Лабораторная работа №2

1 Основы технологии AJAX

Цель работы: изучить основы технологии AJAX. Написать простейшие клиентское и серверное приложение обменивающиеся ассинхронными данными.

Краткие теоретические сведения

AJAX — это модное название для набора техник разработки веб-интерфейсов, позволяющих делать динамические запросы к серверу без видимой перезагрузки веб-страницы: пользователь не замечает, когда его браузер запрашивает данные.

AJAX обеспечивает динамичность и асинхронность web-разработок при отсутствии необходимости обновления страниц.

Пожалуй, любой разработчик мечтает о том, чтобы превратить обычную, неновую web-страничку во что-то более захватывающее. Сейчас можно попробовать вдохнуть немного жизни в web-технологии десятилетней давности, для этого познакомся с AJAX.

При использовании Google или web-клиента Gmail вам уже приходилось сталкиваться с решением, основанном на AJAX. Это технология, которая обеспечивает динамическое и асинхронное поведение, где исключается обновление страниц. Посредством AJAX пользователь может взаимодействовать с web-страницами, подобно работе клиентов с более богатыми возможностями.

В настоящее время многие говорят о AJAX. Технология, следующая за AJAX, вряд ли является новшеством, однако недавно стали появляться некоторые мощные новые приложения, использующие объект XMLHttpRequest, они вдохнули новую жизнь в концепцию обновления образа клиентской части.

Самым примечательным из этих новых приложений является Google Maps. Пользуясь им, можно находить определенную местность на карте планеты, затем переходить к более мелким объектам, прокручивать перетягивать карту без необходимости обновления страницы.

AJAX — это аббревиатура от Asynchronous JavaScript and XML (и DHTML, и т.д.). (как это представил Джис Джеймс Гаррет (Jesse James Garrett), он первым ввел термин 'AJAX' для асинхронного JavaScript + XML). Т.е. это коллекция технологий, существующих с момента появления Web.

Возможности, предоставляемые AJAX:

- Стандартно-базированная презентация с использованием XHTML и CSS;
- Динамическое отображение и взаимодействие с использованием объектной модели документа;
- Взаимообмен данными и манипуляция с задействованием XML и XSLT;
- Асинхронное извлечение данных с использованием XMLHttpRequest;
- JavaScript, связывающий все вместе.

Вкратце AJAX позволяет писать быстрореагирующие веб-приложения, в которых не нужно постоянно обновлять страницы. AJAX — простая технология, поддерживаемая всеми основными браузерами. Как можно вкратце отметить, единственным предварительным условием для внедрения AJAX является знание

JavaScript.

Как работает AJAX

Если вы когда-либо пользовались веб-клиентом Gmail или Google Maps, то замечали возможность проверки правописания и прокрутки по всему изображению, соответственно, без обновления страниц. AJAX — это технология, которая обрабатывает операции в JavaScript и асинхронно запускает на стороне сервера операции, предоставляющие желаемый результат.

В основе технологии AJAX лежит объект XMLHttpRequest. Изначально он появился в Internet Explorer, а затем — в Mozilla/Safari. На момент написания этой статьи уже была поставлена 8-я совместимая версия Opera. Однако, Opera в свое время отличилась нестабильностью с точки зрения реализации XMLHttpRequest.

Объект XMLHttpRequest

Первый объект, о котором вы хотите узнать, возможно, самый новый для вас; он называется XMLHttpRequest. Это объект JavaScript, и он создается так же просто, как показано в листинге 1.

Листинг 1. Создание нового объекта XMLHttpRequest

```
<script language="javascript" type="text/javascript">
var request = new XMLHttpRequest();
</script>
```

Это объект, который управляет всем вашим взаимодействием с сервером – это технология JavaScript в объекте XMLHttpRequest, который общается с сервером. Это не обычный ход работы приложения, и именно здесь заключается почти вся магия Ajax.

В нормальных Web-приложениях пользователи заполняют поля форм и нажимают кнопку *Submit* (подтвердить). Затем форма передается на сервер полностью, сервер обрабатывает сценарий (обычно PHP или Java, возможно, CGI-процесс или что-то в этом роде), а потом передает назад всю новую страницу. Эта страница может быть HTML-страницей с новой формой с некоторыми заполненными данными, либо страницей подтверждения, либо, возможно, страницей с какими-то выбранными вариантами, зависящими от введенных в оригинальную форму данных. Естественно, пока сценарий или программа на сервере не обработается и не возвратится новая форма, пользователи должны ждать. Их экраны очистятся и будут перерисовываться по мере поступления новых данных от сервера. Вот где проявляется низкая интерактивность – пользователи не получают немедленной обратной реакции и определенно чувствуют себя не так, как при работе с настольными приложениями.

Ajax по существу помещает технологию JavaScript и объект XMLHttpRequest *между* вашей Web-формой и сервером. Когда пользователи заполняют формы, данные передаются в какой-то JavaScript-код, а не прямо на сервер. Вместо этого JavaScript-

код собирает данные формы и передает запрос на сервер. Пока это происходит, форма на экране пользователя не мелькает, не мигает, не исчезает и не блокируется. Другими словами, код JavaScript передает запрос в фоновом режиме; пользователь даже не замечает, что происходит запрос на сервер. Более того, запрос передается асинхронно, а это означает, что ваш JavaScript-код (и пользователь) не ожидают ответа сервера. То есть, пользователи могут продолжать вводить данные, прокручивать страницу и работать с приложением.

Затем сервер передает данные обратно в ваш JavaScript-код (все еще находящийся в вашей Web-форме), который решает, что делать с данными. Он может обновить поля формы "на лету", придавая свойство немедленности вашему приложению – пользователи получают новые данные без подтверждения или обновления их форм. JavaScript-код может даже получить данные, выполнить какие-либо вычисления и передать еще один запрос, и все это без вмешательства пользователя! В этом заключается мощь XMLHttpRequest. Он может общаться с сервером по своему желанию, а пользователь даже не догадывается о том, что происходит на самом деле. В результате мы получаем динамичность, чувствительность, высокую интерактивность настольного приложения вместе со всеми возможностями интернет.

Добавление JavaScript-кода

После того, как вы разберетесь с XMLHttpRequest, оставшийся JavaScript-код превращается в рутинную работу. Фактически, вы будете использовать JavaScript-код для небольшого числа основных задач:

- Получить данные формы: JavaScript-код упрощает извлечение данных из вашей HTML-формы и передает их на сервер.
- Изменить значения в форме: Форма обновляется тоже легко, от установки значений полей до замены изображений "на лету".
- Выполнить анализ HTML и XML: Вы будете использовать JavaScript-код для управления DOM и для работы со структурой вашей HTML-формы и всеми XML-данными, возвращаемыми сервером.

Для выполнения первых двух задач вы должны очень хорошо знать метод getElementById(), приведенный в листинге 2.

Листинг 2. Сбор и установка значений полей при помощи JavaScript-кода

```
// Получить значение поля "phone" и записать его в переменную phone
var phone = document.getElementById("phone").value;

// Установить значения в форме, используя массив response
document.getElementById("order").value = response[0];
document.getElementById("address").value = response[1];
```

Вы должны начать понимать, что нет ничего чрезмерно сложного во всем этом. Как только вы освоите XMLHttpRequest, оставшаяся часть вашего Ajax-приложения будет простым JavaScript-кодом, похожим на приведенный в листинге 2, смешанным с немного более умным HTML. К тому же, время от времени есть немного работы с DOM. Итак, давайте рассмотрим это.

Получение объекта Request

Поскольку XMLHttpRequest является центральным для Ajax-приложений (и, возможно, нов для многих из вас) начнем с него. Как вы видели в листинге 1, создать этот объект и использовать его должно быть просто, не правда ли? Подождите минуточку.

Помните те ужасные войны браузеров, происходившие несколько лет назад, и как ничто не работало одинаково в разных браузерах? Поверите вы или нет, но те войны продолжаются до сих пор, хотя и с намного меньшим масштабом. И, сюрприз: XMLHttpRequest – это одна из жертв этих войн. Поэтому вы должны выполнить несколько различных действий для обеспечения возможности работы XMLHttpRequest.

Работа с браузерами Microsoft

Браузер Microsoft Internet Explorer для обработки XML использует анализатор MSXML. Поэтому, когда вы пишете Ajax-приложения, которые должны работать в Internet Explorer, необходимо создать объект особым способом.

Однако, это не так то и легко. На самом деле в ходу две различных версии MSXML. Версия MSXML зависит от версии технологии JavaScript, установленной в Internet Explorer, поэтому вам нужно написать код, подходящий для обеих версий. Взгляните на листинг 3, в котором приведен код для создания XMLHttpRequest в браузерах Microsoft.

Листинг 3. Создание объекта XMLHttpRequest в браузерах Microsoft

```
var request = null;

function createRequest () {
 try {
  request = new ActiveXOblect("Msxml2.request");
 } catch (othermicrosoft) {
  try {
   request = new ActiveXObject("Microsoft.request");
  } catch (failed) {
   request = null;
  }
 }
}
```

Все это пока может не иметь смысла, но это нормально. Пока вы должны записать в своей голове две основных строки:

request = new ActiveXObject("Msxml2.request");

и

request = new ActiveXObject("Microsoft.request");.

В двух словах, этот код пытается создать объект, используя одну версию MSXML; если это не получится, создается объект для второй версии. Изящно, да? Если ничего не сработало, переменная request устанавливается в false, для того чтобы указать вашему коду, что что-то не так. В этом случае вы, возможно, работаете с браузером не от Microsoft и должны использовать другой код для выполнения работы.

Работа с Mozilla и браузерами не от Microsoft

Если Internet Explorer не ваш браузер, либо вы пишете код для браузеров не от Microsoft, вам нужен другой код. Фактически, это простая строка, которую вы видели в листинге 1:

var request = new XMLHttpRequest object;.

Эта намного более простая строка создает объект XMLHttpRequest в Mozilla, Firefox, Safari, Opera и в большой степени в каждом браузере не от Microsoft, поддерживающем Ajax в любой форме или разновидности.

Объединение

Мы хотим поддерживать *все* браузеры. Кто хочет писать приложение, работающее только в Internet Explorer, или приложение, работающее только во всех остальных браузерах? Еще хуже, хотите ли вы написать ваше приложение дважды? Конечно, нет! Итак, объединим поддержку для Internet Explorer и для остальных браузеров. В листинге 4 приведен код, делающий это.

Листинг 4. Создание объекта XMLHttpRequest для всех браузеров

```
var request = null;

function createRequest () {

 try {
  request = new XMLHttpRequest();
 } catch (trymicrosoft) {
  try {
   request = new ActiveXOblect("Msxml2.request");
  } catch (othermicrosoft) {
   try {
    request = new ActiveXObject("Microsoft.request");
   } catch (failed) {
    request = null;
   }
  }
 }

 if (request == null) alert("Ошибка создания XMLHttpRequest объекта!");
}
```

Основу этого кода можно разделить на три шага:

1. Создаем переменную request для ссылки на объект XMLHttpRequest, который вы создадите.
2. Создаем объект для неMicrosoft браузеров, если это не получается, то переходим к следующему пункту.
3. В блоке try создайте объект в браузерах Microsoft:
    - В блоке try создайте объект с использованием объекта Msxml2.request.
    - Если это не получится, В блоке try создайте объект с использованием объекта Microsoft.request.

В конце этого процесса request должен ссылаться на корректный объект XMLHttpRequest, независимо от используемого пользователем браузера.

Запрос/ответ в мире Ajax

Итак, вы уже знакомы с Ajax и имеете базовое представление об объекте XMLHttpRequest и о том, как создать его. Если вы читали внимательно, то вы даже понимаете, что это технология JavaScript общается с любым Web-приложением на сервере, а не ваша HTML-форма, которую вы подтвердили напрямую.

Что мы пропустили? Как на самом деле использовать XMLHttpRequest. Поскольку это критический код, который вы будете использовать в некоторых формах в *каждом* вашем Ajax-приложении, рассмотрим коротко, как выглядит базовая модель запрос/ответ в Ajax.

Выполнение запроса

У вас есть ваш новый объект XMLHttpRequest; приведем его в движение. Во-первых, нам понадобится JavaScript-метод, который ваша Web-страница может вызвать (например, когда пользователь вводит текст или выбирает вариант из меню). Затем, нужно следовать одной и той же основной схеме практически во всех ваших Ajax-приложениях:

1. Получить какие-либо данные из Web-формы.
2. Создать URL для подключения.
3. Открыть соединение с сервером.
4. Установить функцию для сервера, которая выполнится после его ответа.
5. Передать запрос.

В листинге 5 приведен пример Ajax-метода, который выполняет именно эти операции и именно в этом порядке:

Листинг 5. Выполнить запрос с Ajax

```
function getResult() {

//Создать URL для подключения
var url = "test.php";
```

```
 //Создаем объект XMLHttpRequest
 createRequest();

 //Открыть соединение с сервером
 request.open("GET", url, true);

 //Установить функцию для сервера, которая будет выполнена после его
ответа
 request.onreadystatechange = updatePage;

 //Отправить запрос
 request.send(null);
}
```

Многое из этого не требует пояснений. Код устанавливает PHP-сценарий в качестве URL для подключения. Затем открывается соединение; это первое место, где вы опять увидели в действии XMLHttpRequest. Указывается метод соединения (GET) и URL. Последний параметр, когда установлен в true, запрашивает асинхронное соединение (то есть, делает это способом, сооветствующим названию Ajax). При использовании false код ждал бы выполнения запроса и не продолжал бы работу до получения ответа. При использовании true ваши пользователи могут работать с формой (и даже вызывать другие JavaScript-методы) пока сервер обрабатывает этот запрос в фоновом режиме.

Свойство onreadystatechange request (вспоминайте, это ваш экземпляр объекта XMLHttpRequest) позволяет вам информировать сервер о том, что следует делать после завершения работы (что может быть через пять минут или через пять часов). Поскольку код не собирается ждать сервер, вы должны дать серверу знать, что делать, так чтобы вы смогли среагировать. В данном случае будет инициирован конкретный метод (называемый updatePage()) после завершения сервером обработки вашего запроса.

Наконец, вызывается send() со значением null. Поскольку вы добавили данные для передачи на сервер в URL запроса, вам не надо передавать что-либо в запросе. Таким образом, передается ваш запрос, и сервер может делать то, что вы указали ему делать.

Если вы кроме этого ничего больше не делаете, обратите внимание на то, насколько все просто и понятно! В отличие от осознания вами асинхронной природы Ajax, все это действительно простые вещи. Вы оцените то, как это освобождает вас для концентрации внимания на крутых приложениях и интерфейсах, а не на сложном HTTP-коде запроса/ответа.

Код в листинге 5 очень прост. Данные являются простым текстом и могут быть включены как часть URL-запроса. GET посылает запрос вместо более сложного POST. Не добавляется XML, заголовки контента, не передаются данные в теле запроса.

На странице test.php, которую мы указываем в запросе производится либо обработка принятой информации, если таковая передавалась в запросе, например, как test.php?city=chernihiv, либо просто возвращается результат.

Листинг 5.1 пример test.php

```
<?php

//Sending zipCode
echo rand(10000, 99999);

?>
```

Обработка ответа

Теперь вы должны разобраться с ответом сервера. Пока вы должны знать только два момента:

- Не делать ничего, пока свойство request.readyState не будет равно 4.
- Сервер будет записывать свой ответ в свойстве request.responseText.

Первый момент (состояния готовности) – о нем вы должны сами разобраться, т.е. уточнить какие бывают стадии HTTP-запроса. Пока вы просто проверяйте на равенство определенному значению (4), и все будет работать. Второй момент (использование свойства request.responseText для получения ответа от сервера) является простым. В листинге 6 приведен пример метода (который сервер может вызвать), основанного на значениях, переданных в листинге 5.

Листинг 6. Обработка ответа от сервера

```
function updatePage() {
 if(request.readyState == 4) {
  var response = request.responseText;
  //Что-то делаем с пришедшими нам данными, например, присваиваем
их полю с id="zipCode"
  document.getElementById("zipCode").value = response;
 }
}
```

Опять же, код не является трудным или сложным. Он ожидает, пока сервер не вызовет его с нужным состоянием готовности, и затем использует значение, которое сервер возвращает, для установки другого поля формы. В результате поле zipCode неожиданно появляется с ZIP-кодом, но пользователь *ни разу не щелкнул по кнопке*! Это поведение настольного приложения, о чем мы говорили ранее.

Вы, возможно, заметили, что поле zipCode является обычным текстовым полем. После возврата сервером ZIP-кода и установки этого поля методом updatePage() в значение ZIP-кода города/штата пользователи *могут* переопределить это значение. Так сделано умышленно по двум причинам: сохранить этот пример простым и показать вам, что иногда нужно, чтобы пользователи имели возможность

переопределить значения, возвращенные сервером. Помните об обоих моментах; они важны при хорошем дизайне пользовательского интерфейса.

Перехват в Web-формах

Что нам осталось? В сущности, не много. Вы имеете JavaScript-метод, собирающий введенную пользователем в форму информацию, передаете ее серверу, предоставляете еще один JavaScript-метод для обработки ответа и даже устанавливаете значение поля, когда этот ответ приходит. Все что осталось на самом деле – *вызвать* этот первый метод и запустить полный процесс. Вы могли бы, очевидно, добавить кнопку в вашу HTML-форму, но это же старый, добрый 2001 год, не так ли? Воспользуемся возможностями технологии JavaScript, как показано в листинге 7.

Листинг 7. Запуск Ajax-процесса

```
<form>
 <p>City: <input type="text" name="city" id="city" size="25"
onChange="getResult();" /></p>
 <p>Zip Code: <input type="text" name="zipCode" id="zipCode" size="5"
readonly /></p>
</form>
```

Для более легкой работы с HTML при помощи JavaScript были разработаны несколько фреймворков, которые упрощают повседневные рутинные функции и помогают программисту не задумываться о реализации примитивов, например, доступа к полям DOM-а, их изменение стандартными средствами JavaScript, а предоставляют удобные инструменты для данных функций. Из наиболее известных на сегодняшний день стоит отметить два фреймворка: Ext JS и jQuery.

ExtJS

Ext JS — библиотека JavaScript для разработки веб-приложений и пользовательских интерфейсов, изначально задуманная как расширенная версия Yahoo! UI Library, однако преобразовавшаяся затем в отдельный фреймворк. Использует адаптеры для доступа к библиотекам Yahoo! UI Library, jQuery или Prototype/script.aculo.us. Поддерживает технологию AJAX, анимацию, работу с DOM, реализацию таблиц, вкладок, обработку событий и все остальные новшества «Web 2.0».

С версии 2.1 библиотека ExtJS распространяется по условиям трёх лицензий: Commercial License, Open Source License и OEM / Reseller License.

Начиная с версии Ext JS 3.0 библиотека разбивается на две части: Ext Core (набор JavaScript функций, позволяющий создавать динамические веб-страницы и рапространяемый по MIT-лицензии) и Ext JS (набор виджетов для создания пользовательских интерфейсов, распространяемый аналогично Ext JS 2.1 по условиям трёх лицензий).

jQuery

jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API по работе с Ajax. Т.к. jQuery является более простой для изучения остановимся на ней подробнее.

Возможности:

- переход по дереву DOM, включая поддержку Xpath как плагина;
- события;
- визуальные эффекты;
- AJAX-дополнения;
- JavaScript-плагины

Философия

Точно так же, как CSS отделяет визуализацию от структуры HTML, JQuery отделяет поведение от структуры HTML. Например, вместо прямого указания на обработчик события нажатия кнопки, управление передаётся JQuery, идентифицирующей кнопки и затем преобразовывающий его в обработчик события клика. Такое разделение поведения и структуры также называется принципом ненавязчивого JavaScript.

Библиотека jQuery содержит функционал, полезный для максимально широкого круга задач. Тем не менее, разработчиками библиотеки не ставилась задача совмещения в jQuery функций, которые подошли бы всюду, поскольку это привело бы к большому коду, бóльшая часть которого не востребована. Поэтому была реализована архитектура компактного универсального ядра библиотеки и плагинов. Это позволяет собрать для ресурса именно тот JavaScript-функционал, который на нём был бы востребован.

Для использования jQuery достаточно скачать с сайта http://jquery.com/ последнюю версию фремворка и подключить его в свой проект.

Листинг 8. Пример подключения jQuery как один внешний файл

```
<head>
<script type="text/javascript" src="путь/к/jQuery.js"></script>
</head>
```

Вся работа с jQuery ведётся с помощью функции $. Если на сайте применяются другие JavaScript библиотеки, где $ может использоваться для своих нужд, то можно использовать её синоним — jQuery. Второй способ считается более правильным, а чтобы код не получался слишком громоздким можно писать его следующим образом:

Листинг 9. Использование функции $

```
jQuery(function($) {
  // Тут код скрипта, где в $ будет jQuery
})
```

Работу с jQuery можно разделить на 2 типа:

- Получение jQuery-объекта с помощью функции $(). Например, передав в неё CSS-селектор, можно получить jQuery-объект всех элементов HTML попадающих под критерий и далее работать с ними с помощью различных методов jQuery-объекта.
- Вызов глобальных методов у объекта $, например, удобных итераторов по массиву.

Типичный пример манипуляции сразу несколькими узлами DOM заключается в вызове $ функции со строкой селектора CSS, что возвращает объект jQuery, содержащий некоторое количество элементов HTML-страницы. Эти элементы затем обрабатываются методами jQuery. Например,

Листинг 10.

```
$("div.test").add("p.quote").addClass("blue").slideDown("slow");
```

находит все элементы с классом test, а также все элементы <p> с классом quote, и затем добавляет им всем класс blue и визуально плавно спускает вниз. Методы, начинающиеся с $., удобно применять для обработки глобальных объектов. Например:

Листинг 11.

```
$.each([1,2,3], function() {
  document.write(this + 1);
});
```

добавит на страницу 234.

Для примера немного изменим предыдущие файлы и добавим в них использование фреймворка jQuery. Подключаем фреймворк в файл index.html:

Листинг 12. Подключение фрейморка jQuery

```
<script type="text/javascript" src="jquery.js"></script>
```

Добавим в нашу форму дополнительное поле, в котором будем изменять значение параметра учета количества сделанных изменений:

Листинг 13. Добавления дополнительного поля в форму

```
<form>
 <p>City: <input type="text" name="city" id="city" size="25"
onChange="getResult();" /></p>
 <p>ZipCode: <input type="text" name="zipCode" id="zipCode" size="5"
readonly /> получен с помощью AJAX запроса</p>
 <p>Значение было изменено <input type="text" name="jq" id="jq"
size="3" value="0" readonly /> раз с помощью jQuery</p>
</form>
```

Дополним функцию updatePage() вызовом дополнительной функции jq(), которая, используя механизмы jQuery, будет изменять данные в поле с id=”jq”, в котором указывается количество обновлений страницы:

Листинг 14. Исходный код функции jq()

```
function jq() {
 //Записываем в переменную X значение поля с id='jq'
   $x = $("#jq").attr("value");
 //Увеличиваем переменную на 1
   $x++;
 //Обновляем значение атрибута value поля с id='jq'
   $("#jq").attr("value", $x);
}
```

Про допонительные возможности jQuery вы можете узнать из документации и примеров использования на сайте разработчиков jquery.com.

Порядок выполнения работы

1. Ознакомится с теоретическими сведениями
2. Изучить работу скриптов, приведенных в теоретических сведениях.
3. Реализовать задание согласно варианта.
4. Составиь отчет о выполнении работы.

Варианты заданий

1. Проверка логина при регистрации – проверка на существование вводимого логина в базе данніх пользователей, при наборе логина в соответствующем поле.
2. Корзина товаров в онлайн магазине. Добавление/удаление.
3. Города и области – выборка области по выбранному городу.

4. Подсчет хеша при заполнении строки.
5. Транслитерация.
6. Простейший почтовій клиент – выборка темы письма по запросу и по клику на теме письма доставка его тела.
7. Мониторинг сервисов на ПК. Опрос списка запущенных процессов, и ассинхронное обновление.
8. Гостевая книга, Комментарии – добавление/удаление.
9. Чат.
10. Доработка заданий 3 и 4 лабораторной работы.

Используемая литература

1. Оригинал статьи http://www.ibm.com/developerworks/web/library/wa-ajaxintro1.html?S_TACT=105AGX99&S_CMP=CP
2. AJAX из викиучебника – http://ru.wikibooks.org/wiki/AJAX
3. Официальный сайт jQuery – jquery.com
4. Страница в wikipedia - http://ru.wikipedia.org/wiki/JQuery