第一章 了解 ActionScript3.0

1、 ActionScript3.0 的一些新特性：

- 引入显示列表的概念。显示列表用于创建、管理显示对象的层次结构，任何 Flash 应用程序实际上就是显示列表。在显示列表中，采用新的深度机制来管理显示对象的显示层次，使显示对象的深度管理更加人性化。
- 使用新的事件模型。ActionScript3.0 中的事件模型与第 2 用户界面组件有点类似，是采用观察者模式设计的。新增了事件流、默认行为等功能，很多在 ActionScript2.0 中难以实现的功能，使用 ActionScript3.0 将会非常简单。
- 引入了 E4X，使得操作 XML 更加方便、快捷。在以往版本的 ActionScript 中，使用 XML 对象前，需要将其转换为数组或对象，而 ActionScript3.0 可直接操作 XML 对象。
- 支持正则表达式。正则表达式在查找和替换模式方面有很大的优势，以往需要几十行代码实现的功能，使用正则表达式只需几行。

2、 一般来说，舞台(Stage)是用来显示 Flash 元素的平台，而主时间轴(Main Timeline)则用来控制 Flash 元素的显示，它们在编程中扮演相当重要的角色。舞台是放置显示对象的最终容器，所有的显示对象都直接或间接地包含在舞台中。舞台的名字是 stage，它是 Stage 类的对象。通过点语法可以读取舞台容器中的非孤立变量值，但由于 Flash 为这些变量规定了只读属性，所以暂时无法通过类似“stage.stageWidth=800”这样的语句为舞台中的变量重新赋值。Stage 类不是动态类，不能向舞台 stage 动态地添加属性。主时间轴的名字是 root，它是由 MainTimeline 类创建的对象。主时间轴其实是一个特殊的影片剪辑实例，它的图层、帧的使用方法与影片剪辑实例完全相同。写在帧中的代码都属于主时间轴。可以这样认为，写在帧中的变量都在 MainTimeline 类中，而 root 是由 MainTimeline 类创建的对象，这些变量都成了 root 的属性，因此可用“root.属性名”的形式来访问这些变量。MainTimeline 类是动态类，在主时间轴上定义的变量会被添加到 MainTimeline 类中，成为主时间轴的属性，而通过 root 动态添加的属性并没有添加到 MainTimeline 类中。

3、 舞台和主时间轴都是显示对象，它们可以通过系统定义的变量 stage 和 root 来控制，而有些显示对象则比较特殊，可以通过【属性】面板定义的变量来控制。实例的属性分为读写属性和只读属性两种。

4、 在进行时间轴编程时，需要了解时间轴帧的播放顺序、图层的执行顺序。时间轴上代码的执行顺序为：按照包含这些代码的帧在时间轴上出现的先后顺序排列，并且在默认情况下，帧是循环播放的。不同图层的代码的执行顺序为从上到下。

5、 虽然 Flash AS3.0 支持把代码放在时间轴中，但在实际应用中，如果把很多的代码放在时间轴中，会导致代码难以管理或比较混乱等问题。或者说，用类来组织大量的代码更加合适，类代码只能放在 as 文件中。

6、 对于一个代码较多的应用程序，可以考虑把代码分成几个大的功能模块，每个模块代码放在一个 as 文件中(模块化程序设计)，通过 include 导入到帧中，实现的效果与写在帧上是一样的，但代码的维护代价更低。如果 as 文件与 fla 文件不在同一目录，则应带上目录名。例如：include“../singleline.as”

7、 前面编写代码可以说是基于过程编写的，代码的扩展性、复用性等较低。
类编写完之后，要通过 import 导入到 fla 文件中。如果类与 fla 文件不在同一目录，一般要通过【文件】|【发布设置】添加新路径。类路径可以理解为保存代码的目录，只要在 Flash 里设置了类路径，保存在类路径下的代码就可以被 Flash 导入进来。

不要把 include 导入(我觉得可以译为”包含”)与 import 导入混合起来，include 的代码与写在

帧上的代码是一样的。而 import 导入的是类，类的编写与一般的代码编写不同。

第二章 简单数据类型

1、 第一章提到的变量 stage 的类型为 Stage 类型，即类是一种数据类型。当通过【属性】面板定义变量时，这个变量的类型也被自动声明了。例如，定义影片剪辑实例的变量时，变量的类型为 MovieClip 类型。

2、 每一种简单数据类型都与一个类相关联，类的类名就是数据类型的名字。例如，int 类型与 int 类相关联，这种类一般称为包装类。与其他传统的编程语言不同，AS3.0 中包装类对象并不是复杂数据类型，而是简单数据类型，所以下面两种方式定义的变量都是一样的，speed1 和 speed2 都属于简单数据类型：

```
var speed1:int = 4;
var speed2:int = new int(4);
```

3、 任何一个简单数据都可以看成是类的实例。例如，可以直接对数据进行属性和方法的调用：trace((1.2345).toFixed()); (1.2345)中的括号是不能少的，因为这个数据中有个小数点，有了括号就可以区分调用方法的点语法。

4、 在 AS1.0 和 AS2.0 中，可以用同一个变量名重复定义变量，但在 AS3.0 中，这是不允许的。在基于时间轴编程中，如果在第 1 帧中定义一个变量，又在后面的帧中定义一个同名的变量，AS3.0 也会作为“重复定义变量”处理。解决的方法很简单，就是在程序的开头先定义变量，然后在不同的地方就可以使用这个变量。

5、 在时间轴的帧上写代码时，不能使用 MovieClip 类的属性名和方法名作为变量名。如 x、name、width 和 play、stop 等。因为时间轴有一个关联的类，这个类继承了 MovieClip 类。

6、 变量的默认值：

比较类型	*AS2.0*	*AS3.0*
Number	Undefined	NaN
String	Undefined	null
Boolean	Undefined	false
int		0
uint		0

7、 在时间轴上定义变量有 3 种方式：使用 var 定义变量；使用 this 动态添加变量；直接定义变量。在这 3 种变量定义中，应该使用 var 来定义变量，因为它的执行效率最高。

8、 如果在定义变量时，没有声明变量的数据类型，系统会以“*”类型进行处理。“*”数据类型是 AS3.0 中新增的数据类型，可以与 Object 类型配合使用，用来声明不确定的数据类型。

第三章 复杂数据类型

1、 new 运算符可以创建类的实例，它可以通过调用类的构造函数来创建一个实例，完成分配空间等任务，并且返回一个实例引用。

2、 只能通过实例来访问的属性和方法，分别称为实例属性和实例方法；只能通过类名来访问的属性和方法，分别称为类属性和类方法，也称为静态属性和静态方法。也就是说，它们是属于类的，而不是属于实例的。

3、 AS3.0 中的 delete 发生了很大的变化，它不能删除使用 var 定义的变量，只能删除动态创建的变量。而对于数组，delete 只能删除数组的元素，不能删除索引。要真正删除数组的元素，必须使用 splice()方法。

4、 typeof()函数返回数据的类型，所有的数字类型输出的都是 number，而对于复杂数据类型变量，输出的都是 object，所以 typeof()函数并不能给出非常具体的数据类型信息。

5、 如果要检验变量的准确数据类型，可以使用 is 运算符。is 用来判断实例是否属于某类型，也可以判断类的继承关系，包括对于接口的继承执行，如果类型匹配，返回 true，否则返回 false。如果只知道变量，却想获得变量的数据类型信息，可用 getQualifiedClassName()函数。该函数返回的是包括包名在内的完整信息，这个信息是一个字符串，根据字符串用 getDefinitionByName()函数就可以得到类信息。

6、 除了使用 Array 类的构造函数来创建数组外，还可以使用“[]”运算符来创建数组。Flash 中数组元素的类型可以不同。由于数组属于复杂数据类型，它是通过引用来操作的。

7、 this 关键字是对当前对象的引用。例如在主时间轴中，this 引用的是主时间轴，可以用下面的代码来测试：//输出 true　　trace(this==root);

8、 点语法可以用数组运算符“[]”来代替，因此访问实例的属性和方法也可以使用数组运算符。访问的规则是把属性名作为字符串放在数组运算符中。如：

```
varmc:MovieClip = new MovieClip();
trace(mc[“x”]);
mc[“play”]();
```

使用数组运算符的好处在于访问的动态性，即可以把变量名、属性名、方法名保存在字符串变量中，通过变量和数组运算符就可以访问属性和方法。有时候点语法不能很好工作时，就可用数组运算符代替。如在严谨模式下，尝试给 root 动态添加属性时，会提示错误信息，如果用数组运算符就可以通过编译器的检查。

第四章 流程控制

1、 for in 循环通常只出现在对象的属性中，可以用一个变量名称来搜索对象，然后执行每个对象中的陈述式。它的一般形式为：for (变量 in 对象或数组) {循环体}

2、 for each in 是 AS3.0 新增的循环语句，其语法与 for in 循环语句类似，格式为：
for each(变量 in 对象或数组) {循环体}

3、 AS3.0 引入了标签，一种新的用来关联循环程序块的标识符，标签用来做循环体中 break 和 continue 命令的目标对象。在循环嵌套的情况下，如果要从内循环中跳出整个循环是做不到的，因为 break 命令只能退出当前的程序块。

4、 除了利用上述的循环控制语句实现程序的循环结构外，在 Flash 中还有一种特殊的循环结构——帧循环。这是为了适应 Flash 动画时间轴的特殊程序执行环境，通过切换帧或者定时重复执行某段程序而产生的一种循环结构类型。Flash 中的帧在默认情况下是循环执行的，利用 gotoAndPlay()函数可以实现简单的帧循环。

5、 Flash 提供的 enterframe 事件可以以帧频的速度执行代码，而且代码可以集中在一帧，enterframe 事件是制作运动效果的较佳选择。使用 enterframe 事件可以实现代码的重复执行，但它执行的速度与帧频有关，所以可能也会出现动画不流畅的情况。当不需要使用 enterframe 事件时，一定要用 removeEventListener()函数来删除 enterframe 事件。

6、 帧循环是以帧频速度的一半重复执行代码，而 enterframe 事件是与帧频一样的速度重复执行代码，所以它们都会受到帧频的影响。

7、 setInterval(函数名，时间间隔，函数参数);　　每隔一定的时间，就调用函数。
clearInterval();　　移除 setInterval()　setTimeout(函数名,时间间隔,函数参数);
setTimeout()函数的使用方法与 setInterval()函数的使用方法相同，它的作用也是每隔一定的时间就调用函数，相应的有 clearTimeout()函数移除。

8、 AS3.0 新增了 Timer 类来实现间隔调用程序，Timer 类封装了许多属性、方法和事件。Timer 类不会像 setInterval()重复积累调用，减少了出错的几率。其次，可以自定义间隔时间，实现与帧频的脱离。所以，Timer 类基本上具有了 setInterval()函数和 enterFrame 事件的优点，是制作间隔效果的首选。

第五章 函数

1、 AS3.0 中的函数有较大变化，它删除了许多全局函数。例如，stop()函数在 AS2.0 中是一个全局函数，在 AS3.0 中则由 MovieClip 类的 stop()方法来代替。AS3.0 中的全局函数主要有下面几类：

- trace()函数；
- int()、Number()等类型转换函数；
- isNaN()函数

这些函数在 AS3.0 中称为顶级函数或全局函数，即可以在程序的任何位置调用。还有一些函数是位于一定的包内。如果在时间轴上编程，可以不理会函数是否在某个包中，但进行类编程时，如果要使用包内的函数，必须先导入包。因此，可以把内置函数分为顶级函数和包函数两大类。

2、 从面向对象编程的角度来说，函数即方法，方法是在类中定义的函数。方法可以分为实例方法和类方法，在时间轴中自定义的函数都属于实例方法。在实例的当前位置调用函数，可以省略实例名。如果是在实例的其他位置调用函数，实例名不能省略。

3、 类、函数和接口都是属于复杂数据类型，复杂数据类型又可称为引用数据类型。所以，函数与类一样，也是通过引用来操作数据的。函数名即是对函数的引用。

4、 用 function 关键字定义函数与调用函数的先后顺序无关。当用变量来引用函数并调用函数时，要先定义变量，再调用函数，所以就有顺序关系，因为变量要先定义才能使用。

5、 编写较复杂的函数时，一般是逐步增加代码，并且分步调试。

6、 函数的参数可以是任何数据类型的变量，数据类型可以分为简单数据类型和复杂数据类型。当使用简单数据类型变量作为参数时，传递的是值；当使用复杂数据类型变量作为参数时，传递的是引用。值和引用是简单数据类型和复杂数据类型的最大区别。

7、 AS3.0 新增了两个功能，可以在定义参数时，给参数一个默认值，而且，可以给函数不确定的参数。在使用参数的默认值功能时，默认参数必须从右往左，即所有的默认参数要写在必须参数的后面。

8、 凡是在函数里用关键字 var 声明的变量都是局部变量，它只能在函数中使用，调用函数结束时，局部变量就不发挥作用。在时间轴上编程时，在函数以外定义的变量，包括使用关键字 var 定义的变量都属于实例变量，也可以称为实例属性。在时间轴上定义的变量也称为时间轴变量，因为在定义变量的时间轴中都可以直接访问这个变量。

第六章 显示对象

1、 通过抽象与继承，把所有的显示类大致分成了交互类和非交互类。交互类的名称为 InteractiveObject 类。在 InteractiveObject 类中，定义了具有的交互属性、方法和事件。而对于非交互类，由于这个类没有处理事件的任何功能，所以在非交互类中其实不存在任何属性、方法和事件。因此，AS3.0 并没有把非交互类加到内置类中，而是让所有非交互类直接继承 DisplayObject 类。

2、 DisplayObject 类的部分属性、方法和事件

DisplayObject 类	*属性、方法或事件名*
属性	x、y、width、scaleX 等
方法	GlobalToLoca()、getRect()等
事件	enterFrame、removed 等

3、 AS3.0 中的非交互类有 AVM1Movie 类、Bitmap 类、MorphShape 类、Shape 类、StaticText 类和 Video 类，这些类的具体说明如下：

非交互类	*说明*
AVM1Movie 类	用于早版本的影片

Bitmap 类	创建位图
MorphShape 类	用于补间动画
Shape 类	创建背景等图形
StaticText 类	创建静态文本
Video 类	创建视频

在 6 种非交互类中，有些类只能在创作环境中手工创建实例，有些类只能使用程序来创建，有些则二者均可。例如，静态文本只能使用工具栏上的“文本工具”来创建；Bitmap 类的实例使用“new”来创建；Shape 类的实例二者均可。

4、interactiveObject 类的部分属性、方法和事件

interactiveObject 类	*属性、方法或事件名*
属性	contextMenu、doubleClickEnabled 等
方法	InteractiveObject()
事件	Click、doubleClick 等

4、 具有交互功能的显示类都继承于 InteractiveObject 类，这些显示类可以分为 SimpleButton 类、TextField 类、Loader 类、Sprite 类、Stage 类和 MovieClip 类，具体说明如下：

交互类	*说明*
SimpleButton 类	创建按钮实例
TextField 类	创建动态文本实例
Loader 类	加载外部影片、图像等
Sprite 类	编写交互代码
Stage 类	创建舞台
MovieClip 类	创建影片剪辑实例

在 6 种交互类中，按其是否能嵌套其他显示实例，可以分为容器类和非容器类。容器类的实例能嵌套其他显示实例，Loader 类、Sprite 类、Stage 类和 MovieClip 类属于容器类。非容器类的实例不能嵌套其他显示实例，如 SimpleButton 类和 TextField 类。容器类的抽象类即 DisplayObjectContainer 类。

5、 DisplayObjectContainer 类主要实现了添加显示实例、移除显示实例和深度管理等功能，部分属性、方法和事件说明如下：

DisplayObjectContainer 类	*属性、方法或事件名*
属性	numChildren 等
方法	addChild()等
事件	enterFrame、removed 等继承事件

6、 Flash 中的绘图是通过 Graphics 来实现的，Graphics 类是直接继承了 Object 类，它具有很多的绘图方法，大致可以把这些方法分为两类：一是定义绘图样式的方法；二是用于绘制和清除图形的方法。

Graphics 类有关样式的方法包括线条样式和填充样式两类，如下表：

方法名	*说明*
lineStyle	定义线条样式
lineGradientStyle	定义渐变线条样式
beginFill	定义固体填充样式
beginGradientFill	定义渐变填充样式
beginBitmapFill	定义位图填充
endFill	结束填充方法

Graphics 类有关绘图的方法包括绘制线条和绘制开头两类，如下表：

方法名	说明
moveTo	定义绘制线条的起点
lineTo	定义绘制线条的终点
curveTo	绘制曲线
drawCircle	绘制圆形
drawEllipse	绘制椭圆
drawRect	绘制矩形
drawRoundRect	绘制圆角矩形
Clear	清除绘图

与线条绘制不同，在填充绘图时，如果填充结束时，需要调用 endFill()方法表示结束填充。除了使用纯色填充外，还可以使用渐变填充和位图填充。

7、 Stage 类的实例只有一个，即常说的舞台。在默认情况下，已经有一个显示实例添加到舞台中，这个显示实例就是主时间轴。通过变量 root 或关键字 this 可以引用主时间轴实例，向其中添加其他显示实例。当然，也可以把动态文本实例等显示实例直接放在 stage 下，但这并不是一个好的编程习惯。在实际应用中，应把相关的显示实例放在相同的容器中，进行分类管理。

8、 要在显示列表中访问某个显示实例，就需要使用路径。路径就像文件系统中的文件一样，通过一个个文件夹就可以找到某个文件。通过路径一级级下去，就可以找到某个显示实例。所以，通过 Flash 编程时，要有容器的概念，这个容器就像文件夹一样，可以有效的管理显示实例。也就是说，任何一个显示实例肯定在某个显示容器中。所以在实例化显示类时，都需要使用 addChild()方法把显示实例添加到显示列表中。下表列出了添加和移除显示实例的方法：

方法名	说明
addChild	把显示实例添加到显示列表中
addChildAt	通过索引把显示实例添加到显示列表中
removeChild	从显示列表移除显示实例
removeChildAt	通过索引从显示列表移除显示实例

9、 如果把主菜单放到主时间轴下后，又把主菜单放到舞台下，这时并不会在主时间轴和舞台上都含有主菜单。当进行两次添加动作后，主时间轴下的主菜单实际上是被移动到了舞台，即先从主时间轴移除，再添加到舞台下。因此，千万不要通过这种方法来实现显示实例的复制。

10、 任何一个 Flash 应用程序其实都可以看成是显示列表。访问显示实例的方法可以分为两种：一种是从下向上访问；另一种是从上向下访问。对于任意一个除 stage 之外的显示实例，肯定有一个上一级的显示实例。Stage 没有上一级显示实例，所以 stage 属于顶级容器。当上一级的显示实例存在时，肯定只有一个，所以访问上一级显示实例比较简单，只需通过 parent 属性即可。从上往下访问就较麻烦，因为一个显示实例容器中可能不止一个显示实例。

11、 要访问下一级显示实例，可以使用 DisplayObjectContainer 类中的方法，如下表：

方法名	说明
getChildAt	根据索引来访问
getChildByName	根据实例名来访问
getChildIndex	得到某个实例的索引
getObjectsUnderPoint	得到包含某坐标的实例数组

contains	判断是否含有某实例

12、 当多个显示实例存在于容器时，要访问这个容器下的显示实例，可以采用深度索引来访问，即通过 getChildAt()方法来访问。深度越大，显示实例就显示在越上层。但这样的方法有时候并不好，因为深度是通过数字来管理的。当容器中有大量显示实例时，程序员并不能记忆所有显示实例的深度。因此，这种方法一般只适合访问容器内所有显示实例的情况。一般来说，如果要访问某个实例，采用实例名访问的方式比较好，即 getChildByName()方法。

13、 综上所述，访问下级实例的方法有两种：一是通过实例名来访问；二是通过深度索引来访问。在访问下一级实例时，不能通过简单的点语法来访问。通过点语法访问子级实例在特殊情况下是可以的。例如，显示列表中的实例都是手工创建的影片剪辑实例时。

14、 与深度有关的方法如下表：

方法名	*说明*
setChildIndex	设置某实例的特定深度
swapChildren	通过实例引用交换深度
swapChildrenAt	通过深度索引来交换深度
addChildAt	添加到指定深度的显示列表中
getChildAt	通过深度得到显示实例
getChildIndex	通过显示实例得到深度
removeChildAt	通过深度移除显示实例

15、 Flash 管理深度的方法称为 z 顺排列，就是用一个数组存放要排序的显示实例，通过数组操作来实现相同的排序功能。它的基本思路是把一些要排序的显示实例放置到一个数组中。当创建第一个显示实例时，这个实例就是数组的第一个元素，元素的索引 0 就是显示实例的深度，当创建多个显示实例时，处理的方法也是一样的。越迟添加的显示实例肯定深度越大，总是显示在最上面。创建的显示实例的深度按照 0,1,2,…来保存。在 AS3.0 中，不能跨深度指定显示实例的深度。例如，在显示容器内有 5 个显示实例，它们的深度应是 0,1,2,3,4，这时不能指定某个显示实例的深度大于 4，也不能交换大于 4 的深度。

16、 在 Flash 中可以手工创建显示实例，也可以程序创建显示实例。如果使用程序创建显示实例，容器中第一个添加的显示实例的深度为 0，然后依次递增。如果同时使用程序和手工来创建显示实例，手工创建的显示实例的深度小，程序创建的显示实例的深度较大，所以程序创建的显示实例会位于手工创建的显示实例的上面。

对于手工创建的显示实例，如果位于同一图层的同一帧，先创建的显示实例的深度小，显示在下面。如果显示实例位于不同的图层，不管这些实例的创建顺序如何，上面图层的深度比下面图层的深度要大。

17、 AS3.0 新增了 FrameLabel 类管理帧标签，FrameLabel 类只有 frame 和 name 只读属性，分别表示帧数和帧标签。在进行帧跳转时，建议不要使用具体的帧数，而应该用帧标签来代替。当要跳转的帧数改变后，使用帧标签就不用修改程序，只要改变相应帧即可，而直接使用帧数就需要修改程序。

18、 Flash 应用程序中经常会有许多体积较大的元素，当程序正在下载这些元素时，需要给用户一个正在加载的信息，这就是预载。预载可以分为自身预载和预载外部文件两种。不管哪种预载，都需要通过 LoaderInfo 类的相关属性和方法来实现。LoaderInfo 类的相关属性和事件如下表：

属性或事件	*说明*
bytesTotal	需加载的字节总数

bytesLoaded	已加载的字节数
progress	加载过程事件
complete	加载完成事件
init	加载完成并初始化完成事件

19、 通过 Tween 类可以使用程序来创建补间动画，通过指定目标影片剪辑的属性使得若干帧数或秒数具有动画效果，从而对影片剪辑进行移动、调整大小和淡入淡出操作。

20、 使用 TransitionManager 类定义动画效果，它允许将 10 种动画效果中的 1 种应用于影片剪辑。在创建自定义组件时，可以使用 TransitionManager 类将动画效果应用于组件可视界面中的影片剪辑。

第七章 事件处理

1、 AS3.0 事件处理最关键的就是 addEventListener()函数(方法)，这个方法是在 EventDispatcher 类定义的，DisplayObject 类就是它的子类。EventDispatcher 类中包含了 6 个实例方法如下表

方法名	*说明*
EventDispatcher()	构造方法，用于创建实例
addEventListener()	用于注册侦听器
dispatchEvent()	用于自定义事件
hasEventListener()	检测是否具有某个侦听器
removeEventListener()	用于移除侦听器
willTrigger()	检测是否具有某个侦听器

2、 AS3.0 的事件是由很多事件类来管理的，这些类包含了许多属性、方法，以及非常复杂的事件。在所有事件类中，Event 类处理一般事件，MouseEvent 类处理鼠标事件，KeyBoardEvent 类处理键盘事件……。这些类的架构方式与显示类相似，都是通过继承来实现的。事件类都继承了 Event 类，即 Event 类是所有事件类的父类。

3、 事件流用于描述事件发生在显示列表中，遍历其所有节点的过程，它可以分成 3 个阶段：

- 捕获阶段(capture phase)：从顶部(如 stage)到目标
- 目标阶段(target phase)：目标
- 冒泡阶段(bubbling phase)：从目标到顶部

4、 在使用 addEventListener()函数时，有名为 useCapture 的第 3 个参数，表示是否使用捕获阶段，并且这个参数的默认值是 false，即默认情况下没有捕获阶段，只有目标阶段和冒泡阶段。如果要使用捕获阶段，可把第 3 个参数设置成 true。

5、 Event 类的 bubbles 属性是一个只读属性，用来表示此事件是否可以使用冒泡机制(true/false)，例如 MouseEvent 类的单击事件就可以使用冒泡机制，而 Event 类的完成(complete)事件就不能使用冒泡机制。Event 类的 eventPhase 属性也是只读属性，表示事件流的阶段。这个属性通过数字来表示，数字 1、2、3 分别表示捕获阶段、目标阶段、冒泡阶段。

6、 Event 类的 target 和 currentTarget 属性是两个很相似的属性。对于一个简单的事件处理过程，分清 target 和 currentTarget 并没有必要，因为它们一般指向同一个对象。但在一个相对复杂的显示列表中，这两个属性是不同的。

如果为父级注册一个单击事件侦听器，单击父级时 target 和 currentTarget 都指向父级；单击子级时，target 指向子级，而 currentTarget 指向父级。因此，在很多应用中，一般认为 currentTarget 指向父级。

但是，如果为父级和子级斗都注册一个侦听器，那 target 属性是指向单击的目标，而 currentTarget 属性是指在处理的事件，即活动的目标。因为 AS3.0 的事件处理有 3 个阶

段，并且默认时采用冒泡机制，当单击子级时，currentTarget 属性应先指向目标，并向上冒泡，即先指向子级，再指向父级。

因此，currentTarget 属性应具备两条件：一是它注册了侦听器，二是正在处理事件。而 target 属性就指事件流中的目标。target 属性在事件流的目标阶段，而 currentTarget 属性在事件流的 3 个阶段。以单击事件为例，只有当事件流处于目标阶段时，currentTarget 属性与 target 属性的指向才相同，当事件流处于冒泡阶段和捕获阶段时，target 属性总是指向被单击的对象，而 currentTarget 属性指向当前事件活动的对象。

即使在没有发生事件流的处理时，有时也需区分 target 和 currentTarget 属性。

7、 事件的默认行为是指当事件发生时，Flash 播放器会自动显示与此事件相关的行为，因为这个行为是由 Flash 播放器自动显示的，所以这样的行为称为默认行为。当然，可以通过 preventDefault()方法来阻止默认行为的发生。但是，并不是所有事件对象相关的默认行为都能被阻止，例如单击事件的默认行为就不能被阻止，可以通过 Event 类的 cancelable 属性来查看某个事件的默认行为能否被阻止(true/false)。

8、 当为同一个发送者注册多个相同的事件时，事件的执行会有一定的优先级。默认情况下，事件是按照函数注册的先后顺序来调用的。实际上，在注册侦听器的时候可以为侦听器指定优先级。addEventListener()函数的第 4 个参数一般用 priority 表示，用来指定事件的优先级。优先级是用数字表示的，默认值为 0，所以默认的事件优先级都是一样的。如果事件具有相同的优先级，就按事件注册的先后顺序来执行。指定第 4 个参数时，可以用整数、正数、负数，按照数字的大小来决定事件执行的顺序，大表示优先执行。

9、 Event 类提供了两个方法来阻止事件的传递。其中，stopPropagation()是停止事件流中当前节点以后的事件，对当前节点并不影响；stopImmediatePropagation()方法将马上阻止后面的所有事件。

10、 EventDispacher 类的 addEventListener()方法还有第 5 个参数 useWeakReference，表示是否引用弱引用，其默认值为 false，即不使用弱引用。前面使用的事件都是系统内置的，如果想定义自己的事件，可以使用 EventDispacher 类的 dispatchEvent()方法，该方法只有一个参数，这个参数必须是 Event 类或其子类的实例。

11、 EventDispatcher 类的子类 DisplayObject 类具有 6 个基本事件，这 6 个事件都是由 Event 类来管理的。下标列出了 DisplayObject 类的事件：

事件名	*静态属性名*	*说明*
activate	Event.ACTIVATE	当播放器获得系统的焦点时发生
deactivate	Event.DEACTIVATE	当播放器失去系统的焦点时发生
added	Event.ADDED	当显示实例被添加到显示列表时发生
removed	Event.REMOVED	当显示实例从显示列表中删除时发生
enterFrame	Event.ENTER_FRAME	播放头进入新的帧时发生
render	Event.RENDER	更新显示列表时发生

由于 DisplayObject 类的这 6 个事件是由 Event 类管理的，所以在处理事件时，必须要在 DisplayObject 类和 Event 类之间发生联系。DisplayObject 类的实例是事件的发送者，而 Event 类的静态属性是事件名，Event 类的实例是事件对象，接收器函数必须接受 Event 类的实例作为参数，这就是事件的基本原理。

12、 AS3.0 的显示实例如果没有被添加到显示列表中或者不可见时，它本身还是存在的。addChild()方法只是把显示实例添加到显示列表中，而不是创建显示实例；同样，removeChild()方法也只是把显示实例从显示列表中删除，而不是删除显示实例。创建显示实例的方法与创建普通对象的方法一样，都是通过 new 来创建。

判断显示实例是否在舞台中，可以通过显示实例的 stage 属性来判断。如果显示实例没

有添加到显示列表，说明显示实例不在舞台中，它的 stage 属性值为 null；如果显示实例在舞台中，它的 stage 属性引用舞台实例。

13、 KeyboardEvent 类的属性：

属性	*说明*
altKey	Alt 键是否可用
charCode	键的 ASCII 码
ctrlKey	Ctrl 键是否可用
keyCode	键控代码值
keyLocation	键在键盘中的位置
shiftKey	Shift 键是否可用

14、 KeyboardEvent 类的方法：

方法	*说明*
KeyboardEvent()	构造函数，用于创建实例
clone()	复制实例
toString()	返回全部属性的字符串
updateAfterEvent()	更新显示，与设置的帧速无关

KeyboardEvent 类提供了 updateAfterEvent()方法用来更新显示，它可以让对象的显示与帧频无关，即保证了对象的显示更加流畅。

15、 MouseEvent 类的常用属性：

属性	*说明*
buttonDown	鼠标是否被按下
delta	使用滚轮时，每单位滚动的值
localX	鼠标的本地 x 坐标
localY	鼠标的本地 y 坐标
relatedObject	鼠标滑进滑出时指向的显示实例
stageX	鼠标的全局 x 坐标
stageY	鼠标的全局 y 坐标

relatedObject 属性只在鼠标滑进滑出事件发生时才有用。例如，当鼠标从一个对象移动到另一个对象时，会触发第一个对象的滑出事件，此时 relatedObject 属性就引用另一个对象。由于 relatedObject 属性是对象，这个对象也会有很多属性和方法，在调用这些属性和方法的时候，最好先判断对象是否存在，因为 relatedObject 属性并不是随便什么时候都有。

16、 使用 mouseWheel 事件时要注意一点：当在创作环境中测试 mouseWheel 事件时，由于创作环境的快捷键会屏蔽播放器的快捷键，需单击对象激活后 mouseWheel 事件才有效，但在播放器独立测试时，可以直接使用 mouseWheel 事件。

第八章 Flash 数学基础

1、 在创建 Flash 文档时，除了 stage 被创建外，还创建了主时间轴 root，root 是 MovieClip 类的实例，它的默认位置在舞台的左上角，即坐标(0,0)处。改变了 root 的坐标，root 下的所有显示实例的坐标也会跟着改变。因此，在进行 Flash 开发应用时，最好把所有的显示实例都放在 root 下，而不是直接放在 stage 下，这样只需改变 root 的坐标，就能非常方便地改变这些显示实例的位置，而 stage 虽然有 x 属性，但这个属性不能修改。

2、 中心点与注册点是两个很容易混淆的概念，它们可以位于显示实例的同一位置，也可以位于显示实例的不同位置。Flash 中的中心点是以小圆圈表示，注册点以小十字表示，中心点与注册点的位置与显示实例的坐标有关。

3、 在【信息】面板中，宽和高为元件的大小，中间的小圆圈表示中心点的位置。

单击【信息】面板上的元件位置，这时左上角的小方框变成十字形，移动正方形的中心点，【信息】面板中的坐标将不发生变化。

【信息】面板上显示的坐标与元件位置有关，当元件位置位于左上角时，以元件的左上角在场景中的位置显示坐标。当元件位置在中间时，以元件的中心点在场景中的位置显示坐标。

4、影片剪辑实例有很多属性，其中 x 和 y 分别代表 x 坐标和 y 坐标。这两个属性可以被获取，也可以被设置，获取和设置坐标属性时用点运算符。影片剪辑实例的坐标是由其注册点在父级显示实例中的位置决定的。由于这时的父级显示实例即 root 位于(0,0)处，所以影片剪辑实例的坐标与舞台中的坐标对应。

5、影片剪辑有全局坐标和本地坐标，以父级显示实例为基准的坐标称为本地坐标，相对舞台左上角的坐标称为全局坐标。DisplayObject 类提供了两个方法进行全局坐标和局部坐标的转换，其中，localToGlobal()方法是把本地坐标转换为全局坐标，globalToLocal()方法是把全局坐标转换为本地坐标，这两个方法都需要传递 Point 类的实例。

Flash 中除了显示实例的坐标外，还有鼠标坐标，用 DisplayObject 类的 mouseX 属性和 mouseY 属性来表示，鼠标坐标同样存在着全局坐标和本地坐标之分，stage.mouseX(Y)得到全局坐标，实例名.mouseX(Y)得到本地坐标。

6、Flash 显示编程中的注册点直接决定了显示实例的坐标、角度、缩放等属性。在默认情况下，任何一个程序创建的显示实例的注册点都是在左上角。例如，如果要旋转显示实例，它将绕着左上角进行旋转；如果缩放显示实例，它将以左上角为定点，向右向下缩放。如果要指定任意点为注册点，则需要一些技巧(例如包装对象法和坐标转换法)，因为在 DisplayObject 类及其子类中并没有现成的方法可以使用。

7、在【变形】面板中，【旋转】单选按钮是用来显示或者设置显示实例的角度，范围是-360~360，如果输入小于-360 或大于 360，Flash 自动按这两个值处理。当设置角度时，显示实例以中心点为定点，按一定方向旋转。显示实例的初始角度为 0，以中心点为定点，以顺时针方向为正，范围为 0~180；以逆时针方向为负，范围为-180~0。【变形】面板中的角度和 rotation 代表的角度的变化规律相同，初始角度为 0，以顺时针方向为正，范围为 0~180；以逆时针方向为负，范围为-180~0。但旋转定点不一样，前者是以中心点为定点，后者是以注册点为定点。

8、Math 是静态类，即其中的所有属性和方法都是静态的，它们都与数学有着密不可分的关系，掌握 Math 类就能开发简单的 Flash 游戏。Math 类中经常用到的知识有勾股定理、正余弦函数、正余弦曲线、随机方法等。

当显示实例超出舞台或者不需要时，应及时删除 enterFrame 事件，并把显示实例从显示列表中删除，否则将会占用大量内存。

9、AS3.0 不再提供全局函数 random()，而统一使用 Math 类提供了 random()方法取随机数。这个方法被调用后，将随机返回大于等于 0 并小于 1 的数字。

要使 random()返回更大范围的小数，只需乘上一个数字。例如：

```
for(var i:int=0;i<100;i++)
trace(Math. random()*10);   //返回 100 个大于等于 0 小于 10 的小数
for(var i:int=0;i<100;i++)
trace(Math. random()*10+10);   //返回 100 个大于等于 10 小于 20 的小数
```

由于 random()方法返回的是小数，所以经常需要对小数进行取整。在 AS2.0 中，取整只能通过 floor()和 round()方法，而在 AS3.0 中，有了更多的选择，因为 AS3.0 提供了 int 和 uint 整数类型，当把小数赋值给 int 和 uint 类型的变量时，小数部分将被忽略。例如：

```
for(var i:int=0;i<100;i++) {
```

```
    var t:uint = Math.random()*10+10;
    trace(t);
}                    //输出大于等于 10 小于 20 的整数，包括 10 但不包括 20
for(var i:int=0;i<100;i++) {
    var t:uint = Math.random()*11+10;
    trace(t);
}                    //输出大于等于 10 小于等于 20 的整数，包括 10 和 20
for(var i:int=0;i<100;i++) {
    var t:uint = Math.random()*21-10;
    trace(t);
}                    //输出-10 到 10 的随机整数，包括-10 和 10
```

在使用 floor()和 round()方法进行取整时，要注意两者的区别：Math()类中的 floor()方法返回参数 x 中指定的数字或表达式的下限值(即[x])，round()方法返回参数 x 四舍五入后的整数。

10、 Point 类表示二维坐标系统中的位置，即用横坐标和纵坐标表示点，利用 Point 类可以有效地处理坐标系中点的问题。Point 类提供了 3 个实例属性，其中，length 表示坐标(0,0)到指定点之间的距离，x 属性表示横坐标，y 属性表示纵坐标。如果要求线段的长度，可以调用 Point 类的静态方法 distance()。

如果规定起点为 p1，终点为 p2，这个线段就成为了有向线段。有向线段可用来表示运动的效果，线段的长度就是运动的速度大小，线段的方向就是运动的方向。

取线段 p1p2 上的一个点 p，就把有向线段分成了 p1p 和 pp2 两条线段，点 p 就称为定比分点。Point 类提供了静态方法 interpolate()来求定比分点的坐标，通过给定线段长度的比例，可以很方便地求出定比分点的坐标。求定比分点的原理是根据比例和已知的数值来求另一个数值，这个原理在编程中有很多用途。

11、 点的坐标都是存在于一定的坐标系中，常见的坐标系有笛卡尔坐标系和极坐标系两种。Point 类中的静态方法 polar()可以把极坐标转换成笛卡尔坐标，它的第一个参数是半径，第二个参数以弧度为单位。Point 类没有提供方法把笛卡尔坐标转换为极坐标。

12、 AS3.0 用 Rectangle 类来描述矩形，Rectangle 类用 4 个属性来描述矩形。其中，x 属性和 y 属性表示矩形左上角的坐标，width 和 height 表示矩形的宽和高。DisplayObject 类的 getBounds()和 getRect()方法可以返回一个 Rectangle 类的实例，通过 Rectangle 类的实例就可得到背景矩形的 4 个属性。getRect()方法得到的矩形区域不包括形状上的任何笔触，而 getBounds()方法得到的矩形区域包括笔触，且笔触的默认宽度是 1 像素。

第九章 Flash 物理基础

1、 匀速直线运动：运动=位置+速度；

加速运动：速度=速度+加速度；

有摩擦力：速度*=小数；

弹性力：加速度=中间坐标—物体坐标；

匀速圆周运动：x=cos(弧度)*半径，y=sin(弧度)*半径，弧度+=某值；

两点间的运动(缓冲运动)：mc.x += (目标位置 x – mc.x)*缓冲系数

mc.y += (目标位置 y – mc.y)*缓冲系数

2、 碰撞检测的方法有很多，AS3.0 提供了一些内置的方法来检测碰撞，同时也可使用简单的数学计算来检测碰撞。

形状与形状碰撞

DisplayObject 类提供了 hitTestObject()与 hitTestPoint()方法来检测对象之间的碰撞，其中

hitTestObject()根据对象的边框来检测碰撞。如果两个对象在任意一点上重复或交叉，hitTestObject()方法会返回 true，否则返回 false。

点与形状碰撞

如果要检测某个点是否与对象碰撞，可使用 hitTestPoint()方法，通过设置 hitTestPoint()方法的第 3 个参数，可决定碰撞对象时是否使用形状检测。

矩形与形状碰撞

矩形检测是指矩形与形状的检测，从本质上讲，可以把矩形检测看成是矩形的 4 个顶点与形状的检测，即点与形状的检测。

第十章 面向对象编程基础

1、 包是 AS3.0 中有效管理类的一个机制，所有的内置类都按一定的规则放在不同的包中。合理地使用包可以分类管理类，防止命名冲突。

2、 使用相对路径保存的类文件一般只用于特定的 fla 文件，因此，使用这种方法保存可重复使用的类文件就不合适了。Flash 提供了绝对路径保存文件的方法，这样就可以把类放在一定的目录中，而对于 fla 文件来说，不管放在什么位置，都能使用这些类文件。

用绝对路径保存文件需要设置类路径，在 fla 文件和 as 文件中都可以设置类路径，这两种类路径是有区别的：在 as 文件中设置的类路径称为全局类路径，而在 fla 文件中设置的类路径称为文档级类路径。不管是哪种类路径，都可以设置相对路径和绝对路径。一旦设置了全局类路径，就能使所有的 fla 文件共享，而文档级类路径只适用于设置过类路径的 fla 文件。

3、 AS3.0 除了类继承，还有原型继承，这种继承方式早在 AS1.0 就已经有了。虽然在 AS1.0 中不支持用 class 关键字来定义类，但仍然可以用一个特殊的对象来定义，这个对象叫做原型(prototype)对象。它把本来应该使用关键字 class 定义的抽象数据类型，定义在一个具体的对象内。基于原型的 AS1.0 使用一个存在的对象作为其他对象的模型(或原型)。在基于类的语言中，一个类在作用相当于该类对象的模板，而在基于原型的语言中，对象的模板是另一个对象的原型。 如：

Teacher.prototype = new Person();

则 Person 类和 Teacher 之间的关系类似一种继承关系，也就是所谓的原型链(prototype chain)。每个原型链的基类都是 Object 类，Object 类包含了一个叫做 Object.prototype 的静态属性，它被指定为所有被创建对象的原型。

4、 文档类是 AS3.0 新出现的一种特性，文档类可以使用【属性】面板来实例化。从本质上讲，文档类是一种显示类；从位置上看，文档类位于显示列表的根。

第一章曾经讲到，当创建 flash 文档时，系统会自动创建 MainTimeline 类，MainTimeline 类其实就是文档类，MainTimeline 类的实例就是主时间轴。但是，MainTimeline 类在应用程序中并不是必须的，它本质是一个继承 MovieClip 类的文档类。也就是说，可以编写自己的文档类，来替换 MainTimeline 类。从这个角度讲，AS3.0 编程可以放弃时间轴，文档类的出现实现了代码与设计的真正分离。

文档类必须直接或间接地继承 Sprite 类或 MovieClip 类。如果在主时间轴中只有一帧，则选择 Sprite 类作为基类；如果主时间轴有多帧，则选 MovieClip 类。如果文档类继承 Sprite 类，在时间轴上就不能写代码了，因为 Sprite 类是没有的时间轴的。如果要在时间轴上写代码，要继承 MOvieClip 类。在文档类和帧中都编写代码时，代码的执行是有顺序的。先执行文档类的构造函数，再执行帧中的代码。

一般情况下，文档类都是非动态的，虽然 MovieClip 类是动态的，但 MovieClip 类的子类却不是动态的，因此，要动态地向文档类实例添加属性，必须把文档类定义成动

态的。此时实例哈过的文档类就像一般的主时间轴一样，可以动态地添加属性和方法。

MainTimeline 类是系统默认创建的动态文档类，MainTimeline 类是直接继承 MovieClip 类的。而文档类可以继承 Sprite 类或其子类。由于文档类的特殊性，即使文档类中的属性或方法被设置成最低级别 private，在时间轴也同样能访问，因为时间轴的代码就像写在文档类中，在类中可以访问 private 属性或方法。

一般来说，当在主时间轴的帧中编写代码时，使用位于 flash*包中的类和函数时，并不需要导入，但如果使用了文档类，则需要导入包，因此，运行时帧代码被处理成了类代码，在类中使用包中的类和函数时，需要先导入包。

当结合界面进行编程时，并不需要像以前版本的 ActionScript 那样，先定义变量来引用界面中的对象，在 AS3.0 中是可以直接使用的。若是在文档类中，有关界面中显示对象的包也要导入。

5、 在以往版本的 AS 中，要从库中导出资源，需要使用特定的函数。而 AS3.0 则增强、规范了从库中导出资源的功能，使用链接类可以很方便地从库中导出资源。实例化链接类与普通类一样，使用 new 运算符调用构造函数即可。

不管是从库中导出影片剪辑，还是按钮、位图等资源，都统一使用上面的方式，因此，从创建方式上讲，AS3.0 的功能更加规范，而且可以导出位图等，大大增强了导出的功能。

链接类只适用于创建显示对象。从本质上看，链接类是继承 MovieClip 类的一种类，通过继承扩展了显示类的功能。

6、 利用库中的链接类，就可以非常方便地创建动态资源库，即把资源放在一个 swf 文件中，如果其他 swf 文件需要使用这些资源，只需要把包含资源的 swf 文件导入。

7、 ApplicationDomain 类的用途是存储 AS3.0 定义表，swf 文件中的所有代码定义都存在于应用程序域中。可以使用应用程序域划分位于同一个安全域中的类，这允许同一个类存在多个定义，并且还允许子级重用父级定义。

通俗的说，在 A.swf 文件中包含多类，如果 A.swf 文件被 main.swf 文件加载，那么，main.swf 文件就可以使用 A.swf 文件中的类。但是，main.swf 可能加载不只是 A.swf，它还可以加载 B.swf、C.swf 等，而且 A.swf、B.swf 和 C.swf 可以包含相同类名的类，ApplicationDomain 类的作用就是把这些类进行保存，同名的类可以替换，也可以单独存在。所以，ApplicationDomain 类可以看作管理加载类的区域，有 3 个区域用来保存类。

- 加载到子域

  类似于“继承”，子域可以直接获得父域所有的类定义；反之，父域得不到子域的类定义。和继承关系不同的是，如果子域中有和父域同名的类，子域定义会被忽略而使用父域的定义。

- 加载到同域

  类似集合里的合并关系。被加载的 swf 文件里的所有类定义被合并到当前域中可以直接使用。和加载到子域相同，和当前域同名的域也会被忽略。

- 加载到新域

  swf 载入指定域之前，先要检查该域及其父域中是否存在同名类，重复定义一概忽略。如果加载别人写的程序，或者使用旧版本的主程序加载新版本的模块，为避免类名冲突就要加载到新域独立运行以使用自己的类。