5.2 основные операторы

Возможность создания AS-сценариев — одно из самых замечательных свойств программы Flash. Не будет преувеличением сказать, что именно благодаря этой возможности технология Flash и пользуется такой большой популярностью.

Для того, чтобы научиться писать сценарии, нужно вначале ознакомиться с языком ActionScript, который используется во Flash. Язык этот весьма прост для изучения, а способ создания сценария, как видно из предыдущего раздела, приспособлен для того, чтобы сделать процесс создания сценария максимально удобным: если, предположим, вы не помните, какие операнды следует ввести для того или иного оператора, программа сама подскажет их вам.

Давайте прежде всего ознакомимся с основными операторами ActionScript. «Основные операторы» — операторы, встречающиеся наиболее часто — даже вынесены в отдельную папку в левой части окна Frame/Object Actions. Эта папка называется Basic Actions. Интересно, что в этой папке (в отличие от папки Actions) операторы перечислены описательно, например, Get URL или If Frame Is Loaded вместо getURL и ifFrameLoaded, как следует писать в коде.

Итак, начнём с оператора gotoAndPlay. Этот оператор осуществляет немедленный безусловный переход в указанный кадр. При этом содержимое текущего кадра не прорисовывается. Однако, если после этого оператора в текущем сценарии расположены ещё какие-либо операторы, они *будут* выполнены. То есть оператор gotoAndPlay всегда исполняется последним в кадре, даже если он расположен где-то в середине или даже самым первым. В этом операторе обязательно следует указать кадр, в который осуществляется переход, например:

```
gotoAndPlay (5);
```

осуществляет переход в пятый кадр текущей сцены. Обратите внимание на точку с запятой в конце строки. В ActionScript этот символ используется в качестве разделителя между операторами (как и во многих других языках). Поэтому привыкайте ставить точку с запятой после каждого оператора (правда, в «нормальном» режиме ввода этот символ добавляется в код автоматически).

Если в кадре, в который осуществляется переход, определена метка (метка определяется в поле Label на вспомогательной панели Frame, и отображается прямо на линейке времени, если там достаточно места), то вместо номера кадра в операторе gotoAndPlay можно указать метку кадра. Например, если в вашем ролике есть кадр с меткой fade, то вы можете написать в другом кадре

```
gotoAndPlay ("fade");
```

для осуществления безусловного перехода в этот кадр. Метка кадра пишется здесь в кавычках.

Иногда бывает более целесообразно указывать метку кадра для перехода, чем номер кадра. Дело в том, что если вы в процессе дальнейшей работы удалите или вставите какие-либо кадры, то вся нумерация кадров изменится. Соответственно придётся вручную изменять и все номера кадров в операторах типа gotoAndPlay. А метки кадров будут сдвигаться вместе с самими кадрами, поэтому если в операторах типа gotoAndPlay использованы метки, вам уже не придётся изменять их вручную.

Можно, кстати, использовать в операторе gotoAndPlay вместо номера кадра *выражение*, с помощью которого он будет вычислен, например, x+(2*y). Подробнее о

выражениях рассказано в разделе 5.4.

Во всех предыдущих случаях переход осуществляется на кадры *текущей сцены*. Если в вашем ролике несколько сцен, то в операторе gotoAndPlay можно указать не один, а два операнда. В этом случае первым из них должно быть название сцены, заключённое в кавычки, а вторым — номер кадра, вычисляемое выражение или метка кадра (как описывалось выше). Например, оператор

```
gotoAndPlay ("Scene 1", 28);
```

осуществляет переход в 28-й кадры сцены «Scene 1».

Как видно из названия, оператор gotoAndPlay подразумевает продолжение воспроизведения ролика после перехода. Если вы хотите, чтобы после перехода в другой кадр ролик остановился, вместо оператора gotoAndPlay следует использовать оператор gotoAndStop, например, вот так:

```
gotoAndStop (10);
```

Всё сказанное выше об операторе gotoAndPlay применимо и к gotoAndStop.

Для безусловного перехода существуют ещё четыре оператора. Немедленный переход к следующей сцене осуществляет оператор

```
nextScene ();
```

а переход к предыдущей сцене —

```
prevScene ();
```

Аналогично для перехода к следующему и предыдущему кадру текущей сцены используются соответственно операторы

```
nextFrame ();
```

и

```
prevFrame ();
```

Давайте рассмотрим очень простой пример на использование оператора gotoAndPlay. Допустим, мы хотим сделать так, чтобы градиентный заголовок веб-странички «Выезжал» слева и затем начинал переливаться всеми цветами радуги.

Вначале переименуем единственный пока слой в header. Создадим текстовое поле, напишем в нём сам заголовок, выберем его шрифт и преобразуем в мувик. Вставим в 36-ю позицию ключевой кадр клавишей F6. Вернёмся в первый кадр и сдвинем наш заголовок влево за границу рабочей области, уменьшим его горизонтальный размер и наклоним влево (это для того, чтобы было впечатление «встречного ветра». На самом деле этот эффект можно сделать гораздо более реалистичным, скомбинировав несколько анимаций и используя параметр Easing).. Создадим анимацию движения.

Теперь создадим новый слой, назовём его gradient. Нарисуем в нём прямоугольник без контура, по размерам несколько превосходящий рабочую область. Зальём его разнообразным линейным градиентом (например, из 7 цветов). Теперь преобразуем слой header в слой-маску, а слой gradient — в замаскированный. Перейдём в этот слой, скопируем

градиентный прямоугольник, поставим копию непосредственно рядом с «оригиналом» и развернём командой Modify -> Transform -> Flip Horizontal. Теперь выделим оба прямоугольника вместе и преобразуем их в мувик. Создадим в 36-й позиции ключевой кадр клавишей F6. Если сейчас просмотреть ролик, мы увидим, что градиентный заголовок «приезжает», приближаясь с левой стороны.

Теперь создадим вторую анимацию. Перейдём в слой gradient и вставим ключевые кадры клавишей F6 в 96-ю и 156-ю позиции. Вернёмся в 96-й кадр и сдвинем мувик влево, так, чтобы рабочую область закрывал правый прямоугольник. Создадим анимацию движения в 36-м и 96-м кадрах. Мувик будет перемещаться влево и обратно вправо.

Перейдём в слой header, отметим 156-ю позицию и вставим там обычный кадр клавишей F5. Опять просмотрим ролик. Теперь заголовок выезжает, потом переливается, резко исчезает, опять выезжает и переливается, и т.п. Осталось сделать самое главное. Чтобы заголовок «выезжал» только один раз, а потом постоянно проигрывал анимацию «переливов цвета», отметим вначале 36-й кадр и поставим в нём метку begin. Теперь перейдём в 156-й кадр и вставим в него код:

```
gotoAndPlay ("begin");
```

Вот теперь ролик готов. Из последнего кадра переход осуществляется не в 1-й, а в 36-й, за счёт чего зациклен не весь ролик, а только нужная его часть, а «вступление» проигрывается только один раз.

Теперь перейдём к рассмотрению других операторов.

Самые простые операторы — это оператор начала воспроизведения ролика после остановки

```
play ();
```

и оператор останова воспроизведения

```
stop ();
```

Оба эти оператора не имеют параметров. Их мы уже рассматривали выше.

Если вы хотите почему либо остановить в один момент проигрывание всех звуков, которые в данный момент воспроизводятся у вас в ролике, используйте оператор

```
stopAllSounds ();
```

также не имеющий параметров. Он обычно используется, если вы хотите предусмотреть в ролике кнопку Mute, отключающую звуковое сопровождение. Обратите внимание, что этот оператор останавливает только звучащие в данный момент звуковые объекты, и вовсе не запрещает воспроизводить их в дальнейшем.

Один из самых важных операторов ActionScript — это оператор загрузки страницы по указанному адресу getURL. Обычно в качестве значения URL в этом операторе указывают абсолютный адрес веб-страницы, например, так:

```
getURL ("http://www.mmv.ru/p/rusmodern");
```

В этом случае вышесказанная страница загружается в окно браузера. Если браузер не открыт, он запускается автоматически. На практике этот оператор является Flash-аналогом

обычной WWW-гиперссылки (тега <A> с атрибутом HREF).

А у обычной гиперссылки, скажете вы, есть атрибут TARGET, указывающий, в какое окно загрузить страницу. По аналогии с этим атрибутом оператор getURL имеет второй, необязательный параметр, в котором точно таким же образом можно указать имя окна для загрузки страницы! Это имя может представлять собой либо имя уже открытого окна или фрейма браузера, либо одно из ключевых слов _top, _self, _blank и _parent. Ключевое слово _top загружает страничку в самый верхний в иерархии фрейм, ключевое слово _self — в то же окно, откуда был сделан вызов, ключевое слово _blank — в новое окно браузера и ключевое слово _parent — в окно, являющееся родительским по отношению к данному. Если вы знаете атрибуту HTML-тега <A>, то всё это покажется вам до боли знакомым.

Кроме того, оператор getURL имеет ещё и третий, тоже необязательный, параметр — способ передачи переменных загружаемой страничке. Так как переменные могут быть переданы либо методом GET, либо методом POST, значением этого параметра должно быть одно из этих двух ключевых слов. Для тех, кто не знаком с синтаксисом HTTP-протокола, кратко поясним, что при передаче переменных методом GET их значения как бы «дописываются» в конец строки адреса и передаются вместе с ней, а тело HTTP-запроса остаётся пустым. Это ускоряет передачу данных, однако при попытке передать слишком много данных этим способом можно столкнуться с ограничениями на длину строки запроса, действующими на многих серверах. Поэтому для передачи большого объёма данных безопаснее пользоваться более медленным, но в некотором смысле более надёжным методом POST, при котором все данные помещаются в тело HTTP-запроса.

К чему я это всё говорю? Дело в том, что при отправке переменных с помощью AS-оператора getURL по указанному адресу будут переданы *все* переменные, существующие на данный момент в ролике, и если их много, то при использовании метода GET могут возникать непредвиденные ошибки.

Вот как может, например, выглядеть строка такого оператора:

```
getURL ("http://www.mysite.ru/cgi-bin/sub.cgi", "_blank", "GET");
```

Однако есть способ передать на указанный адрес только те переменные, которые необходимы. Например, если вам нужно передать только значения переменных x и y, зачем передавать все переменные ролика, которых может быть несколько десятков? Передавая только необходимые данные, вы существенно ускорите обмен пользователя с сервером.

Для того, чтобы передать только те данные, которые необходимы, нужно вообще не включать в строку оператора getURL третий параметр (чтобы программа думала, что передачи переменных не происходит), но при этом в строку адреса их следует включить вручную. Например, если вам нужно передать только переменные x и y, значения которых на данный момент равны, например, 37 и 67, нужно модифицировать оператор следующим образом:

```
getURL ("http://www.mysite.ru/cgi-bin/sub.cgi?x=37&y=67",
"_blank");
```

При этом, разумеется, переменные передаются методом GET. Опытные флэшеры обычно называют такой способ передачи данных наиболее предпочтительным для большинства случаев.

Обычно передача параметров имеет смысл, если запрашиваемая страница является CGI-сценарием. Правда, в некоторых случаях таким образом передают данные и клиентским сценариям (например, JavaScript), содержащимся на HTML-страничке. Читателем, заинтересовавшимся вопросами CGI-программирования, можно порекомендовать книгу

А.Павлова «CGI-программирование» («Питер», СПб, 2000).

Ещё одним часто используемым оператором ActionScript является оператор loadMovieNum. Он запрашивает Flash-ролик по указанному сетевому адресу и загружает его *прямо в окно текущего ролика* При этом новый ролик либо полностью заменяет текущий ролик, либо загружается поверх него.

Поясним сказанное. В роликах Flash определена так называемая многоуровневая структура. Уровни ролика нумеруются, начиная с 0. На каждом уровне может находится только один ролик, не являющийся флэш-символом. При этом основной ролик считается находящимся на уровне 0. Если мы загрузим на уровень 0 какой-либо другой ролик, то он заменит собой предыдущий. Например, если в каком-либо кадре написать

```
loadMovieNum ("auto2.swf", 0);
```

то в этот момент ролик auto2.swf будет загружен вместо текущего. Такие команды хорошо применять в кнопках, чтобы использовать их как своеобразные гиперссылки (наряду с оператором getURL).

Если же написать

```
loadMovieNum ("auto2.swf", 1);
```

то в этот момент ролик auto2.swf будет загружен на уровень 1. При этом он заслонит собой основной ролик, но если его впоследствии выгрузить оператором

```
unloadMovieNum (1);
```

под ним обнаружится «старый» ролик. Более того, если во вновь загруженном ролике имеются прозрачные элементы, то сквозь него будет «просвечивать» старый ролик. Как вы уже, наверное, поняли,оператор unloadMovieNum служит для выгрузки ролика с указанного уровня. При этом, правда, выгружаются и ролики, лежащие «выше» выгружаемого. Например, если загрузить ролики на уровни 1 и 2, а затем выполнить

```
unloadMovieNum (1);
```

то выгружены будут оба ролика. Кстати, из этого следует, что выгрузка ролика с уровня 0 не судит ничего интересного — окно воспроизведения ролика полностью очищается.

Между прочим, можно загрузить ролик не только на один из уровней, но и вместо любого экземпляра мувика. Для этого нужно использовать оператор loadMovie. Например, если в вашем ролике на сцене присутствуют несколько экземпляров мувика, один из которых имеет имя changeme, то после выполнения кода

```
loadMovie ("auto2.swf", "changeme");
```

вы увидите вместо этого экземпляра ролик auto2.swf, который будет вести себя уже как флэш-символ!

В следующих главах в нескольких примерах будет проиллюстрировано применение операторов getURL и loadMovie/loadMovieNum.

Весьма интересным оператором ActionScript является также оператор fscommand Он позволяет передать команду во внешнюю среду. Несколько примеров использования этого оператора мы рассмотрим в разделе 7.11.

Операторы tellTarget и его полный аналог with отличаются от прочих тем, что он не производит каких-либо действий с элементами ролика, а только переопределяет *объект назначения.* Это означает следующее. Оператор tellTarget (или with) обязательно содержит после себя фрагмент кода, заключённый в фигурные скобки {}. Такой фрагмент кода принято называть *блоком*. Весь блок, содержащийся в операторе tellTarget, будет осуществлять действия не с тем объектом, в котором расположен сценарий, а с объектом, определённым в операторе tellTarget/with. (В некотором смысле этот оператор аналогичен JavaScript-оператору with).

Для того, чтобы читателю легче было понять «умозрительное» объяснение, содержащееся в предыдущем абзаце, приведём пример. Допустим, что вы открыли новый ролик и создали в нём мувик с анимацией. Для примера достаточно сделать что-нибудь простое, например, квадрат, превращающийся в круг и обратно.

Теперь поместите в основном ролике экземпляр этого мувика. Выделив его, введите в поле Name вспомогательной панели Instance имя для этого экземпляра, например, «Quad». (Очень важно, каким именем вы называете именно *экземпляр* мувика, а нге его эталон. К этому вопросу мы ещё вернёмся в разделе 6.1)

Теперь создайте в основном ролике какую-либо анимацию движения с участием мувика Quad. Например, это может быть движение по кругу.

После всей этой подготовки перейдём к основной части нашего примера. Создадим в основном ролике второй слой, назовём его Butt. Нарисуем в нём кнопку любого вида и в ставим туда код:

```
on (release) {
    stop ();
}
```

Пока вроде бы ничего нового. При просмотре ролика нажатие на кнопку остановит его проигрывание. Однако обратите внимание, что при нажатии на кнопку останавливается только основной ролик: движение по кругу прекратилось, но трансформация квадрата в круг и обратно продолжается, то есть мувик Quad по-прежнему воспроизводится. Это происходит потому, что кнопка расположена в основном ролике, следовательно, оператор stop останавливает именно его.

Однако с помощью оператора tellTarget можно перенаправить действие оператора stop на вложенный мувик. Заменим код кнопки на следующий:

```
on (release) {
    tellTarget ("Quad") {
        stop ();
    }
}
```

Здесь оператор tellTarget перенаправляет действия всех операторов, находящихся внутри его блока, на мувик Quad. Если сейчас просмотреть ролик, то при нажатии на кнопку вы увидите, что остановилась трансформация квадрата в круг и обратно, а круговое движение объекта продолжается. Следовательно, остановилось только проигрывание мувика Quad, а воспроизведение основного ролика продолжается.

Если же вы хотите нажатием кнопки остановить и основной ролик, и вложенный мувик, ничто не мешает объединить обе команды остановки в одном сценарии:

```
on (release) {
```

```
    stop ();
    with (Quad) {
        stop ();
    }
}
```

Как видите, синтаксис оператора with отличается только отсутствием кавычек у имени целевого объекта.

Впрочем, для данного случая наиболее простым решением было бы вообще сделать флэш-символ, названный нами Quad, не мувиком, а графикой. В этом случае при остановке основного ролика одновременно остановилась бы и анимация графического объекта.

И, наконец, последний из основных операторов AS, которые мы рассматриваем в этом разделе, позволяет проконтролировать, загружен ли уже в память компьютера тот или иной кадр. Поскольку Flash-ролики чаще всего предназначаются для загрузки через Интернет, то естественно, что для загрузки ролика полностью должно пройти какое-то время. Однако по умолчанию воспроизведение начинается сразу же, как только будет загружен первый кадр. В некоторых случаях это может дать нежелательные задержки внутри ролика, когда компьютеру придётся ожидать загрузки следующего кадра.

Чтобы этого избежать, можно использовать оператор ifFrameLoaded. Он проверяет, загружен ли указанный кадр (можно указать метку или номер кадра), и, если это так, выполняет свой блок операторов. В этом блоке можно дать указание перейти на начало воспроизведения ролика.

Приведём пример. Откроем новый ролик, создадим какую-нибудь анимацию, но начало её передвинем на третий кадр. Дадим этому кадру метку «begin». Последнему кадру дадим метку «end». Поскольку анимация начинается не в первом кадре, впишем в последний кадр код

```
gotoAndPlay ("begin");
```

для перехода из последнего кадра к началу анимации, а не к пустому первому кадру. Теперь в первом кадре напишем :

```
ifFrameLoaded ("end") {
    gotoAndPlay ("begin");
}
```

а во втором кадре:

```
gotoAndPlay (1);
```

Теперь воспроизведении анимации не начнётся до тех пор, пока не будет загружен её последний кадр. Действительно, в первом кадре Flash проверит, загружен ли кадр с меткой end. Если он ещё не загружен, то блок при операторе ifFrameLoaded не выполнится и программа перейдёт ко второму кадру. А во втором кадре осуществляется переход обратно на первый кадр. Таким образом, будут по очереди выполняться сценарии первого и второго кадров, пока не будет загружен кадр с меткой end. Тогда в первом кадре выполнится блок при операторе ifFrameLoaded, в котором выполняется безусловный переход на начало анимации (кадр с меткой begin).

Для того, чтобы просмотреть результат, после создания ролика нажмите Ctrl-Enter и затем ещё раз Ctrl-Enter. Второе нажатие Ctrl-Enter включает режим Show Streaming, то есть эмуляцию загрузки данных через Интернет. При этом в меню Debug можно выбрать

желаемую скорость «загрузки». Здесь можно установить шесть различных скоростей, чтобы по очереди пробовать каждую из них. По умолчанию в программе уже установлены скорости для эмуляции загрузки через модем 14.4, 28.8 и 56K. Выбрав пункт Customize, можно добавить свои варианты тестовых скоростей или изменить существующие.

Если сейчас добавить в первый кадр надпись «Подождите, идёь загрузка...», то пользователь, загружающий ваш ролик через интернет будет созерцать эту надпись, пока не загрузится последний кадр и не начнётся анимация. Собственно говоря, сейчас мы с вами уже создали простейший *предзагрузчик*. Предзагрузчиком называют простую анимацию, которая быстро загружается и проигрывается, «развлекая» пользователя во время ожидания загрузки основного ролика. Конечно, приведённый пример весьма примитивен. К вопросу создания более развитых предзагрузчиков мы ещё вернёмся в разделе 7.9.

Мелкий шрифт. Вопрос для самоконтроля. В приведённом примере во время ожидания загрузки последнего кадра проигрываются первые два кадра ролика. При этом надпись «Подождите...», находящаяся в первом кадре, всё время видна на экране. Почему она не «мигает», ведь во втором кадре нет никакого изображения?

Весьма интересен оператор #include. Он загружает указанный файл ActionScript, обычно для того, чтобы текущий ролик мог использовать определённые во внешнем файле функции, объекты и т.д. Этот оператор используется при экспорте файла (создании файла SWF), а также тестировании или публикации. О файлах SWF и публикации мы подробнее поговорим в главе 7. Оператор #include заимствован из языка C.

Итак, мы познакомились с основными операторами ActionScript. Конечно, их набор не исчерпывается перечисленным в этом разделе. В одном из следующих разделов мы кратко рассмотрим другие операторы ActionScript.