☹

SWF 文件格式规范

版本 19

商标

第三方信息

Adobe Systems Incorporated - 2012 年出版

目录

简介

SWF（发音为“swiff”）文件格式通过互联网提供矢量图形、文本、视频和声音，并由 Adobe® Flash® Player 软件支持。SWF 文件格式旨在成为一种高效的交付格式，而不是用于在图形编辑器之间交换图形的格式。它旨在实现以下目标：

- 屏幕显示——该格式主要用于屏幕显示，支持抗锯齿、快速渲染到任何颜色格式的位图、动画和交互式按钮。

- 可扩展性——该格式是一个标记格式，因此可以在保持与 Flash Player 早期版本的向后兼容性的同时，通过添加新功能来进化。

- 网络交付——该格式可以在有限且不可预测的带宽网络上传输。文件被压缩以减小体积，并支持通过流式传输进行增量渲染。SWF 文件格式是一种二进制格式，不像 HTML 那样可读。SWF 文件格式使用位打包和具有可选字段的结构等技术来最小化文件大小。

- 简洁性——格式简单，使得 Flash Player 体积小且易于移植。此外，Flash Player 仅依赖于操作系统的一小部分功能。

- 文件独立性——文件显示时对外部资源（如字体）的依赖最小。

- 可伸缩性——文件在有限的硬件上运行良好，并且可以在硬件更好时充分利用。这种能力很重要，因为计算机有不同的显示器分辨率和位深度。

- 速度——SWF 文件描述的图形渲染速度快。

- 脚本化——该格式包含标签，提供字节码序列以供堆栈机器解释。字节码支持 ActionScript®语言。Flash Player 提供运行时 ActionScript 对象模型，允许与绘图原语、服务器和 Flash Player 功能交互。

SWF 文件扩展名为.swf，MIME 类型为 application/x-shockwave-flash。

SWF 格式经过多个版本的发展。通过 SWF 5，SWF 标签集增加了大量内容。从 SWF 6 开始，SWF 格式变化较少，因为更多的新特性在 ActionScript 级别部分或全部实现。从 SWF 9 开始，可以使用采用新 ActionScript 虚拟机 2（AVM2）的 ActionScript 3.0 语言。任何计划生成使用新特性的 SWF 文件内容的人都应该熟悉 Flash Player 暴露的 ActionScript 对象模型。有关此信息的参考资料包括 ActionScript 开发者指南（见 http://help.adobe.com/en_US/as3/dev/index.html）、ActionScript 3.0 语言参考（见 http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/）和 Adobe ActionScript 虚拟

☹

Adobe ActionScript 虚拟机 2 概述（PDF 文件）请访问 www.adobe.com/go/avm2overview。

如果您需要了解 Flash Player / AIR 名称与 SWF 版本之间的映射（例如，11.4 是 SWF 17），查找给定版本中的功能，或查看 Flash Player 和 SWF 功能的新增内容，请访问：Flash Player 和 Air 功能列表。

Adobe 认真考虑所有关于 SWF 文件格式规范的反馈。如有关于规范中的不明确或可能错误的信息，请发送电子邮件至 Adobe 的 flashformat@adobe.com。所有此类电子邮件提交均应遵守 www.adobe.com/misc/copyright.html 中使用的条款和条件下的提交材料指南。

发布后变更摘要：

- 2013 年 4 月 23 日 - 修复了 ActionStackSwap 中的错误顺序。

第一章：基本数据类型

本节描述了构成 SWF 文件格式中更复杂数据结构的基本数据类型。SWF 文件格式中的所有其他结构都是建立在这些基本类型之上的。

坐标和 twip

SWF 文件格式将所有 x-y 坐标存储为整数，通常以 twip 为单位。在 SWF 格式中，1 个 twip 等于逻辑像素的 1/20。逻辑像素在文件以 100%播放时与屏幕像素相同——也就是说，没有缩放。

例如，一个 800 twip 宽、400 twip 高的矩形被渲染为 40 by 20 逻辑像素。分数像素大小通过抗锯齿来近似。一个 790 by 390 twip（39.5 by 19.5 像素）的矩形看起来边缘略有模糊。

Twips 是大小和精度之间的良好折衷。它们提供了缩放和对象精确放置的亚像素精度，同时每个坐标消耗的比特数非常少。

SWF 文件格式中的坐标使用传统的图形坐标轴：x 轴是水平的，从左侧的最小值到右侧的最大值，y 轴是垂直的，从顶部的最小值到底部的最大值。

整数类型和字节序

SWF 文件格式使用 8 位、16 位、32 位、64 位、有符号和无符号整数类型。所有整数值都通过使用小端字节序存储在 SWF 文件中：最低有效字节首先存储，最高有效字节最后存储，这与 Intel x86 架构相同。SWF 文件格式中字节的位序是大端：最高有效位首先存储，最低有效位最后存储。

例如：

- 32 位值 0x456e7120 存储为 20 71 6e 45。
- 16 位值 0xe712 存储为 12 e7.

所有整数类型都必须是字节对齐的。也就是说，整数值的第一个位必须存储在 SWF 文件中字节的第一个位。

有符号整数通过使用传统的 2 的补码位模式来表示。这些是在大多数现代计算机平台上使用的有符号整数表示法。在 2 的补码系统中，负数的第一位是 1，而零和正数的第一位是 0。负数-n 表示为正零数 n-1 的位运算取反。

☹

整型类型

Type	评论
SI8	有符号 8 位整数值
SI16	有符号 16 位整数值
SI32	有符号 32 位整数值
SI8[n]	有符号 8 位数组——n 为数组元素个数
SI16[n]	有符号 16 位数组——n 为数组元素个数
UI8	无符号 8 位整数值
UI16	无符号 16 位整数值
UI32	无符号 32 位整数值
UI8[n]	无符号 8 位数组——n 为数组元素个数
UI16[n]	无符号 16 位数组——n 为数组元素个数
UI24[n]	无符号 24 位数组——n 为数组元素个数
UI32[n]	无符号 32 位数组——n 为数组元素个数
UI64[n]	无符号 64 位数组——n 为数组元素的数量

有理数

SWF 文件格式支持两种类型的有理数：32 位和 16 位。

32 位有理数是 16.16。也就是说，高 16 位代表小数点前的数字，低 16 位代表小数点后的数字。FIXED 值在 SWF 文件中以 32 位整数的形式存储（使用小端字节序），并且必须字节对齐。

例如：实数 7.5 等价于：0x0007.8000。此值在 SWF 文件中存储为：00 80 07
00.

SWF 8 及以后版本支持 16 位 8.8 有符号定点数。高 8 位代表小数点前的数字，低 8 位代表小数点后的数字。FIXED8 值在 SWF 文件中以 16 位整数的格式存储（使用小端字节序），并且必须字节对齐。

☹

定点类型

Type	评论
已修复	32 位 16.16 定点数
FIXED8	16 位 8.8 定点数

浮点数

SWF 8 及以后版本支持使用与 IEEE 标准 754 兼容的浮点数类型。支持三种浮点数类型。

浮点数类型

Type	评论
FLOAT16	半精度（16 位）浮点数
FLOAT	单精度（32 位）IEEE 标准 754 兼容
双精度	双精度（64 位）IEEE 标准 754 兼容

FLOAT16 与 FLOAT 的特性相同，只是在指数和尾数的位数分配上有所变化：

- 符号位 1 位
- 指数部分 5 位，指数偏置为 16
- 尾数部分 10 位

编码整数

SWF 9 及以后版本支持使用可变字节数编码的整数。支持一种编码整数的类型。

浮点数类型

Type	评论
EncodedU32	可变长度编码的 32 位无符号整数

这是一个使用可变字节数编码的 32 位无符号整数值，以节省空间。所有 EncodedU32 都编码为 1-5 个字节（较大的值需要更多的空间）。编码方法是这样的：如果当前字节中的高位被设置，则下一个字节也是值的一部分。一个字节中的每个位都向值贡献 7 位，其中高位告诉我们是否使用下一个字节，或者这是值的最后一个字节。

☹

这是解析 EncodedU32 的算法：

```
int GetEncodedU32(unsigned char*& pos) { int result
= pos[0]; if (!(result & 0x00000080)) {



      pos++;
      return result;
   }
result = (result & 0x0000007f) | pos[1]<<7; if (!(result &
0x00004000)) { pos += 2; return result; } result = (result &
0x00003fff) | pos[2]<<14; if (!(result & 0x00200000)) { pos
+= 3; return result; } result = (result & 0x001fffff) |
pos[3]<<21; if (!(result & 0x10000000)) { pos += 4; return
result; } result = (result & 0x0fffffff) | pos[4]<<28; pos +=
5; return result; }
```

比特值

比特值是可变长度的比特字段，可以表示三种类型的数字：

1. 无符号整数
2. 有符号整数
3. 签名的 16.16 位定点值。

比特值不必字节对齐。其他类型（如 UI8 和 UI16）总是字节对齐。如果一个字节对齐的类型跟在比特值后面，包含比特值的最后一个字节将用零填充。

以下示例是一个 64 位的流。这 64 位代表 9 个不同长度的值，之后是一个 UI16 值：

比特流从 6 位值（BV1）开始，接着是一个 5 位值（BV2），该值分布在 Byte1 和

☹

随后是一个 5 位值（BV2），它分布在 Byte1 和 Byte2 中。BV3 分布在 Byte2 和 Byte3 中，而 BV4 完全包含在 Byte3 中。Byte 5 包含两个位值：BV7 和 BV8。BV9 后面跟着一个字节对齐的类型（UI16），因此 Byte 6 的最后四位用零填充。

位值

Type	评论
SB[nBits]	有符号位值（nBits 表示存储该值所使用的位数）
UB[nBits]	无符号位值（nBits 表示存储该值所使用的位数）
FB[nBits]	有符号定点位值（nBits 表示存储该值所使用的位数）

当无符号位值扩展到更大的字长时，最左边的位被填充为零。当有符号位值扩展到更大的字长时，最高位被复制到最左边的位中。

这种扩展称为符号扩展。例如，4 位无符号值 UB[4] = 1110 将扩展为 16 位值，如下所示：0000000000001110 = 14。相同的值作为有符号值解释，SB[4] = 1110 将扩展为 1111111111111110 = -2。

有符号位值类似，但必须考虑符号位。35 的有符号值表示为 SB[7] = 0100011。需要额外的零位；否则高位将进行符号扩展，值将被解释为负数。

定点比特值是 32 位 16.16 有符号定点数。也就是说，高 16 位表示小数点前的数字，低 16 位表示小数点后的数字。定点比特值与有符号比特值相同，但解释不同。例如，19 位有符号比特值 0x30000 解释为 196608 十进制。19 位定点比特值 0x30000 解释为 3.0。此值的格式实际上是 3.16 而不是 16.16。

19 位定点比特值 0x30000 解释为 3.0。此值的格式实际上是 3.16 而不是 16.16。

使用位值

位值使用所需范围所需的最少位数进行存储。大多数位值字段使用固定数量的位数。有些使用可变数量的位数，但在所有这些情况下，要使用的位数都明确地在该结构中的另一个字段中声明。在这些可变长度的情况下，生成 SWF 文件的应用程序必须确定表示将要指定的实际值所需的最小位数。对于有符号位值，如果要编码的数字是正数，则需要额外的一位来保留前导 0；否则，符号扩展将位值变为负数。

作为一个变量大小的位值示例，考虑 RECT 结构：

字段	Type	评论
比特数	UB[5]	每个矩形值字段中的比特数
Xmin	SB[比特数]	矩形的最小位置 x
Xmax	SB[比特数]	矩形的最大位置 x
Ymin	SB[比特数]	矩形的最小位置 y
Ymax	SB[比特数]	矩形的最大位置 y

Nbits 字段决定了用于存储坐标值 Xmin、Xmax、Ymin 和 Ymax 的位数。假设矩形的坐标如下：

```
Xmin = 127 十进制 = 1111111 二进制 Xmax = 260 十进制
= 100000100 二进制 Ymin = 15 十进制 = 1111 二进制
Ymax = 514 十进制 = 1000000010 二进制
```

Nbits 通过找到需要最多位数来表示的坐标来计算。在这种情况下，该值是 514（二进制 01000000010）需要 11 位来表示。矩形存储如表所示：

字段	类型与值	评论
比特数	UB[5] = 01011	所需比特数（11）
Xmin	SB[11] = 00100000100	x 最小值（127 twips）
Xmax	SB[11] = 00001111111	x 最大值（260 twips）
Ymin	SB[11] = 00000001111	y 最小值（15 twips）
Ymax	SB[11] = 01000000010	y 最大值（514 twips）

字符串值

字符串值表示一个以空字符终止的字符字符串。字符串值的格式是一个以空字符字节终止的字节序列。

字段	Type	评论
字符串	UI8[零个或多个]	非空字符串字符数据

☹

字符串结束	UI8	字符串结束标记；始终为零

在 SWF 5 或更早版本中，STRING 值使用 ANSI（它是 ASCII 的超集）或 shift-JIS（一种日本编码）进行编码。您不能指定所使用的编码；相反，解码选择根据 Flash Player 运行的区域设置来决定。这意味着 SWF 5 或更早版本中的文本内容只能使用 ANSI 或 shift-JIS 进行编码，目标受众必须在创作过程中已知，否则将出现乱码。

在 SWF 6 或更高版本中，STRING 值始终使用 Unicode UTF-8 标准进行编码。这是一种多字节编码；每个字符由一个到四个字节组成。UTF-8 是 ASCII 的超集；UTF-8 中的字节范围 0 到 127 与 ASCII 映射完全匹配，所有 ASCII 字符 0 到 127 都由一个字节表示。UTF-8 保证当使用多于一个字节来编码除字符 0（空字符）之外的任何字符时，这些字节都不是零。这避免了 UTF-8 字符串中出现内部空字符，这意味着可以像 ASCII 字符串一样安全地处理空字节作为字符串终止符。

语言代码

语言代码标识适用于文本的 spoken language。语言代码与 SWF 文件格式中的字体规范相关联。

注意：语言代码不指定文本编码；它指定 spoken language。

字段	Type	评论
LanguageCode	UI8	语言代码（见下文）

Flash Player 使用语言代码来确定动态文本的换行规则，并在指定设备字体不可用时选择备用字体。未来可能会有更多语言代码的用途。

语言代码为零表示无语言。此代码的结果取决于 Flash Player 运行的区域设置。

在撰写本文时，以下语言代码被 Flash Player 识别：

- 1 = 拉丁语（拉丁-1 所覆盖的西方语言：英语、法语、德语等）
- 2 = 日语
- 3 = 韩语
- 4 = 简体中文
- 5 = 繁体中文

☹

RGB 颜色记录

RGB 记录表示颜色为 24 位的红、绿、蓝值。

字段	Type	评论
Red	UI8	红色颜色值
绿色	UI8	绿色颜色值
Blue	UI8	蓝色颜色值

RGBA 颜色与 alpha 记录

RGBA 记录表示颜色为 32 位红色、绿色、蓝色和 alpha 值。alpha 值为 255 的 RGBA 颜色是完全不透明的。alpha 值为 0 的 RGBA 颜色是完全透明的。alpha 值在 0 到 255 之间的是部分透明的。

字段	Type	评论
Red	UI8	红色颜色值
绿色	UI8	绿色颜色值
Blue	UI8	蓝色颜色值
Alpha	UI8	透明度定义的 alpha 值

带 alpha 记录的 ARGB 颜色

ARGB 记录的行为与 RGBA 记录完全相同，但 ARGB 记录的 alpha 值位于第一个字节。

字段	Type	评论
Alpha	UI8	透明度定义的 alpha 值
Red	UI8	红色颜色值
绿色	UI8	绿色颜色值
Blue	UI8	蓝色颜色值

☹

矩形记录

矩形值表示由最小 x 和 y 坐标位置以及最大 x 和 y 坐标位置定义的矩形区域。RECT 记录必须字节对齐。

字段	Type	评论
比特数	UB[5]	每个后续字段使用的位数
Xmin	SB[比特数]	矩形在 twips 中的 x 最小位置
Xmax	SB[比特数]	矩形在 twips 中的 x 最大位置
Ymin	SB[比特数]	矩形在 twips 中的最小位置
Ymax	SB[比特数]	矩形在 twips 中的最大位置

矩阵记录

MATRIX 记录表示一个标准的 2x3 变换矩阵，这种矩阵在 2D 图形中常用。它用于描述图形对象的缩放、旋转和平移。MATRIX 记录必须是字节对齐的。

字段	Type	评论
有缩放	UB[1]	如果等于 1，则具有缩放值
NScaleBits	如果 HasScale = 1，则 UB[5]	每个尺度值字段中的比特数
水平缩放比例	如果 HasScale = 1，则 FB[NScaleBits]	X 缩放值
Y 轴缩放	如果 HasScale = 1，则 FB[NScaleBits]	Y 轴缩放值
有旋转	UB[1]	如果等于 1，则具有旋转和倾斜值
NRotateBits	如果 HasRotate = 1，则 UB[5]	每个旋转值字段中的比特数
RotateSkew0	如果 HasRotate = 1，则 FB[NRotateBits]	首个旋转和倾斜值
旋转倾斜 1	如果 HasRotate = 1，则 FB[NRotateBits]	第二次旋转和倾斜值
NTranslateBits	UB[5]	每个转换值字段中的比特数
TranslateX	SB[NTranslateBits]	twips 中的 x 转换值
翻译 Y	SB[NTranslateBits]	在 twips 中翻译 y 值

☹

缩放 X、缩放 Y、旋转倾斜 0 和旋转倾斜 1 字段存储为 16.16 定点值。平移 X 和平移 Y 值以 twips 为单位存储为有符号值。

矩阵记录针对常见情况进行了优化，例如仅执行平移的矩阵。在这种情况下，HasScale 和 HasRotate 标志为零，矩阵仅包含 TranslateX 和 TranslateY 字段。

MATRIX 字段到 2x3 矩阵的映射如下：

水平缩放比例	RotateSkew0
旋转倾斜 1	ScaleY
TranslateX	翻译 Y

对于任意坐标 (x, y)，变换后的坐标 (x', y') 计算如下：

```
x’ = x * ScaleX + y * RotateSkew1 + TranslateX y’ = x *
RotateSkew0 + y * ScaleY + TranslateY
```

以下表格描述了矩阵成员在每种操作类型中的使用方法：

	水平缩放比例	RotateSkew0	旋转倾斜 1	ScaleY
旋转	余弦	Sine	负正弦	余弦
缩放	水平缩放 组件	无	无	垂直扩展 组件
切变	无	水平 成比例 恒量	垂直 成比例 恒量	无
反射	水平反射组件	无	无	垂直反射 组件

☹

颜色转换记录

CXFORM 记录定义了一个可以应用于图形对象色彩空间的简单变换。以下是可以进行的两种变换类型：

- 乘法变换
- 加法变换

乘法变换通过一个 8.8 定点乘法项来乘以红色、绿色和蓝色分量。1.0 的定点表示为 0x100 或 256 十进制。

对于任何颜色（R,G,B），变换后的颜色（R', G', B'）计算如下：

```
R’ = (R * RedMultTerm) / 256 G’ = (G *
GreenMultTerm) / 256 B’ = (B *
BlueMultTerm) / 256
```

添加变换会将一个加法项（正或负）添加到正在显示的对象的红色、绿色和蓝色分量中。如果结果大于 255，则结果会被限制为 255。如果结果小于零，则结果会被限制为零。

对于任何颜色（R,G,B），变换后的颜色（R', G', B'）计算如下：

```
R’ = max(0, min(R + RedAddTerm, 255)) G’ = max(0,
min(G + GreenAddTerm, 255)) B’ = max(0, min(B +
BlueAddTerm, 255))
```

加法和乘法变换可以按以下方式组合。首先执行乘法操作：

```
R’ = max(0, min(((R * RedMultTerm) / 256) + RedAddTerm, 255)) G’ = max(0, min(((G *
GreenMultTerm) / 256) + GreenAddTerm, 255)) B’ = max(0, min(((B * BlueMultTerm) / 256) +
BlueAddTerm, 255))
```

CXFORM 记录必须字节对齐。

字段	Type	评论
包含添加项	UB[1]	当等于 1 时，包含颜色添加值
包含多项式项	UB[1]	当等于 1 时，包含颜色乘法值
比特数	UB[4]	每个值字段中的比特数
红色多词项	如果 HasMultTerms = 1，则 SB[Nbits]	红色乘值
绿色多词项	如果 HasMultTerms = 1，则 SB[Nbits]	绿色乘值

蓝色多术语	如果 HasMultTerms = 1，则 SB[Nbits] 蓝色乘值
红色加项	如果 HasAddTerms = 1，则 SB[Nbits] 红色加值
绿色添加项	如果 HasAddTerms = 1，则 SB[Nbits] 绿色添加值
蓝色添加项	如果 HasAddTerms = 1，则 SB[Nbits] 蓝色加值

带 alpha 记录的色彩转换

CXFORMWITHALPHA 记录扩展了 CXFORM 的功能，允许对 alpha 通道以及红、绿、蓝通道应用色彩转换。

以下有两种可能的变换类型：

- 乘法变换
- 加法变换

乘法变换会将红色、绿色、蓝色和 alpha 分量乘以一个 8.8 的定点值。1.0 的定点表示为 0x100 或 256 十进制。因此，乘法运算的结果应该除以 256。

对于任何颜色（R，G，B，A），变换后的颜色（R'，G'，B'，A'）计算如下：

```
R’ = (R * RedMultTerm) / 256 G’ = (G *
GreenMultTerm) / 256 B’ = (B *
BlueMultTerm) / 256 A’ = (A *
AlphaMultTerm) / 256
```

CXFORMWITHALPHA 记录通常用于显示部分透明的对象，通过将 alpha 通道乘以介于 0 到 256 之间的某个值来实现。

添加变换将固定值（正数或负数）添加到正在显示的对象的红色、绿色、蓝色和 alpha 分量中。如果结果大于 255，则结果被限制为 255。如果结果小于零，则结果被限制为零。

对于任何颜色（R，G，B，A），变换后的颜色（R'，G'，B'，A'）计算如下：

```
R’ = max(0, min(R + RedAddTerm, 255)) G’ = max(0, min(G
+ GreenAddTerm, 255)) B’ = max(0, min(B + BlueAddTerm,
255)) A’ = max(0, min(A + AlphaAddTerm, 255))
```

☹

加法和乘法变换可以按以下方式组合。首先执行乘法操作：

```
'R' = max(0, min(((R * RedMultTerm) / 256) + RedAddTerm, 255)) 'G' = max(0, min(((G * GreenMultTerm) / 256) + GreenAddTerm, 255)) 'B' = max(0, min(((B * BlueMultTerm) / 256) + BlueAddTerm, 255)) 'A' = max(0, min(((A * AlphaMultTerm) / 256) + AlphaAddTerm, 255))
```

与 CXFORM 记录类似，CXFORMWITHALPHA 记录是字节对齐的。

字段	Type	评论
包含添加项	UB[1]	当等于 1 时，包含颜色添加值
包含多项式项	UB[1]	当等于 1 时，包含颜色乘法值
比特数	UB[4]	每个值字段中的比特数
红色多词项	如果 HasMultTerms = 1，则 SB[Nbits]	红色乘值
绿色多词项	如果 HasMultTerms = 1，则 SB[Nbits]	绿色乘值
蓝色多术语	如果 HasMultTerms = 1，则 SB[Nbits]	蓝色乘值
AlphaMultTerm	如果 HasMultTerms = 1，则 SB[Nbits]	Alpha 乘值
红色加项	如果 HasAddTerms = 1，则 SB[Nbits]	红色加值
绿色添加项	如果 HasAddTerms = 1，则 SB[Nbits]	绿色添加值
蓝色添加项	如果 HasAddTerms = 1，则 SB[Nbits]	蓝色加值
添加术语	如果 HasAddTerms = 1，则 SB[Nbits]	透明度增加值

第二章：SWF 结构概要

本章提供了构成 SWF 文件元素的摘要。

SWF 头部

所有 SWF 文件都以以下头部开始。类型在第一章：基本数据类型中定义。

字段	Type	评论
签名	UI8	签名字节：“F”表示未压缩，“C”表示 zlib 压缩的 SWF（仅限 SWF 6 及以后版本） “Z”表示 LZMA 压缩的 SWF（仅限 SWF 13 及以后版本）
签名	UI8	签名字节始终为“W”
签名	UI8	签名字节始终为“S”
版本	UI8	单字节文件版本（例如，0x06 表示 SWF 6）
文件长度	UI32	整个文件的长度（字节）
帧大小	RECT	帧大小（twips）
帧率	UI16	每秒 8.8 固定帧数的帧延迟
帧数	UI16	文件中帧的总数

头部以一个三字节的签名开始，该签名可以是以下之一：

- 0x46, 0x57, 0x53（“FWS”）。FWS 签名表示未压缩的 SWF 文件。
- 0x43, 0x57, 0x53（“CWS”）。CWS 表示文件的第一 8 字节之后（即 FileLength 字段之后）的整个文件使用 ZLIB 开放标准进行了压缩。ZLIB 库使用的数据格式由请求评论（RFC）文档 1950 至 1952 描述。CWS 文件压缩仅允许在 SWF 6 或更高版本中使用。
- 0x5a, 0x57, 0x53（“ZWS”）。ZWS 表示文件的第一 8 字节之后（即 FileLength 字段之后）的整个文件使用 LZMA 开放标准进行了压缩：http://www.7-

zip.org/sdk.html. 仅在 SWF 13 或更高版本中允许 ZWS 文件压缩。

紧接着签名的是一个单字节版本号。版本号不是一个 ASCII 字符，而是一个 8 位数字。例如，对于 SWF 4，版本字节是 0x04，而不是 ASCII 字符“4”（0x34）。

FileLength 字段表示 SWF 文件的总长度，包括头部。如果这是一个未压缩的 SWF 文件（FWS 签名），则 FileLength 字段应与文件大小完全匹配。如果这是一个压缩的 SWF 文件（CWS 签名），则 FileLength 字段表示文件解压缩后的总长度，因此通常不匹配文件大小。拥有未压缩的大小可以使解压缩过程更高效。

FrameSize 字段定义了屏幕显示的宽度和高度。该字段以 RECT 结构存储，这意味着其大小可能根据编码坐标所需的位数而变化。FrameSize RECT 始终具有 Xmin 和 Ymin 值为 0；Xmax 和 Ymax 成员定义宽度和高度（请参阅使用位值）。

FrameRate 是每秒期望的播放帧数。如果 Flash Player 在慢速或繁忙的 CPU 上运行，则此速率可能无法保证。

帧数是该 SWF 文件中的总帧数。

SWF 文件结构

在头部之后是一系列标记数据块。所有标记都采用相同的格式，因此任何解析 SWF 文件的程序都可以跳过它不理解的块。块内的数据可以指向块内的偏移量，但不能指向另一个块中的偏移量。这种能力使得标记可以被工具移除、插入或修改，这些工具处理 SWF 文件。

文件属性标记仅适用于 SWF 8 及更高版本。

标题	文件属性 tag	Tag	Tag	...	结束标签

标签格式

每个标签都以标签类型和长度开始。标签头格式可以是短格式或长格式。短标签头用于数据长度不超过 62 字节的标签。长标签头，带有 32 位有符号长度字段，可以用于任何大小不超过 2GB 的标签，远大于目前实际所需的容量。

RECORDHEADER（短格式）

字段	Type	评论	
标签代码和长度	UI16	高 10 位：标签类型	低 6 位：标签长度

注意：TagCodeAndLength 字段是一个双字节词，不是一个 10 位位字段后跟 6 位位字段。
SWF 文件的 Little-endian 字节序使得这两种布局不同。

TagCodeAndLength 字段中指定的长度不包括开始标签的 RECORDHEADER。

如果标签长度为 63 字节或更长，则存储在长标签头中。长标签头由长度为 0x3f 的短标签头和随后的 32 位长度组成。

RECORDHEADER（长）

字段	Type	评论
标签代码和长度	UI16	将 0x3F 的标签类型和长度打包在一起，如短头中所示
长度	UI32	标签的长度

定义和控制标签

SWF 文件中的标签分为以下两类：

- 定义标签定义了 SWF 文件的内容——形状、文本、位图、声音等。每个定义标签都会为其定义的内容分配一个唯一的 ID，称为字符 ID。Flash Player 然后将字符存储在称为字典的仓库中。定义标签本身并不会导致任何内容被渲染。

- 控制标签用于创建和操作字典中字符的渲染实例，并控制文件流。

SWF 文件中的标签顺序

一般而言，SWF 中的标签可以按任何顺序出现。但是，你必须遵守以下规则：

- 对于 SWF 8 及以后的版本，FileAttributes 标签必须是 SWF 文件中的第一个标签。
- 一个标签只能依赖于它之前的标签。一个标签不应依赖于文件中较后的标签。
- 定义角色的定义标签必须出现在引用该角色的任何控制标签之前。
- 流媒体音频标签必须按顺序排列。顺序错误的流媒体音频标签会导致音频播放顺序混乱。
- End 标签总是 SWF 文件中的最后一个标签。

字典

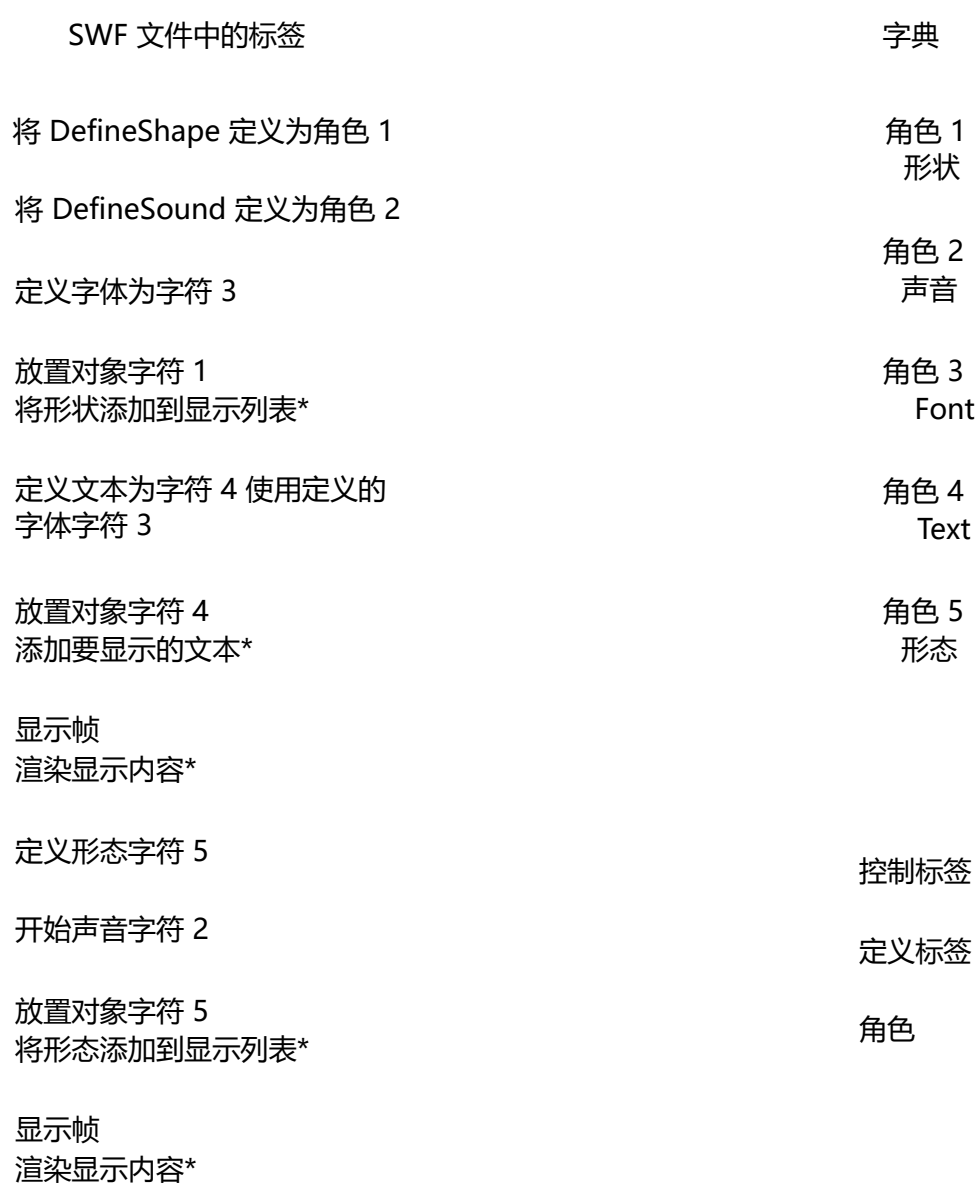
字典是定义并可供控制标签使用的字符的存储库。构建和使用字典的过程如下：

1. 定义标签定义了一些内容，例如形状、字体、位图或声音。
2. 定义标签为内容分配一个唯一的 CharacterId。
3. 内容在字典下以 CharacterId 保存。
4. 控制标签使用 CharacterId 从字典中检索内容，并对内容执行某些操作，例如显示形状或播放声音。

每个定义标签都必须指定一个唯一的 ID。不允许重复的 ID。通常，第一个 CharacterId 是 1，第二个 CharacterId 是 2，以此类推。数字零（0）是特殊的，被视为空字符。

控制标签并非唯一引用字典的标签。定义标签可以使用字典中的字符来定义更复杂的字符。例如，DefineButton 和 DefineSprite 标签引用其他字符来定义其内容。DefineText 标签可以引用字体字符来选择不同的字体用于文本。

下图展示了定义标签、控制标签和字典之间的典型交互：



处理 SWF 文件

Flash Player 会处理 SWF 文件中的所有标签，直到遇到 ShowFrame 标签。此时，显示列表被复制到屏幕上，Flash Player 将处于空闲状态，直到处理下一帧的时间。第一帧的内容是执行第一个 ShowFrame 标签之前所有控制标签操作的累积效果。第二帧的内容是执行从文件开始到第二个 ShowFrame 标签的所有控制标签操作的累积效果，依此类推。

文件压缩策略

由于 SWF 文件通常通过网络连接传输，因此它们应该尽可能紧凑。为此，采用了多种技术，包括以下项目：

- 重复使用——字符字典的结构使得在 SWF 文件中重复使用元素变得容易。例如，形状、按钮、声音、字体或位图可以存储在文件中一次，然后多次引用。

- 压缩——通过使用高效的增量编码方案对形状进行压缩；通常假设线的第一个坐标是上一条线的最后一个坐标。距离也经常相对于最后的位置表示。

- 默认值——一些结构，如矩阵和颜色转换，有一些字段被使用得更为频繁。例如，对于一个矩阵，最常见的字段是平移字段。缩放和旋转则不太常见。因此，如果缩放字段不存在，则默认为 100%。如果旋转字段不存在，则默认没有旋转。这种默认值的用法有助于最小化文件大小。

- 更改编码——按照规则，SWF 文件只存储状态之间的变化。这体现在形状数据结构和显示列表使用的 place-move-remove 模型中。

- 形状数据结构——形状数据结构使用独特的结构来最小化形状的大小，并有效地在屏幕上渲染抗锯齿形状。

摘要

SWF 文件由一个头部和随后的一系列标签组成。标签分为定义标签和控制标签。定义标签定义了称为字符的对象，这些对象存储在字典中。控制标签操作字符，并控制文件流。

第三章：显示列表

显示 SWF 文件的一个帧是一个三阶段的过程：

1. 对象通过定义标签（如 DefineShape、DefineSprite 等）进行定义。每个对象都有一个唯一的 ID，称为字符，并存储在称为字典的仓库中。

2. 从字典中复制选定的字符并放置在显示列表中，显示列表是将在下一帧中显示的字符列表。

3. 完成后，使用 ShowFrame 将显示列表的内容渲染到屏幕上。

在显示列表中，每个字符都被分配一个深度值。深度值决定了字符的堆叠顺序。深度值较低的字符会显示在深度值较高的字符下方。深度值为 1 的字符将显示在堆叠的最底层。一个字符可以在显示列表中出现多次，但深度值不同。在任何给定深度上只能有一个字符。

在 SWF 1 和 2 版本中，显示列表是任何给定时间屏幕上存在的对象的扁平列表。在 SWF 3 及以后的版本中，显示列表是一个层次列表，其中显示列表上的一个元素可以有子元素列表。更多信息，请参阅 DefineSprite。

以下六个标签用于控制显示列表：

- PlaceObject　将字符添加到显示列表中。
- PlaceObject2 & PlaceObject3 添加角色到显示列表，或修改指定深度的角色。
- RemoveObject 从显示列表中删除指定的角色。
- RemoveObject2 删除指定深度的角色。
- 显示帧　渲染显示列表的内容到显示。

注意：较旧的标签 PlaceObject 和 RemoveObject 在 SWF 3 及以后版本中很少使用。

以下图示展示了显示过程。首先定义了三个对象：一个形状、一个文本对象和一个精灵。这些对象被赋予角色 ID 并存储在字典中。角色 1（形状）被放置在深度 1，栈底，在帧渲染时将被所有其他角色遮挡。角色 2（文本）被放置两次；一次在深度 2，一次在深度 4，栈顶。

角色三（精灵）放置在深度 3。

剪裁层

Flash Player 支持在显示列表中的一种特殊对象，称为剪裁层。作为剪裁层的字符不会被显示；相反，它剪裁（或遮罩）其上方的字符。PlaceObject2 中的 ClipDepth 字段指定了剪裁层遮罩的最顶层深度。

例如，如果将一个形状放置在深度 1，且 ClipDepth 为 4，则所有深度在 1 以上，包括深度 4 的形状都将被深度 1 的形状遮挡。放置在深度 4 以上的字符不会被遮挡。

使用显示列表

以下步骤创建并显示动画：

1. 使用定义标签定义每个字符。
   每个字符都分配一个唯一的字符 ID，并添加到字典中。

2. 使用 PlaceObject2 标签将每个字符添加到显示列表中。每个 PlaceObject2 标签指定要显示的字符，以及以下属性：

   - 深度值，用于控制放置的字符的堆叠顺序。深度值较低的字符看起来位于深度值较高的字符之下。深度值为 1 表示字符显示在堆叠底部。在任何给定深度上只能有一个字符。

   - 一个变换矩阵，用于确定放置的角色的位置、缩放、因子和旋转角度。相同的角色可以多次（在不同深度）放置，使用不同的变换矩阵。

   - 一个可选的颜色变换，用于指定放置的角色所应用的颜色效果。
     颜色效果包括透明度和颜色偏移。

- 一个可选的名称字符串，用于标识放置用于 SetTarget 动作的角色。SetTarget 用于在精灵对象内执行动作。

- 一个可选的 ClipDepth 值，用于指定放置角色时将被遮挡的最顶层深度。

- 一个可选的比率值，用于控制放置时变形角色的显示方式。比率为零时显示变形的起始状态，比率为 65535 时显示变形的结束状态。

3. 使用 ShowFrame 标签将显示列表的内容渲染到屏幕上。

4. 使用 PlaceObject2 标签修改显示列表中的每个角色。
   每个 PlaceObject2 都会为给定深度的角色分配一个新的变换矩阵。由于每个深度只能有一个角色，因此没有指定角色 ID。

5. 使用 ShowFrame 标签来显示角色在新的位置。

   对动画的每一帧重复步骤 4 和 5。

   如果角色在帧与帧之间没有变化，则不需要在每一帧后替换未更改的角色。

6. 使用 RemoveObject2 标签从显示列表中移除每个字符。只需深度值即可识别要移除的字符。

显示列表标签

显示列表标签用于向显示列表添加字符和字符属性。

PlaceObject

PlaceObject 标签向显示列表添加一个角色。CharacterId 识别要添加的角色。Depth 字段指定角色的堆叠顺序。Matrix 字段指定角色的位置、缩放和旋转。如果 PlaceObject 标签的大小超出了变换矩阵的末尾，则假定记录中附加了一个 ColorTransform 字段。ColorTransform 字段指定应用于角色的颜色效果（如透明度）。相同的角色可以多次添加到显示列表中，具有不同的深度和变换矩阵。

PlaceObject 在 SWF 3 及以后的版本中很少使用；它被 PlaceObject2 和 PlaceObject3 取代。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 4
字符 ID	UI16	要放置的字符 ID
深度	UI16	角色深度
矩阵	矩阵	变换矩阵数据
颜色变换（可选）	CXFORM	颜色变换数据

PlaceObject2

PlaceObject2 标签扩展了 PlaceObject 标签的功能。PlaceObject2 标签既可以向显示列表添加一个角色，也可以修改已存在于显示列表中的角色的属性。从 SWF 4 到 SWF 5，PlaceObject2 标签有所变化。在 SWF 5 中，增加了剪辑动作。

标签以一组标志开始，指示标签中包含哪些字段。可选字段包括 CharacterId、Matrix、ColorTransform、Ratio、ClipDepth、Name 和 ClipActions。Depth 字段是唯一始终必需的字段。

深度值决定了角色的堆叠顺序。深度值较低的角色将显示在深度值较高的角色之下。深度值为 1 表示角色显示在堆叠的底部。任何给定的深度只能有一个角色。这意味着已存在于显示列表中的角色可以仅通过其深度来识别（即，不需要 CharacterId）。

PlaceFlagMove 和 PlaceFlagHasCharacter 标签表示是否正在向显示列表中添加新角色，或者修改显示列表上已存在的角色。标志的含义如下：

- PlaceFlagMove = 0 且 PlaceFlagHasCharacter = 1 表示将具有 CharacterId ID 的新角色放置在指定的深度处。其他字段设置新角色的属性。
- PlaceFlagMove = 1 且 PlaceFlagHasCharacter = 0 表示修改指定深度的角色。其他字段修改该角色的属性。

  由于任何给定深度只能有一个角色，因此不需要 CharacterId。
- PlaceFlagMove = 1 和 PlaceFlagHasCharacter = 1 指定深度的角色将被移除，并在此深度放置一个新的角色（角色 ID 为 CharacterId）。其他字段设置新角色的属性。

帧替换了指定深度的角色的变换矩阵。例如，一个在一系列帧中移动的角色，在第一帧中设置 PlaceFlagHasCharacter，在后续帧中设置 PlaceFlagMove。

☹

在后续帧中设置 PlaceFlagMove。第一帧将新角色放置在期望的深度，并设置初始变换矩阵。后续

PlaceObject2 的可选字段具有以下含义：

- CharacterId 字段指定要添加到显示列表中的角色。CharacterId 仅在添加新角色时使用。如果修改已存在于显示列表中的角色，则 CharacterId 字段不存在。

- Matrix 字段指定要添加或修改的角色的位置、缩放和旋转。
- ColorTransform 字段指定应用于要添加或修改的角色的颜色效果。
- 比率字段指定要添加或修改的角色的形态比率。此字段仅适用于使用 DefineMorphShape 定义的角色，并控制形态的进度。比率为零时显示形态的开始状态。比率为 65535 时显示形态的结束状态。对于介于零和 65535 之间的值，Flash Player 将在开始和结束形状之间进行插值，并显示中间形状。

- ClipDepth 字段指定将被添加的角色遮挡的最顶层深度。ClipDepth 为零表示这不是一个裁剪角色。

- 名称字段指定要添加或修改的角色的名称。此字段通常与精灵角色一起使用，用于标识精灵以执行 SetTarget 操作。它允许主文件（或其他精灵）在精灵内部执行操作（参见第 13 章：精灵和电影剪辑）。

- 仅适用于放置精灵角色的 ClipActions 字段定义了在发生某些事件时将被调用的一个或多个事件处理器。

最小文件格式版本为 SWF 3。

字段	Type	评论
标题	记录头	标签类型 = 26
PlaceFlagHasClipActions	UB[1]	SWF 5 及以后版本：具有剪辑动作（仅限精灵角色）否则：始终为 0
具有剪辑深度标志	UB[1]	具有剪辑深度
具有名称标志	UB[1]	具有名称
标记具有比例	UB[1]	具有比例
标记具有颜色变换	UB[1]	具有颜色变换

放置标志具有矩阵	UB[1]	矩阵存在
标记放置有角色	UB[1]	放置一个角色
标记移动	UB[1]	定义要移动的角色
深度	UI16	角色深度
字符 ID	如果 PlaceFlagHasCharacter，UI16	要放置的字符 ID
矩阵	如果 PlaceFlagHasMatrix，矩阵	变换矩阵数据
颜色变换	如果有颜色变换标志，CXFORMWITHALPHA	颜色变换数据
比率	如果有比率标志，UI16	
Name	如果有名称标志，STRING	角色名称
裁剪深度	如果 PlaceFlagHasClipDepth，UI16	裁剪深度（见“裁剪层”）
影片动作	如果设置了 PlaceFlagHasClipActions，则 CLIPACTIONS	SWF 5 及以后版本：影片动作数据

影片动作仅对放置精灵角色有效。影片动作定义了精灵角色的事件处理器。

字段	Type	评论
保留	UI16	必须为 0
所有事件标志	CLIPEVENTFLAGS	这些剪辑动作中使用的所有事件
剪辑动作记录	CLIPACTIONRECORD [一个或多个]	单个事件处理器
录像动作结束标志	如果 SWF 版本<=5，则 UI16，如果 SWF 版本>=6，则 UI32	必须为 0

CLIPACTIONRECORD

字段	Type	评论
事件标志	CLIPEVENTFLAGS	此处理程序应用的事件
ActionRecordSize	UI32	字节偏移量从此字段末尾

☹

		到下一个 CLIPACTIONRECORD（或 ClipActionEndFlag）
键码	如果 EventFlags 包含 ClipEventKeyPress: UI8，否则不存在	要捕获的键码（见“DefineButton2”）
动作	ACTIONRECORD [一个或多个]	要执行的操作

放置对象 3

The PlaceObject3 标签扩展了 PlaceObject2 标签的功能。PlaceObject3 添加了以下新特性：

- The PlaceFlagHasClassName field indicates that a class name will be specified, indicating the type of object to place. Because we no longer use ImportAssets in ActionScript 3.0, there needed to be some way to place a Timeline object using a class imported from another SWF, which does not have a 16-bit character ID in the instantiating SWF. Supported in Flash Player 9.0.45.0 and later.

- The PlaceFlagHasImage field indicates the creation of native Bitmap objects on the display list. When PlaceFlagHasClassName and PlaceFlagHasImage are both defined, this indicates a Bitmap class to be loaded from another SWF. Immediately following the flags is the class name (as above) for the BitmapData class in the loaded SWF. A Bitmap object will be placed with the named BitmapData class as it's internal data. When PlaceFlagHasCharacter and PlaceFlagHasImage are both defined, this indicates a Bitmap from the current SWF. The BitmapData to be used as its internal data will be defined by the
  以下字符 ID。这仅在 BitmapData 与类相关联时发生。如果没有与 BitmapData 相关联的类，应使用带有位图填充的 DefineShape。自 Flash Player 9.0.45.0 版本开始支持。

- PlaceFlagHasCacheAsBitmap 字段指定 Flash Player 是否应将显示对象内部缓存为位图。当对象不经常更改时，缓存可以提高渲染速度。

- 可以指定多种不同的混合模式作为正常 alpha 合成之外的替代方案。以下混合模式得到支持：
  - Add
  - 图层透明度
  - 调亮/调暗
  - 叠加/差值
  - 相乘/擦除
  - 滚屏/硬光

- 减去
- 反转

- 可以将多个位图滤镜应用于显示对象。添加滤镜意味着显示对象将被缓存为位图。以下支持位图滤镜：
  - 凸起
  - 投影阴影
  - Blur
  - Glow
  - 颜色矩阵
  - 渐变斜角
  - 卷积
  - 渐变发光效果

最小文件格式版本为 SWF 8。

放置对象 3

字段	Type	评论
标题	记录头	标签类型 = 70
PlaceFlagHasClipActions	UB[1]	SWF 5 及以后版本：具有剪辑动作（仅限精灵角色）否则：始终为 0
具有剪辑深度标志	UB[1]	具有剪辑深度
具有名称标志	UB[1]	具有名称
标记具有比例	UB[1]	具有比例
标记具有颜色变换	UB[1]	具有颜色变换
放置标志具有矩阵	UB[1]	矩阵存在
标记放置有角色	UB[1]	放置一个角色
标记移动	UB[1]	定义要移动的角色

保留	UB[1]	必须为 0
设置不透明背景标志	UB[1]	具有不透明背景。SWF 11 及以上版本。
设置可见性标志	UB[1]	具有可见性标志。SWF 11 及以上版本。
放置标志具有图像	UB[1]	如果有放置标志具有类名或字符 ID 的位图，则使用类名。如果放置标志具有字符，则使用 CharacterId
放置标志具有类名	UB[1]	具有放置对象类名的名称
PlaceFlagHasCacheAsBitmap	UB[1]	启用位图缓存
PlaceFlagHasBlendMode	UB[1]	具有混合模式
PlaceFlagHasFilterList	UB[1]	具有过滤器列表
深度	UI16	角色深度
类名	如果 PlaceFlagHasClassName 或 (PlaceFlagHasImage 且 PlaceFlagHasCharacter)，则	要放置的类名
字符 ID	如果 PlaceFlagHasCharacter，UI16	要放置的字符 ID
矩阵	如果 PlaceFlagHasMatrix，矩阵	变换矩阵数据
颜色变换	如果有颜色变换标志，CXFORMWITHALPHA	颜色变换数据
比率	如果有比率标志，UI16	
Name	如果有名称标志，STRING	角色名称
裁剪深度	如果 PlaceFlagHasClipDepth，UI16	裁剪深度（见裁剪层）
表面过滤器列表	如果 PlaceFlagHasFilterList，则 FILTERLIST	该对象的过滤器列表
混合模式	如果 PlaceFlagHasBlendMode，UI8	0 或 1 = 正常 2 = 层； 3 = 乘； 4 = 屏幕模式； 5 = 加亮；

		6 = 深色化 7 = 差分； 8 = 加； 9 = 减； 10 = 取反； 11 = alpha； 12 = 删除； 13 = 叠加； 14 = 硬光；值 15 至 255 为保留。
位图缓存	如果 PlaceFlagHasCacheAsBitmap，UI8	0 = 禁用位图缓存；1-255 = 启用位图缓存
可见	如果 PlaceFlagHasVisible，UI8	0 = 使位置不可见，1 = 使位置可见
背景颜色	如果 PlaceFlagHasVisible，RGBA	
影片动作	如果设置了 PlaceFlagHasClipActions ，则 CLIPACTIONS	SWF 5 及以后版本：影片动作数据
过滤列表		
字段	Type	评论
过滤器数量	UI8	过滤器数量
过滤器	过滤器数量[NumberOfFilters]	过滤器列表

过滤器

字段	Type	评论
过滤器 ID	UI8	0 = 具有阴影滤镜 1 = 具有模糊滤镜 2 = 具有发光滤镜 3 = 具有斜角滤镜 4 = 具有渐变发光滤镜 5 = 具有卷积滤镜 6 = 具有颜色矩阵滤镜 7 = 具有渐变斜面滤镜

阴影滤镜	如果 FilterID = 0，则 DROPSHADOWF投影滤镜	
模糊滤镜	如果 FilterID = 1，则 BLURFILTER	模糊滤镜
光照滤镜	如果滤镜 ID = 2，光照滤镜	光照滤镜
斜边滤镜	如果 FilterID = 3，斜面滤镜	斜面滤镜
渐变发光滤镜	如果 FilterID = 4，渐变发光滤镜	渐变发光滤镜
卷积滤镜	如果滤镜 ID = 5，卷积滤镜	卷积滤镜
颜色矩阵滤镜	如果 FilterID = 6，颜色矩阵过滤器	颜色矩阵过滤器
渐变斜角过滤器	如果 FilterID = 7，渐变斜角过滤器	渐变斜角滤镜

颜色矩阵滤镜

颜色矩阵滤镜对显示列表对象的像素应用颜色变换。给定显示列表对象中的输入 RGBA 像素，颜色变换的计算方式如下：

结果的 RGBA 值被饱和。

矩阵值从左到右存储，每行从上到下。最后一行始终假设为（0,0,0,0,1），无需存储。

颜色矩阵过滤器

字段	Type	评论
矩阵	FLOAT[20]	颜色矩阵值

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} r0 & r1 & r2 & r3 & r4 \\ g0 & g1 & g2 & g3 & g4 \\ b0 & b1 & b2 & b3 & b4 \\ a0 & a1 & a2 & a3 & a4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

卷积滤镜

卷积滤波器是一种二维离散卷积。它应用于显示对象的每个像素。在以下数学表示中，F 是输入像素平面，G 是输入矩阵，H 是输出像素平面：

☹

$$H_{x,y} = \sum_{j=0}^{\text{矩阵 X - 1}} \sum_{i=0}^{\text{矩阵 Y - 1}} \frac{Fx + i \cdot \frac{\text{MatrixX}}{2} \cdot y + j \cdot \frac{\text{MatrixY}}{2} + \text{偏置 Gi j}}{\text{除数}}$$

☹

卷积应用于 RGBA 颜色组件的每个组件，然后饱和，除非设置了 PreserveAlpha 标志；在这种情况下，alpha 通道值不修改。钳位标志指定如何处理输入像素平面之外的像素。如果设置为 false，则使用 DefaultColor 值，否则像素被钳位到最近的合法输入像素。

卷积滤波器

字段	Type	评论
矩阵 X	UI8	水平矩阵大小
矩阵 Y	UI8	垂直矩阵大小
除数	FLOAT	应用到矩阵值的除数
Bias	FLOAT	应用到矩阵值的偏置
矩阵	FLOAT[矩阵 X * 矩阵 Y]	矩阵值
默认颜色	RGBA	图像外像素的默认颜色
保留	UB[6]	必须为 0
卡钳	UB[1]	卡钳模式
保留透明度	UB[1]	保留 alpha 通道

模糊滤镜

模糊滤镜基于子像素精确的中值滤波器（也称为箱式滤波器）。该滤镜应用于 RGBA 颜色通道的每个通道。

简单非子像素精确中值滤波器的一般数学表示如下，并且可以很容易地扩展以支持子像素精度。

该表示假设 BlurX 和 BlurY 是奇数，以便得到与 Flash Player 相同的结果。
在 Flash Player 中，滤镜窗口始终位于像素中心。

当遍历次数设置为 3 时，它接近高斯模糊滤镜。更高的遍历次数是可能的，但出于性能考虑，Adobe 不推荐这样做。

☹

模糊滤镜

字段	Type	评论
模糊 X	已修复	水平模糊量
模糊 Y	已修复	垂直模糊量
通过	UB[5]	模糊次数
保留	UB[3]	必须为 0

投影滤镜

滴影滤镜基于与模糊滤镜相同的中值滤波器，但滤镜仅应用于 alpha 颜色通道以获得阴影像素平面。

角度参数以弧度为单位。当角度设置为 0 时，阴影显示在对象的右侧。距离以像素为单位。如果使用子像素值，阴影像素平面的值将进行双线性插值。

阴影强度归一化为 1.0 的定点数。通过将阴影像素平面中的每个值相乘来应用强度值。

提供了多种合成选项，以支持内阴影和外阴影，在常规或敲除模式下。

每个像素的最终颜色值是通过将提供的 RGBA 颜色值的颜色通道与阴影像素平面中的相关值相乘得到的。通过使用指定的合成模式之一，将得到的像素值合成在原始输入像素平面上。

滴影滤镜

字段	Type	评论
滴影颜色	RGBA	滴影颜色
模糊 X 轴	已修复	水平模糊量
模糊 Y	已修复	垂直模糊量
角度	已修复	投影的弧度角度
距离	已修复	投影的距离
强度	FIXED8	阴影强度

内部阴影	UB[1]	内阴影模式
击穿	UB[1]	击穿模式
合成源	UB[1]	复合源。总是 1
通过	UB[5]	模糊次数

光照滤镜

光晕滤镜的工作方式与阴影滤镜相同，只是没有距离和角度参数。因此，它可以运行得稍微快一些。

GLOWFILTER

字段	Type	评论
光晕颜色	RGBA	滴影颜色
模糊 X 轴	已修复	水平模糊量
模糊 Y	已修复	垂直模糊量
强度	FIXED8	光芒的强度
内发光	UB[1]	内发光模式
击穿	UB[1]	拦击模式
合成源	UB[1]	合成源。总是 1
通过	UB[5]	模糊次数

斜面滤镜

斜面过滤器在显示列表对象上创建平滑的斜面。

BEVELFILTER

字段	Type	评论
阴影颜色	RGBA	滴影颜色
高光颜色	RGBA	高光颜色（颜色）
模糊 X 轴	已修复	水平模糊量

模糊 Y	已修复	垂直模糊量
角度	已修复	投影的弧度角度
距离	已修复	投影的距离
强度	FIXED8	阴影强度
内部阴影	UB[1]	内阴影模式
击穿	UB[1]	拦击模式
合成源	UB[1]	复合源。总是 1
OnTop	UB[1]	OnTop 模式
通过	UB[4]	模糊次数

渐变发光和渐变斜角滤镜

渐变发光和渐变斜角滤镜是普通发光和斜角滤镜的扩展，允许指定渐变而不是单一颜色。不是将单一颜色值乘以阴影像素平面的值，而是将阴影像素平面的值直接映射到渐变斜坡中，以获得结果颜色像素值，然后使用指定的合成模式进行合成。

渐变发光滤镜

字段	Type	评论
颜色数量	UI8	渐变中的颜色数量
渐变颜色	RGBA[颜色数量]	渐变颜色
渐变比例	UI8[颜色数量]	渐变比例
模糊 X 轴	已修复	水平模糊量
模糊 Y	已修复	垂直模糊量
角度	已修复	渐变发光的弧度角
距离	已修复	渐变发光的距离
强度	FIXED8	渐变发光的强度
内部阴影	UB[1]	内发光模式

击穿	UB[1]	拦击模式
合成源	UB[1]	复合源。总是 1
OnTop	UB[1]	OnTop 模式
通过	UB[4]	模糊次数
颜色数量	UI8	渐变中的颜色数量
渐变颜色	RGBA[颜色数量]	渐变颜色
渐变比例	UI8[颜色数量]	渐变比例
模糊 X 轴	已修复	水平模糊量
模糊 Y	已修复	垂直模糊量
角度	已修复	渐变斜面的角度
距离	已修复	渐变斜面的距离
强度	FIXED8	渐变斜面的强度
内部阴影	UB[1]	内斜面模式
击穿	UB[1]	拦击模式

GRADIENTBEVELFILTER

字段	Type	评论
合成源	UB[1]	复合源。总是 1
OnTop	UB[1]	OnTop 模式
通过	UB[4]	模糊次数

剪辑事件标志

CLIPEVENTFLAGS 序列指定一个或多个精灵事件，事件处理器将应用于这些事件。在 SWF 5 及之前版本中，CLIPEVENTFLAGS 为 2 字节；在 SWF 6 及以后版本中，它为 4 字节。

剪辑事件标志

字段	Type	评论
键盘抬起事件	UB[1]	键抬起事件
键盘按下事件	UB[1]	键按下事件
鼠标抬起事件	UB[1]	鼠标抬起事件
鼠标按下事件	UB[1]	鼠标按下事件
鼠标移动事件	UB[1]	鼠标移动事件
剪辑卸载事件	UB[1]	剪辑卸载事件
Enter 帧事件	UB[1]	帧事件
影片事件加载	UB[1]	影片加载事件
影片事件拖动	UB[1]	SWF 6 及以后版本：鼠标拖动事件。否则：始终为 0
影片事件滚动退出	UB[1]	SWF 6 及以后版本：鼠标滚动退出事件。否则：始终为 0
播放事件滚动	UB[1]	SWF 6 及以上版本：鼠标悬停事件。否则：始终为 0
播放事件释放外部	UB[1]	SWF 6 及以上版本：鼠标释放外部事件。否则：始终为 0
裁剪事件释放	UB[1]	SWF 6 及以上版本：鼠标释放事件。否则：始终为 0
裁剪事件按下	UB[1]	SWF 6 及以上版本：鼠标按下事件。否则：始终为 0
ClipEventInitialize	UB[1]	SWF 6 及以后版本：初始化事件。否则：始终为 0
ClipEventData	UB[1]	接收数据事件
保留	如果 SWF 版本 >=6，未定义行为[5]	总是 0
影片事件构造	如果 SWF 版本 >=6，未定义行为[1]	SWF 7 及以后版本：构造事件 否则：始终为 0

剪辑按键事件	如果 SWF 版本 >=6，未定义行为[1]	键盘按键事件
录像事件拖出	如果 SWF 版本 >=6，未定义行为[1]	鼠标拖出事件
保留	如果 SWF 版本 >= 6，UB[8]	总是 0

SWF 6 中添加的额外事件对应于 Flash 编程工具中的按钮电影剪辑，这些是可以通过脚本以与按钮相同方式脚本化的精灵（参见 BUTTONCONDACTION）。DragOut 通过 Press 事件对应于按钮动作条件中的按钮状态转换事件；它们之间的对应关系在按钮事件描述中显示（参见“事件、状态转换和动作”）。

KeyDown 和 KeyUp 事件并不特定于某个键；这些事件的处理器在键盘上的任何键（可能除了一些特殊键）从按下状态或释放状态转换时都会执行。要找出是哪个键发生了转换，处理器中的动作应调用 ActionScript Key 对象的方法。

KeyPress 事件与 KeyDown 和 KeyUp 的工作方式不同。KeyPress 事件特定于某个键或 ASCII 字符（在 CLIPACTIONRECORD 中指定）。KeyPress 事件以相同的方式工作（参见 BUTTONCONDACTION）。

RemoveObject

The RemoveObject 标签从显示列表中删除指定的字符（在指定的深度处）。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 5
字符 ID	UI16	要删除的字符的 ID
深度	UI16	角色深度

RemoveObject2

The RemoveObject2 tag removes the character at the specified depth from the display list. The minimum file format version is SWF 3.

字段	Type	评论
标题	记录头	Tag type = 28

深度	UI16	角色深度

显示帧

ShowFrame 标签指示 Flash Player 显示显示列表的内容。文件将在一个帧的持续时间内暂停。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 1

第四章：控制标签

控制标签管理 SWF 文件中文件、帧和播放的一些总体方面。

设置背景颜色

SetBackgroundColor 标签用于设置显示的背景颜色。最低文件格式版本为 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 9
背景颜色	RGB	显示背景颜色

帧标签

帧标签标签用于给当前帧指定名称。ActionGoToLabel 使用此名称来识别帧。

最小文件格式版本为 SWF 3。

字段	Type	评论
标题	记录头	标签类型 = 43
Name	字符串	帧标签

在版本 6 或更高版本的 SWF 文件中，FrameLabel 标签的扩展名为命名锚点。命名锚点是一种特殊的帧标签，除了使用 ActionGoToLabel 标记帧以进行查找外，还可以使用 HTML 锚点语法标记帧进行查找。Adobe Flash Player 的浏览器插件版本从 6 版本开始，将检查浏览器地址栏中的锚点规范（形式为 URL 的末尾短语）。

如果在地址栏中存在锚点规范，Flash Player 将从包含指定相同名称的 FrameLabel 标签的帧开始播放；如果没有找到，则像往常一样从帧 1 开始播放。此外，当 Flash Player 到达包含命名锚点的帧时，它将在浏览器地址栏的 URL 中添加一个具有给定锚点名称的锚点规范。这确保了当用户在此处创建书签时，他们可以稍后返回到 SWF 文件中的同一位置，前提是文件中存在命名锚点的粒度。

要创建一个命名锚点，请在锚点名称的空终止符之后插入一个额外的非空字节。这仅适用于 SWF 6 或更高版本。

☹

命名锚点

字段	Type	评论
标题	记录头	标签类型 = 43
Name	空终止的字符串。 (0 为 NULL)	帧标签。
命名锚点标志。	UI8	总是 1。

保护

Protect 标签将文件标记为不可在创作环境中进行编辑。如果 Protect 标签不包含数据（标签长度 = 0），则 SWF 文件无法导入。如果该标签存在于文件中，任何创作工具都应阻止文件加载以进行编辑。

如果 Protect 标签包含数据（标签长度不为 0），则如果指定了正确的密码，SWF 文件可以导入。标签中的数据是一个以空字符终止的字符串，指定了 MD5 加密的密码。指定密码仅支持 SWF 5 或更高版本。

使用的 MD5 密码加密算法是由 Poul-Henning Kamp 编写的，并且可以自由分发。它位于 FreeBSD 的 src/lib/libcrypt/crypt-md5.c 目录中。EnableDebugger 标签也使用 MD5 密码加密算法。

最小文件格式版本是 SWF 2。

字段	Type	评论
标题	记录头	标签类型 = 24

End

结束标签标记文件结束。此标签必须始终位于文件中的最后一个标签。结束标签还用于结束精灵定义。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 0

导出资源

ExportAssets 标签使 SWF 文件的部分内容可供其他 SWF 文件导入（参见“导入资源”）。

例如，属于同一网站的十个 SWF 文件可以共享一个嵌入的自定义字体，如果其中一个文件嵌入字体并导出字体字符。每个导出的字符都由一个字符串标识。任何类型的字符都可以导出。

如果 ExportAssets 中字符的值之前已用不同的标识符导出，Flash Player 会将标签与后者关联。也就是说，如果 Flash Player 已经读取了 Tag1 的给定值，然后在 SWF 文件中再次读取相同的 Tag1 值，则使用第二个 Name1 值。

最小文件格式版本为 SWF 5。

字段	Type	评论
标题	记录头	标签类型 = 56
计数	UI16	导出资产数量
Tag1	UI16	要导出的第一个字符 ID
名称 1	字符串	第一个导出角色的标识符
TagN	UI16	导出最后一个字符的 ID
名称 N	字符串	最后导出角色的标识符

导入资源

ImportAssets 标签从另一个 SWF 文件导入角色。导入的 SWF 文件通过可找到的 URL 引用导出 SWF 文件。导入的资产就像在 SWF 文件中定义的角色一样添加到字典中。

导出 SWF 文件的 URL 可以是绝对路径或相对路径。如果是相对路径，则相对于导入 SWF 文件的路径进行解析。

ImportAssets 标签必须在帧中早于任何依赖于导入资源的后续标签。

ImportAssets 标签在 SWF 8 中被弃用；Flash Player 8 或更高版本忽略此标签。在 SWF 8 或更高版本中，请使用 ImportAssets2 标签代替。

最小文件格式版本为 SWF 5，最大文件格式版本为 SWF 7。

字段	Type	评论
标题	记录头	标签类型 = 57
URL	字符串	源 SWF 文件所在的 URL

计数	UI16	要导入的资产数量
Tag1	UI16	在导入 SWF 文件时使用的第一个导入角色的角色 ID（无需与导出 SWF 文件中的角色 ID 匹配）
名称 1	字符串	第一个导入角色的标识符（必须与导出 SWF 文件中的标识符匹配）
TagN	UI16	在导入 SWF 文件时使用的最后一个导入角色的角色 ID
名称 N	字符串	最后导入字符的标识符

启用调试器

EnableDebugger 标签用于启用调试。EnableDebugger 标签中的密码使用 MD5 算法加密，与 Protect 标签相同。
在 SWF 6 中已弃用 EnableDebugger 标签；Flash Player 6 或更高版本会忽略此标签，因为 ActionScript 调试器所需的调试信息格式在 SWF 6 中已更改。在 SWF 6 或更高版本中，请使用 EnableDebugger2 标签代替。

最小和最大文件格式版本为 SWF 5。

字段	Type	评论
标题	记录头	标签类型 = 58
密码	空终止的字符串。（0 为 NULL）	MD5 加密密码

启用调试器 2

EnableDebugger2 标签用于启用调试。密码字段使用 MD5 算法加密，与 Protect 标签相同。

最小文件格式版本为 SWF 6。

字段	Type	评论
标题	记录头	标签类型 = 64
保留	UI16	总是 0
密码	空终止的字符串。（0 为 NULL）	MD5 加密密码

☹

脚本限制

ScriptLimits 标签包含两个字段，可以用来覆盖默认设置的最大递归深度和 ActionScript 超时时间：MaxRecursionDepth 和 ScriptTimeoutSeconds。

MaxRecursionDepth 字段设置 ActionScript 的最大递归限制。在撰写本文时，默认设置为 256。此默认值可以更改为任何大于零（0）的值。

ScriptTimeoutSeconds 字段设置播放器在显示对话框询问是否停止脚本之前应该处理 ActionScript 的最大秒数。ScriptTimeoutSeconds 的默认值因平台而异，介于 15 到 20 秒之间。此默认值可能会更改。

最小文件格式版本为 SWF 7。

字段	Type	评论
标题	记录头	标签类型 = 65
最大递归深度	UI16	最大递归深度
脚本超时秒数	UI16	在脚本卡死对话框显示之前，最大 ActionScript 处理时间

设置 TabIndex

Flash Player 维护显示的交互和文本对象的标签顺序概念。标签顺序既用于实际的标签切换，在 SWF 6 及以后版本中，也用于确定对象暴露给辅助功能（如屏幕阅读器）的顺序。SWF 7 的 SetTabIndex 标签设置对象在标签顺序中的索引。

如果当前没有角色放置在指定的深度，则此标签将被忽略。

您也可以使用 ActionScript 的 tabIndex 属性来设置标签顺序，但这并不能为静态文本对象设置 tab 索引，因为播放器不提供静态文本对象的脚本反射。幸运的是，这对标签顺序来说不是问题，因为静态文本对象永远不会成为实际的标签停止点。然而，这对可访问性顺序来说是个问题，因为静态文本对象会被可访问性辅助工具暴露出来。当生成旨在可访问且包含静态文本对象的 SWF 内容时，SetTabIndex 标签比 tabIndex 属性更有用。

最小文件格式版本是 SWF 7。

字段	Type	评论
标题	记录头	标签类型 = 66

深度	UI16	角色深度
TabIndex	UI16	Tab 顺序值

文件属性

FileAttributes 标签定义了 SWF 文件的特征。此标签对于 SWF 8 及以后的版本是必需的，并且必须是 SWF 文件中的第一个标签。此外，FileAttributes 标签可以可选地包含在所有 SWF 文件版本中。

HasMetadata 标志标识 SWF 文件是否包含 Metadata 标签。Flash Player 不关心这个位字段或相关标签，但对于搜索引擎来说很有用。

UseNetwork 标志表示 Flash Player 是否应授予 SWF 文件本地或网络文件访问权限，如果 SWF 文件是本地加载的。默认行为是仅允许本地 SWF 文件与本地文件交互，而不与网络交互。但是，通过设置 UseNetwork 标志，本地 SWF 可以放弃其本地文件系统访问权限，以换取对网络的访问权限。任何版本的 SWF 都可以使用 UseNetwork 标志来设置在 Flash Player 8 或更高版本中运行的本地加载 SWF 文件的文件访问权限。

最小文件格式版本为 SWF 8。

字段	Type	评论
标题	记录头	标签类型 = 69
保留	UB[1]	必须为 0
使用 DirectBlit（见表格后的说明）	UB[1]	如果为 1，则 SWF 文件在硬件加速可用的情况下使用硬件加速将图形绘制到屏幕上。如果为 0，则 SWF 文件将不会使用硬件加速图形功能。最低文件版本为 10。
使用 GPU（参见备注）以下表格）	UB[1]	如果为 1，SWF 文件在绘图时将使用 GPU 合成功能，当可用的加速时。如果为 0，SWF 文件将不会使用硬件加速的图形设施。最小文件版本为 10。
有元数据	UB[1]	如果为 1，SWF 文件包含元数据标签。如果为 0，SWF 文件不包含元数据标签。
ActionScript3	UB[1]	如果为 1，此 SWF 使用 ActionScript 3.0。如果为 0，此 SWF 使用 ActionScript 1.0 或 2.0。最低文件格式版本为 9。

保留	UB[2]	必须为 0
使用网络	UB[1]	如果为 1，则当本地加载此 SWF 文件时，将赋予网络文件访问权限。如果为 0，则当本地加载此 SWF 文件时，将赋予本地文件访问权限。
保留	UB[24]	必须为 0

当 SWF 文件在独立 Flash Player 中播放时，UseDirectBlit 和 UseGPU 标志才有意义。当 SWF 文件在网页浏览器插件中播放时，UseDirectBlit 等同于在嵌入 HTML 页面的 SWF 标签中指定 wmode 为“direct”，而 UseGPU 等同于 wmode 为“gpu”。

导入资产 2

ImportAssets2 标签取代了 SWF 8 及以后版本的 ImportAssets 标签。ImportAssets2 目前与 ImportAssets 标签的功能相同。

ImportAssets2 标签可以从另一个 SWF 文件导入角色。导入的 SWF 文件通过可找到的 URL 引用导出 SWF 文件。导入的资产就像 SWF 文件内定义的角色一样添加到字典中。

导出 SWF 文件的 URL 可以是绝对路径或相对路径。如果是相对路径，则相对于导入 SWF 文件的位置进行解析。

ImportAssets2 标签必须位于任何依赖导入资源的后续标签之前。

最小文件格式版本为 SWF 8。

字段	Type	评论
标题	记录头	标签类型 = 71
URL	字符串	源 SWF 文件所在的 URL
保留	UI8	必须为 1
保留	UI8	必须为 0
计数	UI16	要导入的资产数量
Tag1	UI16	在导入 SWF 文件时使用的第一个导入角色的角色 ID（无需与导出 SWF 文件中的角色 ID 匹配）
名称 1	字符串	第一个导入角色的标识符（必须匹配标识符

		在导出 SWF 文件时）
TagN	UI16	在导入 SWF 文件时使用的最后一个导入角色的角色 ID
名称 N	字符串	最后导入字符的标识符

符号类

SymbolClass 标签在 SWF 文件中的符号和 ActionScript 3.0 类之间创建关联。它是 ExportAssets 标签的 ActionScript 3.0 等价物。如果字符 ID 为零，则类与 SWF 的主时间轴相关联。这就是指定 SWF 的根类的方式。SymbolClass 标签中列出的类可供其他 SWF 文件创建（参见 StartSound2、DefineEditText（HasFontClass）和 PlaceObject3（PlaceFlagHasClassName 和 PlaceFlagHasImage））。例如，属于同一网站的十个 SWF 文件可以共享一个嵌入的自定义字体，如果其中一个文件嵌入并导出字体类。

字段	Type	评论
标题	记录头	标签类型 = 76
符号数量	UI16	该标签将关联的符号数量。
Tag1	U16	要关联的符号的 16 位字符标签 ID。
名称 1	字符串	要关联此符号的 ActionScript 3.0 类的完全限定名称。该类必须已被 DoABC 标签声明。
...	...	...
TagN	U16	符号 N 的标签 ID。
名称 N	字符串	符号 N 的完全限定类名

元数据

元数据标签是一个可选标签，用于描述 SWF 文件给外部进程。该标签将 XML 元数据嵌入到 SWF 文件中，例如，搜索引擎可以定位此标签，访问 SWF 文件的标题，并在搜索结果中显示该标题。Flash Player 始终忽略元数据标签。

如果 SWF 文件中包含元数据标签，则文件属性标签也必须在 SWF 文件中，并设置其 HasMetadata 标志。相反，如果文件属性标签设置了 HasMetadata 标志，则元数据标签必须在 SWF 文件中。元数据标签只能出现在 SWF 文件中一次。

☹

元数据格式符合 Adobe 的可扩展元数据平台（XMP™）规范，为 RDF 格式。有关 RDF 和 XMP 的更多信息，请参阅以下来源：

- 《RDF 入门》在 www.w3.org/TR/rdf-primer
- 《RDF 规范》在 www.w3.org/TR/1999/REC-rdf-syntax-19990222
- 《XMP 主页》在 www.adobe.com/products/xmp

以下示例展示了在 SWF 文件中表示元数据字符串的多种可接受方式中的两种。第一个示例提供了关于 SWF 文件的基本信息，包括标题和描述：

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'> <rdf:Description
rdf:about='' xmlns:dc='http://purl.org/dc/1.1'> <dc:title>简单标题</dc:title>
<dc:description>简单描述</dc:description> </rdf:Description> </rdf:RDF>
```

在第二个例子中，标题被描述为多种语言：

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'> <rdf:Description
rdf:about='' xmlns:dc='http://purl.org/dc/1.1'> <dc:title> <rdf:Alt> <rdf:li
xml:lang='x-default'>默认标题</rdf:li> <rdf:li xml:lang='en-us'>美国英语标题</rdf:li>
<rdf:li xml:lang='fr-fr'>法语标题</rdf:li> <rdf:li xml:lang='it-it'>意大利语标题
</rdf:li> </rdf:Alt> </dc:title> <dc:description>简单描述</dc:description>
</rdf:Description> </rdf:RDF>
```

元数据字符串存储在 SWF 文件中，所有不必要的空白都被移除。最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 77
元数据	字符串	XML 元数据

定义缩放网格

DefineScalingGrid 标签引入了 9 切片缩放的概念，允许将组件样式缩放应用于精灵或按钮角色。

当 DefineScalingGrid 标签将一个字符与 9 宫格关联时，Flash Player 在概念上将精灵或按钮划分为九个部分，每个部分都覆盖着类似网格的叠加。当字符缩放时，这九个区域分别

每个九个区域独立缩放。为了保持角色的视觉完整性，角落不缩放，而图像的其余部分根据需要放大或缩小。

字段	Type	评论
标题	记录头	标签类型 = 78
字符 ID	UI16	应用缩放网格的精灵或按钮字符的 ID。
分隔符	RECT	9 宫格的中心区域

分割矩形指定了缩放网格的九个区域中的中心部分，Flash Player 从这个矩形推导出 9 宫格。矩形的宽度和高度必须至少为每个 twip（1/20 像素），否则 Flash Player 会忽略 DefineScalingGrid 标签。

当具有 DefineScalingGrid 关联的精灵或按钮进行缩放时，字符的九个区域将根据以下表格进行缩放：

无缩放	水平缩放	无缩放
垂直缩放	水平和垂直缩放	垂直缩放
无缩放	水平缩放	无缩放

9 宫格缩放不会影响指定字符的子对象或任何文本。这些对象正常变换。

具有定义缩放网格关联的精灵或按钮不能旋转或倾斜，这样做将禁用 9 切片行为。然而，此限制不适用于 9 切片对象的父对象或子对象，并且父对象的旋转或倾斜会以正常方式应用于 9 切片对象。

Flash Player 会拉伸角色中的任何填充以适应形状。

9 切片缩放不会影响任何对象的边界或原点。

如果 9 切片角色缩放小于其原始大小，五个缩放区域将被消耗，直到它们变得非常小。一旦达到最小尺寸，Flash Player 将恢复到正常的非 9 切片缩放。

最小文件格式版本为 SWF 8。

定义场景和帧标签数据

DefineSceneAndFrameLabelData 标签包含 MovieClip 的场景和帧标签数据。场景仅支持主时间轴，对于所有其他电影剪辑，仅导出一个场景。

字段	Type	评论
标题	记录头	标签类型 = 86
SceneCount	EncodedU32	场景数量
偏移量 1	EncodedU32	场景 1 的帧偏移
名称 1	字符串	场景 1 的名称
...	...	...
偏移量 N	EncodedU32	场景 N 的帧偏移
名称 N	字符串	场景 N 的名称
帧标签计数	EncodedU32	帧标签数量
帧编号 1	EncodedU32	帧标签 #1 的帧编号（基于零，全局到符号）
帧标签 1	字符串	帧标签 #1 的帧标签字符串
...	...	...
帧编号 N	EncodedU32	帧标签#N 的帧编号（基于零，全局到符号）
帧标签 N	字符串	帧标签#N 的帧标签字符串

第五章：动作

动作是交互式 SWF 文件的重要组成部分。动作允许文件对事件做出反应，例如鼠标移动或鼠标点击。SWF 3 动作模型及之前版本支持简单的动作模型。SWF 4 动作模型支持功能强大的动作模型，包括表达式评估器、变量、条件分支和循环。SWF 5 动作模型增加了 JavaScript 风格的面向对象模型、数据类型和函数。

SWF 3 动作模型

SWF 3 动作模型由 Flash Player 的十一条指令组成：

指令	See	描述
Play	动作播放	从当前帧开始播放
Stop	动作停止	在当前帧停止播放
下一个帧	ActionNextFrame	跳转到下一帧
上一帧	上一个动作帧	跳转到上一帧
跳转帧	跳转到指定帧	跳转到指定帧
跳转到标签	跳转到标签动作	前往具有指定标签的帧
等待帧	动作等待帧	等待指定的帧
获取 URL	执行获取 URL 操作	获取指定的 URL
停止声音	停止所有声音	停止播放的所有声音
切换质量	切换质量动作	在高画质和低画质之间切换显示
设置目标	动作设置目标	将后续动作的上下文更改为命名对象

一个动作（或动作列表）可以通过按钮状态转换或 SWF 3 动作触发。该动作不会立即执行，而是被添加到待处理动作列表中。该列表在 ShowFrame 标签处执行，或者在按钮状态改变后执行。一个动作可以触发其他动作，在这种情况下，该动作将被添加到待处理动作列表中。动作将一直处理，直到动作列表为空。

默认情况下，时间轴动作，如停止（见 ActionStop）、播放（见 ActionPlay）和转到帧（见 ActionGotoFrame），适用于包含它们的文件。然而，SetTarget 动作（见 ActionSetTarget），

该功能在 Adobe Flash 用户界面中称为“告诉目标”，可以用来向另一个文件或精灵发送动作命令（见“定义精灵”）。

SWF 3 动作

本节中的动作在 SWF 3 中可用。

执行动作

DoAction 指示 Flash Player 在当前帧完成时执行一系列动作。这些动作在遇到 ShowFrame 标签时执行，而不管 DoAction 标签出现在帧的哪个位置。
从 SWF 9 开始，如果 FileAttributes 标签的 ActionScript3 字段为 1，则 DoAction 标签的内容将被忽略。

字段	Type	评论
标题	记录头	标签类型 = 12
动作	ACTIONRECORD [零个或多个]	要执行的动作列表（见下表，ActionRecord）
动作结束标志	UI8 = 0	总是设置为 0

ACTIONRECORD

ACTIONRECORD 由 ACTIONRECORDHEADER 组成，后跟可能的数据有效载荷。ACTIONRECORDHEADER 使用 ActionCode 描述动作。如果动作还携带数据，ActionCode 的最高位将被设置，这表示 ActionCode 后面跟着一个 16 位长度和数据有效载荷。请注意，许多动作没有数据有效载荷，只由一个字节值组成。

ACTIONRECORDHEADER 具有以下布局：

字段	Type	评论
ActionCode	UI8	动作码
长度	如果代码 >= 0x80，UI16	ACTIONRECORDHEADER 中的字节数，不包括 ActionCode 和 Length 字段。

ActionGotoFrame

ActionGotoFrame 指示 Flash Player 跳转到当前文件中的指定帧。

字段	Type	评论
跳转到指定帧	ACTIONRECORDHEADER	ActionCode = 0x81；长度始终为 2
帧	UI16	帧索引

ActionGetURL

ActionGetURL 指示 Flash Player 获取 UrlString 指定的 URL。URL 可以是任何类型，包括 HTML 文件、图片或另一个 SWF 文件。如果文件在浏览器中播放，URL 将在 TargetString 指定的帧中显示。" _level0" 和 "_level1" 特殊目标名称用于分别将另一个 SWF 文件加载到 0 级和 1 级。

字段	Type	评论
执行获取 URL 操作	ACTIONRECORDHEADER	ActionCode = 0x83
网址字符串	字符串	目标网址字符串
目标字符串	字符串	目标字符串

ActionNextFrame

ActionNextFrame 指示 Flash Player 跳转到当前文件中的下一帧。

字段	Type	评论
ActionNextFrame	ACTIONRECORDHEADER	ActionCode = 0x04

ActionPreviousFrame

ActionPreviousFrame 指示 Flash Player 跳转到当前文件的上一帧。

字段	Type	评论
ActionPrevFrame	ACTIONRECORDHEADER	ActionCode = 0x05

ActionPlay

ActionPlay 指示 Flash Player 从当前帧开始播放。

字段	Type	评论
动作播放	ACTIONRECORDHEADER	操作代码 = 0x06

☹

动作停止

ActionStop 指示 Flash Player 在当前帧停止播放文件。

字段	Type	评论
动作停止	ACTIONRECORDHEADER	ActionCode = 0x07

ActionToggleQuality

ActionToggleQuality 在高画质和低画质之间切换显示。

字段	Type	评论
动作切换质量	ACTIONRECORDHEADER	动作代码 = 0x08

动作停止声音

动作停止声音指示 Flash Player 停止播放所有声音。

字段	Type	评论
停止所有声音	ACTIONRECORDHEADER	ActionCode = 0x09

ActionWaitForFrame

ActionWaitForFrame 指示 Flash Player 等待直到指定的帧；否则跳过指定的动作数量。

字段	Type	评论
动作等待帧	ACTIONRECORDHEADER	ActionCode = 0x8A; 长度始终为 3
帧	UI16	等待帧
跳过计数	UI8	如果帧未加载，则跳过的动作数量

设置动作目标

ActionSetTarget 指示 Flash Player 改变后续动作的上下文，使它们应用于命名对象（目标名称）而不是当前文件。

例如，可以使用 SetTarget 动作来控制精灵对象的 Timeline。以下动作序列将名为 "spinner" 的精灵发送到其 Timeline 的第一帧：

1. 设置目标 "spinner"

2. 进入帧 0
3. SetTarget " "（空字符串）
4. 动作结束。（动作代码 = 0）

所有跟随 SetTarget “spinner” 的动作都应用于 spinner 对象，直到 SetTarget “”，这会将动作上下文恢复到当前文件。关于目标名称的完整讨论，请参阅 DefineSprite。

字段	Type	评论
动作设置目标	ACTIONRECORDHEADER	ActionCode = 0x8B
目标名称	字符串	行动目标

跳转到标签

跳转到标签指令 Flash Player 跳转到与指定标签关联的帧。您可以使用 FrameLabel 标签将标签附加到帧上。

字段	Type	评论
跳转到标签动作	ACTIONRECORDHEADER	操作代码 = 0x8C
标签	字符串	帧标签

SWF 4 动作模型

SWF 4 文件格式支持一个功能强大的动作模型，包括表达式评估器、变量、条件分支和循环。

Flash Player 4 集成了栈式机器，用于解释和执行 SWF 4 动作。SWF 4 的关键动作是 ActionPush。此动作用于将一个或多个参数推入栈中。与 SWF 3 动作不同，SWF 4 动作没有参数嵌入在标签中，而是将参数推入栈中，并从栈中弹出结果。

表达式评估器也是基于栈的。算术运算符包括 ActionAdd、ActionSubtract、ActionMultiply 和 ActionDivide。Flash 创作工具将表达式转换为一系列栈操作。例如，表达式 1+x*3 表示为以下动作序列：

```
ActionPush "x"
获取变量
压入 "3"
乘法
压入 "1"
动作添加
```

此表达式的结果位于栈上。

所有堆栈上的值，包括数值，都存储为字符串。在上面的示例中，数值 3 和 1 作为字符串"3"和"1"压入堆栈。

程序计数器

Flash Player 当前执行的点称为程序计数器（PC）。PC 的值定义为当前正在执行的动作之后动作的地址。控制流动作，如 ActionJump 会改变 PC 的值。这些动作类似于汇编器中的分支指令，或其他语言中的 goto 指令。例如，ActionJump 告诉 Flash Player 跳转到动作序列中的新位置。新的 PC 值被指定为当前 PC 的偏移量。正负偏移量都可能发生，因此 Flash Player 可以在动作序列中向前或向后跳转。

SWF 4 动作

SWF 4 中可用的以下动作：

动作类型	动作名称
算术运算符	ActionAdd ActionDivide ActionMultiply ActionSubtract
数值比较	动作等于 动作小于
逻辑运算符	动作与 动作非 动作或
字符串操作	动作字符串添加 动作字符串比较 动作字符串提取 动作字符串长度 动作 MB 字符串提取 动作 MB 字符串长度 动作字符串小于
栈操作	ActionPop ActionPush
类型转换	ActionAsciiToChar ActionCharToAscii ActionToInteger ActionMBAsciiToChar ActionMBCharToAscii
控制流	ActionCall ActionIf ActionJump
变量	动作获取变量 动作设置变量
电影控制	ActionGetURL2 ActionGetProperty ActionGotoFrame2 ActionRemoveSprite ActionSetProperty ActionSetTarget2 ActionStartDrag ActionWaitForFrame2 ActionCloneSprite ActionEndDrag
工具程序	获取时间 动作 随机数 动作 跟踪

栈操作

本节列出了栈操作。

☹

动作推送

ActionPush 将一个或多个值推送到栈中。

字段	Type	评论
ActionPush	ACTIONRECORDHEADER	ActionCode = 0x96
Type	UI8	0 = 字符串字面量 1 = 浮点字面量 SWF 5 及以后版本中可用的类型如下： 2 = null 3 = undefined 4 = 注册 5 = 布尔值 6 = 双精度浮点数 7 = 整数 8 = 常量 8 9 = 常数 16
字符串	如果类型 = 0，字符串	空终止的字符字符串
浮点数	如果类型 = 1，浮点数	32 位 IEEE 单精度小端浮点值
寄存器编号	如果类型 = 4，UI8	注册号
布尔值	如果类型 = 5，UI8	布尔值（值）
双重	如果类型 = 6，双精度	64 位 IEEE 双精度小端双精度值
整数	如果类型 = 7，UI32	32 位小端整数
常量 8	如果类型 = 8，UI8	常量池索引（对于索引小于 256 的情况）（参见 ActionConstantPool）
Constant16	如果类型=9，UI16	常量池索引（对于索引大于等于 256 的情况）（参见 ActionConstantPool）

ActionPush 将一个或多个值压入堆栈。类型字段指定要压入的值的类型。

☹

如果类型=1，要压入的值指定为 32 位 IEEE 单精度小端浮点值。PropertyIds 作为 FLOAT 压入。ActionGetProperty 和 ActionSetProperty 使用 PropertyIds 来访问命名对象的属性。

如果类型=4，则要推送的值是寄存器编号。Flash Player 支持最多 4 个寄存器。使用 ActionDefineFunction2，最多可以使用 256 个寄存器。

在 ActionPush 的第一个字段中，ACTIONRECORD 中的长度定义了 ACTIONRECORD 本身之后跟随的总类型和值字节数。根据 ACTIONRECORD 中指定的字节数，可能跟随多个类型和值字段。

ActionPop

ActionPop 从堆栈中弹出一个值并丢弃它。

字段	Type	评论
ActionPop	ACTIONRECORDHEADER	ActionCode = 0x17

ActionPop 从堆栈中弹出一个值并丢弃该值。

算术运算符

以下章节描述了算术运算符。

动作添加

ActionAdd 将两个数字相加并将结果推回栈中。

字段	Type	评论
ActionAdd	ACTIONRECORDHEADER	ActionCode = 0x0A

ActionAdd 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 将数字 A 和 B 相加。
5. 将结果 A+B 压入栈中。

☹

动作减去

ActionSubtract 从两个数字中减去并返回结果到栈中。

字段	Type	评论
ActionSubtract	ACTIONRECORDHEADER	ActionCode = 0x0B

ActionSubtract 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 从 B 中减去 A。
5. 将结果 B-A 压入栈中。

乘法

ActionMultiply 将两个数字相乘并将结果推回栈中。

字段	Type	评论
ActionMultiply 操作将两个数字相乘。	ACTIONRECORDHEADER	ActionCode = 0x0C

ActionMultiply 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 将 A 乘以 B。
5. 将结果 A*B 压入栈中。

☹

动作分割

ActionDivide 将两个数字相除并将结果推回栈中。

字段	Type	评论
ActionDivide	ACTIONRECORDHEADER	ActionCode = 0x0D

ActionDivide 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 将 B 除以 A。
5. 将结果 B/A 压入栈中。
6. 如果 A 为零，SWF 5 及以后版本将 NaN、Infinity 或-Infinity 压入栈中。在 SWF 4 中，结果为字符串 #ERROR#。

数值比较

ActionEquals

ActionEquals 测试两个数字是否相等。

字段	Type	评论
ActionEquals	ACTIONRECORDHEADER	ActionCode = 0x0E

ActionEquals 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 比较数字是否相等。
5. 如果数字相等，则将 true 压入栈中，适用于 SWF 5 及以后版本。
6. 对于 SWF 4，将 1 压入栈中。

☹

7. 否则，SWF 5 及以后版本将推送 false 到堆栈。（对于 SWF 4，将推送 0 到堆栈。）

无动作

ActionLess 测试一个数字是否小于另一个数字

字段	Type	评论
无动作	ACTIONRECORDHEADER	ActionCode = 0x0F

ActionLess 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 如果 B < A，则对于 SWF 5 及以后版本，将 true 压入栈中（对于 SWF 4，将 1 压入栈中）；否则，对于 SWF 5 及以后版本，将 false 压入栈中（对于 SWF 4，将 0 压入栈中）。

逻辑运算符

ActionAnd

逻辑与操作对两个数字进行逻辑与运算。

字段	Type	评论
逻辑与操作	ACTIONRECORDHEADER	操作码 = 0x10

ActionAdd 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 如果两个数字都不为零，则对于 SWF 5 及以后版本将 true 压入堆栈（对于 SWF 4 将 1 压入堆栈）；否则，对于 SWF 5 及以后版本将 false 压入堆栈（对于 SWF 4 将 0 压入堆栈）。

ActionOr

ActionOr 执行两个数字的逻辑或运算。

☹

字段	Type	评论
动作或	ACTIONRECORDHEADER	动作码 = 0x11

动作或执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 转换为浮点数；非数值评估为 0。
4. 如果任一数字不为零，则对于 SWF 5 及以后版本，将 true 压入堆栈（对于 SWF 4，压入 1）；否则，对于 SWF 5 及以后版本，将 false 压入堆栈（对于 SWF 4，压入 0）。

ActionNot

ActionNot 执行数字的逻辑非操作。

注意：在 SWF 5 文件中，ActionNot 操作符将它的参数转换为布尔值，并推送一个布尔类型的结果。在 SWF 4 文件中，参数和结果都是数字。

字段	Type	评论
ActionNot	ACTIONRECORDHEADER	操作码 = 0x12
结果	布尔值	

ActionNot 执行以下操作：

1. 从堆栈中弹出值。
2. 将值转换为浮点数；非数值值评估为 0。
3. 如果值为零，对于 SWF 5 及以后版本，将 true 压入堆栈（对于 SWF 4，将 1 压入堆栈）。
4. 如果值不为零，对于 SWF 5 及以后版本，将 false 压入堆栈（对于 SWF 4，将 0 压入堆栈）。

☹

字符串操作

ActionStringEquals

ActionStringEquals 测试两个字符串是否相等。

字段	Type	评论
ActionStringEquals	ACTIONRECORDHEADER	ActionCode = 0x13

ActionStringEquals 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将 A 和 B 作为字符串进行比较。（比较区分大小写）
4. 如果字符串相等，则对于 SWF 5 及以后的版本，将 true 压入堆栈（SWF 4 压入 1）。
5. 否则，对于 SWF 5 及以后的版本，将 false 压入堆栈（SWF 4 压入 0）。

动作字符串长度

动作字符串长度计算字符串的长度。

字段	Type	评论
动作字符串长度	ACTIONRECORDHEADER	动作代码 = 0x14

ActionStringLength 执行以下操作：

1. 从栈中弹出一个字符串。
2. 计算字符串的长度并将其推入栈中。

ActionStringAdd

ActionStringAdd 连接两个字符串。

字段	Type	评论
ActionStringAdd	ACTIONRECORDHEADER	ActionCode = 0x21

☹

ActionStringAdd 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 将字符串“BA”连接推入栈中。

ActionStringExtract

ActionStringExtract 从字符串中提取子字符串。

字段	Type	评论
ActionStringExtract	ACTIONRECORDHEADER	ActionCode = 0x15

ActionStringExtract 执行以下操作：

1. 从堆栈中弹出数字计数。
2. 从堆栈中弹出数字索引。
3. 从堆栈中弹出字符串字符串。
4. 将字符串从索引字符开始，长度为 count 个字符的子字符串推入堆栈。
5. 如果索引或 count 不评估为整数，则结果为空字符串。

ActionStringLess

测试字符串是否小于另一个字符串

字段	Type	评论
ActionStringLess	ACTIONRECORDHEADER	ActionCode = 0x29

ActionStringLess 执行以下操作：

1. 从栈中弹出值 A。
2. 从栈中弹出值 B。
3. 如果 B 小于 A 使用字节对字节比较，SWF 5 及以后版本将 true 压入堆栈（SWF 4 版本将 1 压入堆栈）；否则，SWF 5 及以后版本将 false 压入堆栈（SWF 4 版本将 0 压入堆栈）。

☹

动作 MB 字符串长度

ActionMBStringLength 计算字符串长度，并支持多字节。

字段	Type	评论
ActionMBStringLength	ACTIONRECORDHEADER	ActionCode = 0x31

ActionMBStringLength 执行以下操作：

1. 从栈中弹出一个字符串。
2. 计算字符串的字符长度并将其推入栈中。

这是 ActionStringLength 的多字节版本。在支持双字节的系统上，双字节字符被视为单个字符。

ActionMBStringExtract

ActionMBStringExtract 从字符串中提取子字符串，并且是支持多字节的。

字段	Type	评论
ActionMBStringExtract	ACTIONRECORDHEADER	ActionCode = 0x35

它执行以下操作：

1. 从堆栈中弹出数字计数。
2. 从栈中弹出数字索引。
3. 从栈中弹出字符串。
4. 将从索引字符开始的字符串子串及其长度推入栈中。

注意：如果索引或计数未评估为整数，则结果为空字符串。

这是一个多字节感知版本的 ActionStringExtract。索引和计数被视为字符索引，将双字节字符计为单个字符。

类型转换

ActionToInteger

ActionToInteger 将一个值转换为整数。

☹

字段	Type	评论
ActionToInteger	ACTIONRECORDHEADER	ActionCode = 0x18

ActionToInteger 执行以下操作：

1. 从堆栈中弹出值。
2. 将值转换为数字。
3. 抛弃小数点后的所有数字，得到一个整数。
4. 将得到的整数推入栈中。

ActionCharToAscii

ActionCharToAscii 将字符码转换为 ASCII。

字段	Type	评论
ActionCharToAscii	ACTIONRECORDHEADER	ActionCode = 0x32

ActionCharToAscii 执行以下操作：

1. 从堆栈中弹出值。
2. 将值的第一个字符转换为数值 ASCII 字符码。
3. 将结果字符代码推送到堆栈。

ActionAsciiToChar

ActionAsciiToChar 将值转换为 ASCII 字符代码。

字段	Type	评论
ActionAsciiToChar	ACTIONRECORDHEADER	ActionCode = 0x33

ActionAsciiToChar 执行以下操作：

1. 从堆栈中弹出一个值。
2. 将值从数字转换为相应的 ASCII 字符。

☹

3. 将生成的字符推入堆栈。

ActionMBCharToAscii

ActionMBCharToAscii 将字符码转换为 ASCII，并支持多字节。

字段	Type	评论
ActionMBCharToAscii	ACTIONRECORDHEADER	ActionCode = 0x36

ActionMBCharToAscii 执行以下操作：

1. 从堆栈中弹出值。
2. 将值的第一个字符转换为数值字符代码。
   如果值的第一个字符是双字节字符，则构造一个 16 位值，其中第一个字节为高字节，第二个字节为低字节。
3. 将结果字符代码推送到堆栈。

ActionMBAsciiToChar

ActionMBAsciiToChar 将 ASCII 转换为字符码，并支持多字节。

字段	Type	评论
ActionMBAsciiToChar	ACTIONRECORDHEADER	ActionCode = 0x37

ActionMBAsciiToChar 执行以下操作：

1. 从堆栈中弹出值。
2. 将数值转换为相应的字符。如果字符是 16 位值（>= 256），则构建一个双字节字符，第一个字节包含高字节，第二个字节包含低字节。
3. 将生成的字符压入栈中。

☹

控制流程

ActionJump

ActionJump 创建一个无条件分支。

字段	Type	评论
ActionJump	ACTIONRECORDHEADER	ActionCode = 0x99
分支偏移量	SI16	偏移

ActionJump 将 BranchOffset 字节添加到执行流中的指令指针。偏移量是一个有符号量，允许从 -32,768 字节到 32,767 字节的分支。偏移量为 0 指向 ActionJump 操作之后的操作。

ActionIf

如果创建一个条件测试和分支。

字段	Type	评论
ActionIf	ACTIONRECORDHEADER	ActionCode = 0x9D
分支偏移量	SI16	偏移量

ActionIf 执行以下操作：

1. 从堆栈中弹出条件、一个数字。
2. 将条件转换为布尔值。
3. 测试条件是否为真。如果条件为真，则在执行流中的指令指针上添加 BranchOffset 字节数。

注意：播放 SWF 4 文件时，条件不会转换为布尔值，而是与 0 进行比较，而不是与真进行比较。

偏移量是一个有符号的数量，允许从-32768 字节到 32767 字节的分支。偏移量为 0 指向 ActionIf 操作之后的操作。

动作调用

ActionCall 调用一个子程序。

字段	Type	评论
ActionCall	ACTIONRECORDHEADER	ActionCode = 0x9E

ActionCall 执行以下操作：

1. 从堆栈中弹出一个值。此值应为一个与帧标签匹配的字符串，或表示帧编号的数字。该值可以由一个目标字符串前缀，以标识包含被调用帧的电影剪辑。

2. 如果成功找到帧，则执行目标帧中的操作。在目标帧中的操作执行完毕后，执行将恢复到 ActionCall 指令之后的指令。

3. 如果找不到帧，则不执行任何操作。

变量

获取变量

ActionGetVariable 获取变量的值。

字段	Type	评论
ActionGetVariable	ACTIONRECORDHEADER	ActionCode = 0x1C

ActionGetVariable 执行以下操作：

1. 从栈中弹出一个名称，该名称是变量的名称。

2. 将变量的值压入栈中。

另一个执行上下文中的变量可以通过在变量名前加上目标路径和冒号来引用。例如：/A/B:FOO 引用目标路径为 /A/B 的电影剪辑中的变量 FOO。

动作设置变量

ActionSetVariable 设置变量。

字段	Type	评论
ActionSetVariable	ACTIONRECORDHEADER	ActionCode = 0x1D

ActionSetVariable 执行以下操作：

1. 从堆栈中弹出值。
2. 从堆栈中弹出名称，该名称是一个字符串，用于指定要设置的变量。
3. 将变量名称设置为值，并放置在当前执行上下文中。

另一个执行上下文中的变量可以通过在变量名称前加上目标路径和冒号来引用。例如：/A/B:FOO 引用了具有目标路径 /A/B 的 FOO 变量。

电影控制

ActionGetURL2

ActionGetURL2 获取一个 URL，并且是栈基础的。

字段	Type	评论
ActionGetURL2	ACTIONRECORDHEADER	ActionCode = 0x9A 长度始终为 1
发送变量方法	UB[2]	0 = 无; 1 = GET 2 = POST
保留	UB[4]	总是 0
LoadTargetFlag	UB[1]	0 = 目标是浏览器窗口 1 = 目标是精灵的路径
LoadVariablesFlag	UB[1]	0 = 无变量可加载 1 = 加载变量

ActionGetURL2 执行以下操作：

1. 从堆栈中弹出目标

- A LoadTargetFlag 值 0 表示目标是一个窗口。目标可以是空字符串，表示当前窗口。

- 1 表示目标是一个精灵的路径。目标路径可以是斜杠或点语法。

2. 从堆栈中弹出一个 URL；该 URL 指定要检索的 URL。
3. SendVarsMethod 指定用于 HTTP 请求的方法。
   - SendVarsMethod 值为 0 表示这不是表单请求，因此电影剪辑的变量不应编码并提交。

   - SendVarsMethod 的值为 1 表示 HTTP GET 请求。

   - SendVarsMethod 的值为 2 表示 HTTP POST 请求。

4. 如果 SendVarsMethod 的值为 1（GET）或 2（POST），则当前电影剪辑中的变量将通过标准 x-www-form-urlencoded 编码提交到 URL，并使用 method 指定的 HTTP 请求方法。

如果设置了 LoadVariablesFlag，则服务器应返回 MIME 类型为 application/x-www-formurlencoded，格式为 var1=value1&var2=value2&...&varx=valuex 的响应体。此响应用于填充 ActionScript 变量，而不是显示文档。填充的变量可以是时间轴上的（如果 LoadTargetFlag 为 0），也可以是指定的精灵（如果 LoadTargetFlag 为 1）。

如果在未指定 LoadVariablesFlag 的情况下指定 LoadTargetFlag，则服务器应返回 MIME 类型为 application/x-shockwave-flash 的响应，其主体由一个 SWF 文件组成。此响应用于将子文件加载到指定的精灵中，而不是显示 HTML 文档。

ActionGotoFrame2

ActionGotoFrame2 跳转到帧，并且是堆栈基础的。

字段	Type	评论
ActionGotoFrame2	ACTIONRECORDHEADER	操作代码 = 0x9F
保留	UB[6]	总是 0
场景偏差标志	UB[1]	场景偏差标志
播放标志	UB[1]	0 = 跳转到帧并停止 1 = 跳转到帧并播放
场景偏差	如果场景偏差标志 = 1，则 UI16	要添加到帧的数字由堆栈确定

		参数

ActionGotoFrame2 执行以下操作：

1. 从栈中弹出一个帧。

   - 如果该帧是一个数字，n，则要显示的电影的下一帧是当前电影剪辑中的第 n 帧。

   - 如果该帧是一个字符串，则 frame 被视为帧标签。如果指定的标签在当前电影剪辑中存在，则带标签的帧将成为当前帧。如果不存在，则该操作将被忽略。

2. 可以在帧或数字前加上目标路径，例如，/MovieClip:3 或 /MovieClip:FrameLabel。

3. 如果设置了播放标志，则动作跳转到指定的帧并开始播放包含的影片剪辑。
   否则，动作跳转到指定的帧并停止。

ActionSetTarget2

ActionSetTarget2 设置当前上下文，并且是堆栈基础的。

字段	Type	评论
ActionSetTarget2	ACTIONRECORDHEADER	ActionCode = 0x20

ActionSetTarget2 从堆栈中弹出目标并将其设置为当前活动上下文。

此操作的行为与 ActionSetTarget 完全相同，但基于堆栈，以便目标路径可以是表达式评估的结果。

获取属性操作

ActionGetProperty 获取文件属性。

字段	Type	评论
ActionGetProperty	ACTIONRECORDHEADER	ActionCode = 0x22

ActionGetProperty 执行以下操作：

1. 从堆栈中弹出索引。

2. 从堆栈中弹出目标

3. 从目标路径为 target 的电影剪辑中获取枚举为索引的属性值并将其推送到堆栈中。

下表列出了属性索引值。_quality、_xmouse 和_ymouse 属性在 SWF 5 及以后版本中可用。

属性	值
_X	0
_Y	1
_xscale	2
_yscale	3
_currentframe	4
_totalframes	5
_alpha	6
_visible	7
_width	8
_height	9
旋转	10
目标	11
已加载帧数	12
名称	13
_droptarget	14
_url	15
_highquality	16
_focusrect	17
_soundbuftime	18

_质量	19
_xmouse	20
_ymouse	21

动作设置属性

动作设置属性设置文件属性。

字段	Type	评论
ActionSetProperty	ACTIONRECORDHEADER	ActionCode = 0x23

ActionSetProperty 执行以下操作：

1. 从堆栈中弹出值。
2. 从堆栈中弹出一个索引。
3. 从堆栈中弹出目标。
4. 将枚举为索引的属性设置为具有目标路径 target 的电影剪辑中的值 value。

ActionCloneSprite

ActionCloneSprite 克隆一个精灵。

字段	Type	评论
ActionCloneSprite	ACTIONRECORDHEADER	ActionCode = 0x24

ActionCloneSprite 执行以下操作：

1. 从堆栈中弹出深度。
2. 从堆栈中弹出目标。
3. 从堆栈中弹出源。
4. 复制电影剪辑源，给新实例命名为目标，在 Z 轴深度。

动作移除精灵

在 z 轴深度中，ActionRemoveSprite 移除一个克隆精灵。

字段	Type	评论
ActionRemoveSprite	ACTIONRECORDHEADER	ActionCode = 0x25

ActionRemoveSprite 执行以下操作：

1. 从堆栈中弹出目标。
2. 移除由目标路径目标标识的克隆电影剪辑。

ActionStartDrag

ActionStartDrag 开始拖动电影剪辑。

字段	Type	评论
ActionStartDrag	ACTIONRECORDHEADER	ActionCode = 0x27

ActionStartDrag 执行以下操作：

1. 从堆栈中弹出一个目标；目标标识要拖动的电影剪辑。
2. 从堆栈中弹出 lockcenter。如果 lockcenter 评估结果不为零，则拖动电影剪辑的中心锁定到鼠标位置。否则，当开始拖动时，电影剪辑相对于鼠标位置移动。
3. 从栈中弹出 Pops
4. 如果约束计算结果不为零：
    - 从栈中弹出 y2
    - 从栈中弹出 x2
    - 从堆栈中弹出 y1。
    - 从堆栈中弹出 x1。

动作结束拖动

正在进行的拖拽操作结束

字段	Type	评论
ActionEndDrag	ACTIONRECORDHEADER	操作代码 = 0x28

等待帧 2 动作

ActionWaitForFrame2 等待帧加载完成，是堆栈基础的。

字段	Type	评论
ActionWaitForFrame2	ACTIONRECORDHEADER	ActionCode = 0x8D；长度始终为 1
跳过计数	UI8	跳过的动作数量

ActionWaitForFrame2 执行以下操作：

1. 从栈中弹出一个帧。
2. 如果帧已加载，则跳过当前操作后的下一个 n 个操作，其中 n 由 SkipCount 指示。

该帧的评估方式与 ActionGotoFrame2 相同。

工具

ActionTrace

ActionTrace 发送调试输出字符串。

字段	Type	评论
ActionTrace	ACTIONRECORDHEADER	ActionCode = 0x26

ActionTrace 执行以下操作：

1. 从堆栈中弹出值。
2. 在 Adobe Flash 编辑器的测试电影模式下，如果未将调试级别设置为 None，ActionTrace 将向输出窗口追加一个值。

在 Adobe Flash Player 中，没有任何操作。

动作获取时间

ActionGetTime 报告自 Adobe Flash Player 启动以来的毫秒数。

字段	Type	评论
ActionGetTime	ACTIONRECORDHEADER	ActionCode = 0x34

ActionGetTime 执行以下操作：

1. 计算自 Flash Player 启动以来的毫秒数。
2. 将数字推入堆栈。

ActionRandomNumber

ActionRandomNumber 计算一个随机数。

字段	Type	评论
随机数操作	ACTIONRECORDHEADER	操作码 = 0x30

ActionRandomNumber 执行以下操作：

1. 从栈中弹出最大值。
2. 计算一个 0 到（最大值-1）范围内的随机整数。
3. 将随机数推入栈中。

SWF 5 动作模型

SWF 5 与 SWF 4 类似。新动作大大扩展了 ActionScript 的功能。还有新的类型转换、数学和栈操作符动作。

SWF 5 动作

以下是对 SWF 5 动作的概述：

动作类型	动作名称
脚本对象动作	ActionCallFunction 调用方法 常量池

	动作定义函数 动作定义局部变量 动作定义局部变量 2 动作删除动作删除 2 动作枚举动作等于 2 动作获取成员 动作初始化数组 动作初始化对象 动作新建方法 动作新建对象 动作设置成员 动作目标路径 动作使用
类型操作	动作转数字 动作转字符串 动作类型
数学操作	动作加 2 无动作 2 动作模数
堆栈操作符动作	位与操作 位左移操作 ActionBitOr ActionBitRShift ActionBitURShift ActionBitXor ActionDecrement ActionIncrement ActionPush（增强）ActionPushDuplicate ActionReturn ActionStackSwap ActionStoreRegister

脚本对象操作

ActionCallFunction

ActionCallFunction 执行一个函数。该函数可以是 ActionScript 内置函数（例如 parseInt），用户定义的 ActionScript 函数或本地函数。有关更多信息，请参阅 ActionNewObject。

字段	Type	评论
调用函数	ACTIONRECORDHEADER	操作码 = 0x3D

ActionCallFunction 执行以下操作：

1. 从堆栈中弹出函数名（字符串）。
2. 从堆栈中弹出 numArgs (int)。
3. 弹出堆栈上的参数。
4. 调用函数，并将参数传递给它。
5. 将函数调用的返回值推送到堆栈。

   如果没有适当的返回值（即函数中没有返回语句），编译器将生成一个推送“未定义”消息，并将其推入栈中。未定义的返回值应从栈中弹出。

对于所有调用操作（ActionCallMethod、ActionNewMethod、ActionNewObject 和 ActionCallFunction）以及初始化操作（ActionInitObject 和 ActionInitArray），函数的参数以相反的顺序推入栈中，最右边的参数先，最左边的参数后。参数随后按顺序弹出（从第一个到最后一个）。

调用方法

ActionCallMethod 将方法（函数）调用推入栈中，类似于 ActionNewMethod。

字段	Type	评论
ActionCallMethod	ACTIONRECORDHEADER	ActionCode = 0x52

如果存在该命名方法，ActionCallMethod 执行以下操作：

1. 从栈中弹出方法名称。如果方法名称为空或未定义，则对象被视为应调用的函数对象，而不是方法容器的对象。例如，如果使用对象 obj 和空方法名称调用 CallMethod，则与使用以下语法等效：

   ```
   obj();
   ```

   如果一个方法的名字是 foo，则相当于：

   ```
   obj.foo();
   ```

2. 弹出堆栈中的 ScriptObject 对象。
3. 从堆栈中弹出参数数量，args。
4. 弹出堆栈上的参数。
5. 执行带有指定参数的方法调用。
6. 将方法或函数的返回值推入堆栈。
   如果没有适当的返回值（函数没有返回语句），编译器将生成一个 push undefined 并将其推入堆栈。
   未定义的返回值应从堆栈中弹出。

对于所有调用操作（ActionCallMethod、ActionNewMethod、ActionNewObject 和 ActionCallFunction）以及初始化操作（ActionInitObject 和 ActionInitArray），函数的参数以相反的顺序推入栈中，最右边的参数先，最左边的参数后。参数随后按顺序弹出（从第一个到最后一个）。

常量池

ActionConstantPool 创建一个新的常量池，如果已存在则替换旧常量池。

字段	Type	评论
ActionConstantPool	ACTIONRECORDHEADER	ActionCode = 0x88
计数	UI16	后续常量的数量
常量池	字符串[数量]	字符串常量

动作定义函数

注意：自 SWF 7 及以后，ActionDefineFunction 很少使用；它已被 ActionDefineFunction2 取代。

ActionDefineFunction 定义了一个具有给定名称和体大小的函数。

字段	Type	评论
ActionDefineFunction	ACTIONRECORDHEADER	ActionCode = 0x9B
函数名	字符串	函数名称，如果匿名则为空
参数数量	UI16	参数数量（# of parameters）
参数 1	字符串	参数名称 1
参数 2	字符串	参数名称 2

...		
参数 N	字符串	参数名称 N
代码大小	UI16	代码后续的字节数

ActionDefineFunction 按顺序解析（函数名，参数数量，[参数 1，参数 2，...，参数 N]）然后代码大小。

ActionDefineFunction 执行以下操作：

1. 从动作标签中解析函数名称（name）。

2. 跳过标签中的参数。

3. 从标签中解析代码大小。在 DefineFunction 标签之后，接下来的 codeSize 字节动作数据被认为是函数的主体。

4. 获取函数的代码。

ActionDefineFunction 可以用以下方式使用：

使用方法 1：将一个匿名函数压入栈中，该函数不会持久存在。这个函数是一个函数字面量，它在表达式中声明而不是在语句中。匿名函数可以用来定义函数、返回其值并将它赋值给变量，如下面的 ActionScript 所示：

```
area = (function () {return Math.PI * radius *radius;})(5);
```

使用 2 设置一个带有给定 FunctionName 和给定函数定义的变量。这是更传统的函数定义方式。例如，在 ActionScript 中：

```
function Circle(radius) { this.radius = radius;
this.area = Math.PI * radius * radius; }
```

ActionDefineLocal

ActionDefineLocal 定义一个局部变量并设置其值。如果变量已存在，则将其值设置为新的指定值。

字段	Type	评论
ActionDefineLocal	ACTIONRECORDHEADER	ActionCode = 0x3C

ActionDefineLocal 执行以下操作：

1. 从堆栈中弹出值。
2. 从堆栈中弹出一个名称。

ActionDefineLocal2

ActionDefineLocal2 定义一个局部变量而不设置其值。如果变量已存在，则不执行任何操作。局部变量的初始值是未定义的。

字段	Type	评论
ActionDefineLocal2	ACTIONRECORDHEADER	ActionCode = 0x41

ActionDefineLocal2 从堆栈中弹出名称。

ActionDelete

ActionDelete 从 ScriptObject 中删除一个命名属性。

字段	Type	评论
ActionDelete	ACTIONRECORDHEADER	ActionCode = 0x3A

ActionDelete 执行以下操作：

1. 从栈中弹出要删除的属性名称。
2. 弹出对象以删除属性。

ActionDelete2

ActionDelete2 删除一个命名的属性。Flash Player 首先在当前作用域中查找该属性，如果找不到该属性，将继续在包含的作用域中搜索。

字段	Type	评论
ActionDelete2	ACTIONRECORDHEADER	ActionCode = 0x3B

ActionDelete2 弹出要删除的属性名称到栈中。

动作枚举

ActionEnumerate 获取 ActionScript 对象中所有“槽位”的名称——也就是说，对于对象 obj，所有可以用语法 obj.X 获取的名称 X。ActionEnumerate 用于在 ActionScript 中实现 for..in 语句。

注意：某些特殊的槽位名称被省略；有关这些名称的列表，请在 ECMA-262 标准中搜索术语 DontEnum。

字段	Type	评论
ActionEnumerate	ACTIONRECORDHEADER	ActionCode = 0x46

ActionEnumerate 执行以下操作：

1. 从栈中弹出对象变量的名称（可能包含斜杠路径或点路径语法）。
2. 将一个空值推入栈中，以表示槽名称的结束。
3. 将每个槽名称（字符串）推入栈中。

插槽名称的推送顺序是未定义的。

ActionEquals2

ActionEquals2 与 ActionEquals 相似，但 ActionEquals2 了解类型。应用了 ECMA-262 第 11.9.3 节中的等价比较算法。

字段	Type	评论
ActionEquals2	ACTIONRECORDHEADER	ActionCode = 0x49

ActionEquals2 执行以下操作：

1. 从堆栈中弹出 arg1。
2. 从堆栈中弹出 arg2。
3. 将返回值推送到栈中。

获取成员动作

ActionGetMember 获取对象中的命名属性，并将属性的值压入栈中。

字段	Type	评论
ActionGetMember	ACTIONRECORDHEADER	ActionCode = 0x4E

ActionGetMember 执行以下操作：

1. 弹出成员函数的名称。
2. 从栈中弹出 ScriptObject 对象。
3. 将属性的值推入栈中。

例如，假设 obj 是一个对象，并且它被分配了一个属性 foo，如下所示：

```
obj.foo = 3;
```

然后，使用对象设置为 obj 和名称设置为 foo 的 ActionGetMember 将 3 压入栈中。如果指定的属性不存在，则将 undefined 压入栈中。

实际上，对象参数不能是 Object 类型。如果对象参数是原始类型，如数字、Boolean 或字符串，它将自动转换为临时的包装对象，其类为 Number、Boolean 或 String。因此，可以在原始类型的值上调用包装对象的方法。例如，以下代码正确地打印了 5：

```
var x = "Hello";
trace(x.length);
```

在这种情况下，变量 x 包含原始字符串"Hello"。当检索 x.length 时，会使用 String 类型创建一个临时的包装对象，该对象具有 length 属性。

ActionInitArray

ActionInitArray 在 ScriptObject 中初始化一个数组，与 ActionInitObject 类似。新创建的对象被推送到栈中。栈是唯一现有的对新创建对象的引用。后续的 SetVariable 或 SetMember 操作可以将新创建的对象存储在变量中。

字段	Type	评论
ActionInitArray	ACTIONRECORDHEADER	ActionCode = 0x42

ActionInitArray 弹出 elems，然后从堆栈中移除[arg1, arg2, …, argn]。ActionInitArray 执行以下操作：

1. 获取堆栈中的参数（或元素）数量。
2. 如果存在参数，ActionInitArray 将使用正确的元素数量初始化一个数组对象。
3. 将数组初始化为 ScriptObject。
4. 设置对象类型为 Array。
5. 通过从栈中弹出值来初始化数组元素。

对于所有调用操作（ActionCallMethod、ActionNewMethod、ActionNewObject 和 ActionCallFunction）以及初始化操作（ActionInitObject 和 ActionInitArray），函数的参数以相反的顺序推入栈中，最右边的参数先，最左边的参数后。参数随后按顺序弹出（从第一个到最后一个）。

ActionInitObject

ActionInitObject 初始化一个对象，类似于 ActionInitArray。新创建的对象被推送到栈中。栈是该新创建对象的唯一引用。后续的 SetVariable 或 SetMember 操作可以将新创建的对象存储在变量中。

字段	Type	评论
ActionInitObject	ACTIONRECORDHEADER	ActionCode = 0x43

ActionInitObject 从堆栈中弹出元素。从堆栈中弹出[value1, name1, …, valueN, nameN]。

ActionInitObject 执行以下操作：

1. 从堆栈中弹出初始属性的数量。
2. 初始化对象为 ScriptObject。
3. 设置对象类型为 Object。
4. 从栈中弹出每个初始属性。
   对于每个初始属性，从栈中弹出属性值，然后弹出属性名。属性名被转换为字符串。属性值可以是任何类型。

对于所有调用操作（ActionCallMethod、ActionNewMethod、ActionNewObject 和 ActionCallFunction）以及初始化操作（ActionInitObject 和 ActionInitArray），函数的参数以相反的顺序推入栈中，最右边的参数先，最左边的参数后。参数随后按顺序弹出（从第一个到最后一个）。

动作新方法

ActionNewMethod 调用一个构造函数来创建一个新的对象。一个新的对象被构造并作为 this 关键字的值传递给构造函数。可以向构造函数指定参数。构造函数的返回值被丢弃。新构造的对象被推送到栈中，类似于 ActionCallMethod 和 ActionNewObject。

字段	Type	评论
ActionNewMethod	ACTIONRECORDHEADER	ActionCode = 0x53

ActionNewMethod 执行以下操作：

1. 从堆栈中弹出方法名称。
2. 从堆栈中弹出 ScriptObject。如果方法名称为空，则将 ScriptObject 视为构造函数对象并调用。如果方法名称不为空，则调用 ScriptObject 的指定方法。
3. 从堆栈中弹出参数数量。
4. 执行方法调用。
5. 将新构造的对象推入栈中。如果没有适当的返回值发生（例如，函数没有返回语句），编译器会生成一个推入未定义值并将其推入栈中。未定义的返回值应该从栈中弹出。

对于所有调用操作（ActionCallMethod、ActionNewMethod、ActionNewObject 和 ActionCallFunction）以及初始化操作（ActionInitObject 和 ActionInitArray），函数的参数以相反的顺序推入栈中，最右边的参数先，最左边的参数后。参数随后按顺序弹出（从第一个到最后一个）。

ActionNewObject

ActionNewObject 调用构造函数。创建一个新对象并将其作为 this 关键字传递给构造函数。此外，还可以在栈上指定传递给构造函数的参数。

构造函数的返回值被丢弃。新构造的对象被推入栈中。

ActionNewObject 与 ActionCallFunction 和 ActionNewMethod 相似。

字段	Type	评论
ActionNewObject	ACTIONRECORDHEADER	ActionCode = 0x40

ActionNewObject 执行以下操作：

1. 从栈中弹出对象名称（字符串）this。

2. 从栈中弹出 numArgs（整数）。

3. 弹出栈中的所有参数。

4. 以构造函数的形式调用命名对象，传递指定的参数以及一个新构造的对象作为 this 关键字。

构造函数的返回值被丢弃。

新创建的对象被推送到栈中。

对于所有调用操作（ActionCallMethod、ActionNewMethod、ActionNewObject 和 ActionCallFunction）以及初始化操作（ActionInitObject 和 ActionInitArray），函数的参数以相反的顺序推入栈中，最右边的参数先，最左边的参数后。参数随后按顺序弹出（从第一个到最后一个）。

ActionSetMember

ActionSetMember 设置对象的属性。如果该属性不存在，则创建它。任何现有的属性值将被覆盖。

字段	Type	评论
ActionSetMember	ACTIONRECORDHEADER	ActionCode = 0x4F

ActionSetMember 执行以下操作：

1. 从堆栈中弹出新的值。
2. 从堆栈中弹出对象名称。
3. 从堆栈中弹出对象。

ActionTargetPath

如果堆栈中的对象是 MovieClip 类型，则对象的 target 路径以点表示法推入堆栈。如果对象不是 MovieClip，则结果为未定义，而不是电影剪辑的 target 路径。

字段	Type	评论
动作目标路径	ACTIONRECORDHEADER	动作代码 = 0x45

ActionTargetPath 执行以下操作：

1. 从堆栈中弹出对象。
2. 将目标路径推入堆栈。

ActionWith

定义脚本中的 With 块。

字段	Type	评论
ActionWith	ACTIONRECORDHEADER	ActionCode = 0x94
Size	UI16	代码后续的字节数

ActionWith 执行以下操作：

1. 弹出与 With 相关的对象。
2. 从 ActionWith 标签中解析 With 块的尺寸（主体长度）。
3. 检查调用深度是否超过最大深度，SWF 6 及以后版本的最大深度为 16，SWF 5 版本为 8。如果 With 深度超过最大深度，则跳过 Size 字节的数据，而不是执行。
4. 在 ActionWith 标签之后，下一个 Size 字节的动作代码被认为是 With 块的主体。
5. 将 With 块添加到作用域链中。

类型操作

动作转数字

将栈顶的对象转换为数字，并将数字推回栈中。

对于对象类型，将调用 ActionScript 的 valueOf()方法将对象转换为 Number 类型以进行 ActionToNumber 转换。原始类型之间的转换，如从 String 到 Number，是内置的。

字段	Type	评论
ActionToNumber	ACTIONRECORDHEADER	ActionCode = 0x4A

ActionToNumber 执行以下操作：

1. 从堆栈中弹出对象。
2. 将数字推入栈中。

动作转字符串

ActionToString 将栈顶的对象转换为字符串，并将字符串推回栈中。

对于 Object 类型，ActionScript 的 toString() 方法被调用以将对象转换为字符串类型，用于 ActionToString。

字段	Type	评论
ActionToString	ACTIONRECORDHEADER	ActionCode = 0x4B

ActionToString 执行以下操作：

1. 从堆栈中弹出对象。
2. 将字符串推入栈中。

动作类型

ActionTypeOf 将对象类型推入栈中，相当于 ActionScript 的 TypeOf() 方法。可能的数据类型有：

数字
布尔
字符串
对象
电影剪辑
null
未定义
函数

字段	Type	评论
动作类型为	ACTIONRECORDHEADER	ActionCode = 0x44

ActionTypeOf 执行以下操作：

1. 从堆栈中弹出值以确定类型。
2. 将对象的类型作为字符串推入堆栈。

数学动作

动作加 2

ActionAdd2 与 ActionAdd 类似，但根据参数的数据类型执行加法的方式不同。使用 ECMA-262 第 11.6.1 节中的加法运算符算法。如果应用字符串连接，则连接的字符串是 arg2 后跟 arg1。

字段	Type	评论
ActionAdd2	ACTIONRECORDHEADER	ActionCode = 0x47

ActionAdd2 执行以下操作：

1. 从堆栈中弹出 arg1。
2. 从堆栈中弹出 arg2。
3. 将结果推回堆栈。

无动作 2

ActionLess2 计算 arg1 是否小于 arg2，并将布尔返回值推入堆栈。此操作类似于 ActionLess，但根据参数的数据类型执行不同的比较。

ECMA-262 第 11.8.5 节中使用的抽象关系比较算法。

字段	Type	评论
ActionLess2	ACTIONRECORDHEADER	ActionCode = 0x48

ActionLess2 执行以下操作：

1. 从堆栈中弹出 arg1。
2. 从堆栈中弹出 arg2。
3. 比较 arg2 < arg1。
4. 将返回值（一个布尔值）压入堆栈。

动作模数

ActionModulo 计算 x 对 y 的模。如果 y 为 0，则将 NaN（0x7FC00000）压入栈中。

字段	Type	评论
ActionModulo	ACTIONRECORDHEADER	ActionCode = 0x3F

ActionModulo 执行以下操作：

1. 从栈中弹出 then 和 y。
2. 将值 x % y 推送到栈上。

栈操作符动作

位与操作

ActionBitAnd 从栈中弹出两个数字，执行按位与操作，并将一个 S32 数字推送到栈上。在执行按位操作之前，参数被转换为 32 位无符号整数。结果是带符号的 32 位整数。

字段	Type	评论
ActionBitAnd	ACTIONRECORDHEADER ActionCode = 0x60	

ActionBitAnd 执行以下操作：

1. 从栈中弹出 arg1 然后 arg2。
2. 将结果推入栈中。

位左移操作

ActionBitLShift 弹出位移计数 arg，然后值从栈中弹出。值参数被转换为 32 位有符号整数，并且只使用最低的 5 位作为位移计数。值参数中的位按位移计数向左移动。ActionBitLShift 将一个 S32 数字推入栈中。

字段	Type	评论
ActionBitLShift	ACTIONRECORDHEADER	ActionCode = 0x63

动作位左移执行以下操作：

1. 弹出栈中的移位计数参数，然后从栈中弹出值。

2. 将结果推入栈中。

ActionBitOr

ActionBitOr 从栈中弹出两个数字，执行按位或操作，并将一个 S32 数字压入栈中。在执行按位操作之前，参数被转换为 32 位无符号整数。结果是带符号的 32 位整数。

字段	Type	评论
ActionBitOr	ACTIONRECORDHEADER	ActionCode = 0x61

ActionBitOr 执行以下操作：

1. 从栈中弹出 arg1 然后 arg2。

2. 将结果推入栈中。

ActionBitRShift

ActionBitRShift 从堆栈中弹出移位计数。弹出堆栈中的值。值参数被转换为 32 位有符号整数，并且只使用最低的 5 位作为移位计数。

将 arg 值中的位向右移动移位计数。ActionBitRShift 将 S32 数字推入堆栈。

字段	Type	评论
ActionBitRShift	ACTIONRECORDHEADER	ActionCode = 0x64

ActionBitRShift 执行以下操作：

1. 从堆栈中弹出移位计数。

2. 从堆栈中弹出要移位的值。

3. 将结果推入栈中。

动作位无符号右移

ActionBitURShift 从堆栈中弹出值和移位计数参数。值参数被转换为 32 位有符号整数，只使用最低的 5 位作为移位计数。

将 arg 值中的位按移位计数向右移动。ActionBitURShift 将一个 UI32 数字推入堆栈。

字段	Type	评论
ActionBitURShift	ACTIONRECORDHEADER	ActionCode = 0x65

ActionBitURShift 执行以下操作：

1. 从堆栈中弹出移位计数。
2. 从堆栈中弹出要移位的值。
3. 将结果推入栈中。

ActionBitXor

ActionBitXor 从堆栈中弹出两个数字，执行按位异或操作，并将一个 S32 数字推入堆栈。

在执行按位操作之前，将参数转换为 32 位无符号整数。结果是带符号的 32 位整数。

字段	Type	评论
ActionBitXor	ACTIONRECORDHEADER	ActionCode = 0x62

ActionBitXor 执行以下操作：

1. 从堆栈中弹出 arg1 和 arg2
2. 将结果推回堆栈。

动作递减

ActionDecrement 从堆栈中弹出一个值，将其转换为数字类型，减去 1，然后将其推回堆栈。

字段	Type	评论
ActionDecrement	ACTIONRECORDHEADER	ActionCode = 0x51

ActionDecrement 执行以下操作：

1. 从堆栈中弹出数字。
2. 将结果推送到堆栈上。

ActionIncrement

ActionIncrement 从堆栈中弹出一个值，将其转换为数字类型，增加 1，然后将其推回堆栈。

字段	Type	评论
动作增量	ACTIONRECORDHEADER	动作代码 = 0x50

动作增量执行以下操作：

1. 从堆栈中弹出数字。
2. 将结果推入栈中。

ActionPush（增强）

在 SWF 5 中，ActionPush 增加了八种新类型。有关 ActionPush 的更多信息，请参阅 SWF 4 动作。

ActionPushDuplicate

ActionPushDuplicate 将栈顶（当前返回值）的副本推入栈中。

字段	Type	评论
动作推送重复	ACTIONRECORDHEADER	ActionCode = 0x4C

操作返回

ActionReturn 强制将返回项推离栈并返回。如果返回不合适，则返回项将被丢弃。

字段	Type	评论
ActionReturn	ACTIONRECORDHEADER	ActionCode = 0x3E

ActionReturn 从栈中弹出一个值。

ActionStackSwap

ActionStackSwap 交换堆栈上顶部两个 ScriptAtoms。

字段	Type	评论
ActionStackSwap	ACTIONRECORDHEADER	ActionCode = 0x4D

ActionStackSwap 执行以下操作：

1. 从栈中弹出 Item1，然后是 Item2。
2. 将 Item1 和 Item2 依次推回栈中。

ActionStoreRegister

ActionStoreRegister 读取栈中的下一个对象（不弹出）并将其存储在四个寄存器之一中。如果使用 ActionDefineFunction2，则最多可提供 256 个寄存器。

字段	Type	评论
ActionStoreRegister	ACTIONRECORDHEADER	ActionCode = 0x87
寄存器编号	UI8	

ActionStoreRegister 从 StoreRegister 标签中解析寄存器编号。

SWF 6 动作模型

SWF 6 添加了 DoInitAction 动作定义标签和一些新的动作字节码。

SWF 6 动作

以下是在 SWF 6 中可用的动作：

- DoInitAction
- 行为实例化
- 行为枚举 2
- 行为严格等于
- 行为大于
- 动作字符串大于

初始化动作

DoInitAction 标签与 DoAction 标签类似：它定义了一系列要执行的字节码。然而，使用 DoInitAction 定义的动作用于比常规的 DoAction 动作更早，并且只执行一次。

在某些情况下，动作必须在创建特定精灵的第一个实例的 ActionScript 表示之前执行。最常见的此类动作是调用 Object.registerClass 将 ActionScript 类与精灵关联起来。此类调用通常在 ActionScript 语言的 #initclip 预言中找到。DoInitAction 用于实现 #initclip 预言。

DoInitAction 标签指定了动作将应用到的特定精灵。一个帧可以包含多个 DoInitAction 标签；它们的动作将按照标签出现的顺序执行。然而，SWF 文件中对于任何特定的精灵只能有一个 DoInitAction 标签。

指定的动作将在 DoInitAction 标签出现的帧的正常动作之前立即执行。这仅在第一次遇到该帧时发生；稍后再次播放到同一帧时，DoInitAction 中提供的动作将被跳过。

从 SWF 9 开始，如果 FileAttributes 标签的 ActionScript3 字段为 1，则将忽略 DoInitAction 标签的内容。

注意：在 DoAction 标签的开始处指定动作与在 DoInitAction 标签中指定动作不同。Flash Player 在执行 DoAction 标签中的第一个动作之前会采取一些步骤，最重要的是创建代表精灵的 ActionScript 对象。DoInitAction 中的动作在这些隐式步骤执行之前发生。

字段	Type	评论
标题	记录头	标签类型 = 59
精灵 ID	UI16	这些动作应用到的精灵
动作	ACTIONRECORD[零个或多个]	执行动作列表
动作结束标志	UI8	总是设置为 0

行为实例化

ActionInstanceOf 实现了 ActionScript 的 instanceof() 操作符。这是一个布尔操作符，表示左操作数（通常是对象）是否是作为右操作数传递的构造函数表示的类的实例。

此外，从 SWF 7 或更高版本开始，ActionInstanceOf 还支持接口。如果右操作数构造函数是接口对象的引用，并且左操作数实现了此接口，ActionInstanceOf 准确报告左操作数是右操作数的实例。

字段	Type	评论
ActionInstanceOf	ACTIONRECORDHEADER	ActionCode = 0x54

ActionInstanceOf 执行以下操作：

1. 弹出栈上的 constr 和 obj。
2. 判断 obj 是否为 constr 的实例。
3. 将返回值（一个布尔值）压入堆栈。

行为枚举 2

ActionEnumerate2 与 ActionEnumerate 类似，但使用对象类型的堆栈参数，而不是使用字符串来指定其名称。

字段	Type	评论
ActionEnumerate2	ACTIONRECORDHEADER	ActionCode = 0x55

ActionEnumerate2 执行以下操作：

1. 弹出堆栈中的对象。

2. 将一个空值推入栈中，以表示槽名称的结束。

3. 将 obj 中的每个槽位名称（字符串）推入栈中。

注意：槽位名称的推入顺序是未定义的。

行为严格等于

ActionStrictEquals 与 ActionEquals2 类似，但两个参数必须为同一类型才能被认为是相等的。实现了 ActionScript 语言中的‘===’运算符。

字段	Type	评论
ActionStrictEquals	ACTIONRECORDHEADER	ActionCode = 0x66

ActionStrictEquals 执行以下操作：

1. 从堆栈中弹出 arg1 然后 arg2。

2. 将返回值（布尔值）推送到堆栈。

行为大于

ActionGreater 是 ActionLess2 的完全相反。最初并没有 ActionGreater，因为它可以通过反转参数推入的顺序，然后执行 ActionLess 再执行 ActionNot 来模拟。然而，这种参数反转导致了参数评估顺序的颠倒，这在少数情况下导致了意外。

字段	Type	评论
ActionGreater	ACTIONRECORDHEADER	ActionCode = 0x67

ActionGreater 执行以下操作：

1. 从堆栈中弹出 arg1 和 arg2。

2. 比较 arg2 是否大于 arg1。

3. 将返回值（一个布尔值）推入堆栈。

动作字符串大于

ActionStringGreater 是 ActionStringLess 的相反操作。此操作码的添加原因与 ActionGreater 相同。

字段	Type	评论
ActionStringGreater	ACTIONRECORDHEADER	ActionCode = 0x68

ActionStringGreater 执行以下操作：

1. 从堆栈中弹出 arg1 和 arg2。
2. 比较 arg2 是否大于 arg1，使用字节对字节比较。
3. 将返回值（一个布尔值）推入堆栈。

SWF 7 动作模型

SWF 7 动作

以下是在 SWF 7 中可用的动作：

- ActionDefineFunction2
- 行为扩展
- 行为播发操作
- 行为实现操作
- 行为尝试
- 动作投掷

动作定义函数 2

动作定义函数 2 与动作定义函数类似，具有一些额外功能，可以帮助加快函数调用的执行速度，通过防止在函数的激活对象中创建未使用的变量，并通过允许用变量数量的寄存器替换局部变量来实现。使用动作定义函数 2，一个函数可以为其自己分配多达 256 个寄存器的私有集合。参数或局部变量可以被寄存器替换，该寄存器加载了值而不是将值存储在函数的激活对象中。（激活对象是一个隐式局部作用域，包含命名参数和局部变量。有关激活对象的进一步描述，请参阅 ECMA-262 标准。）

对象。（激活对象是一个隐式局部作用域，包含命名参数和局部变量。有关激活对象的进一步描述，请参阅 ECMA-262 标准。）

ActionDefineFunction2 还包括六个标志来指示 Flash Player 预加载变量，以及三个标志来抑制变量。通过设置 PreloadParentFlag、PreloadRootFlag、PreloadSuperFlag、PreloadArgumentsFlag、PreloadThisFlag 或 PreloadGlobalFlag，可以在函数执行前将常用变量预加载到寄存器中（分别对应_parent、_root、super、arguments、this 或_global）。使用标志 SuppressSuper、

抑制参数，以及抑制此，常见的变量 super、参数和 this 不会被创建。通过使用抑制标志，Flash Player 避免预先评估变量，从而节省时间并提高性能。

对于_parent、_root 或_global 没有提供抑制标志，因为 Flash Player 总是按需评估这些变量；永远不会浪费时间去预先评估这些变量。

不允许为任何变量同时指定预加载标志和抑制标志。

ActionDefineFunction2 指定的函数体应使用 ActionPush 和 ActionStoreRegister 为分配到寄存器的局部变量，不能使用 ActionGetVariable 和 ActionSetVariable。

Flash Player 6 版本 65 及以后支持 ActionDefineFunction2。

字段	Type	评论
动作定义函数 2	ACTIONRECORDHEADER	操作码 = 0x8E
函数名	字符串	函数名称，匿名时为空
参数数量	UI16	参数数量（# of parameters）
注册计数	UI8	分配的寄存器数量（从 0 到 254），最多 255 个寄存器
预加载父标志	UB[1]	0 = 不预加载_parent 到寄存器 1 = 预加载_parent 到寄存器
预加载根标志	UB[1]	0 = 不预加载_root 到寄存器 1 = 预加载_root 到寄存器
抑制超类标志	UB[1]	0 = 创建超级变量 1 = 不创建超级变量
PreloadSuperFlag	UB[1]	0 = 不预加载超级变量到寄存器 1 = 预载超级到寄存器
SuppressArgumentsFlag	UB[1]	0 = 创建参数变量

		1 = 不要创建参数变量
预加载参数标志	UB[1]	0 = 不将参数预加载到寄存器中 1 = 将参数预加载到寄存器中
抑制此标志	UB[1]	0 = 创建此变量 1 = 不要创建此变量
PreloadThisFlag	UB[1]	0 = 不要将其预加载到寄存器中 1 = 预载到寄存器中
保留	UB[7]	总是 0
PreloadGlobalFlag	UB[1]	0 = 不要预载 _global 到寄存器 1 = 预载全局变量到寄存器
参数	REGISTERPARAM[NumParams]	见 REGISTERPARAM，后续
代码大小	UI16	代码后续的字节数

REGISTERPARAM 被定义为如下：

字段	Type	评论
注册	UI8	对于函数的每个参数，可以指定一个寄存器。如果指定的寄存器为零，则参数将在激活对象中创建为名为 ParamName 的变量，可以通过 ActionGetVariable 和 ActionSetVariable 进行引用。如果指定的寄存器不为零，则参数将被复制到寄存器中，可以通过 ActionPush 和 ActionStoreRegister 进行引用，并且不会在激活对象中创建变量。
ParamName	字符串	参数名称

ActionDefineFunction2 后跟的函数体由进一步的动作代码组成，就像 ActionDefineFunction 一样。

Flash Player 首先将每个参数复制到指定的寄存器中，以选择寄存器编号

相应的 REGISTERPARAM 记录。接下来，预加载的变量按此顺序复制到从 1 开始的寄存器中，包括参数、super、_root、_parent 和_global，跳过任何不需要预加载的变量。（SWF 文件必须准确指定预加载变量将使用的寄存器，并确保没有任何参数使用此范围内的寄存器号，否则该参数将被预加载变量覆盖。）

NumParams 的值应等于参数寄存器的数量。RegisterCount 的值应等于 NumParams 加上预加载变量和所需局部变量寄存器的数量。

例如，如果 NumParams 为 2，RegisterCount 为 6，PreloadThisFlag 为 1，PreloadRootFlag 为 1，则 REGISTERPARAM 记录可能会指定寄存器 3 和 4。寄存器 1 将代表 this，寄存器 2 将代表_root，寄存器 3 和 4 将代表第一个和第二个参数，寄存器 5 和 6 将用于局部变量。

行为扩展

ActionExtends 实现了 ActionScript 的 extends 关键字。ActionExtends 在两个类之间创建继承关系，称为子类和超类。

SWF 7 将 ActionExtends 添加到文件格式中，以避免对超类构造函数的无效调用（这在 ActionScript 1.0 下建立继承关系时会发生）。考虑以下代码：

```
Subclass.prototype = new Superclass();
```

在 ActionExtends 存在之前，此代码会导致对 Superclass 超构造函数的虚假调用。现在，当 ActionScript 编译器遇到类 A 扩展 B 的代码时，会生成 ActionExtends 来设置 A 和 B 之间的继承关系。

字段	Type	评论
ActionExtends	ACTIONRECORDHEADER	ActionCode = 0x69

ActionExtends 执行以下操作：

1. 从栈中弹出 ScriptObject 父类构造函数。
2. 从栈中弹出 ScriptObject 子类构造函数。
3. 创建一个新的 ScriptObject。
4. 将新 ScriptObject 的 proto 属性设置为超类的 prototype 属性。
5. 将新 ScriptObject 的 constructor 属性设置为超类。
6. 将子类的 prototype 属性设置为新的 ScriptObject。这些步骤等同于以下 ActionScript 代码：

```
子类原型 = new Object(); 子类原型. proto = 超类原型; 子类原型. 构造函
数 = 超类;
```

行为播发操作

这些步骤等同于以下 ActionScript：ActionCastOp 实现了 ActionScript 的强制类型转换操作符，允许将一个数据类型转换为另一个数据类型。ActionCastOp 从堆栈中弹出一个对象，并尝试将其转换为类或由构造函数表示的接口的实例。

字段	Type	评论
ActionCastOp	ACTIONRECORDHEADER	ActionCode = 0x2B

ActionCastOp 执行以下操作：

1. 从堆栈中弹出脚本对象以取消引用。
2. 从堆栈中弹出构造函数。
3. 判断对象是否为构造函数的实例（与 ActionInstanceOf 执行相同的比较）。
4. 如果对象是构造函数的实例，则弹出的脚本对象将被推回堆栈。

如果对象不是构造函数的实例，则将一个 null 值压入栈中。

行为实现操作

ActionImplementsOp 实现了 ActionScript 的 implements 关键字。ActionImplementsOp 动作指定了类实现的接口，以便由 ActionCastOp 使用。ActionImplementsOp 还可以指定接口实现的接口，因为接口可以扩展其他接口。

字段	Type	评论
ActionImplementsOp	ACTIONRECORDHEADER	ActionCode = 0x2C

ActionImplementsOp 执行以下操作：

1. 从栈中弹出构造函数。构造函数表示将要实现接口的类。构造函数必须具有原型属性。
2. 从栈中弹出已实现接口的数量。
3. 对于每个接口计数，从栈中弹出构造函数。构造函数表示一个接口。

4. 将构造函数的接口列表设置为上一步收集到的数组，并将接口数量设置为第 2 步弹出的数量。

行为尝试

ActionTry 定义异常情况的处理程序，实现了 ActionScript 的 try、catch 和 finally 关键字。

字段	Type	评论
ActionTry	ACTIONRECORDHEADER	ActionCode = 0x8F
保留	UB[5]	总是零
捕获注册标志	UB[1]	0 - 不将捕获的对象放入注册表（而是在命名变量中存储）1 - 将捕获的对象放入注册表（不在命名变量中存储）
最后块标志	UB[1]	0 - 没有 finally 块 1 - 有 finally 块
捕获块标志	UB[1]	0 - 无捕获块 1 - 有捕获块
尝试大小	UI16	尝试块长度
捕获大小	UI16	捕获块长度
finally 块大小	UI16	finally 块长度
捕获名称	如果 CatchInRegisterFlag = 0，则 STRING	捕获变量的名称
捕获注册	如果 CatchInRegisterFlag = 1，UI8	注册以捕获到
尝试体	UI8[尝试大小]	尝试块的主体
捕获体	UI8[捕获大小]	捕获块（如果有的话）的主体
FinallyBody	UI8[FinallySize]	（如果有的话）finally 块的主体

捕获大小（CatchSize）和最终大小（FinallySize）字段始终存在，无论捕获块标志（CatchBlockFlag）或最终块标志（FinallyBlockFlag）设置是否为 1。

try、catch 和 finally 块不使用结束标签来标记各自块的结束。相反，通过 try 大小（TrySize）、捕获大小（CatchSize）和最终大小（FinallySize）值来设置块长度。

动作投掷

ActionThrow 实现了 ActionScript 的 throw 关键字。ActionThrow 用于指示或抛出异常条件，该条件由使用 ActionTry 声明的异常处理器处理。

如果 try 块中的任何代码抛出对象，则控制权传递到存在的话的 catch 块，然后传递到存在的话的 finally 块。finally 块始终执行，无论是否抛出错误。

如果函数内部发生异常条件，并且该函数不包含 catch 处理器，则函数和任何调用函数将退出，直到找到 catch 块（在所有级别执行所有 finally 处理器）。

可以抛出任何 ActionScript 数据类型，尽管通常的使用是抛出对象。

字段	Type	评论
动作投掷	ACTIONRECORDHEADER	动作代码 = 0x2A

动作投掷将待抛值弹出栈

SWF 9 动作模型

SWF 9 增加了 DoABC 动作定义标签。此标签包含一个 .abc 字节码块，由 ActionScript 3.0 虚拟机解析。DoABC 标签是传递 ActionScript 3.0 字节码的主要载体。

DoABC

DoABC 标签类似于 DoAction 标签：它定义了一系列要执行的字节码。然而，DoABC 标签中的字节码是在 ActionScript 3.0 虚拟机上运行的。

字段	Type	评论
标题	记录头	标签类型 = 82
标志	UI32	一个 32 位的标志值，可能包含以下位被设置： kDoAbcLazyInitializeFlag = 1：表示 ABC 块不应立即执行，而只进行解析。稍后的 finddef 可能会引起其脚本的执行。
Name	字符串	分配给字节码的名称
ABC 数据	BYTE[]	一个由 ActionScript 3.0 虚拟机解析的.abc 字节码块，直到标签结束。

关于 ABCData 字段的内容和格式详情，请参阅 Adobe ActionScript 虚拟机 2（AVM2）概述，网址为 www.adobe.com/go/avm2overview/。

SWF 10 动作模型

SWF 10 的动作模型没有变化。

第六章：形状

SWF 形状架构旨在设计得紧凑、灵活，并且能够在屏幕上非常快速地渲染。它与大多数矢量格式类似，形状是通过称为路径的边列表定义的。路径可以是闭合的，即路径的起点和终点相遇以闭合图形，或者开放的，即路径形成开放式线条。路径可以包含直线边、曲线边和“笔抬起并移动”命令的混合。后者允许通过单个形状结构描述多个不相连的图形。

填充样式定义了路径所包围区域的显示效果。SWF 文件格式支持的填充样式包括颜色、渐变或位图图像。

线样式定义了路径轮廓的显示效果。线样式可以是任何厚度和颜色的线条。

大多数矢量格式只允许每个路径一个填充和线样式。SWF 文件格式通过允许每个边缘都有自己的线和填充样式来扩展这一概念。当填充样式在路径中间改变时，这可能会导致不可预测的结果。

Adobe Flash 创作工具还支持每条边两种填充样式，每边一种：填充样式 0 和填充样式 1。当形状的两边都填充时，应始终先使用填充样式 0，然后使用填充样式 1。

形状概述

形状由以下元素组成：

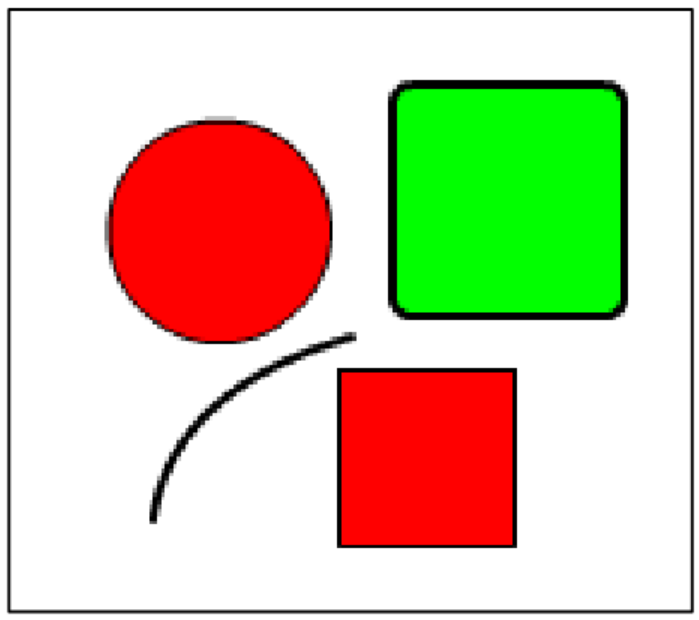
- CharacterId—一个 16 位值，唯一标识此形状在字典中的“字符”。CharacterId 可以在控制标签（如 PlaceObject）中引用。字符可以被重复使用，并与其他字符组合以形成更复杂的形状。

- 边界框——完全包围形状的矩形。
- 填充样式数组——一个形状中使用的所有填充样式的列表。
- 线样式数组——一个形状中使用的所有线样式的列表。
- 形状记录数组——形状记录的列表。形状记录可以定义直线或曲线边缘、样式更改或移动绘图位置。

注意：线条和填充样式仅定义一次，可由形状的任何边使用（并重复使用）。

形状示例

以下示例看起来是一组形状的集合，但可以用一个 DefineShape 标签来描述。



红色圆形、红色正方形和绿色圆角矩形是闭合路径。曲线是开放路径。红色正方形由所有直线边组成，红色圆形由所有曲线边组成，而圆角矩形则是由曲线边和直线边交替组成。

有两种填充样式，绿色和红色纯色，以及两种线条样式，1 像素黑色和 2 像素黑色。红色圆形和红色正方形共享相同的填充和线条样式。圆角矩形和曲线共享相同的线条样式。

下面是如何使用 SWF 文件格式描述这个示例。

定义填充样式：

1. 首先，使用 FILLSTYLEARRAY 定义填充样式。两种独特的填充样式是纯红色和纯绿色。

2. 接着是 LINESTYLEARRAY，其中包含两种独特的线条样式：1 像素黑色和 2 像素黑色。

3. 然后是一个形状记录数组（参见形状记录）。

所有形状记录具有相似的结构，但可以具有不同的含义。形状记录可以定义直线或曲线边缘、样式变化，或者它可以移动当前绘图位置。

定义曲线线：

1. 第一个形状记录选择了 2 像素宽的线条样式，并通过设置 StateMoveTo 标志将绘图位置移动到曲线线的顶部。

2. 下一个形状记录是一个曲线边缘，它延伸到线的底部。路径没有闭合。

定义红色正方形：

下一形状记录选择 1 像素线条样式和红色填充样式。它还将绘图位置移动到红色矩形的左上角。

2. 以下四个形状记录是直线边。最后一个边必须结束在左上角。Flash Player 要求闭合图形必须显式连接。也就是说，第一个点和最后一个点必须重合。

定义红色圆圈：

1. 下一个形状记录不会改变任何样式设置，但将绘图位置移动到红色圆圈的顶部。

2. 以下八个形状记录是定义圆的曲线边。同样，路径必须从哪里开始就结束在哪里。

定义绿色圆角矩形：

1. 下一个形状记录选择 2 像素宽的线条样式和绿色填充。它还将绘图位置移动到圆角矩形的左上角。

2. 以下十二个形状记录是直线形状记录（边）和曲线形状记录（圆角）的混合。路径在开始的地方结束。

形状结构

填充样式

SWF 文件格式支持三种基本的形状填充类型。

- 实色填充 实色填充是一种简单的 RGB 或 RGBA 颜色，用于填充形状的一部分。alpha 值为 255 表示完全不透明填充。alpha 值为 0 表示完全透明填充。任何介于 0 到 255 之间的 alpha 值将表示部分透明。

- 渐变填充 渐变填充可以是线性渐变或径向渐变。有关渐变定义的详细介绍，请参阅渐变。

- 位图填充 位图填充指的是位图字符 Id。有两种样式：裁剪和铺贴。裁剪位图填充如果填充超出位图边缘，会在边缘重复颜色。铺贴填充如果填充超出位图边缘，会重复位图。

填充样式数组

填充样式数组列举了多种填充样式。填充样式数组的格式在以下表格中描述：

字段	Type	评论
填充样式数量	UI8	填充样式的数量。
FillStyleCountExtended	如果填充样式计数等于 0xFF，UI16	扩展填充样式的计数。仅支持 Shape2 和 Shape3。
填充样式	FILLSTYLE[FillStyleCount]	填充样式数组

FILLSTYLE

以下表格描述了文件中填充样式值的格式：

字段	Type	评论
填充样式类型	UI8	填充样式类型： 0x00 = 实心填充 0x10 = 线性渐变填充 0x12 = 径向渐变填充 0x13 = 焦点径向渐变填充（仅限 SWF 8 文件格式及以后版本） 0x40 = 重复位图填充 0x41 = 修剪位图填充 0x42 = 非平滑重复位图 0x43 = 非平滑裁剪位图
颜色	如果类型 = 0x00，则 RGBA（如果 Shape3）；RGB（如果 Shape1 或 Shape2）	带不透明度信息的纯色填充颜色。
渐变矩阵	如果类型为 0x10、0x12 或 0x13，则使用 MATRIX	渐变填充的矩阵

渐变	如果类型为 0x10 或 0x12，则 GRADIENT；如果类型为 0x13，则 FOCALGRADIENT（仅限 SWF 8 及	渐变填充
位图 ID	如果类型为 0x40、0x41、0x42 或 0x43，则 UI16	位图填充字符的 ID。
BITMAP BitmapMatrix	如果类型为 0x40、0x41、0x42 或 0x43，则 MATRIX	位图填充的矩阵。

线型样式

线型样式数组列举了多个线型样式。

LINESTYLEARRAY

线型数组的格式在以下表格中描述：

字段	Type	评论
线样式数量	UI8	线样式的数量。
线样式数量扩展	如果线样式数量 = 0xFF，则 UI16	线样式的扩展计数。
线样式	如果是 Shape1、Shape2 或 Shape3，则 LINESTYLE[count]。如果是	线样式的数组。

线型

线型表示线的宽度和颜色。文件中线条样式值的格式在以下表中描述：

字段	Type	评论
宽度	UI16	线宽以 twips 为单位。
颜色	RGB（形状 1 或形状 2）RGBA（形状 3）	包含 Alpha 通道信息的颜色值，用于形状 3。

备注 1：在 SWF 8 中引入 LINESTYLE2 之前，SWF 文件格式中的所有线条都具有圆角连接和圆帽。可以通过一个非常窄的形状来模拟不同的连接样式和端样式，该形状看起来与所需的笔触相同。

注意 2：SWF 文件格式不支持虚线或点线样式。可以通过将路径分割成一系列短线条来模拟虚线。

LINESTYLE2

LINESTYLE2 在 LINESTYLE 记录的基础上扩展了功能，允许使用新的连接类型和端点样式，以及缩放选项和填充描边的功能。为了使用 LINESTYLE2，形状必须使用 DefineShape4 定义，而不是 DefineShape、DefineShape2 或 DefineShape3。

虽然 LINESTYLE 记录仅允许使用圆角连接和圆端点样式，但 LINESTYLE2 还支持斜接连接和斜边连接，以及方形端点样式和无端点样式。以下图表展示了完整的连接和端点样式系列：

斜接连接　　圆角连接　　斜角连接

无帽　　圆帽　　方形帽

当使用 LINESTYLE2 进行斜接时，必须指定 MiterLimitFactor，并用于计算最大斜接长度：

```
最大斜接长度 = LINESTYLE2 MiterLimitFactor * LINESTYLE2 宽度
```

如果斜接超过最大斜接长度，Flash Player 将截断斜接。注意，MiterLimitFactor 是一个 8.8 位定点值。

LINESTYLE2 还包括像素提示选项，以纠正模糊的垂直或水平线。

字段	Type	评论
宽度	UI16	线宽以 twips 为单位。
StartCapStyle	UB[2]	开始端点样式：；0 = 圆形端点；1 = 无端点；2 = 方形端点
JoinStyle	UB[2]	连接样式：0 = 圆角连接；1 = 斜接连接；2 = 斜边连接
有填充标志	UB[1]	如果为 1，填充在 FillType 中定义。如果为 0，使用颜色字段。
无水平缩放标志	UB[1]	如果为 1，笔触粗细不会缩放

		对象在水平方向上缩放。
如果没有启用 NoVScaleFlag，笔触粗细将不会缩放	UB[1]	如果设置为 1，当对象垂直缩放时，笔触粗细将不会缩放。
PixelHintingFlag	UB[1]	如果设置为 1，所有锚点都将对齐到完整像素。
保留	UB[5]	必须为 0。
NoClose	UB[1]	如果为 1，当笔画的最后一个点与第一个点相匹配时，笔画不会闭合。Flash Player 将应用端点而不是连接。
EndCapStyle	UB[2]	封头样式：0 = 圆顶；1 = 无顶；2 = 方顶
斜接限制系数	如果连接样式 = 2，则 UI16	斜接限制系数是一个 8.8 的定点值。
颜色	如果 HasFillFlag = 0，RGBA	包含 Alpha 通道的颜色值。
填充类型	如果 HasFillFlag = 1，则使用 FILLSTYLE	这笔画的填充样式。

形状结构

SHAPE 结构定义了一个没有填充样式或线条样式信息的形状。

SHAPE

SHAPE 被 DefineFont 标签使用，用于定义字符字形。

字段	Type	评论
填充索引位数	UB[4]	填充索引位数数量。
线索引位数	UB[4]	线索引位数数量。
形状记录	SHAPERECORD[一个或多个]	形状记录（见下文）。

SHAPEWITHSTYLE

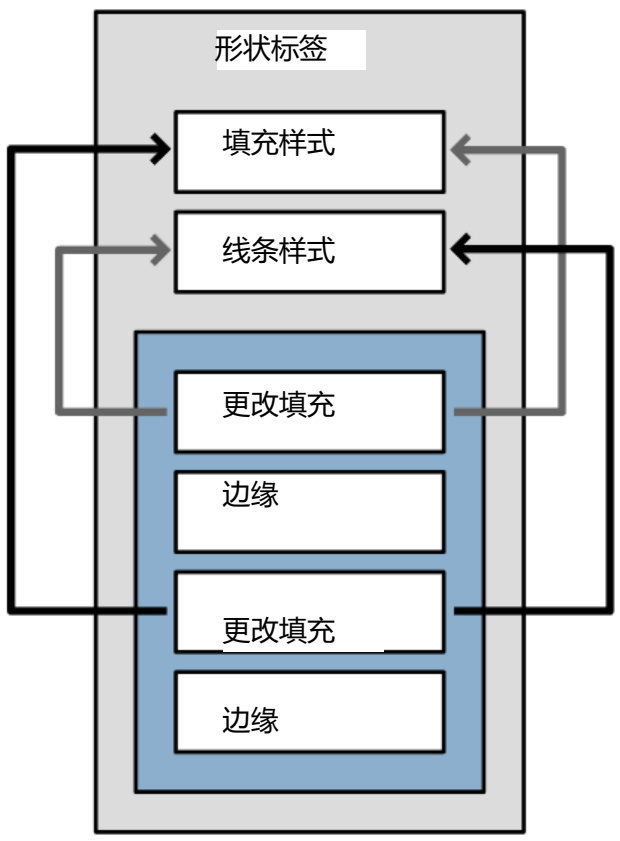
SHAPEWITHSTYLE 结构通过包含填充样式和线条样式信息扩展了 SHAPE 结构。

SHAPEWITHSTYLE 由 DefineShape 标签使用。

字段	Type	评论
填充样式	填充样式数组	填充样式数组
线样式	线样式数组	线样式的数组。
填充索引位数	UB[4]	填充索引位数数量。
线索引位数	UB[4]	线索引位数数量。
形状记录	SHAPERECORD[一个或多个]	形状记录（见下文）。

注意：LINESTYLELARRAY 和 FILLSTYLEARRAY 从索引 1 开始，而不是索引 0。

以下图示 SHAPEWITHSTYLE 结构。



首先，定义填充样式和线条样式。这些样式只定义一次，并通过数组索引引用。

蓝色区域表示形状记录数组。第一个形状记录从填充样式数组中选择填充，并将绘图位置移动到形状的起始位置。随后是一系列定义形状的边记录。下一个记录更改填充样式，随后使用这种新样式填充的边记录。

此标签是一个完全自主的对象。样式更改记录仅指明在此标签中定义的填充和线条样式。

形状记录

形状记录有四种类型：

- 结束形状记录

- 样式变更记录
- 直边唱片
- 弯边唱片

每个形状记录都以 TypeFlag 开始。如果 TypeFlag 为零，则形状记录为非边缘记录，并跟随五个标志信息位。

结束形状记录

形状记录结束标记仅表示形状记录数组的结束。它是一个非边缘记录，所有五个标志都等于零。

字段	Type	评论
类型标志	UB[1]	非边缘记录标志。始终为 0。
形状结束标记	UB[5]	形状结束标志。始终为 0。

样式更改记录

样式更改记录也是一种非边缘记录。它可以用于以下操作：

1. 选择绘制填充或线条样式。
2. 移动当前绘图位置（不进行绘制）。
3. 用一组新样式替换当前填充和线条样式数组。

由于填充和线条样式通常在新的路径开始时发生变化，因此在一个记录中执行多个操作是有用的。例如，假设一个 DefineShape 标签定义了一个红色圆圈和一个蓝色正方形。在圆圈闭合后，需要移动绘图位置，并将红色填充替换为蓝色填充。样式更改记录可以在此处使用单个形状记录实现。

字段	Type	评论
类型标志	UB[1]	非边缘记录标志。始终为 0。
StateNewStyles	UB[1]	新样式标志。仅由 DefineShape2 和 DefineShape3 使用。
StateLineStyle	UB[1]	线型更改标志
状态填充样式 1	UB[1]	填充样式 1 更改标志

状态填充样式 0	UB[1]	填充样式 0 更改标志。
状态移动到。	UB[1]	移动到标志。
移动位。	如果是 StateMoveTo，UB[5]	移动位数
MoveDeltaX	如果是 StateMoveTo，SB[MoveBits]	ΔX 值
移动 ΔY	如果状态移动到，SB[移动位]	ΔY 值
填充样式 0	如果状态为 FillStyle0，则 UB[填充位]	填充 0 样式。
填充样式 1	如果是 StateFillStyle1，UB[填充位]	填充 1 样式。
线样式	如果是 StateLineStyle，UB[线条位]	线型样式
填充样式	如果启用 StateNewStyles，则 FILLSTYLEARRAY	新填充样式的数组
线样式	如果启用 StateNewStyles，则 LINESTYLEARRAY	新行样式数组。
填充索引位数	如果 StateNewStyles，UB[4]	新样式的填充索引位数。
线索引位数	如果 StateNewStyles，UB[4]	新样式的行索引位数量。

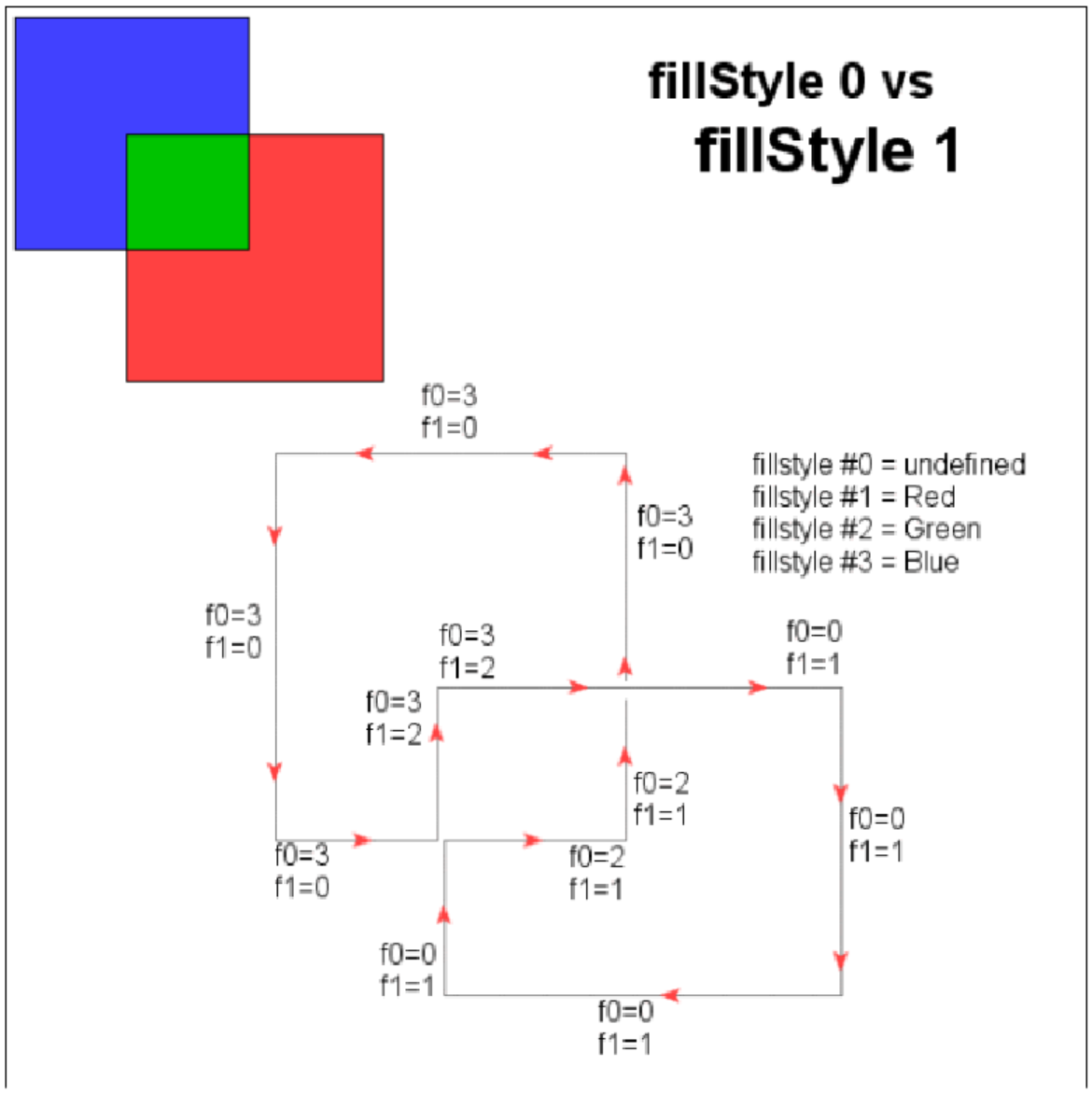
MoveDeltaX 和 MoveDeltaY 是相对于形状原点。

样式数组从索引 1 开始，而不是索引 0。FillStyle = 1 指的是填充样式数组中的第一个样式，FillStyle = 2 指的是填充样式数组中的第二个样式，依此类推。填充样式索引为零表示路径不填充，线条样式索引为零表示路径没有描边。初始时，填充和线条样式索引设置为零——没有填充或描边。

FillStyle0 和 FillStyle1

Adobe Flash 创作工具支持每条边两个填充样式，每边一个：填充样式 0 和填充样式 1。对于不自相交或重叠的形状，应使用填充样式 0。对于重叠的形状，情况更为复杂。

例如，如果一个形状由两个重叠的正方形组成，并且只定义了填充样式 0，Flash Player 将在路径重叠的区域渲染一个“洞”。这个区域可以使用填充样式 1 进行填充。在这种情况下，规则是对于任何有向矢量，填充样式 0 是矢量左侧的颜色，填充样式 1 是矢量右侧的颜色（如下面的图中所示）。



注意：FillStyle0 和 FillStyle1 应与 FILLSTYLEARRAY 索引区分开来。FillStyle0 和 FillStyle1 是包含 FILLSTYLEARRAY 索引的变量。

边缘记录

边缘记录具有类型标志 1。边缘记录有两种类型：直线和曲线。StraightFlag 确定类型。

StraightEdgeRecord

StraightEdgeRecord 存储边缘为 X-Y 增量。增量添加到当前绘图位置，这成为新的绘图位置。边缘在旧的和新的绘图位置之间渲染。

直线边缘记录支持三种类型的线：

1. 通用线。
2. 水平线。
3. 垂直线。

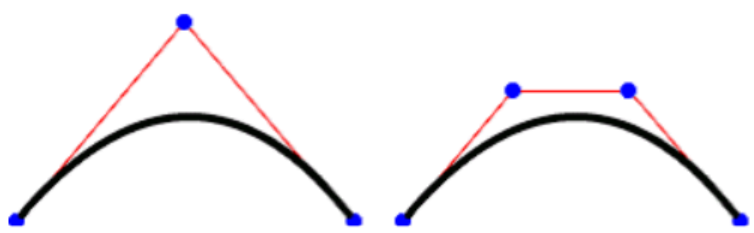
通用行存储 X 和 Y 的增量，水平和垂直行分别只存储 X 增量 Y 增量。

字段	Type	评论
类型标志	UB[1]	这是一个边缘记录。总是 1。
直线标志	UB[1]	直边。总是 1。
比特数	UB[4]	每个值的位数（比实际数值少 2 位）。
通用线标志	UB[1]	通用线等于 1。垂直/水平线等于 0。
垂直线标志	如果 GeneralLineFlag 等于 0，则 SB[1]	垂直线等于 1。水平线等于 0。
DeltaX	如果 GeneralLineFlag 等于 1 或者如果 VertLineFlag 等于 0，则 SB[NumBits+2]	X 增量
ΔY	如果 GeneralLineFlag = 1 或者如果 VertLineFlag = 1，则 SB[NumBits+2]	Y 增量。

CurvedEdgeRecord

SWF 文件格式与大多数矢量文件格式不同，因为它使用二次贝塞尔曲线而不是三次贝塞尔曲线。PostScript™使用三次贝塞尔曲线，大多数绘图应用程序也是如此。SWF 文件格式使用二次贝塞尔曲线，因为它们可以更紧凑地存储，并且可以更有效地渲染。

下图显示了二次贝塞尔曲线和三次贝塞尔曲线。



二次贝塞尔曲线有 3 个点：2 个曲线上的锚点，和 1 个曲线外的控制点。三次贝塞尔曲线有 4 个点：2 个曲线上的锚点，和 2 个曲线外的控制点。

曲边记录存储边缘为两个 X-Y 增量。定义曲线的三个点如下：

二次贝塞尔曲线的计算方法如下：

1. 第一个锚点是当前绘图位置。
2. 控制点是当前绘图位置 + 控制增量。
3. 最后一个锚点为当前绘图位置 + ControlDelta + AnchorDelta。最后一个锚点变为当前绘图位置。

字段	Type	评论
类型标志	UB[1]	这是一个边缘记录。总是 1。
直线标志	UB[1]	曲边。始终为 0。
比特数	UB[4]	每个值的位数（比实际数值少 2 位）。
控制 DeltaX	SB[NumBits+2]	X 控制点变化。
控制 Y 增量	SB[NumBits+2]	Y 控制点变化
锚点 X 增量	SB[NumBits+2]	X 锚点变化。
AnchorDeltaY	SB[NumBits+2]	Y 锚点变化。

之间转换二次贝塞尔曲线和三次贝塞尔曲线。

将二次贝塞尔曲线的单个离曲线控制点替换为两个新的离曲线控制点以形成三次贝塞尔曲线。将每个新的离曲线控制点放置在曲线上的一个锚点和原始离曲线控制点之间的直线上。新的离曲线控制点应位于曲线上的锚点和原始离曲线控制点之间距离的 2/3 处。上图中二次和三次贝塞尔曲线的图示说明了这种替换。

由于您是从三次曲线转换为二次曲线，因此三次贝塞尔曲线只能用二次贝塞尔曲线来近似。这涉及到曲线的递归细分，直到三次曲线和二次等价曲线在某个任意公差范围内匹配。

关于如何用二次贝塞尔曲线近似三次贝塞尔曲线的讨论，请参阅以下内容：

- 在 stevehollasch.com/cgindex/curves/cbezquadspline.html 将贝塞尔曲线转换为二次样条
- TrueType 参考手册，将轮廓转换为 TrueType 格式，请访问 developer.apple.com/fonts/TTRefMan/RM08/appendixE.html

形状标签

定义形状

DefineShape 标签定义了一个形状，该形状将被控制标签如 PlaceObject 等后续使用。ShapeId 唯一标识此形状在字典中的‘字符’。ShapeBounds 字段是包含形状的矩形。SHAPEWITHSTYLE 结构包括构成形状的所有路径、填充样式和线条样式。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 2。
形状 ID	UI16	该字符的 ID
形状边界	RECT	形状的边界。
形状。	SHAPEWITHSTYLE	形状信息。

定义形状 2

定义形状 2 扩展了定义形状的功能，支持在样式列表中支持超过 255 种样式，并在单个形状中支持多个样式列表。最小文件格式版本为 SWF 2。

字段	Type	评论
标题	记录头	标签类型 = 22。
形状 ID	UI16	该角色的 ID
形状边界	RECT	形状的边界
形状列表	SHAPEWITHSTYLE	形状信息。

DefineShape3

DefineShape3 扩展了 DefineShape2 的功能，通过扩展所有 RGB 颜色字段以支持带有不透明度信息的 RGBA。

最小文件格式版本为 SWF 3。

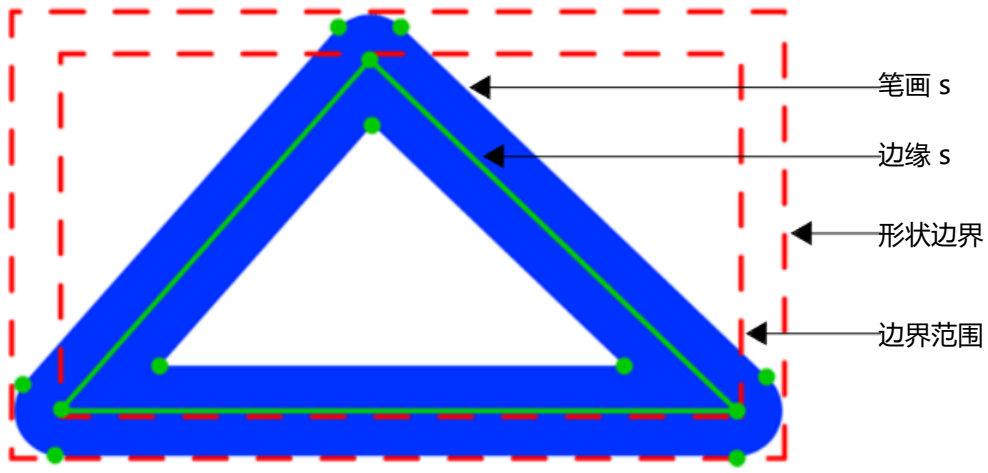
字段	Type	评论
标题	记录头	标签类型 = 32。
形状 ID	UI16	该字符的 ID
形状边界	RECT	形状的边界。
形状。	SHAPEWITHSTYLE	形状信息。

定义形状 4

定义形状 4 通过使用新的线型记录扩展了定义形状 3 的功能。线型 2

LINESTYLE2 允许新的连接和端点类型，以及缩放选项和填充描边的功能。

DefineShape4 不仅指定了形状边界，还指定了形状的边缘边界。形状边界沿着笔画的外侧计算，而边缘边界则取自边缘的外侧，如下所示。EdgeBounds 字段帮助 Flash Player 准确确定某些布局。



此外，DefineShape4 包含了新的提示标志 UsesNonScalingStrokes 和 UsesScalingStrokes。这些标志帮助 Flash Player 创建最佳的无效化区域。

最小文件格式版本为 SWF 8。

字段	Type	评论
标题	记录头	标签类型 = 83。
形状 ID	UI16	该角色的 ID
形状边界	RECT	形状的边界。
EdgeBounds	RECT	形状的边界，不包括描边。
保留	UB[5]	必须为 0。
使用填充的 Winding 规则	UB[1]	如果为 1，则使用填充的 Winding 规则。最低文件格式版本为 SWF 10
使用非缩放描边	UB[1]	如果为 1，则形状至少包含一条非缩放笔划。
UsesScalingStrokes	UB[1]	如果为 1，则形状至少包含一条缩放笔划。
形状。	SHAPEWITHSTYLE	形状信息。

第七章：渐变

渐变是 SWF 形状的特殊填充类型。它们创建颜色渐变，在两个或多个固定颜色之间进行插值。

下面是 SWF 渐变模型的概述：

- 渐变有两种样式：线性渐变和径向渐变。此外，在 SWF 8 文件格式中，新增了一种径向渐变类型，允许设置焦点。
- 每个渐变都有自己的变换矩阵，并且可以独立于其父形状进行变换。
- 在 SWF 7 文件格式及其之前版本中，渐变最多可以有八个控制点，而在 SWF 8 及其之后版本中，最多可以有十五个控制点。颜色在控制点之间进行插值，以创建颜色渐变。
- 每个控制点由一个比例和一个 RGBA 颜色定义。比例决定了控制点在渐变中的位置；RGBA 值决定了其颜色。

以下是 SWF 渐变的示例（从左到右）：

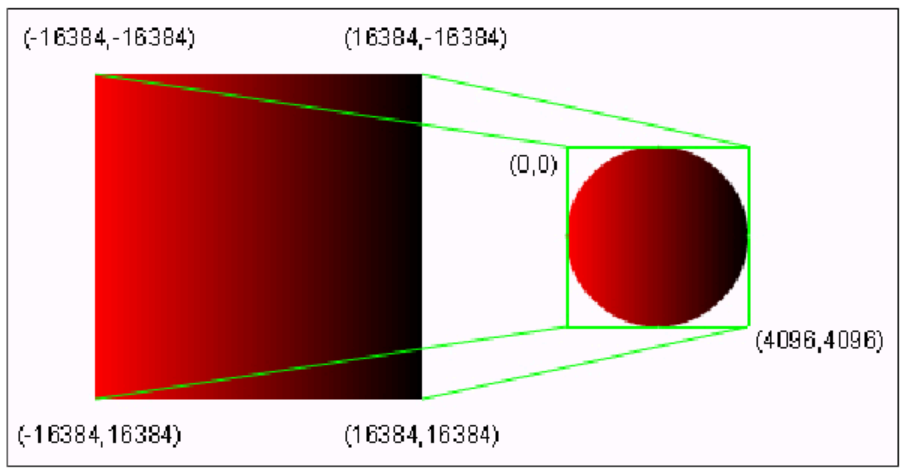
- 一个简单的白到黑线性渐变。
- 简单的白色到黑色的径向渐变。
- 由七个控制点组成的“彩虹”渐变；红、黄、绿、青、蓝、紫和红。
- 一个三点渐变，两端不透明，中心点透明。结果是 alpha 通道的渐变，允许背景中的菱形形状显示出来。

渐变变换

所有渐变都在一个称为渐变方格的标准空间中定义。渐变方格以（0，0）为中心，从（-16384，-16384）延伸到（16384，16384）。

每个渐变都使用标准变换矩阵从渐变方格映射到显示表面。
这个矩阵存储在 FILLSTYLE 结构中。

以下图中，将线性渐变映射到直径为 4096 个单位的圆上，圆心位于(2048,2048)。



该映射所需的 2x3 矩阵为：

```
|        0.125          0.000       |
|        0.000          0.125       |
| 2048.000  2048.000 |
```

梯度缩放至原始大小的八分之一（32768 / 4096 = 8），并平移至（2048，2048）。

梯度控制点

渐变中控制点的位置由 0 到 255 的比率值确定。对于线性渐变，比率为零映射到渐变方形的左侧，比率为 255 映射到右侧。对于径向渐变，比率为零映射到渐变方形的中心点，比率为 255 映射到适合渐变方形内部的最大圆。

控制点的颜色由 RGBA 值确定。alpha 值为零表示在此点渐变完全透明。alpha 值为 255 表示在此点渐变完全不透明。

控制点按比率排序，比率最小者排在前面。

渐变结构

渐变结构是 FILLSTYLE 结构的一部分。

渐变

SWF 8 及以后版本支持最多 15 个渐变控制点、扩散模式和新的插值类型。

注意，对于 DefineShape、DefineShape2 或 DefineShape3 标签，SpreadMode 和 InterpolationMode 字段必须为 0，且 NumGradients 字段不能超过 8。

字段	Type	评论

扩散模式	UB[2]	0 = 填充模式；1 = 反射模式；2 = 重复模式；3 = 保留
插值模式	UB[2]	0 = 正常 RGB 模式插值；1 = 线性 RGB 模式插值；2 和 3 = 保留
渐变数量	UB[4]	1 至 15
渐变记录	GRADRECORD[nGrads]	渐变记录（见下文）

焦点梯度

在 DefineShape4 中必须声明 FOCALGRADIENT，而不是 DefineShape、DefineShape2 或 DefineShape3。

值的范围是-1.0 到 1.0，其中-1.0 表示焦点接近径向渐变圆的左侧边界，0.0 表示焦点位于径向渐变圆的中心，1.0 表示焦点接近径向渐变圆的右侧边界。

字段	Type	评论
扫描模式	UB[2]	0 = 填充模式；1 = 反射模式；2 = 重复模式；3 = 保留
插值模式	UB[2]	0 = 正常 RGB 模式插值；1 = 线性 RGB 模式插值；2 和 3 = 保留
渐变数量	UB[4]	1 至 15
渐变记录	GRADRECORD[nGrads]	渐变记录（见下文）
焦点	FIXED8	焦点位置

GRADRECORD

GRADRECORD 定义了一个渐变控制点：

字段	Type	评论
比率	UI8	比率值
颜色	RGB（形状 1 或形状 2）RGBA（形状 3）	渐变颜色

第八章：位图

SWF 文件格式规范支持多种位图格式。所有位图都经过压缩以减小文件大小。适用于不精确图像（如照片）的损失压缩由 JPEG 位图提供；适用于精确图像（如图表、图标或屏幕截图）的无损压缩由 ZLIB 位图提供。这两种类型的位图都可以选择性地包含 alpha 通道（不透明度）信息。

JPEG 格式正式定义为 ITU T.81 或 ISO/IEC 10918-1，是由独立联合图像专家组开发的开源标准。本文件未描述 JPEG 格式。有关 JPEG 格式的通用信息，请参阅 www.jpeg.org 上的 JPEG。有关 JPEG 格式的规范，请访问国际电信联盟 www.itu.int/，并搜索推荐 T.81。SWF 文件中的 JPEG 数据使用附录 B 中指定的 JPEG 交换格式进行编码。Flash Player 还支持流行的 JFIF 格式，它是 JPEG 交换格式的扩展。

在所有将非 JPEG 像素数据数组存储在位图标签中的情况下，像素以行主序排列，类似于英文文本，在行内从左到右，整体从上到下。

定义位图

此标签定义了一个使用 JPEG 压缩的位图字符。它仅包含从帧标题开始的 JPEG 压缩图像数据。一个单独的 JPEGTables 标签包含用于编码此图像的 JPEG 编码数据（表/杂项段）。

注意：SWF 文件中只允许一个 JPEGTables 标签，因此使用 DefineBits 定义的所有位图必须共享相同的编码表。

该标签中的数据以 JPEG SOI 标记 0xFF, 0xD8 开始，以 EOI 标记 0xFF, 0xD9 结束。在 SWF 文件格式版本 8 之前，SWF 文件可能包含一个错误的头部 0xFF, 0xD9, 0xFF, 0xD8，位于 JPEG SOI 标记之前。

该标签的最小文件格式版本为 SWF 1。

字段	Type	评论
标题	RECORDHEADER（长）	标签类型 = 6
角色 ID	UI16	该字符的 ID
JPEG 数据	UI8[图像数据大小]	JPEG 压缩图像

JPEG 表格

此标签定义了使用 JPEG 图像定义的 JPEG 编码表（Tables/Misc 段）。

定义位图标签。一个 SWF 文件中只能有一个 JPEGTables 标签。

该标签中的数据以 JPEG SOI 标记 0xFF, 0xD8 开始，以 EOI 标记 0xFF, 0xD9 结束。在 SWF 文件格式版本 8 之前，SWF 文件可能包含一个错误的头部 0xFF, 0xD9, 0xFF, 0xD8，位于 JPEG SOI 标记之前。

该标签的最小文件格式版本为 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 8
JPEG 数据	UI8[编码数据大小]	JPEG 编码表

定义位 JPEG2

此标签定义了一个使用 JPEG 压缩的位图字符。它与 DefineBits 不同，因为它包含 JPEG 编码表和 JPEG 图像数据。此标签允许在单个 SWF 文件中定义具有不同编码表的多个 JPEG 图像。

该标签中的数据以 JPEG SOI 标记 0xFF, 0xD8 开始，以 EOI 标记 0xFF, 0xD9 结束。在 SWF 文件格式版本 8 之前，SWF 文件可能包含一个错误的头部 0xFF, 0xD9, 0xFF, 0xD8，位于 JPEG SOI 标记之前。

除了指定 JPEG 数据外，DefineBitsJPEG2 还可以包含 PNG 图像数据和 GIF89a 非动画图像数据。

- 如果 ImageData 以 0x89 0x50 0x4E 0x47 0x0D 0x0A 0x1A 0x0A 这八个字节开头，则 ImageData 包含 PNG 数据。
- 如果 ImageData 以 0x47 0x49 0x46 0x38 0x39 0x61 这六个字节开头，则 ImageData 包含 GIF89a 数据。

此标签的最小文件格式版本为 SWF 2。嵌入 PNG 或 GIF89a 数据的最小文件格式版本为 SWF 8。

字段	Type	评论
标题	RECORDHEADER（长）	标签类型 = 21
角色 ID	UI16	该字符的 ID
图像数据	UI8[数据大小]	压缩图像数据，格式为 JPEG、PNG 或 GIF89a

定义位 JPEG3

GIF89a 格式此标签定义了一个具有 JPEG 压缩的位图字符。此标签扩展了 DefineBitsJPEG2，增加了 alpha 通道（不透明度）数据。不透明度/透明度信息不是 JPEG 图像的标准功能，因此 alpha 通道信息与 JPEG 数据分开编码，并使用 ZLIB 标准进行压缩。ZLIB 库所使用的数据格式由请求评论（RFC）文档 1950 至 1952 描述。

该标签中的数据以 JPEG SOI 标记 0xFF, 0xD8 开始，以 EOI 标记 0xFF, 0xD9 结束。在 SWF 文件格式版本 8 之前，SWF 文件可能包含一个错误的头部 0xFF, 0xD9, 0xFF, 0xD8，位于 JPEG SOI 标记之前。

除了指定 JPEG 数据外，DefineBitsJPEG2 还可以包含 PNG 图像数据和 GIF89a 非动画图像数据。

- 如果 ImageData 以 0x89 0x50 0x4E 0x47 0x0D 0x0A 0x1A 0x0A 这八个字节开头，则 ImageData 包含 PNG 数据。
- 如果 ImageData 以 0x47 0x49 0x46 0x38 0x39 0x61 这六个字节开头，则 ImageData 包含 GIF89a 数据。

如果 ImageData 包含 PNG 或 GIF89a 数据，则不支持的 BitmapAlphaData 是可选的。

此标签的最小文件格式版本为 SWF 3。嵌入 PNG 或 GIF89a 数据的最小文件格式版本为 SWF 8。

字段	Type	评论
标题	RECORDHEADER（长）	标签类型 = 35。
角色 ID	UI16	该角色的 ID
Alpha 数据偏移量	UI32	ImageData 中字节数的计数
图像数据	UI8[数据大小]	压缩图像数据，格式为 JPEG、PNG 或 GIF89a
位图 Alpha 数据	UI8[Alpha 数据大小]	ZLIB 压缩的 alpha 数据数组。仅在标签包含 JPEG 数据时支持。每像素一个字节。解压缩后的总大小必须等于 JPEG 图像的（宽度 * 高度）。

定义无损位图字符

定义一个包含使用 ZLIB 压缩的 RGB 位图数据的无损位图字符。ZLIB 库所使用的数据格式由请求评论（RFC）文档 1950 至 1952 描述。

两种位图被支持。彩色映射图像定义了一个最多包含 256 种颜色的调色板，每种颜色由一个 24 位的 RGB 值表示，然后使用 8 位像素值来索引调色板。直接图像使用 15 位（32,768 种颜色）或 24 位（约 1700 万种颜色）存储实际的像素颜色值。

该标签的最小文件格式版本为 SWF 2。

字段	Type	评论
标题	RECORDHEADER（长）	标签类型 = 20
角色 ID	UI16	该字符的 ID
位图格式	UI8	压缩数据格式： 3 = 8 位调色板图像 4 = 15 位 RGB 图像 5 = 24 位 RGB 图像
位图宽度	UI16	位图图像宽度
位图高度	UI16	位图图像高度
位图颜色表大小	如果位图格式等于 3 UI8；否则不存在	此值比实际颜色表中的颜色数少一， 允许最多 256 种颜色。
Zlib 位图数据	如果位图格式等于 3，则 COLORMAPDATA 如果位图格式等于 4 或 5，则 BITMAPDATA	ZLIB 压缩的位图数据

COLORMAPDATA 和 BITMAPDATA 结构包含图像数据。这些结构各自压缩为单个数据块。它们的压缩前布局如下。

注意：这些结构中的像素数据字段行宽必须向上取整到下一个 32 位字边界。例如，宽度为 253 像素的 8 位图像必须每行填充到 256 字节。为了确定适当的填充，请确保考虑单个像素结构的大小；15 位像素占用 2 字节，24 位像素占用 4 字节（参见 PIX15 和 PIX24）。

COLORMAPDATA		
字段	Type	评论
ColorTableRGB	RGB[颜色表大小]	定义颜色索引到 RGB 值的映射。 RGB 值的数量是 BitmapColorTableSize

		+ 1.
RGB 值数量是 BitmapColorTableSize + 1	UI8[图像数据大小]	颜色索引数组。条目数量为 BitmapWidth * BitmapHeight，受填充限制（参见此表之前的说明）。

BITMAPDATA		
字段	Type	评论
位图像素数据	如果 BitmapFormat = 4，则 PIX15[图像数据大小] 如果 BitmapFormat = 5，则 PIX24[图像数据大小]	像素颜色数组。条目数量为 BitmapWidth * BitmapHeight，受上述注释中的填充限制。

PIX15		
字段	Type	评论
Pix15 保留	UB[1]	总是 0
Pix15 红色	UB[5]	红色值
Pix15 绿色	UB[5]	绿色值
Pix15 蓝色	UB[5]	蓝色值

PIX24		
字段	Type	评论
Pix24 保留	UI8	总是 0
Pix24 红色	UI8	红色值
Pix24 绿	UI8	绿色值
Pix24 蓝	UI8	蓝色值

定义无损位块

DefineBitsLossless2 扩展了 DefineBitsLossless，并支持不透明度（alpha 值）。在彩色映射图像中，调色板颜色使用 RGBA 值定义，直接图像存储每个像素的 32 位 ARGB 颜色。

DefineBitsLossless2 中不可用 15 位中间颜色深度。

此标签的最小文件格式版本为 SWF 3。

字段	Type	评论
标题	RECORDHEADER（长）	标签类型 = 36
角色 ID	UI16	该字符的 ID
位图格式	UI8	压缩数据格式 3 = 8 位调色板图像 5 = 32 位 ARGB 图像
位图宽度	UI16	位图图像宽度
位图高度	UI16	位图图像高度
位图颜色表大小	如果 BitmapFormat = 3，则 UI8；否则不存在	此值比实际颜色表中的颜色数少一，允许最多 256 种颜色。
Zlib 位图数据	如果 BitmapFormat = 3，则 ALPHACOLORMAPDATA 如果位图格式为 4 或 5，则 ALPHABITMAPDATA	ZLIB 压缩的位图数据

COLORMAPDATA 和 BITMAPDATA 结构包含图像数据。这些结构各自压缩为单个数据块。它们的压缩前布局如下。

注意：ALPHACOLORMAPDATA 像素数据字段中的行宽度必须向上舍入到下一个 32 位字边界。例如，宽度为 253 像素的 8 位图像必须填充到每行 256 字节。ALPHABITMAPDATA 中的行宽度始终是 32 位对齐的，因为 ARGB 结构是 4 字节。

ALPHACOLORMAPDATA		
字段	Type	评论
ColorTableRGB	RGBA[颜色表大小]	定义颜色索引到 RGBA 值的映射。 RGBA 值的数量为 BitmapColorTableSize + 1。

颜色映射像素数据	UI8[图像数据大小]	颜色索引数组。条目数量为 BitmapWidth * BitmapHeight，受填充限制（参见此表之前的说明）。

字符映射数据		
字段	hType	评论
位图像素数据	ARGB[图像数据大小]	像素颜色数组。条目数量为 BitmapWidth * BitmapHeight。RGB 数据必须已经乘以 alpha 通道值。

定义位图 JPEG4

该标签定义了一个使用 JPEG 压缩的位图字符。该标签扩展了 DefineBitsJPEG3，增加了去块滤波参数。虽然此标签也支持 PNG 和 GIF89a 数据，但去块滤波器不应用于此类数据。

此标签的最小文件格式版本为 SWF 10。

字段	Type	评论
标题	RECORDHEADER（长）	标签类型 = 90。
角色 ID	UI16	该角色的 ID
Alpha 数据偏移量	UI32	ImageData 中字节数的计数
去块参数	UI16	要输入到去块滤波器的参数。该参数描述了去块滤波器的相对强度，范围为 0-100%，以归一化的 8.8 定点格式表示。
图像数据	UI8[数据大小]	以 JPEG、PNG 或 GIF89a 格式压缩的图像数据。
位图 Alpha 数据	UI8[Alpha 数据大小]	ZLIB 压缩的 alpha 数据数组。仅在标签包含 JPEG 数据时支持。每像素一个字节。解压缩后的总大小必须等于 JPEG 图像的（宽度 * 高度）。

第九章：形状变形

形状变形是指一个形状随时间变化而变成另一个形状。SWF 文件格式规范支持灵活的变形模型，允许在变形过程中改变多个形状属性。SWF 文件格式仅定义了变形的起始和结束状态。Adobe Flash Player 负责在端点之间进行插值并生成“中间”状态。

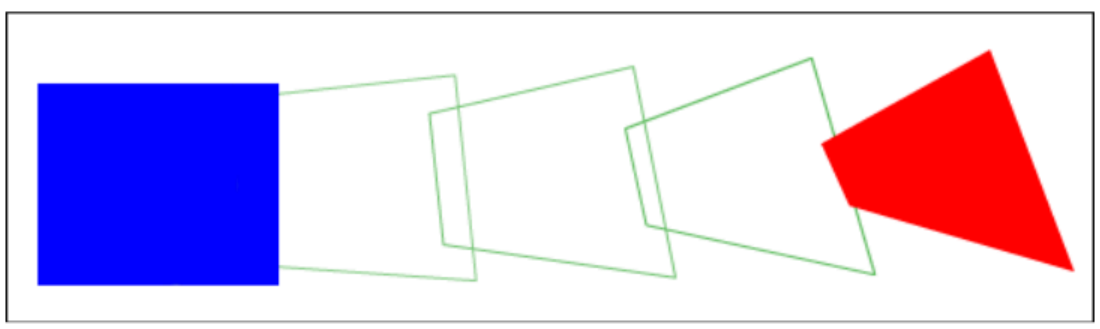
以下形状属性可以在变形过程中改变：

- 形状中每条边的位置。
- 轮廓的颜色和粗细。
- 形状的填充颜色（如果用颜色填充）。
- 位图变换（如果用位图填充）。
- 渐变填充的变换（如果用渐变填充）。
- 渐变中每个点的颜色和位置（如果用渐变填充）。

以下限制适用于变形：

- 起始形状和结束形状必须具有相同数量的边。
- 起始形状和结束形状必须具有相同的填充类型（即，纯色、渐变或位图）。
- 样式更改记录必须对起始形状和结束形状相同。
- 如果填充了位图，两个形状必须填充相同的位图。
- 如果填充了渐变，两个渐变必须具有相同数量的颜色点。

以下插图显示了蓝色矩形到红色四边形的变形过程，跨越五帧。绿色轮廓代表变形序列的“中间”形状。两个形状具有相同数量的点，并且具有相同类型的填充，即实心填充。除了改变形状外，形状的颜色也逐步从蓝色变为红色。



涉及定义和播放形态序列的两个标签：

- 定义形态形状
- PlaceObject2

DefineMorphShape 定义了形态的起始和结束状态。形态对象不使用先前定义的形状；它被视为一种特殊类型的形状，只有一个字符 ID。DefineMorphShape 包含起始和结束形状的边列表。它还定义了在形态序列开始和结束时的填充和线条样式。

PlaceObject 2 标签在形态序列中的某个时间点显示形态对象。比率字段控制形态的进度。比率为零产生与起始条件相同的形状。比率为 65535 产生与结束条件相同的形状。

定义形态形状

DefineMorphShape 标签定义了变形序列的起始和结束状态。变形对象应使用 PlaceObject2 标签显示，其中比例字段指定变形的进度。

最小文件格式版本为 SWF 3。

字段	Type	评论
标题	记录头	标签类型 = 46
字符 ID	UI16	该字符的 ID
起始边界	RECT	起始形状的边界
终止边界	RECT	终止形状的边界
偏移量	UI32	表示到终止边的偏移量
形状填充样式	MORPHFILLSTYLEARRAY	填充样式信息存储方式与标准形状相同；然而，每个填充都包含基于单一样式类型的交错信息，以适应变形。
形状线条样式	形态线样式数组	线条样式信息存储方式与标准形状相同；然而，每条线条都包含基于单一样式类型的交错信息，以适应变形。
起始边	形状	包含一组边和表示样式变化的样式位（例如，移动到、填充样式和线条样式）。编号

		边的数量必须等于 EndEdges 中的边的数量。
终止边	形状	仅包含边集，没有样式信息。 边的数量必须等于 StartEdges 中的边数。

- 开始边界框。这定义了形状在变形开始时的边界框。
- 形态结束边界 - 这定义了形态末端的边界框。
- 形态填充样式 - 这包含起始和结束形状的交错填充样式数组。起始形状的填充样式后面跟着对应结束形状的填充样式。
- 形态线条样式 - 这包含交错线条样式数组。
- 起始边缘 - 此数组指定起始形状的边缘以及两个形状的样式更改记录。由于样式更改记录对于起始和结束形状必须相同，因此它们仅在起始边缘数组中定义。
- EndEdges - 此数组指定结束形状的边，不包含样式更改记录。StartEdges 中指定的边的数量必须等于 EndEdges 中的边的数量。

数量 严格来说，MoveTo 记录属于 StyleChangeRecords 类别；然而，它们应包含在 StartEdges 和 EndEdges 数组中。

在形态序列过程中，边的类型可能会改变。一条直线边可以变成曲线边，反之亦然。在这种情况下，将两条边都视为曲线。直线边可以通过将离曲线点（控制点）放置在直线边的中点，并将曲线点（锚点）放置在直线边的末端来轻松表示为曲线。计算如下：

```
CurveControlDelta.x = StraightDelta.x / 2;
CurveControlDelta.y = StraightDelta.y / 2;
CurveAnchorDelta.x = StraightDelta.x / 2;
CurveAnchorDelta.y = StraightDelta.y / 2;
```

定义形态形状 2

DefineMorphShape2 标签通过使用新的形态线条样式记录扩展了 DefineMorphShape 的功能。MORPHLINESTYLE2 允许使用新的连接和端点类型以及缩放选项，以及填充形态形状的线条的能力。

DefineMorphShape2 不仅指定了形状边界，还指定了形状的边缘边界。虽然形状边界是沿着线条的外侧计算的，但边缘边界是从边缘的外侧获取的。有关形状边界与边缘边界的示例，请参阅 DefineShape4 中的图表。新的 StartEdgeBounds 和 EndEdgeBounds 字段帮助 Flash Player 准确确定某些布局。

此外，DefineMorphShape2 包含了新的提示信息，UsesNonScalingStrokes 和 UsesScalingStrokes。这些标志帮助 Flash Player 创建最佳的无效化区域。

最小文件格式版本为 SWF 8。

字段	Type	评论
标题	记录头	标签类型 = 84
字符 ID	UI16	该字符的 ID
起始边界	RECT	起始形状的边界
终止边界	RECT	终止形状的边界
起始边框界限	RECT	起始形状的界限，不包括描边
结束边框界限	RECT	终止形状的边界，不包括描边
保留	UB[6]	必须为 0
使用非缩放描边	UB[1]	如果为 1，则形状至少包含一条非缩放笔划。
UsesScalingStrokes	UB[1]	如果为 1，则形状至少包含一条缩放笔划。
偏移量	UI32	表示到终止边的偏移量
形状填充样式	MORPHFILLSTYLEARRAY	填充样式信息存储方式与标准形状相同；然而，每个填充都包含基于单一样式类型的交错信息，以适应变形。
形状线条样式	形态线样式数组	线条样式信息存储方式与标准形状相同；然而，每条线条都包含基于单一样式类型的交错信息，以适应变形。
起始边	形状	包含一组边和指示样式变化的样式位（例如，移动到、填充样式和线条样式）。边的数量必须等于 EndEdges 中的边的数量。
终止边	形状	仅包含一组边，没有样式信息。边的数量必须等于 StartEdges 中的边的数量。

形状填充样式

MORPHFILLSTYLEARRAY

形状填充样式数组列举了多个填充样式。

字段	Type	评论
填充样式数量	计数 = UI8	填充样式的数量。
FillStyleCountExtended	如果计数 = 0xFF UI16	填充样式的扩展计数。
填充样式	MORPHFILLSTYLE[count]	填充样式数组

MORPHFILLSTYLE

填充样式表示闭合形状的填充方式。

字段	Type	评论
填充样式类型	UI8	填充样式类型： 0x00 = 实心填充 0x10 = 线性渐变填充 0x12 = 径向渐变填充 0x13 = 焦点径向渐变填充（仅限 SWF 8 文件格式及以后版本） 0x40 = 重复位图 0x41 = 修剪位图填充 0x42 = 非平滑重复位图 0x43 = 非平滑裁剪位图
开始颜色	如果类型 = 0x00，RGBA	起始形状的实心填充颜色及透明度信息
结束颜色	如果类型 = 0x00，RGBA	结束形状的透明度信息实心填充颜色。
开始渐变矩阵	如果类型为 0x10 或 0x12，则使用 MATRIX	渐变填充的开始形状。

结束渐变矩阵	如果类型为 0x10 或 0x12，则使用 MORPHGRADIENT。	渐变填充的结束形状矩阵。
渐变	如果类型为 0x10 或 0x12，则 MORPHGRADIENT。	渐变填充
位图 ID	如果类型为 0x40、0x41、0x42 或 0x43，则 UI16	位图填充字符的 ID。
开始位图矩阵。	如果类型为 0x40、0x41、0x42 或 0x43，则 MATRIX	位图填充的开始形状矩阵。
结束位图矩阵	如果类型为 0x40、0x41、0x42 或 0x43，则 MATRIX	结束形状的位图填充矩阵。

形变渐变值

形变渐变值控制填充样式的渐变信息。

形态梯度

梯度信息的格式在以下表格中描述：

字段	Type	评论
渐变数量	UI8	1 至 8
渐变记录	形态梯度记录[梯度数量]	渐变记录（见下文）。

MORPHGRADRECORD

渐变记录格式在以下表格中描述：

字段	Type	评论
开始比例	UI8	起始形状的比率值。
开始颜色	RGBA	起始形状的渐变颜色。
结束比率。	UI8	结束形状的比率值。
结束颜色	RGBA	渐变颜色用于结束形状。

形态线条样式

形状线样式数组列举了多种填充样式。

MORPHLINESTYLEARRAY

下表描述了线样式数组的格式。

字段	Type	评论
线样式数量	UI8	线样式的数量。
线样式数量扩展	如果 count = 0xFF UI16	线样式的扩展计数。
线样式	MORPHLINESTYLE[count]，（如果 MorphShape1） MORPHLINESTYLE2[count]，（如果 MorphShape2）	线样式的数组。

线样式表示线的宽度和颜色。

MORPHLINESTYLE

文件中线条样式值的格式在以下表格中描述。

字段	Type	评论
起始宽度	UI16	起始形状中线条的宽度，单位为 twips。
结束宽度	UI16	结束形状中线条的宽度，单位为 twips。
开始颜色	RGBA	起始形状的包含 alpha 通道信息的颜色值。
结束颜色	RGBA	结束形状的包含 alpha 通道信息的颜色值。

MORPHLINESTYLE2

MORPHLINESTYLE2 在 MORPHLINESTYLE 记录的基础上增加了新类型的连接和端点样式，以及缩放选项和填充形态线条的能力。为了使用 MORPHLINESTYLE2，形状必须使用 DefineMorphShape2 定义，而不是 DefineMorphShape。

虽然 MORPHLINESTYLE 记录只允许圆角连接和圆形端头，但 MORPHLINESTYLE2 还支持斜接和斜角连接，以及方形端头和无端头。有关可用的连接和端头的示例，请参阅 LINESTYLE2 描述中的图示。

当使用 MORPHLINESTYLE 进行斜接连接时，必须指定 MiterLimitFactor，并将其与

一个斜接限制系数必须指定，并且与起始宽度或结束宽度一起用于计算最大斜接长度：

最大斜接长度 = MORPHLINESTYLE2 MiterLimitFactor * MORPHLINESTYLE2

宽度

如果斜接超过最大斜接长度，Flash Player 将截断斜接。注意，MiterLimitFactor 是一个 8.8 位定点值。

MORPHLINESTYLE2 还包括像素提示选项，以纠正模糊的垂直或水平线。

字段	Type	评论
起始宽度	UI16	起始形状中线条的宽度，单位为 twips。
结束宽度	UI16	结束形状中线条的宽度，单位为 twips。
StartCapStyle	UB[2]	起始端点样式：；0 = 圆形端点；1 = 无端点；2 = 平方端点
JoinStyle	UB[2]	连接样式：；0 = 圆角连接；1 = 斜接连接；2 = 斜接连接
有填充标志	UB[1]	如果为 1，填充定义在 FillType 中。如果为 0，则使用 StartColor 和 EndColor 字段。
无水平缩放标志	UB[1]	如果为 1，则对象水平缩放时，描边粗细不会缩放。
NoVScaleFlag	UB[1]	如果设置为 1，当对象垂直缩放时，笔触粗细将不会缩放。
PixelHintingFlag	UB[1]	如果设置为 1，所有锚点都将对齐到完整像素。
保留	UB[5]	必须为 0。
NoClose	UB[1]	如果为 1，当笔画的最后一个点与第一个点相匹配时，笔画不会闭合。Flash Player 将应用端点而不是连接。
EndCapStyle	UB[2]	封头样式：0 = 圆顶；1 = 无顶；2 = 方顶
斜接限制系数	如果 JoinStyle = 2，UI16	斜接限制系数作为一个 8.8 的定点值。
开始颜色	如果 HasFillFlag = 0，RGBA	起始形状的包含 alpha 通道信息的颜色值。
结束颜色	如果 HasFillFlag = 0，RGBA	结束形状的包含 alpha 通道信息的颜色值。
填充类型	如果 HasFillFlag = 1，MORPHFILLSTYLE	填充样式。

第 10 章：字体与文本

SWF 文件格式规范支持多种文本绘制原语。在 SWF 6 或更高版本的文件中，所有文本都使用 Unicode 编码表示，消除了对播放区域文本和字符串的依赖。自第 10 版起，Flash Player 还支持从右到左的脚本，以及希伯来语、阿拉伯语、泰语和其他复杂脚本的支持。

符号文本和设备文本

SWF 文件格式支持两种文本：矢量文本和设备文本。矢量文本通过在 SWF 文件中嵌入字符形状来工作，而设备文本则使用播放平台的文本渲染功能。

矢量文本在所有播放平台上看起来完全相同。它可以使用 Flash Player Stage 上所有形状使用的标准抗锯齿来绘制，或者在 SWF 8 文件格式及以后版本中使用高级文本渲染引擎进行渲染。使用矢量文本比设备文本创建的 SWF 文件更大，尤其是在使用来自大型字符集的许多不同字符的文件中。

设备文本由承载 Flash Player 的操作系统进行抗锯齿处理，其外观取决于播放平台。设备文本的字体可以通过两种方式指定：直接指定，即作为将在播放平台上逐字查找的字体名称；或者间接指定，使用少数几个特殊字体名称之一，这些名称映射到名称在不同平台上有所不同但尽可能在平台上外观相似的可用字体。

使用 DefineFont、DefineFont2 或 DefineFont3 标签定义符号文本字符。设备文本字体使用 DefineFont 和 DefineFontInfo 标签一起定义，或使用 DefineFont2 标签。如果设备文本字体仅用于动态文本，则不需要包含任何字符符号（见下一节），尽管如果对指定字体在任何平台上播放时是否可用有任何疑问，最好还是包含它们。只要提供了符号和字符代码，就可以将给定的 DefineFont 或 DefineFont2 标签用作某些文本块的符号字体，以及其他文本块的设备字体。

静态文本和动态文本

文本可以定义为静态文本，或者在 SWF 4 文件格式或更高版本中，可以定义为动态文本。动态文本可以在运行时程序化地更改，并且可选地可以供 Adobe Flash Player 用户编辑。

动态文本可以模拟几乎所有的静态文本功能；除了实现努力和版本兼容性之外，精确定位单个字符是静态文本的唯一优势。动态文本还具有静态文本不具备的许多格式化功能。这些丰富的格式化功能以 HTML 文本标记标签的子集的形式表达。

静态文本使用 DefineText 标签定义。动态文本使用 DefineEditText 标签定义。这两个标签都引用 DefineFont 或 DefineFont2 标签以获取它们的字符源。DefineEditText

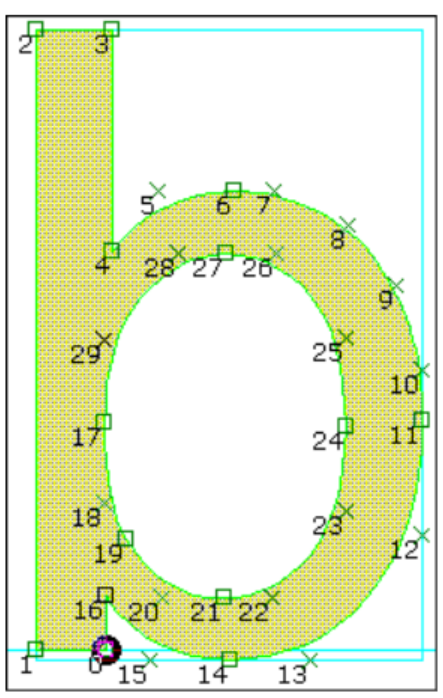
定义 EditText 标签需要使用 DefineFont2 标签而不是 DefineFont 标签；DefineText 标签可以使用 DefineFont 或 DefineFont2 标签。

DefineEditText 标签提供了一个标志，表示是否使用符号文本或设备文本。然而，DefineText 标签没有。这意味着对于静态文本，SWF 文件格式没有提供指示是否使用符号文本或设备文本的方法。这种情况通过运行时标志得到解决。通常，所有静态文本都渲染为符号文本。当 Flash Player 插件嵌入到 HTML 页面中时，一个名为 devicefont 的 HTML 标签选项将导致 Flash Player 尽可能地将所有静态文本渲染为设备文本；通常情况下，使用符号文本作为后备。DefineEditText 标签能够指定符号文本或设备文本，这也是动态文本优于静态文本的另一个原因。

符号文本

符号定义

符号在称为 EM 方格的标准坐标系中定义一次。给定字体的每个点大小都使用相同的符号集。为了渲染不同点大小的符号，Flash Player 将符号从 EM 坐标缩放到点大小坐标。



- 字体符号—不使用高级文本渲染引擎—不包含任何用于改善小字体质量的提示信息。然而，抗锯齿可以显著提高缩小文本的可读性。字体符号在约 12 点（以 100%查看）时仍保持可读。在 12 点以下，建议使用高级抗锯齿来提高字体符号的可读性。这在小字号下提供更优的文本质量，并包括额外的字体元信息以改善渲染。

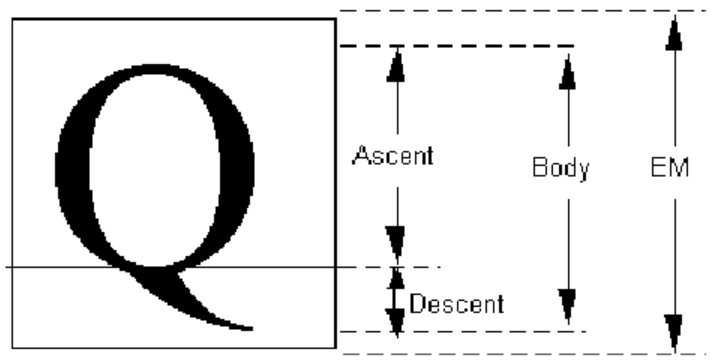
- 真型字体可以轻松转换为 SWF 符号。一个简单的算法可以将真型字体使用的二次 B 样条曲线替换为 SWF 符号使用的二次贝塞尔曲线。

示例：

左侧是 Monotype Arial 字体中 TrueType 字母'b'的符号。它由曲线和直线边缘组成。正方形表示曲线上的点，而十字表示曲线外的点。黑色圆圈是符号的参考点。蓝色轮廓表示符号的边界框。

EM 平方

EM 平方是一个用于调整和定位字符的假想正方形。EM 平方通常足够大，可以完全包含所有字符，包括带重音的字符。它包括字体的上升、下降以及一些额外的间距，以防止文本行相互碰撞。



SWF 字符始终定义在 1024x1024 单位的 EM 平方上。来自其他来源（如 TrueType 字体）的字符可能定义在不同的 EM 平方上。要在 SWF 文件格式中使用这些字符，它们应该缩放到适合 1024x1024 单位的 EM 平方。

将 TrueType 字体转换为 SWF 字符

真型字体轮廓使用二次 B 样条定义，这些样条可以轻松转换为 SWF 轮廓使用的二次贝塞尔曲线。

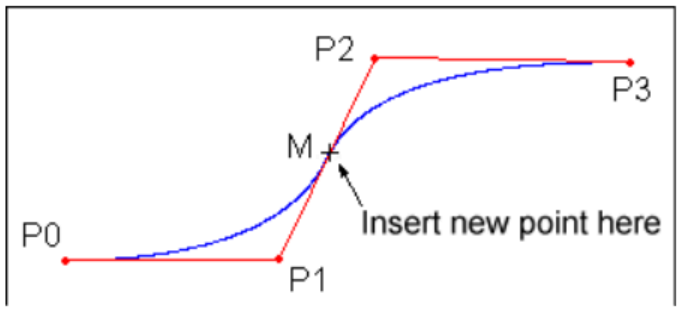
SWF 轮廓使用的二次贝塞尔曲线。

真型 B 样条由一个曲线上的点开始，接着是多个非曲线上的点，然后是另一个曲线上的点。任何两个非曲线点之间的中点都保证在曲线上。SWF 贝塞尔曲线由一个曲线上的点开始，接着是一个非曲线上的点，然后是另一个曲线上的点。

从真型到 SWF 曲线的转换涉及在两个连续的非曲线点之间插入一个新的曲线上的点。

示例：

以下是一个四点 B 样条。P0 和 P3 是曲线上的点。P1 和 P2 是连续的离曲线点。



该曲线可以通过在 P1 和 P2 的中点插入一个新点 M，表示为两个二次贝塞尔曲线。结果是两个二次贝塞尔曲线；P0,P1,M 和 M,P2,P3。

将 TrueType 字形转换为 SWF 字形的完整过程如下：

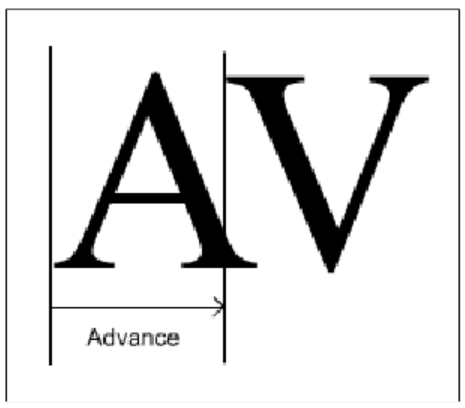
1. 取消 y 坐标的值。（在 TrueType 字体中，y 轴向上；在 SWF 字体中，y 轴向下。）

2. 将 x 和 y 坐标从 TrueType 字体的 EM 方框缩放到 SWF 字体的 EM 方框（总是 1024）。

3. 在每对离曲线点之间插入一个曲线（锚点）点。

字距和进位值

轻量级间距定义了两个字符之间的水平距离。一些字体系统将间距信息存储在每个字体定义中。与 SWF 文件格式相比，它将间距信息存储在每个字符实例中（每个字符文本块中的每个字符）。这被称为进位值。



在前面的示例中，A 字符与 V 字符重叠。在这种情况下，进位宽度比 A 字符的宽度窄。

高级文本渲染引擎

字符文本可以使用正常的 Flash Player 渲染器渲染，或者在 SWF 8 及以后版本中使用高级文本渲染引擎。

高级文本渲染引擎是 Flash Player 渲染器内支持的高质量文本渲染器。与使用普通渲染器进行文本渲染相比，高级系统具有以下优势：

- 即使在小字号下也易于阅读。
- 即使在小字号下也能保持字体的美学外观和感觉。
- 支持像素对齐，实现超清晰文本（当使用左对齐的动态文本时）。
- 与字形文本相比，通常性能有所提升。
- 当 Flash Player 检测到 LCD 屏幕时，进行 LCD 子像素渲染。

然而，高级文本渲染引擎的一个局限性是，与字形文本相比，它动画效果不佳。

高级文本渲染引擎使用连续笔触调制（CSM）参数来调整其性能。CSM 是笔触重量和边缘锐度的连续调制。CSM 使用两个渲染参数：内部和外部截止值。这些参数的最佳值非常主观，可能取决于用户偏好、照明条件、显示属性、字体、前景和背景颜色以及点大小。然而，在大多数情况下，可以通过一组小的插值值实现高质量的字体。

创建高级抗锯齿边界的函数具有外部截止（在此以下不绘制边界）和内部截止（在此以上边界不透明）。在这两个截止值之间是一个线性函数，从外部截止的零值到内部截止的最大值。

调整外部和内部截止值会影响笔划粗细和边缘锐度。这两个参数之间的间距与经典方法的过滤器半径的两倍相当：间距窄提供更锐利的边缘，而间距宽则提供更柔和、更过滤的边缘。当间距为零时，生成的密度图像是双色调位图。当间距非常宽时，生成的密度图像具有类似水彩的边缘。通常，用户在小点尺寸时更喜欢锐利、高对比度的边缘，而在动画文本和较大点尺寸时则更喜欢柔和的边缘。

抗锯齿方法的间距：间距窄提供更锐利的边缘，而间距宽则提供更柔和、更过滤的边缘。当间距为零时，生成的密度图像是双色调位图。当间距非常宽时，生成的密度图像具有类似水彩的边缘。通常，用户在小点尺寸时更喜欢锐利、高对比度的边缘，而在动画文本和较大点尺寸时则更喜欢柔和的边缘。

外部截止值通常为负值，内部截止值通常为正值，它们的中间值通常接近零。调整这些参数以将中间值移向负无穷大可以增加笔划粗细；将中间值移向正无穷大可以减小笔划粗细。请注意，外部截止值应始终小于或等于内部截止值。

Flash Player 为每个使用的先进抗锯齿字体创建一个 CSM 参数表，该表是文本大小和文本颜色的函数。此默认表通常为各种点大小提供合适的 CSM 设置集。但是，您可以使用 ActionScript 函数 setAdvancedAntialiasingTable()指定一个用户定义的表来替换默认表。

CSM 参数旨在使字体更易于阅读，而不是创建效果。极端的 CSM 值会导致渲染伪影。要应用文本效果，使用合理的 CSM 值然后应用过滤器或混合效果会更好。

定义字体和定义文本

SWF 文件格式支持的四种文本类型（静态符号、静态设备、动态符号和动态设备）中，最复杂的是静态符号文本。其他类型都是对定义静态符号文本的规则的简化变体。

静态符号文本使用两个标签进行定义：

- DefineFont 标签定义了一组符号。
- DefineText 标签定义了在字体中显示的文本字符串。

定义字体标签定义了后续定义文本标签使用的所有符号。定义字体包含一个 SHAPERECORD 数组，该数组描述了符号的轮廓。这些形状记录与用于定义非文本形状的 DefineShape 使用的记录相同。为了将文件大小保持在最小，仅将定义字体标签中实际使用的符号包含在内。

DefineText 标签存储要显示的实际文本字符串，表示为一系列字形索引。它还包括文本对象的边界框、变换矩阵以及颜色和大小等样式属性。

DefineText 包含一个 TEXTRECORD 数组。TEXTRECORD 选择当前字体、颜色和点大小，以及文本中下一个字符的 x 和 y 位置。这些样式适用于所有后续字符，直到另一个 TEXTRECORD 改变样式。TEXTRECORD 还包含对当前字体字形表的索引数组。字符不是通过字符代码引用，如传统编程中那样，而是通过字形表中的索引引用。字形数据还包括文本字符串中每个字符的进位值。

注意：必须在引用它的任何 DefineText 标签之前始终有 DefineFont 标签。

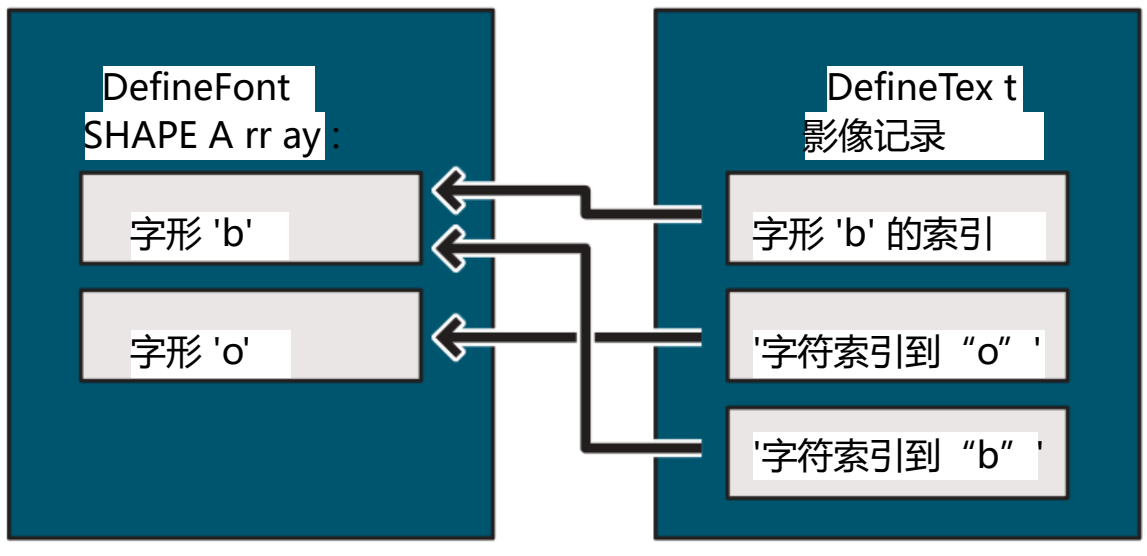
静态字形文本示例

考虑显示静态字符文本“bob”的例子，使用 Arial 字体，字号为 24 点。

首先，使用 DefineFont 标签定义字符。字符表，类型为 SHAPE，包含两个 SHAPERECORD。索引 0 是小写字母 b 的形状（见图）。索引 1 是小写字母 o 的形状。（bob 中的第二个 b 是重复的，不需要）。DefineFont 还包括一个唯一的 ID，以便可以通过 DefineText 标签进行选择。

接下来，使用 DefineText 标签定义文本本身。TEXTRECORD 设置第一个字符的位置，选择 Arial 字体（使用字体的字符 ID），并将字号设置为 24，以便字体缩放到正确的尺寸。（记住，字符是在 EM 坐标中定义的——实际的点大小是 DefineText 标签的一部分）。它还包含一个 GLYPHENTRY 数组。每个字符条目包含对字体形状数组的索引。在这个例子中，第一个字符条目索引为 0（对应 b 形状），第二个条目索引为 1（o），第三个条目索引为 0（b 再次）。每个 GLYPHENTRY 还包括一个用于准确定位字符的进位值。

以下图表说明了 DefineText 标签与 DefineFont 标签的交互方式：



字体标签

DefineFont

定义字体标签定义了特定字体中每个符号的形状轮廓。只有后续定义文本标签使用的符号才会被实际定义。

定义字体标签不能用于动态文本。动态文本需要使用定义字体 2 标签。最低文件格式版本为 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 10
字体 ID	UI16	该字体字符的 ID
偏移表	UI16[nGlyphs]	形状偏移数组
字形形状表	SHAPE[nGlyphs]	形状数组

字体 ID 唯一标识字体。它可以由后续的 DefineText 标签用来选择字体。与所有 SWF 字符 ID 一样，字体 ID 必须在 SWF 文件中的所有字符 ID 中是唯一的。

如果您为 DefineFont 标签提供 DefineFontInfo 标签，请注意 DefineFont 标签中图元的顺序必须与 DefineFontInfo 标签中字符码的顺序相匹配，且必须按码点顺序排序。

OffsetTable 和 GlyphShapeTable 一起使用。这两个表具有相同数量的条目，并且偏移量顺序和形状顺序之间存在一对一的匹配。OffsetTable 指向 GlyphShapeTable 中的位置。每个偏移量条目存储从偏移量表开始到对应形状位置的字节数差。由于 GlyphShapeTable 紧接在 OffsetTable 之后，因此可以通过将 OffsetTable 的第一个条目除以 2 来推断每个表中的条目数（即字体中的图元数）。

GlyphShapeTable 中每个 SHAPE 的第一个 STYLECHANGERECORD 不使用 LineStyle 和 LineStyles 字段。此外，每个形状的第一个 STYLECHANGERECORD 必须将字段 StateFillStyle0 和 FillStyle0 都设置为 1。

定义字体信息

定义字体信息标签将字形字体（使用定义字体定义）到设备字体的映射。它提供字体名称和样式以传递给播放平台的文本引擎，以及字符代码表

并提供一个字符代码表，用于标识对应 DefineFont 标签中每个符号所代表的字符，以便将 DefineText 标签的符号索引转换为字符字符串。

DefineFontInfo 标签的存在并不强制将字形字体转换为设备字体；它只是使该选项可用。实际选择字形或设备使用是根据 devicefont（参见引言）的值或 DefineEditText 标签中 UseOutlines 的值来决定的。如果播放平台上的设备字体不可用，Flash Player 将回退到字形文本。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 13。
字体 ID	UI16	该信息针对的字体 ID。
FontNameLen	UI8	字体名称的长度。
FontName	UI8[FontNameLen]	字体的名称（见下文）。
字体标志保留	UB[2]	保留位字段。
字体标志小文本	UB[1]	SWF 7 文件格式或更高版本：字体为小。字符符号对齐在像素边界上，用于动态和输入文本。
字体标志 ShiftJIS	UB[1]	ShiftJIS 字符码
字体标志 ANSI	UB[1]	ANSI 字符码
字体加粗	UB[1]	字体是斜体的。
字体标志斜体	UB[1]	字体是粗体的。
字体标志宽代码	UB[1]	如果为 1，则代码表是 UI16 数组；否则，代码表是 UI8 数组。
代码表	如果是字体标志宽代码，则 UI16[nGlyphs]，否则，UI8[nGlyphs]	符号到代码表，按升序排列。

代码表中的条目必须按代码点升序排列，按它们提供的值排序。代码表中的条目顺序也必须与应用于此 DefineFontInfo 标签的 DefineFont 标签中的符号顺序相匹配。这为相应 DefineFont 标签中符号的顺序提出了要求。

SWF 6 或更高版本的文件需要 Unicode 文本编码。这一要求的一个方面是，所有字符代码表都必须使用 UCS-2（UCS-2 通常是 UTF-16 的前 64k 代码点）。这种编码为每个字符使用固定的 2 个字节。这意味着当在 SWF 6 或更高版本的文件中定义 DefineFontInfo 标签时，必须设置 FontFlagsWideCodes，清除 FontFlagsShiftJIS 和 FontFlagsANSI，并且 CodeTable 必须由 UI16 条目组成（始终以小端字节序编码），使用 UCS-2 进行编码。

SWF 6 或更高版本文件的一个 Unicode 要求是，字体名称必须始终使用 UTF-8 进行编码。在 SWF 5 或更早的文件中，字体名称以平台特定的方式进行编码，在它们被创建的系统代码页中。播放平台将使用其当前代码页来解释它们，可能会得到不一致的结果。如果播放平台是 ANSI 系统，字体名称将被解释为 ANSI 字符串。如果播放平台是日本 shift-JIS 系统，字体名称将被解释为 shift-JIS 字符串。还有许多其他可能的播放平台代码页值，这些值决定了这一决策。这种播放区域依赖性是不理想的，这就是为什么 SWF 6 文件格式转向了字体名称的标准编码。请注意，DefineFontInfo 标签中的字体名称字符串不是空终止的；相反，它们的长度由 FontNameLen 字段指定。FontNameLen 指定 FontName 中的字节数，这不一定等于字符数，因为某些编码可能每个字符使用多个字节。

字体名称通常被原样使用，直接传递给播放平台的字体系统以定位字体。然而，存在一些特殊的间接字体名称，这些名称根据播放平台映射到不同的实际字体名称。这些间接映射被硬编码到每个特定平台的 Flash Player 端口中，每个平台的字体都是从系统默认字体或其他很可能可用的字体中选择。作为次要考虑因素，间接映射的选择是为了最大化间接字体在不同平台之间的相似性。

以下表格描述了受支持的间接字体名称。

西方间接字体

字体名称	示例
_sans	你好，世界
_serif	你好，世界
打字机	Hello world

日文间接字体

字体名称:	_ゴシック
英文名称:	哥特

UTF-8 字节字符串（十六进制）	:5F E3 82 B4 E3 82 B7 E3 83 83 E3 82 AF
示例外观：	こんな、漢字とカタカナ。

字体名称：	_等幅
英文名称：	Tohaba（哥特单线）
UTF-8 字节字符串（十六进制）	:5F E7 AD 89 E5 B9 85
示例外观：	こんな、漢字とカタカナ。

字体名称：	_明朝
英文名称：	明朝体
UTF-8 字节字符串（十六进制）	:5F E6 98 8E E6 9C 9D
示例外观：	こんな、漢字とカタカナ。

DefineFontInfo2

当生成 SWF 6 或更高版本时，建议您使用新的 DefineFontInfo2 标签而不是 DefineFontInfo。DefineFontInfo2 与 DefineFontInfo 相同，但增加了一个语言代码字段。如果您使用较旧的 DefineFontInfo，则语言代码将被假定为零，这会导致行为依赖于 Flash Player 运行的区域设置。

最小文件格式版本为 SWF 6。

字段	Type	评论
标题	记录头	标签类型 = 62。
字体 ID	UI16	该信息针对的字体 ID。
FontNameLen	UI8	字体名称的长度。
FontName	UI8[FontNameLen]	字体名称。
字体标志保留	UB[2]	保留位字段。
字体标志小文本	UB[1]	SWF 7 或更高版本：字体较小。字符字形对齐在像素边界上，用于动态和输入文本。

字体标志 ShiftJIS	UB[1]	总是 0。
字体标志 ANSI	UB[1]	总是 0。
字体加粗	UB[1]	字体是斜体的。
字体标志斜体	UB[1]	字体是粗体的。
字体标志宽代码	UB[1]	总是 1。
LanguageCode	语言代码	语言 ID。
代码表	UI16[nGlyphs]	在 UCS-2 中的字形到代码表的映射，按升序排列。

DefineFont2

DefineFont2 标签扩展了 DefineFont 的功能。增强功能包括以下内容：

- 偏移表中的 32 位条目，用于具有超过 64K 个符号的字体。
- 通过整合 DefineFontInfo 的所有功能，映射到设备字体。
- 动态符号文本的字体度量，以改善布局。

DefineFont2 标签是唯一可用于动态文本的字体定义。

最小文件格式版本为 SWF 3。

字段	Type	评论
标题	记录头	标签类型 = 48。
字体 ID	UI16	该字体字符的 ID。
FontFlagsHasLayout	UB[1]	包含字体度量/布局信息。
字体标志 ShiftJIS	UB[1]	ShiftJIS 编码。
字体标志小文本	UB[1]	SWF 7 或更高版本：字体较小。字符字形对齐在像素边界上，用于动态和输入文本。
字体标志 ANSI	UB[1]	ANSI 编码。
字体标志宽偏移。	UB[1]	如果为 1，则使用 32 位偏移。
字体标志宽代码	UB[1]	如果 1，字体使用 16 位代码；否则字体

		使用 8 位代码。
字体使用 16 位代码；否则为 8 位。	UB[1] FontFlagsItalic	斜体字体。
字体标志斜体	UB[1]	粗体字体。
字体粗体	语言代码	SWF 5 或更早版本：始终为 0；SWF 6 或更高版本：语言代码
LanguageCode	UI8	名称长度
FontNameLen	UI8[FontNameLen]	字体名称（参见 DefineFontInfo）
FontName	UI16	字形数量。对于设备字体，可能为零。
NumGlyphs	如果设置了 FontFlagsWideOffsets，则使用 UI32[NumGlyphs]，否则使用 UI16[NumGlyphs]	与 DefineFont 中的相同
偏移表	如果设置了 FontFlagsWideOffsets，则使用 UI32，否则使用 UI16	从 OffsetTable 起始位置到 CodeTable 起始位置的字节计数。
代码表偏移量	SHAPE[NumGlyphs]	与 DefineFont 中相同。
字形形状表	如果 FontFlagsWideCodes，则 UI16[NumGlyphs]，否则 UI8[NumGlyphs]。	按升序排序。SWF 6 或更高版本始终使用 UCS-2。
代码表	如果 FontFlagsHasLayout，则 UI16	字体升部高度。
字体上升	如果 FontFlagsHasLayout，UI16	字体下划线高度
字体下倾	如果 FontFlagsHasLayout，SI16	字体行高（见下文）。
字体行距	如果 FontFlagsHasLayout，SI16[NumGlyphs]	将值提前用于动态文本中的每个符号
字体进位表	如果 FontFlagsHasLayout，RECT[NumGlyphs]	在 Flash Player 版本 7 之前未使用（但必须存在）。
字体边界表	如果 FontFlagsHasLayout，UI16	在 Flash Player 版本 7 之前未使用（始终设置为 0 以节省空间）。
字间距计数	如果 FontFlagsHasLayout，则 KERNINGRECORD[KerningCount]	在 Flash Player 版本 7 之前未使用（当 KerningCount 为 0 时省略）。
字体字距调整表		

在 SWF 6 或更高版本的文件中，DefineFont2 具有与 DefineFontInfo 相同的 Unicode 要求。与 DefineFontInfo 标签类似，CodeTable（因此也包括 OffsetTable、GlyphShapeTable 和 FontAdvanceTable）必须按代码点顺序排序。

如果仅使用 DefineFont2 标签进行动态设备文本，且不希望有字形渲染回退，则将 NumGlyphs 设置为 0，并省略所有具有 NumGlyphs 条目的表。这将显著减小 DefineFont2 标签的大小。没有字形的 DefineFont2 标签无法支持静态文本，静态文本使用字形索引来选择字符，也无法支持字形文本，字形文本需要字形形状定义。

布局信息（上升、下降、行距、进位表、边界表、字距调整表）仅对动态字形文本有用。这些信息取代了静态字形文本中使用的每个字符放置信息。DefineFont2 标签中的布局信息是相当标准的字体度量信息，通常可以直接从标准字体定义中提取，例如 TrueType 字体。

注意：行距是垂直行间距度量。它是从一行下降的底部到下一行上升的顶部的距离（以 EM-square 坐标表示）。

与 DefineFont 类似，在 DefineFont2 中，每个 SHAPE 在 GlyphShapeTable 中的第一个 STYLECHANGERECORD 不使用 LineStyle 和 LineStyles 字段。此外，每个形状的第一个 STYLECHANGERECORD 必须同时设置字段 StateFillStyle0 和 FillStyle0 为 1。

DefineFont2 标签为字体边界表和字距调整表预留空间。这些信息在 Flash Player 版本 7 及以下版本中未使用。然而，为了构成一个有效的 DefineFont2 标签，这些信息必须存在。为 FontBoundsTable 提供最小（低比特）RECT，并且始终将 KerningCount 设置为 0，这允许省略 FontKerningTable。

DefineFont3

DefineFont3 标签与 DefineFontAlignZones 标签一起在 SWF 8 中引入。DefineFontAlignZones 标签是可选的，但建议用于使用高级抗锯齿的 SWF 文件，并且它修改 DefineFont3 标签。

DefineFont3 标签扩展了 DefineFont2 的功能，通过在 GlyphShapeTable 中以 20 倍的分辨率表达 SHAPE 坐标。所有 EMSquare 坐标在导出时都乘以 20，允许以 1/20 单位的分数分辨率。这允许更精确地定义符号，并导致更好的视觉质量。

最小文件格式版本为 SWF 8。

字段	Type	评论
标题	记录头	标签类型 = 75。

字体 ID	UI16	该字体字符的 ID。
FontFlagsHasLayout	UB[1]	包含字体度量/布局信息。
字体标志 ShiftJIS	UB[1]	ShiftJIS 编码。
字体标志小文本	UB[1]	SWF 7 或更高版本：字体较小。字符字形对齐在像素边界上，用于动态和输入文本。
字体标志 ANSI	UB[1]	ANSI 编码。
字体标志宽偏移。	UB[1]	如果为 1，则使用 32 位偏移。
字体标志宽代码	UB[1]	必须是 1。
字体加粗	UB[1]	斜体字体。
字体标志斜体	UB[1]	粗体字体。
LanguageCode	语言代码	SWF 5 或更早版本：始终为 0；SWF 6 或更高版本：语言代码
FontNameLen	UI8	名称长度
FontName	UI8[FontNameLen]	字体名称（参见 DefineFontInfo）
NumGlyphs	UI16	字形数量。对于设备字体，可能为零。
偏移表	如果设置了 FontFlagsWideOffsets，则使用 UI32[NumGlyphs]，否则使用 UI16[NumGlyphs]	与 DefineFont 中的相同
代码表偏移量	如果设置了 FontFlagsWideOffsets，则使用 UI32，否则使用 UI16	从 OffsetTable 起始位置到 CodeTable 起始位置的字节计数。
字形形状表	SHAPE[NumGlyphs]	与 DefineFont 中的相同
代码表	UI16[NumGlyphs]	按升序排序。SWF 6 或更高版本始终使用 UCS-2。
字体上升	如果 FontFlagsHasLayout，则 UI16	字体升部高度。
字体下倾	如果 FontFlagsHasLayout，则 UI16	字体下划线高度
字体行距	如果 FontFlagsHasLayout，SI16	字体行高（见下文）。

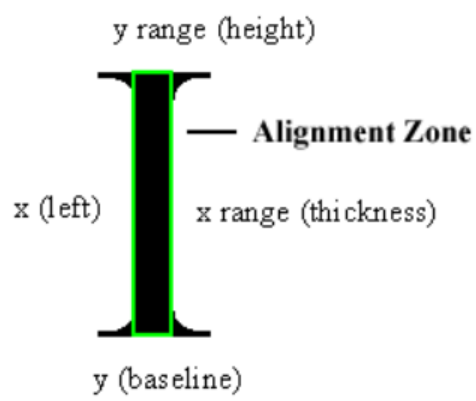
字体进位表	如果 FontFlagsHasLayout，SI16[NumGlyphs]	将值提前用于动态文本中的每个符号
字体边界表	如果 FontFlagsHasLayout，RECT[NumGlyphs]	在 Flash Player 版本 7 之前未使用（但必须存在）。
字间距计数	如果 FontFlagsHasLayout，则 UI16	在 Flash Player 版本 7 之前未使用（始终设置为 0 以节省空间）。
字体字距调整表	如果 FontFlagsHasLayout，KERNINGRECORD [KerningCount]	在 Flash Player 版本 7 之前未使用（当 KerningCount 为 0 时省略）。

定义字体对齐区域

DefineFont3 标签可以被 DefineFontAlignZones 标签修改。高级文本渲染引擎使用对齐区域来建立字形边界以进行像素对齐。对齐区域对于高质量显示字体至关重要。

对齐区域定义了字形的强垂直和水平组件的边界框。该框由左坐标、厚度、基线坐标和高度描述。厚度或高度通常设置为 0 的小值。

例如，考虑字母 I。字母 I 在其基线和字母顶部有强烈的水平线。字母 I 还有在茎的边缘出现的强烈垂直线——不是短的上横或装饰。这些强烈的垂直线和水平线定义了字母中心块的定位区域。



最小文件格式版本为 SWF 8。

定义字体对齐区域		
字段	Type	评论
标题	记录头	标签类型 = 73。
字体 ID	UI16	要使用的字体 ID，由 DefineFont3 指定。

CSM 表提示	UB[2]	字体粗细提示。指字体中典型笔画的粗细。0 = 薄；1 = 中；2 = 厚；Flash Player 为许多字体维护一组 CSM 表。但是，如果字体未在 Flash Player 的内部表中找到，则使用此提示来选择合适的表。
保留	UB[6]	必须为 0。
区域表	ZONERECORD[字符数]	每个字符的定位区域信息。

ZONE 记录		
字段	Type	评论
NumZoneData	UI8	区域数据条目数量。始终为 2。
区域数据	ZONEDATA[NumZoneData]	压缩对齐区域信息。
保留	UB[6]	必须为 0。
ZoneMaskY	UB[1]	如果存在 Y 对齐区域，则设置。
区域遮罩 X	UB[1]	如果有 X 个对齐区域，则设置。

ZONE 数据		
字段	Type	评论
对齐坐标	FLOAT16	对齐区域的 X（左）或 Y（基线）坐标。
范围	FLOAT16	对齐区域的宽或高。

字距记录

字距调整记录定义了两个字符在 EM 平方坐标中的距离。某些字符对如果移动得更近或更远，看起来会更加美观。FontKerningCode1 和 FontKerningCode2 字段是左右字符的字符代码。FontKerningAdjustment 字段是一个有符号整数，用于定义要添加到左字符进位值中的值。

字段	Type	评论
字体字距调整码 1	如果 FontFlagsWideCodes，则 UI16，否则 UI8	左字符的字符码
字体字距调整码 2	如果字体标志为宽代码，则使用 UI16，否则使用 UI8	右字符的字符码
字体字距调整	SI16	相对于左字符进位值的调整

定义字体名称

DefineFontName 标签包含嵌入到 SWF 文件中的字体的名称和版权信息。

最小文件格式版本为 SWF 9。

字段	Type	评论
标题	记录头	标签类型 = 88
字体 ID	UI16	该字体的 ID，此条目所指
FontName	字符串	字体名称。对于以 Type 1 开始的字体，这是 PostScript 的 FullName。对于以 sfnt 格式开始的字体，如 TrueType 和 OpenType，这是名称 ID 4，平台 ID 1，语言 ID 0（全名，Mac OS，英语）。
字体版权	字符串	随机版权信息字符串

静态文本标签

定义文本

DefineText 标签定义了一块静态文本。它描述了文本对象中每个字符的字体、大小、颜色和精确位置。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 11。
角色 ID	UI16	此文本字符的 ID。
文本边界	RECT	文本边界范围。
文本矩阵	矩阵	文本的变换矩阵。
字形位	UI8	每个字形索引中的位。
前进位	UI8	每个前进值中的位
文本记录	TEXTRECORD[零个或多个]	文本记录。
记录结束标志	UI8	必须为 0。

TextBounds 字段是包围此文本块中所有字符的矩形。

GlyphBits 和 AdvanceBits 字段分别定义了每个 GLYPHENTRY 记录中 GlyphIndex 和 GlyphAdvance 字段使用的位数。

文本记录

TEXTRECORD 用于设置后续字符的文本样式。它可以用来选择字体、更改文本颜色、更改点大小、插入换行符或设置下一个字符在文本中的 x 和 y 位置。新的文本样式将一直应用，直到另一个 TEXTRECORD 更改样式。

TEXTRECORD 还定义了文本对象中的实际字符。字符通过当前字体的字形表索引来引用，而不是通过字符代码。每个 TEXTRECORD 包含一组具有相同文本样式且位于同一文本行上的字符。

字段	Type	评论
TextRecordType	UB[1]	总是 1。
风格标志保留	UB[3]	总是 0。
风格标志包含字体	UB[1]	如果指定了文本字体，则为 1。
风格标志包含颜色	UB[1]	如果指定了文本颜色。
StyleFlagsHasYOffset	UB[1]	如果指定了 y 偏移量。

样式标志具有 X 偏移	UB[1]	如果指定了 X 偏移量。
字体 ID	如果 StyleFlagsHasFont，UI16	以下文本的字体 ID。
文本颜色	如果有 StyleFlagsHasColor，RGB 如果此记录是 DefineText2 标签的一部分，RGBA	以下文本的字体颜色。
X 偏移量	如果有 StyleFlagsHasXOffset，SI16	下文文本的 X 偏移量。
Y 偏移量	如果 StyleFlagsHasYOffset，则 SI16	下文文本的 Y 偏移量。
文本高度	如果有字体，UI16	以下文本的字体高度。
字形数量	UI8	记录中的符号数量。
符号条目。	GLYPHENTRY[符号数]。	符号条目（见下文）。

FontID 字段用于选择之前定义的字体。此 ID 唯一标识 SWF 文件中之前定义的 DefineFont 或 DefineFont2 标签。

TextHeight 字段定义了字体的高度（以 twips 为单位）。例如，50 像素的高度相当于 1000 的 TextHeight（50 * 20 = 1000）。

XOffset 字段定义了 TextBounds 矩形左边缘到字形参考点的偏移量（字形参考点是 EM 方框内的点，从该点开始第一个曲线段）。通常，参考点位于基线附近，靠近字形的左侧（请参阅字形文本的示例）。XOffset 通常用于创建缩进文本或非左对齐文本。如果没有指定 XOffset，则假定偏移量为零。

YOffset 字段定义了从 TextBounds 矩形顶部到字形参考点的偏移量。
文本 Y 偏移量通常用于插入换行，将文本位置移动到新行的起始位置。

GlyphCount 字段定义了字符串中的字符数以及 GLYPHENTRY 表的大小。

字形条目

GLYPHENTRY 结构描述了文本行中的单个字符。它由当前字体的字形表中的索引和一个进阶值组成。进阶值是此字符的参考点与下一个字符的参考点之间的水平距离。

字段	Type	评论
字符索引	UB[字符位]	当前字体中的字符索引。
字符进位	SB[进位位]	字形进位值。

定义文本 2

DefineText2 标签几乎与 DefineText 标签相同。唯一的区别是，包含在 DefineText2 标签内的 Type 1 文本记录使用 RGBA 值（而不是 RGB 值）来定义 TextColor。这允许字符部分或完全透明。

使用 DefineText2 定义的文本始终以符号渲染。设备文本永远不会包含透明度。

最小文件格式版本为 SWF 3。

字段	Type	评论
标题	记录头	标签类型 = 33。
角色 ID	UI16	此文本字符的 ID。
文本边界	RECT	文本边界范围。
文本矩阵	矩阵	变换矩阵。
字形位	UI8	每个字形索引中的位。
前进位	UI8	每个前进值中的位
文本记录	TEXTRECORD[零个或多个]	文本记录。
记录结束标志	UI8	必须为 0。

动态文本标签

DefineEditText

DefineEditText 标签定义了一个动态文本对象，或文本字段。

文本字段与一个 ActionScript 变量名相关联，文本字段的内容存储在该变量中。SWF 文件可以读取和写入变量的内容，该变量始终与显示的文本保持同步。如果未设置只读标志，用户可以交互式地更改文本字段的值。

DefineEditText 使用的字体必须使用 DefineFont2 定义，而不是 DefineFont。最低文件格式版本为 SWF 4。

字段	Type	评论
标题	记录头	标签类型 = 37。
角色 ID	UI16	该动态文本字符的 ID。

边界	RECT	完全包围文本框的矩形。
有文本	UB[1]	0 = 文本框没有默认文本。1 = 文本框最初显示由 InitialText 指定的字符串。
自动换行	UB[1]	0 = 文字不会自动换行，将向右滚动。1 = 当行尾到达时，文字将自动换行。
多行	UB[1]	0 = 文本字段只有一行。1 = 文本字段为多行且可滚动。
密码	UB[1]	0 = 字符将按输入显示。1 = 所有字符都显示为星号。
只读	UB[1]	0 = 文本编辑已启用。1 = 文本编辑已禁用。
具有文字颜色	UB[1]	0 = 使用默认颜色。1 = 使用指定颜色（文字颜色）。
HasMaxLength	UB[1]	0 = 文本长度无限。1 = 字符串最大长度由 MaxLength 指定。
HasFont	UB[1]	0 = 使用默认字体。1 = 使用指定的字体（FontID）和高度（FontHeight）。（如果 HasFontClass 为真，则不能为真）。
有字体类	UB[1]	0 = 无字体类，1 = 指定字体类和高度（如果已设置 HasFont，则不可能为真）。支持 Flash Player 9.0.45.0 及更高版本。
自动调整大小	UB[1]	0 = 固定大小。1 = 根据内容调整大小（仅限 SWF 6 或更高版本）。
布局	UB[1]	提供布局信息。
无选择	UB[1]	启用或禁用交互式文本选择。
边框	UB[1]	在文本字段周围绘制边框。
WasStatic	UB[1]	0 = 作为动态文本创建；1 = 作为静态文本创建
HTML	UB[1]	0 = 纯文本内容。1 = HTML 内容（见下文）。
使用轮廓	UB[1]	0 = 使用设备字体。1 = 使用符号字体。
字体 ID	如果有字体，则使用 UI16	要使用的字体 ID。
字体类。	如果有 FontClass，则输入 STRING。	从另一个 SWF 加载并用于此文本的字体类名称。

字体高度	如果有字体，则使用 UI16	字体高度（twips 单位）
文本颜色	如果有文字颜色，RGBA	文字颜色
最大长度	如果有最大长度，UI16	文本长度被限制在此。
对齐	如果有布局，UI8	0 = 左；1 = 右；2 = 居中；3 = 居中对齐
左边距	如果有布局，UI16	左边距（twips 单位）
右边距	如果有布局，UI16	右边距（以 twips 为单位）
缩进	如果有布局，UI16	缩进（以 twips 为单位）
行间距	如果有布局，SI16	行距（一行底端下划线与下一行顶端上划线之间的垂直距离）
变量名	字符串	存储文本字段内容的变量名称。可能使用点符号或斜杠符号进行非全局变量的限定。
初始文本	如果有文本字符串	初始显示的文本

如果设置了 HTML 标志，InitialText 的内容将被解释为 HTML 标签语言的子集，其中包含一些在 HTML 中通常不存在的添加。以下标签被支持：

Tag	描述
...	定义一个段落。属性 align 可能存在，其值为 left、right 或 center。
 	插入换行符。
<a> ... 	定义超链接。属性 href 必须存在。属性 target 是可选的，用于指定窗口名称。
 ... 	定义使用指定字体的文本范围。以下属性可用： • face：指定必须与 DefineFont2 标签中提供的字体名称匹配的字体名称；• size：以 twips 为单位指定，可以包含一个前置的 '+' 或 '-' 以表示相对大小；• color：指定为 #RRGGBB 十六进制三联色
...	定义一段加粗文本的范围。
... 定义一段斜体文本的范围。	

<u> ... </u>	定义一段下划线文本的范围。
...	定义一个项目符号段落。《ul》标签不是必需的，也不受支持。编号列表不受支持。
<textformat> ... </textformat>	定义具有特定格式选项的文本范围。以下属性可用： • 左边距，指定左边距（以 twips 为单位）； • 右边距，指定右边距（以 twips 为单位）； • 缩进，指定左边缩进（以 twips 为单位）； • blockindent，用于指定缩进量（以 twips 为单位）； • leading，用于指定行首缩进量（以 twips 为单位）； • tabstops，用于指定以逗号分隔的制表位列表，每个制表位都以 twips 为单位指定
空白	插入一个制表符，它将移动到由 <textformat> 标签定义的下一个制表位。

CSMTextSettings

除了本章前面讨论的高级文本渲染标签之外，渲染引擎还支持用于修改文本字段的标签。CSMTextSettings 标签修改之前流过的 DefineText、DefineText2 或 DefineEditText 标签。CSMTextSettings 标签可以打开或关闭文本字段的先进抗锯齿，还可以用来定义质量和选项。

最小文件格式版本为 SWF 8。

字段	Type	评论
标题	记录头	标签类型 = 74。
文本 ID	UI16	此标签应用于 DefineText、DefineText2 或 DefineEditText 的 ID。
使用 Flash 类型	UB[2]	0 = 使用正常渲染器。1 = 使用高级文本渲染引擎。
网格适配	UB[3]	0 = 不使用网格适配。不使用对齐区域和 LCD 子像素信息。1 = 像素网格适配。仅支持左对齐的动态文本。此设置提供最佳

		在高级抗锯齿文本可读性方面，字母清晰且与像素对齐。2 = 子像素网格适配。将字母对齐到 LCD 显示器使用的 1/3 像素。也可以提高 CRT 输出的质量。
保留	UB[3]	必须为 0。
厚度	F32	关联文本字段的厚度属性。设置为 0.0 以使用默认（抗锯齿表）值。
锐度	F32	关联文本字段的锐度属性。设置为 0.0 以使用默认（抗锯齿表）值。
保留	UI8	必须为 0。

此设置提供最终的厚度和锐度字段，这是针对特定文本字段的 CSM 参数的解释。
厚度和锐度设置将覆盖该文本字段的默认 CSM 设置。

在渲染时，CSM 参数将从厚度和锐度计算得出：

```
outsideCutoff = (0.5f * sharpness - thickness) * fontSize; insideCutoff =
(-0.5f * sharpness - thickness) * fontSize;
```

在截止计算中使用字体大小会导致 CSM 参数的线性缩放，而线性缩放在应用显著缩放时通常是一个较差的近似。
当文本字段将进行缩放时，通常最好使用默认表格或提供自己的抗锯齿表格。

DefineFont4

DefineFont4 仅支持新的 Flash 文本引擎。嵌入式字体的字体数据存储在 CFF 格式中。

最小文件格式版本是 SWF 10。

字段	Type	评论
标题	记录头	标签类型 = 91
字体 ID	UI16	该字体字符的 ID。
字体标志保留	UB[5]	保留位字段。
字体标志具有字体数据	UB[1]	字体已嵌入。字体标签包含 SFNT 字体数据块。
字体加粗	UB[1]	斜体字体

字体标志斜体	UB[1]	粗体字体
FontName	字符串	字体名称。
字体数据	FONTDATA[0 或 1]	当存在时，这是一个 OpenType CFF 字体，如 OpenType 规范中定义，请访问 www.microsoft.com/typography/otspec。以下表必须存 在：‘CFF’，‘cmap’，‘head’，‘maxp’，‘OS/2’，‘post’，以及以下之一（a）‘hhea’和‘hmtx’，或（b）‘vhea’，‘vmtx’，和‘VORG’。‘cmap’表必须包含以下类型的 Unicode‘cmap’子表之一：（0，4），（0，3），（3，10），（3，1）或（3，0）[注：平台 ID，平台特定编码 ID]。表如‘GSUB’，‘GPOS’，‘GDEF’， 和‘BASE’也可能存在。仅对嵌入字体有效。

第 11 章：声音

SWF 文件格式规范定义了一个小型高效的音效模型。SWF 支持多种音频编码格式，并可以使用多种采样率存储音频，包括双声道和单声道。Adobe Flash Player 支持这些声音的速率转换和多声道混音。支持的声道数量取决于 Flash Player 可用的 CPU 资源，但通常是三到八个声道。

SWF 文件格式中有两种类型的音效：

- 事件音效
- 流音效

事件声音会在某些事件发生时播放，例如鼠标点击，或者当 Flash Player 达到某个帧时。事件声音必须在使用之前定义（下载）。如果需要，它们可以用于多个事件的重用。事件声音还可以有“风格”的音效，这会改变基本音效的播放方式。

流式声音与时间轴紧密同步下载并播放。在此模式下，声音数据包与每一帧一起存储。

注意：使用的确切采样率如下。这些是基于 CD 音频的标准采样率，CD 音频的采样率为 44100 Hz。四种采样率分别为八分之一、四分之一、二分之一和精确的 44100 Hz 采样率。

```
5.5  kHz = 5512 Hz
11  kHz = 11025 赫兹
22  kHz = 22050 赫兹
44  kHz = 44100 赫兹
```

音频编码格式

Flash Player 可以使用多种编码格式存储音频。这些格式将在本章后面的部分进行更详细的描述。本节列出了 Flash Player 支持的编码格式、Flash Player 用来引用该格式的格式编号以及首次识别该格式编号的 SWF 版本。

编码格式	音频格式编号	最小 SWF 版本
未压缩，原生字节序	0	1
ADPCM	1	1
MP3	2	4

未压缩，小端序	3	4
内尔莫泽尔 16 kHz	4	10
内尔莫泽尔 8 kHz	5	10
内尔莫泽尔	6	6
Speex	11	10

事件音效

播放事件声音需要几个控制标签和记录：

- DefineSound 标签提供了构成事件声音的音频样本。
- SOUNDINFO 记录定义了应用于事件声音的样式。样式包括淡入、淡出、同步和循环标志，以及包络控制。
- StartSound 标签指示 Flash Player 开始播放声音。
- StartSound2 标签指示 Flash Player 开始播放来自另一个 SWF 的声音类。

定义声音

DefineSound 标签定义一个事件声音。它包括音频编码格式、采样率、每个样本的大小（8 或 16 位）、立体声/单声道标志以及音频样本数组。请注意，并非所有这些参数都会根据音频编码格式得到尊重。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 14
声音 ID	UI16	该声音的 ID。
声音格式	UB[4]	声音数据的格式。参见“音频编码格式”。
音频采样率	UB[2]	采样率。对于 Nellymoser 和 Speex 编解码器，此参数被忽略。MP3 不允许 5.5kHz。0 = 5.5 kHz；1 = 11 kHz；2 = 22 kHz；3 = 44 kHz
音频大小	UB[1]	每个样本的大小。此参数仅适用于未压缩格式。对于内部始终解码为 16 位的压缩格式，此参数被忽略。0 = snd8Bit；1 =

		16 位声音
音频类型	UB[1]	单声道或立体声音频。对于 Nellymoser 和 Speex，此设置被忽略。0 = 单声道；1 = 立体声
0 = 8 位声音；1 = 声音	音频	样本数量。不受单声道/立体声设置的影响；对于立体声音频，这是样本对的数量。
音频数据	UI8[音频数据大小]	音频数据因格式而异。

SoundId 字段唯一标识声音，以便可以通过 StartSound 播放。格式 0（未压缩）和格式 3（未压缩小端）相似。两者都编码未压缩的音频样本。对于 8 位样本，两种格式相同。对于 16 位样本，两种格式在字节顺序上有所不同。使用格式 0 时，16 位样本的编码和解码遵循编码器和 Flash Player 所在平台的本地字节顺序。使用格式 3 时，16 位样本始终以小端顺序（最低有效字节首先）编码，并在 Flash Player 播放前必要时进行字节交换。格式 0 明显不利，因为它引入了播放平台依赖性。对于 16 位样本，格式 3 对于 SWF 4 或更高版本来说，比格式 0 更可取。

声音数据的内容取决于 SoundStreamHead 标签中 SoundFormat 字段的值：

- 如果 SoundFormat 为 0 或 3，则 SoundData 包含原始、未压缩的样本。
- 如果 SoundFormat 为 1，则 SoundData 包含 ADPCM 声音数据记录。
- 如果 SoundFormat 为 2，则 SoundData 包含 MP3 声音数据记录。
- 如果 SoundFormat 为 4、5 或 6，SoundData 包含 Nellymoser 数据（见“Nellymoser 压缩”）。
- 如果 SoundFormat 为 11，SoundData 包含 Speex 数据（见“Speex 压缩”）。

StartSound

StartSound 是一个控制标签，用于启动（或停止）播放由 DefineSound 定义的声音。SoundId 字段标识要播放的声音。SoundInfo 字段定义声音的播放方式。通过在 SOUNDINFO 记录中设置 SyncStop 标志来停止声音。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 15。
声音 ID	UI16	播放声音字符的 ID。

声音信息	音频信息	音效风格信息。

StartSound2

StartSound 是一个控制标签，用于启动（或停止）播放由 DefineSound 定义的声音。SoundId 字段标识要播放的声音。SoundInfo 字段定义声音的播放方式。通过在 SOUNDINFO 记录中设置 SyncStop 标志来停止声音。

最小文件格式版本为 SWF 9。支持 Flash Player 9.0.45.0 及更高版本。

字段	Type	评论
标题	记录头	标签类型 = 89。
声音类名称	字符串	要播放的声音类名称。
声音信息	音频信息	音频风格信息。

音频风格

音频信息

SOUNDINFO 记录修改了事件声音的播放方式。事件声音通过 DefineSound 标签定义。可以修改的声音特性包括：

- 声音是否循环（重复）以及循环的次数。
- 声音播放的开始和结束位置。
- 用于基于时间的音量控制的声音包络。

字段	Type	评论
保留	UB[2]	总是 0。
同步停止	UB[1]	立即停止声音。
SyncNoMultiple	UB[1]	如果已经在播放，则不要开始声音。
包含信封	UB[1]	包含信封信息。
包含循环	UB[1]	包含循环信息。
包含出点信息	UB[1]	包含出点信息描述

包含入点	UB[1]	包含入点信息。
入点	如果有 HasInPoint，则 UI32	跳过声音开始处的样本数量。
出点	如果有出点，UI32	最后一个播放样本的样本位置。
循环次数	如果有循环，UI16	声音循环次数。
环境点	如果有包络，UI8	声音包络点数
包络记录	如果有包络，则 SOUNDENVELOPE[EnvPoints]	声音包络记录

声音包络

声音包络结构定义如下：

字段	Type	评论
Pos44	UI32	包络点的位置，以 44 kHz 采样率的样本数表示。如果使用低于 44 kHz 的采样率，请相应地乘以。
左声道级别	UI16	左声道的音量级别。最小值为 0，最大值为 32768。
右声道级别	UI16	右声道的音量级别。最小值为 0，最大值为 32768。

对于单声道声音，将左声道和右声道的音量设置为相同的值。如果值不同，则取平均值。

流媒体音频

SWF 文件格式支持流式声音模式，在这种模式下，声音数据与时间轴紧密同步播放和下载。在此模式下，声音数据包存储在每个帧中。

当存在流式声音，且播放 CPU 速度过慢，无法维持所需的 SWF 帧率时，Flash Player 会跳过动画帧以保持声音同步并避免丢失声音样本。（跳过的帧中的动作仍然会执行。）

SWF 文件的主时间轴一次只能播放一个流式声音，但每个精灵都可以有自己的流式声音（参见精灵和电影剪辑）。

音频流头部

如果时间轴包含流式声音数据，则必须有 SoundStreamHead 或 SoundStreamHead2 标签

必须在第一个声音数据块之前有一个 SoundStreamHead 或 SoundStreamHead2 标签（参见“SoundStreamBlock”）。SoundStreamHead 标签定义了声音数据的数据格式、推荐的播放格式以及每个 SoundStreamBlock 的平均样本数。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 18。
保留	UB[4]	总是零。
播放声音速率	UB[2]	播放采样率：0 = 5.5 kHz；1 = 11 kHz；2 = 22 kHz；3 = 44 kHz
播放声音大小	UB[1]	播放样本大小。始终为 1（16 位）。
播放声音类型	UB[1]	播放通道数：单声道或立体声。0 = 单声道；1 = 立体声
流媒体声音压缩	UB[4]	流媒体声音数据格式。1 = ADPCM；SWF 4 及以后版本：2 = MP3
流媒体声音采样率	UB[2]	流媒体声音数据的采样率：0 = 5.5 kHz；1 = 11 kHz；2 = 22 kHz；3 = 44 kHz
流媒体声音大小	UB[1]	流媒体声音数据的样本大小。 总是 1（16 位）。
流媒体声音类型	UB[1]	流音数据中的通道数。0 = 单声道；1 = 立体声
流音样本计数	UI16	每个 SoundStreamBlock 中样本的平均数量。不受单声道/立体声设置的影响；对于立体声音频，这是样本对的数量。
播放延迟搜索	如果 StreamSoundCompression=2，则表示 SI16，否则不存在	请参阅“MP3 声音数据”。这里的值应与第一个 SoundStreamBlock 中的 SeekSamples 字段相匹配。

PlaybackSoundRate、PlaybackSoundSize 和 PlaybackSoundType 字段仅供参考；Flash Player 可能会忽略它们。

音频流头部 2

SoundStreamHead2 标签与 SoundStreamHead 标签相同，区别在于它允许 StreamSoundCompression 和 StreamSoundSize（SWF 3 文件格式）有不同的值。

字段	Type	评论
标题	记录头	标签类型 = 45
保留	UB[4]	总是零。
播放声音速率	UB[2]	播放采样率：0 = 5.5 kHz；1 = 11 kHz；2 = 22 kHz；3 = 44 kHz
播放声音大小	UB[1]	播放样本大小：0 = 8 位；1 = 16 位
播放声音类型	UB[1]	播放声道数：0 = 单声道（sndMono）；1 = 立体声（sndStereo）
流媒体声音压缩	UB[4]	声音数据格式。参见“音频编码格式”。
流媒体声音采样率	UB[2]	流式声音数据的采样率：MP3 不允许 5.5 kHz。0 = 5.5 kHz；1 = 11 kHz；2 = 22 kHz；3 = 44 kHz
流媒体声音大小	UB[1]	每个样本的大小。对于压缩格式始终为 16 位。对于未压缩格式可能为 8 或 16 位：0 = 8 位；1 = 16 位
流媒体声音类型	UB[1]	流式声音数据中的通道数：0 = 单声道（sndMono）；1 = 立体声（sndStereo）
流音样本计数	UI16	每个 SoundStreamBlock 中样本的平均数量。不受单声道/立体声设置的影响；对于立体声音频，这是样本对的数量。
播放延迟搜索	如果 StreamSoundCompression=2，则表示 SI16，否则不存在	查看 MP3 音频数据。这里的值应与第一个 SoundStreamBlock 中的 SeekSamples 字段匹配。

PlaybackSoundRate、PlaybackSoundSize 和 PlaybackSoundType 字段仅供参考；Flash Player 可能会忽略它们。

音流块

SoundStreamBlock 标签定义了与帧数据交织的音频数据，以便在通过网络连接流式传输 SWF 文件时播放声音。SoundStreamBlock 标签必须由 SoundStreamHead 或 SoundStreamHead2 标签 precede。每个 SWF 帧中只能有一个 SoundStreamBlock 标签。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	RECORDHEADER（长）	标签类型 = 19。
流式音频数据	UI8[压缩数据大小]	压缩声音数据。

StreamSoundData 的内容取决于 SoundStreamHead 标签中 StreamSoundCompression 字段的值：

- 如果 StreamSoundCompression 为 0 或 3，StreamSoundData 包含原始、未压缩的样本。
- 如果 StreamSoundCompression 为 1，StreamSoundData 包含 ADPCM 声音数据记录。
- 如果流声音压缩为 2，则 StreamSoundData 包含一个 MP3 声音数据记录。
- 如果流声音压缩为 6，则 StreamSoundData 包含一个 NELLYMOSERDATA 记录。

MP3 流声音数据		
字段	Type	评论
样本计数	UI16	该块所表示的样本数量。不受单声道/立体声设置的影响；对于立体声音频，这是样本对的数量。
Mp3SoundData	MP3SOUNDDATA	带有 SeekSamples 值的 MP3 帧。

流媒体声音的帧划分

要获得最佳的流媒体声音播放效果，需要在每个 SWF 帧中提供 SoundStreamBlock 标签，并在每个 SoundStreamBlock 中包含相同数量的声音样本。每个 SWF 帧的理想样本数很容易确定：将采样率除以 SWF 帧率。如果结果是分数，则偶尔写入一个样本多一个或少一个的 SoundStreamBlock，以便使每帧的平均样本数尽可能接近理想值。

对于未压缩的音频，可以在 SoundStreamBlock 中包含任意数量的样本，因此可以

每个 SWF 帧可以包含一个理想数量的样本。对于 MP3 声音，情况不同。MP3 数据本身被组织成帧，一个 MP3 帧包含固定数量的样本（576 或 1152，取决于采样率）。包含 MP3 数据的 SoundStreamBlock 必须包含整个 MP3 帧而不是片段，因此包含 MP3 数据的 SoundStreamBlock 总是包含 576 或 1152 的倍数个样本。

保持 MP3 流媒体音轨与 SWF 播放同步有两个要求：

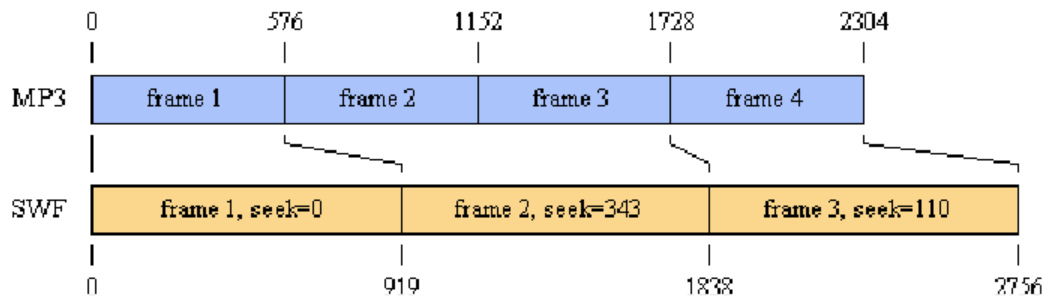
- 合理分配 MP3 帧到 SWF 帧中。
- 在 SoundStreamBlock 标签中提供适当的 SeekSamples 值。

这些技术在本节的其余部分中进行了描述。

对于流式 ADPCM 声音，将 ADPCM 数据包分配到 SWF 帧中的逻辑与将 MP3 帧分配到 SWF 帧中的逻辑相同。然而，对于 ADPCM 声音，没有 SeekSamples 或延迟的概念。因此，MP3 格式对于 SWF 4 或更高版本文件来说更可取。

要确定每个 SWF 帧的理想 MP3 帧数，将每个 SWF 帧的理想样本数除以每个 MP3 帧的样本数。这通常会导致非整数结果。通过交错不同数量的 MP3 帧的 SoundStreamBlocks 来实现理想的平均数。例如，在 SWF 帧速率为 12，采样率为 11 kHz 的情况下，每个 MP3 帧有 576 个样本；每个 SWF 帧的理想 MP3 帧数是(11025 / 12) / 576，大约为 1.6；这可以通过写入一个或两个 MP3 帧的 SoundStreamBlocks 来实现。在写入 SoundStreamBlocks 时，记录理想总样本数与迄今为止写入的总样本数之间的差异。在每个 SoundStreamBlocks 中放入尽可能多的 MP3 帧。

在不超过理想数量的情况下，尽可能使用 SoundStreamBlock。然后，在每个 SoundStreamBlock 中，使用截至前一个 SWF 帧结束的理想样本数与实际样本数的差值作为 SeekSamples 的值。这将使 Flash Player 能够在查找后精确地开始播放声音。以下是此示例的插图：



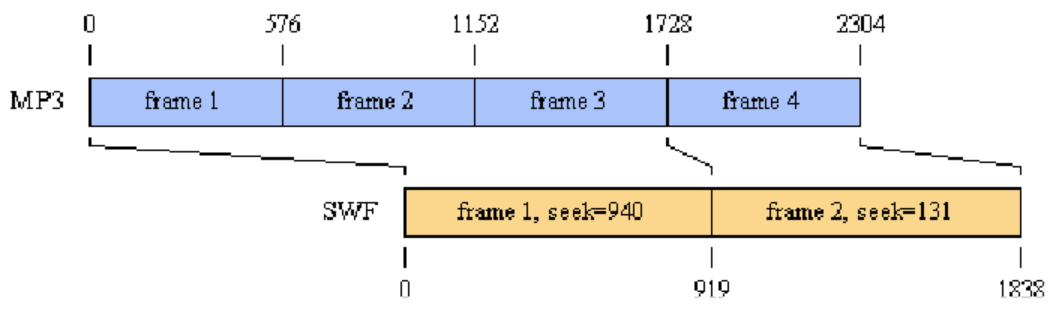
SWF 帧 1 中的 SoundStreamBlock 包含一个 MP3 帧，SeekSamples 设置为 0。帧 2 包含两个 MP3 帧，SeekSamples 设置为 919 - 576 = 343。帧 3 包含一个 MP3 帧，SeekSamples 设置为 1838 - 1728 = 110。

在连续播放中，Flash Player 会将所有 MP3 帧连接起来，并以它们的自然样本率播放它们（这就是为什么可以在 SWF 帧中包含少于理想数量的样本）。在查找特定帧后，例如由 ActionGotoFrame 触发的查找，Flash Player 将跳过由 SeekSamples 指示的样本数。对于

例如，在跳转到第 2 帧之后，它将跳过第 2 帧的 SoundStreamBlock 数据中的 343 个样本，这将导致声音播放从第 919 个样本开始，这是理想值。

如果每帧 SWF 中的 MP3 帧的理想数量小于一，则将存在无法容纳任何 MP3 帧而不超过理想样本数量的 SWF 帧。在这种情况下，写入一个 SampleCount = 0、SeekSamples = 0 且不包含 MP3 数据的 SoundStreamBlock。

一些 MP3 编码器具有初始延迟，在所需声音数据开始之前生成一定数量的静音或无意义的样本。这也有助于 Flash Player 的 MP3 解码器，在所需的样本之前提供一些渐近数据。在这种情况下，确定初始延迟占用的样本数量，并将该数字提供给第一个 SoundStreamBlock 中的 SeekSamples。Flash Player 将在执行查找时将此数字添加到任何其他帧的 SeekSamples 中。延迟还会影响决定将多少 MP3 帧放入 SoundStreamBlock 中的决策。以下是具有 940 个样本延迟的上述示例的修改：



SWF 第 1 帧中的 SoundStreamBlock 包含三个 MP3 帧，这是在不超出调整延迟（由 MP3 时间轴上方的左移表示）后的理想样本数的情况下所能容纳的最大数量。第 1 帧中的 SeekSamples 的值是特殊的；它表示延迟。第 2 帧包含一个 MP3 帧，SeekSamples 设置为 919 - (1728 - 940) = 131。

ADPCM 压缩

ADPCM（自适应差分脉冲编码调制）是一系列音频压缩和解压缩算法。它是一种简单但高效的压缩方案，避免了与更复杂的音效压缩方案相关的任何许可或专利问题，并有助于保持播放器实现的小型化。

对于 SWF 4 或更高版本的文件，MP3 压缩是一种更佳的格式（参见“MP3 压缩”）。MP3 在给定的比特率下产生显著更好的声音。

ADPCM 使用一种改进的差分脉冲编码调制（DPCM）采样技术，其中每个样本的编码是通过计算一个“差分”（DPCM）值，并应用一个包含先前量化值的复杂公式来得到的。结果是压缩码，可以恢复几乎相同的主体音频质量。

常见的实现方式是将 16 位线性 PCM 样本转换为 4 位码，从而实现 4:1 的压缩率。Jack Jansen 编写的公共领域 C 代码可在 www.cwi.nl/ftp/audio/adpcm.zip 找到。

SWF 文件格式扩展了 Jansen 的实现，以支持 2 位、3 位、4 位和 5 位 ADPCM 码。在选择码长时，通常需要在文件大小和音频质量之间进行权衡。SWF 文件格式中使用的码表如下（请注意，此处每个结构仅提供范围的下半部分，上半部分是精确的副本）：

```
int indexTable2[2] = {-1, 2}; int indexTable3[4] = {-1, -1, 2, 4}; int
indexTable4[8] = {-1, -1, -1, -1, 2, 4, 6, 8}; int indexTable5[16] = {-1, -1, -1,
-1, -1, -1, -1, -1, 1, 2,

      4, 6, 8, 10, 13, 16};
```

ADPCM 声音数据

ADPCMSOUNDATA 记录定义了 ADPCM 代码的大小，以及包含 ADPCM 数据的 ADPCMPACKET 数组。

字段	Type	评论
AdpcmCodeSize	UB[2] 0 = 每样本 2 位 1 = 每样本 3 位 2 = 每样本 4 位 3 = 每样本 5 位	每个 ADPCM 码的比特数减 2。每个码的实际大小为 AdpcmCodeSize + 2。
Adpcm 数据包	如果 SoundType = 单声道，则 ADPCMMONOPACKET [一个或多个] 如果 SoundType = 立体声，则 ADPCMSTEREOPACKET [一个或多个]	ADPCM 数据包数组。

ADPCM 数据包的结构根据声音是单声道还是立体声而有所不同。

ADPCM 单声道数据包。		
字段	Type	评论
初始样本。	SI16	第一个样本。与未压缩声音中的第一个样本相同。
初始索引	UB[6]	ADPCM 步长表中的初始索引.*
ADPCM 码数据	UB[4095 * (AdpcmCodeSize+2)]	4095 个 ADPCM 码。每个样本是 (AdpcmCodeSize + 2)位。

ADPCM 立体包		
字段	Type	评论
初始样本左	SI16	左通道的第一个样本。与未压缩声音中的第一个样本相同。
初始索引左	UB[6]	左通道在 ADPCM 步长表*中的初始索引。
初始样本右	SI16	右通道的第一个样本。与未压缩声音中的第一个样本相同。
初始索引右	UB[6]	右通道的 ADPCM 步长表*的初始索引
ADPCM 码数据	UB[8190 * (AdpcmCodeSize+2)]	每通道 4095 个 ADPCM 码，总计 8190。每个样本为(AdpcmCodeSize + 2)位。通道数据先左后右交错。

* 关于 StepSizeTable 的解释，请参阅 Jansen 源代码。

MP3 压缩

MP3 是一种复杂且高级的音频压缩算法，在 SWF 4 及以后的版本中得到支持。它以比 ADPCM 更好的压缩比产生更高质量的音频。一般来说，MP3 指的是 MPEG1 层 3；然而，SWF 文件格式支持 MPEG 的后续版本（V2 和 2.5），这些版本旨在支持更低的比特率。Flash Player 支持 CBR（恒定比特率）和 VBR（可变比特率）的 MP3 编码。

更多关于 MP3 的信息，请参阅 www.mp3-tech.org/和 www.iis.fhg.de/amm/techinf/layer3/index.html。编写 MP3 编码器相当困难，但可能存在公共领域的 MP3 编码库。

注意：请注意，软件和硬件 MP3 编码器和解码器可能有各自的许可要求。

MP3 声音数据

MP3 声音数据在以下表格中描述：

字段	Type	评论
SeekSamples	SI16	跳过的样本数量。
Mp3Frames	MP3 帧[零个或多个]	MP3 帧数组

关于 SeekSamples 字段的说明，请参阅“流式声音的帧细分”。

注意：对于事件声音，SeekSamples 仅限于指定初始延迟。

MP3 帧

MP3FRAME 记录与 MP3 音乐文件中找到的 MPEG 音频帧完全对应。帧的前 32 位包含标题信息，随后是一个字节数组，这些字节是编码的音频样本。

字段	Type	评论
同步字	UB[11]	帧同步。所有位都必须设置为 1。
Mpeg 版本	UB[2]	MPEG2.5 是 MPEG2 的扩展，用于处理非常低的比特率，允许使用较低的采样频率。0 = MPEG 版本 2.5；1 = 保留；2 = MPEG 版本 2；3 = MPEG 版本 1
层	UB[2]	在 SWF 文件中的 MP3 头部，层始终等于 1。MP3 中的“3”指的是层，而不是 Mpeg 版本。: 0 = 保留；1 = 层 III；2 = 层 II；3 = 层 I
保护位	UB[1]	如果保护位等于 0，则 16 位 CRC 紧随头部：0 = 由 CRC 保护；1 = 未保护
比特率	UB[4]	比特率以千位每秒为单位。例如，128 表示 128000 bps。 0 free free 1 32 8 2 40 16 3 48 24 4 56 32 5 64 40 6 80 48 7 96 56 8 112 64 9 128 80 10 160 96 11 192 112 12 224 128 13 256 144 14 320 160 15 bad bad
采样率	UB[2]	采样率（Hz）。值 MPEG1 MPEG2.x MPEG2.5

		----------0 44100 22050 11025 1 48000 24000 12000 2 32000 16000 8000 -- -- --
5 填充位	UB[1]	填充用于使比特率正好匹配。0 = 帧未填充；1 = 帧填充一个额外的槽位
保留	UB[1]	
通道模式	UB[2]	双通道文件由两个独立的单声道通道组成。每个通道使用文件比特率的一半。0 = 立体声；1 = 联合立体声（立体声）；2 = 双通道；2 = 单声道（单声道）
模式扩展	UB[2]	
版权	UB[1]	0 = 音频无版权；1 = 音频有版权
原创	UB[1]	0 = 原始媒体副本；1 = 原始媒体
强调	UB[2]	0 = 无；1 = 50/15 毫秒；2 = 保留；3 = CCIT J.17
样本数据	样本数据大小[UB]	编码的音频样本。

* 样本数据的大小计算如下（使用整数运算）：

```
大小 = (((MpegVersion == MPEG1 ? 144 : 72) * Bitrate) / SamplingRate) + PaddingBit
- 4
```

例如：MPEG1 帧的样本数据大小，比特率为 128000，采样率为 44100，填充位为 1，如下所示：

大小 = (144 * 128000) / 44100 + 1 – 4 = 414 字节

奈利莫瑟压缩

从 SWF 6 开始，提供了一种名为 Nellymoser Asao 的压缩音频格式。这是一种单声道（单声道）格式，针对低比特率语音传输进行了优化。该格式由 Nellymoser 公司开发。

请访问 www.nellymoser.com。

这里提供了 Nellymoser Asao 编码过程的概述。如需了解 Asao 格式的详细信息，请联系 Nellymoser。

Asao 使用声音的频域特性进行压缩。声音数据被分组为 256 个样本的帧。每个帧被转换到频域，并识别出最显著（最高振幅）的频率。选择多个频带进行编码；其余的则被丢弃。然后，每个帧的比特流编码了哪些频带在使用以及它们的振幅。

Speex 压缩

从 SWF 10 版本开始，SWF 文件可以存储使用免费开源的 Speex 语音编解码器（见 speex.org）压缩的音频样本。Speex 音频以格式 11 存储在 DefineSound 标签中。虽然 Speex 支持一系列采样率，但 SWF 中的 Speex 音频始终以 16 kHz 编码；DefineSound 标签的 SoundRate 字段被忽略。在 Speex 的情况下，SoundType 和 SoundSize 字段也被忽略。SWF 中的 Speex 始终为单声道，并且内部解码为 16 位音频样本。

Speex 1.2 beta 3 自 Flash Player 10（10.0.12）版本开始编译进 Flash Player。

第 12 章：按钮

SWF 文件格式中的按钮字符作为交互元素。它们可以程序化地响应用户事件。最常见的事件处理是鼠标的简单点击，但也可以捕获更复杂的事件。

按钮状态

按钮对象可以显示为三种状态之一：正常、悬停或按下。

正常状态是按钮的默认外观。当 SWF 文件开始播放时，以及鼠标在按钮外部时，显示正常状态。悬停状态是当鼠标移动到按钮内部时显示的状态。这允许在 SWF 文件中实现悬停按钮。按下状态是按钮被点击时的外观。当鼠标在按钮内部点击时，显示按下状态。

第四种状态——点击状态——定义了按钮的激活区域。这是一个不可见的状态，永远不会显示。它定义了按钮上对鼠标点击做出反应的区域。这个点击区域不一定是矩形的，也不必反映按钮的可见形状。

每个状态由字典中字符的实例集合组成。每个这样的实例都使用按钮记录来定义，在按钮定义中，它就像一个 PlaceObject 标签。对于上、中、下状态，这些字符在按钮进入该状态时放置在显示列表中。对于点击区域状态，这些字符定义了按钮的激活区域。

以下是一个典型按钮及其四种状态的示例。按钮最初是蓝色的。当鼠标移到按钮上时，它变为紫色。当鼠标在按钮内部按下时，阴影变化以模拟按下按钮的效果。第四种状态——点击区域——是一个简单的矩形。任何在这个形状之外的都属于按钮外部，任何在这个形状之内的都属于按钮内部。

SWF 文件格式不支持单选按钮或复选框。没有“选中”（选择）状态，按钮在鼠标释放后不能“粘住”。此外，无法将按钮分组为互斥选择。然而，这两种行为都可以通过使用按钮动作来模拟。

按钮跟踪

按钮跟踪指的是按钮如何跟踪鼠标的移动。按钮对象可以在两种模式之一中跟踪鼠标，即作为推按钮或作为菜单按钮。

如果点击推按钮，所有鼠标移动事件都将指向推按钮，直到鼠标按钮释放。这被称为捕获鼠标。例如，如果您点击推按钮并将鼠标拖动到按钮外

如果您点击一个按钮并将其拖出按钮外（不释放），按钮将变为悬停状态，指针保持为指向手。

菜单按钮不捕获鼠标。如果您点击菜单按钮并拖动到外部，按钮将变为上状态，指针将恢复为箭头。

事件、状态转换和动作

按钮对象可以在状态转换发生时执行操作（即，当按钮从一个状态转换为另一个状态时）。状态转换是响应某些事件发生的，例如鼠标点击或鼠标进入按钮。在 SWF 文件格式中，事件被描述为状态转换。下表显示了可能的状态转换和相应的 Flash Player 事件：

状态转换	事件	描述	视觉效果
空闲到向上滚动	滚过	鼠标在鼠标按钮弹起时进入击中区域。	按钮从弹起状态变为悬停状态。
悬停到空闲	滚出	鼠标在鼠标按钮抬起时离开击中区域。	按钮从悬停状态变为抬起状态。
从悬停到抬起再到悬停状态。	按压	鼠标在击中区域内按下。	按钮从悬停变为按下状态。
从按下到悬停再到抬起。	释放	鼠标按钮在鼠标位于击中区域时被释放。	按钮从 状态变为上状态。

以下转换仅适用于跟踪推按钮：

状态转换	事件	描述	视觉效果
上下翻转	拖动	鼠标在按下鼠标按钮的同时被拖动到活动区域内。	按钮状态从悬停变为按下。
上下左右	拖出	鼠标在按下鼠标按钮的同时被拖出点击区域外。	按钮从 降至上翻状态。
外部下翻至空闲	释放外部	当鼠标被捕获时，鼠标按钮在击中区域外被释放。	按钮从悬停状态变为正常状态。

以下转换仅适用于跟踪菜单按钮：

状态转换	事件	描述	视觉效果
空闲到下降	拖动	鼠标在按下鼠标按钮的同时被拖动到活动区域内。	按钮从上到下状态变化。
下降到空闲	拖出	鼠标在按下鼠标按钮的同时被拖出点击区域外。	按钮从下到上状态变化。

通常按钮动作仅在 OverDownToOverUp（当鼠标按钮释放时）执行，但 DefineButton2 允许通过任何状态转换触发动作。按钮对象可以执行 SWF 3 动作支持的所有动作（请参阅第 64 页上的“SWF 3 动作”）。

按钮标签

Buttonrecord

按钮记录定义了要在一个或多个按钮状态中显示的字符。按钮状态标志指示该字符属于哪个（或哪些）状态。

按钮记录与按钮状态之间不存在一对一的关系。单个按钮记录可以应用于多个按钮状态（通过设置多个按钮状态标志），并且任何按钮状态都可以存在多个按钮记录。

每个按钮记录还包括变换矩阵和深度（堆叠顺序）信息。这些信息与 PlaceObject 标签中的信息应用方式相同，只不过这两项信息都是相对于按钮字符本身的。

SWF 8 及以后版本支持新的 ButtonHasBlendMode 和 ButtonHasFilterList 字段，以支持按钮的混合模式和位图滤镜。Flash Player 7 及更早版本忽略这两个字段。

字段	Type	评论
按钮保留	UB[2]	保留位；始终为 0
按钮具有混合模式	UB[1]	0 = 无混合模式；1 = 具有混合模式（仅限 SWF 8 及以后版本）
按钮具有过滤器列表	UB[1]	0 = 无过滤器列表；1 = 有过滤器列表（仅限 SWF 8 及以后版本）
按钮状态击测试	UB[1]	存在于击测试状态
按钮按下状态	UB[1]	处于下电状态

按钮状态正常	UB[1]	在超过状态下存在
按钮状态为上	UB[1]	在上状态下存在
角色 ID	UI16	要放置的字符 ID
放置深度	UI16	放置字符的深度
放置矩阵	矩阵	字符放置的变换矩阵
颜色变换	如果在 DefineButton2 中，则使用 CXFORMWITHALPHA	角色颜色转换
过滤器列表	如果在 DefineButton2 内部且 ButtonHasFilterList = 1，则 FILTERLIST	此按钮上的过滤器列表
混合模式	如果在 DefineButton2 中且 ButtonHasBlendMode = 1，UI8	0 或 1=正常；2=层；3=正片叠底；4=滤色；5=变亮；6=变暗；7=差值；8=叠加；9=减去；10=反转；11=alpha；12=擦除；13=叠加；14=强光；值 15 至 255 为保留。

DefineButton

DefineButton 标签定义了一个按钮字符，供后续的控制标签如 PlaceObject 使用。

DefineButton 包含一个按钮记录数组，这些记录代表四种按钮形状：一个上状态字符、一个鼠标悬停状态字符、一个下状态字符和一个点击区域字符。不需要定义所有四种状态，但至少必须有一个按钮记录。例如，如果相同的按钮记录定义了上和悬停状态，则只需要三个按钮记录来描述该按钮。

每个状态允许有多个按钮记录。如果两个按钮记录引用了相同的状态，则在该状态下都会显示这两个记录。

DefineButton 还包含一个 ACTIONRECORDs 数组，当按钮被点击和释放时执行（参见第 64 页的“SWF 3 动作”）。

最小文件格式版本是 SWF 1。

字段	Type	评论
标题	记录头	标签类型 = 7
按钮 ID	UI16	该字符的 ID

角色	BUTTONRECORD[一个或多个]	组成按钮的字符
字符结束标志	UI8	必须为 0
动作	ACTIONRECORD[零个或多个]	要执行的操作
动作结束标志	UI8	必须为 0

定义按钮 2

DefineButton2 扩展了 DefineButton 的功能，允许任何状态转换触发动作。

最小文件格式版本为 SWF 3：

从 SWF 9 开始，如果 FileAttributes 标签的 ActionScript3 字段为 1，则 DefineButton2 标签中不得存在 BUTTONCONDACTION 字段。ActionOffset 必须为 0。这种结构不支持，因为不允许在同一个 SWF 文件中混合 ActionScript 1/2 和 ActionScript 3.0 代码。

字段	Type	评论
标题	记录头	标签类型 = 34
按钮 ID	UI16	该字符的 ID
保留标志	UB[7]	总是 0
跟踪为菜单	UB[1]	0 = 跟踪为正常按钮；1 = 跟踪为菜单按钮
动作偏移量	UI16	从该字段起始字节到第一个 BUTTONCONDACTION 的偏移量（以字节为单位），如果没有动作发生则为 0
字符	BUTTONRECORD [一个或多个]	组成按钮的字符
字符结束标志	UI8	必须为 0
动作	BUTTONCONDACTION [零个或多个]	在特定按钮事件上执行的操作

与 DefineButton2 关联的操作如下指定：

按钮动作		
字段	Type	评论
条件动作大小	UI16	从本字段起始到下一个字段的偏移量（字节）

		按钮动作，或 0（如果为上次动作）
CondIdleToOverDown	UB[1]	空转至超低速降速
CondOutDownToIdle	UB[1]	降速至空转
从过降至过降	UB[1]	过降至过降
从过降至下降	UB[1]	过降至下降
条件下降到上升	UB[1]	下降到上升
条件上升到下降	UB[1]	上升到下降
条件过载至空闲	UB[1]	过载至空闲
条件空闲至过载	UB[1]	空闲至过载
按键按下	UB[7]	SWF 4 或更高版本：键码否则始终为 0 有效的键码： • 1 = 向左箭头 • 2 = → • 3 = 首页 • 4 = 末尾 • 5 = 插入 • 6 = 删除 • 8 = 退格 • 13 = 回车 • 14 = 上箭头 • 15 = 向下箭头 • 16 = 页上 • 17 = 页下 • 18 = 制表符

		• 19 = 转义 • 32 至 126：遵循 ASCII
CondOverDownToIdle	UB[1]	OverDown 至 Idle
动作	ACTIONRECORD [零个或多个]	要执行的操作。请参阅 DoAction。
动作结束标志	UI8	必须为 0

对于每个事件处理器（每个 BUTTONCONDACTION），应填充一个或多个 Cond 位字段。这指定了事件处理器应在何时执行。

CondKeyPress 指定要捕获的特定键。即使应用该按钮的按钮没有输入焦点，CondKeyPress 事件处理器也会执行。对于 32 到 126 的 ASCII 键码，捕获的按键事件是复合的——它考虑了 Shift 键的效果。要捕获原始按键事件，直接对应于键盘上的键（包括修饰键本身），请使用 clip 事件处理器。

定义按钮颜色转换

DefineButtonCxform 定义按钮中每个形状和文本字符的颜色转换。这不适用于 DefineButton2，因为它包含自己的 CXFORM。

最小文件格式版本是 SWF 2。

字段	Type	评论
标题	记录头	标签类型 = 23
按钮 ID	UI16	此信息的按钮 ID
按钮颜色转换	CXFORM	角色颜色转换

定义按钮声音

DefineButtonSound 标签定义了在状态转换时播放哪些声音（如果有）。最低文件格式版本为 SWF 2。

字段	Type	评论
标题	记录头	标签类型 = 17
按钮 ID	UI16	这些声音应用到的按钮的 ID。

按钮声音字符 0	UI16	OverUpToIdle 的音效 ID
按钮音效信息 0	如果 ButtonSoundChar0 不为零，则 SOUNDINFO	OverUpToIdle 的音效风格
按钮声音字符 1	UI16	空闲到上滑声音 ID
按钮声音信息 1	SOUNDINFO（如果按钮声音字符 1 不为零）	空闲到上滑的声音风格
按钮声音字符 2	UI16	上滑到下滑的声音 ID
按钮声音信息 2	若 ButtonSoundChar2 不为零，则播放 SOUNDINFO	OverUpToOverDown 的音效风格
ButtonSoundChar3	UI16	OverDownToOverUp 的音效 ID
按钮声音信息 3	如果 ButtonSoundChar3 不为零，则 SOUNDINFO	OverDownToOverUp 的声音风格

第 13 章：精灵和影片剪辑

一个精灵对应于 Adobe Flash 创作应用中的电影剪辑。它是一个包含在其他 SWF 文件中的 SWF 文件，并支持许多常规 SWF 文件的功能，例如以下内容：

- 大多数可以在主文件中使用的控制标签。
- 一个可以独立于主文件停止、开始和播放的时间轴。
- 一个自动与主音轨混合的流式音频轨道。

精灵对象通过 DefineSprite 标签定义。它包含一个角色 ID、一个帧数和一系列控制标签。在 DefineSprite 标签中不允许使用定义标签（如 DefineShape）。所有控制标签引用的角色必须在精灵外部定义，并且在 DefineSprite 标签之前。

定义完成后，精灵通过主文件中的 PlaceObject2 标签进行显示。变换（在 PlaceObject 中指定）与放置在精灵内部的物体的变换进行连接。这些物体表现得像精灵的子物体，因此当精灵移动时，精灵内部的物体也会移动。同样，当精灵进行缩放或旋转时，子物体也会进行缩放或旋转。当精灵从显示列表中移除时，它会自动停止播放。

精灵名称

当精灵放置在显示列表中时，可以使用 PlaceObject2 标签给它命名。这个名称用于识别精灵，以便主文件（或其他精灵）可以在精灵内部执行操作。这通过 SetTarget 动作实现（参见 ActionSetTarget）。

例如，假设一个精灵对象被放置在主文件中，名称为"spinner"。主文件可以通过以下动作序列将这个精灵发送到其时间轴的第一帧：

1. 设置目标 "spinner"
2. 进入帧 0
3. 设置目标 ""（空字符串）
4. 动作结束。（动作代码 = 0）

注意：所有跟随“SetTarget “spinner”” 的动作都应用于“spinner”对象，直到“SetTarget” “”，这将动作上下文恢复到主文件。

SWF 文件格式支持在精灵中放置精灵，这可能导致对象复杂层次结构。为了处理这种复杂性，SWF 文件格式使用与文件系统用于标识精灵相似的命名约定。

例如，以下大纲显示了在主文件中定义的四个精灵：

```
MainMovie.swf 精灵 A（名称：
Jack）
    精灵 A1（名称：Bert）精灵 A2
    （名称：Ernie）精灵 B（名称：
    Jill）
```

以下 SetTarget 路径标识了前面的精灵：

- /Jack 从主文件中指向 SpriteA。
- 相对路径../指向 SpriteA 的主文件。
- /Jack/Bert 指向来自任何其他精灵或主文件的 SpriteA1。
- Bert 从 SpriteA 指向 SpriteA1。
- 相对路径../Ernie 指向来自 SpriteA1 的 SpriteA2。
- Jill 从 SpriteA1 瞄准 SpriteB。

定义精灵

DefineSprite 标签定义一个精灵角色。它由一个角色 ID 和帧数组成，后跟一系列控制标签。精灵通过结束标签终止。

标题中指定的长度反映了整个 DefineSprite 标签的长度，包括 ControlTags 字段。

定义标签（如 DefineShape）不允许在 DefineSprite 标签中使用。所有控制标签中引用的角色必须在定义精灵之前在文件主体中定义。

最小文件格式版本为 SWF 3。

字段	Type	评论
标题	记录头	标签类型 = 39
精灵 ID	UI16	精灵的字符 ID
帧数	UI16	精灵帧数
控制标签	标签[一个或多个]	一系列标签

以下标签在 DefineSprite 标签内有效：

- 显示帧
- 放置对象
- 放置对象 2
- 删除对象
- 删除 Object2
- 所有动作（见动作）
- 开始声音
- 帧标签
- 音频流头部
- 音频流头部 2
- 音频流块
- 结束

第 14 章：视频

Adobe Flash Player 6 及以上版本支持视频播放。视频可以通过以下方式提供给 Flash Player：

- 使用 SWF 视频标签将视频嵌入到 SWF 文件中。
- 通过 Adobe Flash 媒体服务器以 RTMP 协议传输视频流，该服务器可以选择从 FLV 文件格式的文件中获取视频数据。
- 使用 NetStream.play ActionScript 方法直接将 FLV 文件加载到 Flash Player 中。此方法仅在 Flash Player 7 及以上版本中可用。SWF 和 FLV 文件格式共享相同的视频编码格式。

关于 FLV 文件格式的完整信息，请参阅 Adobe 官方网站 www.adobe.com/go/video_file_format 上的 FLV 文件格式规范。

Sorenson H.263 比特流格式

自 SWF 6 版本起，提供了一个名为 Sorenson H.263 的单个视频格式。该格式基于 H.263，这是一个由 ITU 维护的开源视频编码标准。可以在 www.itu.int/上获取 H.263 标准的副本。

本文档中所有关于 H.263 标准的引用均指 1996 年 5 月发布的 H.263 草案版本，有时称为 H.263v1。这与 1998 年 2 月发布的 H.263 修订版本不同，有时称为 H.263v2 或 H.263+，目前是 ITU 生效的 H.263 版本。

Sorenson H.263 视频格式与 H.263 略有不同。从本质上讲，它是对 H.263 的子集，去除了一些高级功能并增加了一些新特性。这些更改在本节中进行了描述。

Sorenson H.263 视频格式由 Sorenson Media（www.sorenson.com）开发。现有能够为 Flash Player 播放视频的产品包括 Adobe Flash 创作应用和 Sorenson Squeeze for Adobe Flash 8，一款专业的视频压缩应用。您可以通过 Sorenson Spark 编解码器进行视频编码以支持 Flash Player；请联系 Sorenson Media 获取详细信息。

与 H.263 的差异总结

以下 H.263 特性从 Sorenson H.263 视频格式中被移除：

- 块组（GOB）层
- 分屏指示器
- 文档摄像头指示器

- 图片冻结释放
- 基于语法的算术编码
- PB 帧
- 连续存在多点
- 重叠块运动补偿

以下非 H.263 特性被添加到 Sorenson H.263 视频格式中：

- 可丢弃帧（无未来依赖的差异帧）
- 随意设置图片宽度高度，最高可达 65535 像素
- 始终开启无限制的运动矢量支持
- 可用去块标志来建议使用去块滤波器

为了支持这些差异，Sorenson H.263 视频格式在图像层和宏块层都使用与 H.263 不同的头信息。GOB 层不存在。

定义了 Sorenson H.263 视频格式的两个版本。在版本 0 中，块层与 H.263 相同。在版本 1 中，变换系数中的转义码编码方式与 H.263 不同。版本 0 和版本 1 没有其他差异

视频数据包

视频包是 Sorenson H.263 视频包中的顶级结构元素。它对应于 H.263 第 5.1 节中的图像层。这种结构包含在 SWF 文件格式中的 VideoFrame 标签内，也包含在 FLV 文件格式中的 VIDEODATA 结构内。

H263VIDEOPACKET		
字段	Type	评论
图像起始码	UB[17]	类似于 H.263 5.1.1；0000 0000 0000 0000 1
版本	UB[5]	视频格式版本；Flash Player 6 支持 0 和 1
时间参考	UB[8]	参见 H.263 5.1.2
图片大小	UB[3]	000：自定义，1 字节；001：自定义，2 字节；010：CIF（352x288）；011：QCIF

		(176x144); 100: SQCIF (128x96); 101: 320x240; 110: 160x120; 111: 保留
2 字节；010：CIF（352x288）；011：QCIF（176x144）；100：SQCIF（128x96）；101：320x240；110：160x120；111：保留	如果图像大小=000，则 UB[8]；如果图像大小=001，则 UB[16]；否则不存在。注意：UB[16]与 UI16 不同；没有字节交换。	像素宽度
自定义高度	如果图像大小为 000，则 UB[8]；如果图像大小为 001，则 UB[16]；否则不存在。注意：UB[16]与 UI16 不同；不存在字节交换。	像素高度
图片类型	UB[2]	00：帧内；01：帧间；10：可丢弃的帧间；11：保留
解块标志	UB[1]	请求使用解块过滤器（仅供参考，Flash Player 可能会忽略）
量化器	UB[5]	请参阅 H.263 5.1.4
额外信息标志	UB[1]	参见 H.263 5.1.9
额外信息	如果额外信息标志 = 1，则 UB[8]，否则不存在	查看 H.263 5.1.10
...		ExtraInformationFlagExtraInformation 序列会重复，直到遇到 ExtraInformationFlag 为 0
宏块	宏块	请见下文
图片填充	不同	请见 H.263 5.1.13

宏块

宏块是视频结构中的下一层。它对应于 H.263 第 5.3 节中的宏块层。

宏块		
字段	Type	评论
编码宏块标志	UB[1]	参见 H.263 5.3.1。如果为 1，宏块在此结束

宏块结束此处 宏块类型	不同	参见 H.263 5.3.2。可能导致各种字段（见下文）缺失
块模式	不同	参见 H.263 5.3.5
量化器信息	UB[2]	参见 H.263 5.3.6；00: -1；01: -2；10: +1；11: +2
运动矢量数据	变量[2]	参见 H.263 5.3.7。先水平编码后垂直编码
额外运动矢量数据	varies[6]	请参阅 H.263 5.3.8。当宏块类型为 INTER4V 时，包含三个额外的运动矢量数据代码对
块数据	BLOCKDATA[6]	请参阅 H.263 5.4 节。四个亮度块之后跟随两个色度块

块数据

块数据是视频结构的最低层。在 Sorenson H.263 视频格式的第 0 版本中，这一层严格遵循 H.263 第 5.4 节。

在 Sorenson H.263 视频格式的第 1 版本中，变换系数中的转义码（参见 H.263 第 5.4.2 节）的编码方式不同。当出现转义码 0000 011 时，下一个比特是一个格式比特，它指示后续的 LAST、RUN 和 LEVEL 的比特布局。在这两种情况下，一个比特用于 LAST，六个比特用于 RUN。如果格式比特为 0，则使用七个比特用于 LEVEL；如果格式比特为 1，则使用十一个比特用于 LEVEL。7 比特和 11 比特的 LEVEL 表，如以下表格所示，取代了 H.263 中的表 14：

7 比特 LEVEL				11 比特 LEVEL		
索引	级别	Code		索引	级别	Code
-	-64	禁止		-	-1024	禁止
0	-63	1000 001		0	-1023	1000 0000 001
.	.	.		.	.	.
61	-2	1111 110		1021	-2	1111 1111 110
62	-1	1111 111		1022	-1	1111 1111 111
-	0	禁止		-	0	禁止
63	1	0000 001		1023	1	0000 0000

						001
64	2	0000 010		1024	2	0000 0000 010
.	.	.		.	.	.
125	63	0111 111		2045	1023	0111 1111 111

屏幕视频比特流格式

自 SWF 7 起，新增了一种视频格式，称为屏幕视频。屏幕视频是一种简单的无损顺序位图格式，具有阻塞帧间插帧。它旨在发送计算机屏幕动作的捕获。

屏幕视频格式中的像素数据使用 ZLIB 开放标准进行压缩。

块格式

屏幕视频序列中的每一帧都格式化为一系列块。这些块在图像上形成一个网格。在关键帧中，会发送每个块的信息。在帧间，可能会有与前一帧相同的块，并且可以发送特殊信息来表示这一点。

块的宽度和高度从 16 到 256，以 16 的倍数递增。块的高度不必与块宽度匹配。块大小必须在关键帧处才能改变。

块从图像的左下角到右上角按顺序排列，按行排列。对于任何给定的块大小和图像大小的组合，都存在一个固定的块布局。要确定网格中一行的块数，将图像宽度除以块宽度。如果结果不是整数，则每行的末尾有一个部分块，它只包含填充图像剩余宽度的像素数。相同的逻辑适用于图像高度、块高度、网格中的行数以及最后一行的部分块。理解部分块算法对于创建正确的块非常重要，因为部分块中的像素是通过隐含的块宽度和高度知识提取的。

以下是一个阻塞示例。此示例中的图像大小为 120 x 80 像素，块大小为 32 x 32.

#9 32 x 16	#10 32 x 16	#11 32 x 16	#12 24 x 16
#5 32 x 32	#6 32 x 32	#7 32 x 32	#8 24 x 32
#1 32 x 32	#2 32 x 32	#3 32 x 32	#4 24 x 32

视频数据包

视频包是屏幕视频包中的顶级结构元素。这种结构包含在 SWF 文件格式中的 VideoFrame 标签内，也包含在 FLV 文件格式中的 VIDEODATA 结构内。

数据包括关于图像子块尺寸和网格大小的信息，然后是每个块的数据。

SCREENVIDEOPACKET		
字段	Type	评论
块宽度	UB[4]	网格中每个块的像素宽度。此值存储为（实际宽度 / 16）- 1，因此可能的块大小是 16 的倍数，且不超过 256。
图像宽度	UB[12]	整个图像的像素宽度
区块高度	UB[4]	网格中每个区块的像素高度。此值存储为（实际高度 / 16）- 1，因此区块大小为 16 的倍数，且不超过 256。
图片高度	UB[12]	完整图像的像素高度。
图像块。	IMAGEBLOCK[n]。	图像数据块。参见前面的说明，了解如何计算 n。块按从左下角到右上的顺序排列。

图像块

帧中的一个图像块代表一个块。

图像块		
字段	Type	评论
数据大小	UB[16] 注：UB[16] 与 UI16 不相同；不进行字节交换	下列压缩块数据的尺寸。如果是帧间图像，并且此块自上次关键帧以来未更改，则 DataSize 为 0，且 Data 字段不存在。
Data	如果 DataSize > 0，则 UI8[DataSize]	使用 ZLIB 压缩的像素数据。像素按从左下角到右上角的顺序排列。每个像素是三个字节：B、G、R。

屏幕视频 V2 比特流格式

SWF 支持新的屏幕视频格式，即屏幕视频版本 2，它是屏幕视频比特流格式的扩展，并在 Flash Player 8 及更高版本中得到支持。屏幕视频 v2 使用多种技术来减少每个屏幕块的数据量。

在初始的屏幕视频版本中，每个屏幕数据块是一个完整的压缩数据缓冲区，可以解压缩为该块的完整 24 位彩色图像。在屏幕视频 v2 格式中，屏幕数据块可以是图像区域的非完整更新，类似于关键帧和帧间概念。此外，v2 格式还引入了除常规的 24 位 RGB 色彩空间之外的混合 15/7 位色彩空间。15/7 位混合色彩空间对于编码具有少量独特颜色（少于 256）的图像非常有用。在屏幕视频 v2 格式中，块数据有两种类型：

关键块包含该块的全部信息。内容可以解压缩以获得完整的块图像。

间块需要从先前图像或当前图像中获取额外数据，以构建完整的块图像。

V2 颜色空间

The Screen Video v2 可以使用与 v1 相同的 24 位 RGB 颜色空间进行视频数据编码，或者使用 15/7 位混合颜色空间。使用后者颜色空间时，图像对应一个 128 个条目的调色板。解码图像中的每个像素由 1 个或 2 个字节表示。通常，解码器会将 15/7 位混合颜色空间转换为 24 位 RGB 颜色空间。转换过程如下：

- 从解码图像中获取下一个字节
- 如果下一个字节的高位被设置，则清除高位并从解码图像中获取下一个字节；通过将当前字节的低 7 位放置在颜色的 14-8 位，并将下一个字节的 8 位放置在颜色的 7-0 位，形成一个 15 位颜色；将 15 位 RGB 颜色转换为 24 位 RGB
- 如果下一个字节的高位被清除，则使用低 7 位作为索引到 128 个条目的调色板，并检索相应的 24 位 RGB 颜色

此过程如下：v2 视频包可以在任何时候定义一个新的调色板，这将以 v1 IMAGEBLOCK 的形式传输。在没有流定义的调色板的情况下，v2 解码器将回退到默认调色板。有关默认调色板，请参阅附录 C，“屏幕视频 v2 调色板。”

V2 视频包

Video Packet v2 是屏幕视频包的顶级结构元素，用于屏幕视频版本 2。该结构包含在 SWF 文件格式中的 VideoFrame 标签内，也包含在 FLV 文件格式中的 VIDEODATA 结构内。

数据包括关于图像子块尺寸和网格大小的信息，然后是每个块的数据。

SCREENV2VIDEOPACKET		
字段	Type	评论
块宽度	UB[4]	网格中每个块的像素宽度。此值存储为（实际宽度 / 16）- 1，因此可能的块大小是 16 的倍数，且不超过 256。
图像宽度	UB[12]	整个图像的像素宽度
区块高度	UB[4]	网格中每个区块的像素高度。此值存储为（实际高度 / 16）- 1，因此区块大小为 16 的倍数，且不超过 256。
图片高度	UB[12]	完整图像的像素高度。
保留	UB[6]	必须为 0
HasIFrameImage	UB[1]	如果为 1，则具有 IFrameImage
有调色板信息	UB[1]	如果为 1，则有调色板信息
调色板信息	如果有 HasPaletteInfo，一个数据块用于描述调色板。 图像块	

图像块。	IMAGEBLOCKV2[n]	图像数据块。请参阅块格式以了解如何计算 n 的详细信息。块按从左下角到右上的顺序排列，可以是关键块和中间块的组合。
IFrameImage	如果有 IFrameImage，IMAGEBLOCKV2[n]	表示需要与前面的关键块结合以生成图像的块数据。有关如何计算 n 的详细信息，请参阅块格式。 块按从左下角到右上角的顺序排列。

图像块 V2

图像块 v2 结构表示帧中的一个块。

IMAGEBLOCKV2		
字段	Type	评论
数据大小	UB[16] 注：UB[16] 与 UI16 不相同；没有字节交换。	以下压缩块数据的尺寸，包括 ImageFormat、ImageBlockHeader 和 Data 字段。如果是帧间图像，并且此块自上次关键帧以来没有更改，则 DataSize 为 0，且 Data 字段不存在。
格式	图像格式	块数据的压缩格式。
图像块头	如果格式中的 HasDiffBlock = 1，则 IMAGEDIFFPOSITION。如果格式中的 ZlibPrimeCompressCurrent = 1，则 IMAGEPRIMEPOSITION	描述数据的格式和压缩
Data	如果 DataSize > 0，UI8[DataSize]	使用 ZLIB 压缩的像素数据。像素按从左下角到右上角的顺序排列。每个像素是三个字节：B、G、R。

图片格式

IMAGEFORMAT 字节描述了 IMAGEBLOCKV2 结构的颜色深度和压缩方式。

字段	Type	评论

保留	UB[3]	必须为 0
颜色深度	UB[2]	00: 24 位 RGB 图像。10: 15/7 位混合颜色图像
有差异块	UB[1]	如果为 1，则数据从特定行开始和结束，在块内，并不代表整个块。
ZlibPrimeCompressCurrent	UB[1]	如果为 1，则当前数据块是使用 ZLIB 预压缩技术生成的。
ZlibPrimeCompressPrevious	UB[1]	如果为 1，则上一个数据块是使用 ZLIB 预压缩技术生成的。

图像块差异位置

图像块差异位置可以包含在 IMAGEBLOCKV2 ImageBlockHeader 字段中。此结构描述了差异块图像数据的位置和大小。

IMAGEDIFFPOSITION		
字段	Type	评论
行起始	UI8	表示包含图像数据的块的第一扫描行。
高度	UI8	表示图像数据的高度，以连续扫描行计。

图像块主位置

图像块主位置包含在 IMAGEBLOCKV2 ImageBlockHeader 字段中，如果 IMAGEFORMAT 结构指示使用了 ZLIB 预置。此结构指定了用作预置源的图像。

图像优先位置		
字段	Type	评论
区块列	Ui8	指示源块的位置。
区块行	Ui8	指示源块的位置。

On2 Truemotion VP6 比特流格式

SWF 6 或更高版本支持 On2 Truemotion VP6 视频格式，可在 Flash Player 8 及更高版本中播放。VP6 是一种前沿的视频压缩算法，它结合了传统的运动补偿预测和伪离散余弦变换（DCT）编码以及基于上下文的熵编码技术（基于

VP6 是一种领先的视频压缩算法，它结合了传统的运动补偿预测和伪离散余弦变换（DCT）编码以及基于霍夫曼和算术原理的上下文相关熵编码技术（基于霍夫曼和算术原理），并采用新颖的方法超越其他编解码器的质量。VP6 应用广泛的上下文建模、循环内和循环外滤波以及新颖的量化方法，以达到高质量水平。有关此视频格式的更多信息，可从 On2 获取

与 Sorenson H.263 视频格式类似，On2 Truemotion VP6 视频格式使用 ITU-R BT.601 标准中描述的 YCbCr 颜色空间进行颜色信息编码。这种颜色信息以 YUV 4:2:2 格式存储，每个颜色分量使用无符号 8 位值。

函数 SATURATE(A) = MIN(255,MAX(0,A)) Y = SATURATE(+ (0.257 * R) + (0.504 * G) + (0.098 * B) + 16) U = SATURATE(- (0.148 * R) - (0.291 * G) + (0.439 * B) + 128) V = SATURATE(+ (0.439 * R) - (0.368 * G) - (0.071 * B) + 128)

此算法可用于将 YUV 颜色空间像素数据转换回 RGB 颜色空间：

B = SATURATE(1.164(Y - 16) + 2.018(U - 128)) G = SATURATE(1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128)) R = SATURATE(1.164(Y - 16) + 1.596(V - 128))

除了支持标准的 On2 Truemotion VP6 视频流外，Flash Player 8 还增加了对额外 alpha 通道的支持，用于模拟透明度。alpha 通道通过使用包含 alpha 通道信息的第二个 On2 Truemotion VP6 流进行编码。为了编码此类视频流，应使用先前的 RGB-to-YUV 算法对预乘的 ARGB 颜色空间像素数据的 RGB 颜色通道进行编码。使用得到的 YUV 颜色空间数据，可以将视频数据编码为第一个视频流。

对于第二个视频流，可以使用以下算法从预乘的 ARGB 颜色空间像素数据的 alpha 通道中获取 YUV 视频数据：

Y = A U = 0
V = 0

注意：在编码时，第二个视频流必须至少包含与第一个视频流相同数量的关键帧。第一个视频流中出现的每个关键帧都必须在编码时强制第二个视频流出现关键帧，以确保组合视频流可寻址。

解码 alpha 通道视频流时，假设第一个视频流返回 YUV 编码的颜色通道，以屏幕像素的形式为三个通道，分别命名为 Y1、U1 和 V1。第二个视频流返回的数据为 Y2、U2 和 V2。通过以下算法获得 alpha 预乘的 ARGB 像素值：

B = MIN(Y2, SATURATE(1.164(Y1 - 16) + 2.018(U1 - 128))) G = MIN(Y2, SATURATE(1.164(Y1 - 16) - 0.813(V1 - 128) - 0.391(U1 - 128))) R = MIN(Y2, SATURATE(1.164(Y1 - 16) + 1.596(V1 - 128))) A = Y2

U2 和 V2 通道在解码时目前未使用。

VP6 FLV 视频包

VP6 FLV 视频包表示 FLV 文件中的 VP6 视频帧。

VP6FLV 视频包		
字段	Type	评论
水平调整	UB[4]	从总宽度中减去的像素数。得到的宽度用于舞台，视频的最右侧像素将被裁剪。
垂直调整	UB[4]	从总高度中减去的像素数。得到的宽度用于舞台，视频最右侧的像素将被裁剪。
Data	UI8[n]	原始 VP6 视频流数据

VP6 FLV Alpha 视频数据包

VP6 FLV Alpha 视频包表示在 FLV 文件中带有 alpha 通道信息的 VP6 视频帧。

VP6FLVALPHAVIDEOPACKET		
字段	Type	评论
水平调整	UB[4]	从总宽度中减去的像素数。得到的宽度用于舞台，视频最右侧的像素将被裁剪。
垂直调整	UB[4]	从总高度中减去的像素数。得到的宽度用于舞台，视频最右侧的像素将被裁剪。
Alpha 通道视频数据的偏移量	UI24	Alpha 通道视频数据偏移的字节数
Data	UI8[偏移到 Alpha]	表示颜色通道的原始 VP6 视频流数据
Alpha 数据	UI8[n]	表示 alpha 通道的原始 VP6 视频流数据

VP6 SWF 视频包

VP6 SWF 视频数据包表示 SWF 文件中的 VP6 视频帧。

VP6SWFVIDEOPACKET		
字段	Type	评论
Data	UI8[n]	原始 VP6 视频流数据

VP6 SWF 透明度视频数据包

VP6 SWF 透明度视频数据包表示带有透明度通道信息的 VP6 视频帧，在 SWF 文件中。

VP6SWFALPHAVIDEOPACKET		
字段	Type	评论
Alpha 通道视频数据的偏移量	UI24	Alpha 通道视频数据偏移的字节数
Data	UI8[偏移到 Alpha]	表示颜色通道的原始 VP6 视频流数据
Alpha 数据	UI8[n]	表示 alpha 通道的原始 VP6 视频流数据

SWF 视频标签

以下标签定义了 SWF 文件中的嵌入式视频数据。这些标签仅限于 SWF 6 或更高版本中使用。

SWF 文件中嵌入的视频始终是流式传输的：视频帧位于与它们时间关联的 SWF 帧中，视频播放可以在整个视频流下载完毕之前开始。这个过程与流式声音的定义方式类似（参见流式声音）。

定义视频流

DefineVideoStream 定义了一个视频角色，该角色可以稍后放置在显示列表中（参见显示列表）。

DefineVideoStream		
字段	Type	评论
标题	记录头	标签类型 = 60
角色 ID	UI16	该视频角色的 ID
帧数	UI16	组成此流的 VideoFrame 标签数量
宽度	UI16	像素宽度
高度	UI16	像素高度
视频标志保留	UB[4]	必须为 0
视频标志去块处理	UB[3]	000 = 使用 VIDEOPACKET 值：； 001 = 关闭； 010 = 级别 1（快速去块滤波器）； 011 = 级别 2（VP6 仅用，更好的去块滤波器）； 100 = 级别 3（VP6 仅用，更好的去块加快速去伪影滤波器）； 101 = 级别 4（VP6 仅用，更好的去块加更好的去伪影滤波器）； 110 = 保留； 111 = 保留
视频平滑标志	UB[1]	0 = 关闭平滑（更快）； 1 = 开启平滑（更高质量）
编解码器 ID	UI8	2 = Sorenson H.263；3 = 屏幕视频（仅限 SWF 7 及以后版本）；4 = VP6（仅限 SWF 8 及以后版本）；5 = 带 alpha 通道的 VP6 视频（仅限 SWF 8 及以后版本）

视频帧

VideoFrame 提供已通过 DefineVideoStream 定义的视频角色的单个视频帧数据。

在播放过程中，视频帧的时间顺序仅取决于 SWF 帧率。当 SWF 播放达到特定的 SWF 帧时，该 SWF 帧中任何 VideoFrame 标签的视频图像将被渲染。

视频负载中内置的任何定时机制都将被忽略。

并不是每个视频字符在每个指定的帧号都需要 VideoFrame 标签。VideoFrame 标签仅设置与特定帧号关联的视频数据；它不会自动显示视频帧。要显示视频帧，请在 PlaceObject2 或 PlaceObject3 中将帧号指定为 Ratio 字段。

视频帧		
字段	Type	评论
标题	记录头	标签类型 = 61
流 ID	UI16	该帧所属的视频流字符的 ID
帧编号	UI16	该帧在视频流中的序列号
视频数据	如果 CodecID = 2 H263VIDEOPACKET 如果 CodecID = 3 SCREENVIDEOPACKET 如果 CodecID = 4 VP6SWFVIDEOPACKET 如果 CodecID = 5 VP6SWFALPHAVIDEOPACKET 如果 CodecID = 6 SCREENV2VIDEOPACKET	视频帧有效载荷

第 15 章：元数据

从版本 9 开始，SWF 文件格式规范支持包含任意二进制数据块。

文件属性

如果使用 FileAttributes 标签，它必须紧接在 SWF 头部之后。它定义了一组位字段，用于配置 Flash Player。因此，它仅在根 SWF 上被解释。

字段	Type	评论
标题	记录头	标签类型 = 69
保留	UB[3]	必须为 0
有元数据	UB[1]	如果设置为真，SWF 中包含某些位置的元数据。这些元数据被播放器忽略，但被搜索引擎使用。
SWF 标志 AS3	UB[1]	如果设置，则 SWF 包含 AVM-2（Tamarin）字节码。
SWFFlagsNoCrossDomainCache	UB[1]	如果设置，则此 SWF 永远不会被放置在跨域缓存中。
保留	UB[1]	必须为 0
SWFFlagsUseNetwork	UB[1]	如果这是一个从本地驱动器运行的 SWF，请给它网络访问权限，而不是本地访问权限。

启用遥测

遥测是 Flash 播放器的一项功能，用于发送有关运行时和当前运行内容的配置文件信息。EnableTelemetry 标签控制是否将遥测的高级功能包含在配置文件数据中。如果该标签不存在，则只有基本信息可用。

字段	Type	评论
标题	记录头	标签类型 = 93
保留	UB[16]	必须为 0
密码散列	UI8[32]	可选：密码的 UTF-8 表示的 SHA-256 哈希值。如果不存在，SWF 将选择高级遥测。

定义二进制数据

DefineBinaryData 标签允许将任意二进制数据嵌入到 SWF 文件中。DefineBinaryData 是一个定义标签，类似于 DefineShape 和 DefineSprite。它将一个二进制数据块与标准的 SWF 16 位字符 ID 关联。字符 ID 被输入到 SWF 文件的字符字典中。

DefineBinaryData 旨在与 SymbolClass 标签一起使用。SymbolClass 标签可以用来将 DefineBinaryData 标签与 AS3 类定义关联起来。AS3 类必须是 ByteArray 的子类。当类被实例化时，它将自动填充二进制数据资源的内容。

字段	Type	评论
标题	记录头	标签类型 = 87
Tag	UI16	16 位字符 ID
保留	U32	保留空间；必须为 0
Data	二进制	一个二进制数据块，直到标签结束

附录 A：SWF 揭秘：一个简单的 SWF 文件剖析

要编写 SWF 文件，您必须能够读取和理解原始的比特和字节。本附录检查了一个简单的单帧 SWF 文件，该文件仅包含一个矩形。

下面是 SWF 文件的十六进制转储：

```
000000    46  57  53  03  4F  00  00  00    78  00  05  5F  00  00  0F  A0
000010    00  00  0C  01  00  43  02  FF    FF  FF  BF  00  23  00  00  00
000020    01  00  70  FB  49  97  0D  0C    7D  50  00  01  14  00  00  00
000030    00  01  25  C9  92  0D  21  ED    48  87  65  30  3B  6D  E1  D8
000040    B4  00  00  86  06  06  01  00    01  00  00  40  00  00  00
```

SWF 文件始终以头部开始。它描述了文件版本、文件长度（以字节为单位）、帧大小（以 twips 为单位，即像素的二十分之一）、帧率（每秒帧数）和帧数。

类型在第一章“基本数据类型”中定义，见第 14 页。

SWF 文件头		
字段	Type	评论
签名	UI8	签名字节：“F”表示未压缩；“C”表示压缩（仅限 SWF 6 或更高版本）使用 ZLib 压缩。“Z” 表示使用 LZMA 压缩。
签名	UI8	签名字节始终为“W”
签名	UI8	签名字节始终为“S”
版本	UI8	单字节文件版本（例如，0x06 表示 SWF 6）
文件长度	UI32	整个文件的长度（字节）
帧大小	RECT	帧大小（twips）
帧率	UI16	每秒 8.8 固定帧数的帧延迟
帧数	UI16	文件中帧的总数

前三个字节是所有 SWF 文件的标准签名。它们是字符‘F’（或‘C’）、‘W’和‘S’的 ASCII 值，顺序如下。第四个字节表示文件的版本。

0x46 ➜ ‘F’ 0x57 ➜ ‘W’ 0x53 ➜ ‘S’ 0x03 ➜ 3

下面的四个字节表示一个无符号 32 位整数，表示文件大小。从这里开始变得复杂，涉及到机器架构。接下来的四个字节是 0x4F000000，这意味着文件长度是 1325400064 字节，这是一个非常大的数字，没有意义。我们未能做到的是交换所有字节。

在 SWF 文件中，读取单词和 dwords 时字节会进行交换，32 位值 B1B2B3B4 写入为 B4B3B2B1，16 位值 B1B2 写入为 B2B1。单字节写入时不进行位交换，因为没有位交换。这样做的原因是 Mac 和 PC 处理器在存储和检索上的差异。

逆序字节后，我们可以正确读取四个字节，并看到文件长度为 79 字节。

0x4F000000 ➜ 0x0000004F ➜ 79

下九个字节代表 SWF 格式中使用的矩形数据结构。以下是矩形的文件描述：

RECT		
字段	Type	评论
比特数	UB[5]	每个矩形值字段中的比特数
Xmin	SB[比特数]	矩形的最小位置 x
Xmax	SB[比特数]	矩形的最大位置 x
Ymin	SB[比特数]	矩形的最小位置 y
Ymax	SB[比特数]	矩形的最大位置 y

为了理解这些字节，我们需要查看单个比特。

```
78    00    05    5F 00     00    0F A0 00
          ⇩
0111  1000    0000 0000      0000 0101      0101 1111      0000 0000
0000  0000    0000 1111      1010 0000      0000 0000
```

矩形结构中有五个字段：Nbits、Xmin、Xmax、Ymin、Ymax。无符号的 Nbits 字段占据了矩形的第一个五位，表示接下来的四个有符号字段有多长。

SWF 文件表示的另一个微妙之处在于，读取和写入位与读取和写入字和双字不同。在读取和写入位时，不会发生字节交换。这是因为当 Flash Player 读取一个 n 位字段时，它会一次读取一个字节，直到读取完所有 n 位。因此，接下来的五个

接下来的五个位依次读取，结果为 15。

01111 ➜ 15

如果 Nbit 的值为十六，这正好是一个字的大小，那么我们是否将后续字段作为字读取并交换字节？不。由位大小描述的字段始终按字节读取。不需要交换，只需按顺序读取下一个 n 位。

000000000000000 ‹ 0 = Xmin

010101011111000 ‹ 11000 = Xmax

000000000000000 ‹ 0 = Ymin

001111101000000 ‹ 8000 = Ymax

对于标题，使用矩形来存储文件尺寸，其中 Xmax 对应文件宽度，Ymax 对应文件高度，两者均以 twips 为单位。在 SWF 格式中，1 twip 等于像素的二十分之一，因此如果我们将其转换为像素，可以看到我们的文件大小为 550 x 400。

现在我们已经查看并评估了矩形的所有字段，那么那些全部为 0 的最后七个比特怎么办？这些比特被填充为 0，以便结构对齐到字节边界。

0000000 = 填充比特

在任何结构的末尾之后，如果该结构没有完全填满其最后一个字节，那么该最后一个字节将被填充为 0，以保持下一个项目的字节对齐。因此，如果下一个项目是一个字或双字，您可以将其读取为这样的，而不用担心处于字节中间。在这种情况下，最后一个字节中只使用了一个比特，所以最后七个比特被填充为 0。

接下来在头部是帧率。它应该存储为一个 16 位整数，但第一个字节（或者根据您如何看待它，是最后一个字节）被完全忽略。因此，帧率是

12 帧/秒。

0x000C ➜ 0x0C00 ➜ 0x0C ➜ 12 = 帧率

接下来是帧数，它也是一个 16 位整数。因此，帧数为 1。

0x0100 ➜ 0x0001（字节交换）➜ 1 = 帧数

现在我们已经完成了头部。头部之后是一系列标记数据块。以下是关于标记的描述（这里有些简化；需要进行字节交换）：

RECORDHEADER（短格式）		
字段	Type	评论
标签代码和长度	UI16	上 10 位：标签类型；下 6 位：标签长度

RECORDHEADER（长）		
字段	Type	评论
标签代码和长度	UI16	将 0x3F 的标签类型和长度打包在一起，如短头中所示
长度	UI32	标签的长度

标签有两种类型：短标签头和长标签头。无论哪种情况，你都是从第一个单词开始的。

0x4302 ➜ 0x0243 ➜ 0000 0010 0100 0011

标签的前 10 位是未签名的标签类型。标签类型指示数据块中要跟随的数据类型。在这种情况下，标签类型的值为 9，对应于 SetBackgroundColor 块。标签头部的最后 6 位未签名位表示跟随的数据块长度，如果长度不超过 62 字节。如果数据块长度超过 62 字节，则此字段全部为 1，长度由随后的 dword 指示。对于我们要查看的标签，此字段不是全部为 1，因此它确实表示实际长度，该长度为 3 字节。

0000001001 = 9 = SetBackgroundColor000011 = 3 = 主体长度

由于我们知道主体长度为 3 字节，让我们来看看它。SetBackgroundColor 标签只包含 3 字节的 RGB 颜色描述，所以我们将其评估为这样。颜色是其自己的 3 字节数据类型，因此没有字节交换。

0XFFFFFF = 白色

下一个标签是一个长标签，它是一个 DefineShape 标签。

48896 191 0000 0000 1011 1111

0000000010 = 2 = DefineShape

224

111111 = 63 = 身体长度（表示身体长度值位于下一个 dword 中）

587202560　　35　　35 = 主体长度

这里是 DefineShape 的文件描述

定义形状		
字段	Type	评论
标题	记录头	标签类型 = 2
形状 ID	UI16	该字符的 ID
形状边界	RECT	形状边界
形状。	SHAPEWITHSTYLE	形状信息

DefineShape 的主体由一个无符号 16 位字符 ID、定义形状边界的矩形以及包含形状信息的 SHAPEWITHSTYLE 结构组成。

0x0100 ➜　　0x0001 ➜　　1 = 形状 ID

现在是矩形，它定义了边界：

70 FB 49 97 0D 0C 7D 50

⇩

0111 0000 1111 1011 0100 1001 1001 0111 0000 1101　　0000

1100 0111 1101 0101 0000

01110 = 14 = Nbits

00011111011010 = 2010 = Xmin　　/20 转换为像素（从 twips）　　100.5

01001100101110 = 4910 = Xmax　　245.5

00011010000110 = 1670 = 最小 Y 值　　83.5

00111110101010 = 4010 = 最大 Y 值　　200.5

000 = 填充位

以下表格描述了 SHAPEWITHSTYLE 结构：

SHAPEWITHSTYLE		
字段	Type	评论
填充样式	填充样式数组	填充样式数组
线样式	线样式数组	线样式数组
填充索引位数	UB[4]	填充索引位数
线索引位数	UB[4]	线索引位数
形状记录	SHAPERECORD[一个或多个]	形状记录（见下文）

填充样式数组本身有三个字段。第一个字段是一个 8 位整数计数，表示数组中有多少个填充样式。这个计数与标签的长度字段类似，如果全部为 1，则需要查看下一个 16 位才能得到实际长度。以下是文件描述：
FILLSTYLEARRAY

字段	Type	评论
填充样式数量	UI8	填充样式的计数
FillStyleCountExtended	如果 FillStyleCount = 0xFF，则使用 UI16 扩展填充样式的计数。 仅支持 Shape2 和 Shape3。	
填充样式	FILLSTYLE[FillStyleCount]	填充样式数组

在这种情况下，8 位计数等于 0，因此没有后续内容。

0x00 = 0 = 填充样式计数 ➜ 这是填充样式数组的结束，因为它没有元素

线样式数组与填充样式数组完全相同，只是它存储的是线样式。以下是文件描述：

LINESTYLEARRAY		
字段	Type	评论
线样式数量	UI8	线样式的数量
线样式数量扩展	如果 LineStyleCount = 0xFF UI16	扩展线样式的数量

线样式	LINESTYLE[count]	线样式数组

0x01 = 1 = LineStyleCount ➜ 数组中有一个线条样式。

一个线条样式由两部分组成，一个表示线条宽度的无符号 16 位整数和一个颜色。以下是文件描述：

线型		
字段	Type	评论
宽度	UI16	线条宽度的 twips 值
颜色	RGB（形状 1 或形状 2）RGBA（形状 3）	包含 Alpha 通道信息的颜色值，用于形状 3

在这种情况下，颜色是 24 位 RGB，但如果进行 DefineShape3 操作，则为 32 位 RGBA，其中 Alpha 表示颜色的不透明度

0x1400 ➜ 0x0014 = 20 = 宽度 = 1 像素

0x000000 = RGB = 黑色

返回到 ShapeWithStyle，NumFillBits 字段是 4 位，NumLineBits 也是 4 位。

0x0 = 0 = NumFillBits 0x1 = 1 = NumLineBits

现在是形状记录的数组。以下四个表格描述了四种类型的形状记录。以下是文件描述：

ENDSHAPERECORD		
字段	Type	评论
类型标志	UB[1]	非边缘记录标志。始终为 0
形状结束标记	UB[5]	形状结束标志。始终为 0

样式更改记录		
字段	Type	评论

类型标志	UB[1]	非边缘记录标志，始终为 0
StateNewStyles	UB[1]	新样式标志。仅由 DefineShape2 和 DefineShape3 使用。
StateLineStyle	UB[1]	线型更改标志
状态填充样式 1	UB[1]	填充样式 1 更改标志
状态填充样式 0	UB[1]	填充样式 0 更改标志
状态移动到。	UB[1]	移动到标志
移动位。	如果是 StateMoveTo，UB[5]	移动位数
MoveDeltaX	如果状态移动到，SB[移动位数]	ΔX 值
移动 ΔY	如果 StateMoveTo，SB[移动位]	ΔY 值
填充样式 0	如果状态填充样式 0，UB[填充位]	填充 0 样式
填充样式 1	如果状态填充样式 1，填充位	填充 1 样式
线样式	如果是线条样式状态，线位填充	线型样式
填充样式	如果 StateNewStyles，填充样式数组	新填充样式的数组
线样式	如果 StateNewStyles，LINESTYLEARRAY	新行样式数组
填充索引位数	如果 StateNewStyles，UB[4]	新样式的填充索引位数
线索引位数	如果 StateNewStyles，UB[4]	新样式的线条索引位数

STRAIGHTEDGERECORD		
字段	Type	评论
类型标志	UB[1]	这是一个边缘记录。总是 1。

直线标志	UB[1]	直边。总是 1。
比特数	UB[4]	每个值的位数（比实际位数少两个）
通用线标志	UB[1]	通用线等于 1。垂直/水平线等于 0。
DeltaX	如果是 GeneralLineFlag，则 SB[NumBits+2]	X 增量
ΔY	如果是 GeneralLineFlag，则 SB[NumBits+2]	Y 方向增量
垂直线标志	如果设置 GeneralLineFlag，则 SB[1]	垂直线等于 1。水平线等于 0。
DeltaX	如果设置 VertLineFlag，则 SB[NumBits+2]	X 增量
ΔY	如果设置 VertLineFlag，则 SB[NumBits+2]	Y 方向增量

曲线边缘记录		
字段	Type	评论
类型标志	UB[1]	这是一个边缘记录。总是 1。
直线标志	UB[1]	曲边。始终为 0。
比特数	UB[4]	每个值所用的位数（比实际数值少两个）
控制 DeltaX	SB[NumBits+2]	X 控制点变化
控制 Y 增量	SB[NumBits+2]	Y 控制点变化
锚点 X 增量	SB[NumBits+2]	X 锚点变化
AnchorDeltaY	SB[NumBits+2]	Y 锚点变化

ENDSHAPERECORD 定义了形状记录数组的结束。STYLECHANGERECORD 定义了线样式、填充样式、位置或一组新样式的更改。STRAIGHTEDGERECORD 和 CURVEDEDGERECORD 分别定义了直线或曲线边缘。形状记录的第一个位是一个类型标志。0 表示非边缘记录，1 表示边缘记录。查看我们第一个形状记录的第一个位，我们看到它不是一个边缘记录。现在我们必须查看下一个五位，这些位都是标志，告诉我们接下来是什么。如果这五个位都是 0，那么这是一个类型 0 的形状记录，并定义了形状记录数组的结束。

25 C9 92 0D 21

0010 0101 1100 1001 1001 0010 0000 1101 0010 0001

0 = 0 = 非边缘记录

01001 = 5 标志行样式标志为真，并且移动标志为真

由于移动标志为真，接下来的五个位是 MoveBits 字段。此值为 14，因此下一个两个字段，即 MoveDeltaX 和 MoveDeltaY 的大小为 14。这些都是无符号数。

01110 = 移动位

01001100100100 = 4900 (twips) = 245 像素 = X 方向移动增量

00011010010000 = 1680 = 84 像素 = Y 方向移动增量

由于线条样式标志为真，下一个字段是 NumLineBits = 1 位字段，表示线条样式。此字段等于 1。这意味着后续线条的线条样式是线条样式数组中的第一个。

1 = 1 = 线条样式

接下来是其余的形状记录：

ED 48 87 65 30 3B 6D E1 D8 B4 00 00

1110 1101 0100 1000 1000 0111 0110 0101 0011 0000 0011 1011 0110
1101 1110 0001 1101 1000 1011 0100 0000 0000 0000 0000

下一个形状记录以 1 开始，因此它是一个边缘记录。

下一个位表示是直线还是曲线边缘。它是 1，代表直线边缘。接下来的四个位表示任何后续的 delta 字段的大小。NumBits 值的公式是 2 加上那个 4 位字段的值。在这种情况下，NumBits 的值是 13。NumBits 字段之后是一个 1 位线标志。这表示所描述的线是普通线还是水平/垂直线。0 的值对应于水平/垂直线，所以下一个位是 VertLineFlag 字段，表示线是水平还是垂直。该位的值是 1，对应于垂直线。垂直线的下一个字段是带符号的 DeltaY 字段，长度为 nbits = 13 位。该值对应于

116 像素。这就是形状记录的结束。

1 = 1 = 边缘记录

1 = 1 = 直线边缘

1011 = 11 + 2 = 13 = NumBits

0 = 0 = 水平/垂直线

1 = 1 = 垂直线

0100100010000 = 2320 twips = 116 像素 = DeltaY 下三条记录与最后一条非常相似：

1 = 1 = 边缘记录

1 = 1 = 直线边缘

1011 = 11 + 2 = 13 = NumBits

0 = 0 = 水平/垂直线

0 = 0 = 水平线

1010011000000 = -2880 twips (2 的补码数) = -144 像素 = DeltaX

1 = 1 = 边缘记录

1 = 1 = 直线边缘

1011 = 11 + 2 = 13 = NumBits

0 = 0 = 水平/垂直线

1 = 1 = 垂直线

1011011110000 = -2320 twips = -116 pixels = DeltaY

1 = 1 = 边缘记录

1 = 1 = 直线边缘

1011 = 11 + 2 = 13 = NumBits

0 = 0 = 水平/垂直线

0 = 0 = 水平线

0101101000000 = 2880 twips = 144 像素 = DeltaX

最后，最后一个形状记录以 0 开始，这意味着它不是边缘记录。此外，它的所有标志位都等于 0，这意味着它是最后一个形状记录，我们已经完成了形状记录数组。

0 = 0 = 非边缘记录

000000 = 标志（由于它们都是 0，这是形状记录数组的结束）

这是形状记录数组的结束，由于我们已经完成了我们的结构，现在我们必须用 0 填充最后一个字节以保持字节对齐。

000000 = 填充位，以便记录对齐到字节边界

我们也完成了具有样式的形状，因为具有样式的形状记录数组是具有样式的形状的最后一个元素。由于我们已经字节对齐，我们可以继续到下一个标记数据块。

块的标签类型等于 26，对应于 PlaceObject2。长度字段值为 6，因此数据块的长度为 6 个字节。

0x8606 ➜ 0x0686 ➜ 0000 0110 1000 0110

0000011010 = 26 = 标签类型 = PlaceObject2

000110 = 6 = 长度

06 01 00 01 00 00

⇩

0000 0110 0000 0001 0000 0000 0000 0001 0000 0000 0000 0000

这里是 PlaceObject2 标签的文件描述：

PlaceObject2		
字段	Type	评论
标题	记录头	标签类型 = 26。
PlaceFlagHasClipActions	UB[1]	SWF 5 或更高版本：具有剪辑动作（仅限精灵角色）。否则：始终为 0。
具有剪辑深度标志	UB[1]	具有剪辑深度。
具有名称标志	UB[1]	具有名称。
标记具有比例	UB[1]	具有比例
标记具有颜色变换	UB[1]	具有色变换
放置标志具有矩阵	UB[1]	具有矩阵
标记放置有角色	UB[1]	放置一个角色
标记移动	UB[1]	定义要移动的字符。

深度	UI16	角色深度
字符 ID	如果 PlaceFlagHasCharacter UI16	要放置的角色的 ID
矩阵	如果 PlaceFlagHasMatrix 矩阵	变换矩阵数据。
颜色变换	如果 PlaceFlagHasColorTransform CXFORMWITHALPHA	颜色变换数据。
比率	如果 PlaceFlagHasRatio UI16	
Name	如果 PlaceFlagHasName 字符串，则显示角色的名称。	
裁剪深度	如果 PlaceFlagHasClipDepth UI16	剪裁深度（见剪裁层）。
影片动作	如果 PlaceFlagHasClipActions CLIPACTIONS	SWF 5 或更高版本：剪辑动作数据。

主体前八个位都是标志，表示接下来是什么。第六位为 1 表示主体有一个变换矩阵，第七位为 1 表示要放置的对象有一个字符 ID。

00000110➜ 主体有变换矩阵且对象有字符 ID

标志之后是一个 16 位无符号整数，表示字符的深度。在这种情况下，深度为 1，这在文件中只有一个矩形对象时是有意义的。

0x0100 ➜ 0x0001 ➜ 深度 = 1

由于对象具有字符 ID，因此体中的下一个字段是无符号 16 位 ID。由于矩形是文件中的唯一对象，因此矩形的 ID 为 1。

0x0100 ➜ 0x0001 ➜ 字符 ID = 1

PlaceObject2 的最后一个字段是这个变换矩阵。以下是文件描述：

矩阵		
字段	Type	评论
有缩放	UB[1]	如果等于 1，则具有缩放值。

NScaleBits	如果 HasScale = 1，则 UB[5]	每个尺度值字段中的比特数。
水平缩放比例	如果 HasScale = 1，则 FB[NScaleBits]	X 尺度值。
Y 轴缩放	如果 HasScale = 1，则 FB[NScaleBits]	Y 尺度值。
有旋转	UB[1]	如果等于 1，则具有旋转和倾斜值。
NRotateBits	如果 HasRotate = 1，则 UB[5]	每个旋转值字段中的比特数。
RotateSkew0	如果 HasRotate = 1，FB[NRotateBits]	首个旋转和倾斜值。
旋转倾斜 1	如果 HasRotate = 1，FB[NRotateBits]	第二次旋转和倾斜值。
NTranslateBits	UB[5]	每个平移值字段中的位数。
TranslateX	SB[NTranslateBits]	在 twips 中转换值。
翻译 Y	SB[NTranslateBits]	在 twips 中转换 y 值。

由于此形状没有转换信息，矩阵为空。所有标志位均为零。这并不特别高效，但它是有效的。

0x00 ➜ 完全为空的矩阵，剩余位填充

我们已经完成了 PlaceObject2，现在让我们看看下一个标签。

0x4000 ➜ 0x0040 ➜ 0000 0000 0100 0000

标签类型 = 1 = 显示帧长度 = 0

我们看到这个标签是一个显示帧的指令。ShowFrame 没有主体。它的长度为 0，所以我们继续到下一个标签。

0x0000 0x0000 0000 0000 0000 0000

标签类型 = 0 = 结束长度 = 0

我们已经到达了结束标签，这标志着我们的 SWF 文件结束。

附录 B：标签值反向索引

此表提供快速查找，允许通过标签值找到 SWF 规范中的任何标签。

Tag 值	标签名
0	End
1	显示帧
2	定义形状
4	PlaceObject
5	RemoveObject
6	定义位图
7	DefineButton
8	JPEG 表格
9	设置背景颜色
10	DefineFont
11	定义文本
12	执行动作
13	定义字体信息
14	定义声音
15	StartSound
17	定义按钮声音
18	音频流头部
19	音流块
20	定义无损位图字符
21	定义位 JPEG2
22	定义形状 2

23	定义按钮颜色转换
24	保护
26	PlaceObject2
28	RemoveObject2
32	DefineShape3
33	定义文本 2
34	定义按钮 2
35	定义位 JPEG3
36	定义无损位块
37	DefineEditText
39	定义精灵
43	帧标签
45	音频流头部 2
46	定义形态形状
48	DefineFont2
56	导出资源
57	导入资源
58	启用调试器
59	初始化动作
60	DefineVideoStream
61	视频帧
62	DefineFontInfo2
64	启用调试器 2
65	脚本限制
66	设置 TabIndex

69 文件属性

70 放置对象 3

71 导入资产 2

73 定义字体对齐区域

74 CSMTextSettings

75 DefineFont3

76 符号类

77 元数据

78 定义缩放网格

82 DoABC

83 定义形状 4

84 定义形态形状 2

86 定义场景和帧标签数据

87 定义二进制数据

88 定义字体名称

89 StartSound2

90 定义位图 JPEG4

91 DefineFont4

93. 启用遥测

附录 C：屏幕视频 v2 调色板

屏幕视频 v2 编解码器（见第 14 章“视频”），除了 24 位色空间外，还可以使用 15/7 位混合色空间。该编解码器允许比特流编码 128 个颜色的调色板。在没有有效调色板的情况下，代码将回退到以下 128 个条目的 RGB 调色板。

这些 32 位数字的格式为 0x00rrggbb。

```
unsigned int default_screen_video_v2_palette[128] = { 0x00000000,
0x00333333, 0x00666666, 0x00999999, 0x00CCCCCC, 0x00FFFFFF,
0x00330000, 0x00660000, 0x00990000, 0x00CC0000,
0x00FF0000,
0x00003300,
0x00006600,
0x00009900,
0x0000CC00,
65280
51
102
153
0x000000CC,

0x000000FF,
0x00333300,
0x00666600,
0x00999900,
0x00CCCC00,
0x00FFFF00,
0x00003333,
26214

39321
52428

65535
0x00330033,
0x00660066,
0x00990099,
0x00CC00CC,
```

0x00FF00FF
16777011
16777062
16777113
16777164

0xFF33FF,
0xFF66FF,
0xFF99FF,
0xFFCCFF,
3407871
6750207
10092543
13434879
13421772
13421670

13421721
13421823
13382604
13395660
13408716
13434828
0x0033CCCC,
0x0066CCCC,
0x0099CCCC,
0x00FFCCCC,

0x00999933,
0x00999966,
0x009999CC,
0x009999FF,
10040217
10053273
10079385
10092441
3381657
6723993

13408665
16751001
6710835

6710937
6710988
6711039
0x00663366,
0x00669966,
0x0066CC66,
0x0066FF66,

0x00336666
10053222
13395558
16737894
3355494
0x00333399,
0x003333CC,
0x003333FF,
0x00336633,
3381555

3394611
3407667
6697779
10040115
13382451
16724787
13158
0x00336600,
0x00660033,
0x00006633,

0x00330066,
0x00663300,
0x00336699,
0x00669933,
0x00993366,
3381606
6697881
10053171
6724044
10079334

13395609
6737049
10053324
13408614
10079487
13434777
16751052
0x0099FFCC,
0x00CC99FF,

0x00FFCC99,

0x00111111,
0x00222222,
0x00444444,
0x00555555,

```
    0x00AAAAAA
    0x00BBBBBB,
    0x00DDDDDD,
    15658734
};
```