Кафедра: АСОИиУ

Лабораторная работа

На тему: Создание приложений на AJAX

Душанбе, 2009

## Обзор Ajax

Использование Ajax начинается с JavaScript-объекта, называемого XMLHttpRequest. Как и предполагает имя, он позволяет в клиентской части вашего кода выполнять HTTP-запросы и анализирует ответ XML-сервера. Первый шаг в изучении Ajax - создание данных объекта XMLHttpRequest. Метод протокола HTTP при использовании для запроса (GET или POST) и пути затем устанавливаются в объекте XMLHttpRequest.

Теперь вспомните, почему "а" в слове Ajax обозначает "асинхронный"? Когда вы посылаете HTTP-запрос, вы не хотите, чтобы браузер повис, ожидая ответа сервера. Вместо этого вы хотите также реагировать на действие пользователя на странице и работать с ответом сервера, когда тот в конечном итоге придет. Чтобы выполнить это, вам нужно зарегистрировать функцию обратного вызова с помощью XMLHttpRequest и послать асинхронно запрос XMLHttpRequest. Контроль останется за браузером, а функция обратного вызова будет вызвана, когда придет ответ сервера.

На Web-сервере запрос придет так же, как и любой другой HttpServletRequest. После анализа параметров запроса, сервлет выполнит все необходимые действия для приложения, сериализует его запрос в формате XML и вписывает его в HttpServletResponse.

Возвращаясь к клиенту, функция обратного вызова, зарегистрированная в XMLHttpRequest, теперь выполняется для того, чтобы обработать XML-документ, возвращаемый сервером. Наконец, пользовательский интерфейс обновляется в соответствии с данными сервера, используя JavaScript-код для преобразования страницы. Рисунок 1 - диаграмма последовательности изучения Ajax.

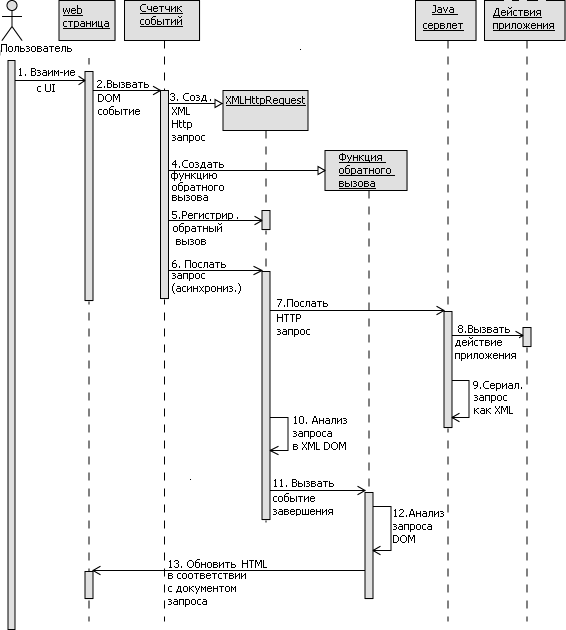


Рисунок 1. Обзор Ajax

Я начну с самого начала: создание XMLHttpRequest и отправка его из браузера. К сожалению, метод создания XMLHttpRequest отличается от браузера к браузеру. Функция в JavaScript в листинге 2 сглаживает эти трудности разных браузеров, определяя корректный подход для данного браузера и возвращая XMLHttpRequest готовым к использованию. Лучше всего думать об этом как о коде-шаблоне: простое копирование его в вашу библиотеку JavaScript и его использование, когда вам понадобится XMLHttpRequest.

Листинг 2. Создание XMLHttpRequest для разных браузеров

/\*

\* Возвращает новый XMLHttpRequest объект или false, если браузер его не поддерживает

\*/

function newXMLHttpRequest() {

var xmlreq = false;

if (window. XMLHttpRequest) {

// Создадим XMLHttpRequest объект для не-Microsoft браузеров

xmlreq = new XMLHttpRequest();

} else if (window. ActiveXObject) {

// Создадим XMLHttpRequest с помощью MS ActiveX

try {

// Попробуем создать XMLHttpRequest для поздних версий

// Internet Explorer

xmlreq = new ActiveXObject("Msxml2. XMLHTTP");

} catch (e1) {

// Не удалось создать требуемый ActiveXObject

try {

// Пробуем вариант, который поддержат более старые версии

// Internet Explorer

xmlreq = new ActiveXObject("Microsoft. XMLHTTP");

} catch (e2) {

// Не в состоянии создать XMLHttpRequest с помощью ActiveX

}

}

}

return xmlreq;

}

Позже мы обсудим технические приемы для браузеров, которые не поддерживают XMLHttpRequest. А сейчас примеры предполагают, что функция newXMLHttpRequest из листинга 2 всегда возвращает значение XMLHttpRequest.

Возвращаясь обратно к сценарию примера корзины покупателя, я бы хотел воспользоваться Ajax для случаев, когда пользователь нажимает Add to Cart кнопку для каталога продуктов. Оператор onclick, вызванный addToCart(), ответственен за обновление состояния корзины в Ajax-вызове (см. листинг 1). Как показано в листинге 3, первое, что необходимо сделать addToCart(), - получить данные XMLHttpRequest с помощью вызова функции newXMLHttpRequest() из листинга 2. Следующим шагом он регистрирует функцию обратного вызова для получения ответа сервера, (я объясню это подробно позже, смотрите листинг 6).

Поскольку запрос изменит состояние на сервере, я буду пользоваться методом HTTP POST, чтобы это выполнить. Отправка данных методом POST выполняется в три шага. Во-первых, мне необходимо открыть соединение с источником сервера, с которым я соединяюсь, что в данном случае является сервлетом, занесенным в cart. do страницы. Далее я устанавливаю заголовок в XMLHttpRequest, говорящий о том, что содержимое запроса - закодированные данные. И, наконец, я посылаю запрос с закодированными данными в теле содержимого.

Листинг 3 показывает все эти шаги вместе.

Листинг 3. Посылаем XMLHttpRequest для кнопки Add to Cart

/\*

\* Добавляет продукт, определенный его кодом,

\* в корзину покупателя

\* itemCode - код продукта для добавления.

\*/

function addToCart(itemCode) {

// Возвращает содержимое XMLHttpRequest

var req = newXMLHttpRequest();

// Оператор для получения сообщения обратного вызова

// из объекта запроса

var handlerFunction = getReadyStateHandler(req, updateCart);

req. onreadystatechange = handlerFunction;

// Открываем HTTP-соединение с помощью POST-метода к сервлету корзины покупателя.

// Третий параметр определяет, что запрос асинхронный.

req. open("POST", "cart. do", true);

// Определяет, что в содержимом запроса есть данные

req. setRequestHeader("Content-Type",

"application/x-www-form-urlencoded");

// Посылаем закодированные данные, говорящие о том, что я хочу добавить

// определенный продукт в корзину.

req. send("action=add&item="+itemCode);

}

После всего этого вы поняли первую часть в механизме Ajax - собственно создание и передача HTTP-запроса от клиента. Следующим шагом будет код сервлета на языке Java для обработки запроса.

## В начало. Обработка запроса сервлета

Обработка XMLHttpRequest с помощью сервлета такая же, как и обработка обычного HTTP-запроса из браузера. Закодированные данные, отсылаемые в содержимом запроса POST, могут быть получены с помощью вызовов HttpServletRequest. getParameter(). Ajax запрашивает участие в HttpSession, такой же, как и в регулярных Web запросах из приложения. Это полезно для примера сценария корзины покупателя, поскольку позволяет мне заключить состояние пользовательской корзины покупателя в JavaBeans и удерживать это состояние во время сессии между двумя запросами.

Листинг 4 - часть простого сервлета, который обрабатывает Ajax запросы, чтобы обновить корзину покупателя. Cart получается из сессии пользователя и его состояние обновляется в соответствии с параметрами запроса. Cart затем сериализуется в XML-формате, а XML образуется в ServletResponse. Важно установить тип содержимого запроса в application/xml, в противном случае XMLHttpRequest не будет анализировать содержимое ответа.

Листинг 4. Код сервлета для обработки Ajax-запросов

public void doPost(HttpServletRequest req,

HttpServletResponse res)

throws java. io. IOException {

Cart cart = getCartFromSession(req);

String action = req. getParameter("action");

String item = req. getParameter("item");

if ((action! = null) &&(item! = null)) {

// Добавить или удалить продукт из Cart

if ("add". equals(action)) {

cart. addItem(item);

} else if ("remove". equals(action)) {

cart. removeItems(item);

}

}

// Сериализуем состояние Cart в XML-формате

String cartXml = cart. toXml();

// Записываем полученный XML в запрос.

res. setContentType("application/xml");

res. getWriter(). write(cartXml);

}

Листинг 5 показывает пример XML-кода, полученный с помощью метода Cart. toXml(). Сделать это достаточно просто. Запомните атрибут generated в cart элементе, который является временной меткой, полученной с помощью System. currentTimeMillis().

Листинг 5. Пример сериализации в XML-формате объекта Cart

<? xml version="1.0"? >

<cart generated="1123969988414" total="$171.95">

<item code="hat001">

<name>Hat</name>

<quantity>2</quantity>

</item>

<item code="cha001">

<name>Chair</name>

<quantity>1</quantity>

</item>

<item code="dog001">

<name>Dog</name>

<quantity>1</quantity>

</item>

</cart>

Если вы взглянете на Cart. java в источнике приложения, доступном в секции Загрузка, вы увидите, что XML получается просто с помощью соединения строк вместе. Примененный в примере, этот способ - зачастую самый худший способ получить XML из кода на языке Java. Я предложу приемы интереснее и лучше в следующей части этой серии.

Теперь вы знаете, что CartServlet запрашивается с помощью XMLHttpRequest. Следующим шагом будет возвращение к коду клиента, где вы можете увидеть, как ответ в XML-формате будет использоваться для обновления состояния страницы.

## Запрос, обрабатывающийся с помощью JavaScript

readyState свойство XMLHttpRequest - числовое значение, которое определяет статус цикла запроса. Оно изменяется от 0 для "неопределенного" и до 4 для "завершенного". Каждый раз, когда меняется readyState, появляется событие readystatechange и вызывается с помощью onreadystatechange свойства оператор.

В листинге 3 вы увидели, как вызывалась функция getReadyStateHandler() для создания оператора. Этот оператор затем приписывается свойству onreadystatechange. getReadyStateHandler() использовал тот факт, что функции - объекты первого класса в JavaScript. Это обозначает, что функции могут быть параметрами к другим функциям и могут также создавать и возвращать значения других функций. Обязанности getReadyStateHandler() - возвращать значение функции, которая проверяет, завершился ли XMLHttpRequest и был отослан ли ответ в XML-формате в оператор, определенный вызовом. Листинг 6 - код для getReadyStateHandler().

Листинг 6. Функция getReadyStateHandler(). /\*

\* Возвращает функцию, которая ожидает, чтобы закончился

\* определенный XMLHttpRequest, затем передает его ответ в XML-формате

|-------10--------20--------30--------40--------50--------60--------70--------80--------9|

|------ - XML error: The previous line is longer than the max of 90 characters - --------|

\* заданному оператору \* req - это XMLHttpRequest, чье состояние изменяется

|-------10--------20--------30--------40--------50--------60--------70--------80--------9|

|------ - XML error: The previous line is longer than the max of 90 characters - --------|

\* responseXmlHandler - функция, передаваемая запрос XML

\*/

function getReadyStateHandler(req, responseXmlHandler) {

// Возвращает неопределенную функцию, которая считывает

// данные XMLHttpRequest return function () {

// Если требуется статус "закончен"

if (req. readyState == 4) {

// Проверяем, пришел ли успешный ответ сервера

if (req. status == 200) {

// Передает XML оператору

responseXmlHandler(req. responseXML);

} else {

// Возникла ошибка HTTP

alert("HTTP error: "+req. status);

}

}

}

}

О getReadyStateHandler()

getReadyStateHandler() - относительно сложный кусок кода, особенно если вы ранее не знали JavaScript. Но с другой стороны преимущество в том, что с помощью включения этой функции в вашу библиотеку JavaScript вы можете спокойно обрабатывать Ajax-запросы сервера без работы с данными XMLHttpRequest. Важно также, что вы понимаете, как использовать getReadyStateHandler() в вашем собственном коде.

В листинге 3 вы видели getReadyStateHandler(), вызванный так: handlerFunction = getReadyStateHandler(req, updateCart). Функция, возвращаемая с помощью getReadyStateHandler(), в этом случае проверит, выполнен ли XMLHttpRequest в переменной req и затем вызовет функцию, называющуюся updateCart, с помощью запроса в формате XML.

## Данные корзины

Листинг 7 - собственно сам код updateCart(). Функция запрашивает XML-документ корзины покупателя, используя DOM-вызовы и обновляя Web-страницу (см. листинг 1) для того, чтобы показать новые элементы содержимого новой корзины. Обратите здесь внимание на вызовы, использованные для того, чтобы достать данные из XML DOM. Атрибут generated элемента cart, созданный, когда Cart был сериализован в формате XML, проверяется, чтобы удостовериться, что более новые данные корзины не перезаписаны более старыми данными. Ajax-запросы по своей сути асинхронны, поэтому эта проверка предохраняет против ответов сервера, которые прибывают в беспорядке.

Листинг 7. Обновление страницы с отображением XML документа корзины

function updateCart(cartXML) {

// Получить корневой &quot; cart&quot; элемент из документа

var cart = cartXML. getElementsByTagName("cart") [0] ;

// Проверим, что более ранний документ корзины не был обработан еще

var generated = cart. getAttribute("generated");

if (generated > lastCartUpdate) {

lastCartUpdate = generated;

// Очистим список HTML, необходимый для отображения содержимого корзины

var contents = document. getElementById("cart-contents");

contents. innerHTML = "";

// Соединяем продукты в корзине

var items = cart. getElementsByTagName("item");

for (var I = 0; I < items. length; I++) {

var item = items [I] ;

// Достаем ключевые понятия из имени и элементов количества

var name = item. getElementsByTagName("name") [0]

. firstChild. nodeValue;

var quantity = item. getElementsByTagName("quantity") [0]

. firstChild. nodeValue;

// Создаем и добавляем список продуктов, HTML элемент для этого продукта

var li = document. createElement("li");

li. appendChild(document. createTextNode(name+" x "+quantity));

contents. appendChild(li);

}

}

// Обновляем итого в корзине, используя значение из документа корзины

document. getElementById("total"). innerHTML =

cart. getAttribute("total");

}

После того, что мы проделали, путешествие в Ajax закончено, хотя вы могли бы захотеть получить Web-приложения и увидеть их в действии (смотрите секцию Загрузка). Пример очень прост, многие строки в нем еще можно улучшить. К примеру, я включил код сервера, чтобы убрать продукты из корзины, но не сделал доступ к ним из интерфейса. Для вас будет хорошим упражнением попытаться построить в приложении существующего JavaScript-кода данную функциональность.

## Задачи использования Ajax

Как и во многих других технологиях, в Ajax есть множество способов сделать ошибки. Некоторые из проблем, которые я обсуждаю здесь, в данный момент имеют удачные и легкие решения, но могут быть улучшены в качестве Ajax-продуктов. Поскольку сообщество разработчиков дает опыт в разработке Ajax-приложений, лучшие практики и справочники будут записаны.

## Польза XMLHttpRequest

Одной из важных задач, встающих перед лицом Ajax-разработчиков, является вопрос, как ответить, когда XMLHttpRequest недоступен. В то время как большинство современных браузеров поддерживают XMLHttpRequest, всегда есть пользователи, у которых XMLHttpRequest не поддерживается или чья безопасность браузера не дала выполниться XMLHttpRequest. Если вы разрабатываете Web-приложение, которое применимо в корпоративной сети, вероятно, у вас есть преимущество в определении, какие браузеры поддерживаются и когда XMLHttpRequest всегда доступен. Если вы разрабатываете для публичной сети, то вы должны знать, что, предполагая, что XMLHttpRequest доступен, вы отрекаетесь от пользователей более старых браузеров, браузеров людей с закрытым доступом или легковесных браузеров с сервисом против выполнения вашего приложения.

Поэтому вам необходимо попытаться сделать ваше приложение с "изящным ухудшением", чтобы остаться функциональным даже в браузерах без поддержки XMLHttpRequest. В примере корзины покупателя лучший способ ухудшить приложение - кнопки Add to Cart, выполняющие обычную роль, обновляя страницу для нового статуса корзины. С помощью Ajax можно добавить в страницу через JavaScript-код, когда страница загружена, прикрепить оператор JavaScript к каждой кнопке Add to Cart, если только доступен XMLHttpRequest. Другой способ - определить XMLHttpRequest, когда пользователь вошел, и затем обеспечить любую Ajax версию приложения или версию, основанную на обычных формах, если понадобится.

## Относительно функциональности (usability)

Некоторые из задач функциональности в Ajax - достаточно общие. К примеру, может быть, важно позволить пользователям узнать, что их ввод зарегистрирован, поскольку обычный механизм обратной связи ("песочные часы" курсора и "крутящийся браузер") не применим для XMLHttpRequest. Выход - заменить кнопки Подтвердить сообщением "Обновляется…", чтобы пользователи не нажимали несколько раз на кнопки в ожидании запроса.

Другая задача состоит в том, что пользователи могут не заметить, что части страницы, которые они видят, уже обновлены. Вы можете решить эту проблему, используя множество визуальных техник и приемов, например, нарисовать глаз пользователя к обновленным местам страницы. Другие задачи, вызванные обновлением страницы при Ajax, включают в себя нерабочую кнопку возвращения обратно, некорректное отображение адреса или невозможность добавить в избранное. Смотрите секцию Ресурсы для статей, которые специализируются на задачах функциональности в Ajax-приложениях.

## Перегрузка сервера

Выполнение Ajax-интерфейса в месте, где действуют обычные формы, может привести к тому, что неожиданно возрастет количество запросов к серверу. К примеру, обычный Google Web-поиск вызывает один запрос на сервер, когда пользователь нажимает кнопку поиска. Однако, Google Suggest, который пытается автоматически закончить ваше действие поиска, посылает несколько запросов к серверу, пока пользователь печатает. Когда вы разрабатываете Ajax-приложение, помните, сколько запросов вы отошлете серверу, и возникнет ли из-за этого перегрузка сервера. Вы можете уменьшить перегрузку сервера, храня запросы на стороне клиента и пряча запросы в клиенте, если это возможно. Вы можете также попытаться разработать Ajax-приложения так, чтобы большая часть действия могла быть выполнена в клиентской части вашего кода без необходимости соединения с сервером.

## Разбираясь с асинхронностью

Очень важно понять, что нет гарантии, что XMLHttpRequest закончатся в той самой очередности, в которой они были отправлены. В самом деле, вам необходимо допустить, что они не вернутся так, и разработать ваше приложение с учетом этой поправки. В примере корзины покупателя время отметки последнего обновленного продукта раньше гарантировало, что более новые данные корзины не могут быть переписаны более старыми, (см. листинг 7). Этот недоработанный подход работает для сценария корзины покупателя, но может не выполняться в других сценариях.

## Заключение

Вы, должно быть, сейчас хорошо понимаете основные принципы программирования в Ajax и особенности компонент сервера и клиента, которые имеют место в Ajax. Это фундамент Ajax-приложения на языке Java. В добавление к сказанному вам следует понимать некоторые из задач разработки более высокого абстрактного уровня, которые появляются с техническими приемами в Ajax. Создание успешного Ajax-приложения потребует общего подхода, начиная с разработки интерфейсов с помощью JavaScript-кода и заканчивая архитектурой сервера, но вы теперь вооружены основными принципами программирования в Ajax, необходимыми для рассмотрения всех этих аспектов.

Очень хорошо, если вы чувствуете уныние из-за сложности написания больших Ajax-приложений с использованием приемов, продемонстрированных здесь. Такие структуры, как Struts, Spring и Hibernate, развиваются, разрабатывая мелкие детали Servlet API и JDBC, поэтому появляются средства для облегчения Ajax-разработки. Некоторые из них сфокусированы исключительно на проблемах клиентской стороны кода, находя легкие пути для добавления визуальных эффектов к вашим страницам или тщательно анализируя использование XMLHttpRequest. Некоторые идут дальше, разрабатывая средства для автоматической генерации Ajax-интерфейсов со стороны сервера. Эти структуры выполняют тяжелую работу за нас, поэтому вы можете сконцентрироваться на подходах более высокого уровня. Я рассмотрю некоторые из них в этой серии.

Сообщество Ajax быстро развивается, и здесь есть большое количество важной информации. Перед тем как читать следующую часть этой серии статей, я бы рекомендовал вам ознакомиться со статьями в секции Ресурсы, особенно если вы - "новичок" в Ajax или в разработке клиентской части кода. Также вам понадобится время разобраться с примером кода и подумать о способах его улучшения.

В следующей статье этой серии я рассмотрю XMLHttpRequest более детально и предложу способы создания XML сразу из вашего JavaBeans. Я также покажу вам альтернативы к XML для передачи Ajax-данных, такие как нотация JSON.