



# АСИНХРОННЫЕ ЗАПРОСЫ



МИХАИЛ КУЗНЕЦОВ



# МИХАИЛ КУЗНЕЦОВ

Разработчик в ING Bank



[@mkuznetcov](https://www.telegram.me/mkuznetcov)

# ПЛАН ЗАНЯТИЯ

1. Введение в синхронность / асинхронность в JavaScript
2. Объект XMLHttpRequest
3. FormData и отправка данных формы



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Вопрос:** какие существуют основные типы полей формы и какие их наиболее важные события?

# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

## Ответ:

Типы полей:

- `input text`;
- `textarea`;
- `select`;
- `radio`;
- `checkbox`.

События:

- `input`;
- `change`;
- `focus`;
- `blur`.



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Вопрос:** с помощью какого события форму отправляют на сервер?



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Ответ: `Submit`.



# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ


**Вопрос:** Какой существует стандартный механизм валидации форм?





# ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

**Ответ:** JavaScript Validation API, метод `checkValidity`.



# **ВВЕДЕНИЕ В СИНХРОННОСТЬ / АСИНХРОННОСТЬ В JAVASCRIPT**

---

# АСИНХРОННОСТЬ В JAVASCRIPT

JavaScript часто описывается как **асинхронный** язык, что означает, что вызовы функций и операции не блокируют основной поток во время их выполнения. Это верно в некоторых ситуациях, например реакция на действие пользователя или запрос на сервер, но JavaScript **выполняется в одном потоке** и поэтому **имеет** множество **синхронных компонентов**.

*Синхронность* — запросы выполняются последовательно друг за другом.

*Асинхронный* — это когда синхронно (т. е. одновременно) с ним может выполняться другая операция.

## Пример синхронного JavaScript

```
1  let x = 10;  
2  console.log(x);  
3  console.log(x + 1);  
4  
5  // 10  
6  // 11
```

**ВАЖНО!** Помните, что при синхронном запросе страница **блокируется**, посетитель **не сможет нажимать** ссылки, кнопки и тому подобное.

# ПРИМЕР АСИНХРОННОГО JAVASCRIPT

```
console.log('1')
setTimeout(function afterTwoSeconds() {
  console.log('2')
}, 2000)
console.log('3')
// будет выведена последовательность 1, 3, 2.
```

**Коллбэк будет вызван**, в данном примере, **через 2 секунды**. Приложение при этом **не остановится**, ожидая, пока истекут эти две секунды.

# ПРОТОКОЛ ПЕРЕДАЧИ ГИПЕРТЕКСТА HTTP

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных:

1. Клиентское приложение формирует запрос;
2. Отправляет его на сервер;
3. Сервер формирует ответ;
4. Передаёт его обратно клиенту.



Обычно HTTP запрос содержит:

- **строку запроса**, в которой указывается версия HTTP протокола и HTTP метод запроса;
- **заголовки**, в которых передаются другие HTTP параметры для успешного HTTP соединения;
- **пустую строку**, чтобы отделить служебную информацию от тела сообщения;
- **необязательное** тело сообщения.

# ПРИМЕР HTTP ЗАПРОСА

```
GET /index.php HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5
Accept: text/html
Connection: close
```

Первая строка — это строка запроса, остальные — заголовки.



## ПРИМЕР ОТВЕТА НА ЗАПРОС

HTTP/1.0 200 OK

Server: nginx/0.6.31

Content-Language: ru

Content-Type: text/html; charset=utf-8

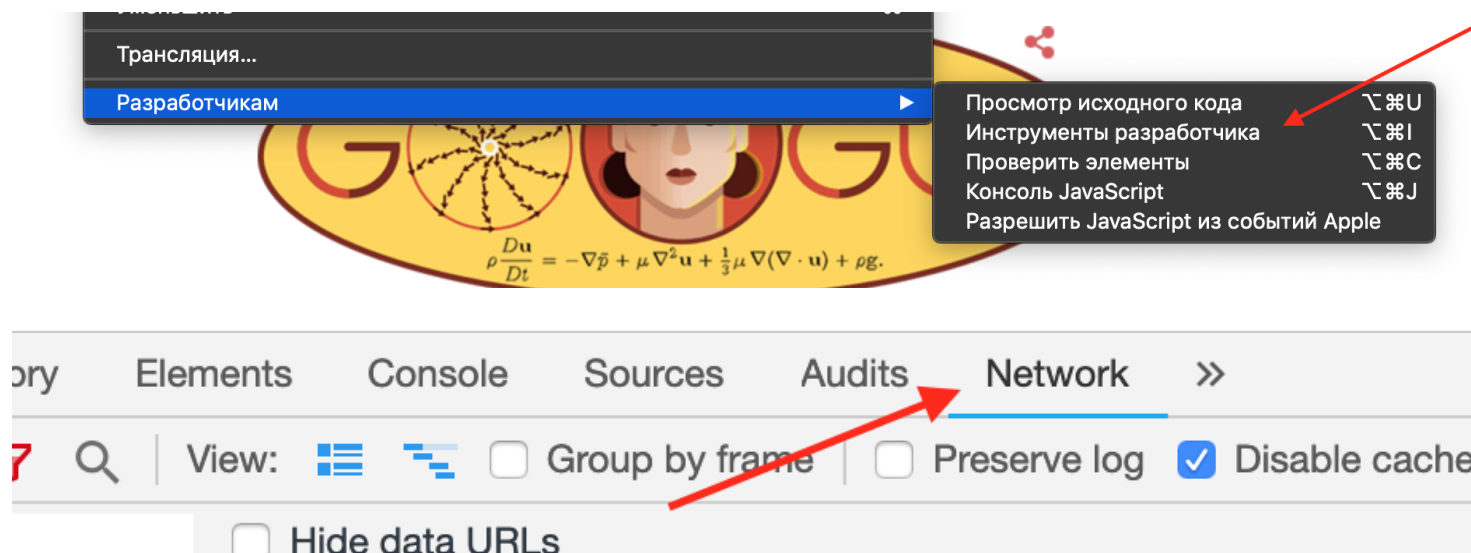
Content-Length: 1234

Connection: close

# КАК ПОСМОТРЕТЬ ЗАПРОС

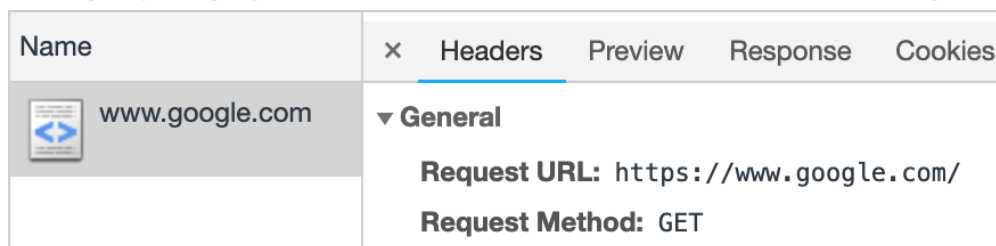
Посмотреть пример такого запроса можно в браузере, например Google Chrome. Для этого перейдем по сл. пути:

— посмотреть -> разработчикам -> инструменты разработчика -> Network



Посмотрим на работу протокола на примере посещения страницы [google.com](https://www.google.com/):

1. Браузер устанавливает соединение с сервером и отправляет запрос:



2. Получаем код, список заголовков и тело ответа:

<b>Status Code:</b> 🟢 200	<b>cache-control:</b> private, max-age=0
	<b>content-encoding:</b> br
	<b>content-length:</b> 72055
	<b>content-type:</b> text/html; charset=UTF-8
	<b>date:</b> Thu, 07 Mar 2019 14:57:30 GMT

# HTTP - ЗАГОЛОВКИ ЗАПРОСА

Заголовки запроса HTTP имеют стандартную для заголовка HTTP структуру: не зависящая от регистра строка, завершаемая двоеточием и значение, структура которого определяется заголовком. Весь заголовок - одна строка.

Заголовки запроса можно разделить на несколько групп:

- Основные заголовки (General headers);
- Заголовки запроса (Request headers);
- Заголовки сущности, например Content-Length.




# HTTP - КОДЫ СОСТОЯНИЯ

Код состояния HTTP — часть первой строки ответа сервера при запросах по протоколу HTTP.

Это целое число из трёх десятичных цифр. Первая цифра указывает на класс состояния.

За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.



Код ответа (состояния) HTTP показывает, был ли успешно выполнен определённый HTTP запрос. Коды сгруппированы в 5 классов:

- **2xx** – обработка запроса завершилась **успешно**;
- **3xx** – в результате обработки запроса сервер **перенаправляет** нас по другому адресу;
- **4xx** – **ошибка** в самом запросе (например, указан неверный путь на сервере или нет доступа к запрашиваемому ресурсу);
- **5xx** – произошла **ошибка** на сервере в процессе обработки запроса (например, ошибка в приложении, обрабатывающем запрос, привела к его падению).



## РЕЗЮМИРУЕМ

Выводы: **протокол HTTP** предназначен для **создания** требуемого высокоскоростного **обмена данными** между **пользователем, сервером** и поставщиком информации.

После программной обработки сервер дает ответ пользователю виде предоставленной ответной информации.

**Но как выполнить запрос на сервер?**



**ОБЪЕКТ**

**XMLHttpRequest**



# ОБЪЕКТ XMLHttpRequest

Объект XMLHttpRequest дает возможность браузеру делать HTTP-запросы к серверу без перезагрузки(асинхронно) страницы.

- 1 // 1. URL запроса
- 2 // 2. Создаём новый объект XMLHttpRequest
- 3 // 3. Конфигурируем его: GET-запрос
- 4 // 4. Отправляем запрос на сервер

Чтобы начать работать с XMLHttpRequest, выполните этот код:

```
let xhr = new XMLHttpRequest();  
// экземпляр объекта XMLHttpRequest  
console.log(xhr);
```




```
XMLHttpRequest {onreadystatechange: null, readyState: 0, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, ...} ⓘ  
  onabort: null  
  onerror: null  
  onload: null  
  onloadend: null  
  onloadstart: null  
  onprogress: null  
  onreadystatechange: null  
  ontimeout: null  
  readyState: 0  
  response: ""  
  responseText: ""  
  responseType: ""  
  responseURL: ""
```

Как правило, XMLHttpRequest используют для загрузки данных. Различают два использования XMLHttpRequest - синхронное и асинхронное.

# СПОСОБЫ ОТЛАДКИ XMLHttpRequest

Для того чтобы проиграть всевозможные варианты, достаточно снова обратиться к консоли разработчика, которая безусловно сообщит если возникнут ошибки, как на примере ниже:

Name	Status	Type	Initiator
 posts.json /xhr/assets	(blocked:devtools)	xhr	j.js:15 Script



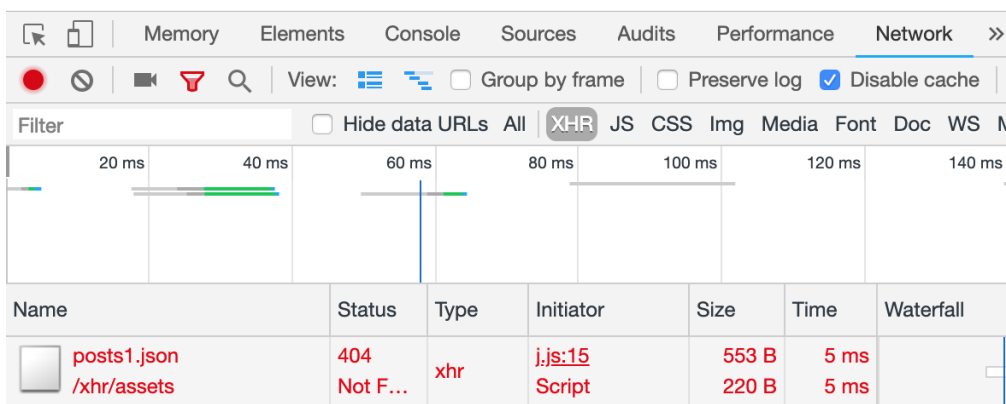
Memory Elements Console Sources Audits Performance Network >>

top Filter Default levels ▾

Navigated to <http://localhost:8888/xhr/>

✖ GET <http://localhost:8888/xhr/assets/posts1.json> 404 (Not Found)

>





Memory Elements Console Sources Audits Performance Network >>

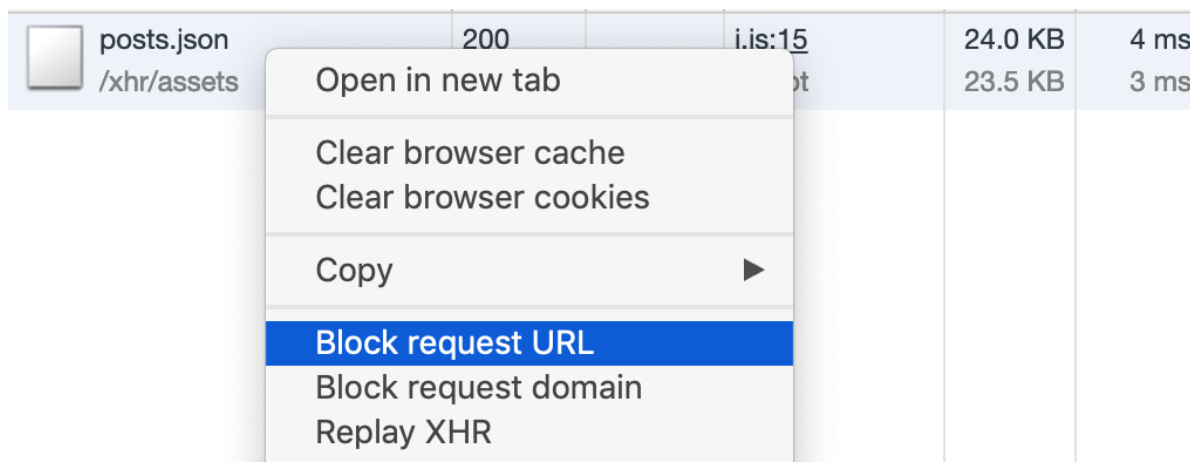
View: [List Icon] [Code Icon] [Group by frame] [Preserve log] [Disabling cache] [Filter]

Hide data URLs All XHR JS CSS Img Media Font Doc WS M

20 ms 40 ms 60 ms 80 ms 100 ms 120 ms 140 ms

Name	Status	Type	Initiator	Size	Time	Waterfall
 posts1.json /xhr/assets	404 Not F...	xhr	j.js:15 Script	553 B 220 B	5 ms 5 ms	

Но также можно симулировать ситуации, например в случае успешного выполнения запроса и получения требуемого ответа:



Мы можем заблокировать URL запроса, и посмотреть что в таком случае отобразит браузер или например заблокировать доступ к запрашиваемому домену.

Таким образом вы всегда будете готовы к разным исходам при выполнении запроса.

# МЕТОДЫ ОБЪЕКТА XMLHttpRequest

Далее мы более подробно разберём основные методы и свойства объекта XMLHttpRequest:

- `open()`;
- `setRequestHeader()`;
- `send()`;
- и др.

## МЕТОД `open()`

Этот метод – как правило, вызывается первым после создания объекта XMLHttpRequest. Добавим его в наш пример:

```
let xhr = new XMLHttpRequest(); // экземпляр объекта XMLHttpRequest
xhr.open(); // создаем запрос
```

Варианты вызова:

- `open( method, URL );`
- `open( method, URL, async );`

В этом примере через `XMLHttpRequest`, с сервера запрашивается страница `***`, для этого откроем соединение с помощью метода `open`:

```
XMLHttpRequest.open(параметр1, параметр2, параметр3);  
// открываем соединение
```

1. Первый параметр `method` - **HTTP-метод**. Как правило, используется `GET` либо `POST`;
2. **URL - адрес** запроса. Можно использовать не только HTTP/HTTPS;
3. Необязательный логический параметр со значением по умолчанию `true`, указывающим, выполнять ли операцию **асинхронно** или нет.

```
1 xhr.open('GET', url); // асинхронно
2 xhr.open('GET', url, false); // синхронно
3 /*
4     При синхронном запросе весь JavaScript "подвиснет",
5     пока запрос не завершится
6 */
```

Если синхронный вызов занял слишком много времени, то браузер предложит закрыть «зависшую» страницу.

**Вывод:** метод `open` настраивает запрос на открытие соединения.



## ОТПРАВЛЯЕМ ЗАПРОС МЕТОДОМ `send()`

Метод `send()` отправляет запрос. Если запрос асинхронный (каким он является по-умолчанию), то возврат из данного метода происходит сразу после отправления запроса.

После инициализации запроса методом `open()` необходимо отправить запрос с помощью метода `send()`:

```
1 let xhr = new XMLHttpRequest(); // экземпляр объекта XMLHttpRequest
2 xhr.open('GET', '/xhr/data.txt'); // создаем асинхронный запрос
3 xhr.send(); // отправляем запрос
```

**Вывод:** Именно этот метод открывает соединение и отправляет запрос на сервер.

## МЕТОД `setRequestHeader()`

Метод `setRequestHeader(name, value)` устанавливает значение HTTP заголовка `name` запроса со значением `value`.

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

Устанавливается после метода `open()`, но до `send()`:

```
1 let xhr = new XMLHttpRequest(); // экземпляр объекта XMLHttpRequest
2 xhr.open('GET', '/xhr/data.txt'); // создаем асинхронный запрос
3 xhr.setRequestHeader('Content-Type', 'application/json');
4 xhr.send(); // отправляем запрос
```

## ДРУГИЕ МЕТОДЫ ДЛЯ РАБОТЫ С ЗАПРОСАМИ

Также существуют такие методы, как:

- `abort()` - предотвращает уже отправленный запрос;
- `getResponseHeader()` - возвращает строку, содержащую текст определённого хэдера (header).

Ознакомиться с особенностями методов объекта XMLHttpRequest можно на странице [документации](#).

## СВОЙСТВА XMLHttpRequest

Объект XMLHttpRequest имеет ряд свойств, которые позволяют проконтролировать выполнение запроса.

Рассмотрим основные свойства, содержащие ответ сервера:

- `status`

**HTTP-код** ответа: 200, 404, 403 и так далее. Может быть также равен 0, если сервер не ответил или при запросе на другой домен.

- `statusText`

Текстовое **описание** статуса от сервера: OK, Not Found, Forbidden и так далее.

- `responseText`

Текст **ответа** сервера.

# СВОЙСТВА `onreadystatechange` И `readyState`

Когда серверу посылается запрос, мы хотим выполнить некоторые действия на основе ответа.

Событие `onreadystatechange` происходит каждый раз, когда свойство `readyState` (состояние готовности) изменяется.

Свойство `readyState` содержит состояние запроса `XMLHttpRequest` и принимает значения **от 0 до 4**:

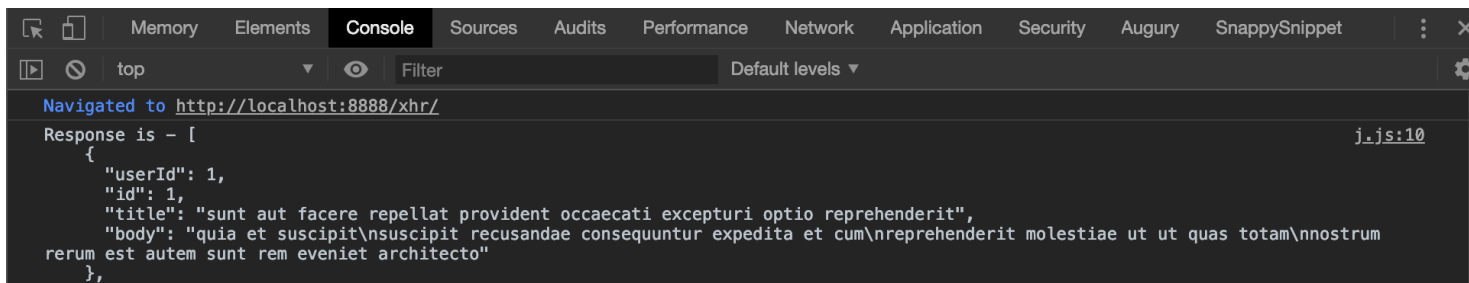
- 0 - запрос не инициализирован;
- 1 - установлено соединение с сервером;
- 2 - запрос получен;
- 3 - обработка запроса;
- 4 - запрос завершен и ответ готов.

# ПРИМЕР

Рассмотри пример уже частично известного нам запроса:

```
1 let xhr = new XMLHttpRequest();
2 xhr.open('GET', '/assets/posts.json');
3 xhr.send();
4 xhr.onreadystatechange = function () {
5     if(xhr.readyState === 4) {
6         console.log(xhr.responseText);
7     };
8 };
```

При таком запросе мы получаем соответствующий запросу ответ:



Свойство `XMLHttpRequest.readyState` возвращает текущее состояние объекта `XMLHttpRequest`. Объект XHR может иметь следующие состояния:

- UNSENT (Объект был создан. Метод `open()` ещё не вызывался);
- OPENED (Метод `open()` был вызван);
- HEADERS\_RECEIVED (Метод `send()` был вызван, доступны заголовки (headers) и статус);
- LOADING (Загрузка; `responseText` содержит частичные данные);
- DONE (Операция полностью завершена).

Пример добавления константы для удобства `xhr.DONE` :

```
xhr.addEventListener('readystatechange', () {  
    if(xhr.readyState === xhr.DONE) console.log(xhr.responseText)  
});
```

## СВОЙСТВО `status` И `statusText`

Свойство `status` содержит статусный код ответа HTTP, который пришел от сервера.

```
if (xhr.readyState === xhr.DONE && xhr.status === 200) {  
    // инструкция  
}
```

С помощью статусного кода можно судить об успешности запроса или об ошибках, которые могли бы возникнуть при его выполнении. Статусы бывают например такими:

- `Status`: 200, `statusText`: "OK (все хорошо)"
- `Status`: 400, `statusText`: "Страница не найдена"

Полный перечень кодов ответы можно найти [здесь](#).



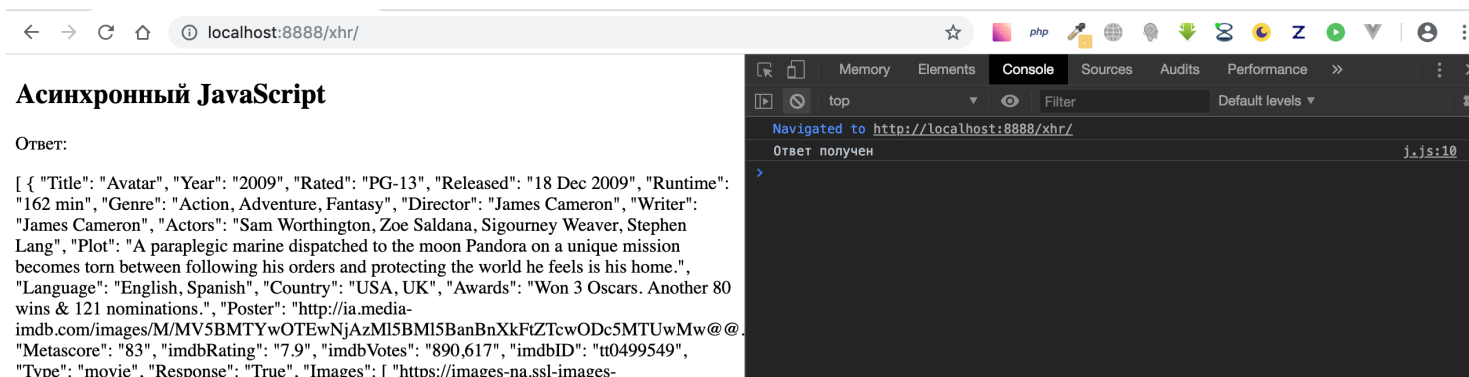
# СВОЙСТВО `responseText` И `responseType`

Только для чтения `XMLHttpRequest` свойство `responseText` возвращает текст ответа.

Если ответ сервера не является XML документом, следует использовать свойство `responseText`.

```
document.getElementById("response").innerHTML=xhr.responseText;
```

Свойство `responseText` возвращает ответ в виде строки, и Вы можете далее использовать эту строку соответствующим образом:



Свойство `responseType` является перечислимым значением, которое возвращает тип ответа.

Он также позволяет автору изменять тип ответа.

# xhr.responseText

Прежде чем отправить запрос, необходимо задать для свойства `xhr.responseText` значение `text`, `arraybuffer`, `blob` или `document`.

Обратите внимание: если установить значение `xhr.responseText = ''` или опустить его, по умолчанию выбирается формат `text`. Другие форматы данных можно найти на странице [документации](#).

"arraybuffer"	ArrayBuffer
"blob"	Blob
"document"	Document
"json"	JSON
"text"	DOMString

## СВОЙСТВО `withCredentials`

Свойство `withCredentials` это `Boolean` который определяет, должны ли создаваться **кросс-доменные Access-Control** запросы с использованием таких идентификационных данных как **cookie**, **авторизационные заголовки** или **TLS сертификаты**.

**ВАЖНО!** Настройка `withCredentials` бесполезна при запросах на тот же домен.

Это часть контроля безопасности, благодаря которому кросс-доменные запросы находятся под пристальным вниманием, и злоумышленникам не получится передать «вредоносную» информацию.

**ВАЖНО!** По умолчанию браузер не передаёт с запросом куки и авторизующие заголовки.

Чтобы браузер передал вместе с запросом куки и HTTP-авторизацию, нужно поставить запросу `xhr.withCredentials = true`:

```
1 var xhr = new XMLHttpRequest();  
2 xhr.open('GET', 'http://example.com/', true);  
3 xhr.withCredentials = true;  
4 xhr.send(null);
```

Если бы свойства `withCredentials` не было, то передавать можно было бы без контроля любые данные.

Также используется для определения, будут ли проигнорированы куки переданные в ответе. Значение по умолчанию - `false`. Поэтому если оставить значение по умолчанию (`false`), то браузер не пошлет никаких идентификационных данных.

# XMLHttpRequest : ИНДИКАЦИЯ ПРОГРЕССА

Запрос `XMLHttpRequest` состоит из двух фаз:

1. Стадия **загрузки (upload)**. На ней данные загружаются на сервер;
2. Стадия **скачивания (download)**. После того, как данные загружены, браузер скачивает ответ с сервера. На этой стадии используется обработчик `xhr.onprogress`.

# СТАДИЯ ЗАКАЧКИ UPLOAD

На стадии загрузки для получения информации используем объект `xhr.upload`. У этого объекта нет методов, он только генерирует события в процессе загрузки.

Полный список событий:

- `loadstart`;
- `progress`;
- `abort`;
- `error`;
- `load`;
- `timeout`;
- `loadend`.

```
1  xhr.upload.onprogress = function() {  
2      alert( 'Загружено на сервер' );  
3  }  
4  
5  xhr.upload.onload = function() {  
6      alert( 'Данные полностью загружены на сервер!' );  
7  }  
8  
9  xhr.upload.onerror = function() {  
10     alert( 'Произошла ошибка при загрузке данных на сервер!' );  
11 }
```



## РЕЗЮМИРУЕМ

Объект `XMLHttpRequest` (или, как его кратко называют, «XHR») дает возможность из JavaScript делать HTTP-запросы к серверу без перезагрузки страницы.

Общий план работы с объектом `XMLHttpRequest` можно представить следующим образом:

1. Создание экземпляра объекта `XMLHttpRequest`.
2. Открытие соединения с сервером методом `open`.
3. Непосредственно отправка запроса методом `send`.

# РЕЗЮМИРУЕМ


Основные методы для отправки запросов `XMLHttpRequest`:

- `open(Method, Url, async)`;
- `send(data)`;
- `onreadystatechange`;


Ответ сервера находится в:

- `responseText`;
- `responseXML`;
- `status`;
- `statusText`.

**Вопрос:** напомним, в чем разница между синхронным и асинхронными запросами?



# FormData И ОТПРАВКА ДАННЫХ ФОРМЫ



XMLHttpRequest 2 добавляет поддержку для нового интерфейса `FormData`. Объекты `FormData` позволяют вам легко представлять поля формы и отправлять их с помощью метода `send()`.

## FormData И ОТПРАВКА ДАННЫХ ФОРМЫ

Объект `FormData` предназначен для кодирования данных, которые необходимо отправить на сервер посредством технологии AJAX (`XMLHttpRequest`).

Для кодирования данных метод `FormData` использует формат `"multipart/form-data"`. Это означает то, что он позволяет подготовить для отправки по AJAX не только текстовые данные, но и файлы (`input` с `type`, равным `file`).

Экземпляр new `FormData([form])` вызывается либо без аргументов, либо с DOM-элементом формы.

```
1 <form name="person">
2   <input name="name" value="Alex">
3   <input name="surname" value="Javascript">
4 </form>
```

```
1 <script>
2   // создать объект для формы
3   var formData = new FormData(document.forms.person);
4 </script>
```

## СТРУКТУРА ОБЪЕКТА `FormData`

Представить себе объект `FormData` можно как набор пар "ключ-значение".

Другими словами, как некоторую коллекцию элементов в которой каждый из них представлен в виде ключа и значения (массива значений).

## РАБОТА С ОБЪЕКТОМ `FormData`

Работа с объектом `FormData` начинается с его создания:

```
var formData = new FormData();
```

После создания объекта `FormData` вы можете использовать его различные методы.



## МЕТОД `append`

Один из наиболее используемых методов – это `append`. Этот метод добавляет в объект `FormData` новую порцию данных (ключ-значение).

Метода `append` имеет такую структуру:

```
formData.append(name, value);  
formData.append(name, value, filename);
```

## ПРИМЕР ИСПОЛЬЗОВАНИЯ

Рассмотрим пример использования метода `append`:

```
formData.append('key', 'value1'); //{"key":"value1"}
```

Если указанный **ключ уже есть** у объекта `FormData`, то данный метод запишет его значение в качестве следующего значения этого ключа.

```
formData.append('key', 'value2'); //{"key":["value1", "value2"]}
```

## МЕТОД `delete`

Для удаления данных из объекта `FormData` предназначен метод `delete`. Он убирает элемент из объекта `FormData` по имени ключа.

```
formData.delete('key');
```

# ИСПОЛЬЗОВАНИЕ `FormData` ДЛЯ КОДИРОВАНИЯ ДАННЫХ ФОРМЫ

Рассмотрим простой `XMLHttpRequest` пример, в котором разберём, как применять объект `FormData` для кодирования данных формы.

Разберем пример простой отправки данных. Для начала создаем экземпляр объекта `FormData` :

```
let formData = new FormData();
```

Далее добавим пару (ключ, значение) с данными который собираемся отправить используя известный нам метод `append()` :

```
formData.append("name", "Alex");  
formData.append("age", 25);
```

Далее мы создадим асинхронный запрос с помощью XMLHttpRequest :

```
1  let xhr = new XMLHttpRequest();
2  xhr.open("POST", "/handler.php", true);
3  xhr.send(formData);
4  /*
5   отправим запрос передав в него экземпляр FormData
6   созданный ранее
7   */
```

# РЕАЛЬНЫЙ ПРИМЕР ОТПРАВКИ ДАННЫХ ИСПОЛЬЗУЯ `xmlHttpRequest` И `FormData`

Данный пример будет выполнять следующие основные действия:

- **отправлять** HTML форму на сервер;
- **обрабатывать** данные формы на сервере;
- **получать** ответ от сервера;
- **выводить** обрабатывая посредством JavaScript.

Разработку этого примера начнём с создания HTML формы и контейнера для вывода результата.

```
1 <form id="message">
2   <label for="name">Имя:</label>
3   <input type="text" class="form-control" name="name">
4
5   <label for="name">Сообщение:</label>
6   <textarea class="form-control" rows="3" name="message"></textarea>
7
8   <button id="send-message" class="btn btn-primary">Отправить сообщение</button>
9 </form>
```

Сценарий на JavaScript, который будет кодировать данные HTML формы ( `FormData` ), отправлять её на сервер ( `XMLHttpRequest` ), получать ответ с сервера и отображать его на странице в виде маркированного списка.

```
1  <script>
2    // получим форму с id = "message"
3    var message = document.getElementById('message');
4    message.addEventListener('submit', (e) => {
5
6        var formData = new FormData(message);
7        var request = new XMLHttpRequest();
8        request.open('POST', 'process.php');
9        request.addEventListener('readystatechange', function() {
10            if (this.readyState == request.DONE && this.status == 200) {
11
12                var data = JSON.parse(this.responseText);
13                var output = '<ul>';
14                for (var key in data) {
15                    output += '<li><b>' + key + "</b>: " + data[key] + '</li>';
16                }
17                output += '</ul>';
18                document.getElementById('result').innerHTML = output;
19            }
20        });
21        request.send(formData);
22        e.preventDefault();
23    });
24 </script>
```



## ЧЕМУ МЫ НАУЧИЛИСЬ?

- Понимать различия синхронного и асинхронного JavaScript;
- Понимать работу протокола HTTP;
- Использовать `XMLHttpRequest`;
- Создавать запросы к серверу
- Передавать данные с помощью `FormData`.

# СПИСОК ЛИТЕРАТУРЫ, ЧТО ПОЧИТАТЬ

- <https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest>
- <https://learn.javascript.ru/ajax-xmlhttprequest>
- <https://javascript.ru/ajax/transport/xmlhttprequest>
- <https://developer.mozilla.org/ru/docs/Web/API/FormData>
- <https://learn.javascript.ru/xhr-forms>
- <https://developer.mozilla.org/ru/docs/Web/HTTP>

# ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаем в Slack!
- Работы должны соответствовать принятому [стилю оформления кода](#).
- Зачет по домашней работе проставляется после того, как приняты все **3 задачи**.



**Задавайте вопросы и напишите отзыв о лекции!**

**МИХАИЛ КУЗНЕЦОВ**

