Урок 2

Ajax и JSON

Знакомство с принципами асинхронного общения на базе JavaScript. Понятие AJAX. Знакомство с протоколом JSON.

Как вы уже могли увидеть в процессе обучения и в реальной жизни, полностью статической страницей сложно удержать пользователя. Она не только скучна, но и не даёт удобно работать с сайтом – все действия, требующие серверной обработки, производятся через перезагрузку страницы, возникает проблема сохранения данных между страницами. А если у пользователя ко всему прочему слабое сетевое соединение, то работа с сайтом замедляется вплоть до полной потери пользователя. Для интернет-магазинов и сайтов, которые стараются удерживать пользователя это критическая проблема.

На этом занятии мы научимся работать со страницей, обновляя её данные и производя общение с сервером без перезагрузки самой страницы. В этом нам поможет технология AJAX.

Что такое AJAX

Само сокращение **AJAX** – это аббревиатура, полученная от первых букв Asynchronous Javascript And Xml, что в переводе означает Асинхронный JS и XML. Говоря проще – это набор технологий и методов для обращения к серверу без перезагрузки страницы.

Если мы применяем AJAX, то мы обновляем при каждом запросе только часть страницы, конкретный блок. По сути мы превращаем наш сайт в полноценной оконное приложение, запускаемое через браузер. Важно помнить, что пользователю надо давать понять, что в данный момент происходит общение с сервером. Для этого существуют индикаторы загрузки, «крутилки», текстовые сообщения.

Все современные браузеры поддерживают AJAX. Проблемы могут быть только у совсем уж старых браузеров эпохи текстовых обозревателей. Однако, некоторые пользователи отключают JavaScript в браузере, поэтому нужно предусматривать такой расклад, давая альтернативные решения, либо оповещая пользователя о том, что с отключенным JS сайт не сможет предоставить все возможные технические решения. Благо, таких пользователей не так уж много.

Итак, AJAX хорош тем, что:

- даёт возможность построения удобного и дружественного интерфейса сайта;
- позволяет вести более активное общение с пользователем;
- позволяет обновлять страницу частями, вместо полной перезагрузки;
- удобен в использовании.

В самой аббревиатуре AJAX фигурирует акроним XML, намекая на то, что именно этот протокол разметки и общения должен применяться. Но использование XML совершенно необязательно. Зачастую применяются более простые протоколы.

Применение AJAX

Сама технология применяется для решения следующих задач:

- **Построение и обновление элементов вёрстки.** Самый популярный пример – это обновление блока корзины на сайтах интернет-магазинов при нажатии кнопки «Положить в корзину».
- **Загрузка дополнительных данных после завершения загрузки основной страницы.** Для ускорения отдачи основного контента пользователю часть данных оставляют на последующую загрузку. Ведь зачастую эти данные могут и не пригодиться. Например, вы выводите главные категории каталога на сайте, а при необходимости подкатегории загружаете по требованию пользователя.

- **«Живой» поиск.** Очень популярный AJAX-элемент систем поиска. Вы начинаете набирать начало поисковой фразы, а сайт предлагает список наиболее подходящих совпадений для выбора, что очень сильно экономит время работы с навигацией.

И этим списком перечень возможностей не ограничивается. AJAX позволяет вести практически любой обмен данными с сервером, накладывая ограничения лишь на формат общения. Как правило, на сайтах применяются следующие протоколы ответа:

- JSON (JavaScript Object Notation) – простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером, наиболее часто применяемый для обмена структурированными данными с сервером. Как понятно из названия, формат полностью совпадает с синтаксической записью объекта в JS.
- XML (eXtensible Markup Language) – некогда крайне популярный, а ныне просто структурированный формат общения с сервером. Очень похож на HTML.
- HTML/текст – тут всё просто, т.к. Вы можете загрузить с сервера HTML разметку или текст и сразу же вставить в страницу. Но это считается дурным тоном.
- Бинарные данные, файлы – используются намного реже, но позволяют через AJAX обмениваться, например, файлами.
- «Холостой» запрос – ответ от сервера на такой запрос не приходит, при этом некие данные на сервере всё же изменяются. Практика полезна для функционала, который не должен оповещать пользователя о своей работе. Например, трекинг поведения пользователя на сайте.

Обмен данными

В JavaScript общение с сервером осуществляется через объект XMLHttpRequest (проще говоря – XHR). Он представляет собой посредника между браузером и сервером, позволяя отправить запрос в понятном для сервера виде и получить ответ, который сможет обработать браузер.

Поскольку любой запрос (даже AJAX) – это всё тот же HTTP-запрос, то он подчиняется всем требованиям и правилам HTTP протокола. Это касается и методов обмена данными (GET, POST, PUT, DELETE и т.д.). Как правило, в AJAX используются два метода – GET и POST.

В GET-запросе происходит обращение к необходимому серверному ресурсу по URL. При этом данные, которые вы хотите отправить на сервер, передаются прямо в адресной строке.

```
http://mysite.com/item?id_item=5
```

При этом GET запрос накладывает ограничения на длину строки, поэтому большие пакеты данных передаются через POST. В POST запросе данные на сервер передаются в специальном пакете.

```
http://mysite.com/item/
```

```
Query string parameters:
param1: 1
param2: value
```

Как правило, для получения информации с сервера по неким фильтрующим параметрам нужно использовать GET-запрос, а для обновления или создания данных из пакета пользовательской информации – POST-запрос.

Клиентская часть кода должна обеспечивать не только передачу и приём данных, но и безопасность отправляемых данных. Однако любые данные от клиента легко подменить, поэтому серверная часть приложения перед запуском данных в бизнес-процессы обязана валидировать их на корректность и безопасность (чуть позже мы поговорим и об этом).

В процессе работы серверной части приложения создаются новые данные, которые нужно отправить обратно клиенту в одном из указанных выше форматов.

AJAX использует асинхронное общение с сервером (по умолчанию). Таким образом, пользователь может не останавливать свою работу с сайтом, пока идёт фоновый обмен информацией. Разумеется, как мы уже обозначили выше, забывать про оповещение пользователя о текущем обмене информацией не стоит.

Объект XMLHttpRequest

Объект XMLHttpRequest (или, сокращенно, XHR) дает возможность браузеру делать HTTP-запросы к серверу без перезагрузки страницы.

В зависимости от используемого пользователем браузера процедура создания объекта XHR меняется. Как часто бывает, отличилось семейство браузеров Internet Explorer. Поэтому для создания XHR мы будем использовать следующую конструкцию

```
var xhr;
// если используется Gecko (Chrome, Mozilla, Opera, Safari)
if (window.XMLHttpRequest) {
    xhr = new XMLHttpRequest();
}
// Internet Explorer
else if (window.ActiveXObject) {
    xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Объект window сочетает в себе и глобальный объект JavaScript, и текущее окно браузера. Именно он подсказывает нам, с каким браузером мы имеем дело.

На этом тонкости не заканчиваются. Определённые версии Mozilla Firefox не могут работать с ответом от сервера в случае, если тот не содержит заголовка XML mime-type. Хоть это и проблема серверного программирования, но знать о ней стоит. На стороне клиента её можно решить, переопределив этот параметр:

```
xhr.overrideMimeType('text/xml');
```

Вполне вероятно, после получения ответа от сервера Вы захотите как-то этот ответ обработать и разместить его данные на странице. Для этого назначается callback-функция. В параметре XHR объекта указывается только имя функции. Если поставить после него скобки, то onreadystatechange будет ожидать получения результата этой функции, чтобы затем использовать его как callback. Также возможно объявить анонимную функцию, определив её прямо на месте.

```
xhr.onreadystatechange = myCallBack;
```

```
xhr.onreadystatechange = function(){ alert(1); }
```

Создав объект и определив его начальные действия, можно приступать к отправке самого запроса на сервер. В этом нам помогут методы open и send.

```
xhr.open('GET', 'http://mysite.com/ajax.file', true);
```

Первый параметр – это HTTP-метод, при помощи которого будет отправлен запрос. Второй параметр задаёт требуемый ресурс на сервере, к которому и будет направлен запрос. По правилам безопасности запросы к доменам, отличающимся от того, со страницы которого направлен запрос (кросс-доменные запросы) запрещены. Третий параметр указывает на асинхронность запроса. Если он равен TRUE, то все скрипты на странице продолжат свою работу в то время, как будет происходить общение с сервером.

Синхронные запросы применяются весьма редко, ведь они по сути блокируют работу со страницей вплоть до окончания общения в рамках запроса. Пользователь не сможет даже прокрутить страницу. В случае, когда синхронный запрос выполняется очень долго, сам браузер предлагает закрыть проблемную страницу.

Однако для некоторых операций синхронные запросы важны. Например, если вы производите авторизацию пользователя через AJAX. Пользователь не может продолжать работу в анонимном состоянии, так что это принудительная мера.

Продолжительность работы асинхронного запроса в JS можно задать в свойстве timeout:

```
xhr.timeout = 15000; // задаётся в миллисекундах
```

Если всё же будет превышено время ожидания, выполнение запроса будет остановлено. При наличии обработчика, будет выполнено событие ontimeout:

```
xhr.ontimeout = function(){
  alert('Слишком долгий запрос, выполнение остановлено!');
};
```

Далее нужно использовать метод send:

```
xhr.send(null);
```

В качестве параметра принимаются произвольные данные, приведённые к виду строки запроса, т.е.:

```
param1=value&param2=42
```

При отправке пакета данных при помощи метода POST требуется переопределить MIME-тип запроса следующим образом:

```
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

После выполнения send сервер должен будет прислать ответ на клиентскую часть. Чуть выше мы задачи функцию обработчик myCallBack. Именно она вызовется, как только сервер отдаст некий ответ.

В первую очередь, стоит проверить, в каком статусе находится запрос:

```
xhr.readyState
```

Данный параметр может принимать 5 значений:

- 0 - запрос не инициализирован;
- 1 - загрузка;
- 2 – запрос принят;
- 3 – обмен данными;
- 4 – запрос выполнен.

```
if (xhr.readyState == 4)
    // ответ на запрос получен
else
    // процесс в работе
```

Далее нужно проверить HTTP-статус ответа. Это цифровой код, который говорит нам о том, как именно завершился процесс. Нас интересует ответ 200 OK. Ответы 3xx говорят о том, что произошёл редирект, 4xx – проблемы с доступом к ресурсу, 5xx – серверная ошибка.

```
if (xhr.status == 200)
    // серверная часть отработала корректно
```

После того, как мы убедимся, что ответ получен, и он корректен, можно приступать к его обработке. Получить тело ответа можно двумя методами:

- responseText – вернёт ответ в виде строки;
- responseXML – вернёт ответ в виде XMLDocument.

Форматы JSON и XML

Как мы проговорили выше, чаще в AJAX используется формат JSON, т.к. он легче и проще. JSON легко читается как людьми, так и компьютером.

Хоть в имени формата и фигурирует JavaScript, сам формат является совершенно независимым от языка JS и очень часто используется во многих других языках для решения различных задач.

JSON по своему применению является одним из максимально удобных форматов для взаимодействия с JavaScript. Ведь он совпадает с синтаксической записью объекта, т.е. он практически сразу же может быть сконвертирован в JS-объект.

Структура у JSON может быть представлена двумя способами:

- Пары ключ-значение, где в качестве ключа выступает регистрозависимая строка, а значением может быть любая форма.
- Упорядоченные значения. Очень похоже на массив или список.

Поскольку JSON стандартизирован, он поддерживается практически любым современным языком программирования, что удобно при замене серверной части – не приходится менять протоколы общения.

Конкретные значения в JSON могут быть следующими:

- объект;
- одномерный массив;
- число;
- true, false и null;
- строка

JavaScript по умолчанию предоставляет набор методов для работы с JSON:

- JSON.parse – чтение объектов из JSON-строки.
- JSON.stringify – конвертация объектов в JSON-строку.

```
var colors = '["red", "green", "blue"]';
colors = JSON.parse(colors);
alert(colors [1] );
```

Метод JSON.parse может разбирать и более сложные структуры. Если указать второй параметр, в качестве которого задаётся функция-обработчик function(key, value), то при чтении JSON-строки в обработчик поочерёдно передаются все создаваемые пары ключ-значение, а на выход может быть подано уже приведённое к нужному типу значение или undefined.

Разберём пример:

```
var bmw = '{"title":"Дата производства","date":"2016-10-12T15:19:37.000Z"}';
```

Если мы захотим обрабатывать значение date как объект Date, то без указания функции обработчика нам всегда будет возвращена ошибка :

```
var bmw_json = JSON.parse(bmw);
alert(bmw_json.date.getDate() ); // так сделать не получится!
```

В этом примере мы вполне можем создать правило, которое указывает, что ключ date - это дата:

```
var bmw_json = JSON.parse(bmw, function(key, value) {
  if (key == 'date') return new Date(value);
  return value;
});
alert(bmw_json.date.getDate() );
```

Обратное действие – это JSON.stringify, который выполняет т.н. сериализацию объектов в строку.

```
var cat = {
   name: "Василий",
   age: 3
};
var str = JSON.stringify(cat);
alert(str);
```

Если же мы работаем с форматом XML, то нам нужно обрабатывать ответ при помощи специальных методов. К примеру, если ответ имеет форму:

```
<?xml version="1.0" ?>
<answer>
    Some anser data
</answer>
```

То обработка ответа будет выглядеть примерно следующим образом:

```
var xml_answer = xhr.responseXML;
var text = xml_answer.getElementsByTagName('answer').item(0);
```

```
alert(text.firstChild.data);
```

Этот код обрабатывает объект XMLDocument, который возвращается в параметре responseXML и применяет методы DOM-доступа к данным, которые лежат в документе XML.

Теперь давайте сравним две одинаковые структуры в JSON и XML:

```
{
  "vendor": "Audi",
  "model": "RS 7",
  "engine": {
     "power": 560,
     "volume": 4
  },
  "available_colors": [
     "red",
     "white",
     "black"
  ]
}
```

```
<?xml version="1.0" ?>
<answer>
   <vendor>Audi</vendor>
   <model>RS 7</model>
   <engine>
      <power>560</power>
      <volume>4</volume>
   </engine>
   <available_colors>
      <color>red</color>
      <color>white</color>
      <color>black</color>
   </available_colors>
</answer>
```

Как видите, один и тот же объект можно записать по-разному – каждый выбирает для себя наиболее удобный способ записи.

Практика

На прошлом занятии мы реализовали меню. Давайте улучшим его таким образом, чтобы оно использовало AJAX. Применять технология будет следующим образом:

1. Формируется принимающая часть.
2. На сервере создан статический файл с содержимым меню в JSON-формате.
3. При загрузке страницы на сервер отправляется запрос содержимого меню.
4. Ответ обрабатывается и встраивается в меню.

Домашнее задание

1. Улучшить меню таким образом, чтобы оно могло иметь многоуровневую структуру.
2. Создать меню, соответствующее меню интернет-магазина (личный кабинет, каталог, промоакции и т.д.).
3. Создать функционал фотогалереи: имеется статичный json-набор миниатюр, на основании которого строится сетка изображений со ссылками на полноразмерные картинки.
4. * Создать два статических ответа {result : “success”} и {result: “error”}. В зависимости от каждого из них навесить на определенный ajax-запрос обработчик результата.

Дополнительные материалы

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Mozilla Developer Network
2. MSDN
3. Safari Developer Library
4. Современный учебник JavaScript