Что такое AJAX я думаю рассказывать не стоит, ибо с приходом веб-два-нуля большинство пользователей уже воротят носом от перезагрузок страниц целиком, а с появлением jQuery реализация упростилась в разы…

Начнем с самого простого – загрузка HTML кода в необходимый нам DOM элемент на странице. Для этой цели нам подойдет метод "load()". Данный метод может принимать следующие параметры:

url – запрашиваемой страницы

data – передаваемые данные (необязательный параметр)

callback – функция которая будет вызвана при завершении запроса к серверу (необязательный параметр)

Теперь на примерах:

// в элемент с id=content будет вставлен весь HTML с указанной страницы

$("#content").load("/get-my-page.html");

// в элемент с id=content будет вставлен HTML с указанной страницы

// выбранный по указанному селектору #wrapper

$("#content").load("/get-my-page.html #wrapper");

// передаем данные на сервер

$("#content").load("/get-my-page.html", {id:18});

// обрабатываем полученные данные

$("#content").load("/get-my-page.html", function(){

alert("Ничего оригинальней не придумал");

});

Из моего опыта работы – вам очень часто придётся пользоваться методом "load()" как описано в первом примере, а еще советую запомнить второй пример, он может выручить, когда надо реализовать загрузку AJAX’ом, а доступа к сервер-сайду у вас нет или он ограничен.

Живой пример:

<!DOCTYPE html>

<html dir="ltr" lang="en-US">

<head>

<meta charset="UTF-8"/>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Пример загрузки HTML посредством AJAX и метода load()</title>

<link rel="profile" href="http://gmpg.org/xfn/11"/>

<link rel="shortcut icon" href="http://anton.shevchuk.name/favicon.ico"/>

<link rel="stylesheet" href="css/styles.css"/>

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

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

<style>

article {

border:1px dotted #ccc

}

</style>

</head>

<body>

<div id="content" class="wrapper box">

<menu>

<a href="index.html" title="go prev" class="button alignleft" rel="prev">← Back</a>

<a href="index.html" title="back to Index" class="button alignleft" rel="index">Index §</a>

<a href="#" title="reload" class="button alignleft" onclick="window.location.reload();return false">Reload ¤</a>

<a href="ajax.datatype.html" title="go next" class="button alignright" rel="next">Next →</a>

<hr/>

<pre><code contenteditable="true"><em>// AJAX is easy</em>

$(<span>'article'</span>).load(<span>'index.html'</span>)</code></pre>

<button type="button" class="code">Run Code</button>

<pre><code contenteditable="true"><em>// easy to load</em>

$(<span>'article'</span>).load(<span>'index.html header'</span>)</code></pre>

<button type="button" class="code">Run Code</button>

<pre><code contenteditable="true"><em>// easy to handle</em>

$(<span>'article'</span>).load(<span>'index.html article'</span>, function(){

$(<span>'article'</span>).show().fadeOut(2000, function(){

$(this).html('<strong>→</strong>').fadeIn();

});

})</code></pre>

<button type="button" class="code">Run Code</button>

</menu>

<header>

<h1>load()</h1>

<h2>Простой пример использования метода load()</h2>

</header>

<article>

<h2>Пробуем загрузить контент →</h2>

</article>

<footer>

©copyright 2014 Anton Shevchuk — <a href="http://anton.shevchuk.name/jquery-book/">jQuery Book</a>

</footer>

<script type="text/javascript">

var \_gaq = \_gaq || [];

\_gaq.push(['\_setAccount', 'UA-1669896-2']);

\_gaq.push(['\_trackPageview']);

(function() {

var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;

ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';

var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);

})();

</script>

</div>

</body>

</html>

Следующий метод с которым я вас познакомлю будет "ajax()" – собственно, он тут главный, и все остальные AJAX методы являются лишь обёрткой (и "load()" в том же числе). Метод "ajax()" принимает в качестве параметра пачку настроек и URL куда стучаться, приведу пример аналогичный вызову "load()":

$.ajax({

url: "/get-my-page.html", // указываем URL и

dataType: "html", // тип загружаемых данных

success: function (data) {

// вешаем свой обработчик события success

$("#content").html(data)

}

});

Тут мы обрабатывали HTML ответ от сервера – это хорошо когда нам полстраницы обновить надо, но данные лучше передавать в "правильном" формате – это XML – понятно, структурировано, и избыточно, и как-то не совсем JavaScript-way, и поэтому наш выбор – это [JSON](https://ru.wikipedia.org/wiki/JSON) (wikipedia в помощь):

{

"note": {

"time":"2012.09.21 13:11:15",

"text":"Рассказать про JSONP"

}

}

Фактически это и есть JavaScript код как есть (JavaScript Object Notation если быть придирчиво точным), при этом формат уже распространён настолько, что работа с данными в другом формате уже не комильфо.

Жизнь не стоит на месте, есть и более удобные форматы, но не в JavaScript’е :)

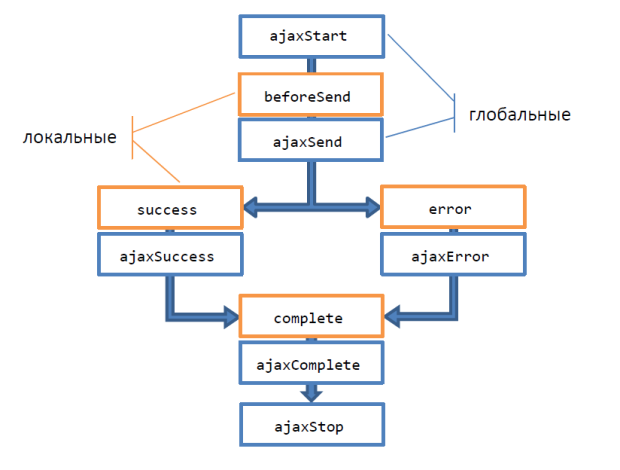
Для загрузки JSON существует быстрая функция-синоним – "jQuery.getJSON(url [,data] [,success(data, textStatus, jqXHR)])" – в качестве обязательного параметра лишь ссылка, куда стучимся, опционально можно указать данные, для передачи на сервер и функцию обратного вызова

Нельзя просто так взять и описать все возможные параметры для вызова "ajax()", таки стоит держать официальный мануал под рукой – <http://api.jquery.com/jQuery.ajax/>

Я неспроста оставляю так много места под ваши заметки:..

**Обработчики AJAX событий**

Для удобства разработки, AJAX запросы бросают несколько событий, и их естественно можно и нужно обрабатывать. jQuery позволяет обрабатывать эти события для каждого AJAX запроса в отдельности, либо глобально. Приведу схемку на которой наглядно видно порядок возникновения событий в jQuery:



[**увеличить изображение**](http://intuit.valrkl.ru/course-768/8_1.png)

Вот и полный список событий с небольшими ремарками:

ajaxStart — данное событие возникает в случае когда побежал первый AJAX запрос, и при этом других активных AJAX запросов в данный момент нет

beforeSend — возникает до отправки запроса, позволяет редактировать XMLHttpRequest, локальное событие

ajaxSend — возникает до отправки запроса, аналогично "beforeSend"

success — возникает по возвращению ответа, когда нет ошибок ни сервера, ни вернувшихся данных, локальное событие

ajaxSuccess — возникает по возвращению ответа, аналогично success

error — возникает в случае ошибки, локальное событие

ajaxError — возникает в случае ошибки

complete — возникает по завершению текущего AJAX запроса (с ошибкой или без — срабатывает всегда), локальное событие

ajaxComplete — глобальное событие, аналогичное complete

ajaxStop — данное событие возникает в случае, когда больше нету активных запросов

Пример для отображения элемента с "id="loading"" во время выполнения любого AJAX запроса (т.е. мы обрабатываем глобальное событие):

$("#loading").bind("ajaxSend", function(){

$(this).show(); // показываем элемент

}).bind("ajaxComplete", function(){

$(this).hide(); // скрываем элемент

});

Это задачка по юзабилити – мы всегда должны держать пользователя сайта в курсе дела о происходящем на странице, и отправка AJAX запроса тоже попадает под разряд "must know". Подобное решение у вас будет практически на любом сайте, где ходит AJAX

Для локальных событий – вносим изменения в опции метода "ajax()":

$.ajax({

beforeSend: function(){

// данный обработчик будет вызван

// перед отправкой данного AJAX запроса

},

success: function(){

// а этот при удачном завершении

},

error: function(){

// этот при возникновении ошибки

},

complete: function(){

// и по завершению запроса (удачном или нет)

}

});

Можно глобальные обработчики отключить принудительно используя флаг "global", для этого выставляем его в "false", и пишем функционал в обход событий "ajaxStart" и "ajaxStop":

$.ajax({

global: false,

beforeSend: function(){

// ...

},

success: function(){

// ...

},

error: function(){

// ...

},

complete: function(){

// ...

}

});

Данная опция частенько помогает избежать конфликтов при работе с AJAX запросами, но не стоит ей злоупотреблять.

**JSONP**

JSONP – это наш старый знакомый JSON с прослойкой в виде callback функции О\_о. Да ладно, давайте на примерах, вот как у нас выглядит ответ сервера в формате JSON:

{

"note": {

"time":"2012.09.21 13:12:42",

"text":"Рассказать зачем нужен JSONP"

}

}

Хорошо, когда у нас эти данные приходят с нашего сервера – обработаем, и всё будет чики-пики, но а если нам потребуется заполучить данные с другого сервера, то политика безопасности в браузерах не позволит отправить XMLHTTPRequest на другой сервер, и надо уже будет что-то придумывать. Можно чуть-чуть напрячься и вспомнить, что подключать JavaScript с другого сервера то мы можем, и он будет выполнен. Вот она – зацепка-то, а если подключаемый скрипт будет содержать вызов нашей функции с подготовленными данными – то это уже что-то:

alertMe({

"note": {

"time":"2012.09.21 13:13:13",

"text":"Каков же профит от использования JSONP?"

}

})

Таким образом, описав в своём коде функцию "alertMe()" мы сможем обработать данные с удаленного сервера. Зачастую, сервера ловят параметр "callback" или "jsonp", и используют его как имя функции обёртки:

<script type="text/javascript" src="http://domain.com/getUsers/?callback=alertMe">

</script>

Ну это было предыстория, теперь вернёмся к jQuery и методу "ajax()":

$.ajax({

url: "http://domain.com/getUsers/?callback=?", // указываем URL dataType: "jsonp",

success: function (data) {

// обрабатываем данные

}

});

В запрашиваемом URL наблюдательный читатель заметит незаконченную структуру "callback=?", так вот вместо "?" будет подставлено имя ново сгенерированной функции, внутри которой будет осуществляться вызов функции "success()". Вместо этой прокси-функции можно использовать и свою функцию, достаточно указать её имя в качестве параметра "jsonpCallback" при вызове "ajax()". Оба этих подхода есть в примере:

<!DOCTYPE html>

<html dir="ltr" lang="en-US">

<head>

<meta charset="UTF-8"/>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Пример использования JSONP</title>

<link rel="profile" href="http://gmpg.org/xfn/11"/>

<link rel="shortcut icon" href="http://anton.shevchuk.name/favicon.ico"/>

<link rel="stylesheet" href="css/styles.css"/>

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

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

<script>

function toJsonContainer(data, status, xhr) {

$('#json').text(data);

}

function buildImages(data) {

$.each(data.items, function(i, item){

$("<img/>").attr("src", item.media.m).appendTo("#images");

if ( i == 3 ) return false;

});

}

</script>

<style>

article pre {

font-size: 14px;

padding: 20px;

}

</style>

</head>

<body>

<div id="content" class="wrapper box">

<menu>

<a href="ajax.script.html" title="go prev" class="button alignleft" rel="prev">← Prev</a>

<a href="index.html" title="back to Index" class="button alignleft" rel="index">Index §</a>

<a href="#" title="reload" class="button alignleft" onclick="window.location.reload();return false">Reload ¤</a>

<a href="ajax.google.html" title="go next" class="button alignright" rel="next">Next →</a>

<hr/>

<pre><code>$.ajax({

url: <span>"http://anton.shevchuk.name/book/code/ajax/example.php?callback=?"</span>,

dataType: <span>"jsonp"</span>,

success: toJsonContainer

});</code></pre>

<button type="button" class="code">Run Code</button>

<pre><code>$.ajax({

url: <span>"http://anton.shevchuk.name/book/code/ajax/example.php?callback=?"</span>,

dataType: <span>"jsonp"</span>,

jsonpCallback: <span>"toJsonContainer"</span>

});</code></pre>

<button type="button" class="code">Run Code</button>

<h3>Flickr API</h3>

<pre><code contenteditable="true">$.getJSON(

<span>"http://api.flickr.com/services/feeds/photos\_public.gne?jsoncallback=?"</span>,

{

tags: <span>"orange"</span>,

tagmode: <span>"any"</span>,

format: <span>"json"</span>

},

buildImages

);</code></pre>

<button type="button" class="code">Run Code</button>

</menu>

<header>

<h1>Пример использования JSONP с простым сервером</h1>

<h2>Код сервера <a href="http://anton.shevchuk.name/book/code/ajax/example.code">code/ajax/example.code</a></h2>

</header>

<article>

<pre id="json">

</pre>

</article>

<article>

<div id="images">

</div>

</article>

<footer>

©copyright 2014 Anton Shevchuk — <a href="http://anton.shevchuk.name/jquery-book/">jQuery Book</a>

</footer>

<script type="text/javascript">

var \_gaq = \_gaq || [];

\_gaq.push(['\_setAccount', 'UA-1669896-2']);

\_gaq.push(['\_trackPageview']);

(function() {

var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;

ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';

var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);

})();

</script>

</div>

</body>

</html>

А еще стоит упомянуть, что можно указать как обзывается callback- параметр используя параметр "jsonp", таким образом указав "jsonp:"my"" в URL будет добавлена структура "my=?"

На данный момент достаточно много сервисов предоставляют API с поддержкой JSONP:

* Google – [поиск](https://developers.google.com/custom-search/v1/overview) и большинство сервисов
* Yahoo – [поиск](http://developer.yahoo.com/boss/search/) и большинство сервисов
* [Flickr](http://www.flickr.com/services/api/) – работа с поиском данного сервиса
* [MediaWiki](https://en.wikipedia.org/w/api.php) – соответственно и производные – Wikipedia, Wiktionary
* [CNET](http://developer.cnet.com/) – поиск по новостному порталу

Использование подобного подхода позволяет обходить ограничения накладываемые сервисами на количество запросов с одного IP, плюс не грузит сервер дополнительной работой по проксированию запросов от пользователя к сервисам.

**Лечим JavaScript зависимость**

Любовь к AJAX бывает чрезмерной, и в погоне к Web2.0 (3.0, 4.0, … — желаемое подчерк-нуть) мы создаём сайты в которых все наши действия бегут через [XMLHTTPRequest](https://ru.wikipedia.org/wiki/XMLHttpRequest" \t "_blank). Нет, это конечно не плохо — снижаем нагрузку на сервер, канал и т.д. и т.п., но есть одно "но" — у нас есть поисковые машины, которые не озадачивают себя выполнением JavaScript кода, а контент, спрятанный за AJAX запросом, им отдать всё таки нужно. Следовательно, у нас возникает необходимость дублирования навигации (это как минимум) для клиентов без JavaScript.

Стоит помнить, что есть ещё пользователи, у которых отключён JavaScript в браузере (или даже не поддерживается, привет тебе, рысь), но эти знают, что делают. А есть ещё скрипты, которые ломаются, и не дают обычным пользователям воспользоваться навигацией по сайту, а пользователей это очень сильно расстраивает, так что эта глава не "просто так".

Как же всё это обойти и на грабли не наступить? Да всё очень просто — создавайте обычную навигацию, которую вы бы делали не слышав ни разу о AJAX и компании:

<ul class="navigation">

<li><a href="/">Home</a></li>

<li><a href="/about.html">About Us</a>

</li> <li><a href="/contact.html">Contact Us</a></li>

</ul>

<section id="content"><!-- Content --></section>

Данный пример работает у нас без JavaScript’a, все страницы в нашем меню используют один и тот же шаблон для вывода информации, и по факту у нас изменяется лишь содержимое <div> с "id="content"". Теперь приступим к загрузке контента посредством AJAX – для этого добавим следующий код:

$(function() {

// вешаем обработчик на все ссылки в нашем меню navigation

$("ul.navigation a").click(function(){

var url = $(this).attr("href"); // возьмем ссылку

url =+ "?ajax=true"; // добавим к ней параметр ajax=true

$("#content").load(url); // загружаем обновлённое содержимое

return false; // возвращаем false

// - дабы не сработал переход по ссылке

});

});

В данном примере мы предполагаем, что сервер, видя параметр "ajax=true" вернет нам не полностью всю страницу, а лишь обновление для искомого элемента <div id="content">.

Конечно, сервер должен быть умнее и не требовать явного указания для использования AJAX’а, а должен вполне удовлетвориться, словив header "X\_REQUESTED\_WITH" со значением "XMLHttpRequest". Большинство современных фреймворков для web-разработки с этим справляются "из коробки".

Если же управлять поведением сервера проблематично, и он упёрто отправляет нам всю страницу целиком, то можно написать следующий код:

$(function() {

// вешаем обработчик на все ссылки в нашем меню navigation

$("ul.navigation a").click(function(){

var url = $(this).attr("href"); // возьмем ссылку

// загружаем страницу целиком, но в наш контейнер вставляем

// лишь содержимое #content загружаемой страницы

$("#content").load(url + " #content > \*");

return false; // возвращаем false

});

});

Если в подгружаемом содержимом так же есть ссылки – то вы уже должны знать как "оживить" события.

**Прокачиваем AJAX**

У нас есть три способа для "прокачки" AJAX'а в jQuery: это создание префильтров, добавление новых конверторов и транспортов.

**Префильтры**

Префильтр – это функция, которая будет вызвана до "ajaxStart", в ней вы сможете изменить как объект "jqXHR", так и любые сопутствующие настройки:

// регистрация AJAX префильтра

$.ajaxPrefilter(function( options, originalOptions, jqXHR ) {

// наши манипуляции над настройками и jqXHR

});

Для чего всё это? Да вот простая задачка – не ждать "старый" AJAX ответ, если мы запрашиваем URL заново:

// коллекция текущих запросов

var currentRequests = {};

$.ajaxPrefilter(function( options, originalOptions, jqXHR ) {

// наша произвольная настройка

if ( options.abortOnRetry ) {

if ( currentRequests[ options.url ] ) {

// отменяем старый запрос

currentRequests[ options.url ].abort();

}

currentRequests[ options.url ] = jqXHR;

}

});

// вызов с использованием фильтра

$.ajax({

/\* ... \*/

abortOnRetry: true

})

Ещё можно изменить опции вызова, вот пример который по флагу "crossDomain" пересылает запрос на заранее подготовленную проксирующую страницу на нашем сервере:

$.ajaxPrefilter(function( options ) {

if ( options.crossDomain ) {

options.url = "/proxy/" + encodeURIComponent( options.url );

options.crossDomain = false;

}

});

Префильтры можно "вешать" на определенный тип "dataType" (т.е. в зависимости от ожидаемого типа данных от сервера будут срабатывать различные фильтры):

$.ajaxPrefilter("json script", function(options, original, jqXHR) {

/\* ... \*/

});

Ну и последнее, для переключение "dataType" на какой-нить другой нам достаточно будет вернуть необходимое значение:

$.ajaxPrefilter(function( options ) {

// это наша функция-детектор необходимых URL

if ( isActuallyScript( options.url ) ) {

// теперь "ждём" script

return "script";

}

});

Будьте очень осторожны когда оперируете глобальными настройками, да ещё через такую неявную фичу как фильтры – задокументируйте подобные подходы в сопроводительной документации, иначе разработчики которые будут в дальнейшем сопровождать ваш код будут сильно ругаться (в качестве оных можете оказаться и вы сами, ну через пару месяцев)

**Конверторы**

Конвертор – функция обратного вызова, которая вызывается в том случае, когда полученный типа данных не совпадает с ожидаемым (т.е. "dataType" указан неверно). Всё конверторы хранятся в глобальных настройках "ajaxSettings":

// формат ключа "из\_формата в\_формат"

// в качестве входного формата можно использовать "\*"

converters: {

"\* text": window.String, // что угодно приводим к тексту

"text html": true, // текст к html (флаг true == без изменений)

"text json": jQuery.parseJSON, // текст к JSON

"text xml": jQuery.parseXML // разбираем текст как xml

}

Для расширения набора конверторов потребуется функция "$.ajaxSetup()":

$.ajaxSetup({

converters: {

"text mydatatype": function( textValue ) {

if ( valid( textValue ) ) {

// разбор пришедших данных

return mydatatypeValue;

} else {

// возникла ошибка

throw exceptionObject;

}

}

}

});

Имена "dataType" должны всегда быть в нижнем регистре

Конверторы следует использовать, если требуется внедрить произвольные форматы "dataType", или для конвертации данных в нужный формат. Необходимый "dataType" указываем при вызове метода "$.ajax()":

$.ajax( url, { dataType: "mydatatype" });

Конверторы можно задавать так же непосредственно при вызове "$.ajax()", дабы не засорять общие настройки:

Конверторы можно задавать так же непосредственно при вызове "$.ajax()",

дабы не засорять общие настройки:

$.ajax( url, {

dataType: "xml text mydatatype",

converters: {

"xml text": function( xmlValue ) {

// получаем необходимые данные из XML

return textValue;

}

}

});

Чуть-чуть пояснений – мы запрашиваем "XML", который конвертируем в текст, который будет передан в наш конвертор из "text" в "mydatatype"

**Транспорт**

Использование своего транспорта – это крайняя мера, прибегайте к ней, только в том случае если с поставленной задачей нельзя справиться с использованием префильтров и конверторов

Транспорт – это объект, который предоставляет два метода – "send()" и "abort()" – они будут использоваться внутри метода "$.ajax()". Для регистрации своего метода транспортировки следует использовать метод "$.ajaxTransport()", будет это выглядеть как-то так:

$.ajaxTransport( function( options, originalOptions, jqXHR ) {

if ( /\* transportCanHandleRequest \*/ ) {

return {

send: function( headers, completeCallback ) {

/\* отправляем запрос \*/

},

abort: function() {

/\* отменяем запрос \*/

}

};

}

});

Проясню чуток параметры с которыми будем работать:

* options – настройки запроса (то что указываем при вызове "$.ajax()")
* originalOptions – "чистые" настройки, даже без учёта изменений "по умолчанию"
* jqXHR – объект "jQuery XMLHttpRequest"
* headers – заголовки запроса в виде связки ключ-значение
* completeCallback – функция обратного вызова, её следует использовать для оповещения о завершении запроса

Функция "completeCallback" имеет следующую сигнатуру:

function ( status, statusText, responses, headers ) {

/\* какой-то код \*/

}

где:

* status – HTTP статус ответа.
* statusText – текстовая интерпретация ответа
* responses (опционально) – это объект содержащий ответы сервера во всех форматах, которые поддерживает транспорт, для примера: родной "XMLHttpRequest" будет выглядеть как "{ xml: XMLData, text: textData }" при запросе XML документа
* headers (опционально) – строка содержащие заголовки ответа сервера, ну если конечно транспорт может их получить (вот например метод "XMLHttpRequest.getAllResponseHeaders()" может).

Как и префильтры, транспорт можно привязывать к определенному типу запрашиваемых данных:

$.ajaxTransport( "script", function( options, originalOptions, jqXHR ) {

/\* привязываемся лишь к script\*/

});

А теперь мега-напряг – пример транспорта "image":

$.ajaxTransport( "image", function( options ) {

if (options.type === "GET" && options.async ) {

var image;

return {

send: function( \_ , callback ) {

image = new Image();

function done( status ) { // подготовим

if ( image ) {

var statusText =

( status == 200 ) ? "success" : "error",

tmp = image;

image = image.onreadystatechange = image.onerror

= image.onload = null;

callback( status, statusText, { image: tmp } );

}

}

image.onreadystatechange = image.onload = function() {

done( 200 );

};

image.onerror = function() {

done( 404 );

};

image.src = options.url;

},

abort: function() {

if ( image ) {

image = image.onreadystatechange = image.onerror

= image.onload = null;

}

}

}; // /return

} // /if

}); // /ajaxTransport

Рабочий пример вы увидите ниже, но я хотел бы ещё раз напомнить, что это "advanced level", и данный раздел лишний в учебнике "для начинающих".

<!DOCTYPE html>

<html dir="ltr" lang="en-US">

<head>

<meta charset="UTF-8"/>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Пример создания транспорта для AJAX</title>

<link rel="profile" href="http://gmpg.org/xfn/11"/>

<link rel="shortcut icon" href="http://anton.shevchuk.name/favicon.ico"/>

<link rel="stylesheet" href="css/styles.css"/>

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

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

<script>

$.ajaxTransport("image", function (options) {

if (options.type === "GET" && options.async) {

var image;

return {

send:function (\_, callback) {

image = new Image();

// подготовим функцию done

function done(status) {

if (image) {

var statusText = ( status == 200 ) ? "success" : "error", tmp = image;

image = image.onreadystatechange = image.onerror = image.onload = null;

callback(status, statusText, { image:tmp });

}

}

image.onreadystatechange = image.onload = function () {

done(200);

};

image.onerror = function () {

done(404);

};

image.src = options.url;

},

abort:function () {

if (image) {

image = image.onreadystatechange = image.onerror = image.onload = null;

}

}

};

}

});

</script>

</head>

<body>

<div id="content" class="wrapper box">

<menu>

<a href="index.html" title="go prev" class="button alignleft" rel="prev">← Back</a>

<a href="#" title="reload" class="button alignleft" onclick="window.location.reload();return false">Reload ¤</a>

<hr/>

<pre><code contenteditable="true">$.ajax(<span>'images/events.png'</span>, {

dataType:<span>'image'</span>,

success: function(data) {

$(<span>'article'</span>).html(data);

}

});</code></pre>

<button type="button" class="code">Run Code</button>

</menu>

<header>

<h1>Своя реализация AJAX-транспорта</h1>

<h2>Пример надуманный</h2>

</header>

<article>

<p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus rutrum,

lectus eu varius consectetur, libero velit hendrerit augue, ut posuere enim neque

in libero. Donec eget sagittis nibh. Suspendisse sed tincidunt urna. Cras quis

euismod neque. Maecenas auctor ultricies posuere. Pellentesque luctus pulvinar dui

eget semper. Donec sodales odio eu sapien varius luctus. Donec dictum feugiat diam

at malesuada. Sed nec massa in augue condimentum faucibus quis ut diam. Quisque

nisl sem, semper nec vulputate vel, mattis sit amet justo. Aliquam purus felis,

tempor at scelerisque quis, tincidunt in neque. Etiam ut risus diam. Pellentesque

fermentum risus id elit feugiat cursus. Ut fringilla dictum diam, sed iaculis

lorem pulvinar ut. Cras vel elit id velit commodo viverra sit amet vel orci.

</p>

</article>

<footer>

©copyright 2014 Anton Shevchuk — <a href="http://anton.shevchuk.name/jquery-book/">jQuery Book</a>

</footer>

<script type="text/javascript">

var \_gaq = \_gaq || [];

\_gaq.push(['\_setAccount', 'UA-1669896-2']);

\_gaq.push(['\_trackPageview']);

(function() {

var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;

ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';

var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);

})();

</script>

</div>

</body>

</html>