Bilibili　播放器特殊弹幕语言

注：

本文档整理自　wiki.bilibili.tv .

不定期追踪更新　.暂不完全整理　.

今后可能会加入　wiki　没有的内容　.

资料性质 .

为编辑简便未能按照学习规律整理　,望见谅 .

请用户根据自身情况选择阅读　/ 学习顺序 .

有整理错误请及时指出　.

整体范例 / 错误代码等请到　wiki.bilibili.tv　　自行查阅

联系方式　:enex@9ch.name

更新历史：

[11.08.29]　增加 ECMAScript　　部分

Bilibili　播放器特殊弹幕语言

1.Player -　播放器控制相关

1.1 play

public function play():void
开始播放媒体文件

1.2 pause

public function pause():void
暂停视频流的回放。如果视频已经暂停，则调用此方法将不会执行任何操作。要在暂时视频后恢复播放，请调用 play()　。

1.3 seek

public function seek(offset:Number):void
搜索与指定位置最接近的关键帧（在视频行业中也称为　　I　帧）。关键帧位于从流的开始处算起的偏移位置（以毫秒为单位）。
视频流通常是使用以下两种类型的帧进行编码的：关键帧（或　　I　帧）和 P　帧。关键帧包含完整图像；而　P　帧是一个中间帧，它在两个关键帧之间提供额外的视频信息。通常，视频流每　　10　到 50　帧中有一个关键帧。

1.3.1　参数

offset:Number　— 要在视频文件中移动到的时间近似值（以毫秒为单位）　。

1.4 jump

public function jump(av:String,page:int=1,newwindow:Boolean=false):void
跳至只定　AV 号指定页的视频

1.4.1　参数

av:String　— 要跳转的视频 (如 av120040) 。
page:Number　— 要跳转的视频页数。
newwindow:Boolean　— 是否打开新窗口进行跳转

1.4.2　示例

```
Player.jump("av120040",1);
```

1.5 state

state:String [　　只读]
返回播放器播放状态

1.5.1　返回

此事件具有以下属性：

属性	值
playing	播放中
stop	已停止播放
pause	暂停中

1.6 time

time:Number [　只读]
播放头的位置（以毫秒为单位）　。

1.7 commentTrigger

function commentTrigger(f:Function,timeout:Number=1000):uint
监听发送弹幕
注意 ：此函数不会因播放器暂停而终止执行

1.7.1　回调函数定义

function commentCallback(cd:CommentData):void;

1.7.2　参数

f:Function　　—　发送弹幕时执行的回调函数
timeout:Number　　—　监听超时时间

1.7.3　示例

```
Player.commentTrigger(function(data){
  trace(data.time+":"+data.txt);
},30000);
```

1.8 commentTrigger

function keyTrigger(f:Function,timeout:Number=1000):uint
监听键盘输入
注意 ：
此函数不会因播放器暂停而终止执行
此函数只能监听数字键盘　　0-9　及　上下左右　Home, End, Page UP, Page Down

1.8.1　回调函数定义

function keyCallback(key:int,timeout:Number=1000):void;

1.8.2　参数

f :Function　　—　键盘按下时的回调函数
timeout　:Number　—　监听超时时间

1.8.3　示例

```
Player.keyCallback(function(key){
   trace("You click key "+key);
 },30000);
```

1.9 setMask

function setMask(obj:DisplayObject):void
设置播放器遮罩

1.9.1　参数

obj:DisplayObject　　—　作为遮罩的图形对象

1.10 createSound

public function createSound(t:String,onLoad:Function=null):ScriptSound
建立声音元件

1.10.1 参数

t :String　　— 播放声音类型
onLoad:Function　　— 载入完成时的回调函数

1.11 commentList

commentList: Array of CommentData
获取当前弹幕列表

1.11.1 示例

```
var l=Player.commentList.length;
var yes_num = 0;
var no_num = 0;
for (i=0;i<l;i++){
  var cmt=Player.commentList[i];
  if (cmt.txt==''          是'') yes_num++;
  else if (cmt.txt==''             否'') no_num++;
}
$.createComment('    投票结果：  \n   选择是的人数有    "+yes_num+"   人 \n   选择否的人数有    "+no_num+"   人
',{x:100,y:250});
```

1.12 refreshRate

refreshRate: int
弹幕刷新速度 （ 毫秒 ） 默认 :170
取值范围　10-500
精度上限 0.1 秒

2.Display - 弹幕显示相关

2.1 概述

包括了舞台弹幕的操作　　可使用别名 $

2.2 fullScreenWidth

fullScreenWidth:uint [　　只读]
返回变为全屏大小时使用的显示器宽度　　（如果立即进入该状态） 。如果用户有多台显示器，　则使用的显示器是此时显示大部分舞台的显示器。
注意 ：在检索值和变为全屏大小之间，如果用户有机会将浏览器从一台显示器移到另一台显示器，则该值可能不正确。

2.2.1 示例

```
trace('   当前屏幕大小  :'+Display.screenWidth+'x'+Display.screenHeight);
```

2.3 fullScreenHeight

fullScreenWidth:uint [　　只读]
返回变为全屏大小时使用的显示器宽度　　（如果立即进入该状态） 。如果用户有多台显示器，　则使用的显示器是此时显示大部分舞台的显示器。
注意 ：在检索值和变为全屏大小之间，如果用户有机会将浏览器从一台显示器移到另一台显示器，则该值可能不正确。

2.3.1 示例

```
trace("  当前屏幕大小 :"+Display.screenWidth+"x"+Display.screenHeight);
```

2.4 width

width:Number [只读]
指示显示对象的宽度，以像素为单位。

2.5 height

height:Number [只读]
指示显示对象的高度，以像素为单位。

2.6 createMatrix

public function createMatrix():Matrix;
使用指定参数创建新的 [Matrix] 对象。

2.6.1 返回

Matrix 一创建的 [Matrix] 对象。

2.6.2 示例

```
var g = $.createShape({x:0,y:0,lifeTime:3});
 var colors = [0xFF0000, 0x0000FF];
 var alphas = [1, 1];
 var ratios = [0x00, 0xFF];
 var matr = $.createMatrix();
 matr.createGradientBox(20, 20, 0, 0, 0);
 g.graphics.beginGradientFill("linear", colors, alphas, ratios, matr, "pad");
 g.graphics.drawRect(0,0,100,100);
```

2.7 createComment

public function createComment(text:String,param:Object):CommentField
使用指定参数创建新的弹幕对象。

2.7.1 参数

text — 弹幕对象中的文字信息。
param — 创建参数 请参阅 2.11 通用创建参数

2.7.2 返回

CommentField — 新创建的弹幕对象

2.7.3 示例

以下示例为创建一个内容为 ● 的弹幕，存活在屏幕中五秒 其中移动时间 3秒

```
$.createComment("    ●",{
        motion:{
                x:{ fromValue: 4, toValue: 320, lifeTime: 2, startDelay:600, easing:"None"}, /** x                                显
示后延迟   0.3  秒  动态移动    0.5  秒   不使用加速算法    **/
                y:{ fromValue: 30, toValue: 360, lifeTime:1.5, easing:"Linear"} /**y                              动态移动时间     3
秒 **/
        },
        lifeTime: 5 /**          总存活时间   5  秒  */,
        color:Utils.hue(50+bi++*5),
        alpha:0.8
});
```

以下示例会创建一个元件，内容为当前播放时间，存活十秒内会不断变化文字大小、色彩、位置及透明度

```
var a = $.createComment("----",{x:150,y:150,lifeTime:10});
var iu=0;
interval(function(){                    a.textColor=Utils.hue(10*iu++);                    a.alpha=0.2+iu%7/10;
a.htmlText="CURRENT:"+Utils.formatTimes(Player.time/1000);                                  a.fontsize=iu*2;
a.x+=Math.random()*20-10; a.y+=Utils.rand(-10,10); a.rotationZ=Utils.rand(0,90);},50,200);
```

2.8 createShape

public function createShape(text:String,param:Object):Shape
使用指定参数创建新的图型对象。

2.8.1 参数

param — 创建参数   请参阅 2.11   通用创建参数

2.8.2 返回

Shape— 新创建的图型对象   创建图型请参阅   Display.createGraphic.graphics

2.8.3 示例

下面的示例在显示对象注册点   (0, 0)   右侧 250   个像素的位置绘制一个绿色圆形对象，宽度和高度为   100   个像素。   绘制 4   条曲线以生成一个圆，并将其填充为绿色。

```
var g = $.createShape({lifeTime:2,x:10,y:250});
 g.graphics.beginFill(0xFFCC00);
 g.graphics.moveTo(250, 0);
 g.graphics.curveTo(300, 0, 300, 50);
 g.graphics.curveTo(300, 100, 250, 100);
 g.graphics.curveTo(200, 100, 200, 50);
 g.graphics.curveTo(200, 0, 250, 0);
 g.graphics.endFill();
```

下面的示例绘制一个矩形和一个圆，它们使用从红色到绿色、再到蓝色的渐变笔触。

```
var g = $.createShape({lifeTime:3,x:50});
g.graphics.beginGradientFill("linear", [0xFF0000, 0x00FF00, 0x0000FF],[1,1,1],[0x00,0x7f,0xff] , $.createGradientBox(200, 40, 0, 0, 0));
g.graphics.drawRect(0,0,200,40);
g.graphics.drawCircle(100, 120, 50);
g.graphics.endFill();
```

以下示例会创建一个元件，内容为当前播放时间，存活十秒内会不断变化文字大小、色彩、位置及透明度

```
var a = $.createComment("----",{x:150,y:150,lifeTime:10});
var iu=0;
interval(function(){ a.textColor=Utils.hue(10*iu++); a.alpha=0.2+iu%7/10;
a.htmlText="CURRENT:"+Utils.formatTimes(Player.time/1000); a.fontsize=iu*2;
a.x+=Math.random()*20-10; a.y+=Utils.rand(-10,10); a.rotationZ=Utils.rand(0,90);},50,200);
```

2.9 createCanvas

public function createCanvas(text:String,param:Object):CommentCanvas
使用指定参数创建新的容器。

2.9.1 参数

param — 创建参数 请参阅 2.11 通用创建参数

2.9.2 返回

CommentCanvas — 新创建的图型容器

2.9.3 示例

```
var c = $.createCanvas({x:100,y:100});
 $.createComment("    ●",{
        motion:{
                x:{ fromValue: 4, toValue: 320, lifeTime: 2, startDelay:600, easing:"None"}, /** x 显示后延迟 0.3 秒 动态移动 0.5 秒 不使用加速算法 **/
                y:{ fromValue: 30, toValue: 360, lifeTime:1.5, easing:"Linear"} /**y 动态移动时间 3 秒 **/
        },
        lifeTime: 5 /**         总存活时间 5 秒 */,
        color:Utils.hue(50+bi++*5),
        alpha:0.8,
        parent:c
});
```

2.10 createButton

public function createButton(text:String,param:Object):CommentButton
使用指定参数创建新的按钮。

2.10.1 参数

param — 创建参数 请参阅 2.11 通用创建参数

text:String — 按钮标题 创建参数附加值
onclick:Function — 点击处理函数 创建参数附加值

2.10.2 返回

CommentButton — 新创建的按钮

2.10.3 示例

```
a=$.createButton({x:50,y:50,text:"TEST",onclick:function(){
  trace("test");
 }});
```

2.11 通用创建参数

Object

2.11.1 参数

x — 新创建元件的 X 轴座标
y — 新创建元件的 Y 轴座标
lifeTime — 元件的生存时间
alpha — 元件的透明度
color — 文字类元件的色彩
fontsize — 文字类元件的大小
parent — 元件的父元件 可选 (进阶应用)
motion — 元件移动策略 可选

可选属性 x, y, alpha , rotationZ , rotationY

属性值

类型	值	用途
必填	fromValue	起始移动属性值
可选	toValue	结束移动属性值 如留空则不移动
可选	lifeTime	以秒为单位的移动生存时间 如留空则与整体生存时间一致
可选	startDelay	以毫秒为单位的起始移动延时时间
可选	easing	补间效果值： None, Back, Bounce, Circular, Cubic, Elastic, Exponential, Sine, Quintic, Linear

2.12 createGlowFilter

public function createGlowFilter(color:uint = 0xFF0000, alpha:Number = 1.0, blurX:Number = 6.0, blurY:Number = 6.0, strength:Number = 2, quality:int = 1, inner:Boolean = false, knockout:Boolean = false):[GlowFilter]
用指定参数初始化新的 GlowFilter 实例。

2.12.1 参数

color :uint (default = 0xFF0000) — 光晕颜色，采用十六进制格式 0xRRGGBB 默认值为 0xFF0000 。
alpha :Number (default = 1.0) — 颜色的 Alpha 透明度值。有效值为 0 到 1 。例如，0.25 设置透明度值为 25%。
blurX :Number (default = 6.0) — 水平模糊量。有效值为 0 到 255 （浮点）。2 的乘方值（如 2 、4、8、16 和 32）经过优化，呈示速度比其他值更快。
blurY :Number (default = 6.0) — 垂直模糊量。有效值为 0 到 255 （浮点）。2 的乘方值（如 2 、4、8、16 和 32）经过优化，呈示速度比其他值更快。
strength :Number (default = 2) — 印记或跨页的强度。该值越高，压印的颜色越深，而且发光与背景之间的对比度也越强。有效值为 0 到 255 。
quality :int (default = 1) — 应用滤镜的次数。使用 BitmapFilterQuality 常量：

?low

?middle
?high

有关详细信息，请参阅 quality 属性的说明。

inner :Boolean (default = false) — 指定发光是否为内侧发光。值 true 指定发光是内侧发光。值 false 指定发光是外侧发光（对象外缘周围的发光） 。
knockout :Boolean (default = false) — 指定对象是否具有挖空效果。值为 true 将使对象的填充变为透明，并显示文档的背景颜色。 。
inner :Boolean (default = false) — 指定光暈是否為內光暈。 true 值會指定內光暈。 false 值會指定外光暈（ 也就是物件外緣周圍的光暈 ）。
knockout :Boolean (default = false) — 指定物件是否具有去底色特效。 true 值可以讓物件的填色透明，並顯露出文件的背景顏色。

3.Utils - 工具库

3.1 hue

function hue(v:int):int;
将 0-360 的值映射到色相环上 , 例如
0 -> 0x0000ff
120 -> 0xff0000
240 -> 0x00ff00

3.1.1 参数

v:int — 一个整数

3.1.2 返回

Number — 一个色彩代码

3.2 rgb

function rgb(r:int,g:int,b:int):int;
将 RGB值映射到色彩值上

3.2.1 参数

r:int — 一个整数 RED
g:int — 一个整数 GREEN
b:int — 一个整数 BLUE

3.2.2 返回

Number — 一个色彩代码

3.3 formatTimes

function formatTimes(time:Number):String
格式化播放时间

3.3.1 参数

time:Number — 以秒为单位的播放时间

3.3.2 返回

String — 格式化后的播放时间

3.3.3 示例

```
Utils.formatTimes(Player.time/1000);
```

3.4 delay

function delay(f:Function,time:Number=1000):void
延迟执行函数
注意 ：此函数不会因播放器暂停而终止执行

3.4.1 别名

timer 参考 timer

3.4.2 参数

f:Function — 要延迟执行的函数
time:Number — 以毫秒为单位的延迟时间

3.4.3 示例

```
Utils.delay(function(){trace("test delay");},1000);
 timer(function(){trace("test delay");},1000);
```

3.5 interval

function interval(f:Function,time:Number=1000,times:uint=1):void
定时执行函数
注意 ：此函数不会因播放器暂停而终止执行

3.5.1 别名

interval 参考 interval

3.5.2 参数

f:Function — 要定时执行的函数
time:Number — 以毫秒为单位的定时时间
times:Number — 以次为单位的执行次数 0 为无限次

```
Utils.interval(function(){trace("test delay");},1000,5);
 interval(function(){trace("test delay");},1000,5);
```

3.5.3 示例

3.6 distance

function distance(x1:Number,y1:Number,x2:Number,y2:Number):Number;
计算座标距离

3.6.1 参数

x1:Number — 计算起始座标 X 轴
y1:Number — 计算起始座标 Y 轴
x2:Number — 计算结束座标 X 轴
y2:Number — 计算结束座标 Y 轴

3.6.2 返回

Number — 以像素为单位的座标距离

3.7 rand

function rand(min:Number,max:Number):Number
返回一个伪随机数 n，其中 min <= n < max。因为该计算不可避免地包含某些非随机的成分，所以返回的数字以

保密方式计算且为“ 伪随机数 ” 。

3.7.1 参数

min:Number — 伪随机数最小值
max:Number — 伪随机数最大值

3.7.2 返回

Number — 伪随机数 n , 其中 min <= n < max

```
trace("Random value: "+Utils.rand(5,10));
```

3.7.3 示例

4.Math - 数学函数库

4.1 abs

public static function abs(val:Number):Number
计算并返回由参数 val 指定的数字的绝对值。

4.1.1 参数

val:Number — 已返回绝对值的数字。

4.1.2 返回

Number — 指定参数的绝对值。

4.2 acos

public static function acos(val:Number):Number
以弧度为单位计算并返回由参数 val 指定的数字的反余弦值。

4.2.1 参数

val:Number — -1.0 到 1.0 之间的一个数字。

4.2.2 返回

Number — 参数 val 的反余弦值。

4.3 asin

public static function asin(val:Number):Number
以弧度为单位计算并返回由参数 val 指定的数字的反正弦值。

4.3.1 参数

val:Number — -1.0 到 1.0 之间的一个数字。

4.3.2 返回

Number — 介于负二分之 pi 和正二分之 pi 之间的一个数字。

4.4 atan

public static function atan(val:Number):Number
以弧度为单位计算并返回角度值，该角度的正切值已由参数 val 指定。返回值介于负二分之 pi 和正二分之 pi 之间。

4.4.1 参数

val:Number — 表示角的正切值的一个数字。

4.4.2 返回

Number — 介于负二分之 pi 和正二分之 pi 之间的一个数字。

4.5 atan2

public static function atan2(y:Number, x:Number):Number
以弧度为单位计算并返回点 y/x 的角度，该角度从圆的 x 轴（其中， 0,0 表示圆心）沿逆时针方向测量。返回值介于正 pi 和负 pi 之间。请注意， atan2 的第一个参数始终是 y 坐标。

4.5.1 参数

y:Number — 该点的 y 坐标。
x:Number — 该点的 x 坐标。

4.5.2 返回

Number — 一个数字。

4.6 ceil

public static function ceil(val:Number):Number
返回指定数字或表达式的上限值。数字的上限值是大于等于该数字的最接近的整数。

4.6.1 参数

val:Number — 一个数字或表达式。

4.6.2 返回

Number — 最接近且大于等于参数 val 的整数。

4.7 cos

public static function cos(angleRadians:Number):Number
以弧度为单位计算并返回指定角度的余弦值。 要计算弧度，请参阅 Math 类的概述。

4.7.1 参数

angleRadians:Number — 一个数字，它表示一个以弧度为单位的角度。

4.7.2 返回

Number — -1.0 到 1.0 之间的一个数字。

4.8 exp

public static function exp(val:Number):Number
返回自然对数的底 (e) 的 x 次幂的值， x 由参数 x 指定。常量 Math.E 可以提供 e 的值。

4.8.1 参数

val:Number — 指数；一个数字或表达式。

4.8.2 返回

Number — e 的 x 次幂， x 由参数 val 指定。

4.9 floor

public static function exp(val:Number):Number

返回由参数 val 指定的数字或表达式的下限值。下限值是小于等于指定数字或表达式的最接近的整数。

4.9.1 参数

val:Number — 一个数字或表达式。

4.9.2 返回

Number — 最接近且小于等于参数 val 的整数。

4.10 log

public static function log(val:Number):Number
返回参数 val 的自然对数。

4.10.1 参数

val:Number — 其值大于 0 的数字或表达式。

4.10.2 返回

Number — 参数 val 的自然对数。

4.11 max

public static function max(val1:Number, val2:Number, ... rest):Number
计算 val1 和 val2 （或更多的值）并返回最大值。

4.11.1 参数

val1:Number — 一个数字或表达式。
val2:Number — 一个数字或表达式。
... rest — 一个数字或表达式。 Math.max() 可以接受多个参数。

4.11.2 返回

Number — 参数 val1 和 val2 （或更多值）的最大值。

4.12 min

public static function min(val1:Number, val2:Number, ... rest):Number
计算 val1 和 val2 （或更多的值）并返回最小值。

4.12.1 参数

val1:Number — 一个数字或表达式。
val2:Number — 一个数字或表达式。
... rest — 一个数字或表达式。 Math.min() 可以接受多个参数。

4.12.2 返回

Number — 参数 val1 和 val2 （或更多值）的最小值。

4.13 pow

public static function pow(base:Number, pow:Number):Number
计算并返回 base 的 pow 次幂。

4.13.1 参数

base:Number — 将自乘参数 pow 次的数字。
pow:Number — 指定参数 base 的自乘次数的数字。

4.13.2 返回

Number — base 的 pow 次幂的值。

4.14 random

public static function random():Number
返回一个伪随机数 n，其中 0 <= n < 1 。因为该计算不可避免地包含某些非随机的成分，所以返回的数字以保密方式计算且为“伪随机数” 。

4.14.1 返回

Number — 一个伪随机数。

4.15 round

public static function round(val:Number):Number
将参数 val 的值向上或向下舍入为最接近的整数并返回该值。如果 val 与最接近的两个整数等距离（即该数字以 .5 结尾），则该值向上舍入为下一个较大的整数。

4.15.1 参数

val:Number — 要舍入的数字。

4.15.2 返回

Number — 参数 val 舍入为最近的整数。

4.16 sin

public static function sin(angleRadians:Number):Number
以弧度为单位计算并返回指定角度的正弦值。要计算弧度，请参阅 Math 类的概述。

4.16.1 参数

angleRadians:Number — 一个数字，它表示一个以弧度为单位的角度。

4.16.2 返回

Number — 一个数字；指定角度的正弦值（介于 -1.0 和 1.0 之间）。

4.17 sqrt

public static function sqrt(val:Number):Number
计算并返回指定数字的平方根。

4.17.1 参数

val:Number — 一个大于等于 0 的数字或表达式。

4.17.2 返回

Number — 如果参数 val 大于等于 0，则为一个数字，否则为 NaN（非数字）。

4.18 tan

public static function tan(angleRadians:Number):Number
计算并返回指定角度的正切值。要计算弧度，请参阅 Math 类的概述。

4.18.1 参数

angleRadians:Number — 一个数字，它表示一个以弧度为单位的角度。

4.18.2 返回

Number — 参数 angleRadians 的正切值。

5.Display.createGraphic.graphics

5.1 beginFill

public function beginFill(color:uint, alpha:Number = 1.0):void
指定一种简单的单一颜色填充，可将该填充用于随后调用对象的其它 Graphics 方法（如 lineTo() 或 drawCircle() ）。 该填充将保持有效， 直到您调用 beginFill() 或 beginGradientFill() 方法。 调用 clear() 方法会清除填充。
在调用 endFill() 方法之前，不会呈现填充。

5.1.1 参数

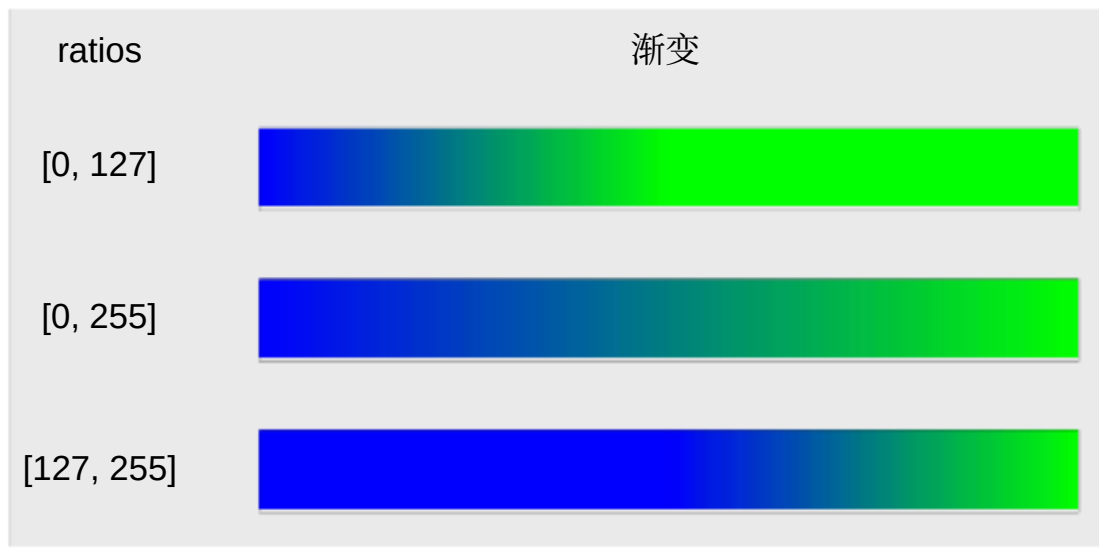
color:uint — 填充的颜色 (0xRRGGBB)。
alpha:Number (default = 1.0) — 填充的 Alpha 值（从 0.0 到 1.0 ）。

5.2 beginGradientFill

public function beginGradientFill(type:String, colors:Array, alphas:Array, ratios:Array, matrix:Matrix = null, spreadMethod:String = "pad", interpolationMethod:String = "rgb", focalPointRatio:Number = 0):void
指定一种渐变填充， 可将该填充用于随后调用对象的其它 Graphics 方法（如 lineTo() 或 drawCircle() ）。 该填充将保持有效，直到您调用 beginFill() 或 beginGradientFill() 方法。 调用 clear() 方法会清除填充。
在调用 endFill() 方法之前，不会呈现填充。

5.2.1 参数

type:String — 用于指定要使用哪种渐变类型的 GradientType 值： linear 或 radial 。
colors:Array — 要在渐变中使用的 RGB 十六进制颜色值数组（例如，红色为 0xFF0000 ，蓝色为 0x0000FF ，等等）。 可以至多指定 15 种颜色。 对于每种颜色，请确保在 alphas 和 ratios 参数中指定对应的值。
alphas:Array — colors 数组中对应颜色的 alpha 值数组；有效值为 0 到 1 。 如果值小于 0 ，则默认值为 0 。如果值大于 1 ，则默认值为 1 。
ratios:Array — 颜色分布比例的数组；有效值为 0 到 255 。 该值定义 100% 采样的颜色所在位置的宽度百分比。 值 0 表示渐变框中的左侧位置， 255 表示渐变框中的右侧位置。
注意： 该值表示渐变框中的位置， 而不是最终渐变的坐标空间，坐标空间可能比渐变框宽或窄。 为 colors 参数中的每个值指定一个值。
例如，对于包括蓝和绿两种颜色的线性渐变，下例显示了基于不同 ratios 数组值的渐变中的颜色配比：

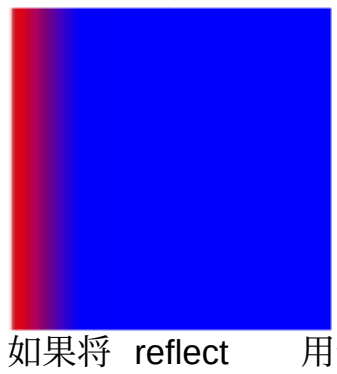


数组中的值必须持续增加； 例如， [0， 63， 127， 190, 255] 。

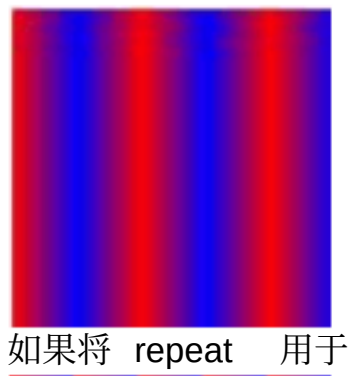
matrix:Matrix (default = null) — 一个由 Matrix 类定义的转换矩阵。 Matrix 类包括 createGradientBox() 方法，通过该方法可以方便地设置矩阵，以便与 beginGradientFill() 方法一起使用，亦可使用 $.createGradientBox() 。
spreadMethod:String (default = "pad") — 用于指定要使用哪种 spread 方法的 值：pad、reflect 或 repeat 。例如，请考虑两种颜色之间的简单线性渐变：

```
var g = $.createShape({x:0,y:0,lifeTime:3});
var colors = [0xFF0000, 0x0000FF];
var alphas = [1, 1];
var ratios = [0x00, 0xFF];
var matr = $.createMatrix();
matr.createGradientBox(20, 20, 0, 0, 0);
g.graphics.beginGradientFill("linear", colors, alphas, ratios, matr, "pad");
g.graphics.drawRect(0,0,100,100);
```

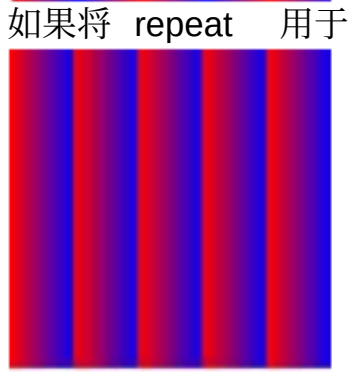
此示例将 pad 用于 spread 方法，并且渐变填充看起来将类似于下图：



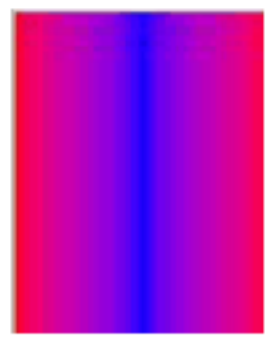
如果将 reflect 用于 spread 方法，则渐变填充看起来将类似于下图：



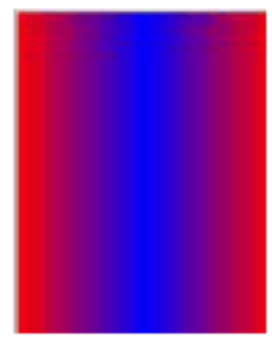
如果将 repeat 用于 spread 方法，则渐变填充看起来将类似于下图：



interpolationMethod:String (default = "rgb") — 用于指定要使用哪个值的 值：linearRGB 或 rgb 例如，假设有两种颜色之间的简单线性渐变 （spreadMethod 参数设置为 reflect ）。 不同的插值方法对外观的影响如下所示：

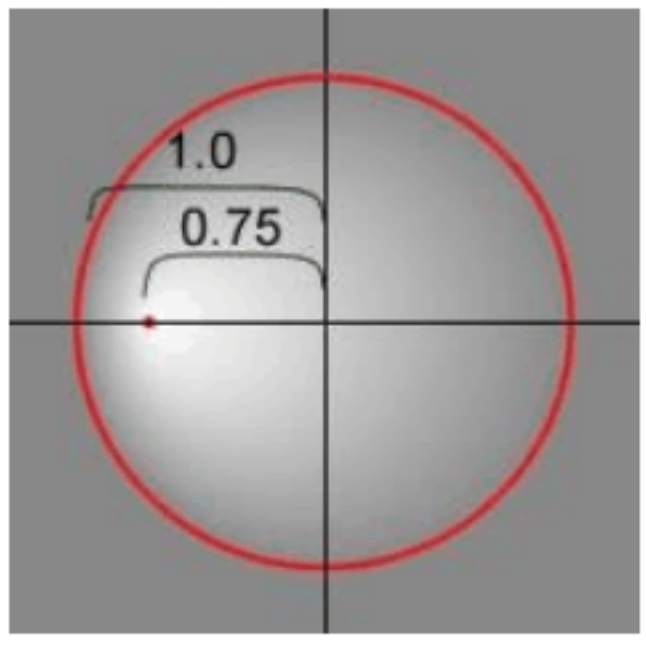


linearRGB



rgb

focalPointRatio:Number (default = 0) — 一个控制渐变的焦点位置的数字。 0 表示焦点位于中心。 1 表示焦点位于渐变圆的一条边界上。 -1 表示焦点位于渐变圆的另一条边界上。 小于 -1 或大于 1 的值将舍入为 -1 或 1 。 例如，下例显示 focalPointRatio 设置为 0.75 ：

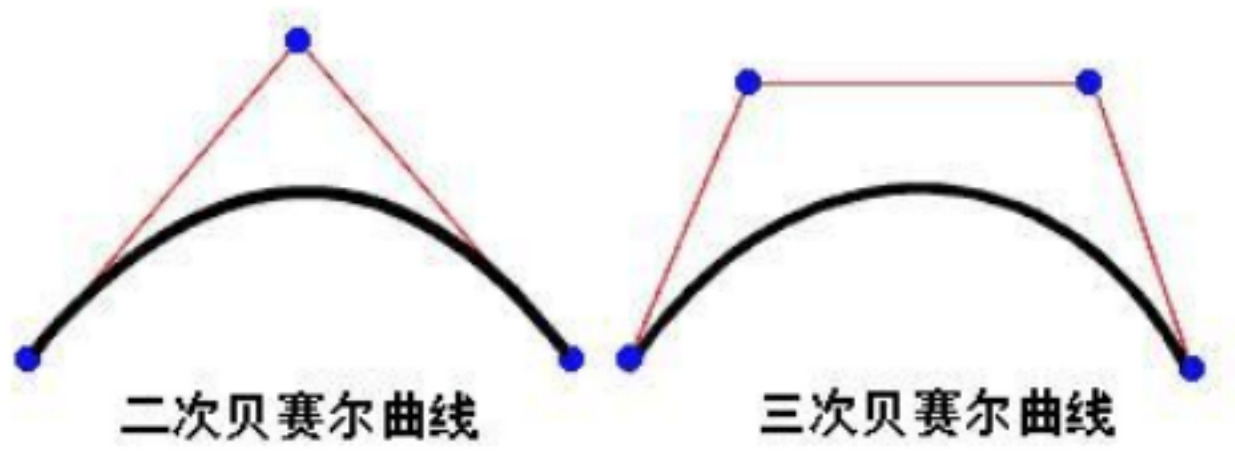


5.3 clear

public function clear():void
清除绘制到此 Graphics 对象的图形，并重置填充和线条样式设置。

5.4 curveTo

public function curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number):void
通过由 (controlX, controlY) 指定的控制点，使用当前线条样式绘制一条从当前绘画位置开始到 (anchorX, anchorY) 结束的曲线。 当前绘画位置随后设置为 (anchorX, anchorY) 。 如果正在其中绘制的影片剪辑包含用 Flash 绘画工具创建的内容，则调用 curveTo() 方法将在该内容下面进行绘制。 如果在调用 moveTo() 方法之前调用了 curveTo() 方法，则当前绘画位置的默认值为 (0, 0)。 如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。
绘制的曲线是二次贝塞尔曲线。 二次贝塞尔曲线包含两个锚点和一个控制点。 该曲线内插这两个锚点，并向控制点弯曲。



5.4.1 参数

controlX:Number — 一个数字，指定控制点相对于父显示对象注册点的水平位置。
controlY:Number — 一个数字，指定控制点相对于父显示对象注册点的垂直位置。
anchorX:Number — 一个数字，指定下一个锚点相对于父显示对象注册点的水平位置。
anchorY:Number — 一个数字，指定下一个锚点相对于父显示对象注册点的垂直位置。

5.4.2 示例

下面的示例在显示对象注册点 (0, 0) 右侧 250 个像素的位置绘制一个绿色圆形对象，宽度和高度为 100 个像素。 绘制 4 条曲线以生成一个圆，并将其填充为绿色。
请注意， 由于二次贝塞尔方程式所具有的特性， 此对象并不是完美的圆。 绘制圆的最佳方法是使用 Graphics 类的 drawCircle() 方法。

```
var g = $.createShape({lifeTime:2,x:10,y:250});
 g.graphics.beginFill(0xFFCC00);
 g.graphics.moveTo(250, 0);
 g.graphics.curveTo(300, 0, 300, 50);
 g.graphics.curveTo(300, 100, 250, 100);
 g.graphics.curveTo(200, 100, 200, 50);
 g.graphics.curveTo(200, 0, 250, 0);
 g.graphics.endFill();
```

下面的示例使用 curveTo() 方法绘制一个新月。
绘制两条 1 个像素粗的曲线， 并将两条曲线之间的区域填充为白色。 moveTo() 方法用于将当前绘制位置放在坐标 (100, 100) 上。 第一条曲线将绘制位置移到 (100, 200) ，这是其目标点。 第二条曲线将该位置恢复为开始位置 (100, 100) ，这是其目标点。 水平控制点决定了不同的曲线大小。

```
var newMoon = $.createShape({lifeTime:2,x:0,y:0});


  newMoon.graphics.lineStyle(1, 0);
  newMoon.graphics.beginFill(0xFFFFFF);
  newMoon.graphics.moveTo(100, 100);
  newMoon.graphics.curveTo(30, 150, 100, 200);
  newMoon.graphics.curveTo(50, 150, 100, 100);
  newMoon.graphics.endFill();
```

下面的示例使用 curveTo() 方法绘制一把扇。

```
function drawArc(r,ang,color,rttz) {
        var myObj = $.createShape({x:270,y:200,lifeTime:3});
        myObj.graphics.beginFill(color);
        myObj.graphics.lineStyle(1);
        var a=ang/180*Math.PI/0.01;
        myObj.graphics.lineTo(r,0);
        for (var i=0; i<=a; i++) {

        myObj.graphics.curveTo(Math.cos((i-0.5)*0.01)*r,Math.sin((i-0.5)*0.01)*r,Math.cos(i*0.01
)*r,Math.sin(i*0.01)*r);
        }
        myObj.graphics.endFill();
        myObj.rotationZ=rttz;
};
drawArc(100,120,0xff99ff,-150);
drawArc(30,120,0xff99ff,30);
```

5.5 drawCircle

public function drawCircle(x:Number, y:Number, radius:Number):void
绘制一个圆。 您必须在调用 drawCircle() 方法之前，通过调用 linestyle() 、lineGradientStyle() 、beginFill() 或 beginGradientFill() 方法来设置线条样式和 / 或填充。

5.5.1 参数

x:Number — 相对于父显示对象注册点的圆心的 x 位置（以像素为单位） 。
y:Number — 相对于父显示对象注册点的圆心的 y 位置（以像素为单位） 。
radius:Number — 圆的半径（以像素为单位） 。

5.6 drawEllipse

public function drawEllipse(x:Number, y:Number, width:Number, height:Number):void
绘制一个椭圆。 您必须在调用 drawEllipse() 方法之前，通过调用 linestyle() 、lineGradientStyle() 、beginFill() 或 beginGradientFill() 方法来设置线条样式和 / 或填充。

5.6.1 参数

x:Number — 相对于父显示对象注册点的椭圆圆心的 x 位置（以像素为单位） 。
y:Number — 相对于父显示对象注册点的椭圆圆心的 y 位置（以像素为单位） 。
width:Number — 椭圆的宽度（以像素为单位） 。
height:Number — 椭圆的高度（以像素为单位） 。

5.6.2 示例

下面的示例使用 drawEgg() 函数绘制三个不同大小的鸡蛋（三种尺寸的椭圆） ，具体取决于 eggSize 参数。 构造函数调用 drawEgg() 函数，并为应该绘制鸡蛋的位置传递水平和垂直参数以及鸡蛋类型 (eggSize) 。 （可使用鸡蛋（椭圆形）高度和宽度来确定其显示位置。 ）drawEgg() 函数绘制不同大小的椭圆， 并使用 beginFill() 方法将其填充为白色。 没有事先为此函数编写错误处理代码。

```
var SMALL = 0;
 var MEDIUM = 1;
 var LARGE = 2;


 function drawEgg(eggSize, x, y)  {
        var myEgg = $.createShape({x:0,y:0,lifeTime:3});

 myEgg.graphics.beginFill(0xFFFFFF);
 myEgg.graphics.lineStyle(1);

 switch(eggSize) {
        case SMALL:
               myEgg.graphics.drawEllipse(x, y, 60, 70);
               break;
        case MEDIUM:
               myEgg.graphics.drawEllipse(x, y, 120, 150);
               break;
        case LARGE:
               myEgg.graphics.drawEllipse(x, y, 150, 200);
               break;
               default:
                      trace ("Wrong size! There is no egg.");
               break;
        }

        myEgg.graphics.endFill();
}
drawEgg(SMALL, 0, 100);
drawEgg(MEDIUM, 100, 60);
drawEgg(LARGE, 250, 35);
```

5.7 drawRect

public function drawRect(x:Number, y:Number, width:Number, height:Number):void
绘制一个矩形。 您必须在调用 drawRect() 方法之前，通过调用 linestyle() 、lineGradientStyle() 、beginFill() 或 beginGradientFill() 方法来设置线条样式和 / 或填充。

5.7.1 参数

x:Number — 一个表示相对于父显示对象注册点的水平位置的数字（以像素为单位） 。
y:Number — 一个表示相对于父显示对象注册点的垂直位置的数字（以像素为单位） 。
width:Number — 矩形的宽度（以像素为单位） 。
height:Number — 矩形的高度（以像素为单位） 。

5.8 drawRoundRect

public function drawRoundRect(x:Number, y:Number, width:Number, height:Number, ellipseWidth:Number, ellipseHeight:Number):void

绘制一个圆角矩形。 您必须在调用 drawRoundRect() 方法之前，通过调用 linestyle() 、lineGradientStyle() 、beginFill() 或 beginGradientFill() 方法来设置线条样式和 / 或填充。

5.8.1 参数

x:Number — 一个表示相对于父显示对象注册点的水平位置的数字（以像素为单位） 。
y:Number — 一个表示相对于父显示对象注册点的垂直位置的数字（以像素为单位） 。
width:Number — 圆角矩形的宽度（以像素为单位） 。
height:Number — 圆角矩形的高度（以像素为单位） 。
ellipseWidth:Number — 用于绘制圆角的椭圆的宽度（以像素为单位） 。
ellipseHeight:Number — 用于绘制圆角的椭圆的高度（以像素为单位） 。 （可选）如果未指定值，则默认值与为 ellipseWidth 参数提供的值相匹配。

5.9 endFill

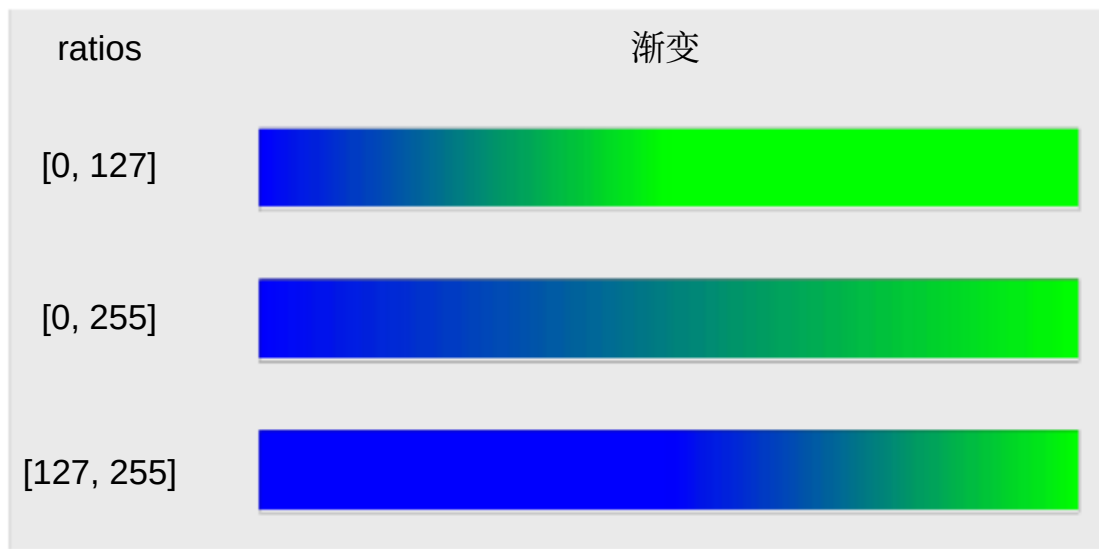
public function endFill():void
对从上一次调用 beginFill() 或 beginGradientFill() 方法之后添加的直线和曲线应用填充。 Flash 使用的是对 beginFill() 或 beginGradientFill() 方法的先前调用中指定的填充。 如果当前绘画位置不等于 moveTo() 方法中指定的上一个位置，而且定义了填充，则用线条闭合该路径，然后进行填充。

5.10 lineGradientStyle

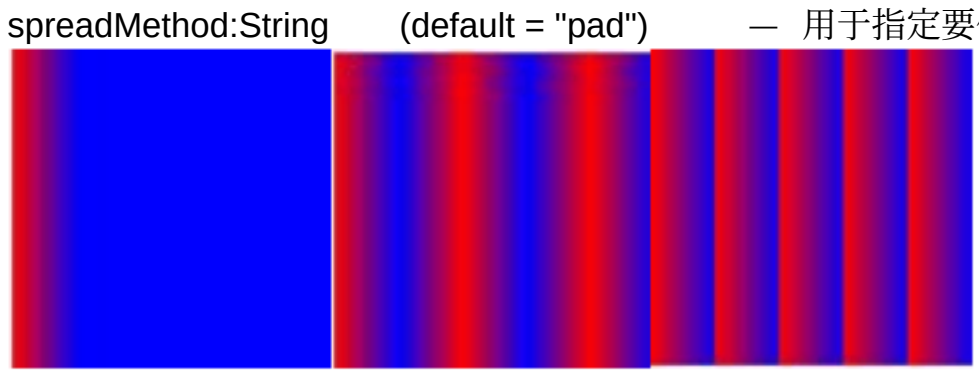
public function lineGradientStyle(type:String, colors:Array, alphas:Array, ratios:Array, matrix:Matrix = null, spreadMethod:String = "pad", interpolationMethod:String = "rgb", focalPointRatio:Number = 0):void
指定一种线条样式的渐变， Flash Player 可将该渐变用于随后调用对象的其它 Graphics 方法（如 lineTo() 或 drawCircle() ）。 线条样式仍然有效，直到使用不同的参数调用 lineStyle() 方法或 lineGradientStyle() 方法为止。 可以在绘制路径的中间调用 lineGradientStyle() 方法以为路径中的不同线段指定不同的样式。
在调用 lineGradientStyle() 之前调用 lineStyle() 以启用笔触，否则线条样式的值仍然是 undefined 。
调用 clear() 会将线条样式设置回 undefined 。

5.10.1 参数

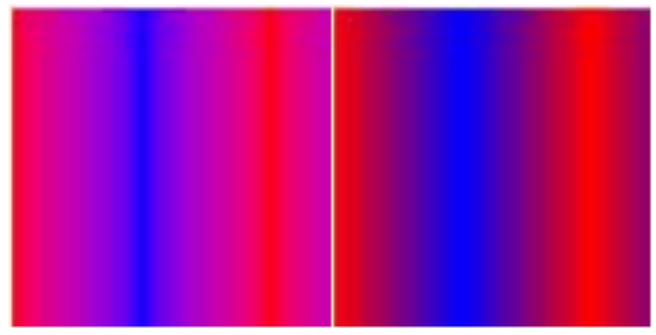
type:String — 用于指定要使用哪种渐变类型的 GradientType 值： linear 或 radial 。
colors:Array — 要在渐变中使用的 RGB 十六进制颜色值数组（例如，红色为 0xFF0000 ，蓝色为 0x0000FF ，等等）。
alphas:Array — colors 数组中对应颜色的 alpha 值数组；有效值为 0 到 100 。 如果值小于 0 ，Flash Player 将使用 0 。 如果值大于 100 ， Flash Player 将使用 100 。
ratios:Array — 颜色分布比率的数组；有效值为 0 到 255 。 该值定义 100% 采样的颜色所在位置的宽度百分比。 值 0 表示渐变框中的左侧位置， 255 表示渐变框中的右侧位置。 该值表示渐变框中的位置， 而不是最终渐变的坐标空间，坐标空间可能比渐变框宽或窄。 为 colors 参数中的每个值指定一个值。
例如，对于包括蓝和绿两种颜色的线性渐变，下例显示了基于不同 ratios 数组值的渐变中的颜色配比：



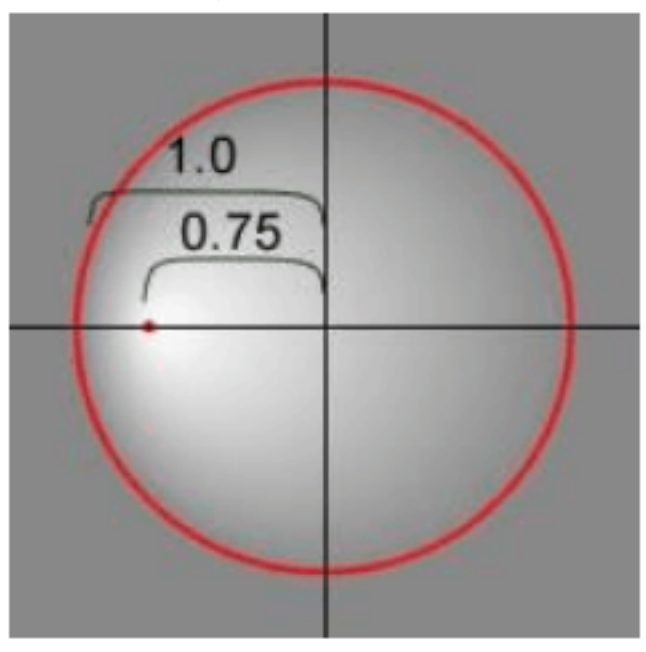
matrix:Matrix (default = null) — 一个由 Matrix 类定义的转换矩阵。 Matrix 类包括 createGradientBox() 方法，通过该方法可以方便地设置矩阵，以便与 beginGradientFill() 方法一起使用，亦可使用 $.createGradientBox() 。
spreadMethod:String (default = "pad") — 用于指定要使用哪种 spread 方法的 值：



pad reflect repeat
interpolationMethod:String (default = "rgb") — 用于指定要使用哪个值的 值：linearRGB 或 rgb
例如，假设有两种颜色之间的简单线性渐变 （spreadMethod 参数设置为 reflect ）。 不同的插值方法对外观的影响如下所示：



linearRGB rgb
focalPointRatio:Number (default = 0) — 一个控制渐变的焦点位置的数字。 0 表示焦点位于中心。 1 表示焦点位于渐变圆的一条边界上。 -1 表示焦点位于渐变圆的另一条边界上。 小于 -1 或大于 1 的值将舍入为 -1 或 1 。 例如，下例显示 focalPointRatio 设置为 0.75 ：



5.10.2 示例

下面的示例绘制一个矩形和一个圆，它们使用从红色到绿色、再到蓝色的渐变笔触。
Matrix 类中的方法 createGradientBox() 用于将渐变框定义为 200 个像素宽、 40 个像素高。 线条粗细设置为 5 个像素。 必须为 lineGradientStyle() 方法定义笔触粗细。 渐变设置为线性。 渐变颜色设置为红色、绿色和蓝色。 颜色透明度 （Alpha 值）设置为 1（不透明）。 渐变的分布是均匀的， 其中在 0（渐变框中的左侧位置） 、128（渐变框的中间位置）和 255（渐变框中的右侧位置）处的颜色采样率为 100%。 矩形宽度范围内包含渐变的所有色谱；而圆包含色谱中间的 50%。

```
var g = $.createShape({lifeTime:3,x:50});
g.graphics.beginGradientFill("linear", [0xFF0000, 0x00FF00, 0x0000FF],[1,1,1],[0x00,0x7f,0xff] , $.createGradientBox(200, 40, 0, 0, 0));
g.graphics.drawRect(0,0,200,40);
g.graphics.drawCircle(100, 120, 50);
g.graphics.endFill();
```

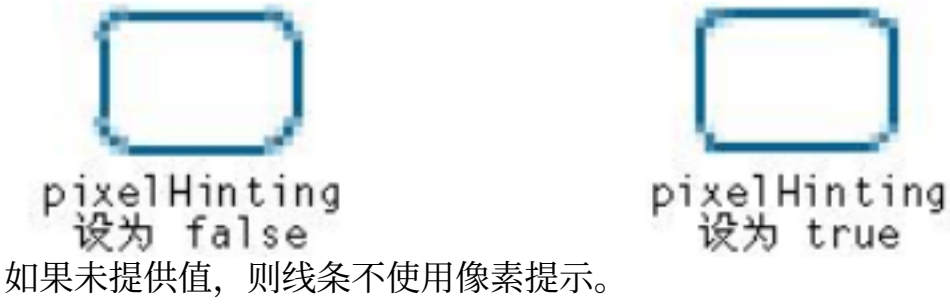
5.11 lineStyle

public function lineStyle(thickness:Number, color:uint = 0, alpha:Number = 1.0, pixelHinting:Boolean = false, scaleMode:String = "normal", caps:String = null, joints:String = null, miterLimit:Number = 3):void
指定一种线条样式， Flash 可将该样式用于随后调用对象的其它 Graphics 方法（如 lineTo() 或 drawCircle() ）。 线条样式仍然有效，直到使用不同的参数调用 lineGradientStyle() 方法或 lineStyle() 方法为止。 可以在绘制路径的中间调用 lineStyle() ，以便为路径中的不同线段指定不同的样式。
注意 ：调用 clear() 方法会将线条样式设置回 undefined 。

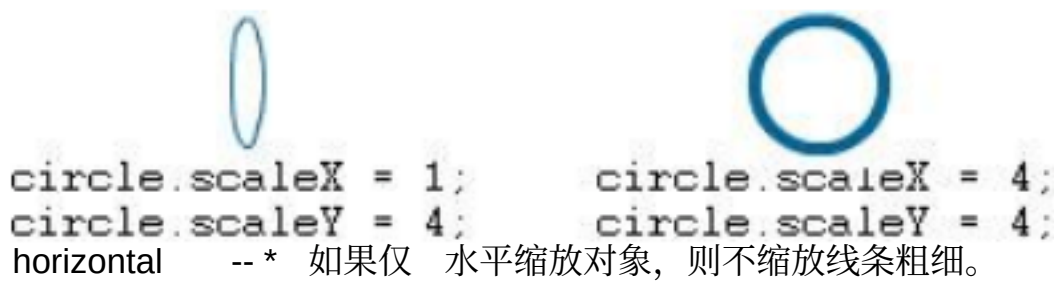
5.11.1 参数

thickness:Number — 一个整数，以磅为单位表示线条的粗细；有效值为 0 到 255 。 如果未指定数字，或者未定义该参数，则不绘制线条。 如果传递的值小于 0 ，则默认值为 0 。 值 0 表示极细的粗细；最大粗细为 255 。 如果传递的值大于 255 ，则默认值为 255 。
color:uint (default = 0) — 线条的十六进制颜色值（例如，红色为 0xFF0000 ，蓝色为 0x0000FF 等）。 如果未指明值，则默认值为 0x000000 （黑色）。 可选。
alpha:Number (default = 1.0) — 表示线条颜色的 Alpha 值的数字；有效值为 0 到 1 。 如果未指明值，则

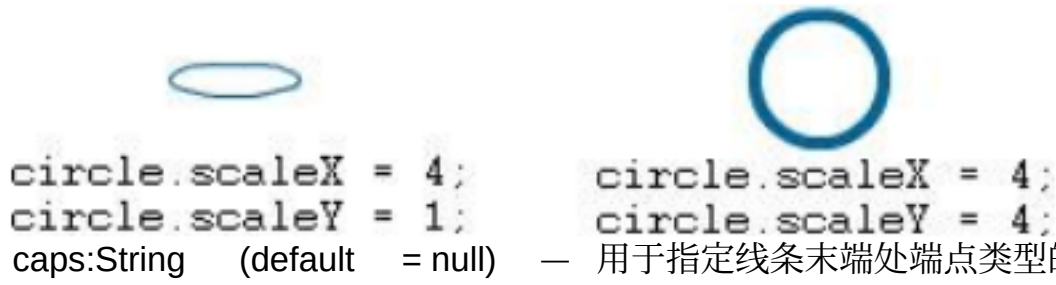
默认值为 1（纯色）。 如果值小于 0，则默认值为 0。 如果值大于 1，则默认值为 1。
pixelHinting:Boolean (default = false) — 用于指定是否提示笔触采用完整像素的布尔值。 它同时影响曲线锚点的位置以及线条笔触大小本身。 在 pixelHinting 设置为 true 的情况下， Flash Player 将提示线条宽度采用完整像素宽度。 在 pixelHinting 设置为 false 的情况下，对于曲线和直线可能会出现脱节。 例如， 下图显示了 Flash Player 如何呈现两个相同的圆角矩形， 不同之处是用于 lineStyle() 方法的 pixelHinting 参数设置不同（将图像放大 200% 以强调差异）：



如果未提供值，则线条不使用像素提示。
scaleMode:String (default = "normal") — 用于指定要使用哪种缩放模式的 LineScaleMode 类的值：
normal -- 在缩放对象时总是缩放线条的粗细（默认值） 。
none -- 从不缩放线条粗细。
vertical -- 如果仅 垂直缩放对象，则不缩放线条粗细。 例如，考虑下面的圆形，它们是用一个像素的线条绘制的， 每个圆的 scaleMode 参数都被设置为 vertical 。 左边的圆仅在垂直方向上缩放， 而右边的圆则同时在垂直和水平方向上缩放：



horizontal -- * 如果仅 水平缩放对象，则不缩放线条粗细。 例如，考虑下面的圆形，它们是用一个像素的线条绘制的， 每个圆的 scaleMode 参数都被设置为 horizontal 。 左边的圆仅在水平方向上缩放， 而右边的圆则同时在垂直和水平方向上缩放：



caps:String (default = null) — 用于指定线条末端处端点类型的 CapsStyle 类的值。 有效值为： none、round 和 square 。 如果未指示值，则 Flash 使用圆头端点。
例如，以下图示显示了不同的 capsStyle 设置。 对于每种设置，插图显示了一条粗细为 30 的蓝色线条（应用 capsStyle 的线条），以及重叠于其上的粗细为 1 的黑色线条（未应用 capsStyle 的线条）：

joints:String (default = null) — JointStyle 类的值，指定用于拐角的连接外观的类型。 有效值为： bevel 、miter 和 round 。 如果未指示值，则 Flash 使用圆角连接。
例如，以下图示显示了不同的 joints 设置。 对于每种设置， 插图显示了一条粗细为 30 的带拐角的蓝色线条 （应用 jointStyle 的线条），以及重叠于其上的粗细为 1 的带拐角的黑色线条（未应用 jointStyle 的线条）：

注意：对于设置为 JointStyle.MITER 的 joints ，您可以使用 miterLimit 参数限制尖角的长度。
miterLimit:Number (default = 3) — 一个表示将在哪个限制位置切断尖角的数字。 有效值的范围是 1 到 255（超出该范围的值将舍入为 1 或 255 ）。此值只可用于 jointStyle 设置为 "miter" 的情况下。 miterLimit 值表示向外延伸的尖角可以超出角边相交所形成的结合点的长度。 此值表示为线条 thickness 的因子。 例如， miterLimit 因子为 2.5 且 thickness 为 10 像素时，尖角将在 25 像素处切断。
例如，请考虑下列带拐角的线条，每个线条都以 thickness 20 进行绘制，但它们的 miterLimit 分别设置为 1 、2 和 4 。 重叠在其上的黑色参考线条显示了结合处的联结点：

请注意，对于给定的 miterLimit 值，会有一个被切断的尖角的特定最大角度。 下表列出了部分示例：

miterLimit 值：	小于此角度将被切断：
1.414	90 度
2	60 度
4	30 度
8	15 度

5.12 lineTo

public function lineTo(x:Number, y:Number):void
使用当前线条样式绘制一条从当前绘画位置开始到 (x, y) 结束的直线； 当前绘画位置随后会设置为 (x, y) 。 如果正在其中绘制的显示对象包含用 Flash 绘画工具创建的内容，则调用 lineTo() 方法将在该内容下面进行绘制。 如果在对 moveTo() 方法进行任何调用之前调用了 lineTo() ，则当前绘画的默认位置为 (0, 0) 。 如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

5.12.1 参数

x:Number — 一个表示相对于父显示对象注册点的水平位置的数字（以像素为单位） 。
y:Number — 一个表示相对于父显示对象注册点的垂直位置的数字（以像素为单位） 。

5.12.2 示例

下面的示例使用 lineTo() 方法绘制一个梯形，从像素 (100, 100) 开始。
线条粗细设置为 10 个像素，颜色为金色且不透明，线条终点设置为 none （因为联接了所有线条） ，两条线之间的联接点设置为 MITER 并将尖角限制设置为 10 以绘制尖锐的边角。

```
var trapezoid = $.createShape({lifeTime:2,x:250,y:0});
trapezoid.graphics.lineStyle(10, 0xFFD700, 1, false, "vertical","none", "miter", 10);
trapezoid.graphics.moveTo(100, 100);
trapezoid.graphics.lineTo(120, 50);
trapezoid.graphics.lineTo(200, 50);
trapezoid.graphics.lineTo(220, 100);
trapezoid.graphics.lineTo(100, 100);
```

5.13 moveTo

public function moveTo(x:Number, y:Number):void
将当前绘画位置移动到 (x, y) 。 如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

5.13.1 参数

x:Number — 一个表示相对于父显示对象注册点的水平位置的数字（以像素为单位） 。
y:Number — 一个表示相对于父显示对象注册点的垂直位置的数字（以像素为单位） 。

5.13.2 示例

下面的示例使用 moveTo() 和 lineTo() 方法绘制一条 3 个像素粗的虚线。
通过使用 lineStyle() 方法，将线条粗细设置为 3 个像素。 还将其设置为不进行缩放。 颜色设置为红色，并且不透明度为 25%。 CapsStyle 属性设置为 square （默认值为 round ）。

```
var trapezoid = $.createShape({lifeTime:2,x:250,y:0});
 trapezoid.graphics.lineStyle(3, 0x990000, 0.25, false,"none","square");
 trapezoid.graphics.moveTo(10, 20);
 trapezoid.graphics.lineTo(20, 20);
 trapezoid.graphics.moveTo(30, 20);
 trapezoid.graphics.lineTo(50, 20);
 trapezoid.graphics.moveTo(60, 20);
 trapezoid.graphics.lineTo(80, 20);
 trapezoid.graphics.moveTo(90, 20);
 trapezoid.graphics.lineTo(110, 20);
 trapezoid.graphics.moveTo(120, 20);
 trapezoid.graphics.lineTo(130, 20);
```

6.CommentField

6.1 属性

alwaysShowSelection : Boolean
如果设置为 true 且文本字段没有焦点， Flash Player 将以灰色突出显示文本字段中的所选内容。
background : Boolean
指定文本字段是否具有背景填充。
backgroundColor : uint
文本字段背景的颜色。
border : Boolean
指定文本字段是否具有边框。
borderColor : uint
文本字段边框的颜色。
bottomScrollV : int [只读]
一个整数（从 1 开始的索引），指示指定文本字段中当前可以看到的最后一行。
condenseWhite : Boolean
一个布尔值，指定是否删除具有 HTML 文本的文本字段中的额外空白（空格、换行符等等） 。
defaultTextFormat : flash.text:TextFormat
指定应用于新插入文本（例如，用户输入的文本或使用 replaceSelectedText() 方法插入的文本）的格式。
gridFitType : String
用于此文本字段的网格固定类型。
htmlText : String
包含文本字段内容的 HTML 表示形式。
length : int [只读]
文本字段中的字符数。
multiline : Boolean
指示字段是否为多行文本字段。
numLines : int [只读]
定义多行文本字段中的文本行数。
restrict : String
指示用户可输入到文本字段中的字符集。
sharpness : Number
此文本字段中字型边缘的清晰度。
text : String
作为文本字段中当前文本的字符串。

textColor : uint
文本字段中文本的颜色（采用十六进制格式）　　。
textHeight : Number [　　只读]
文本的高度，以像素为单位。
textWidth : Number [　　只读]
文本的宽度，以像素为单位。
thickness : Number
此文本字段中字型边缘的粗细。
wordWrap : Boolean
一个布尔值，指示文本字段是否自动换行。
fontsize : Number
此文本字段中字型的大小。
bold : Boolean
一个布尔值，指示文本字是否粗体。

6.2 方法

6.2.1 appendText

public function appendText(newText:String):void
将 newText　参数指定的字符串追加到文本字段的文本的末尾。此方法要比对　　text　属性的加法赋值　(+=)　（如 someTextField.text += moreText　　）更有效，对于包含大量内容的文本字段尤其有效。

6.2.1.1 参数

newText:String　　— 要追加到现有文本末尾的字符串。

7.ECMAScript　相关 (待整理)

7.1 ECMAScript　基础

7.1.1 基本特性

ECMAScript 的语言特性和　Java、C、Perl　都有许多相似之处，其中不少特性都是从这些语言借鉴而来，同时它们之间也存在许多差异。下面列举一些　　ECMAScript 的基本特性。
· 和　Java　一样，　ECMAScript　区分大小写，注释的格式相同，通过　　{} 确定代码块，原始数据类型存储在堆栈，对象的引用存储在堆中
· ECMAScript　是一种松散的语言，　ECMAScript　通过　var　操作符声明变量，并且不限类型，例如　　var n = 25　，那么 n 就是数字类型，　var n = "string"　　，那么　n 就是　String　类型
· 在每一行代码后，　可以不写分号，　ECMAScript 自动认为该行的末尾为该行代码的最后；　　ECMAScript　中的变量可以不用初始化，在幕后系统将自动完成初始化操作
· 同一变量可以赋予不同类型的数据；　　变量的第一个字符只能是字母、　　下划线或 $ ，其他的字符可以是下划线、　$、或任意的字母、数字、字符
· 和其他语言一样，变量最好遵循驼峰书写法，或　　Pascal　表示法、或匈牙利表示法
· 和大多数语言不同的是，　　ECMAScript　变量在使用之前可以不必声明，系统会自动将该变量声明为全局变量，例如　var m = " Good " ; n = m + " Morning " ; alert(n)　　输出结构是　" Good Morning "
· 在大多数语言里，　String　是对象，在　ECMAScript 中却是原始数据类型
· 注释以单斜杠和星号开头（　/* ），以星号和单斜杠结尾（　*/ ）

7.1.2 原始数据类型

ECMAScript 原始数据类型有五种：　undefined　、Null　、Boolean、Number、String　。
· typeof　—判断变量和值的数据类型，通常有　　undefined　、boolean 、number、string　、object　五种类型。
·undefined　—当变量被声明但没有初始化，　　或函数没有明确返回一个值的时候，　　该变量或函数即为　undefined　　类型。
· null　—undefined　是 null　的一种派生，当代表一个对象的值不存在时，该对象返回　　null　。
· Boolean —包含两个值，　true and false　　，　false　不等于 0，但 0可以转换为　false　。
· Number—可以定义 32位整型数据或 64位浮点型数据。定义数字类型变量时，在数字前加　　0即为八进制，加　0x　为十六进制，它们计算后返回的结果统一为十进制。通过　　var f = 1.0　　可以定义一个浮点类型变量，有意思的是，当 f　被用于计算之前，它实际是以　　String　　类型存储的。当浮点类型数据很大或很小时（可以前后移动六位）　　，将使用　E表示法来表示浮点数据，　最大可以存储　17位数据。另外，isFinite()　　方法可以判断一个数值是否有限，　　isNaN()

方法可以判断一个数据是非数字类型。

· String — String 在 ECMAScript 中是原始数据类型，并且是唯一没有空间大小限制的数据类型。和 Java 不同的是， var s = " javascript " 和 var s = 'javascript' 均是合法的表示方法。

7.1.3 数据转换

在不同数据类型之间转换是任何一门编程语言的一个重要特性， ECMAScript 提供了一系列简单的方法来实现数据的转换，大多数数据类型都提供了简单的转换方法，对于复杂的转换则有一些全局方法来完成，不管是哪一种方法， ECMAScript 中数据转换都非常简单。

Boolean 、number 和 string 数据类型是原始数据类型，但它们同时是伪对象，拥有自己的属性和方法，可以通过 toString() 方法来实现 string 类型的转换。 ECMAScript 定义所有的对象，不管是伪对象还是真实的对象，都可以实现 toString() 方法，string 被列为伪对象的行列， 自然也拥有 toString() 方法。将数字类型数据转换为 string 的时候，可以在 toString() 方法中加入 2、 8、 16 参数，来实现不同进制的数据输出，例如 var n = 10; trace(n.toString(2)) 输出为 1010， trace(n.toString(8)) 输出为 12， n.toString() 和 n.toString(10) 相同。

ECMAScript 提供了两种方法来实现 string 类型转化为数字类型的方法： parseInt() 和 parseFloat() 。其他类型转换将会返回 NaN（Not a Number ）。

7.1.4 操作符和语句

ECMAScript 中大多数操作符、语句和 Java 都比较类似，但也有一些其特有的，如 label 语句， with 语句， for-in 语句等等。

7.1.4.1 Functions

Functions 是 ECMAScript 的核心，在任何时候任何地方都可以运行的一组代码语句。

```
function functionName(arg0, arg1,          ,,      , argN) {
  statements
}
```

当 function 没有返回值或 return 语句后没有值的时候，该 function 实际上会被系统定义为 undefined ，当 function 返回值的时候， function 可以不必明确指定为某种数据类型。

7.1.4.2 关于重载

重载是面向对象语言的基本特性之一，但 ECMAScript 的 functions 并不能重载，在同一范围里可以定义两个完全相同的函数，在调用函数的时候，最后的一个函数发挥作用。这种特性比较麻烦，但可以通过 arguments 对象来实现和重载类似的功能。

```
function func() {
  if (arguments.length == 1 ) {
    trace(arguments[ 0 ] + 5 );
  } else if (arguments.length == 2 ) {
    trace(arguments[ 0 ] + arguments[ 1 ]);
  }
}
func( 5 ); /*          输出  10 */
func( 10 , 15 ); /*            输出  25 */
```

前面提到过，在同一范围里可以定义两个完全相同的 function ，在调用 function 的时候，最后的一个 function 发挥作用。

```
function func(i) {
  trace(i + 10 );
}
function func(i) {
  trace(i + 20 );
}
func( 5 ); /*          输出  25 */
```

可以看出，是调用了最后的一个 function 使得数据结果为 25，如果使用 Function 类来定义以上两个函数，那为什么会使用最后的一个 function 可能会更明确一些。

```
var func = new Function(       “ i ”, “ alert(i + 10 )       ”);
var func = new Function(       “ i ”, “ alert(i + 20 )       ”);
func( 5 );
```

func 指向了另外一个引用，从而值发生了改变，func 是作为 function 对象的引用而存在的，并且允许两个变量指向同一个 function 。

和 Function 类相关的属性、方法有许多，例如 length 、toString() 、valueOf() 等等。其中 toString() 在调试程序中使用较多。

7.2 ECMAScript 变量

请使用 var 运算符声明变量。

变量名需要遵守一些简单的规则。

7.2.1 声明变量

ECMAScript 中的变量是用 var 运算符（ variable 的缩写）加变量名定义的。例如：

```
var test = "hi";
```

在这个例子中，声明了变量 test ，并把它的值初始化为 "hi" （字符串）。由于 ECMAScript 是弱类型 的，所以解释程序会为 test 自动创建一个字符串值，无需明确的类型声明 。

还可以用一个 var 语句定义两个或多个变量：

```
var test1 = "hi", test2 = "hello";
```

前面的代码定义了变量 test1 ，初始值为 "hi" ，还定义了变量 test2 ，初始值为 "hello" 。

不过用 同一个 var 语句定义的变量不必具有相同的类型 ，如下所示：

```
var test = "hi", age = 25;
```

这个例子除了（再次）定义 test 外，还定义了 age，并把它初始化为 25。即使 test 和 age 属于两种不同的数据类型，在 ECMAScript 中这样定义也是完全合法的。

与 Java 不同， ECMAScript 中的变量 并不一定要初始化 （它们是在幕后初始化的，将在后面讨论这一点） 。因此，

```
var test;
```

下面这一行代码也是有效的：

此外，与 Java 不同的还有变量 可以存放不同类型的值 。这是弱类型变量的优势。例如，可以把变量初始化为字符串类型的值，之后把它设置为数字值，如下所示：

```
var test = "hi"; trace(test); test = 55; trace(test);
```

这段代码将毫无问题地输出字符串值和数字值。但是，如前所述，使用变量时，好的编码习惯是始终存放相同类型的值。

7.2.2 命名变量

变量名需要遵守两条简单的规则：

· 第一个字符必须是字母、下划线（ _）或美元符号（ $）

· 余下的字符可以是下划线、美元符号或任何字母或数字字符

下面的变量都是合法的：

```
var test; var $test; var $1; var _$te$t2;
```

7.2.3 著名的变量命名规则

只是因为变量名的语法正确，并不意味着就该使用它们。变量还应遵守以下某条著名的命名规则。

7.2.3.1 Camel 标记法

首字母是小写的，接下来的字母都以大写字符开头。

7.2.3.2 Pascal 标记法

首字母是大写的，接下来的字母都以大写字符开头。

7.2.3.3 匈牙利类型标记法

在以 Pascal 标记法命名的变量前附加一个小写字母（或小写字母序列） ，说明该变量的类型。

例如， i 表示整数， s 表示字符串， a 表示数组， b 表示布尔值， f 表示浮点数， fn 表示函数， o 表示对象， re 表示正则表达式， v 表示变量。

7.2.4 变量声明不是必须的

ECMAScript 另一个有趣的方面（也是与大多数程序设计语言的主要区别） ，是在使用变量之前不必声明。例如：

```
var sTest = "hello "; sTest2 = sTest + "world"; trace(sTest2);
```

在上面的代码中， 首先， sTest 被声明为字符串类型的值 "hello" 。接下来的一行， 用变量 sTest2 把 sTest 与字符串 "world" 连在一起。变量 sTest2 并没有用 var 运算符定义，这里只是插入了它，就像已经声明过它一样。

ECMAScript 的解释程序遇到未声明过的标识符时，用该变量名创建一个全局变量，并将其初始化为指定的值。

这是该语言的便利之处，不过如果不能紧密跟踪变量，这样做也很危险。最好的习惯是像使用其他程序设计语言一样，总是声明所有变量。

7.3 ECMAScript 关键字

ECMA-262 定义了 ECMAScript 支持的一套 关键字（ keyword ）。

这些关键字标识了 ECMAScript 语句的开头和 / 或结尾。根据规定，关键字是保留的，不能用作变量名或函数名。

下面是 ECMAScript 关键字的完整列表：

break
case
catch
continue
default
delete
do
else
finally
for
function
if
in

instanceof
new
return
switch
this
throw
try
typeof
var
void
while
with

注意 ：如果把关键字用作变量名或函数名，可能得到诸如 " Identifier Expected " （应该有标识符、期望标识符）这样的错误消息。

7.4 ECMAScript 保留字

ECMA-262 定义了 ECMAScript 支持的一套 保留字（ reserved word ）。

保留字在某种意思上是为将来的关键字而保留的单词。因此保留字不能被用作变量名或函数名。

ECMA-262 第三版中保留字的完整列表如下：

abstract
boolean
byte
char
class
const
debugger
double
enum
export
extends
final
float
goto
implements
import
int
interface
long
native
package
private
protected
public
short
static
super
synchronized
throws
transient
volatile

注意 ：如果将保留字用作变量名或函数名，那么除非将来的浏览器实现了该保留字，否则很可能收不到任何错误消息。当浏览器将其实现后，该单词将被看做关键字，如此将出现关键字错误。

7.5 ECMAScript 位运算符

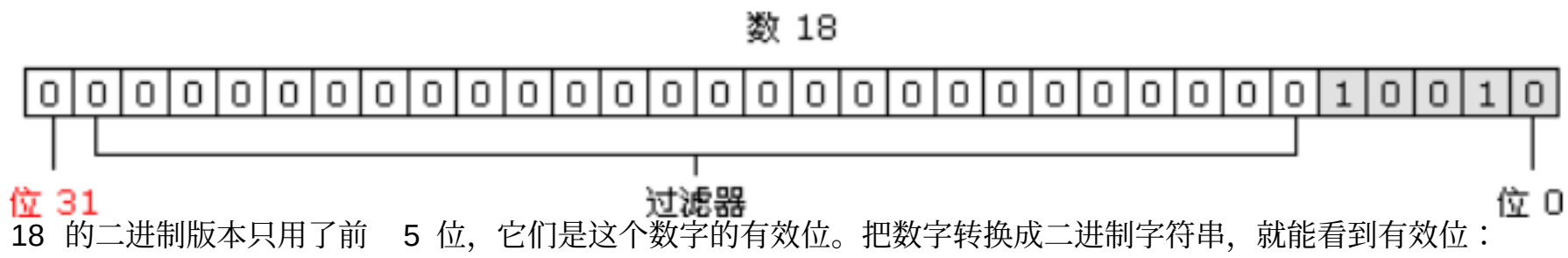
位运算符是在数字底层（即表示数字的 32 个数位）进行操作的。

7.5.1 整数

ECMAScript 整数有两种类型， 即有符号整数 （允许用正数和负数） 和无符号整数 （只允许用正数） 。在 ECMAScript 中，所有整数字面量默认都是有符号整数，这意味着什么呢？

有符号整数使用 31 位表示整数的数值，用第 32 位表示整数的符号， 0 表示正数， 1 表示负数。数值范围从

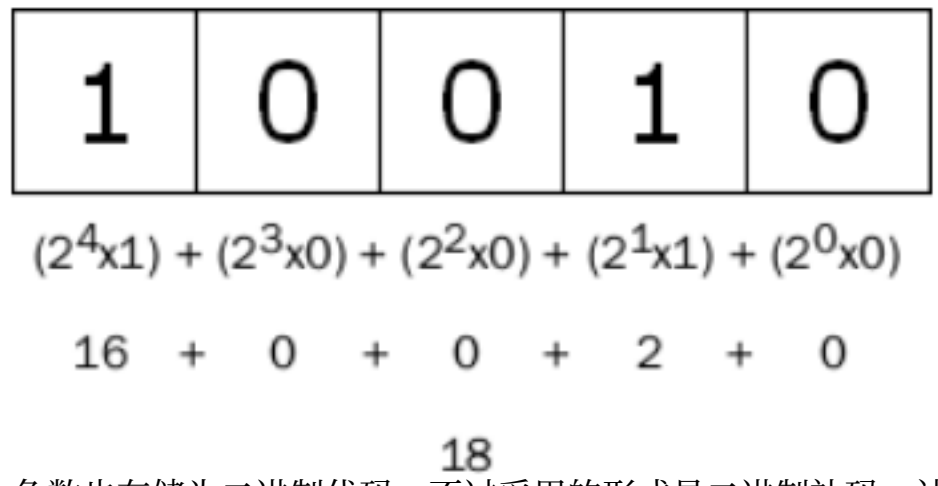
-2147483648 到2147483647。
可以以两种不同的方式存储二进制形式的有符号整数，一种用于存储正数，一种用于存储负数。正数是以真二进制形式存储的，前 31位中的每一位都表示 2的幂，从第 1位（位 0）开始，表示 2 0，第 2位（位 1）表示 2 1。没用到的位用 0填充，即忽略不计。例如，下图展示的是数 18的表示法。



18 的二进制版本只用了前 5 位，它们是这个数字的有效位。把数字转换成二进制字符串，就能看到有效位：

```
var iNum = 18; trace(iNum.toString(2));                    /*  输出 "10010"*/
```

这段代码只输出 "10010" ，而不是 18 的 32 位表示。 其他的数位并不重要， 因为仅使用前 5 位即可确定这个十进制数值。如下图所示：



负数也存储为二进制代码，不过采用的形式是二进制补码。计算数字二进制补码的步骤有三步：

1. 确定该数字的非负版本的二进制表示（例如，要计算 -18 的二进制补码，首先要确定 18 的二进制表示）
2. 求得二进制反码，即要把 0 替换为 1，把 1 替换为 0
3. 在二进制反码上加 1

要确定 -18 的二进制表示，首先必须得到 18 的二进制表示，如下所示：

```
0000 0000 0000 0000 0000 0000 0001 0010
```

接下来，计算二进制反码，如下所示：

```
1111 1111 1111 1111 1111 1111 1110 1101
```

最后，在二进制反码上加 1，如下所示：

```
1111 1111 1111 1111 1111 1111 1110 1101 + 1 = 1111 1111 1111 1111 1111 1111 1110 1110
```

因此， -18 的二进制表示即 1111 1111 1111 1111 1111 1111 1110 1110 。记住，在处理有符号整数时，开发者不能访问 31 位。
有趣的是， 把负整数转换成二进制字符串后， ECMAScript 并不以二进制补码的形式显示， 而是用数字绝对值的标准二进制代码前面加负号的形式输出。例如：

```
var iNum = -18; alert(iNum.toString(2));                    //  输出 "-10010"
```

这段代码输出的是 "-10010" ，而非二进制补码，这是为避免访问位 31。为了简便， ECMAScript 用一种简单的方式处理整数，使得开发者不必关心它们的用法。
另一方面，无符号整数把最后一位作为另一个数位处理。在这种模式中，第 32位不表示数字的符号，而是值 2 31。由于这个额外的位， 无符号整数的数值范围为 0到 4294967295。对于小于 2147483647的整数来说， 无符号整数看来与有符号整数一样，而大于 2147483647的整数则要使用位 31（在有符号整数中，这一位总是 0）。
把无符号整数转换成字符串后，只返回它们的有效位。
注意 ：所有整数字面量都默认存储为有符号整数。只有 ECMAScript 的位运算符才能创建无符号整数。

7.5.2 位运算 NOT

位运算 NOT 由否定号（ ~）表示，它是 ECMAScript 中为数不多的与二进制算术有关的运算符之一。
位运算 NOT 是三步的处理过程：
1. 把运算数转换成 32 位数字
2. 把二进制数转换成它的二进制反码
3. 把二进制数转换成浮点数

例如：

```
var iNum1 = 25;                    /*25  等于  00000000000000000000000000011001 */
var iNum2 = ~iNum1;              /* 转换为  11111111111111111111111111100110 */
trace(iNum2);                /*  输出 "-26"*/
```

位运算 NOT实质上是对数字求负，然后减 1，因此 25 变-26 。用下面的方法也可以得到同样的方法：

```
var iNum1 = 25; var iNum2 = -iNum1 -1; trace(iNum2);                    /*  输出 -26*/
```

7.5.3 位运算 AND

位运算 AND由和号（ &）表示，直接对数字的二进制形式进行运算。它把每个数字中的数位对齐，然后用下面的规则对同一位置上的两个数位进行 AND运算：

第一个数字中的数位	第二个数字中的数位	结果
1	1	1
1	0	0
0	1	0
0	0	0

例如，要对数字 25 和3 进行 AND运算，代码如下所示：

```
var iResult = 25 & 3; alert(iResult);                    //  输出 "1"
```

25 和3 进行 AND运算的结果是 1。为什么？分析如下：

```
25 = 0000 0000 0000 0000 0000 0000 0001 1001
3 = 0000 0000 0000 0000 0000 0000 0000 0011
AND = 0000 0000 0000 0000 0000 0000 0000 0001
```

可以看出，在 25 和3 中，只有一个数位（位 0）存放的都是 1，因此，其他数位生成的都是 0，所以结果为 1。

7.5.4 位运算 OR

位运算 OR 由符号（ | ）表示，也是直接对数字的二进制形式进行运算。在计算每位时， OR 运算符采用下列规则：

第一个数字中的数位	第二个数字中的数位	结果
1	1	1
1	0	1
0	1	1
0	0	0

仍然使用 AND运算符所用的例子，对 25 和3 进行 OR运算，代码如下：

```
var iResult = 25 | 3; alert(iResult);                    //  输出 "27"
```

25 和 3 进行 XOR 运算的结果是 26：

```
25 = 0000 0000 0000 0000 0000 0000 0001 1001
3 = 0000 0000 0000 0000 0000 0000 0000 0011
OR = 0000 0000 0000 0000 0000 0000 0001 1011
```

可以看出，在两个数字中，共有 4 个数位存放的是 1，这些数位被传递给结果。二进制代码 11011 等于 27。

7.5.5 位运算 XOR

位运算 XOR由符号（ ^）表示，当然，也是直接对二进制形式进行运算。 XOR不同于 OR，当只有一个数位存放的是1 时，它才返回 1。真值表如下：

第一个数字中的数位	第二个数字中的数位	结果

1	1	0
1	0	1
0	1	1
0	0	0

对 25 和 3 进行 XOR运算，代码如下：

```
var iResult = 25 ^ 3; alert(iResult);                    //  输出 "26"
```

25 和3 进行 XOR运算的结果是 26：

```
25 = 0000 0000 0000 0000 0000 0000 0001 1001

3 = 0000 0000 0000 0000 0000 0000 0000 0011

XOR = 0000 0000 0000 0000 0000 0000 0001 1010
```

可以看出，在两个数字中，共有 4 个数位存放的是 1，这些数位被传递给结果。二进制代码 11010 等于 26。

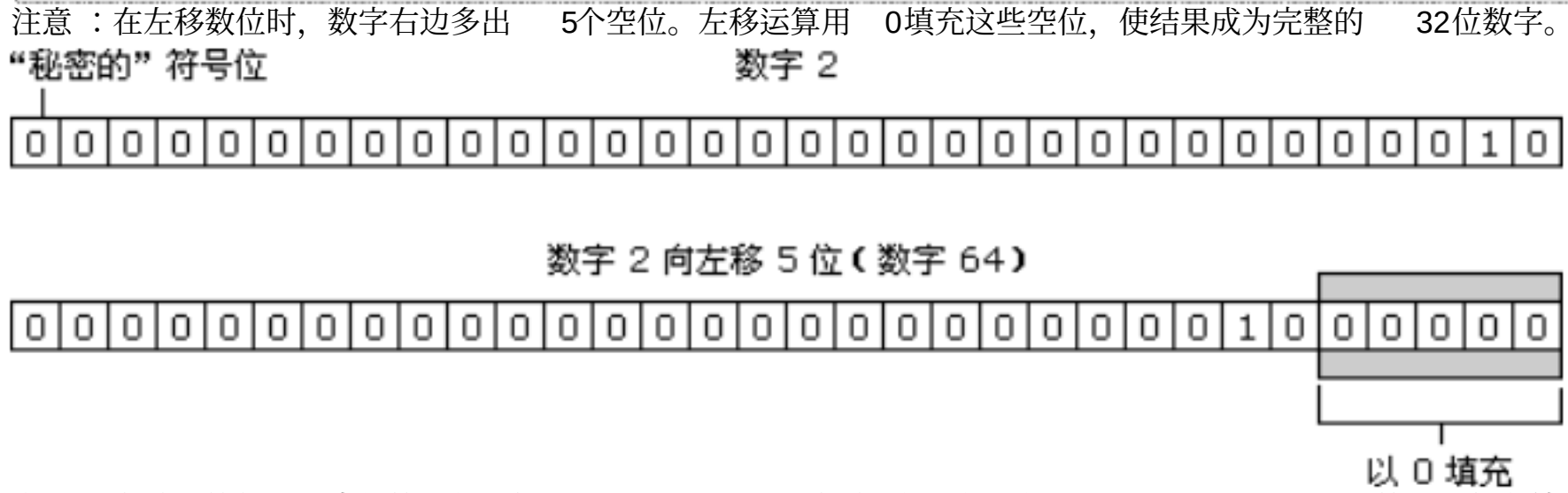
7.5.6 左移运算

左移运算由两个小于号表示（ <<)。它把数字中的所有数位向左移动指定的数量。例如，把数字 2（等于二进制中的10）左移 5 位，结果为 64（等于二进制中的 1000000）：

```
var iOld = 2;                /*  等于二进制  10 */

var iNew = iOld << 5;            /*  等于二进制  1000000 十进制  64*/
```

注意 ：在左移数位时，数字右边多出 5个空位。左移运算用 0填充这些空位，使结果成为完整的 32位数字。



注意 ：左移运算保留数字的符号位。例如，如果把 -2 左移 5位，得到的是 -64 ，而不是 64。“ 符号仍然存储在第 32 位中吗？ ” 是的， 不过这在 ECMAScript 后台进行，开发者不能直接访问第 32个数位。 即使输出二进制字符串形式的负数，显示的也是负号形式（例如， -2 将显示 -10 。）

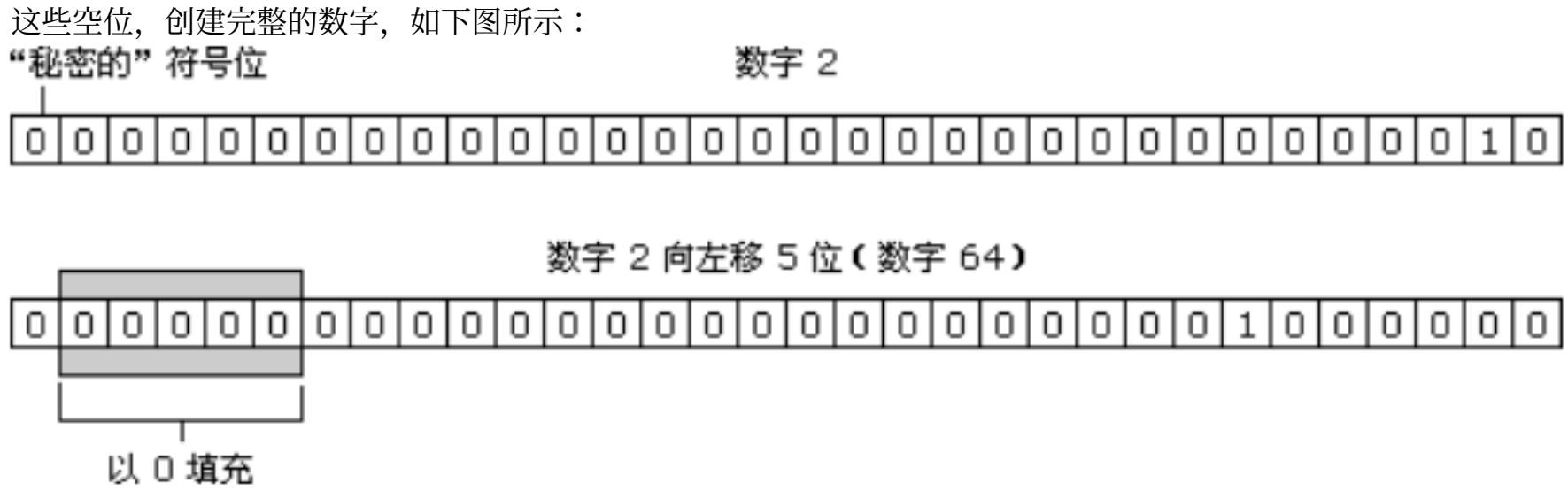
7.5.7 有符号右移运算

有符号右移运算符由两个大于号表示（ >>)。它把 32 位数字中的所有数位整体右移，同时保留该数的符号（正号或负号）。有符号右移运算符恰好与左移运算相反。例如，把 64 右移 5 位，将变为 2：

```
var iOld = 64;               /*  等于二进制  1000000 */

var iNew = iOld >> 5;            /*  等于二进制  10 十进制  2*/
```

同样，移动数位后会造成空位。这次，空位位于数字的左侧，但位于符号位之后。 ECMAScript 用符号位的值填充这些空位，创建完整的数字，如下图所示：



7.5.8 无符号右移运算

无符号右移运算符由三个大于号（ >>>）表示，它将无符号 32 位数的所有数位整体右移。对于正数，无符号右移运算的结果与有符号右移运算一样。
用有符号右移运算中的例子，把 64 右移 5 位，将变为 2：

```
var iOld = 64;                  //  等于二进制  1000000
 var iNew = iOld >>> 5;         //  等于二进制  10 十进制  2
```

对于负数，情况就不同了。
无符号右移运算用 0 填充所有空位。对于正数，这与有符号右移运算的操作一样，而负数则被作为正数来处理。
由于无符号右移运算的结果是一个 32 位的正数，所以负数的无符号右移运算得到的总是一个非常大的数字。 例如，如果把 -64 右移 5 位，将得到 134217726。如果得到这种结果的呢？
要实现这一点， 需要把这个数字转换成无符号的等价形式 （尽管该数字本身还是有符号的） ，可以通过以下代码获得这种形式：

```
var iUnsigned64 = -64 >>> 0;
```

然后，用 Number 类型的 toString() 获取它的真正的位表示，采用的基为 2：

```
alert(iUnsigned64.toString(2));
```

这将生成 11111111111111111111111111000000，即有符号整数 -64 的二进制补码表示，不过它等于无符号整数 4294967232。
出于这种原因，使用无符号右移运算符要小心。

7.6 ECMAScript 一元运算符

一元运算符只有一个参数，即要操作的对象或值。它们是 ECMAScript 中最简单的运算符。

7.6.1 前增量 /前减量运算符

直接从 C（和 Java）借用的两个运算符是前增量运算符和前减量运算符。
所谓前增量运算符，就是数值上加 1，形式是在变量前放两个加号（ ++）：

```
var iNum = 10; ++iNum;
```

第二行代码把 iNum 增加到了 11，它实质上等价于：

```
var iNum = 10; iNum = iNum + 1 ;
```

同样，前减量运算符是从数值上减 1，形式是在变量前放两个减号（ -- ）：

```
var iNum = 10; --iNum;
```

在这个例子中，第二行代码把 iNum 的值减到 9。
在使用前缀式运算符时，注意增量和减量运算符都发生在计算表达式之前。考虑下面的例子：

```
var iNum = 10; --iNum; alert(iNum);             /*  输出 "9" */
trace(--iNum);              /*  输出 "8" */
trace(iNum);        /*  输出 "8" */
```

第二行代码对 iNum 进行减量运算， 第三行代码显示的结果是（ "9" ）。第四行代码又对 iNum 进行减量运算， 不过这次前减量运算和输出操作出现在同一个语句中， 显示的结果是 "8" 。为了证明已实现了所有的减量操作， 第五行代码又输出一次 "8" 。
在算术表达式中，前增量和前减量运算符的优先级是相同的，因此要按照从左到右的顺序计算之。例如：

```
var iNum1 = 2; var iNum2 = 20; var iNum3 = --iNum1 + ++iNum2;                   /*  等于 "22" */
var iNum4 = iNum1 + iNum2;              /*  等于 "22"*/
```

在前面的代码中， iNum3 等于 22，因为表达式要计算的是 1 + 21 。变量 iNum4 也等于 22，也是 1 + 21 。

7.6.2 后增量 /后减量运算符

还有两个直接从 C（和 Java）借用的运算符，即后增量运算符和后减量运算符。
后增量运算符也是给数值上加 1，形式是在变量后放两个加号（ ++）：

```
var iNum = 10; iNum++;
```

不出所料，后减量运算符也是从数值上减 1，形式为在变量后加两个减号（ -- ）：

```
var iNum = 10; iNum--;
```

第二行代码把 iNum 的值减到 9。 与前缀式运算符不同的是，后缀式运算符是在计算过包含它们的表达式后才进行增量或减量运算的。考虑以下的例子：

```
var iNum = 10; iNum--; alert(iNum);                /*  输出 "9" */
 trace(iNum--);              /*  输出 "9" */
 trace(iNum);  /*        输出 "8"*/
```

与前缀式运算符的例子相似，第二行代码对 iNum 进行减量运算，第三行代码显示结果（ "9" ）。第四行代码继续显示 iNum 的值，不过这次是在同一语句中应用减量运算符。由于减量运算发生在计算过表达式之后，所以这条语句显示的数是 "9" 。执行了第五行代码后， alert 函数显示的是 "8" ，因为在执行第四行代码之后和执行第五行代码之前，执行了后减量运算。

在算术表达式中，后增量和后减量运算符的优先级是相同的，因此要按照从左到右的顺序计算之。例如：

```
var iNum1 = 2; var iNum2 = 20; var iNum3 = iNum1-- + iNum2++;                    //  等于 "22"
 var iNum4 = iNum1 + iNum2;                //  等于 "22"
```

在前面的代码中， iNum3 等于 22，因为表达式要计算的是 2 + 20。变量 iNum4 也等于 22，不过它计算的是 1 + 21，因为增量和减量运算都在给 iNum3 赋值后才发生。

7.6.3 一元加法和一元减法

大多数人都熟悉一元加法和一元减法，它们在 ECMAScript 中的用法与您高中数学中学到的用法相同。

一元加法本质上对数字无任何影响：

```
var iNum = 20; iNum = +iNum; trace(iNum);  /*            输出 "20"*/
```

这段代码对数字 20 应用了一元加法，返回的还是 20。

尽管一元加法对数字无作用，但对字符串却有有趣的效果，会把字符串转换成数字。

```
var sNum = "20"; trace(typeof sNum);          /*  输出 "string" */
var iNum = +sNum; trace(typeof iNum);         /*  输出 "number"*/
```

这段代码把字符串 "20" 转换成真正的数字。当一元加法运算符对字符串进行操作时，它计算字符串的方式与 parseInt() 相似，主要的不同是只有对以 "0x" 开头的字符串 （表示十六进制数字） ，一元运算符才能把它转换成十进制的值。因此，用一元加法转换 "010" ，得到的总是 10，而 "0xB" 将被转换成 11。

另一方面，一元减法就是对数值求负（例如把 20 转换成 -20 ）：

```
var iNum = 20; iNum = -iNum; trace(iNum);                /*  输出 "-20"*/
```

与一元加法运算符相似，一元减法运算符也会把字符串转换成近似的数字，此外还会对该值求负。例如：

```
var sNum = "20"; trace(typeof sNum);          /*  输出 "string" */
var iNum = -sNum; trace(iNum);                /*  输出 "-20" */
trace(typeof iNum);           /*  输出 "number"*/
```

在上面的代码中， 一元减法运算符将把字符串 "-20" 转换成 -20（一元减法运算符对十六进制和十进制的处理方式与一元加法运算符相似，只是它还会对该值求负） 。

8.Function

8.1 trace

function tracex(s:String):void
添加指定内容至日志中

8.1.1 参数

s:String — 要添加的内容

8.2 clear

function clear():void
清空日志内容

8.3 getTimer

function getTimer():int
获取从启动播放器到现在经过的毫秒数

8.4 parseInt

public function parseInt(str:String, radix:uint = 0):Number
将字符串转换为整数。如果参数中指定的字符串不能转换为数字，则此函数返回 NaN。以 0x 开头的字符串被解释为十六进制数字。以 0 开头的整数不会被解释为八进制数字。必须指定 8 的基数才能解释为八进制数字。有效整数前面的空白和 0 以及后面的非数字字符将被忽略。

8.4.1 参数

str:String — 要转换为整数的字符串。
radix:uint (default = 0) — 表示要分析的数字的基数（基）的整数。合法值为 2 到 36 。

8.4.2 返回

Number — 一个数字或 NaN（非数字）。

8.5 parseInt

public function parseFloat(str:String):Number
将字符串转换为浮点数。 此函数读取或分析 并返回字符串中的数字， 直到此函数遇到不是初始数字一部分的字符。如果字符串不是以可以分析的数字开头， parseFloat() 将返回 NaN。有效整数前面的空白将被忽略， 有效整数后面的非数字字符也将被忽略。

8.5.1 参数

str:String — 要读取并转换为浮点数的字符串。

8.5.2 返回

Number — 一个数字或 NaN（非数字）。

8.6 timer

public function timer(closure:Function, delay:Number):uint
在指定的延迟（以毫秒为单位）后运行指定的函数。

8.6.1 参数

closure:Function — 要执行的函数的名称。不要包括引号或圆括号，并且不要指定要调用的函数的参数。例如，使用 functionName ，而不要使用 functionName() 或 functionName(param) 。
delay:Number — 执行函数之前的延迟时间（以毫秒为单位） 。

8.6.2 返回

uint — 超时进程的唯一数字标识符。使用此标识符可通过调用 clearTimeout() 方法取消进程。

8.7 interval

public function interval(closure:Function, delay:Number, times: Number=1):Timer
以指定的间隔（以毫秒为单位）运行函数。

8.7.1 参数

closure:Function — 要执行的函数的名称。不要包括引号或圆括号，并且不要指定要调用的函数的参数。例如，使用 functionName ，而不要使用 functionName() 或 functionName(param) 。
delay:Number — 间隔（以毫秒为单位） 。
times:Number — 运行次数。

8.7.2 返回

Timer — 超时进程的唯一标识符。使用此标识符可通过调用 timer.stop() 方法取消进程。

8.8 foreach

8.8.1 参数

public function foreach(loop:Object,f:Function):void
遍历指定 Object

8.8.2 参数

loop :Object — 被遍历的 Object
f :Function — 遍历回调函数

8.8.3 回调函数定义

function foreachCallback(key:String,value:*):void;
====回调函数参数定义 ====
key :String — 键值名
value :* — 值

8.8.4 示例

```
a={a:"b",b:"c"};
foreach(a,function(key,obj){
  trace(key+":"+obj);
});
```

8.9 clone

function clone(object:Object):Object
复制指定 Object
注意：此功能无法复制函数

8.9.1 参数

object :Object — 被复制的 Object

8.9.2 示例

```
var a={test:2};
 var b=clone(a);
a.test=1;
trace(b.test);
```

8.10 load

function load(library:String,onComplete:Function):void
加载外部库

8.10.1 参数

library :String — 库名称
onComplete :Function — 加载完成时执行的回调函数

9.绘图 API 使用基础知识

9.1 使用绘图 API 简介

绘图 API 是 Script 中的一项内置功能的名称，您可以使用该功能来创建矢量图形（直线、曲线、形状、填充和渐变），并在屏幕上显示它们。 Display.createGraphic.graphics 类提供了这一功能。
如果刚刚开始学习使用代码进行绘制，可以使用 Graphics 类中包含的几种方法来简化绘制常见形状（如圆、椭圆、矩形以及带圆角的矩形） 的过程。 您可以将它们作为空线条或填充形状进行绘制。 当您需要更高级的功能时，还可以使用 Graphics 类中包含的用于绘制直线和二次贝塞尔曲线的方法，您可以将这些方法与 Math 类中的三角函数配合使用来创建所需的任何形状。

9.2 常见绘图 API 任务

以下是您可能需要在 Script 中使用绘图 API 完成的任务，本章对这些任务进行了介绍：
定义线条样式和填充样式以绘制形状
绘制直线和曲线
使用方法来绘制形状（如圆、椭圆和矩形）
使用渐变线条和填充进行绘制
定义矩阵以创建渐变
将三角函数与绘图 API 配合使用
将绘图 API 与动画相结合

9.3 重要概念和术语

以下参考列表包含将会在本章中遇到的重要术语：
锚点 (Anchor point) ：二次贝塞尔曲线的两个端点之一。
控制点 (Control point) ：该点定义了二次贝塞尔曲线的弯曲方向和弯曲量。弯曲的线绝不会到达控制点；但是，曲线就好像朝着控制点方向进行绘制的。
坐标空间 (Coordinate space) ：显示对象中包含的坐标（其子元素所在的位置）的图形。
填充 (Fill) ：形状内的实心部分，它包含一条用颜色填充的线条，或者整个形状都没有轮廓。
渐变 (Gradient) ：此颜色是指从一种颜色逐渐过渡到一种或多种其它颜色（相对于纯色而言） 。
点 (Point) ：坐标空间中的一个位置。在 Script 使用的二维坐标系中，点是按其 x 轴和 y 轴位置（点坐标）来定义的。
二次贝塞尔曲线 (Quadratic Bé zier curve) ：一种由特定数学公式定义的曲线类型。在这种类型的曲线中，曲线形状是根据锚点（曲线端点）和控制点（定义曲线的弯曲方向和弯曲量）的位置计算的。
缩放 (Scale) ：相对于原始大小的对象大小。用作动词时，对象缩放是指伸展或缩小对象以更改其大小。
笔触 (Stroke) ：形状的轮廓部分，它包含一条用颜色填充的线条，或未填充的形状的多个线条。
平移 (Translate) ：将点的坐标从一个坐标空间更改为另一个坐标空间。
X 轴 (X axis) ： Script 使用的二维坐标系中的水平轴。
Y 轴 (Y axis) ： Script 使用的二维坐标系中的垂直轴。

10.CommentData

10.1 txt

txt :String
弹幕内容

10.2 time

time :Number
播放时间 (以秒为单位)

10.3 color

color :uint
色彩

10.4 pool

pool :int
弹幕池号码

10.5 mode

mode:int
弹幕模式

10.6 fontSize

fontSize :uint
弹幕字体大小

11.Global

11.1 _set

public function _set(key:String,val:*):void
用于设置保存的变量值

11.1.1 参数

key :String 一键值
val :* 一保存的值

11.2 _get

public function _get(key:String):*
用于取出保存的键字

11.2.1 参数

key :String 一键值
别名：— _

11.2.2 返回

保存的值

12.Shape

12.1 构造函数

public function createShape(Object)
创建新的 Shape 对象。

12.2 graphics

graphics:Graphics [只读]
指定属于该 Shape 对象的 Graphics 对象，可通过此对象执行矢量绘图命令。

13.ScriptManager

脚本资源管理器

13.1 clearTimer

public function clearTimer():void
终止正在运行的所有定时器

14. 不兼容 ECMAScript 部份说明

14.1 for..in

for..in 的实现改为以 Function#foreach 函数完成

```
a={a:"b",b:"c"};
foreach(a,function(key,obj){
  trace(key+":"+obj);
});
```

14.1.1 示例

14.2 函数默认值指定

使用 if 判断是否为 undefined 后指定值

14.2.1 示例

```
function a(a){
  if (a==undefined) a=1;
  trace(a);
}
a();
```

14.3 prototype

无法使用 prototype 关键字
使用以下方式创建类

14.3.1 示例

```
function newObject(){
   var newObj = {};
   newObj.init=function(){ this.initValue=1; };
   newObj.output=function(){ trace(this.initValue); };
   return newObj;
 }

 var test = newObject();
 var test2 = newObject();
 test.init();
 test.output();
 test2.output();
```

14.4 new

目前不支持使用　new 关键字　复制使用函数　Function#clone
创建 Object 请使用 var tp={};
创建 Array 请使用 var tp=[];

以下示例将复制一个类

```
var a={test:2};
var b=clone(a);
a.test=1;
trace(b.test);
```

14.4.1 示例

14.5 函数返回的对象不能继续调用函数

由于函数与成员的优先级问题　　调用需要使用括号

```
var a={};
a.b=function(){return a;};
(a.b()).b();
```

14.5.1 示例