Бесплатная электронная книга

# учусь ajax

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#ajax

# Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ajax

It is an unofficial and free ajax ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ajax.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# глава 1: Начало работы с ajax

## замечания

AJAX (синхронный **J** avaScript нный **X** ML) позволяет запрашивать внешние данные , **не** блокируя выполнение кода. Во многих случаях это реализуется при запросе частей страницы или информации с сервера (через XMLhttpRequests), а затем обработки и отображения его с помощью javascript.

Неблокирующий характер AJAX является тем, что делает его таким распространенным программным шаблоном. Поскольку javascript блокируется в браузере, синхронный внешний вызов заставит браузер не отвечать на все вызовы до тех пор, пока он не вернет данные или не погаснет. Фактически, ваше приложение полностью зависит от внешней архитектуры и насколько хорошо оно будет работать.

Вызов AJAX обычно абстрагируется, чтобы обеспечить дополнительную функциональность или читаемость, но реализации (обычно) основаны на спецификации XMLHttpRequest .

## Examples

### Установка или настройка

## Что такое AJAX?

> AJAX означает асинхронный JavaScript и XML. В двух словах, это использование объекта XMLHttpRequest для связи с серверными сценариями. Он может отправлять и получать информацию в различных форматах, включая JSON, XML, HTML и даже текстовые файлы. -Mozilla Developer Network 2016

Самый простой способ реализации AJAX, особенно если вы планируете общаться с серверами, - это использовать jQuery.

## Что такое jQuery?

> jQuery - это быстрая, маленькая и многофункциональная библиотека JavaScript. Это упрощает использование HTML-обходных документов и манипуляций, обработки событий, анимации и Ajax с помощью простого в использовании API, который работает во множестве браузеров. -jquery.com

Для тех, кто не использовал много jQuery, подумайте об этом как о функциях, которые мы можем использовать для облегчения нашей жизни. Это идеально подходит для использования с AJAX, поскольку он сокращает количество кода, который мы должны писать, чтобы выполнить то же самое!

## Как добавить jQuery на ваш веб-сайт

Если вам нужно использовать Ajax, вам нужно добавить jQuery в свой проект. http://jquery.com/download/ В этой ссылке вы можете увидеть много способов добавить jquery. Вы можете использовать загруженную версию jQuery или использовать CDN. http://jquery.com/download/#jquery-39-s-cdn-provided-by-maxcdn . Но есть риск для безопасности, если вы используете CDN. Becauese проект призывает использовать jquery, поэтому хакер может манипулировать вызовом. Настолько лучше, если вы можете использовать загруженную версию. Давайте посмотрим, как добавить jquery в html-проект. Это просто. Первый пример - использовать Загруженный источник. Используйте эту ссылку для загрузки http://jquery.com/download/#jquery . Если вы просто хотите использовать jquery, я предлагаю вам скачать **Скачать сжатый, производственный jQuery 3.1.1.** Когда вы загружаете его, добавьте jquery-version.min.js в соответствующее место (например, папку javascript вашего проекта). Затем просто добавьте тег с src = jquery / location, как показано ниже.

```
<head>

<script src="path/from/html/page/to/jquery.min.js"></script>

</head>
```

Давайте посмотрим, как использовать CDN. Эта ссылка http://jquery.com/download/#using-jquery-with-a-cdn вы можете увидеть различные CDN (сеть доставки контента).

```
<head>

<script   src="https://code.jquery.com/jquery-3.1.1.min.js"   integrity="sha256-
hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="   crossorigin="anonymous"></script>

</head>
```

Как вы можете видеть, вам просто нужно добавить теги, которые поставщик CDN предоставляет. Теперь добавьте некоторые скрипты на страницу html, чтобы проверить, работает ли она.

```
<script>
$(document).ready(function(){
    alert("jQuery Works")
});
</script>
```

Если вы видите предупреждение **jQuery** , значит, вы добавили его правильно.

## Простой пример jQuery для связи с сервером

Взято с веб-сайта API `jQuery.ajax` :

```
$.ajax({
    method: "POST",
    url: "some.php",
    data: {
        name: "John",
        location: "Boston"
    },
    success: function(msg) {
        alert("Data Saved: " + msg);
    },
    error: function(e) {
        alert("Error: " + e);
    }
});
```

Этот кусок кода из-за jQuery легко читается и понимает, что происходит.

- `$.ajax` - этот бит вызывает функцию `ajax` jQuery.
- `method: "POST"` - эта строка здесь заявляет, что мы будем использовать метод POST для связи с сервером. Читайте по типам запросов!
- `url` - это переменная объявляет , когда запрос будет **отправлен** в. Вы посылаете запрос где - то. Это идея.
- `data` - довольно прямолинейно. Это данные, которые вы отправляете с запросом.
- `success` - эта функция здесь вы пишите, чтобы решить, что делать с данными, которые вы получаете назад. `msg` ! как показывает пример, в настоящее время он просто создает предупреждение с `msg` которое возвращается.
- `error` - эта функция здесь вы пишете, чтобы отображать сообщения об ошибках, или предоставлять действия для работы, когда запрос `ajax` прошел через ошибки.
- альтернативой `.done` является

  ```
  success: function(result) {
      // do something
  });
  ```

## Синхронизация - асинхронные запросы Async

### Асинхронный вызов ajax

При таком типе ajax-кода код не дожидается завершения вызова.

```
$('form.ajaxSubmit').on('submit',function(){
    // initilization...
    var form   = $(this);
    var formUrl    = form.attr('action');
    var formType   = form.attr('method');
    var formData   = form.serialize();

    $.ajax({
        url: formUrl,
        type: formType,
        data: formData,
        async: true,
        success: function(data) {
            // .....
        }
    });
    //// code flows through without waiting for the call to complete
    return false;
});
```

### Синхронный вызов ajax

При таком вызове ajax код ожидает завершения вызова.

```
$('form.ajaxSubmit').on('submit',function(){
    // initilization...
    var form   = $(this);
    var formUrl    = form.attr('action');
    var formType   = form.attr('method');
    var formData   = form.serialize();
    var data = $.ajax({
            url: formUrl,
            type: formType,
            data: formData,
            async: false
        }).responseText;
    //// waits for call to complete
    return false;
});
```

## Простой запрос Ajax, отправленный с помощью объекта XMLHttpRequest

```
var httpRequest = new XMLHttpRequest();

httpRequest.onreadystatechange = getData;

httpRequest.open('GET', 'https://url/to/some.file', true);
httpRequest.send();

function getData(){
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        alert(httpRequest.responseText);
    }
}
```

`new XMLHttpRequest()` создает новый объект *XMLHttpRequest* - это то, что мы отправим нашему запросу с помощью

`onreadystatechange` сообщает нашему запросу называть `getData()` каждый раз, когда он изменяет статус

`.open()` создает наш запрос - это принимает *метод запроса* (« **GET** », « **POST** » и т. д.), URL-адрес страницы, которую вы запрашиваете, и, необязательно, должен ли запрос быть **асинхронным**

`.send()` отправляет наш запрос - это необязательно принимает данные для отправки на сервер, например `.send(data)`

наконец, `getData()` - это функция, о которой мы говорили, должна вызываться каждый раз, когда **изменяется статус** нашего запроса. если readyState равен **DONE,** то он предупреждает `responseText` который является только данными, полученными с сервера.

Более подробную информацию можно найти в руководстве по началу работы на MDN.

## Использование ajax в ванильном javascript с простым обратным вызовом

Вот наша функция для создания простого ajax-вызова, написанного в vanilla javascript (не es2015):

```
function ajax(url, callback) {
    var xhr;

    if(typeof XMLHttpRequest !== 'undefined') xhr = new XMLHttpRequest();
    else {
        var versions = ["MSXML2.XmlHttp.5.0",
                        "MSXML2.XmlHttp.4.0",
                        "MSXML2.XmlHttp.3.0",
                        "MSXML2.XmlHttp.2.0",
                        "Microsoft.XmlHttp"]

         for(var i = 0, len = versions.length; i < len; i++) {
            try {
                xhr = new ActiveXObject(versions[i]);
                break;
            }
            catch(e){}
         } // end for
    }

    xhr.onreadystatechange = ensureReadiness;

    function ensureReadiness() {
        if(xhr.readyState < 4) {
            return;
        }

        if(xhr.status !== 200) {
            return;
```

```
        }

        // all is well
        if(xhr.readyState === 4) {
            callback(xhr);
        }
    }

    xhr.open('GET', url, true);
    xhr.send('');
}
```

и его можно использовать как:

```
ajax('myFile.html', function(response) {
    document.getElementById('container').innerHTML = response.responseText;
});
```

Если вы хотите использовать Ecmascript 6 (также известный как es2015), вы можете использовать метод **выборки** , который возвращает обещание:

```
fetch('myFile.json').then(function(res){
    return res.json();
 });
```

Для получения дополнительной информации о es2015 Обещания следуйте по ссылке ниже: Promises

### Выполнение асинхронного вызова AJAX с использованием TypeScript

# Добавление продукта в корзину для покупок

В следующем примере показано, как добавлять продукт (или что-либо) в таблицу базы данных асинхронно, используя AJAX и TypeScript.

```
declare var document;
declare var xhr: XMLHttpRequest;

window.onLoad = () =>
{
    Start();
};

function Start()
{
    // Setup XMLHttpRequest (xhr).
    if(XMLHttpRequest)
    {
        xhr = new XMLHttpRequest();
```

```
    }
    else
    {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }

    AttachEventListener(document.body, "click", HandleCheckBoxStateChange);
}

function AttachEventListener(element: any, e, f)
{
    // W3C Event Model.
    if(element.addEventListener)
    {
        element.addEventListener(e, f, false);
    }
    else if(element.attachEvent)
    {
        element.attachEvent("on" + e, (function(element, f)
        {
            return function()
            {
                f.call(element, window.event);
            };
        })
        (element, f));
    }

    element = null;
}

function HandleCheckBoxStateChange(e)
{
    var element = e.target || e.srcElement;

    if(element && element.type == "checkbox")
    {
        if(element.checked)
        {
            AddProductToCart(element);
        }
        else
        {
            // It is un-checked.
            // Remove item from cart.
        }
    }
    else
    {
        break;
    }
}

AddProductToCart(e)
{
    var element = <HTMLInputElement>document.getElementById(e.id);

    // Add the product to the Cart (Database table)
    xhr.onreadystatechange = function()
    {
        if(xhr.readyState == 4)
```

```
        {
            if(xhr.status == 200)
            {
                if(element != null)
                {
                    console.log("200: OK");
                }
                else
                {
                    console.log(":-(");
                }
            }
            else
            {
                // The server responded with a different response code; handle accordingly.
                // Probably not the most informative output.
                console.log(":-(");
            }
        }
    }

    var parameters = "ProductID=" + encodeURIComponent(e.id) + "&" + "Action=Add&Quantity=" +
element.value;

    xhr.open("POST", "../Cart.cshtml");
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.setRequestHeader("Content-length", parameters.length.toString());
    xhr.setRequestHeader("Connection", "close");

    xhr.send(parameters);

    return e.id;
}
```

Чтобы сделать этот пример завершен, обновите свой серверный код, чтобы фактически вставить эти данные в базу данных. В приведенном выше коде предполагается, что вы используете C # и имеете файл `Cart.cshtml` . Однако просто замените `cshtml` на `php` и напишите свою собственную логику на стороне сервера, используя язык по вашему выбору.

Прочитайте Начало работы с ajax онлайн: https://riptutorial.com/ru/ajax/topic/1082/начало-работы-с-ajax

# глава 2: Callbacks

## Examples

### Интерпретация ошибок с обратным вызовом «ошибка»

Ошибки при правильном управлении сервером будут возвращены вашему клиенту с определенным кодом статуса HTTP, отличным от 2xx (см. RFC 2616, раздел 10 ).

Рекомендуется глобально `$.ajaxSetup()` ваши ошибки из вашего `$.ajaxSetup()` как показано в примере ниже. Поэтому все ошибки, исходящие из ваших вызовов ajax, будут автоматически интерпретироваться из настройки ajax.

```
$.ajaxSetup({
    error: function (jqXHR, exception, errorThrown) {
        var message;
        var statusErrorMap = {
            '400': "Server understood the request, but request content was invalid.",
            '401': "Unauthorized access.",
            '403': "Forbidden resource can't be accessed.",
            '500': "Internal server error.",
            '503': "Service unavailable."
        };
        if (jqXHR.status) {
            message = statusErrorMap[jqXHR.status];
            if (!message) {
                message = "Unknown Error.";
            }
        } else if (exception == 'parsererror') {
            message = "Error.\nParsing JSON Request failed.";
        } else if (exception == 'timeout') {
            message = "Request Time out.";
        } else if (exception == 'abort') {
            message = "Request was aborted by the server";
        } else {
            message = "Unknown Error.";
        }

        // How you will display your error message...
        console.log(message);
        console.log(errorThrown);
    }
});
```

Вы также можете «перегрузить» обратный вызов `error` в определенном `$.ajax()` когда вы ждете сообщения об ошибке.

```
$.ajax({
    url: './api',
    data: { parametersObject },
    type:'post',
    dataType: 'json',
```

```
    success:function(output){
        // Interpret success
    },
    error: function(xhr,textStatus,ErrorThrown){
        // Specific error will not be interpreted by $.ajaxSetup
    }
});
```

Прочитайте Callbacks онлайн: https://riptutorial.com/ru/ajax/topic/7175/callbacks

# кредиты

| S. No | Главы | Contributors |
|---|---|---|
| 1 | Начало работы с ajax | acupajoe, adielhercules, Alessio Cantarella, Community, delete me, JhWebDevGuy, jignesh prajapati, MAZux, Menuka Ishan, Nicholas Qiao, TimTheEnchanter, Tolga Evcimen |
| 2 | Callbacks | maxime_039 |