApp");
nc.onStatus = function(infoObj){
```

```
    if (info.code == "NetConnection.Connect.Success" && nc.isConnected){
        trace("We are connected");
    }
};
```

NetConnection.onStatus

可用性

Flash Communication Server MX

用法

```
myNetConnection.onStatus = function(infoObject) {
    //这里是你的代码
};
```

参数

infoObject 一个信息对象。

返回

无

描述

事件处理器。每当NetConnection对象的状态发生改变时被调用。例如，如果与服务器的连接在一个RTMP连接中丢失了，
则NetConnection.isConnected属性就会被设置为false，并且NetConnection.onStatus会以一个NetConnection.Connect.closed状态消息
被调用。对于AMF连接，NetConnection.onStatus仅被用于指示一个失败的连接。使用这个事件处理器来进行连接检查。

例子

下面的范例定义了一个函数用于onStatus处理器，它会输出消息以指示NetConnection是否是成功的：

```
nc = new NetConnection();
nc.onStatus = function(info){
    if (info.code == "NetConnection.Connect.Success") {
        _root.gotoAndStop(2);
    }    else {
        if (! nc.isConnected){
            _root.gotoAndStop(1);
        }
    }
};
```

NetConnection.uri

可用性

Flash Communication Server MX

用法

myNetConnection.uri

描述

属性（只读）。一个字符串，指出由NetConnection.connect()方法传递的URI。在一个对NetConnection.connect()的调用之
前，或在NetConnection.close被调用之后，这个属性被设置为null。

protectObject()

可用性

Flash Media Server 2

用法

protectObject(userObj)

参数

userObj 一个要被包装到一个C包装对象中的对象。

返回

一个对象。

描述

方法（全局）；将用户定义的或内建的对象保护在一个C包装对象之中。任何被传递给protectObject()函数的对象的方法都
将变成系统调用，称其为系统调用是因为应用程序代码不能直接访问或检查这些方法-只能经由这个包装进行访问。你只能在
secure.asc文件中使用这个函数。

Flash Media Server 2有两个脚本执行模式：安全和普通。在安全模式中，只有secure.asc文件（假如它存在的话）被装载并
计算-没有其他的应用程序脚本被装载。同样的，全局getGlobal()和protectObject()函数仅在安全模式是可用的。它们是非常强大
的，因为它们提供了对脚本执行环境的完全访问，并可以让你创建系统对象。一旦secure.asc文件被装载，这个服务器就会切换
到普通脚本执行模式，直到这个应用程序被卸载。

在一个对象被保护后，你必须确保它不是直接可访问的；不要在全局变量中引用它，或是使其成为一个可访问对象的一个
成员。由protectObject()调用返回的包装对象把所有的方法调用分派到下层的用户对象，但阻止对成员数据的访问。其结果就
是，你不能直接枚举或修改成员。包装对象保持对下层用户对象的一个明确的引用，这确保了对象是有效的。包装遵循普通的
引用规则，并会在其被引用期间存在。

要获得更多的信息，参看《开发媒体应用程序》中的“实现安全系统对象”。

例子

在secure.asc被执行后，对load()的调用会直接通过用户定义的系统调用，就像下面的范例中展示的这样：

```
var sysobj = {};
sysobj._load = load; // 隐藏load函数
load = null; // 使其成为不可用的未授权代码
sysobj.load = function(fname){
```

```
        // 用户定义的代码来验证/修改fname
        return this._load(fname);
    }
    var global = getGlobal();获取全局对象
    var global = getGlobal();
    // 现在保护我们的sysobj并使其作为'system'全局可用。
    // 同时，设置它的属性，以便它是只读的且不可删除。


    global["system"] = protectObject(sysobj);

    setAttributes( global, "system", false, true, true );

    // 现在出于兼容性的考虑添加一个全局load()函数。
    // 现在出于兼容性的考虑添加一个全局load()函数。
    // 使其成为只读且不可删除的。


    global["load"] = function(path){
        return system.load( path );
    }

    setAttributes( global, "load", false, true, true );
```

setAttributes()

可用性

Flash Media Server 2

用法

setAttributes(obj, propName, enumerable, readonly, permanent)

参数

obj 一个对象。

propName 一个字符串；要上传变量的URL。属性的名字存在于obj参数中。设置不存在属性上的属性不会有效果。

enumerable 下列值之一：true、false，或null。如果为true，则属性可枚举，如果为false，则属性不可枚举；值null保持这个
属性不改变。不可枚举的属性在枚举时（for var i in obj）会被隐藏。

readonly 下列值之一：true、false，或null。如果为true，则属性是只读的，如果为false，则是可写的；值null保持这个属性不
改变。任何试图赋予某个新值的尝试都将会忽略。典型的使用是，在属性是可写的情况下，你为其指派一个值使这个属性成为
只读的。

permanent 下列值之一：true、false，或null。如果为true，则属性是永久的（不可删除的），如果为false，则是可删除的；

值null保持这个属性不改变。任何试图删除一个永久属性的尝试（通过调用delete obj.prop）都将被忽略。

描述

方法（全局）；可以让你避免某些方法和属性被枚举、可写和可删除。

在一个Flash Media Server服务器端脚本中，一个对象中的任何属性总是可枚举、可写和可删除的。你可以调用setAttributes
()来改变一个属性的默认属性或定义常数。

例子

下面的范例避免__resolve方法在枚举中出现：

```
Object.prototype.__resolve = function(methodName){ ... };
setAttributes( Object.prototype, "__resolve", false, null, null );
```

下面的范例在一个Constants对象上创建了三个常数，然后使它们成为永久和只读的：

```
Constants.Kilo = 1000; setAttributes(Constants, "Kilo", null, true, true);
Constants.Mega = 1000*Constants.Kilo;
setAttributes(Constants, "Mega", null, true, true);
Constants.Giga = 1000*Constants.Mega; setAttributes(Constants, "Giga", null, true, true);
```

setInterval()

可用性

Flash Communication Server MX

用法

```
setInterval(function, interval[, p1, ..., pN]);
setInterval(object, methodName, interval[, p1, ..., pN]);
```

参数

function 一个定义过的ActionScript函数名或是对一个匿名函数的引用。

object 一个对象，源自ActionScript Object对象。

interval 调用function间的时间（间隔），以毫秒为单位。

p1, ..., pN 传递给function的可选的参数。

返回

针对这个调用的一个唯一的ID。如果interval没有被设置，则这个方法返回-1。

描述

方法（全局）。以一个特定的时间间隔反复的调用一个函数或方法，直到clearInterval()方法被调用。这个方法允许一个服
务器端脚本来运行一个普通的例行程序。setInterval方法会返回一个唯一的ID，你可以通过把这个唯一的ID传递给clearInterval()
方法来停止这个例行程序。

注意：标准的JavaScript支持setInterval方法的一种额外的用法，setInterval(stringToEvaluate, timeInterval)，但这种用法不被服
务器端通讯ActionScript所支持。

例子

下面的范例使用一个匿名函数来每隔一秒钟把消息“interval called”发送到服务器日志：

```
setInterval(function(){trace("interval called");}, 1000);
```

下面的范例也使用一个匿名函数来每隔一秒钟把消息“interval called”发送到服务器日志，但它是把消息作为一个参数传递

给这个函数的：

```
setInterval(function(s){trace(s);}, 1000, "interval called");
```

下面的范例使用一个命名的函数，callback1，来把消息“interval called”发送到服务器日志：

```
function callback1(){trace("interval called");
}
setInterval(callback1, 1000);
```

下面的例子也使用一个命名的函数，callback2，把消息“interval called”发送到服务器日志，但它是把消息作为一个参数传
递给函数的：

```
function callback2(s){
    trace(s);
}
setInterval(callback2, 1000, "interval called");
```

SharedObject类

可用性

Flash Communication Server MX

共享对象可以让你在服务器上存储数据，并可以让你实时的在多个客户机应用程序间共享数据。共享对象可以是临时的，
或者它们也可以在一个应用程序被关闭后永久性的维持在服务器上；你可以把这些共享对象看作是实时的数据传送设备。

注意：本条目解释服务器端SharedObject类。你也可以使用客户机端SharedObject类来在客户机上创建对象。

下面的列表描述了在服务器端ActionScript中使用共享对象的通常的方法：

在一个服务器上存储和共享数据

一个共享对象，它可以在服务器上存储数据，供其它客户机来检索。例如，你可以打开一个远端共享对象，比如一个

电话列表，这个电话列表永久性的存在于服务器上。任何时候当一个客户机对这个共享作了一个改变，则这一修订过的数

据对于所有当前连接到这个对象的客户机或是以后连接到这个对象的客户机而言都是可用的。如果这个对象也永久于本

地，而一个客户机在没有连接到服务器的情况下改变了这个对象中的数据，则下次当这个客户机连接到这个对象时，这些

改变会被拷贝到远端共享对象。

实时共享数据

用户进入或离开聊天室时，这个对象被更新，并且连接到这个对象的所有客户机都会看到更新过的聊天室用户列表。

要在服务器端ActionScript中利用共享对象，理解下面的信息是非常重要的：

服务器端ActionScript方法SharedObject.get()创建远端共享对象（RSO）；没有能创建本地共享对象（LSO）的服务器端

方法。

RSO文件的扩展名是.fso；RSO被存储在服务器上创建它们的那个应用程序的一个子目录中。

方法。

RSO文件的扩展名是.fso；RSO被存储在服务器上创建它们的那个应用程序的一个子目录中。

服务器端共享对象可以是非永久性的（在一个应用程序实例的持续期中存在）或永久性的（在一个应用程序关闭后仍

存储于服务器上）。

要创建一个永久性共享对象，把SharedObject.get()方法的persistence参数设置为true。永久性的共享对象可以让你维持

一个应用程序的状态。

每一个RSO由一个唯一的名字来标识，并包含一个名字-值对的列表，被称为属性，就像任何其他ActionScript对象那

样。名字必须是一个具唯一性的字符串，而值可以是任何ActionScript数据类型。

注意：不同于客户机端共享对象，服务器端共享对象没有数据属性。

要得到一个服务器端共享对象属性的值，调用SharedObject.getProperty()。要设置一个服务器端共享对象属性的值，调

用SharedObject.setProperty()。

要清除一个共享对象，调用ShareObject.clear()方法；要删除多个共享对象，调用Application.clearSharedObjects()方法。

服务器端共享对象可以由当前应用程序实例拥有或由另一个应用程序实例拥有。这另一个应用程序实例可以位于同一

台服务器上或是位于一个不同的服务器上。引用由另一个应用程序实例所拥有的共享对象被称为代理共享对象。

如果你写了一个服务器端脚本，该脚本会修改多个属性，则你可以在更新对象前通过调用SharedObject.lock()方法在更新期

间阻止其他客户机修改这个对象。然后，你可以调用SharedObject.unlock()来提交改变并允许其他改变可以被执行。在lock()和

unlock()方法中调用SharedObject.mark()来以分组方式呈递change事件。

当你获得一个对某个代理共享对象的引用时，对这个对象所作的任何改变都会被发送到拥有这个对象的实例。任何改变的

成功或失败都会利用SharedObject.onSync事件处理器发送，假如它被定义了的话。

SharedObject.lock()和SharedObject.unlock()方法不能锁定或解锁代理共享对象。

注意：要获得有关共享对象的更多信息，参看《开发媒体应用程序》中的“关于流和共享对象”、“共享对象流”，以及

“共享对象文件”。

SharedObject类的方法汇总

方法	描述
SharedObject.clear()	删除一个永久性共享对象的所有属性。
SharedObject.close()	取消对一个共享对象的订阅。
SharedObject.commit()	静态的；存储一个特定的永久性共享对象实例或是存储所有被标记为dirty的共享对象实

例。

SharedObject.flush() 保存一个永久性共享对象的当前状态。

SharedObject.get() 静态的；创建一个共享对象或是返回对一个现有的共享对象的引用。

SharedObject.lock()SharedObject.getProperty() 获得一个共享对象属性的值。锁定共享对象实例。阻止任何来自客户机的对这个对象的改变，直到

SharedObject.unlock()方法被调用。

SharedObject.mark() 把所有的改变事件作为一个单一消息呈递给一个订阅客户机。

SharedObject.purge() 导致服务器移除所有比指定的版本更旧的被删除属性。

SharedObject.send() 向订阅了这个共享对象的客户机发送一条消息。

SharedObject.purge()SharedObject.setProperty() 导致服务器移除所有比指定的版本更旧的被删除属性。为一个共享对象属性设置一个新值。

SharedObject.send() 向订阅了这个共享对象的客户机发送一条消息。

SharedObject.setProperty() 为一个共享对象属性设置一个新值。

SharedObject.size() 返回一个共享对象中的有效属性的数量。

SharedObject.unlock() 解锁一个用SharedObject.lock()锁定的共享对象实例。

SharedObject类的属性汇总

属性	描述
SharedObject.autoCommit	一个布尔值，指出服务器是（true）否（false）周期性的提交所有永久性共享对象进行存储。
SharedObject.isDirty	一个布尔值，指出这个永久性SharedObject自上一次它被存储后是（true）否（false）已经被修改。
SharedObject.name	共享对象的名字。
SharedObject.resyncDepth	这个深度指出了什么时候一个共享对象的被删除的值应该被永久性的删除。
SharedObject.version	一个共享对象的当前版本号。

SharedObject类的事件处理器汇总

事件	描述
SharedObject.handlerName	一个属性名的占位符，它指定了一个函数对象，当一个共享对象接收到一个广播消

息，且这个广播消息的方法名匹配这个属性名时，这个函数会被调用。

SharedObject.onStatus	为一个共享对象报告错误、警告，以及状态消息。
SharedObject.onSync	当一个共享对象改变时被调用。

SharedObject.autoCommit

可用性

Flash Media Server 2

用法

mySO.autoCommit

描述

属性；一个布尔值，指出服务器是（true）否（false）会周期性的存储所有永久性共享对象。如果autoCommit为false，则这
个应用程序必须调用SharedObject.commit()来保存共享对象，否则数据就会丢失。

默认情况下，这个属性是true。通过在Application.xml文件中使用下面的配置键来指定初始状态，这个值可以被覆盖，就像
下面的范例中展示的这样：

```
<SharedObjManager>
    <AutoCommit>false</AutoCommit>
</SharedObjManager>
```

SharedObject.clear()

可用性

Flash Communication Server MX

用法

mySO.clear()

参数

无

返回

如果成功，返回true；否则，返回false。

描述

方法。删除所有的属性并发送一个“clear”事件至订阅一个永久性共享对象的所有客户机。这个永久性的data对象也会被
从永久性的共享对象中删除。

例子

var myShared = SharedObject.get("foo", true);下面的范例在共享对象 上调用 方法：

var len = myShared.clear();myShared clear

SharedObject.close()

可用性

可用性Flash Communication Server MX
Flash Communication Server MX
用法
SharedObject.close()
参数
无
返回
无
描述
方法。分离对一个共享对象的引用。一个对SharedObject.get方法的调用会返回对一个共享对象实例的引用。这个引用会保
持有效，直至存储这个引用的变量不再被使用，这个脚本会被进行垃圾回收。要立即销毁一个引用，你可以调用
SharedObject.close。当你不再需要代理一个共享对象时，你可以使用SharedObject.close。
例子
在这个范例中，mySO被捆绑了一个对共享对象foo的引用。当你调用mySO.close时，你就把mySO这个引用从共享对象foo
上分离了。

```
mySO = SharedObject.get("foo");
  //在这里插入代码
mySO.close();
```

SharedObject.commit()

可用性
Flash Media Server 2
用法
SharedObject.commit([name])
参数
name 一个字符串，指出要存储的永久性共享对象的名字。如果没有名字被指定，或是传递了一个空字符串，则所有的永久
性共享对象都会被存储。这个参数是可选的。
返回
一个布尔值，指出成功（true）或失败（false）。
描述
方法（静态的）；存储一个特定的永久性共享对象实例或是存储所有的其isDirty属性的值为true的永久性共享对象实例。如
果SharedObject.autoCommit属性为false，且当一个共享对象被本地存储时你需要对其进行管理时，使用这个

方法。

例子

下面的代码中当应用程序停止时，提交所有改变的共享对象进行本地存储：

```
application.onAppStop = function (info){
    // 在这里插入代码。
    SharedObject.commit();
}
```

SharedObject.flush()

可用性

Flash Communication Server MX

用法

mySO.flush()

参数

无

返回

一个布尔值。如果成功，返回true；否则，返回false。

描述

方法。保存一个永久性共享对象的当前状态。

例子

下面的范例把对共享对象foo的引用放置在了变量myShared中。然后，这段代码锁定了这个共享对象，以便没有人能够对这
个共享对象做任何改变，之后，通过调用myShared.flush保存这个共享对象。在这个共享对象被保存后，它被解锁，以便可以接
受未来对它的改变。

```
var myShared = SharedObject.get("foo", true);
myShared.lock();
// 在这里插入的代码在共享对象上操作。
myShared.flush();
myShared.unlock();
```

SharedObject.get()

可用性

Flash Communication Server MX

用法

SharedObject.get(name, persistence [, netConnection])

参数

name 要返回的共享对象实例的名字。

persistence 一个布尔值。true用于永久性共享对象；false用于非永久性共享对象。如果没有值被指定，则默认值是false。

netConnection 一个NetConnection对象，它描述了至一个应用程序实例的连接。你可以传递这个参数以得到对位于另一个服
务器上的一个共享对象或由另一个应用程序实例持有的一个共享对象的引用。由name参数指定的共享对象的所有的更新通告都
是被代理到这个实例的，并且，当一个永久性共享对象改变时，远端实例会通告本地实例。当你调用SharedObject.get时，被用
作netConnection参数的NetConnection对象不需要是连接的。当NetConnection状态变为连接时，服务器会连接到远端共享对象。
这个参数是可选的。

返回

对一个SharedObject类的实例的引用。

描述

方法（静态的）；创建一个共享对象或返回对一个现有共享对象的引用。要在一个共享对象上执行任何操作，服务器端脚

有两种类型的共享对象-永久性的和非永久性的，它们有独立的命名空间。这意味着一个永久性的和非永久性的共享对象可
以有相同的名字，并作为两个截然不同的共享对象而存在。共享对象作用于应用程序实例的命名空间中，并由一个字符串来识
别。共享对象名应该符合URI规范。

你也可以调用SharedObject.get来得到对一个位于另一个应用程序实例的命名空间中的共享对象的引用，这里所指的另一个
实例可以位于同一台服务器上或是位于一台不同的服务器上，这种情况被称为代理共享对象。要得到一个对来自另一个实例的
共享对象的引用，创建一个NetConnection对象并使用NetConnection.connect()方法来连接到拥有那个共享对象的应用程序实例。
实例可以位于同一台服务器上或是位于一台不同的服务器上，这种情况被称为代理共享对象。要得到一个对来自另一个实例的
共享对象的引用，创建一个NetConnection对象并使用NetConnection.connect()方法来连接到拥有那个共享对象的应用程序实例。
把NetConnetion对象作为SharedObject.get方法的netConnection参数传递。在任何客户机请求这个代理共享对象前，服务器端脚本
必须得到对这个代理共享对象的引用。要实现这点，在application.onAppStart处理器中调用SharedObject.get。

如果你使用一个netConnection参数来调用SharedObject.get，并且本地的应用程序实例已经有一个同名的共享对象，则这个
共享对象会被转换成一个代理共享对象。针对连接到一个代理共享对象的客户机的所有共享对象消息都会被发送到雇主实例。

当被用作netConnection参数的NetConnection对象的连接状态从连接改为断开连接时，代理共享对象会被设置为空闲，并且
接收自订阅者的任何消息都将被丢弃。当一个连接丢失时，NetConnection.onStatus处理器会被调用。之后，你可以重建一个至
远端实例的连接，并调用SharedObject.get，这会把代理共享对象的状态从空闲改为连接。

在一个代理共享对象已经被连接的情况下，如果你用一个新的NetConnection对象来调用SharedObject.get，则如果这个新的
NetConnection对象的URI不匹配当前的NetConnection对象的话，则这个代理共享对象就会取消自先前共享对象的订阅，并发送
一个“clear”事件至所有的订阅者，然后订阅至新的共享对象实例。当一个对代理共享对象的订阅是成功

的时，所有的订阅者
都会被重新初始化到新的状态。这个过程可以让你把一个共享对象从一个应用程序实例移植到另一个应用程序实例而无需断开
客户机的连接。

由代理共享对象接收自订阅者的更新会被基于这个代理共享对象版本的当前状态和订阅者的版本进行检查，以确定这个更
新是否可以被拒绝。如果这个改变可以被拒绝，那么，这个代理共享对象就不会把消息转寄到远端实例；拒绝消息会被发送给
订阅者。

相应的客户机端ActionScript方法是SharedObject.getRemote。

例子

这个范例在函数onProcessCmd中创建了一个名为foo的共享对象。这个函数被传递以一个参数-cmd，这个参数被指派给共
享对象中的一个属性。

```
function onProcessCmd(cmd){
  //在这里插入代码
  var shObj = SharedObject.get("foo", true);
  propName = cmd.name;
  shObj.getProperty (propName, cmd.newAddress);
}
```

下面的范例使用一个代理共享对象。一个代理共享对象驻留在一个服务器上或位于一个应用程序实例（被称为雇主）中，
这种雇主应用程序实例与客户机连接到的服务器或应用程序实例（被称为代理）有所不同。当客户机连接到代理并得到一个远
端共享对象时，代理会连接到雇主并赋予客户机一个对这个共享对象的引用。下面的代码是在main.asc文件中的：

```
application.appStart = function() {
      nc = new NetConnection();
      nc.connect("rtmp://" + master_server + "/" + master_instance);
      proxySO = SharedObject.get("myProxy",true,nc);
    //现在，任何时候当客户机请求一个被称为myProxy的共享对象时，
    //它们就会接收到来自master_server/master_instance的myProxy共享对象。
};
```

SharedObject.getProperty()

可用性

Flash Communication Server MX

用法

mySO.getProperty(name)

参数

name 共享对象中的属性名。

返回

一个SharedObject属性的值。

描述

方法。取回一个共享对象中的一个命名属性的值。返回值是与这个属性关联的拷贝，对返回值所作的任何改变不会更新这
个共享对象。要更新一个属性，使用SharedObject.setProperty()方法。

例子

下面的范例得到name属性的值，并把它传递给变量value：

value = sharedInfo.getProperty(name);

SharedObject.getPropertyNames()

可用性

Flash Communication Server MX

用法

mySO.getPropertyNames()

参数

无

返回

一个数组，包含了一个共享对象的所有的属性名。

描述

方法。列举一个给定的共享对象的所有的属性。

例子

这个范例在myInfo共享对象上调用getPropertyNames，并把名字放置到names变量中。然后，它在一个for循环中列举这些属
性。

```
myInfo = SharedObject.get("foo");
var addr = myInfo.getProperty("address");
myInfo.setProperty("city", San Francisco");
var names = myInfo.getPropertyNames();
for (x in names){
   var propVal = myInfo.getProperty(names[x]);
   trace("Value of property " + names[x] + " = " + propVal);
}
```

可用性

Flash Communication Server MX

用法

mySO.onBroadcastMsg = function([p1,..., pN]){}

用法

mySO.onBroadcastMsg = function([p1,..., pN]){}

参数

onBroadcastMsg 一个属性名。

p1, ..., pN 可选的参数，如果消息包含用户定义的参数的话，则这些参数会被传递给处理器方法。这些参数是用户定义的
JavaScript对象，它被传递给SharedObject.send方法。

返回

任何返回值都被服务器忽略。

样名字的广播消息时，这个处理器会被调用。你必须定义一个函数对象并把它指派到这个事件处理器。一个共享对象可以从客
户机端SharedObject.send()方法接收一个广播消息。

用在这个函数体中的关键字this被设置为由SharedObject.get返回的共享对象实例。

如果你不希望服务器接收一个特定的广播消息的话，就不要定义这个处理器。

例子

下面的例子定义了一个名为broadcastMsg的函数：

例子

下面的例子定义了一个名为broadcastMsg的函数：

```
var mySO = SharedObject.get("userList", false);
mySO.broadcastMsg = function(msg1, msg2){
    trace(msg1 + " : " + msg2);
};
```

SharedObject.isDirty

可用性

Flash Media Server 2

用法

```
mySO.isDirty
```

描述

属性（只读）；一个布尔值，指出这个永久性SharedObject自上一次它被存储后是（true）否（false）已经被修改。

SharedObject.commit()方法会存储其isDirty属性为true的共享对象。

对于非永久性共享对象，这个属性总是false。

例子

下面的范例会保存这个共享对象，假如它已经被改变了的话：

```
var so = SharedObject.get("foo", true);
if (so.isDirty){
    SharedObject.commit(so.name);
}
```

SharedObject.lock()

可用性

Flash Communication Server MX

用法

mySO.lock()

参数

无

返回

一个整数，指出锁定计数：或更大的数字表示成功；-1表示失败。对于代理共享对象，则总是返回-1。

描述

方法。锁定共享对象实例。这个方法赋予服务器端脚本对共享对象的排它的访问权。当SharedObject.unlock()方法被调用
时，所有的改变都会被批处理，并且，一个更新消息会经由SharedObject.onSync处理器被发送到订阅这个共享对象的所有的客
户机。如果你嵌套了SharedObject.lock()和SharedObject.unlock()方法，则要确保对每一个lock都有一个unlock；否则，客户机就
会被封锁了对共享对象的访问。

你不能在代理共享对象上使用SharedObject.lock方法。

例子

这个范例锁定了myShared共享对象，执行那些将被插入的代码，然后解锁这个对象：

```
var myShared = SharedObject.get("foo");
myShared.lock();
//在这里插入那些操作这个共享对象的代码
myShared.unlock();
```

SharedObject.mark()

可用性

Flash Media Server 2

用法

mySO.mark(handlerName, p1, ..., pN)

参数

handlerName 在客户机端SharedObject实例上调用指定的函数属性。例如，如果handlerName参数是onChange，则客户机会使
用所有p1……pN参数来调用SharedObject.onChange()方法。

注意：不要使用内建的方法名作为处理器名。例如，如果处理器名是close的话，则订阅的流将被关闭。

p1……pN ActionScript类型的参数，包含对其他ActionScript对象的引用。当其在Flash客户机上被执行时，这些参数会被传
递给指定的处理器。

返回

一个布尔值。如果消息被分派到客户机，则返回true；否则，返回false。

描述

方法；把所有的change事件作为一个单一消息呈递给一个订阅客户机。

在一个服务器端脚本中，你可以在对lock()和unlock()方法的调用间调用setProperty()方法来更新多个共享对象属性。所有的
订阅客户机都会经由SharedObject.onSync处理器接收一个change事件通知。但不管怎么说，服务器可能会压缩多个消息以优化
带宽的使用，因此，change事件通知可能不会以它们在代码中的顺序被发送。

当你想要在一个设置中的所有属性都被更新后执行代码时使用mark()方法。

如果一个应用程序调用lock()方法来锁定一个共享对象，调用setProperty()方法来做一系列属性改变，并在调用unlock()方法
前调用mark()方法，则Flash Media Server会保证所有的change事件被作为一个单一消息呈递到订阅客户机。你可以调用传递给
mark()方法的handlerName参数来在mark()调用被更新前了解所有的属性改变。

例子

下面的范例调用mark()方法两次来为客户机集合两套共享对象属性的更新：

```
var myShared = SharedObject.get("foo", true);

myShared.lock();
myShared.setProperty("name", "Stephen");
myShared.setProperty("address", "Xyz lane");
myShared.setProperty("city", "SF");
myShared.mark("onAdrChange", "name");
myShared.setProperty("account", 12345);
myShared.mark("onActChange");
myShared.unlock();
```

下面的范例展示了进行接收的客户机端脚本：

```
connection = new NetConnection();
connection.connect("rtmp://flashmediaserver/someApp");
var x = SharedObject.get( "foo", connection.uri, true);
x.connect(connection);
x.onAdrChange = function(str) {
    // 共享对象已经被更新，
    // 现在可以查看"name"、"address"和"city"了。
}


x.onActChange = function(str) {
    //共享对象已经被更新，
    //      // 现在可以查看现在可以查看"account""account"属性了。属性了。
}
```

SharedObject.name

可用性

Flash Communication Server MX

用法

mySO.name

描述

属性（只读）。一个共享对象的名字。

SharedObject.onStatus

可用性
Flash Communication Server MX
用法
mySO.onStatus = function(info) {}
参数
info 一个信息对象。
返回
无
描述
事件处理器。报告与一个共享对象的本地实例或一个永久性共享对象相关联的错误、警告，以及状态消息。
例子
下面的范例为共享对象soInstance定义了一个onStatus事件处理器：

```
soInstance = SharedObject.get("foo", true);
soInstance.onStatus = function(infoObj){
   //处理SO状态消息
};
```

SharedObject.onSync

可用性
Flash Communication Server MX
用法
mySO.onSync = function(list){}
参数
list 一个对象数组，包含了一个共享对象的自上次onSync方法被调用后已经改变的属性的相关信息。针对代理共享对象的通
告与针对由本地应用程序实例拥有的共享对象的通告是不同的。下面的表格列出了对每一种共享对象的描述。

本地共享对象

代码	含义

change　　一个属性被一个订阅者改变。
delete　　一个属性被一个订阅者删除。
name　　一个被改变或被删除的属性的名字。
oldValue　　一个属性的旧版本。这对于change和delete消息而言都是true；在客户机上，oldValue不会为delete设置。

注意：在服务器端利用SharedObject.setProperty()方法改变或删除一个属性总是会成功的，因为没有这些改变的通告。

代码	含义
success	共享对象的一个服务器改变被接受。
reject	共享对象的一个服务器改变被拒绝。位于远端实例上的值没有被改变。
successchange	一个属性被另一个订阅者改变。共享对象的一个服务器改变被接受。
reject	共享对象的一个服务器改变被拒绝。位于远端实例上的值没有被改变。
change	一个属性被另一个订阅者改变。
delete	一个属性被删除。当一个服务器删除了一个共享对象，或是另一个订阅者删除了一个属性时，这个通告会出现。
clear	一个共享对象的所有的属性都被删除了。这可以在当服务器的共享对象脱离了与雇主共享对象的同步，或是当永久性共享对象从一个实例移植到另一个实例时发生。典型的情况是，这个事件会跟随一个change消息来恢复所有的服务器共享对象的属性。
name	一个已经被改变或被删除的属性的名字。
oldValue	属性的旧值。这仅对reject、change和delete代码是可用的。

注意：当一个共享对象已经被成功的与服务器同步时，SharedObject.onSync方法会被调用。假如共享对象中没有改变的话，则这个列表对象可能是空的。

返回

无

描述

事件处理器。当一个共享对象改变时调用。使用onSync处理器来定义一个函数，这个函数用于处理由订阅者对共享对象所做的改变。

对于代理共享对象，定义这个函数来获取由服务器和其他订阅者所做改变的状态。

注意：你不能在服务器端ActionScript的SharedObject类的prototype属性上定义onSync处理器。

例子

下面的例子创建了一个函数，任何时候当共享对象so的一个属性改变时，这个函数就会被调用：

```
//创建一个新的NetConnection对象
nc = new NetConnection();
nc.connect("rtmp://server1.xyx.com/myApp");
//创建这个共享对象
so = SharedObject.get("MasterUserList", true, nc);
```

```
//这个列表参数是一个对象数组，

        case "success":// onSync()被调用后未成功改变的属性的成功信息。
so.onSync = function(list) {            trace ("success");
            break;
        case "change":
            trace ("change");
            break;
        case "reject":
            trace ("reject");
            break;
        case "delete":
            trace ("delete");

            break;              break;

        case "delete":          case "clear":

            trace ("delete");
            break;
        case "clear":
            trace ("clear");
            break;
      }
   }
};
```

SharedObject.purge()

可用性

Flash Communication Server MX

用法

mySO.purge(version)

参数

version 一个版本号。所有比这个版本更旧的被删除数据都会被删除。

返回

无

描述

方法。导致服务器清除所有比指定版本更旧的被删除属性。尽管你也可以通过设置SharedObject.resyncDepth属性来完成这
个任务，但SharedObject.purge()方法可以赋予脚本对要删除哪个属性以更多的控制。

例子

这个范例删除了myShared共享对象的比SharedObject.version - 3更旧的所有属性。

```
var myShared = SharedObject.get("foo", true);
```

```
mySharedlock();
mySharedpurge(mySharedversion - 3);
mySharedunlock();
```

SharedObject.resyncDepth

可用性

Flash Communication Server MX

用法

mySO.resyncDepth

描述

属性。一个整数，指出什么时候一个共享对象的被删除的属性应该被永久性的删除。你可以在一个服务器端脚本中使用这
个属性来重新同步共享对象及控制什么时候共享对象被删除。默认值是无限。
的值时，客户机共享对象的所有当前元素就都会被删除，合法的属性会被发送到客户机，并且，客户机会接收到一个“如果共享对象当前的修订号减去被删除属性的修订号比SharedObject.resyncDepth的值更大，则属性会被删除。同样，如果clear”
消息。

当你要添加和删除许多属性，但你并不想发送太多的消息至Flash客户机时，这个方法就显得尤其有用。假设一个客户机连
接到一个有个属性的共享对象，然后断开了连接，在这个客户机断开连接之后，连接到这个共享对象的其他的客户机删除了
个属性并添加了属性**（译者特别说明：这是根据原文档翻译的结果。但根据上下文看，这里所写的数字明显有错
误。）**，当这个客户机再次连接时，它可以，比如说，接收一个先前存在的个属性的删除消息，以及一个关于两个属性的改
变消息。你可以使用SharedObject.resyncDepth来发送一个“clear”消息，后跟一个两个属性的改变消息，这就可以使客户机免
遭接收到个删除消息的待遇。

例子

遭接收到个删除消息的待遇。 mySO，如果修订版本号的差异大于的话：

例子

下面的范例会重新同步共享对象mySO，如果修订版本号的差异大于的话：

```
mySo = SharedObject.get("foo");
mySo.resyncDepth = 10;
```

SharedObject.send()

可用性

Flash Communication Server MX

用法

methodName 一个客户机共享对象实例的一个方法名。例如，如果你指定doSomething，则这个客户机

必须用所有的p1, ...,
pN参数来调用SharedObject.doSomething方法。

p1, ..., pN 任何ActionScript类型的参数，包括对其他ActionScript对象的引用。当methodName方法在Flash客户机上执行时，
这些参数被传递给这个指定的methodName方法。

返回

一个布尔值。如果消息被发送到客户机，则返回true；否则，返回false。

描述

方法。在一个客户机端脚本中执行一个方法。你可以使用SharedObject.send()来在订阅一个共享对象的所有的Flash客户机上
异步的执行一个方法。服务器不会接收任何来自客户机的关于对这个消息所做响应的成功、失败或返回值的通告。

方法。在一个客户机端脚本中执行一个方法。你可以使用SharedObject.send()来在订阅一个共享对象的所有的Flash客户机上
异步的执行一个方法。服务器不会接收任何来自客户机的关于对这个消息所做响应的成功、失败或返回值的通告。

例子

这个范例调用SharedObject.send方法来在客户机端ActionScript中执行doSomething方法，并传递给doSomething以字符串
“this is a test”。

```
var myShared = SharedObject.get("foo", true);
myShared.send("doSomething", "this is a test");
```

下面的范例是客户机端ActionScript代码，它定义了doSomething方法：

```
connection = new NetConnection();
connection.connect("rtmp://www.macromedia.com/someApp");
var x = SharedObject.getRemote("foo", connection.uri, true);
x.connect(connection);
x.doSomething = function(str) {
//处理这个字符串
};
```

SharedObject.setProperty()

可用性

Flash Communication Server MX

用法

```
mySO.setProperty(name, value)
```

参数

name 共享对象中的属性名。

value 与属性关联的一个ActionScript对象，或是null以删除这个属性。

返回

无

描述

方法。更新一个共享对象中一个属性的值。

服务器端的name参数与客户机端的data属性（property）中的一个属性（attribute）是一样的。例如，下面的两个代码行是等
价的；第一行是服务器端ActionScript，第二行是客户机端ActionScript：

```
mySO.setProperty(nameVal, "foo");
clientSO.data[nameVal] = "foo";
```

()。如果你没有首先调用SharedObject.lock()，而直接调用了SharedObject.setProperty()的话，则改变会在共享对象上发生，并
且，在SharedObject.setProperty()返回之前，所有的对象订阅者都会被通告。如果在调用
SharedObject.setProperty()之前先调用了
SharedObject.lock()，则当SharedObject.unlock()方法被调用时，所有的改变会被批处理并发送。当共享对象的本地拷贝被更新
时，客户机端的SharedObject.onSync处理器会被调用。

注意：如果在一个服务器端脚本中只有一个源（客户机或者服务器）正在更新一个共享对象，则你不需要使用lock()或
unlock()方法或onSync处理器。

注意：如果在一个服务器端脚本中只有一个源（客户机或者服务器）正在更新一个共享对象，则你不需要使用lock()或
unlock()方法或onSync处理器。

例子

这个范例使用SharedObject.setProperty方法用值San Francisco来创建属性city。然后，它在一个for循环中列举所有的属性值
并用一个trace动作把结果打印到输出窗口。

```
myInfo = SharedObject.get("foo");
var addr = myInfo.getProperty("address");
myInfo.setProperty("city", "San Francisco");
var names = sharedInfo.getPropertyNames();
for (x in names){
   var propVal = sharedInfo.getProperty(names[x]);
   trace("Value of property " + names[x] + " = " + propVal);
}
```

SharedObject.size()

可用性

Flash Communication Server MX

用法

mySO.size()

参数

无

返回

一个整数，指出属性的数量。

描述

方法。返回一个共享对象中合法属性的总数。

例子

下面的范例得到一个共享对象中属性的数量，并把这个值赋给了变量len：

```
var myShared = SharedObjecd.get(application, "foo", true);
var len = myShared.size();
```

SharedObject.unlock()

可用性

Flash Communication Server MX

用法

```
mySO.unlock()
```

参数

无

返回

一个整数，指出锁定计数：如果成功，则返回或更大的值；否则，返回-1。对于代理共享对象，这个方法总是返回-1。

描述

方法。解锁一个共享对象实例。这会导致脚本释放对共享对象排它的访问权，并使其他的客户机可以更新这个实例。这个
方法也导致服务器提交在SharedObject.lock()方法被调用之后所有的改变，并向所有的客户机发送一个更新消息。

你不能使用在代理共享对象上使用SharedObject.unlock方法。

例子

这个范例展示了解锁一个共享对象是多么的容易：

```
var myShared = SharedObject.get("foo", true);
myShared.lock();
//插入操作共享对象的代码
myShared.unlock();
```

SharedObject.version

可用性

Flash Communication Server MX

用法

```
mySO.version
```

描述

属性(只读)。共享对象的当前版本号。由Flash Player客户机或服务器端脚本使用SharedObject.setProperty方法对共享对象
进行改变会增加version属性的值。

SOAPCall类

可用性

Flash Media Server 2

SOAPCall类是所有Web服务调用所返回的对象类型。最典型的情况就是，当一个WSDL被解析且一个存根被生成时，这些
对象会被自动的构造。

SOAPCall类的属性汇总

属性	描述
SOAPCall.request	一个XML对象，描述当前的SOAP请求。
SOAPCall.response	一个XML对象，描述最新的SOAP响应。

SOAPCall类的事件处理器

事件处理器	描述
SOAPCall.onFault	当一个方法失败且返回一个错误时调用。
SOAPCall.onResult	当一个方法成功调用并返回时调用。

SOAPCall.onFault

可用性

Flash Media Server 2

用法

mySOAPCall.onFault(fault)

参数

fault 这个fault参数是一个XML SOAP Fault的对象版本。

返回

无。

描述

事件处理器；当一个方法失败且返回一个错误时调用。

SOAPCall.onResult

可用性

Flash Media Server 2

用法

mySOAPCall.onResult(result){}

参数

result 由这个操作（如果有任何操作的话）返回的被解码的ActionScript对象。要得到返回的原始XML而非解码后的结果，
访问SOAPCall.response属性。

返回

无。

描述

事件处理器；当一个方法成功并返回时调用。

SOAPCall.request

可用性

Flash Media Server 2

用法

mySOAPCall.request

描述

属性；一个XML对象，描述当前的SOAP请求。

SOAPCall.response

可用性

Flash Media Server 2

用法

mySOAPCall.response

描述

属性；一个XML对象，描述最新的SOAP响应。

SOAPFault类

可用性

Flash Media Server 2

SOAPFault类是返回至WebService.onFault和SOAPCall.onFault函数的错误对象的对象类型。这个对象是作为一个失败的结果
返回的，且是一个SOAP Fault XML类型的ActionScript映射。

SOAPFault类的属性汇总

属性	描述
属性	描述
SOAPFault.detail	一个字符串，指出与错误关联的与特定应用程序相关的信息，比如一个堆栈跟踪或 由Web服务引擎返回的其他信息。
SOAPFault.faultactor	一个字符串，指出故障的源。
SOAPFault.faultcode	一个字符串，指出描述这个错误的简短的标准资格名。
SOAPFault.faultstring	一个字符串，指出错误的供人阅读的描述。

SOAPFault.detail

可用性

Flash Media Server 2

用法

mySOAPFault.detail

描述

属性；一个字符串，指出与错误关联的与特定应用程序相关的信息，比如一个堆栈跟踪或由Web服务引擎返回的其他信
息。

SOAPFault.faultactor

可用性

Flash Media Server 2

用法

mySOAPFault.faultactor
描述
属性；一个字符串，指出故障的源。如果不涉及中间件的话，则这个属性是可选的。

属性；一个字符串，指出故障的源。如果不涉及中间件的话，则这个属性是可选的。

SOAPFault.faultcode

可用性
Flash Media Server 2

描述用法

属性；一个字符串，指出描述这个错误的简短的标准资格名。

SOAPFault.faultstring

SOAPFault.faultstring

可用性
Flash Media Server 2
用法
mySOAPFault.faultstring
描述
属性；一个字符串，指出错误的供人阅读的描述。
例子
下面的范例在Web Service Definition Language（WSDL-Web服务定义语言）装载失败时在一个文本域中展示故障代码：

```
// 准备WSDL位置：
var wsdlURI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";

// 利用WSDL位置例示Web服务对象：
stockService = new WebService(wsdlURI);

// 处理WSDL解析和Web服务例示事件。
stockService.onLoad = function(wsdl){
    wsdlField.text = wsdl;
}
```

```
// 如果这个WSDL装载失败，onFault事件就会被激发：
stockService.onFault = function(fault){
    wsdlField.text = fault.faultstring;
}
```

Stream类

可用性

Flash Communication Server MX

Stream类可以让你处理一个Flash Media Server应用程序中的每一个流。一个流（stream）是一个Flash Player和Flash Media Server或两个服务器之间的一路（one-way）连接。当NetStream.play或NetStream.publish在一个客户机端脚本中被调用时，Flash Media Server会自动创建一个具有唯一名字的Stream对象。你也可以通过调用Stream.get()在服务器端ActionScript中创建一个流。一个用户可以同时访问多个流，并且，同一时刻可以有几百或几千个Stream对象活动。

你可以使用Stream类的属性和方法来在一个播放列表中移动流、从其他的服务器拉来流，以及播放和记录流。

你也可以使用Stream类来经由一个流来播放MP3文件，以及与MP3文件关联的ID3标签。

你可以为Stream类的一个特定的实例创建其他的具有任何合法的ActionScript类型的Stream属性，包括对其他ActionScript对象的引用。这些属性是可用的，直到这个流被从应用程序中删除。

Stream类的属性汇总

属性（只读）	描述
Stream.bufferTime	指出一个流在被播放前要用多长时间来缓冲消息。
Stream.name	一个活动流的唯一的名字。
Stream.syncWrite	一个布尔值，用来控制当流被记录时什么时候流把缓冲区中的内容写入一个FLV文件。

Stream类的方法汇总

方法	描述
Stream.clear()	删除之前由服务器记录的一个流。
Stream.flush()	清出一个流。
Stream.get()	静态的；返回对一个Stream对象的引用。
Stream.length()	静态的；返回一个记录的流的以秒为单位的长度。
Stream.play()	控制Stream对象的数据源。
Stream.record()	记录进入流的所有的数据。
Stream.send()	发送一个带有参数的调用至一个流的所有的订阅者。
Stream.setBufferTime()	以秒为单位设置缓冲时间的长度。
Stream.setVirtualPath()	为视频流的回放设置虚拟目录路径。
Stream.size()	静态的；以字节为单位返回流的大小。

Stream类的事件处理器汇总

事件处理器	描述
Stream.onStatus	当状态发生改变时调用。

Stream.bufferTime

可用性

Flash Communication Server MX

用法

myStream.bufferTime

描述

属性（只读）。指出一个流在被播放前要用多长时间来缓冲消息。只有当播放一个来自某个远端服务器的流或是当播放一
个本地记录的流时，这个属性才被应用。要设置这个属性，使用Stream.setBuffertime。

一个消息是在Flash Media Server和Flash P.layer之间来回发送的数据。这些数据被分成小包（消息），每一个消息都有一个
类型（音频、视频，或数据）。

Stream.clear()

可用性

Flash Communication Server MX

用法

myStream.clear()

参数

无

返回

一个布尔值。如果调用成功，则返回true；否则，返回false。

描述

方法。从服务器上删除一个记录的流文件（FLV）。

例子

这个范例删除一个名为foo.flv的记录的流。在这个流被删除之前，这个范例定义了一个onStatus处理器，这个处理器使用两
个信息对象错误代码-NetStream.Clear.Success和NetStream.Clear.Failed来发送状态消息至应用程序的日志文件和Application检查器。

```
s = Stream.get("foo");
if (s){
   s.onStatus = function(info){
      if(info.code == "NetStream.Clear.Success"){
         trace("Stream cleared successfully.");
      }
      if(info.code == "NetStream.Clear.Failed"){
         trace("Failed to clear stream.");
      }
   };
   s.clear();
}
```

注意：Stream信息对象几乎与客户机端ActionScript NetStream信息对象完全一样。

Stream.flush()

可用性

Flash Media Server 2

用法

myStream.flush()

参数

无

返回

一个布尔值。如果缓冲区被成功清出，则返回true；否则，返回false。

描述

方法。清出一个流。如果这个流是用于记录的，则flush()方法会把与这个流关联的缓冲区的内容写入到FLV文件。

高度推荐你在只包含数据的流上调用flush()。如果你在既包含数据又包含音频、视频或二者都有的流上

调用flush()方法的
话，会发生同步问题。

例子

下面的范例清出myStream流：

```
// 建立这个服务器流
application.myStream = Stream.get("foo");

if (application.myStream){
    application.myStream.record();
    application.myStream.send("test", "hello world");
    application.myStream.flush();
}
```

Stream.get()

可用性

Flash Communication Server MX

用法

myStream.get(name)

参数

name 要返回的流实例的名字。

返回

对一个流实例的引用。

描述

方法（静态）。返回对一个Stream对象的引用。如果请求的对象没有被找到，则一个新的实例会被创建。

例子

这个范例得到流foo并把它指派给变量playStream。然后，它从playStream上调用Stream.play方法。

```
function onProcessCmd(cmd){
   var playStream = Stream.get("foo");
   playStream.play("file1", 0, -1);
}
```

Stream.length()

可用性

Flash Communication Server MX

用法

myStream.length(name[, virtualKey])

参数

name 一个记录的流（FLV）文件或MP3文件的名字。要得到一个MP3文件的长度，以mp3:作为文件名的前导。例

如，"mp3:beethoven"。

virtualKey 一个字符串，指出一个键值。自Flash Media Server 2起，流名不总是唯一的；你可以用同一个名字创建多个流，
把它们放置到不同的物理目录中，并使用vhost.xml文件中的VirtualDirectory和VirtualKeys部分把客户机导向适当的流。因为
Stream.length()方法并不与一个客户机相关联，而是连接到服务器上的一个流，因此，你可能需要指定一个虚拟键来标识正确的
流。要获得有关键的更多信息，参看Client.virtualKey。这个参数是可选的。

返回

一个记录的流文件或MP3文件的以秒为单位的长度。

例子

这个范例得到记录的流文件myVideo的长度并把它指派给变量streamLen：

```
function onProcessCmd(cmd){
   var streamLen = Stream.length("myVideo");
   trace("Length: " + streamLen + "\n");
}
   trace("Length: " + streamLen + "\n"); 的长度并把它指派给变量streamLen：
}
```

这个范例得到MP3文件beethoven.mp3的长度并把它指派给变量streamLen：

```
function onProcessCmd(cmd){
   var streamLen = Stream.length("mp3:beethoven");
   trace("Length: " + streamLen + "\n");
}
```

Stream.name

可用性

Flash Communication Server MX

用法

myStream.name

描述

属性（只读）。包含与一个活动流关联的一个唯一的字符串。你也可以把这用作一个索引来寻找一个应用程序中的一个
流。

例子

下面的函数getStreamName，获取一个流引用myStream作为一个参数，并返回与之关联的流的名字。

```
function getStreamName (myStream) {
   return myStream.name;
```

Stream.onStatus

可用性

Flash Communication Server MX

用法

Flash Communication Server MX

用法

myStream.onStatus = function([infoObject]) {}

参数

infoObject 一个可选的参数，依照状态消息被定义。

返回

无

描述

事件处理器。每次一个Stream对象的状态改变时调用。例如，如果你在一个流中播放一个文件，则Stream.onStatus就会被调
用。使用Stream.onStatus来检查何时播放开始和结束，何时记录开始和结束，等等。

例子

这个范例定义了一个函数，任何时候当Stream.onStatus事件被调用时，这个函数就会被执行：

```
s = Stream.get("foo");
s.onStatus = function(info){
    //在这里插入代码
};
```

Stream.play()

可用性

Flash Communication Server MX

用法

myStream.play(streamName, [startTime, length, reset, remoteConnection, virtualKey])

参数

streamName 任何被发布的实况流、记录的流（FLV文件），或是MP3文件的名字。

为了回放FLV文件-针对记录的流的默认的Flash文件格式，指定流的名字（例如，"myVideo"）。要回放你已经保存在服务
器上的MP3文件，或MP3文件的ID3标签，你必须用mp3:或id3:作为流名的前导（例如，"mp3:bolero"或"id3:bolero"）。

startTime 流回放的开始时刻，以秒为单位。如果没有值被指定的话，它被设置为-2。如果startTime是-2的话，则服务器会
果指定的实况流不可用的话，则服务器会等待一个发布者。如果startTime大于或等于，则服务器会播放一个具有在streamName
中指定的名字的记录流，从给定的时刻开始播放。如果没有记录的流被发现，则play方法会被忽略。如果指定了一个除-1以外的
负值的话，则服务器会把它解释为-2。这是一个可选的参数。

length 播放的长度，以秒为单位。对于一个实况流，值-1会一直播放流，只要这个流存在。任何正值会播放流相应的秒
数。对于一个记录的流，值-1会播放整个文件，值回返回第一个视频帧。任何正值播放流相应的秒数。默认情况下，这个值
是-1。这是一个可选的参数。

reset 一个布尔值，或数字，刷新播放流。如果reset是false（），则服务器会维持一个播放列表，对Stream.play的每一个调
用都会被追加到播放列表的末端，以便下一个播放不会开始，直到上一个播放结束。你可以使用这个技术来创建一个动态的播
是-1。这是一个可选的参数。reset是true（），则任何播放的流都会停止，而播放列表会被重置。默认情况下，这个值是true。

reset 一个布尔值，或数字，刷新播放流。如果reset参数指定一个值或 reset是false（），则服务器会维持一个播放列表，对Stream.play的每一个调false（）
用都会被追加到播放列表的末端，以便下一个播放不会开始，直到上一个播放结束。你可以使用这个技术来创建一个动态的播
放列表。如果reset是true（），则任何播放的流都会停止，而播放列表会被重置。默认情况下，这个值是true。

你也可以为reset参数指定一个值或，当播放包含消息数据的记录的流文件时，这是有用的。这些值分别类似于false（）
和true（）：值维持一个播放列表，而值重置播放列表。但不管怎么说，差别是为reset指定或会立即返回指定的记录流中
所有的消息，而不是以这些消息最初被记录时的间隔（这是默认的行为）。

remoteConnection 对一个NetConnection对象的引用，这个对象被用于连接到一个远端服务器。如果这个参数被提供的话，
则请求的流会自远端服务器播放。这个参数是可选的。

virtualKey 一个字符串，指出一个键值。自Flash Media Server 2起，流名不总是唯一的；你可以用同一个名字创建多个流，
把它们放置到不同的物理目录中，并使用vhost.xml文件中的VirtualDirectory和VirtualKeys部分把客户机导向适当的流。因为
Stream.length()方法并不与一个客户机相关联，而是连接到服务器上的一个流，因此，你可能需要指定一个虚拟键来标识正确的
流。要获得有关键的更多信息，参看Client.virtualKey。这个参数是可选的。

返回

一个布尔值。如果Stream.play调用被服务器接受了并被添加到播放列表了，则返回true；否则，返回false。如果服务器没有
发现这个流或如果出现一个错误，则Stream.play方法会失败。要得到有关Stream.play调用的信息，你可以定义一个
Stream.onStatus处理器来捕获播放状态或错误。

如果streamName参数是false，则流会停止播放。如果停止是成功的，则一个布尔值true会被返回；否则，返回false。

描述

方法。用一个可选的开始时刻、持续时间，以及用于刷新任何先前播放的流的重置标记来控制一个流的数据
源。Stream.play()方法也有一个参数可以让你引用一个NetConnection对象来播放一个来自另一个服务器的流。Stream.play()方法
允许你做下列事情：

在服务器间绑定流。

创建一个插孔用来在实况流和记录的流之间进行切换。

将不同的流联合成一个记录流。

你可以联合多个流以便为客户机创建一个播放列表。服务器端Stream.play()方法的行为与客户机端NetStream.play方法的行
为有点不同。服务器上的一个play调用类似于客户机上的一个publish调用。它控制着引入一个流的数据源。

当你在服务器上调用
Stream.play时，服务器就成为了发布者。因为服务器有比客户机更高的优先权，因此，如果服务器在同一个流上调用一个play方
法的话，则客户机会被强制不能发布这个流。

一般而言，如果任何记录的流被包含在一个服务器播放列表中，则你就不能把这个服务器流作为一个实况流进行播放。

注意：播放自一个远端服务器的流依赖于NetConnection对象是一个实况流。

如果你需要一个值来开始一个流，则你可能需要改变服务器上的Application.xml文件中的"Enhanced seeking"标记。"Enhanced
seeking"是Application.xml文件中的一个布尔值标记。默认情况下，这个标记被设置为false。当一个播放发生时，服务器会搜索最
接近的视频帧并从那个帧开始播放。例如，如果你希望从时间处开始播放，而实际上只有时间和时间处才有关键帧，则
关键帧-基于时间处的先前存在的关键帧-用于从到的每一个关键帧。即使在搜索时刻不存在一个关键帧，服务器也会生
成一个关键帧，这会在服务器上造成一些处理时间。

例子

这个范例展示了流如何能够被绑定于服务器之间：

```
application.myRemoteConn = new NetConnection();
application.myRemoteConn.onStatus = function(info){
application.myRemoteConn = new NetConnection();    trace("Connection to remote server status " + info.code + "\n");
application.myRemoteConn.onStatus = function(info){
   trace("Connection to remote server status " + info.code + "\n");
   //告诉所有的客户机
   for (var i = 0; i < application.clients.length; i++){
      application.clients[i].call("onServerStatus", null,
      info.code, info.description);
   }//使用NetConnection对象连接到一个远端服务器
application.myRemoteConn.connect(rtmp://movie.com/movieApp);
//建立服务器流
application.myStream = Stream.get("foo");
if (application.myStream){
   application.myStream.play("Movie1", 0, -1, true, application.myRemoteConn);
if (application.myStream){}
下面的范例展示了 application.myStream.play("Movie1", 0, -1, true, application.myRemoteConn);
}
```

下面的范例展示了Stream.play被用作一个插孔，用于在实况流和记录的流之间进行切换：

```
//建立服务器流
application.myStream = Stream.get("foo");
if (application.myStream){
   //这个服务器流将播放"Live1"、"Record1"和"Live2"每个秒钟
   application.myStream.play("Live1", -1, 5);
   application.myStream.play("Record1", 0, 5, false);
   application.myStream.play("Live2", -1, 5, false);
}
```

下面的范例把不同的流联合成一个记录流：

```
//建立服务器流
application.myStream = Stream.get("foo");
if (application.myStream){
  //如同前面的例子，这个服务器流将播放"Live1"、"Record1"和"Live2"每个秒钟
  //但这次，所有的数据都将被记录到一个记录流"foo"。
  application.myStream.record();
  application.myStream.play("Live1", -1, 5);
  application.myStream.play("Record1", 0, 5, false);
  application.myStream.play("Live2", -1, 5, false);
}
```

下面的范例使用Stream.play来停止播放流foo：

```
application.myStream.play(false);
```

下面的范例创建了一个三个MP3文件（beethoven.mp3、mozart.mp3和chopin.mp3）的播放列表，并在实况流foo上依次播放
每一个文件：

```
application.myStream = Stream.get("foo");
if(application.myStream) {
  application.myStream.play("mp3:beethoven", 0);
  application.myStream.play("mp3:mozart", 0, false);
  application.myStream.play("mp3:chopin.mp3", 0, false);
}
```

在下面的例子中，存在于记录的流文件log.flv中的数据消息以它们最初被记录的间隔被返回。

```
application.myStream = Stream.get("data");
if (application.myStream) {
  application.myStream.play("log", 0, -1);
}
```

在下面的范例中，存在于记录的流文件log.flv中的数据消息被一次性返回，而不是以它们最初被记录的间隔被返回。

```
application.myStream = Stream.get("data");
if (application.myStream) {
  application.myStream.play("log", 0, -1, 2);
}
```

Stream.record()

可用性

Flash Communication Server MX

用法

myStream.record(flag)

参数

myStream.record(flag)这个参数可以有值record、append或false。如果值是record，则数据文件会被覆盖，假如它存在的话。如果值是

参数

flag 这个参数可以有值record、append或false。如果值是record，则数据文件会被覆盖，假如它存在的话。如果值是

append，则引入数据会被追加到现有文件的末尾。如果值是false，则任何先前记录都会停止。默认情况下，这个值是record。

返回

一个布尔值。如果记录成功，则返回true；否则，返回false。

描述

方法。记录通过一个Stream对象的所有的数据。

例子

这个例子打开一个流s，并且，当其打开时，播放并记录sample。因为没有值被传递给record方法，因此，默认值record被传
递。

```
//要开始记录
s = Stream.get("foo");
if (s){
   s.play("sample");
   s.record();
}
//要停止记录
s = Stream.get("foo");
if (s){
   s.record(false);
}
```

Stream.send()

可用性

Flash Communication Server MX

用法

myStream.send(handlerName, [p1, ..., pN])

参数

handlerName 调用在客户机端ActionScript代码中指定的处理器。handlerName的值是与订阅的Stream对象有关的一个方法的
名字。例如，如果handlerName是doSomething，则位于这个流级的doSomething方法会被用所有的p1, ..., pN参数调用。不同于
Client.call()和NetConnection.call()中的方法名，这个处理器名只能是一级深度（也就是说，它不能以object/method这种形式出
现）。

注意：不要用一个内建的方法作为一个处理器的名字。例如，如果这个处理器名是close的话，则这个订阅流将被关闭。

返回 参数可以是任何ActionScript类型，包括对其他ActionScript对象的引用。当这个指定的处理器在Flash客户机上被

一个布尔值。如果消息被发送到客户机，则返回true；否则，返回false。

描述

方法。把一个消息发送给订阅流的所有的客户机，并且，这个消息会被在客户机上指定的处理器处理。因为服务器比客户

机有更高的优先权，因此，服务器可以在一个由一个客户机拥有的流上发送消息。不同于Stream.play()方法，服务器不需要为了
发送一个消息而从客户机上获取一个流的所有者身份。在send()被调用之后，客户机仍然作为一个发布者拥有这个流。
发送一个消息而从客户机上获取一个流的所有者身份。在send()被调用之后，客户机仍然作为一个发布者拥有这个流。

例子　　Stream对象上的Test方法，并发送给它字符串“hello world”：

这个范例调用客户机端Stream对象上的Test方法，并发送给它字符串“hello world”：

```
application.streams["foo"].send("Test", "hello world");
```

下面的范例是接收Stream.send调用的客户机端ActionScript。方法Test被定义于Stream对象：

```
tStream.Test = function(str) {
  //插入用来处理字符串的代码
}
}
```

Stream.setBufferTime()

可用性

Flash Communication Server MX

用法

myStream.setBufferTime()

描述

方法。增加消息队列的长度。当你播放一个来自远端服务器的流时，Stream.setBufferTime方法会发送一个消息至这个远端
服务器，用来调整消息队列的长度。消息队列的默认长度是秒。当经由一个低带宽的网络播放一个高质量的记录流时，你应该
把缓冲时间设置的更长。

当一个用户点击了一个应用程序中的一个搜索按钮，被缓冲的包就会被发送到服务器。这个被缓冲的在一个Flash Media
Server应用程序中的搜索会在服务器上发生；Flash Media Server不支持客户机端缓冲。搜索时间可以比缓冲尺寸更小或更大，并
且，它与缓冲尺寸没有直接的关系。每次服务器接收到一个来自Flash Player的搜索请求，它就会清理服务器上的消息队列。服
务器会试图搜索期待的位置并开始再次填充队列。同时，在一个搜索请求之后，Flash Player也会清理它自己的缓冲器，并且，
最终，在服务器开始发送新的消息之后，Flash Player的缓冲器会被填充。

Stream.setVirtualPath()

可用性

Flash Media Server 2

用法

myStream.setVirtualPath(virtualPath, directory, virtualKey)

参数

virtualPath 一个字符串，指出一个流的虚拟目录路径。如果这个流不是定位于这个虚拟目录，则默认的虚拟目录路径会被
搜索。

strDirectory 一个字符串，指出存储流的物理目录。

virtualKey 一个字符串，设置或移除每一个虚拟目录入口的键值。

返回

无。

描述

方法；设置或移除视频流回放的虚拟目录路径。

Flash Media Server可以向客户机呈递两种视频编解码器：Sorenson Spark和On2 VP6编解码器。Flash Player 8支持两种编解
码器；Flash Player 7和更早的版本仅支持Sorenson Spark。你可以使用Stream.setVirtualPath()方法来动态的供给Flash Player 8客户
机一种视频流的更好版本。

当Flash Player从Flash Media Server请求一个流时，Flash Player的版本决定了服务器端Client.virtualKey属性（典型的是Flash
Player版本的一个映射）。然后，Flash Media Server从这个虚拟键被映射到的那个虚拟目录向客户机提供一个流。例如，如果客
户机是Flash Player 8，这个虚拟键可能会映射到On2流目录，而这个客户机会被提供一种更高质量的流。如果客户机是Flash
Player 7，则这个虚拟键可能会映射到默认的流目录。这一功能允许你以对每一个客户机平台最适宜的编码来向所有的客户机提
供相同的内容。

虚拟键和虚拟目录之间的映射是在vhost.xml文件中建立的。（要获得有关vhost.xml文件的更多信息，参看《管理Flash
Media Server》中的“Vhost.xml文件”。）当你调用setVirtualPath()时，你实际上正在改变这个文件中的VirtualDirectory标签中的
值。

下面的范例展示了vhost.xml文件中的VirtualKeys部分，在这里，虚拟键可以被映射到一个Flash Player版本范围：

```
<VirtualKeys>
    <Key from="WIN 7,0,19,0" to="WIN 9,0,0,0"></Key>
    <Key from="WIN 6,0,0,0" to="WIN 7,0,18,0"></Key>
    <Key from="MAC 6,0,0,0" to="MAC 7,0,55,0"></Key>
</VirtualKeys>
```

默认情况下，在这个键标签中没有值。要实现这个功能，你必须添加键值，或者是直接在vhost.xml文件中添加，或者是使
用setVirtualPath()方法。例如，要实现这个功能，把第一个键设置为A，把第二个键设置为B，就像下面的范例中展示的这样：

```
<VirtualKeys>
    <Key from="WIN 7,0,19,0" to="WIN 9,0,0,0">A</Key>
    <Key from="WIN 6,0,0,0" to="WIN 7,0,18,0">B</Key>
```

</VirtualKeys>

下面的范例展示了vhost.xml文件的VirtualDirectory部分可能看起来像个什么样（默认情况下，键值和Streams值并不存在于
这个文件中）。虚拟键被映射到一个虚拟路径和一个物理目录，二者之间用分号分隔（例如，foo;c:\streams）。要为不同的
这个文件中）。虚拟键被映射到一个虚拟路径和一个物理目录，二者之间用分号分隔（例如，版本建立多个虚拟目录，为每一个Streams
foo;c:\streams）。要为不同的
Flash Player版本建立多个虚拟目录，为每一个Streams标签使用具有不同物理目录的相同的虚拟路径，就像下面的范例中所展示
的这样：

```
<VirtualDirectory>
<!-- 为记录的流指定虚拟目录映射。 -->
<!-- 要为一个流指定多个虚拟目录映射，添加额外的<Streams>标签； -->
<!-- 每一个对应一个虚拟目录映射。 -->
<!-- 虚拟目录的语法如下所示： -->
<!-- <Streams>foo;c:\data</Streams>. 这将把所有其名字以"foo/"开始的流映射到物理目录c:\data. -->
<!-- 例如，名为"foo/bar"的流将映射到物理文件"c:\data\bar.flv"。类似的，如果你有一个 -->
<!-- 名为"foo/bar/x"的流，则我们会首先试图找到"foo/bar"的虚拟目录映射。-->
<!-- 如果没能找到的话，则我们会继续查找"foo"的虚拟目录映射。-->
<!-- 因为这样的虚拟目录映射是存在的，因此，这个流 -->
<!-- "foo/bar"会对应到文件"c:\data\bar\x.flv"。 -->
    <Streams key="A">foo;c:\streams\on2</Streams>
    <Streams key="B">foo;c:\streams\sorenson</Streams>
    <Streams key="">foo;c:\streams</Streams>
</VirtualDirectory>
```

注意：你也可以在vhost.xml文件中直接改变VirtualKeys和VirtualDirectory标签的值。这是最常见的用法。

例子

下面的代码把虚拟键设置为B，把虚拟路径设置为/foo，把物理目录设置为c:\streams\on2：

```
myStream.setVirtualPath("B", "/foo", "c:\streams\on2");
```

Stream.size()

可用性

Flash Media Server 2

用法

```
Stream.size(name[, virtualKey])
```

参数

name 一个字符串，指出一个流的名字。你可以在name属性中使用格式标签来指定类型。

virtualKey 一个字符串，指出一个键值。自Flash Media Server 2起，流名不总是唯一的；你可以用同一个名字创建多个流，
把它们放置到不同的物理目录中，并使用vhost.xml文件中的VirtualDirectory和VirtualKeys部分把客户机导向适当的流。因为

Stream.length()方法并不与一个客户机相关联，而是连接到服务器上的一个流，因此，你可能需要指定一个虚拟键来标识正确的
流。要获得有关键的更多信息，参看Client.virtualKey。这个参数是可选的。

返回

一个数字，指出流的大小；如果请求的流没有找到，则返回。

描述

方法（静态的）；以字节为单位返回一个记录的流的大小。

例子

下面的范例分别返回一个流和一个MP3流的大小：

```
function onProcessCmd(cmd){

    trace("Size: " + streamSize + "\n");       //

}       var streamSize = Stream.size("foo");


//对于mp3

function onProcessCmd(cmd){

function onProcessCmd(cmd){       // Insert code here...

    // Insert code here...       var streamSize = Stream.size("mp3:foo" );

    var streamSize = Stream.size("mp3:foo" );
    trace("Size: " + streamSize + "\n");
}
```

Stream.syncWrite()

可用性

Flash Media Server 2

用法

myStream.syncWrite

描述

属性；一个布尔值，用来控制当流被记录时什么时候流把缓冲区中的内容写入一个FLV文件。当syncWrite为true时，所有经
由这个流传递的消息都会被立即清出至FLV文件。高度推荐用户仅在只包含数据的流中把syncWrite设置为true。当在一个即包含
数据又包含音频、视频，或二者都有的流中把syncWrite设置为true时可能会导致同步问题。

```
// 假定foo是一个只包含数据的流
application.myStream = Stream.get("foo");

if (application.myStream){
```

```
        application.myStream.syncWrite = true;
        application.myStream.record();
        application.myStream.syncWrite = true;        application.myStream.send("test", "hello world");
        application.myStream.record();}
        application.myStream.send("test", "hello world");
    }
```

trace()

可用性

Flash Communication Server MX

用法

trace(expression)

参数

expression 任何合法的ActionScript表达式。

返回

无

描述

方法（全局）。求值一个表达式并显示其值。当服务器正运行于控制台模式时，这个表达式会出现在Console（控制台）窗
口中；这个表达式也会被发布到logs/application应用程序名流，它在管理控制台或在Application检查器中可用。

在其被输出前，如果可能的话，trace表达式中的值会被转换成字符串。

你可以使用trace()函数来调试脚本，记录编程笔记，或是在测试文件时显示消息。trace()函数类似于JavaScript中的alert函
数。

WebService类

可用性

Flash Media Server 2

描述

你可以使用WebService类来创建和访问一个WSDL/SOAP Web服务。构成Flash Media Server Web服务功能的有这样几个类：
WebService类、SOAPFault类、SOAPCall类，以及Log类。

注意：WebService类不能检索由Web服务返回的复杂数据或数组。同样，WebService类也不支持安全功能。

下面的步骤勾勒出了创建和访问一个Web服务的过程：

要创建和访问一个Web服务：

1. 准备WSDL位置：

```
var wsdlURI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";
```

2. 利用这个WSDL位置例示这个Web服务对象：

```
stockService = new WebService(wsdlURI);
```

3. （可选的）通过WebService.onLoad处理器来处理WSDL解析和Web服务例示事件：

```
// 处理WSDL装载事件。
stockService.onLoad = function(wsdl){
    wsdlField.text = wsdl;
}
```

4. （可选的）如果WSDL没有装载的话，处理这个失败：

```
// 如果WSDL没能装载，则onFault事件被激发。
stockService.onFault = function(fault){
    wsdlField.text = fault.faultstring;
}
```

5. （可选的）设置SOAP标题：

```
// 如果需要标题，可以像下面这样来添加它们：
var myHeader = new XML(headerSource);
stockService.addHeader(myHeader);
```

6. 调用一个Web服务操作：

```
// 方法调用返回一个异步回调。
callback = stockService.doCompanyInfo("anyuser", "anypassword", "MACR");
// 注意：如果服务本身没有被创建，则回调是undefined。
// （且service.onFault也会被调用）。
```

7. 处理由调用返回的输出或是错误故障：

```
// 处理一个成功的结果。
callback.onResult = function(result){
// 接收SOAP输出，本例中它被解串为一个结构体（ActionScript对象）。
    stock.companyInfo = result;
}
// 处理一个错误结果。
callback.onFault = function(fault){
// 捕获SOAP故障并按照这个应用程序的需要来处理它。
    stock.companyInfo = fault.faultstring;
}
```

WebService类的事件处理器汇总

事件处理器	描述
WebService.onFault	当一个错误在WSDL解析期间发生时调用。
WebService.onLoad	当Web服务已经成功被装载且被解析为它的WSDL文件时调用。

WebService类的构造器

可用性用法

new WebService(wsdlURI)

参数

wsdlURL 一个字符串，指定一个WSDL URL的URL。

返回

一个返回WebService对象。

一个描述WebService对象。

描述

构造器；创建一个新的WebService对象。在你调用任何WebService类的方法前，你必须使用这个构造器来创建一个
WebService对象。

例子

下面的范例准备了一个WSDL位置并把它传递给WebService构造器以创建一个新的WebService对象stockService：

```
var wsdlURI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";
stockService = new WebService(wsdlURI);
```

WebService.onFault

可用性

Flash Media Server 2

用法

myWS.onFault(fault){}

参数

fault 一个XML SOAP故障的对象版本。

返回

无。

描述

事件处理器；当一个错误在WSDL解析期间发生时调用。Web服务的特性是把解析和网络问题转换成SOAP故障以简化处
理。描述

事件处理器；当一个错误在WSDL解析期间发生时调用。Web服务的特性是把解析和网络问题转换成SOAP故障以简化处
理。

例子

下面的范例在WSDL装载失败且onFault事件激发时在一个文本域中显示故障代码：

```
// 准备WSDL位置：
var wsdlURI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";
```

```
// 利用WSDL位置例示Web服务对象：
stockService = new WebService(wsdlURI);

// 处理WSDL解析和Web服务例示事件。
stockService.onLoad = function(wsdl){
    wsdlField.text = wsdl;
}

// 如果这个WSDL装载失败，onFault事件被激发：
stockService.onFault = function(fault){
    wsdlField.text = fault.faultstring;
}
```

WebService.onLoad

可用性

Flash Media Server 2

用法

myWS.onLoad(wsdldocument)

参数

wsdldocument 一个WSDL XML文档。

返回

无。

描述

描述 Web服务成功装载且解析了它的WSDL文件时调用。操作可以在应用程序中在这个事件出现之前被调用；

事件处理器；当Web服务成功装载且解析了它的WSDL文件时调用。操作可以在应用程序中在这个事件出现之前被调用；

当其发生时，它们会在内部排队但并不真的被发送，直到WSDL已装载。

例子

在下面的范例中，onLoad事件被用于处理WSDL解析：

```
//准备WSDL位置：
var wsdlURI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";

//利用WSDL位置例示Web服务对象：
stockService = new WebService(wsdlURI);

//处理WSDL解析和Web服务例示事件。
stockService.onLoad = function(wsdl){
    wsdlField.text = wsdl;
}
```

XML类

可用性

Flash Media Server 2

描述

XML类可以让你装载、解析、发送、建立，以及操作XML文档树。在调用任何XML类的方法之前，你必须使用构造器new
XML()来创建一个XML对象。

一个XML文档在Flash中是通过XML类来描述的。层次结构中的每一个元素都由一个XMLNode对象来描述。

注意：XML和XMLNode对象是模仿W3C DOM Level 1 Recommendation（http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html）：该推荐方法指定节点接口和文档接口。文档接口继承自节点接口，并添加了诸如createElement()
和createTextNode()这样的方法。在ActionScript中，XML和XMLNode对象旨在区分类似功能。

XML类的属性汇总

属性	描述
XML.attributes	一个对象，包含指定的XML对象的所有属性。
XML.childNodes	只读；一个指定的XML对象的孩子数组。数组中的每一个元素都是对一个描述了一个子节点的 XML对象的引用。
XML.contentType	传输到服务器的MIME类型。
XML.docTypeDeclXML.docTypeDecl	设置并返回有关一个设置并返回有关一个XMLXML对象的对象的DOCTYPEDOCTYPE声明的信息。声明的信息。
XML.firstChild	只读；引用指定节点的列表中的第一个孩子。
XML.ignoreWhite	当设置为true时，在解析过程中丢弃那些只包含空白的文本节点。
XML.lastChild	只读；引用指定节点的列表中的最后一个孩子。
XML.loaded	检查指定的XML对象是否被装载。
XML.localNameXML.localName	只读；只读；XMLXML节点名的本地名部分。节点名的本地名部分。
XML.namespaceURI	只读；如果XML节点有一个前缀，则namespaceURI就是这个前缀（URI）的XML命名空间 （xmlns）声明的值，其代表性的称谓是命名空间URI。
XML.nextSibling	只读；引用父节点的孩子列表中的下一个兄弟。
XML.nodeName	一个XML对象的节点名。
XML.nodeType	只读；指定的节点（XML元素或文本节点）的类型。
XML.previousSibling	只读；引用父节点的孩子列表中的上一个兄弟。

XML.status	一个数字状态代码，指出一个XML文档解析操作的成功或失败。
XML.xmlDecl	指定有关一个文档的XML声明的信息。

XML类的方法汇总

方法	描述
XML.addRequestHeader()	为POST操作添加或改变HTTP标题。
XML.appendChild()	向指定的对象的孩子列表的末尾追加一个节点。

方法	描述
XML.addRequestHeader()	为POST操作添加或改变HTTP标题。
XML.appendChild()	向指定的对象的孩子列表的末尾追加一个节点。
XML.cloneNode()	克隆指定的节点，并且，你也可以可选的递归克隆所有的孩子。
XML.createElement()	创建一个新的XML元素。
XML.createTextNode()	创建一个新的XMl文本节点。
XML.getBytesLoaded()	返回指定的XML文档已装载的字节数。
XML.getBytesTotal()	以字节为单位返回XML文档的大小。
XML.getNamespaceForPrefix()	返回与节点的指定前缀相关联的命名空间的URI。
XML.getPrefixForNamespace()	返回与节点的指定的命名空间URI相关联的前缀。
XML.hasChildNodes()	如果指定的节点有子节点，则返回true；否则，返回false。
XML.insertBefore()	在指定的节点的孩子列表的一个现有节点的前面插入一个节点。
XML.load()	从一个URL处装载一个文档（由XML对象指定）。
XML.parseXML()	把一个XML文档解析为指定的XML对象树。
XML.removeNode()	从其父中删除指定的节点。
XML.send()	把指定的XML对象发送到一个URL。
XML.sendAndLoad()	把指定的XML对象发送到一个URL，并把服务器的响应装载到另一个XML对象。
XML.toString()	把指定的节点及其任何孩子转换成XML文本。

XML类的事件处理器汇总

事件处理器	描述
XML.onData	当XML文本已经从服务器上被完全下载时，或者当从一个服务

器下载XML文本时发
生了一个错误时调用。

XML.onHTTPStatus 当Flash Media Server从服务器接收一个HTTP状态代码时调用。这个处理器可以让你
捕获并根据HTTP状态代码来动作。

XML.onLoad 返回一个布尔值，指出XML对象是否被用XML.load()或XML.sendAndLoad()成功装
载。

XML类的集合汇总

属性	描述
XML.attributes	返回一个联合数组，它包含指定的节点的所有属性。
XML.childNodes	只读；返回一个数组，它包含对指定的节点的子节点的引用。

XML类的构造器

可用性

Flash Media Server 2

用法

new XML([source])

参数

source 一个字符串；这个XML文本被解析以创建新的XML对象。

返回

一个对XML对象的引用。

描述

构造器；创建一个新的XML对象。在你可以调用任何XML类的方法前，你必须使用构造器来创建一个XML对象。

注意：使用createElement()和createTextNode()方法来把元素和文本节点添加到一个XML文档树。

例子

下面的范例创建了一个新的空XML对象：

```
var my_xml = new XML();
```

下面的范例通过解析在source参数中指定的XML文本创建了一个XML对象，并用作为结果的XML文档树填充这个新创建的
XML对象：

```
var other_xml = new XML("<state name=\"California\"><city>San Francisco</city></state>");
```

XML.addRequestHeader()

可用性

Flash Media Server 2

用法

my_xml.addRequestHeader(headerName, headerValue)

my_xml.addRequestHeader(["headerName_1", "headerValue_1" ... "headerName_n", "headerValue_n"])

参数

headerName一个字符串，表示HTTP请求标头名称。

headerValue 一个字符串，表示与header关联的值。

返回

无。

描述

方法；添加或更改与POST动作一起发送的HTTP请求标题（如Content-Type或SOAPAction）。这个方法有两种可能的使用
情况：你可以传递两个字符串、header和headerValue，或者你可以传递一个字符串数组、替换的标题名称和标题值。

如果通过多次调用来设置相同的标题名称，则每个后继值将替换在上一次调用中设置的值。

下列标准HTTP头不能用这个方法来添加或改变用这个方法来添加或改变：Accept-Ranges、Age、Allow、Allowed、
Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-
Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、
Vary、Via、Warning和WWW-Authenticate。

例子

下面的范例把一个名为SOAPAction的值为Foo的定制HTTP头添加到一个名为my_xml的XML对象上：

```
my_xml.addRequestHeader("SOAPAction", "'Foo'");
```

下面的范例创建了一个名为headers的数组，它包含两个可交替的HTTP头及其关联的值。这个数组被作为一个参数传递给
addRequestHeader()方法。

```
var headers = new Array("Content-Type", "text/plain", "X-ClientAppVersion", "2.0");
my_xml.addRequestHeader(headers);
```

XML.appendChild ()

可用性

Flash Media Server 2

用法

my_xml.appendChild(childNode)

参数

childNode 一个XML Node对象，描述了要被从它的当前位置移动到my_xml对象的孩子列表中的节点。

返回

无。

描述

方法；把指定的节点追加到XML对象的孩子列表中。这个方法直接在由childNode参数引用的节点上操作；它不是追加节点
的拷贝。如果要被追加的节点已经存在于另一个树结构中，则把节点追加到新位置将导致它被从它当前的位置处移除。如果
childNode参数引用的节点已经存在于另一个XML树结构中，则追加的子节点会在其从它现在的父节点中移除后被放置到新的树
结构中。

例子

下面的范例执行列表中的动作：

1. 创建两个空XML文档，doc1和doc2。
2. 利用createElement()方法创建一个新节点，并利用appendChild()方法把它追加到名为doc1的XML文档中。
3. 通过把根节点从doc1移动到doc2展示如何利用appendChild()方法来移动一个节点。
4. 从doc2克隆根节点并把它追加到doc1。
5. 创建一个新的节点并把它追加到XML文档doc1的根节点。

```
var doc1 = new XML();
var doc2 = new XML();

// 创建一个根节点并把它添加到doc1。
var rootnode = doc1.createElement("root");
doc1.appendChild(rootnode);
trace ("doc1: " + doc1); // 输出：doc1: <root />
trace ("doc2: " + doc2); // 输出：doc2:

// 把根节点移动到doc2。
doc2.appendChild(rootnode);
trace ("doc1: " + doc1); // 输出：doc1:
trace ("doc2: " + doc2); // 输出：doc2: <root />

// 克隆根节点并把它追加到doc1.
var clone = doc2.firstChild.cloneNode(true);
doc1.appendChild(clone);
trace ("doc1: " + doc1); // 输出：doc1: <root />
trace ("doc2: " + doc2); // 输出：doc2: <root />

// 创建一个新节点追加到doc1的根节点（名为clone）。
var newNode = doc1.createElement("newbie");
clone.appendChild(newNode);
trace ("doc1: " + doc1); // 输出：doc1: <root><newbie /></root>
```

中，trace()语句的输出会出现在应用程序的日志文件和Application检查器中。

XML.attributes()

可用性

Flash Media Server 2

用法

my_xml.attributes用法

my_xml.attributes

描述

属性；一个对象，包含指定的XML对象的所有属性。联合数组使用键作为索引，而不是使用常规数组所用的顺序排列的整
数索引。在XML.attributes联合数组中，键索引是一个字符串，描述了属性的名字。与键索引关联的值是与属性关联的字符串
值。例如，如果你有一个名为color的属性，则你可以使用color作为键索引来检索这个属性的值，就像下面的代码中展示的那
样：

```
var myColor = doc.firstChild.attributes.color
```

例子

下面的范例展示了XML属性名：

```
// 创建一个具有名为'name'的其值为'Val'的属性的名为'mytag'的标签。
// 把'name'属性的值指派给变量y。
var y = doc.firstChild.attributes.name;
trace (y);  // 输出：Val

// 创建一个值为'first'名为'order'的新属性。
doc.firstChild.attributes.order = "first";
// 创建一个值为'first'名为'order'的新属性。
doc.firstChild.attributes.order = "first";

// 把'order'属性的值指派给变量z。
var z = doc.firstChild.attributes.order
trace(z);  // 输出：first
```

XML.childNodes()

可用性

Flash Media Server 2

用法

my_xml.childNodes

描述

属性（只读）；一个指定的XML对象的孩子数组。数组中的每一个元素都是对一个描述了一个子节点的XML对象的引用。
这是一个只读属性，且不能被用于操作子节点。使用XML.appendChild()、XML.insertBefore()和XML.removeNode()方法来操作子

节点。对于文本节点（nodeType==3），这个属性是undefined的。

例子

下面的范例展示了如何使用XML.childNodes属性来返回一个子节点数组：

```
// 创建一个新的XML文档。
var doc = new XML();

// 创建一个根节点。
var rootNode = doc.createElement("rootNode");

// 创建三个子节点。
var oldest = doc.createElement("oldest");
var middle = doc.createElement("middle");
var youngest = doc.createElement("youngest");

// 添加rootNode作为XML文档树的根。
doc.appendChild(rootNode);

// 添加每一个子节点作为rootNode的孩子。
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// 创建一个数组并使用rootNode 来填充它。
var firstArray:Array = doc.childNodes;
trace (firstArray);
// 输出：<rootNode><oldest /><middle /><youngest /></rootNode>

// 创建另一个数组并使用子节点来填充它。
var secondArray = rootNode.childNodes;
trace(secondArray);
// 输出：<oldest />,<middle />,<youngest />
```

XML.cloneNode()

可用性

Flash Media Server 2

用法

```
my_xml.cloneNode(deep)
```

参数

deep 一个布尔值；如果设置为true，则指定的XML对象的孩子将被递归克隆；如果不希望这样，就设置为false。

返回

一个XML Node对象。

描述

方法；构造并返回一个与指定的XML对象具有相同的类型、名称、管理控制台值，以及属性的新的XML Node对象。如果
deep被设置为true，则所有的子节点都会被递归克隆，这会产生一个原始对象的文档树的精确的拷贝。

被返回的节点的克隆体不再与克隆它们的树相关联。因此，nextSibling、parentNode和previousSibling具有值null。如果deep参
数被设置为false，或者my_xml节点没有子节点了，则firstChild和lastChild也是null。

例子

下面的范例展示了如何使用XML.cloneNode()方法来创建一个节点的拷贝：

```
// 创建一个新的XML文档。
var doc = new XML();

// 创建一个根节点。
var rootNode = doc.createElement("rootNode");

// 创建三个子节点。
var oldest = doc.createElement("oldest");
var middle = doc.createElement("middle");
var youngest = doc.createElement("youngest");

// 添加rootNode作为XML文档树的根。
doc.appendChild(rootNode);

// 添加每一个子节点作为rootNode的孩子。
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// 利用cloneNode()创建middle节点的一个拷贝。
var middle2 = middle.cloneNode(false);

// 在middle和youngest节点之间把克隆节点插入rootNode。
rootNode.insertBefore(middle2, youngest);
trace(rootNode);
// 输出（带有添加的换行）：
// <rootNode>
//       <oldest />
//       <middle />
//       <middle />
//       <youngest />
// </rootNode>

// 利用cloneNode()创建rootNode的一个拷贝来示范一个deep拷贝。
var rootClone = rootNode.cloneNode(true);

// 把包含所有子节点的克隆插入到rootNode。
```

```
rootNode.appendChild(rootClone);
trace(rootNode);
// 输出（带有添加的换行）：
// <rootNode>
//     <oldest/>
//     <middle/>
//     <middle/>
//     <youngest/>
//     <rootNode>
//         <oldest/>
//         <middle/>
//         <middle/>
//         <youngest/>
//     </rootNode>
```

XML.contentType()

可用性

Flash Media Server 2

用法

Flash Media Server 2

用法

my_xml.contentType

描述

属性；当你调用XML.send()或XML.sendAndLoad()方法时发送到服务器的MIME内容类型。默认值是application/x-www-form-urlencoded，这是用于大多数HTML格式的标准MIME内容类型。

例子

下面的范例创建了一个新的XML文档并检查它的默认内容类型：

```
// 创建一个新的XML文档。
var doc = new XML();

// 跟踪默认的内容类型。
trace(doc.contentType);

// 输出：application/x-www-form-urlencoded
```

XML.createElement()

可用性

Flash Media Server 2

用法

my_xml.createElement(name)

参数

name 将要创建的XML元素的标签名。

返回

一个XML节点；一个XML元素。

描述

属性；使用在name参数中指定的名字创建一个新的XML元素。这个新元素最初是没有父、子和兄弟的。这个方法返回对描
述了这个元素的新创建的XML对象的引用。这个方法和XML.createTextNode()方法是为一个XML对象创建节点的构造器方法。

例子

下面的范例使用createElement()方法创建了三个XMLNode对象：

```
// 创建一个XML文档。
var doc = new XML();

// 使用createElement()创建三个XML节点。
var element1 = doc.createElement("element1");
var element2 = doc.createElement("element2");
var element3 = doc.createElement("element3");

// 把新节点放置到XML树中。
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

trace(doc);
// 输出：<element1><element2 /><element3 /></element1>
```

XML.createTextNode()

可用性

Flash Media Server 2

用法

my_xml.createTextNode(text)

参数

text 一个字符串；这个文本用于创建新的文本节点。

返回

一个XML节点。

描述

方法；使用指定的文本创建一个新的XML文本节点。这个新节点最初是没有父的，且文本节点不能拥有孩子或兄弟。这个
方法返回对描述了这个新文本节点的XML对象的引用。这个方法和XML.createElement()方法是为一个XML对象创建节点的构造

器方法。

例子

下面的范例使用createTextNode()方法创建了两个XML文本节点并将它们放置到现有的XML节点中：

```
// 创建一个XML文档。
var doc = new XML();

// 使用createElement()创建三个XML节点。
var element1 = doc.createElement("element1");
var element2 = doc.createElement("element2");
var element3 = doc.createElement("element3");

// 把新节点放置到XML树中。
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);
var textNode1 = doc.createTextNode("textNode1");
var textNode2 = doc.createTextNode("textNode2");

// 把新节点放置到XML树中。
element2.appendChild(textNode1);
element3.appendChild(textNode2);

trace(doc);
// 输出（标签间添加有换行）：
trace(doc);
// //      <element2>textNode1</element2>输出（标签间添加有换行）：
// <element1>
//      <element2>textNode1</element2>
//      <element3>textNode2</element3>
// </element1>
```

XML.docTypeDecl()

可用性

Flash Media Server 2

用法

my_xml.docTypeDecl

描述

属性；指定有关XML文档的DOCTYPE声明的信息。在XML文本被解析为一个XML对象后，XML对象的XML.docTypeDecl
属性被设置为XML文档的DOCTYPE声明的文本（例如，<!DOCTYPE greeting SYSTEM "hello.dtd">）。这个属性利用一个
DOCTYPE声明的字符串表示来设置，而不是使用一个XMLNode对象进行设置。

ActionScript XML解析器不是一个验证解析器。DOCTYPE声明被读取并存储在XML.docTypeDecl属性

中，但没有DTD验证
被执行。

如果在一个解析操作中没有DOCTYPE声明出现，则XML.docTypeDecl属性被设置为undefined。XML.toString()方法输出
XML.docTypeDecl的内容，这些内容是紧跟在存储于XML.xmlDecl中的XML声明之后且位于XML对象中的任何其它文本之前的
那部分内容。如果XML.docTypeDec1是undefined，则没有DOCTYPE声明。

例子

下面的范例使用XML.docTypeDecl属性来为一个XML对象设置DOCTYPE声明：

```
my_xml.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

XML.firstChild

可用性

Flash Media Server 2

用法

```
my_xml.firstChild
```

描述

属性（只读）；计算指定的XML对象并引用父节点的孩子列表中的第一个孩子。如果这个节点没有孩子，则这个属性为
null。如果这个节点是一个文本节点，则这个属性为null。这是一个只读属性，且不能被用来操作子节点；使用appendChild()、
insertBefore()，以及removeNode()方法来操作子节点。

例子

下面的范例展示了如何使用XML.firstChild来遍历一个节点的子节点：

```
// 创建一个新的XML文档。
var doc = new XML();

// 创建一个根节点。
var rootNode = doc.createElement("rootNode");

// 创建三个子节点。
var oldest = doc.createElement("oldest");
var middle = doc.createElement("middle");
var youngest = doc.createElement("youngest");

// 添加rootNode作为XML文档树的根。
// 添加每一个子节点作为rootNode的孩子。
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

rootNode.appendChild(youngest);//      的子节点。
for (var aNode = rootNode.firstChild; aNode != null; aNode = aNode.nextSibling) {
```

```
// 使用firstChild来遍历rootNode的子节点。
for (var aNode = rootNode.firstChild; aNode != null; aNode = aNode.nextSibling) {
    trace(aNode);
}

// 输出：
// <oldest />
// <middle />
// <youngest />
```

XML.getBytesLoaded()

可用性
Flash Media Server 2
用法
my_xml.getBytesLoaded()
参数
无。
返回
一个整数，指出装载的字节数。
描述
方法；返回为XML文档加载的字节数。您可以将getBytesLoaded()的值与getBytesTotal()的值进行比较，以确定XML文档已
加载的百分比。

XML.getBytesTotal()

可用性
Flash Media Server 2
用法
my_xml.getBytesTotal()
参数
无。
返回
一个整数。
描述
方法；以字节为单位返回XML文档的大小。
getNamespaceForPrefix

XML.getNamespaceForPrefix()

可用性

Flash Media Server 2

用法

my_xml.getNamespaceForPrefix(prefix)

prefix 一个字符串；这个方法针对这个前缀返回相关联的命名空间。

返回

一个字符串；与指定的前缀相关联的命名空间。

描述

方法；返回与节点的指定前缀相关联的命名空间的URI。为了确定这个URI，getPrefixForNamespace()从这个节点开始根据

需要上行搜索方法；返回与节点的指定前缀相关联的命名空间的URIXML。为了确定这个URI，getPrefixForNamespace()从这个节点开始根据

需要上行搜索XML层次结构，并返回给定前缀的第一个XML命名空间声明的命名空间URI。

如果指定的前缀没有被定义命名空间，则这个方法返回null。

如果你指定了一个空字符串（""）作为前缀，且这个节点被定义了一个默认命名空间的话（就像xmlns="http://www.example.com/"中这样），则这个方法会返回这个默认命名空间URI。

例子

下面的范例创建了一个非常简单的XML对象，并输出一个对getNamespaceForPrefix()调用的结果：

```
        + "<exu:Child id='3' />"function createXML() {
    var str = "<Outer xmlns:exu=\"http://www.example.com/util\">"          + "</Outer>";
    return new XML(str).firstChild;
}

var xml = createXML();
trace(xml.getNamespaceForPrefix("exu")); // 输出：http://www.example.com/util
trace(xml.getNamespaceForPrefix("")); // 输出：null
```

XML.getPrefixForNamespace()

可用性

Flash Media Server 2

用法

my_xml.getPrefixForNamespace(nsURI)

参数

nsURI 一个字符串；这个方法针对这个命名空间URI返回相关联的前缀。

返回

一个字符串；与指定的命名空间相关联的前缀。

描述

方法；返回与节点的指定命名空间URI相关联的前缀。为了确定这个URI，getPrefixForNamespace()从这个节点开始根据需

要上行搜索XML层次结构，并返回具有一个匹配nsURI的命名空间的第一个XML命名空间声明的前缀。

如果没有xmlns被指派给给定的URI，则这个方法返回null。如果有一个xmlns被指派给这个给定的URI，

但没有前缀与这个
指派相关联，则这个方法返回一个空字符串（""）。

例子

下面的范例创建了一个非常简单的XML对象，并输出一个对getPrefixForNamespace()方法的调用的结果。由xmlDoc变量描
述的Outer XML节点，定义了一个命名空间URI并把它指派给exu前缀。用这个定义的命名空间URI（"http://www.example.com/util"）调用这个getPrefixForNamespace()方法返回前缀exu，但用一个未定义的URI（"http://www.example.com/other"）调用这个方法会返回null。由child1变量描述的第一个exu:Child节点，也定义了一个命名空间
URI（"http://www.example.com/child"），但没有把它指派到一个前缀。在定义了但未被指派的命名空间URI上调用这个方法会
返回一个空字符串。

```
function createXML() {
    var str = "<Outer xmlns:exu=\"http://www.example.com/util\">"
        + "<exu:Child id='1' xmlns=\"http://www.example.com/child\"/>"
        + "<exu:Child id='2' />"
        + "<exu:Child id='3' />"
        + "</Outer>";
    return new XML(str).firstChild;
}

var xmlDoc = createXML();
trace(xmlDoc.getPrefixForNamespace("http://www.example.com/util")); // 输出：exu
trace(xmlDoc.getPrefixForNamespace("http://www.example.com/other")); // 输出：null

var child1 = xmlDoc.firstChild;
trace(child1.getPrefixForNamespace("http://www.example.com/child")); // 输出：[empty string]
trace(child1.getPrefixForNamespace("http://www.example.com/other")); // 输出：null
```

XML.hasChildNodes()

可用性
Flash Media Server 2
用法
my_xml.hasChildNodes()
参数
无。
返回
一个布尔值。
描述
例子方法；如果指定的XML对象有子节点，则返回 true ；否则，返回 false 。
下面的范例创建了一个新的XML 包。如果根节点有子节点，则节点循环会通过每一个子节点并显示节

点的名字和值。

```
var my_xml = new XML("<login><username>hank</username><password>rudolph</password></login>");
if (my_xml.firstChild.hasChildNodes()) {
    // if (my_xml.firstChild.hasChildNodes()) {
    //  使用firstChild来遍历rootNode的子节点。
    for (var aNode = my_xml.firstChild.firstChild; aNode != null; aNode=aNode.nextSibling) {
        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

下面的输出将出现：

```
username:hank
password:rudolph
```

XML.ignoreWhite

可用性

Flash Media Server 2

用法

```
my_xml.ignoreWhite
XML.prototype.ignoreWhite
```

参数

boolean 一个布尔值。

描述

属性；默认设置为false。当设置为true时，在解析过程中，只包含空白的文本节点将被丢弃。带有前导和尾随空白的文本节
点不受影响。

用法：可以为单个XML对象设置ignoreWhite属性，如以下代码所示：

```
my_xml.ignoreWhite = true;
```

用法：可以为XML对象设置默认ignoreWhite属性，如以下代码所示：

```
XML.prototype.ignoreWhite = true;
```

例子

下面的示例加载一个XML文件，该文件带有一个只包含空白的文本节点；foyer标签包含十四个空格字符。要运行此示例，
请创建一个名为flooring.xml的文本文件，然后将以下标签复制到其中：

```
 <house>
 <kitchen> ceramic tile </kitchen>
 <bathroom>linoleum</bathroom>
 <foyer> </foyer>
 </house>
```

var flooring:XML = new XML();Flash文档，然后将它保存到XML文件所在的同一目录。将以下代码放入主时间轴：

```
// 把ignoreWhite属性设置为true（默认值为false）
flooring.ignoreWhite = true;

// 在装载完成后，跟踪这个XML对象。
flooring.onLoad = function(success:Boolean) {
// 在装载完成后，跟踪这个XML对象。
flooring.onLoad = function(success:Boolean) {
    trace(flooring);
}

// 把这个XML装入这个flooring对象。
flooring.load("flooring.xml");

// 输出（为看得清楚添加了换行）：
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer />
</house>
```

如果您以后将flooring.ignoreWhite的设置更改为false，或者，只是整个删除该代码行，foyer标签中的十四个空格字符将被保
留：

```
...
// 把ignoreWhite属性设置为false（默认值）。
flooring.ignoreWhite = false;
...
// 输出（为看得清楚添加了换行）：
<house>
    <bathroom>linoleum</bathroom>
    <foyer> </foyer>
</house>
```

XML.insertBefore()

可用性

Flash Media Server 2

参数

childNode 要被插入的XMLNode对象。

beforeNode 这个XMLNode对象的前面就是childNode节点的插入点。

返回

无。

描述

无。方法；把一个新的子节点插入到XML对象的孩子列表中，其插入位置在beforeNode节点之前。如果beforeNode不是my_xml

描述

方法；把一个新的子节点插入到XML对象的孩子列表中，其插入位置在beforeNode节点之前。如果beforeNode不是my_xml
的一个孩子的话，则插入将失败。

例子

下面的范例是XML.cloneNode()范例的一段摘录：

```
// 利用cloneNode()创建middle节点的一个拷贝。
var middle2 = middle.cloneNode(false);

//在middle和youngest节点之间把克隆节点插入rootNode。
rootNode.insertBefore(middle2, youngest);
```

可用性

Flash Media Server 2

用法

my_xml.lastChild

用法

my_xml.lastChild

描述

属性（只读）；一个XMLNode值，它引用节点的孩子列表中的最后一个孩子。如果这个节点没有孩子，则XML.lastChild属
性为null。这个属性不能被用于操作子节点；使用appendChild()、insertBefore()，以及removeNode()方法来操作子节点。

例子

下面的范例使用XML.lastChild属性来遍历一个XMLNode对象的子节点，开始于节点的孩子列表中的最后一个项目，结束于
节点的孩子列表中的第一个孩子：

```
// 创建一个新的XML文档。
var doc = new XML();

// 创建一个根节点。
var rootNode = doc.createElement("rootNode");

// 创建三个子节点。
var oldest = doc.createElement("oldest");
var middle = doc.createElement("middle");
var youngest = doc.createElement("youngest");

// 添加rootNode作为XML文档树的根。
doc.appendChild(rootNode);
```

```
// 添加每一个子节点作为rootNode的孩子。
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// 使用lastChild来遍历rootNode的子节点。
for (var aNode = rootNode.lastChild; aNode != null; aNode = aNode.previousSibling) {
    trace(aNode);
}

/*
output:
<youngest />
<middle />
<oldest />
*/
```

下面的范例创建了一个新的XML包，并使用XML.lastChild属性来遍历根节点的子节点：

```
// 创建一个新的XML文档。
var doc = new XML("<rootNode><oldest /><middle /><youngest /></rootNode>");

var rootNode = doc.firstChild;

// 使用lastChild来遍历rootNode的子节点。
for (var aNode = rootNode.lastChild; aNode != null; aNode=aNode.previousSibling) {
    trace(aNode);
}
/*
output:
<oldest />
<middle />
*/
```

XML.load()

XML.load()

可用性

Flash Media Server 2

用法

my_xml.load(url)

描述

url 一个字符串；将要被装载的XML文档所在的位置。如果发出这个调用的SWF文件正运行于一个Web浏览器中，则url必须
与这个SWF文件位于相同的域。

返回

无。方法；从指定的URL处装载一个XML文档，并用下载的XML数据替换指定的XML对象的内容。URL的路径是相对路径，且
是利用HTTP调用的。装载过程是异步的；它不会在load()方法被执行后立刻被完成。

当load()方法被执行时，这个XML对象的属性loaded被设置为false。当XML数据完成下载时，loaded属性会被设置为true，且
onLoad事件处理器会被调用。XML数据不会被解析，直到它被完全下载。如果这个XML对象先前包含了任何XML树的话，则它
们会被丢弃。

您可以定义一个在调用XML对象的onLoad事件处理函数时执行的自定义函数。
们会被丢弃。例子
您可以定义一个在调用XML对象的onLoad事件处理函数时执行的自定义函数。
例子

下面的简单范例使用XML.load()方法：

```
// 创建一个新的XML对象。
// 把ignoreWhite属性设置为true（默认值是false）
flooring.ignoreWhite = true;

// 在装载被完成后，跟踪这个XML对象。
flooring.onLoad = function(success) {
    trace(flooring);
flooring.onLoad = function(success) {
    trace(flooring);
};

// 把这个XML装入flooring对象。
flooring.load("flooring.xml");
```

有关flooring.xml文件的内容，以及这个范例所产生的输出，参看XML.ignoreWhite的范例。

XML.loaded

可用性

Flash Media Server 2

用法

```
my_xml.loaded
```

描述

属性；一个布尔值，如果由XML.load()调用启动的文档装载过程成功完成，为true；否则，为false。

例子

下面的范例在一个简单的脚本中使用XML.loaded属性：

```
var my_xml = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success) {
    trace("success: "+success);
```

```
    trace("loaded:    "+my_xml.loaded);
    trace("status:    "+my_xml.status);
};
my_xml.load("http://www.flash-mx.com/mm/problems/products.xml");
```

当onLoad处理器被调用时，信息会写入日志文件。如果这个调用成功完成，则loaded状态true会被写入到日志文件，就像下
面的范例中展示的这样：

```
success: true
loaded:    true
status:    0
```

XML.localName

可用性

Flash Media Server 2

用法

my_xml.localName

描述

属性（只读）；XML节点的名字中的本地名字部分。这是不带命名空间前缀的元素名。例如，节点<contact:mailbox/>
bob@example.com</contact:mailbox>具有本地名“mailbox”，以及前缀“contact”，二者共同构成了完整的元素名
“contact.mailbox”。

你可以经由XML节点对象的XML.prefix属性来访问命名空间前缀。XML.nodeName属性返回完整的名字（包括前缀和本地
名）。

例子

下面的范例使用位于同一个目录中的一个SWF文件和一个XML文件。这个名为“SoapSample.xml”的XML文件包含下列内
容：

```
<?xml version="1.0"?>
 <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
 <soap:Body xmlns:w="http://www.example.com/weather">
 <w:GetTemperature>
 <w:City>San Francisco</w:City>
 </w:GetTemperature>
 </soap:Body>
 </soap:Envelope>
```

这个SWF文件的源代码中包含下列脚本（注意Output字符串的注释）：

```
var xmlDoc = new XML()
xmlDoc.ignoreWhite = true;
xmlDoc.load("SoapSample.xml")
xmlDoc.onLoad = function(success)
{
```

```
    var tempNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("w:GetTemperature localname: " + tempNode.localName); // Output: ... GetTemperature
    var soapEnvNode = xmlDoc.childNodes[0];
    trace("soap:Envelope localname: " + soapEnvNode.localName); // Output: ... Envelope
}
```

XML.namespaceURI

可用性

Flash Media Server 2

用法

my_xml.namespaceURI

描述

属性（只读）；如果XML节点有一个前缀的话，则namespaceURI就是这个前缀（这个URI）的xmlns声明的值，它最具代表
性的称谓就是命名空间URI（namespace URI）。xmlns声明位于当前节点中或是位于XML层次结构中一个更高的节点中。

如果这个XML节点没有一个前缀的话，则namespaceURI属性的值会依赖于是否有一个默认的命名空间被定义（就像在
xmlns="http://www.example.com/"中这样）。如果有一个默认的命名空间，则namespaceURI属性的值就是默认的命名空间的值。
如果没有默认的命名空间的话，则这个节点的namespaceURI属性就是一个空字符串（""）。

你可以使用XML.getNamespaceForPrefix()方法来识别与一个特定的前缀相关联的命名空间。namespaceURI属性会返回与这
个节点名相关联的前缀。

例子

下面的范例展示了namespaceURI属性是如何由前缀的使用所影响的。在这个范例中所使用的XML文件被命名为

```
<?xml version="1.0"?>
 <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
 <soap:Body xmlns:w="http://www.example.com/weather">
 <w:GetTemperature>
 <w:City>San Francisco</w:City>
 </w:GetTemperature>
 </w:GetTemperature>
 </soap:Body>
 </soap:Envelope>
```

服务器端ASC文件的源代码中包含下列脚本（注意Output字符串的注释）。对于描述了w:GetTemperature节点的tempNode，
namespaceURI的值是定义在soap:Body标签中的。对于描述了soap:Body节点的soapBodyNode，namespaceURI的值是由其上节点中
的soap前缀的定义决定的，而不是由soap:Body节点包含的w前缀的定义所决定的。

```
var xmlDoc = new XML();
xmlDoc.load("SoapSample.xml");
```

```
xmlDoc.ignoreWhite = true;
xmlDoc.onLoad = function(success:Boolean)
{
    var tempNode:XMLNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("w:GetTemperature namespaceURI: " + tempNode.namespaceURI);
        // Output: ... http://www.example.com/weather

    trace("w:GetTemperature soap namespace: " + tempNode.getNamespaceForPrefix("soap"));
        // Output: ... http://www.w3.org/2001/12/soap-envelope

    var soapBodyNode = xmlDoc.childNodes[0].childNodes[0];
    trace("soap:Envelope namespaceURI: " + soapBodyNode.namespaceURI);
        // Output: ... http://www.w3.org/2001/12/soap-envelope
}
```

下面的范例使用不带前缀的XML标签。它使用位于同一个目录中的一个SWF文件和一个XML文件。这个名为NoPrefix.xml
的XML文件包含下列标签：

```
<?xml version="1.0"?>
 <rootnode>
 <simplenode xmlns="http://www.w3.org/2001/12/soap-envelope">
 <innernode />
 </simplenode>
 </rootnode>
```

服务器端脚本文件的源代码包含下列代码（注意Output字符串的注释）。rootNode没有一个默认的命名空间，因此，它的
namespaceURI值是一个空字符串。simpleNode定义了一个默认的命名空间，因此，它的namespaceURI值是这个默认的命名空
间。innerNode没有定义一个默认的命名空间，但使用了由simpleNode定义的默认的命名空间，因此，它的namespaceURI值与
simpleNode的一样。

```
var xmlDoc = new XML()
xmlDoc.load("NoPrefix.xml");
xmlDoc.ignoreWhite = true;
xmlDoc.onLoad = function(success)
{
        // Output: [empty string]

    var simpleNode = xmlDoc.childNodes[0].childNodes[0];
    trace("simpleNode Node namespaceURI: " + simpleNode.namespaceURI);
        // Output: ... http://www.w3.org/2001/12/soap-envelope

        // Output: ... http://www.w3.org/2001/12/soap-envelope

    var innerNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("innerNode Node namespaceURI: " + innerNode.namespaceURI);
        // Output: ... http://www.w3.org/2001/12/soap-envelope
```

}

XML.nextSibling

可用性

Flash Media Server 2

用法

my_xml.nextSibling

描述描述

属性（只读）；一个XMLNode值，它引用了父节点的孩子列表中的下一个兄弟（sibling）。如果这个节点没有下一个兄弟
节点，则这个属性为null。这个属性不能被用于操作子节点；使用appendChild()、insertBefore()，以及removeNode()方法来操作子
节点。

例子

下面的范例是来自下面的范例是来自XML.firstChildXML.firstChild属性例子的一个摘录。它展示了你可以如何使用属性例子的一个摘录。它展示了你可以如何使用XML.nextSiblingXML.nextSibling属性来遍历一个属性来遍历一个XMLNodeXMLNode
对象的子节点。对象的子节点。

```
for (var aNode = rootNode.firstChild; aNode != null; aNode = aNode.nextSibling) {
    trace(aNode);
}
```

XML.nodeName

可用性

Flash Media Server 2

用法

my_xml.nodeName

描述

属性；一个字符串，描述了这个XML对象的节点名。如果这个XML对象是一个XML元素（nodeType==1），则nodeName是
描述了XML文件中的这个节点的标签名。例如，TITLE是一个HTML TITLE标签的nodeName。如果这个XML对象是一个文本节
点（nodeType==3），则nodeName是null。

例子

下面的范例创建了一个元素节点和一个文本节点，并检查了每一个节点的节点名：

```
// 创建一个XML文档。
var doc = new XML();

// 利用createElement()创建一个XML节点。
var myNode = doc.createElement("rootNode");
```

```
// 把这个新节点放置到这个XML树中。
doc.appendChild(myNode);

// 利用createTextNode()创建一个XML文本节点。
var myTextNode = doc.createTextNode("textNode");

// 把这个新节点放置到这个XML树中。
myNode.appendChild(myTextNode);

trace(myNode.nodeName);
trace(myTextNode.nodeName);

/*
output:
rootNode
null
*/
```

下面的范例创建了一个XML包。如果这个根节点有子节点的话，则这个代码会遍历每一个子节点来显示节点的名字和值。

把下面的ActionScript添加到你的ASC文件：

```
var my_xml = new XML("<login><username>hank</username><password>rudolph</password></login>");
if (my_xml.firstChild.hasChildNodes()) {
    // 使用firstChild来遍历rootNode的子节点。
    for (var aNode = my_xml.firstChild.firstChild; aNode != null; aNode=aNode.nextSibling) {
        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

下面的节点名会出现：

```
username:hank
password:rudolph
```

XML.nodeType

可用性

Flash Media Server 2

用法

```
my_xml.nodeType
```

描述nodeType是一个来自W3C DOM Level 1 Recommendation（http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html）中的NodeType枚举的数字值。下面的表格列出了这些值：　　。

整数值	定义的常量
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

在Flash Player中，内建的XML类只支持（ELEMENT_NODE）和（TEXT_NODE）。
例子
下面的范例创建了一个元素节点和一个文本节点，并检查了每一个节点的节点类型：

```
// 创建一个XML文档。
var doc = new XML();

// 利用createElement()创建一个XML节点。
var myNode = doc.createElement("rootNode");

// 把这个新节点放置到这个XML树中。
doc.appendChild(myNode);

// 利用createTextNode()创建一个XML文本节点。
var myTextNode = doc.createTextNode("textNode");

// 把这个新节点放置到这个XML树中。
myNode.appendChild(myTextNode);

trace(myNode.nodeType);/*
output:trace(myTextNode.nodeType);

1
3
*/
```

XML.nodeValue

XML.nodeValue

可用性

Flash Media Server 2

用法

my_xml.nodeValue

描述

属性；XML对象的节点值。如果这个XML对象是一个文本节点，则nodeType是，且nodeValue是这个节点的文本。如果这
个XML对象是一个XML元素（nodeType为），则nodeValue是null且是只读的。

例子

下面的范例创建了一个元素节点和一个文本节点，并检查了每一个节点的节点类型：

```
// 创建一个XML文档。
var doc = new XML();

// 利用createElement()创建一个XML节点。
var myNode = doc.createElement("rootNode");

// 把这个新节点放置到这个XML树中。
doc.appendChild(myNode);

// 利用createTextNode()创建一个XML文本节点。
var myTextNode = doc.createTextNode("textNode");

// 把这个新节点放置到这个XML树中。
myNode.appendChild(myTextNode);

trace(myNode.nodeValue);
trace(myTextNode.nodeValue);

/*
output:
null
myTextNode
*/
```

下面的范例创建并解析一个XML包。这个代码利用firstChild属性和firstChild.nodeValue遍历每一个子节点并显示节点值。

```
var my_xml = new
XML("<login><username>morton</username><password>good&amp;evil</password></login>");
trace("using firstChild:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
trace("");       trace("\t"+my_xml.firstChild.childNodes[i].firstChild);
trace("using firstChild.nodeValue:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
      trace("\t"+my_xml.firstChild.childNodes[i].firstChild.nodeValue);
}
```

下面的信息被写入日志文件：
using firstChild:
下面的信息被写入日志文件：
using firstChild:
    morton
    good&evil

using firstChild.nodeValue:
    morton
    good&evil

XML.onData

可用性
Flash Media Server 2
用法
my_xml.onData = function(src) {}
参数
src 一个字符串或undefined；服务器发送的原始数据，通常为XML格式。
返回
无。
描述
事件处理器；当XML文本从服务器完全下载后，或当从服务器下载XML文本的过程中出现错误时调用。在解析XML之前调
用此处理函数，因此你可以用它调用一个自定义解析例程，从而不必使用Flash XML解析程序。src参数是一个包含从服务器下
载的XML文本的字符串，除非下载过程出现错误。在出现错误的情况下，src参数为undefined。
默认情况下，XML.onData事件处理函数调用XML.onLoad。你可以用自定义行为覆盖XML.onData事件处理函数，但是，除
非你在XML.onData实现过程中调用XML.onLoad，否则不会对其进行调用。
例子
下面的范例展示了默认情况下XML.onData事件处理器看起来是怎样的：

```
XML.prototype.onData = function (src) {
  if (src == undefined) {
    this.onLoad(false);
  } else {
    this.parseXML(src);
    this.loaded = true;
    this.onLoad(true);
  }
}
```

你可以覆盖XML.onData事件处理函数，以截获XML文本而不对它进行解析。

XML.onHTTPStatus

可用性

Flash Media Server 2

用法

myXML.onHTTPStatus(httpStatus){}

参数

httpStatus 由服务器返回的HTTP状态代码。例如，值表示服务器尚未找到任何与请求的URI匹配的内容。在
ftp://ftp.isi.edu/in-notes/rfc2616.txt处的HTTP规范的.4和.5节中，可以找到HTTP状态代码。

描述

事件处理器；当Flash Media Server接收来自服务器的HTTP状态代码时调用。使用此处理函数，你可以捕获HTTP状态代码
并根据该状态代码进行操作。

onHTTPStatus处理函数在onData前调用，它在加载失败时触发对值为undefined的onLoad的调用。要特别注意的是，在触发
onHTTPStatus之后，随后始终触发onData，而不管是否会覆盖onHTTPStatus。若要充分地使用onHTTPStatus处理函数，请编写
结果函数以捕获onHTTPStatus调用的结果；然后可以在onData或onLoad处理函数中使用该结果。如果未调用onHTTPStatus，则
表示FMS没有尝试发出URL请求。

如果Flash Media Server无法从服务器获取状态代码或者如果它无法与服务器进行通讯，则将默认值传递到你的ActionScript
代码。

例子

下面的范例展示了如何使用onHTTPStatus来帮助调试，此示例收集HTTP状态代码并将它们的值和类型分配给XML对象的
实例（请注意，本示例在运行时创建实例成员this.httpStatus和this.httpStatusType）。onData方法使用这些实例成员跟踪有关在调
试时非常有用的HTTP响应的信息。

```
var myXml:XML = new XML();

myXml.onHTTPStatus = function(httpStatus:Number) {
    this.httpStatus = httpStatus;
    if(httpStatus < 100) {
        this.httpStatusType = "flashError";
    }
    else if(httpStatus < 200) {
        this.httpStatusType = "informational";
    }
    else if(httpStatus < 300) {
        this.httpStatusType = "successful";
    }
    else if(httpStatus < 400) {
```

```
            this.httpStatusType = "redirection";
        }
        else if(httpStatus < 500) {
            this.httpStatusType = "clientError";
        }
        else if(httpStatus < 600) {
            this.httpStatusType = "serverError";
        }
    }

    myXml.onData = function(src:String) {
        trace(">> " + this.httpStatusType + ": " + this.httpStatus);
        if(src != undefined) {
            this.parseXML(src);
            this.loaded = true;
            this.onLoad(true);
        }
        else {
            this.onLoad(false);
        }
    }

    myXml.onLoad = function(success:Boolean) {
    }

    myXml.load("http://weblogs.macromedia.com/mxna/xml/rss.cfm?query=byMostRecent&languages=1");
```

XML.onLoad

可用性

Flash Media Server 2

用法

my_xml.onLoad = function (success) {}

参数

success 一个布尔值，如果使用XML.load()或XML.sendAndLoad()操作成功加载了XML对象，则值为true；否则为false。

返回

无。

描述

事件处理器；接收到来自服务器的XML文档时由Flash Media Server调用。如果成功接收了XML文档，则success参数为
true。如果未收到该文档，或从服务器接收响应时出现错误，则success参数为false。默认情况下，此方法的实现不处于活动状
态。若要覆盖默认实现，必须指定一个包含自定义动作的函数。

例子

下面的示例包括用于简单电子商务店面应用程序的ActionScript。sendAndLoad()方法传输一个包含用户名和密码的XML元
素，并使用var my_xml = new XML(login_str);

```
var myLoginReply_xml = new XML();
myLoginReply_xml.ignoreWhite = true;
myLoginReply_xml.onLoad = function(success){
    if (success) {
        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
    if (success) {
        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
            (myLoginReply_xml.firstChild.attributes.success == "true")) {
            gotoAndStop("loggedIn");
        } else {
            gotoAndStop("loginFailed");
        }
    } else {
        gotoAndStop("connectionFailed");
    }
};
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm", myLoginReply_xml);
```

XML.parentNode

可用性

Flash Media Server 2

用法

my_xml.parentNode

不能被用于操作子节点；要操作子节点，使用appendChild()、insertBefore()，以及removeNode()方法。

例子 XMLNode值，引用指定的XML对象的父节点；如果这个节点没有父节点的话，则返回null。这个属性

下面的范例创建了一个XML包，并把username节点的父节点写入到日志文件中：

```
var my_xml = new
XML("<login><username>morton</username><password>good&amp;evil</password></login>");

// 第一个孩子是这个<login />节点。
var rootNode = my_xml.firstChild;
// 第一个孩子是这个<login />节点。
var rootNode = my_xml.firstChild;

// 根的第一个孩子是这个<username />节点。
var targetNode = rootNode.firstChild;
trace("the parent node of '"+targetNode.nodeName+"' is: "+targetNode.parentNode.nodeName);
trace("contents of the parent node are:\n"+targetNode.parentNode);
```

```
/* 输出（为了清楚起见添加了换行）：

'username'的父节点是：login
这个父节点的内容是：
<login>
   <username>morton</username>
   <password>good&amp;amp;evil</password>
</login>

*/
```

可用性

Flash Media Server 2

用法

my_xml.parseXML(source)

参数

source 一个字符串，表示要解析并传递到指定的XML对象的XML文本。

返回

source 无。 一个字符串，表示要解析并传递到指定的XML对象的XML文本。

返回

无。

描述

方法；解析source参数中指定的XML文本，并使用结果XML树填充指定的XML对象。XML对象中任何现有的树将被放弃。

例子

下面的示例创建并分析XML包：

```
var xml_str = "<state name=\"California\"><city>San Francisco</city></state>"

// 在XML构造器中定义XML源：
var my1_xml = new XML(xml_str);
trace(my1_xml.firstChild.attributes.name); // 输出：California

// 利用XML.parseXML方法定义XML源：
var my2_xml = new XML();
my2_xml.parseXML(xml_str);
trace(my2_xml.firstChild.attributes.name);
```

XML.prefix

可用性

Flash Media Server 2

用法

Flash Media Server 2

用法

my_xml.prefix

描述

属性（只读）；XML节点名的前缀部分。例如，在节点
<contact:mailbox/>bob@example.com</contact:mailbox>中前缀
“contact”和本地名“mailbox”共同组成了完整的元素名contact.mailbox。

例子

一个目录包含一个服务器端脚本文件和一个XML文件。这个名为“SoapSample.xml”的XML文件包含下列内容：

```
<?xml version="1.0"?>
 <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
 <soap:Body xmlns:w="http://www.example.com/weather">
 <w:GetTemperature>
 <w:City>San Francisco</w:City>
 </w:GetTemperature>
 </soap:Body>
 </soap:Envelope>
```

服务器端脚本文件的源代码包含下列代码（注意Output字符串的注释）：

```
var xmlDoc = new XML();
xmlDoc.ignoreWhite = true;xmlDoc.ignoreWhite = true;
xmlDoc.load("SoapSample.xml");
xmlDoc.onLoad = function(success)
{
    var tempNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("w:GetTemperature prefix: " + tempNode.prefix); // Output: ... w
    var soapEnvNode = xmlDoc.childNodes[0];      var soapEnvNode = xmlDoc.childNodes[0];
    trace("soap:Envelope prefix: " + soapEnvNode.prefix); // Output: ... soap      trace("soap:Envelope
prefix: " + soapEnvNode.prefix); // Output: ... soap
}
```

XML.previousSibling

可用性

Flash Media Server 2

用法

my_xml.previousSibling

描述

属性（只读）；一个XMLNode值，引用父节点的孩子列表中的上一个兄弟。如果这个节点没有上一个兄弟节点，则这个属
性拥有值null。这个属性不能被用来操作子节点；要操作子节点，使用XML.appendChild()、
XML.insertBefore()，以及

XML.removeNode()方法来代替。

例子

下面的范例是XML.lastChild属性范例的一个摘录，向你展示了你可以如何使用XML.previousSibling属性来遍历一个
XMLNode对象的子节点：

```
for (var aNode = rootNode.lastChild; aNode != null; aNode = aNode.previousSibling) {
    trace(aNode);
}
```

XML.removeNode()

可用性

Flash Media Server 2

用法

my_xml.removeNode()

参数

无。

返回

无。

描述

方法；从其父中移除指定的XML对象并删除这个节点的所有子代。

例子

下面的范例创建了一个XML包，然后删除指定的XML对象及其子代节点：

```
var xml_str = "<state name=\"California\"><city>San Francisco</city></state>";

var my_xml = new XML(xml_str);
var cityNode = my_xml.firstChild.firstChild;
trace("before XML.removeNode():\n"+my_xml);
cityNode.removeNode();
trace("");
trace("after XML.removeNode():\n"+my_xml);
/* 输出（为了清楚起见添加了换行）：


before XML.removeNode():
<state name="California">
   <city>San Francisco</city>
</state>

after XML.removeNode():
after XML.removeNode():
<state name="California" />

*/
```

XML.send()

可用性

Flash Media Server 2

用法

my_xml.send(url, [fileObj])

参数

url 一个字符串；指定的XML对象的目标URL。

fileObj 文件；（可选的）一个File对象，它不是只读的，响应可以被写入其中。如果这个File对象不是打开的，则Flash
Media Server会打开这个文件，写入到这个文件中，然后关闭这个文件。如果这个File对象是打开的，则Flash Media Server会写
入这个文件并保持文件的打开状态。

返回

无。

描述

GET方法。方法；将指定的XML对象编码为XML文档并在浏览器中利用POST方法将其发送到指定的URL。Flash测试环境中只能使用

例子

下面的范例定义了一个XML包并为这个XML对象设置了内容类型。然后，数据被发送到一个服务器并把结果写入到一个
File对象中。

File对象中。

```
var my_xml = new XML("<highscore><name>Ernie</name><score>13045</score></highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", myFile);
```

XML.sendAndLoad()

可用性

Flash Media Server 2

用法

my_xml.sendAndLoad(url, targetXMLobject)

参数

url 一个字符串；指定的XML对象的目标URL。如果发出此调用的SWF文件正在Web浏览器上运行，则url必须与SWF文件位
于同一个域中。

targetXMLobject 一个用XML构造器方法创建的目标XML对象，它将接收来自服务器的返回信息。

返回

无。

描述

将指定的XML对象编码为XML文档，使用POST方法将其发送到指定的URL，下载服务器的响应，并将其加载到参数中指
定的targetXMLobject中。服务器响应加载的方式与XML.load()方法使用的方式相同。

执行sendAndLoad()方法时，XML对象的loaded属性被设置为false。在XML数据下载完毕后，如果成功加载数据，则loaded属
性被设置为true，并调用onLoad事件处理函数。直到XML数据完全下载后，才开始分析。如果该XML对象以前包含任何XML
树，这些树将被放弃。

例子

```
var login_str = "<login username=\""+username_txt.text+"\" password=\""+password_txt.text+"\"
/>";ActionScript。XML.sendAndLoad()方法传输一个包含用户名和密码的
var my_xml = new XML(login_str);
var myLoginReply_xml = new XML();
myLoginReply_xml.ignoreWhite = true;
myLoginReply_xml.onLoad = myOnLoad;
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm", myLoginReply_xml);
function myOnLoad(success) {
    if (success) {my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm",
myLoginReply_xml);
function myOnLoad(success) {
    if (success) {
        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
            (myLoginReply_xml.firstChild.attributes.success == "true")) {
            gotoAndStop("loggedIn");
        } else {
            gotoAndStop("loginFailed");
        }
    } else {
        gotoAndStop("connectionFailed");
    }
}
```

XML.status

可用性

Flash Media Server 2

用法

my_xml.status

描述

属性；自动设置并返回一个数值，该数值指示XML文档是否被成功地解析为XML对象。以下是数字状态代码和说明：

0 没有错误；成功地完成了分析。

-2 一个CDATA 部分没有正确结束。

-3 XML 声明没有正确结束。

-4 DOCTYPE 声明没有正确结束。

-5 一个注释没有正确结束。

-6 一个XML 元素有格式错误。

-7 内存不足。

-8 一个属性值没有正确结束。

-9 一个开始标签和结束标签不匹配。

-10 遇到一个没有匹配的开始标签的结束标签。

例子

下面的示例将一个XML包加载到SWF文件中。这时会显示一条状态消息，内容视XML的加载和解析成功与否而定。请将以

下ActionScript添加到你的FLA或AS文件：

```
var my_xml = new XML();
my_xml.onLoad = function(success) {
    if (success) {
        if (my_xml.status == 0) {
            trace("XML was loaded and parsed successfully");
        } else {
            trace("XML was loaded successfully, but was unable to be parsed.");
        }
        var errorMessage;
        switch (my_xml.status) {
        case 0 :
            errorMessage = "No error; parse was completed successfully.";
            break;
        case -2 :
            errorMessage = "A CDATA section was not properly terminated.";
            break;
        case -3 :
            errorMessage = "The XML declaration was not properly terminated.";
            break;
        case -4 :
            errorMessage = "The DOCTYPE declaration was not properly terminated.";
            break;
        case -5 :
            errorMessage = "A comment was not properly terminated.";
            break;
        case -6 :
            errorMessage = "An XML element was malformed.";
            errorMessage = "Out of memory.";                    break;
            break;
        case -8 :
            errorMessage = "An attribute value was not properly terminated.";
            break;
        case -9 :
            errorMessage = "A start tag was not matched with an end tag.";
```

```
                break;              case -9 :
                errorMessage = "A start tag was not matched with an end tag.";
                break;
            case -10 :
                errorMessage = "An end tag was encountered without a matching
                    start tag.";
                break;
            default :
                errorMessage = "An unknown error has occurred.";
                break;
            }
            trace("status: "+my_xml.status+" ("+errorMessage+")");
        }
    };
    my_xml.load("http://www.flash-mx.com/mm/badxml.xml");
```

XML.toString()

XML.toString()

可用性

Flash Media Server 2

用法

my_xml.toString()

参数

无。

返回

一个字符串。

描述

方法；计算这个指定的XML对象，构造出这个XML结构的一个文本表示，包括节点、孩子、属性，并把结果作为一个字符
串返回。

对于顶级XML对象（即那些由构造器创建的），XML.toString()方法输出文档的XML声明（存储在XML.xmlDecl属性中），
后跟文档的DOCTYPE声明（存储在XML.docTypeDecl属性中），后跟这个对象中的所有XML节点的文本表示。如果
XML.xmlDecl属性是undefined的话，则XML声明不会输出。如果XML.docTypeDecl属性是undefined的话，则DOCTYPE声明不会
输出。

例子

下面的XML.toString()方法的范例把<h1>test</h1>发送到日志文件：

```
var node = new XML("<h1>test</h1>");
trace(node.toString());
```

XML.xmlDecl

可用性

Flash Media Server 2

用法

my_xml.xmlDecl

描述

属性；一个字符串，指定一个文档的XML声明的相关信息。在这个XML文档被解析成一个XML对象后，这个属性会被设置
成这个文档的XML声明的文本。这个属性是利用XML声明的一个字符串表示来设置的，而不是使用一个XMLNode对象进行设
置的。如果在解析过程中没有遭遇到XML声明，则这个属性会被设置为undefined.XML。XML.toString()方法在输出XML对象中
任何其它的文本之前首先会输出XML.xmlDecl属性的内容。如果XML.xmlDecl属性包含undefined类型，则没有XML声明会被输
出。

例子

下面的范例装载一个XML文件并输出有关这个文件的信息：

```
var my_xml = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success)
{
        if (success)
        {
            trace("xmlDecl: " + my_xml.xmlDecl);
            trace("contentType: " + my_xml.contentType);
            trace("docTypeDecl: " + my_xml.docTypeDecl);
            trace("packet: " + my_xml.toString());
        }
        else
        {
            trace("Unable to load remote XML.");
        }
}
my_xml.load("http://foo.com/crossdomain.xml");
```

XMLSocket类

可用性

Flash Media Server 2

描述

XMLSocket类可实现客户机端套接字，这使得Flash Media Server可以与由IP地址或域名标识的服务器计算机进行通讯。对
于要求滞后时间较短的客户机/服务器应用程序，如实时聊天系统，XMLSocket类非常有用。传统的基于HTTP的聊天解决方案
频繁轮询服务器，并使用HTTP请求来下载新的消息。与此相对照，XMLSocket聊天解决方案保持与服务器的开放连接，这一连
接允许服务器即时发送传入的消息，而无需客户机发出请求。

注意：你也可以使用XMLSocket类来创建一个XMLStream对象。

若要使用XMLSocket类，服务器计算机必须运行可识别XMLSocket类使用的协议的守护程序。下面的列表说明了该协议：

XML消息通过全双工TCP/IP流套接字连接发送。

每个XML消息都是一个完整的XML文档，以一个零（）字节结束。

通过一个XMLSocket连接发送和接收的XML消息的数量没有限制。

XMLSocket对象连接到服务器的方式和位置受以下限制：

XMLSocket.connect()方法只能连接到端口号大于或等于的TCP端口。这种限制的一个后果是，与XMLSocket对象

通讯的服务器守护程序也必须分配到端口号大于或等于的端口。端口号小于的端口通常用于系统服务（如

FTP、Telnet和HTTP），因此，出于安全方面的考虑，禁止XMLSocket对象使用这些端口。这种端口号方面的限制可

以减少不恰当地访问和滥用这些资源的可能性。

要使用XMLSocket类的方法，你必须首先使用构造器new XMLSocket来创建一个XMLSocket对象。

XMLSocket类的方法汇总

方法	描述
XMLSocket.close()	关闭一个打开的套接字连接。
XMLSocket.connect()	建立一个至指定的服务器的连接。
XMLSocket.send()	把一个XML对象发送到服务器。

XMLSocket类的事件处理器汇总

属性	描述
XMLSocket.onClose	当一个XMLSocket连接被关闭时调用。
XMLSocket.onConnect	当一个由XMLSocket.connect()发起的连接请求成功或失败时由Flash Media Server调

用。

XMLSocket.onData	当一个XML消息从服务器被下载时调用。
XMLSocket.onXML	当一个XML对象自服务器抵达时调用。

XMLSocket类的构造器

可用性

Flash Communication Server MX 1.5

用法

new XMLSocket(streamOrFlash)

参数

streamOrFlash 一个字符串，指出这个对象是一个XMLSocket对象还是一个XMLStreams对象。这个参数可以是下列两个值
中的一个："flash"或"stream"。

返回

对一个XMLSocket对象或一个XMLStreams对象的引用。

描述

构造器；创建一个新的XMLSocket对象（"flash"）或一个新的XMLStreams对象（"stream"）。XMLSocket和XMLStreams对
象最初并不连接到任何服务器。你必须调用XMLSocket.connect()来把这个对象连接到一个服务器。

例子

下面的范例创建了一个XMLSocket对象：

```
var socket = new XMLSocket("flash");
```

下面的范例创建了一个XMLStreams对象：

```
var stream = new XMLSocket("stream");
```

XMLSocket.close()

可用性

Flash Media Server 2

用法

myXMLSocket.close()

参数

无。

返回

无。

描述

方法；关闭由这个XMLSocket对象指定的连接。

例子

下面的范例创建了一个XMLSocket对象，试图连接到服务器，然后关闭这个连接：

```
var socket = new XMLSocket();
socket.connect(null, 2000);
socket.close();
```

XMLSocket.connect()

可用性

Flash Media Server 2

用法

```
myXMLSocket.connect(host, port)
```

参数

host 一个字符串；一个完全限定DNS域名，或一个aaa.bbb.ccc.ddd形式的IP地址。也可指定null以连接到本地主机。

port 一个数字；用于建立连接的主机上的TCP端口号。该端口号必须为或更高。

返回

一个布尔值；如果成功为true，否则为false。

描述

方法；使用指定的TCP端口（必须为或更高的端口）建立一个到指定Internet主机的连接，并根据是否成功建立了连接
返回true或false。如果您不知道Internet主机的端口号，请与您的网络管理员联系。

如果您为host参数指定null，则会与本地主机连接。

服务器端ActionScript XMLSocket.connect()方法可以连接到不与SWF文件同域的计算机。

注意：客户机端ActionScript XMLSocket.connect()方法有限制。

如果XMLSocket.connect()返回true值，则表示连接过程的初始阶段是成功的；随后将调用XMLSocket.onConnect方法以确定
最终连接是成功还是失败。如果XMLSocket.connect()返回false，则无法建立连接。

例子

下面的范例使用XMLSocket.connect()来连接到本地主机：

```
var socket = new XMLSocket()
socket.onConnect = function (success) {
    if (success) {
        trace ("Connection succeeded!")
  } else {
        trace ("Connection failed!")
    }
}
if (!socket.connect(null, 2000)) {
    trace ("Connection failed!")
}
```

XMLSocket.onClose

可用性

Flash Media Server 2

用法

myXMLSocket.onClose = function() {}

参数

无。

返回

无。

描述

事件处理器；仅在服务器关闭某个打开的连接时调用。此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定

例子

下面的示例在服务器关闭一个打开的连接时执行trace语句：

```
var socket = new XMLSocket();
socket.connect(null, 2000);

socket.connect(null, 2000);socket.onClose = function () {

socket.onClose = function () {       trace("Connection to server lost.");

    trace("Connection to server lost.");
}
```

XMLSocket.onConnect

可用性

Flash Media Server 2

用法

myXMLSocket.onConnect = function(success) {}

参数

success 一个布尔值，指示是否成功建立了套接字连接（true或false）。

返回

无。

描述

事件处理器；在通过XMLSocket.connect()启动的连接请求成功或失败后，由Flash Media Server调用。如果连接成功，则
success参数为true；否则success参数为false。

此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含自定义动作的函数。

例子

下面的范例定义了一个用于onConnect处理器的函数：

```
socket = new XMLSocket();
socket.onConnect = myOnConnect;
```

```
socket.connect(null,2000);

function myOnConnect(success) {
    if (success) {
        trace("Connection success")
    } else {
        trace("Connection failed")
    }
}
```

XMLSocket.onData

可用性

Flash Media Server 2

用法

myXMLSocket.onData = function(src) {}

参数

src 一个字符串，包含服务器发送的数据。

返回

无。

描述

事件处理器；当某个消息已从服务器下载并以零（）字节结束时调用。您可以覆盖XMLSocket.onData以截获服务器发送
的数据，而不将其分析为XML。如果您传输的是任意格式的数据包，而且希望在数据到达时直接操纵这些数据，而不让Flash
Media Server将数据分析为XML，则此方法很有用。

默认情况下，XMLSocket.onData方法调用XMLSocket.onXML方法。如果您用自定义行为覆盖XMLSocket.onData，除非您在
XMLSocket.onData实现过程中调用XMLSocket.onXML，否则不会对其进行调用。

例子

在此示例中，src参数是一个字符串，其中包含从服务器下载的XML文本。零（）字节终止符不包含在该字符串中。

```
XMLSocket.prototype.onData = function (src) {
  this.onXML(new XML(src));
}
```

XMLSocket.onXML

可用性

Flash Media Server 2

用法

myXMLSocket.onXML = function(object) {}

参数

object 一个XML对象，它包含从服务器上接收到的经过分析的XML文档。

返回

无。

描述

事件处理器；在包含XML文档的指定XML对象通过打开的XMLSocket连接到达时，由Flash Media Server调用。XMLSocket
连接可用于在客户机与服务器之间传输无限量的XML文档。每个文档都以零（）字节结束。当Flash Media Server收到字节
批经过分析的XML都将被视为单个XML文档，并传递给onXML方法。

此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含您定义的动作的函数。

例子

下面的函数在一个简单的聊天应用程序中覆盖onXML方法的默认实现。函数myOnXML指示该聊天应用程序识别一个XML
元素MESSAGE，该元素的格式如下所示：

<MESSAGE USER="John" TEXT="Hello, my name is John!" />.

<MESSAGE USER="John" TEXT="Hello, my name is John!" />.

```
var socket = new XMLSocket();
```

假定下面的函数displayMessage()是一个用户定义的函数，该函数显示用户收到的消息：

```
socket.onXML = function (doc) {
    var e = doc.firstChild;
    if (e != null && e.nodeName == "MESSAGE") {
        displayMessage(e.attributes.user, e.attributes.text);
    }
}
```

XMLSocket.send()

可用性

Flash Media Server 2

用法

myXMLSocket.send(object)

参数

object 一个要传输到服务器的XML对象或其它数据。

返回

无。

描述

方法；将在object参数中指定的XML对象或数据转换成一个字符串，并将其传输到服务器，后面跟有一个零（）字节。如
果object是一个XML对象，则该字符串是此XML对象的XML文本表示形式。发送操作是异步的；它将立即返回，但数据可能会
以后传输。XMLSocket.send()方法不返回指示数据是否成功传输的值。

如果myXMLSocket对象未连接到服务器（使用XMLSocket.connect()），则XMLSocket.send()操作将失

败。

例子

下面的示例说明如何指定用户名和密码，以将XML对象my_xml发送到服务器：

```
var myXMLSocket = new XMLSocket();
var my_xml = new XML();
var myLogin = my_xml.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
my_xml.appendChild(myLogin);
myXMLSocket.send(my_xml);
```

XMLStreams类

可用性

Flash Media Server 2

描述

XMLStreams类是XMLSocket类的一个变种-它拥有与XMLSocket类完全相同的方法、属性和事件，但它是以片段来传输和
接收数据的。要创建一个XMLStreams对象，使用XMLSocket构造器并传输"stream"作为参数。

Flash Media Server可以以流格式传输XML数据（例如，当一个Jabber服务器或IM应用程序需要时）。流XML数据经由一个
普通的XMLSocket连接传递，但它是以一个stream:stream标签开始的，包含XML内容片断，并以一个/stream:stream标签结束。

任何时候当其接收它们时，onData事件被调用并返回完整的XML标签。/stream:stream标签关闭流。任何时候当一个完整的
标签被这个流接收时，都会有一个对onData的异步调用。

注意：作为一种安全防范，如果在关闭XML标签之前到达了K数据的话，则引入的数据将被丢弃。这不应成为一种通常的
担忧-至少一个完整的XML标签应该在KB内到达。

例子

如果你希望你的FMS应用程序与一个使用XML流的Jabber服务器通讯的话，创建一个XMLStreams对象。XMLStreams对象
连接到一个远端XML流服务器，当完整的XML部分出现在流中时，onData事件会被调用。

```
myXMLStreams = new XMLSocket("stream");
```

附录A 服务器端信息对象

Application、NetConnection和Stream类提供了一个onStatus事件处理器，这个事件处理器使用一个信息对象来提供信息、状
态或错误消息。要对这个事件处理器进行响应，你必须创建一个函数来处理这个信息对象，并且，你必须知道返回的信息对象
的格式和内容。
态或错误消息。要对这个事件处理器进行响应，你必须创建一个函数来处理这个信息对象，并且，你必须知道返回的信息对象
的格式和内容。

你可以在你的main.asc文件的顶端定义下面的全局函数以显示针对你传递给函数的参数的所有的状态消息。你只需把这个
代码放置到main.asc文件中一次。

```
function showStatusForClass(){
  for (var i=0;i<arguments.length;i++){
    arguments[i].prototype.onStatus = function(infoObj){
    trace(infoObj.code + " (level:" + infoObj.level + ")");
  }
}
showStatusForClass(NetConnection, Stream);
```

一个信息对象有下列属性：级、代码、描述，以及细节。所有的信息对象都有级和代码属性，但只有某些信息对象有描述
和/或细节属性。下面的表格列出了每一个信息对象的代码和级属性，及其含义。

Application信息对象

下面的表格列出了Application对象的信息对象。

代码	级	含义
NetConnection.Call.Failed	Error	NetConnection.call方法不能调用服务器端方法或命令。这个信息对象 还有一个description属性，它是一个字符串，指出了失败的原因。
NetConnection.Connect.AppShutdown	Error	应用程序已经被关闭（例如，如果应用程序溢出内存，并且必须被关 闭以防止服务器崩溃），或是服务器已经被关闭。
NetConnection.call.BadVersion	Error	在NetConnection.connect方法中指定的URI没有指定“rtmp”作为协 议。当连接到Flash Communication Server时，必须指定“rtmp”作为协议。
NetConnection.Connect.Closed	Status	连接被成功的关闭。
NetConnection.Connect.Failed	Error	连接尝试失败。
NetConnection.Connect.Rejected	Error	客户机没有权限来连接到应用程序，或是应用程序期待一个与所传递 来的不同的参数。这个信息对象也有一个application属性，它包含了application.rejectConnection服务器端方法返回的值。
NetConnection.Connect.Success	Status	连接尝试成功。

Stream信息对象

Stream类的信息对象类似于客户机端NetStream类的信息对象。

代码	级	含义
NetStream.Clear.Success	Status	一个记录的流被成功的删除。
NetStream.Clear.Failed	Error	一个记录的流删除失败。
NetStream.Publish.Start	Status	发布尝试成功。
NetStream.Publish.BadName	Error	试图发布一个已经被其他人发布的流。
NetStream.Failed	Error	使用一个Stream方法的尝试失败。这个信息对象还有一个description属性，它是一个字符串，指出了失败的原因。
NetStream.Unpublish.Success	Status	一个不发布尝试成功。
NetStream.Record.Start	Status	开始记录。
NetStream.Record.NoAccess	Error	试图记录一个只读流。
NetStream.Record.Stop	Status	停止记录。
NetStream.Record.Failed	Error	记录一个流的尝试失败。
NetStream.Play.InsufficientBW	Warning	默认情况下，流出信息以秒为间隔被监视；如果数据延迟，则客户机会在秒内被通知。取样的间隔时间可以在Application.xml文件的Client标签中进行设置。
NetStream.Play.Start	Status	开始播放。这个信息对象还有一个details属性，这个属性是一个字符串，提供了被播放的流的名字。这在多播放时是有用的，当从播放列表中的一个元素切换到另一个元素时，details属性展示了流的名字。
NetStream.Play.StreamNotFound	Error	试图播放一个不存在的流。
NetStream.Play.Stop	Status	停止播放。
NetStream.Play.Failed	Error	回放一个流的尝试失败。这个信息对象还有一个details属性，这个属性是一个字符串，提供了被播放的流的名字。这在多播放时是有用的，当从播放列表中的一个元素切换到另一个元素时，details属性展示了流的名字。
NetStream.Play.Reset	Status	播放列表被重置。
NetStream.Play.PublishNotify	Status	最初发布一个流成功。这个消息被发送到所有的订阅者。
NetStream.Play.UnpublishNotify	Status	自一个流的不发布是成功的。这个消息被发送给所有的订阅者。