Internet и desktop приложения

Если присмотреться внимательно, сближение платформ можно увидеть невооружённым глазом. Как известно, и та и другая платформа имеют свои преимущества и недостатки, а именно:

В течение последних лет процедура установки и обновления настольных приложений всё время упрощается, а сами приложения становятся всё более независимыми от операционной системы.

Рост пропускной способности каналов постепенно уравнивает скорость работы и передачи данных в Веб с настольными приложениями. Бедность базового HTML-интерфейса скрашивается как развитием самого стандарта, так и дополнительных технологий (CSS, XSLT, Flash).

И только с необходимостью перезагрузки целой страницы для получения новых данных и неизбежным ожиданием результата каждого действия сделать до сих пор ничего было нельзя. Ajax в корне меняет всё дело, поскольку позволяет запрашивать данные с сервера в фоновом режиме и актуализировать лишь часть страницы по их приходу.

Кроме того, поскольку объём пакета данных получается небольшим, скорость работы приложения значительно возрастает. Появляется возможность моделировать работу более сложных интерфейсных элементов наподобие desktop интерфейса и задействовать хорошо знакомые шаблоны взаимодействия:

- фоновое сохранение данных без перезагрузки страницы
- валидация форм на сервере
- динамическая подгрузка данных для форм
- становятся ненужными страницы-подтверждения

В течении этого года следует ожидать повальный переход к использованию Ajax наиболее информационно-насыщенными сайтами, такими как:

- Веб-магазины
- Различные онлайн-службы (проведение опросов, словари и т.д.)
- Почта
- Блоги
- Форумы

AJAX в действии

В отличие от множества "перспективных" подходов AJAX очень нагляден. Сайт, сделанный с помощью AJAX, субъективно работает гораздо быстрее обычного сайта. По крайней мере, он быстрее откликается на любые действия пользователя. Классические и, пожалуй, лучшие примеры использования AJAX - проекты Google Maps и Gmail - и это при том, что программисты Google во время работы над ними ни о каком AJAX и знать не знали.

Запросы пользователей обрабатываются очень быстро, потому что использование идеологии AJAX позволяет не перезагружать страничку целиком, а обновлять на ней только те элементы, которые требуют обновления. У того же Gmail с недавнего времени есть обычный HTML-интерфейс для совместимости со старыми браузерами, и любой желающий может убедиться в том, что работает он в несколько раз медленнее, чем классический интерфейс Gmail.

Как это работает

Одно из главных затруднений, с которым сталкиваются разработчики интерфейсов веб-приложений, состоит в том, что после того, как страница оказалась в браузере клиента, связь браузера с сервером заканчивается. Любое действие с элементом интерфейса требует повторного обращения к серверу с повторной загрузкой новой страницы. Из-за этого веб-приложение теряет свою элегантность и медленно работает. В данной статье я расскажу о том, как данную проблему можно решить с помощью JavaScript и объекта XMLHttpRequest.

Я уверен, что вам знакома традиционная модель интерфейса веб-приложений. Пользователь запрашивает страницу с сервера, которая на сервере создается, а затем пересылается браузеру. У данной страницы есть HTML-элементы, описывающие форму, в которую пользователь вводит данные. После этого пользователь отсылает данные на сервер и получает новую страницу, основанную на введенных данных, и процесс повторяется. Весь этот процесс определяется самой природой HTTP-протокола и отличается от того, как мы работаем с обычными приложениями, интерфейс которых неразрывно связан с программной логикой.

Возьмем простой пример ввода серийного номера в каком-либо Windows-приложении. Согласно правилам, после того, как вы закончите вводить замысловатый набор цифр и букв в поля, рядом с ними появится зеленая "галочка", означающая, что вы ввели правильный номер. Она появляется моментально, как результат логики "вшитой" в интерфейс. Как только вы закончили набирать номер, программа проверяет его и выдает ответ.

В веб-интерфейсе это стандартное поведение выглядит совершенно по-другому. Разумеется, поля, в которые вы вводите серийный номер выглядят точно так же, но по завершении ввода, пользователю надо, нажав кнопку, отправить страницу на сервер, который проверит введенные данные. Обратно пользователю вернется новая страница, где будет выведено сообщение о правильном или неправильном серийном номере. Пользователю в случае неудачи надо вернуться на предыдущую страницу и снова повторить попытку. И так до бесконечности.

Разумеется не часто веб-приложение требует от пользователя ввести серийный номер, но есть бесчисленное множество других примеров, где быстрая реакция интерфейса на действия пользователя очень бы пригодилась. А так как вся программная логика находится на сервере, получить такой результат в традиционном веб-приложении весьма сложно.

Благодаря JavaScript определенное количество программной логики можно перенести в HTML-страницу, что позволит быстро реагировать на действия пользователя. Однако у этого решения есть один главный недостаток. Первая проблема заключается в том, что как только JavaScript попадает в браузер пользователя вместе со страницей, программная логика доступна для просмотра невооруженным глазом. В случае например с проверкой правильности введенного адреса e-mail это может быть и не страшно, но если проверка связана с серийным номером, алгоритм проверки становится доступным всем, кто скачал страницу, а это неприемлемо.

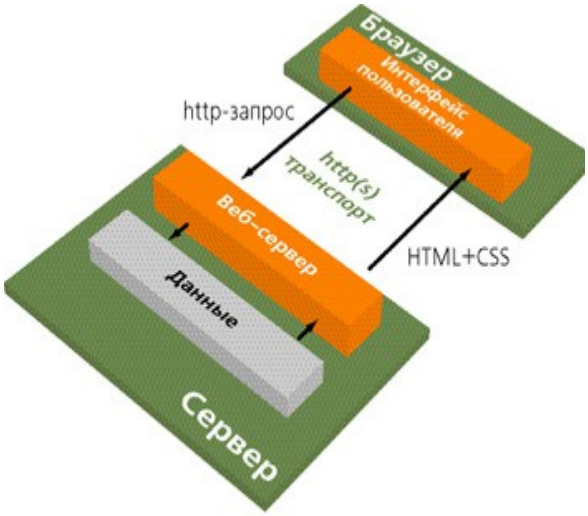
Вторая проблема заключается в том, что серьезную программную логику в страницу поместить невозможно, так как интерфейс просто не предназначен для этого. Вся логика должна находиться на уровне приложения, а не пользовательского интерфейса, а это значит - мы опять возвращаемся на сервер. Проблема дополняется еще тем, что не всегда с уверенностью можно ожидать наличия JavaScript в браузере клиента. В то время, как большинство пользователей оставляют поддержку JavaScript в своих браузерах включенной, существует значительное количество пользователей, которые этого не делают, или пользуются таким браузером, где JavaScript отсутствует или вообще не нужен как класс. Следовательно всю логику, которую делает JavaScript на стороне клиента, все равно придется проверять на сервере на всякий случай.

AJAX расшифровывается как **Asynchronous (асинхронный) JavaScript + XML** и технологией в строгом смысле слова не является. Это просто аббревиатура, обозначающая подход к созданию веб-приложений с помощью следующих технологий:

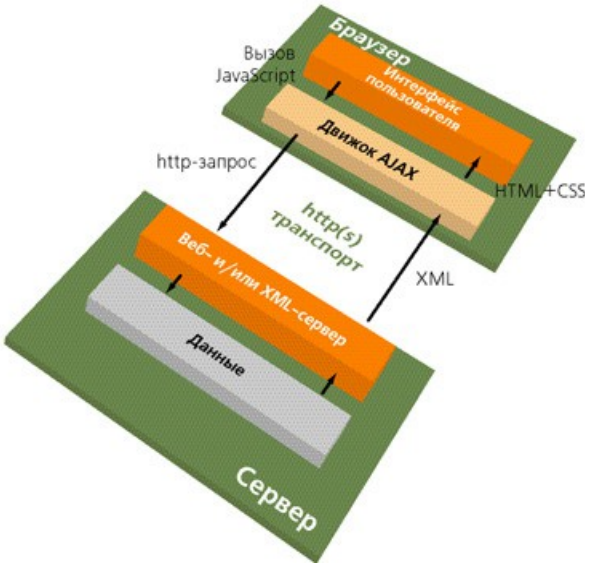
- стандартизированное представление силами XHTML и CSS;
- динамическое отображение и взаимодействие с пользователем с помощью DOM;
- обмен и обработка данных в виде XML и XSLT;
- асинхронные запросы с помощью XMLHttpRequest;
- JavaScript.

Если раньше на каждый запрос сервер выдавал новую страницу, то теперь он отсылает лишь те данные, которые нужны клиенту, а HTML из них прямо в браузере формирует AJAX Framework.

Асинхронность проявляется в том, что далеко не каждый клик пользователя доходит до сервера, причем обратное тоже справедливо - далеко не каждая реакция сервера обусловлена запросом пользователя. Большую часть запросов формирует AJAX Framework, причем его можно написать так, что он будет загружать информацию превентивно, предугадывая действия пользователя.



Так устроено классическое веб-приложение



Устройство веб-приложения, использующего AJAX

Понятно, что с такой схемой работы качественная нагрузка на сервер меняется - если раньше запросов было мало, но каждый из них требовал значительных ресурсов (серверу нужно вытащить информацию из БД, сформировать из нее веб-страницу и отдать браузеру), то теперь задача сервера упрощается (формировать веб-страницы не нужно, да и объем передаваемых данных меньше), но запросов обрабатывать приходится больше.

Почему именно AJAX

Желание сблизить Интернет и desktop приложения всегда занимало огромное количество программистов по всему миру, в связи с этим появлялись новые способы и технологии работы с Интернет интерфейсом. И если с чисто художественной и эстетической точки зрения эти технологии перевернули Веб, то с точки зрения взаимодействия с пользователем Интернет оставался прежним.

Что из этого получится

AJAX появился совсем недавно. Один из первых реально работающих проектов появился в 2004 году (тот самый Gmail). Саму аббревиатуру изобрели только в начале этого года. Однако все ключевые технологии, необходимые для создания AJAX-приложений, были известны давным-давно.

Пожалуй, было бы интересно поднять подшивки компьютерных изданий того времени. Наверняка только ленивый не пинал Microsoft за игнорирование общепринятых стандартов и продвижение собственных технологий. А прошло несколько лет, и оказалось, что вполне проприетарный XMLHttpRequest может оказаться весьма полезным, а кто его создал и почему - это уже вопрос десятый.

Среди причин, помешавших AJAX сразу занять заслуженное место под солнцем, можно назвать неготовность индустрии (в 1997 году возможности привычных технологий были еще далеко не исчерпаны, и множество программистов с удовольствием играли с Perl или PHP, не помышляя о смене архитектуры). Кроме того, заставить один и тот же код работать на всех браузерах в то время было невозможно - сейчас это сделать гораздо проще, хотя до сих пор это не тривиальная задача. Ну и наконец, клиентские машины стали мощнее, и если в 1997 году для многих Javascript был лишь примочкой, только замедляющей работу, то сегодня вполне можно передавать клиенту код из нескольких тысяч строк - если у пользователя нормальное

подключение, то на передачу кода времени уйдет немного, а выполняться он будет без задержек на любой современной машине.

Проблемы AJAX

На многих браузерах AJAX-приложения просто не работают (хотя последние версии IE, Firefox и даже Opera - начиная с восьмой версии - нормально его поддерживают). Кроме того, сейчас на AJAX возлагаются определенные надежды (по крайней мере, компанией Adaptive Path, которая придумала саму аббревиатуру и усиленно продвигает этот подход к созданию веб-приложений), и очень вероятно, что AJAX их оправдает не в полной мере.

Дело в том, что AJAX - не универсальная технология. Даже если отвлечься от совместимости, сама архитектура AJAX предполагает совершенно иной подход к созданию приложений, иной подход к созданию интерфейсов и, очевидно, годится далеко не для всех задач. Почему "очевидно", скептики объяснить не могут, но добавление в архитектуру приложения "посредника" между пользователем и хранилищем данных накладывает определенные, хотя и не совсем ясные пока ограничения. В любом случае, информация об ограничениях такого рода может быть добыта только методом проб и ошибок.

Есть и еще одно соображение. Добрый десяток лет веб-интерфейсы развивались в соответствии с установкой на ожидание ответа: каждый разработчик знал, что пользователю, нажавшему на любой элемент управления, придется ждать ответа сервера, и как-то старался уменьшить количество таких запросов, чтобы не заставлять пользователя тратить половину своего времени на ожидание загрузки страницы. С AJAX необходимость этого пусть и не исчезла совсем, но актуальность, по большому счету, потеряла. Более того, можно попытаться предположить, какое действие пользователь может сделать дальше - и подгрузить на всякий случай необходимые данные. Все эти предположения практически повторяют анализ Роберта Морриса из IBM, который еще три года назад заявил, что важнейшие инновации в программном обеспечении лежат не в технологиях, а в изменениях пользовательского интерфейса.

С AJAX проблема заключается в том, что писать такие приложения - задача трудоемкая. Нужно написать и отладить на JavaScript движок из десяти или двадцати тысяч строк кода плюс реализовать серверную часть. Причем копировать удачные решения практически не у кого: по большому счету, несколькими действительно масштабными AJAX-проектами может похвастаться только Google, но у них проекты довольно специфические - Google Suggest (сервис, подсказывающий наиболее популярные запросы), Gmail и Google Maps.