ИНТЕРНЕТ

Штенников Дмитрий Геннадьевич,
Николаев Дмитрий Геннадьевич

ОСНОВЫ ИСПОЛЬЗОВАНИЯ MACROMEDIA FLASH COMMUNICATION SERVER (FLASH MEDIA SERVER) ДЛЯ СОЗДАНИЯ ИНТЕРНЕТ-ПРИЛОЖЕНИЙ (часть 1)

В статьях по использованию Macromedia Flash были рассмотрены аспекты создания Интернет-приложений на основе клиентского программирования. Разумеется, подобный подход мог быть интересен в случае создания статичных проектов или проектов, использующих дополнительные программные средства для создания динамического контента. Тем не менее, собственные средства реализации динамичности для Интернет-проекта существуют и в языке ActionScript. Условием для его применения является использование Flash Communication Server'a, о котором пойдет речь ниже.

ВВЕДЕНИЕ

Macromedia Flash получило свое развитие как метод легкого создания и распространения простейшей анимации и графики в сети для платформ мощных приложений. По информации от Macromedia Flash Player 6 (плагин для Flash MX) уже в июне 2004 был доступен более чем 94 % автоматизированным рабочим местам в развитых странах с выходом в Интернет. С согласия пользователя Flash movie может записывать в реальном времени аудио и видео от микрофона или от камеры и передавать это на Flash Communication Server MX (сервер связи).

Сервер может перераспределять потоки данных другим пользователям, имеющих Flash Player на своем компьютере. Связь в реальном времени делает возможным разрабатывать широкий диапазон приложений.

Функции Flash и Flash Communication Server MX могут быть использованы для создания:

– различных видеоконференций, корпоративных конференций и чатов с общими составляющими, такими как общий чат, whiteboards, графики;

– приложений с Video on demand и данных с пользовательскими интерфейсами, которые могут включать закрытые заголовки и средства управления оболочкой;

– прямой трансляции с помощью настраиваемого пользовательского взаимодействия, такого как небольшой чат и диалог в форме вопросов и ответов;

– многопользовательских игр, моделирования и других приложений с добавлением аудио и видео, если необходимо.

КЛИЕНТЫ И СЕРВЕРЫ

FlashCom – это серверное приложение, которое установлено на хосте машины, подобно серверу сети; однако FlashCom работает иначе. Вместо принятия множества лаконичных соединений от браузеров, требующих web-страницу или другой ресурс,

FlashCom принимает продолжительные подключения от Flash Movie, выполняемые во Flash Player. Каждый Flash Movie может совместно использовать данные с другим Flash Movie через сервер, применяя Протокол Обмена сообщениями в реальном времени (RTMP). В отличие от модели HTTP (запрос/ответ), используемой браузерами, чтобы связаться с web-серверами, RTMP ведет подключение к FlashCom Серверу беспрерывно. Таким образом, нет необходимости в специальных шагах для поддержки сеанса передачи информации. Как только сервер принимает подключение клиента, соединение может использоваться для обмена аудио, видео и ActionScript данными, пока клиент или сервер не разъединятся.

Flash Player может выполняться в пределах Standalone Player или в пределах web-браузера. Flash Player (и любой Movie проигрывающий в его пределах) считается клиентом. FlashCom не может инициировать подключение к Movie, подключение должно быть начато от Flash Player'а, выполняющегося на стороне клиента. Архитектура клиент/сервер для приложений FlashCom показана на рисунке 1.

Через Web-браузер или через Flash Player загружаются файлы Flash Movie (файлы с расширением .swf), затем передаются .swf файлы в Player для его выполнения. Flash Movie обеспечивает пользовательский интерфейс и может попытаться соединиться через Player с любым FlashCom Сервером. После соединения Flash Movie может поддерживать связь с сервером. Кроме того, он может связываться через сервер с Movie, играющими на других Flash клиентах. Flash Movie может передавать аудио и видео на FlashCom Сервер так, чтобы другие клиенты Flash с доступом к тому же самому серверу могли проиграть записи, сохраненные на сервере, и живые потоки от других клиентов.

Живой поток – это тот поток, который выложен на сервер одним клиентом так, чтобы другие клиенты могли иметь доступ к нему. Таким образом, данные клиента прибывают на сервер, сервер дублирует их и пересылает их каждому клиенту, где эти данные можно просмотреть и прослушать. Записанные потоки сохраняются на сервере и могут проигрываться, начиная с любой точки в пределах потока, могут быть приостановлены и перезапущены. Также возможно остановить записанный поток, найти любую точку в его пределах и начать проигрывать снова.

Если множество FlashCom Серверов связаны друг с другом, клиенты, соединенные с одним сервером, могут связаться с клиентами с других серверов. Способность взаимодействия между серверами и клиентами, связанными с ними, делает возможным крупномасштабные приложения типа прямых трансляций для многочисленной аудитории.

FlashCom может принимать много различных приложений. Несколько запросов приложения могут быть обработаны в одно и то же время. Каждому запросу дают его собственное уникальное имя. И когда клиент соединяется с сервером, он всегда указывает имя запроса приложения. Например, множество отдельных запросов приложения, названных chatRoom, могут быть доступными. Каждый запрос имеет свое собственное уникальное имя и может обеспечивать уникальные ресурсы для клиента.

Рисунок 2 иллюстрирует трех клиентов, связанных с одним и тем же самым запросом chatRoom приложения.

СОЗДАНИЕ ПРИЛОЖЕНИЯ

Благодаря природе приложений связи типа клиент/сервер (распределенная модель, в которой имеются два типа приложений: приложения-клиенты, посылающие запросы или вызывающие соответствующие события), разработчик обычно создает на сто-

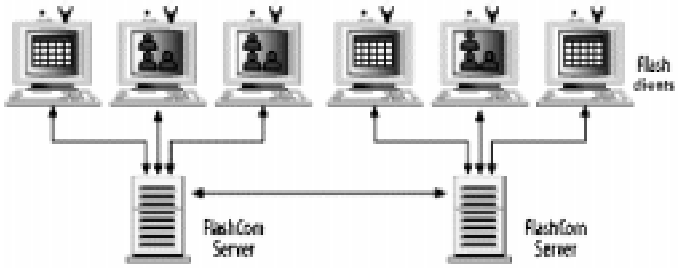
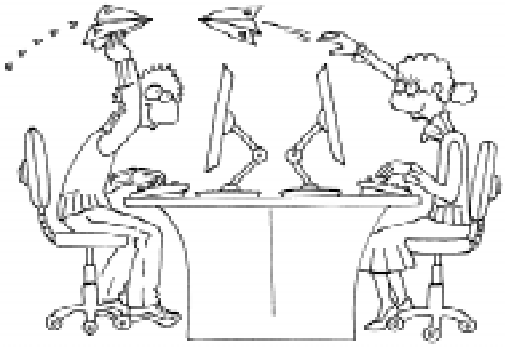


Рисунок 1. Архитектура клиент/сервер.



...используя Протокол Передачи сообщений в реальном времени...

роне клиента Flash Movie, чтобы управлять пользовательским взаимодействием и отдельной серверной частью приложения FlashCom, с которой происходит соединение. На стороне клиента Flash Movie может быть написан на ActionScript 1.0, 2.0 или 3.0.

Серверное приложение FlashCom написанно на (SSAS), который подобен клиентскому ActionScript. Для создания приложения FlashCom сначала необходимо создать домашнюю директорию на сервере.

ActionScript файлы с исходным кодом должны быть помещены в домашнюю директорию приложения, которая дает каждому приложению его уникальный серверный режим.

Разработка Movie Flash почти всегда включает программирование с ActionScript, язык создания сценария Flash основан на ECMA стандарте (подобно JavaScript). В дополнение к типичным объектам, ActionScript

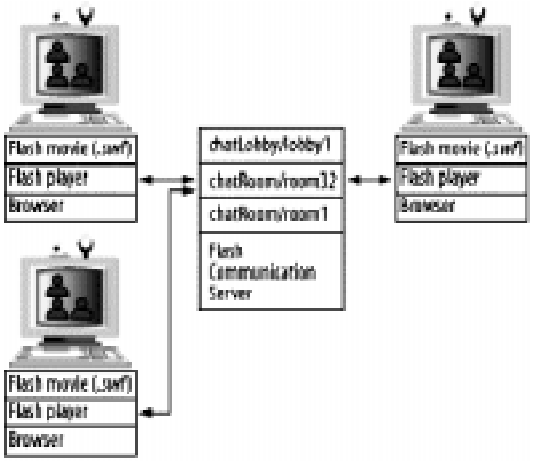


Рисунок 2. Клиенты и chat-room.

поддерживает специальный MovieClip тип данных. Movie Clips – основные блоки строительства для Flash анимации, и это – основа для компонентов более высокого уровня типа Button, DataGrid, и Tree.

Для многих приложений с видео Вы можете использовать заготовку FLVPlayer, который не требует никакой авторизации Flash, доступного на сайте: http://www.peldi.com/blog/FLVPlayer.html

FLVPlayer может быть настроен для соединения с любым приложением и для включения автоматической пропускной способности и выбора потока. Видео Player также доступен как часть Macromedia Video Kit: http://www.macromedia.com/software/studio/flashvideokit

Вы можете создавать простые приложения соединения, такие как видеоконференц-приложения, используя заготовку компонентов типа SimpleConnect, PeopleList и VideoConference компонентов в Macromedia Flash. Вы можете перетащить компоненты из панели Flash's Components на Сцену, чтобы создать интерфейс пользователя. Используя панель Flash's Properties (состояние), Вы можете группировать компоненты, чтобы использовать их вместе и присваивать им адреса запроса приложения на FlashCom Сервере для подключения.

Для создания более разнообразных приложений необходимо использовать ActionScript, чтобы создавать или настраивать компоненты и разрабатывать уникальные интерфейсы пользователя. Связанные между собой ActionScript классы делают использование компонентов и приложений более легким. На стороне клиента они включают NetConnection, NetStream, Camera, SharedObject и Microphone классы. ActionScript классы на стороне сервера включают Application, Client, Stream и SharedObject классы.

ПРОТОКОЛ ОБМЕНА СООБЩЕНИЯМИ В РЕАЛЬНОМ ВРЕМЕНИ

Flash Player соединяется с FlashCom, используя Протокол Передачи сообщений в реальном времени (RTMP). RTMP использует TCP (протокол управления передачей)

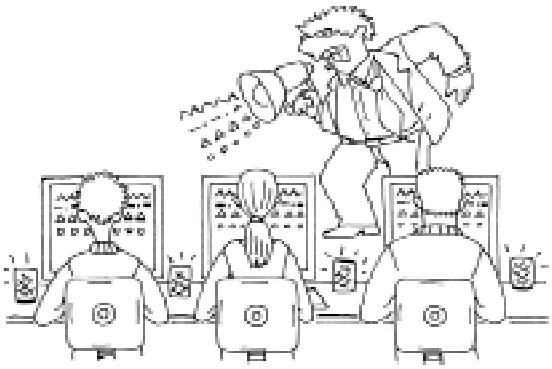
для передачи пакетов между Flash Player и сервером. TCP – надежный протокол, который гарантирует доставку каждого пакета данных. RTMP может транспортировать аудио, закодированное в MP3 и Nellymoser форматах, видео, закодированное в формате Flash Video (FLV), и данные, закодированные с использованием формата AMF. AMF обеспечивает эффективное преобразование в последовательную и параллельную формы данных так, чтобы и простые, и сложные ActionScript данные могли быть переданы между клиентом и сервером без необходимости вручную их кодировать или расшифровать.

FLASHCOM ПРОТИВ ТРАДИЦИОННЫХ MEDIA СЕРВЕРОВ

Использование TCP как основного протокола для Flash связи упрощает управление транспортировкой аудио, видео и ActionScript данных, передающихся между клиентом и сервером. Однако TCP – протокол PPP (двухпунктовый), который подразумевает, что каждый клиент требует отдельного TCP клиента/сервера подключения; следовательно, он не может передавать пакеты от сервера ко многим клиентам на сетевом уровне. Если живой аудио поток был послан одновременно множеству клиентов, сервер должен послать дубликаты аудиоданных каждому клиенту через дискретные подключения. Традиционные серверы средств информации, предназначенные, прежде всего, для того чтобы распределять потоки средств информации от сервера к клиенту, обычно обеспечивают возможность послать поток, используя UDP (протокол пользовательских датаграмм) так же, как TCP. UDP может использоваться для многонаправленных или однонаправленных потоков. Когда допускается, что единственная копия потока передана, то любой клиент может ее получить. Мультивещание имеет два огромных преимущества: уменьшает загрузку на сервере и пропускную способность, требуемую, чтобы послать потоки большому числу клиентов.

К сожалению, огромное большинство системных служб Internet (ISPs) блокирует multicast в их сетях из-за беспокойства за безопасность. Следовательно, серверы средств информации обеспечивают аварийный переход к однонаправленной передаче данных по UDP. Однонаправленная передача означает, что сервер должен дублировать данные потока и посылать их в отдельном потоке каждому клиенту. Если по некоторым причинам клиент не принимает однонаправленный поток, UDP серверы средств информации, естественно, снижают скорость передачи дублированного потока данных каждому клиенту по TCP. Почти такой же способ использует FlashCom.

Даже без широковещания UDP имеет некоторые преимущества перед традиционными серверами средств информации. UDP – не «надежный» протокол. Он не гарантирует, что каждый пакет, переданный от сервера, до клиента дойдет. Для аудио и видео потоков потеря некоторых пакетов не может оказать значимого влияния на качество воспроизведения. Чем более перегружена сеть, тем большее количество пакетов будет потеряно, так что качество потока может ухудшиться без большого замедления доставки потока. Напротив, TCP регулирует сетевую перегрузку и пропускную способность, замедляя передачу данных и снова посылая потерянные пакеты. Чтобы поддерживать передачу медиа данных по TCP, количество, посылаемых данных должно быть динамически отрегулировано в ответ на сетевую пропускную способность и перегрузку. RTMP разработан, чтобы регулировать количество передаваемой видео и аудио информации, отбрасывая аудио сообщения и видео фреймы



Если живой аудио поток был послан одновременно множеству клиентам, сервер должен послать дубликаты аудио данных каждому клиенту...

в ответ на недостаточную сетевую пропускную способность. На перезагруженных сетях это не столь же эффективно, как на сетях основанных на UDP-протоколах для передачи медиа-потоков, но на современных сетях это соответствует очень хорошему выполнению.

RTMP поддерживает больше, чем протоколы передачи медиа от традиционных серверов средств информации. Он также поддерживает динамическую передачу многочисленных потоков, которые могут содержать аудио, видео и ActionScript данные, и от сервера до клиента и от клиента на сервер. RTMP управляет передачей аудио, видео и ActionScript данных отдельно. ActionScript данные никогда не будут потеряны, потому что любая потеря может иметь катастрофическое влияние на приложение. Аудио и видео данные буферизированы отдельно на сервере.

Если аудио данные в аудио буфере подходят к некоторому порогу, все данные в буфере сбрасываются, и вновь прибывшие данные могут начать накапливаться в буфере, которые нужно переслать каждому клиенту. Видео данные управляются подобным способом, за исключением того, что данные в буфере сбрасываются, когда прибывает новый ключевой кадр. Сбрасывание видео данных, когда прибывает ключевой кадр, гарантирует, что клиент никогда не получит частичные модификации фрейма для искаженного ключевого кадра. Иначе видео изображение могло бы состоять из мозаики блоков размерами 8×8 пикселей от 2-х разных фреймов. RTMP также располагает приоритетами по данным. Аудио дается самый высокий приоритет, потому что это необходимо для общения в реальном масштабе времени, видео дается самый низкий приоритет, а ActionScript дается промежуточный приоритет между аудио и видео.

КЛАССЫ СОЕДИНЕНИЯ

В то время как RTMP удобно управляет передачей многочисленных потоков данных через сети, реальная сила Flash и FlashCom для разработчиков реализуется в классах высокого уровня, которые делают соединение с сервером, распределение информации и передачу аудио и видео простым и гибким.

СОЕДИНЕНИЕ С СЕРВЕРОМ

NetConnection класс соединяет клиента с запросом приложения на сервере. В самом простом случае Вы можете создавать подключение, вызывая функцию `NetConnection.connect()` с URI отдаленного приложения:

```
nc = new NetConnection();
nc.connect("rtmp://echo.ryerson.ca/
                campusCameras/connector");
```

Для того чтобы определить, было ли успешно установлено подключение, нужно запросить `onStatus()` метод на объекте NetConnection перед вызовом функции `connect()`:

```
nc = new NetConnection();
nc.onStatus = function (info) {
trace (" код подключения: " + info.code);
};
nc.connect("rtmp://echo.ryerson.ca/
                campusCameras/connector");
```

В этом примере, адрес RTMP включает хост (`echo.ryerson.ca`), имя приложения (`campusCameras`) и название запроса (`connector`).

ПЕРЕДАЧА АУДИО, ВИДЕО И ACTIONSCRIPT ДАННЫХ

Как только Вы устанавливаете подключение, используя объект NetConnection в

клиенте, Вы можете использовать его, чтобы посылать или получать поток, содержащий аудио и/или видео. Предположим, что Вы знаете, что зарегистрированный файл видео Flash по имени Ryerson_High_Speed.flv доступен в общем каталоге запросов, Вы можете прикрепить видео поток к видео объекту (здесь назван videoArea (видео область)), используя `Video.attachVideo()`, и проигрывать его, используя `NetStream.play()`. Объект NetStream, in_ns создан через объект NetConnection, nc из предшествующего примера:

```
in_ns = new NetStream (nc);
videoArea.attachVideo (in_ns);
in_ns.play ("public/Ryerson_High_Spee");
```

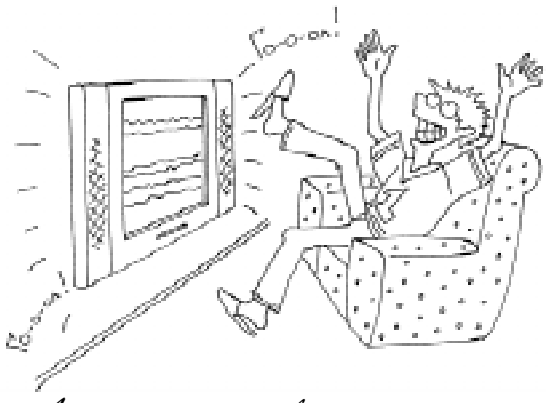
В этом случае видео появляется в видео объекте, который был помещен на сцене во время авторизации и которому было дано имя запроса videoArea; аудио в пределах видео потока прослушивается автоматически. Обратите внимание, что .flv расширение не включено в поток URI, проходящего в метод `play()`.

Помещение потока в приложении Flash Movie также просто. Здесь вы создаете новый объект NetStream, out_ns и прикрепляете аудио и видео потоки от пользовательского микрофона и камеры перед передачей потока, используя `NetStream.publish()`:

```
out_ns = new NetStream (nc);
out_ns.attachAudio (Microphone.get());
out_ns.attachVideo (Camera.get());
out_ns.publish (userName);
```

Класс Camera может получить видео поток от вэб-камеры или другого видео источника, класс Microphone может получить аудио поток от микрофона или другого источника.

Вы можете создавать многочисленные потоки (NetStream объекты) для единственного NetConnection, как показано на рисунке 3. Каждый поток может создавать или запускать поток к серверу или от сервера, но данные в пределах потока передаются только в одном направлении. Если каждый пользователь послал единственный поток, самое простое назвать поток уникальным именем его пользователя или уникальным ID номером.



Аудио дается самый высокий приоритет...

На сервере класс Stream может использоваться для создания плейлистов потоков и даже для проигрывания и публикации потоков через многочисленные серверы. Плейлист (список файлов для воспроизведения) определяет последовательность потоков, которые будут запущены один за другим. Сервер буферизует их, так что они проигрываются без задержки; когда один поток заканчивается, другой начинается.

КАМЕРА, МИКРОФОН И ВИДЕО

Классы Camera и Microphone обеспечивают доступ к видео и аудио источникам (камеры и микрофоны) на системе клиента. Ко многим камерам и микрофонам можно получить доступ с помощью того же Flash Movie. Только один видео и аудио поток может быть запущен в пределах отдельного NetStream; однако, может быть много NetStream'ов в пределах одного NetConnection, как показано на рисунке 4.

Flash Player ответствен за кодирование всех аудио и видео данных. Видео кодирование от каждого видео источника в настоящее время ограничено кодированием скорости передачи данных в битах в единственном разрешении. Camera и Microphone классы могут использоваться, чтобы управлять количеством аудио и видео данных, посы-

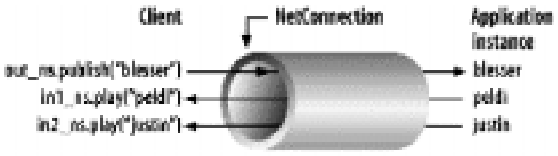


Рисунок 3. Использование потоков.

лаемых серверу, и могут быть динамически откорректированы, чтобы соответствовать ограничениям пропускной способности клиентов, связанных с приложением.

Класс Camera обеспечивает удобный способ изменять разрешающую способность, разряд фрейма и качественные параметры настройки для каждого видео источника. Класс Microphone позволяет устанавливать частоту отсчетов, частоту амплитудно-импульсной модуляции, усиление, уровень затухания и другие. Video объект используется, чтобы отобразить видео в пределах Flash movie, и может быть динамически изменен и перемещен.

СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ ДАННЫХ В РЕАЛЬНОМ ВРЕМЕНИ

Приложения в реальном масштабе времени часто требуют, чтобы данные были разделены или переданы между множеством приложений (movie). SharedObject класс знаком многим программистам Flash как способ создать своего рода supercookie (в системах с удаленным доступом – пароль, порождаемый сервером при первом подключении и отсылаемый пользователю; при последующих подключениях пользователь должен предоставлять серверу этот пароль).

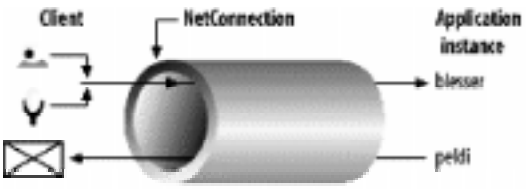


Рисунок 4. Множество потоков NetStream

Локальный общедоступный объект (LSO) может загружать ActionScript данные на клиенте между сеансами. Приложения Flash communication могут использовать удаленные общедоступные объекты (RSO), чтобы совместно использовать информацию в реальном времени между movie, выполняющимися на различных клиентах. Если movie изменяет свойство отдаленного общедоступного объекта, то же самое свойство изменяется в каждом другом movie, связанном с ним. Когда происходят общедоступные изменения объекта, каждый клиент уведомлен относительно изменений. Общедоступные объекты могут быть использованы для:

– регистрации всех movie, связанных с приложением видеоконференции по имени каждого доступного живого потока;

– обновления позиции элементов в игре;

– владения позицией, формой и информацией о цвете каждого графического элемента в общедоступном whiteboard приложении;

– владения данными для каждой формы элемента в общедоступной форме.

В объектно-ориентированном программировании (ООР) данные в пределах всех объектов в приложении определяют текущее состояние приложения. Общедоступные объекты обеспечивают удобный механизм, чтобы владеть состоянием приложения связи, распределенного через многочисленных клиентов и серверы.

Удаленные общедоступные объекты могут быть временные или постоянные. Временные, совместно используемые объекты, сохраняются только в том случае, если они используются.

Постоянные общедоступные объекты сохраняются на сервере, когда они не используются, позволяя клиентам возобновить работу с того места, где они закончили, как только они повторно соединяются с сервером. Общедоступные объекты типа Proxied – механизм для создания общедоступных объектов, доступ к которым идет через запросы множества приложений. Один запрос приложения всегда имеет общедоступный объект, но другие могут создавать сетевое подключение к главному запросу и создавать proxies (по существу, псевдоним) общедос-

тупного объекта. Таким образом, клиенты, соединенные с любым запросом, могут соединяться с тем же самым общедоступным объектом. Эта функция часто важна для создания приложений крупного масштаба.

Код для работы с общедоступными объектами на клиенте немного отличается от требуемого на сервере. Работа с общедоступными объектами также немного более сложна, чем работа с потоками или руководящими сетевыми подключениями. Вы должны сделать четыре основных действия, когда работаете с общедоступными объектами:

– вызовите отдаленный общедоступный объект, используя SharedObject.getRemote();

– настройте общедоступный объект так, чтобы ваша программа могла ответить на изменения, примененные к общедоступному объекту каждым movie или сервером;

– подключитесь к общедоступному объекту;

– после того, как подключитесь, обновите свойства общедоступного объекта по мере необходимости.

На клиенте SharedObject.getRemote() метод возвращает удаленный общедоступный объект. Однако общедоступный объект обычно должен быть установлен с помощью метода `onSync()` и затем подключен, используя объект NetConnection, прежде чем сам он может использоваться. Здесь мы получаем общедоступный объект, названный пользователями, с которыми каждый movie соединится.

Метод `onSync()` задается так, чтобы, в зависимости от цели демонстрации, отображать информацию об изменениях, примененных к общедоступному объекту, как только они происходят. Затем общедоступный объект связывается с сервером:

```
users_so = SharedObject.getRemote("users",
                                  nc.uri);
users_so.onSync = function (infoList) {
for (var i in infoList) {
var info = infoList[i];
switch (info.code) {
case "change":
var id = info.name;
trace("пользователь соединяется с id: " + id);
trace("и именем: " + users_so.data[id]);
break;
case "delete":
var id = info.name;
trace("пользователь разъединяется с id: " + id);
break;
}
}
};
users_so.connect(nc);
```

Метод `onSync()` – часто наиболее интересная часть общедоступного объекта. Когда локальная копия (копия файла, хранимая на локальном сетевом компьютере) общедоступного объекта синхронизирована с сервером и всякий раз, когда любые свойства в общедоступном объекте изменяются, метод `onSync()` вызывается автоматически. Метод `onSync()` «пропускает» через себя массив информационных объектов. Каждый объект содержит информацию об общедоступном объекте или о том, что случилось со слотом (свойство) в общедоступном объекте. Каждый информационный объект имеет свойство кода, которое описывает происходящее «clear», если все свойства удалены, «change», если свойство добавлено или модифицировано, или «delete», если свойство удалено. Имя слота (свойства), который был модифицирован или удален, всегда обнаруживается в названии свойства информационного объекта. Чтобы добавлять или обновлять (модифицировать) данные в общедоступном слоте объекта в пределах Flash, используйте свойство данных общедоступного объекта:

```
users_so.data ["guest_4"] = "Брайен";
```

Аналогично Вы можете извлекать значение в слоте, используя свойство данных:

```
trace (" пользовательское название(имя): "
            + users_so.data ["guest_4"]);
```

КЛИЕНТ И ПРИКЛАДНЫЕ ОБЪЕКТЫ

Всякий раз, когда клиент соединяется с прикладным запросом на сервере, объект Client создается на стороне сервера, чтобы представить удаленного клиента. Объект Client может использоваться серверными скриптами, чтобы отправлять и получать сообщения, посылаемые к каждому и от каждого индивидуального удаленного клиента Flash.

Каждый запрос приложения также имеет единственный объект приложения, который обеспечивает удобный способ управления циклом жизни запроса. Приложение – одноэлементный запрос Application class на стороне сервера.

Client и application объекты вместе со способностью сделать запрос к серверу приложения сети предусматривают основные возможности, необходимые для подтверждения подлинности клиента, как только те подключаются.

УДАЛЕННЫЕ МЕТОДЫ

В типичном приложении индивидуальный клиент должен сделать запрос, чтобы сервер выполнил действие от своего лица. Например, необходимо запросить у сервера длину зарегистрированного потока перед его проигрыванием. Сервер также должен послать запрос клиенту, для того чтобы предпринять что-либо. Например, серверу нужно, чтобы клиент использовал уникальную строку, предоставленную сервером, чтобы дать имя потокам, которые клиент создает. Запросы удаленного метода – путь, по которому клиент и сервер могут вызывать методы друг у друга.

Пример приложения может вызывать метод клиента, использующего `Client.call()` метод. Клиент может вызывать метод, например, приложения, используя `NetConnection.call()` метод, скрипт сервера может вызывать метод индивидуального клиента, чтобы позволять клиенту узнать его уникальный ID номер:

```
client.call ("setID", null, id);
```

Если метод клиента, названный `setID()` вызван, используя `Client.call()` от сервера, он должен быть определен на объекте клиента NetConnection, в противном случае ничего не произойдет. Например, скрипт клиента мог бы выглядеть следующим образом:

```
nc = новый NetConnection ();
nc.setID = function (id) {
myID = id;
};
```

Наоборот, клиент может вызывать метод сервера, используя объект NetConnection:

```
nc.call ("getStreamLength",
         streamInfoResponder, streamName);
```

Для результата, который будет возвращен отдаленным методом, второй параметр в функции `NetConnection.call()` должен быть объектом с `onResult()` методом.

FlashCom также поддерживает механизмы, которые посылают отдаленный запрос метода многочисленным клиентам в одно и то же время. Клиенты, соединенные с одним и тем же общедоступным объектом или пропускающие один и тот же поток, могут получать отдаленный запрос метода одновременно. Например, популярный способ обновлять текстовую область чата состоит в том, чтобы использовать `send()` метод общедоступного объекта для отправки текстового сообщения каждому клиенту:

```
chat_so.send ("showMessage", " Добро
                    пожаловать в чат. ");
```

В этом случае `showMessage()` метод должен быть определен на общедоступном объекте или ничего не произойдет:

```
chat_so.showMessage = функциональный
                              (сообщение) {
chatTextArea.text + = сообщение + "\n ";
chatTextArea.vPosition =
                chatTextArea.maxVPosition;
};
```


Штенников Дмитрий Геннадьевич,
кандидат технических наук,
доцент кафедры Компьютерные образовательные технологии
СПбГУ ИТМО,

Николаев Дмитрий Геннадьевич,
ассистент кафедры Компьютерные образовательные технологии
СПбГУ ИТМО.