ЛАБОРАТОРНАЯ РАБОТА №3

**По курсу «Интернет-программирование»**

Изучение технологий асинхронного (фонового) обмена данными для динамических страниц

**Цель работы.**

Ознакомиться с технологиями фонового обмена данными для Интернет-приложений. Практически освоить написание сценариев с применением технологий «фонового» обмена данными.

**Приобретаемые компетенции:**

1. Понимание основных принципов работы технологий фонового обмена данными.
2. Практическая реализация сценария фонового обмена данными между клиентом и сервером по лабораторному заданию.

# **Введение**

По умолчанию обмен данными между клиентом и сервером в Интернет-приложении с помощью браузера загружает код страницы и связанные с ней ресурсы, полученные с сервера. Такой способ содержит ряд неудобств для случаев когда на странице имеет смысл обновить только часть информации, например, в зависимости от выбранной страны, загрузить список городов этой страны. Очевидно, что для таких случаев желательно иметь способы получения данных с сервера без перезагрузки уже отображающейся страницы т.е. фоновый обмен данными.

## **Технология AJAX**

Для реализации фонового обмена в 2005 году появилась концепция AJAX - Asynchronous Javascript and XML, базирующаяся на использовании API класса XMLHttpRequest. Первая реализация XMLHttpRequest появилась в 2000 году, а последняя официальная спецификация в 2012 году. XMLHttpRequest позволяет с помощью метода send асинхронно в фоновом режиме отправить запрос на сервер и получить ответ в виде DOMString или XML, соответственно в свойство responseText или responseXML. Свойства и методы класса XMLHttpRequest Приведены в Приложении 1. Практически для работы с AJAX удобно пользоваться интерфейсом фреймворка jQuery, содержащего удобные методы для работы с XMLHttpRequest, которые описаны в документации по адресу — <https://api.jquery.com/jquery.ajax/>.

К характерным особенностями AJAX можно отнести:

* Инициирование на стороне сервера работы полноценной страницы с соответствующим потреблением ресурсов.
* Избыточность протокола HTTP для ряда задач частого обмена, короткими, сравнительно простыми сообщениями. Например, чатов.
* Невозможность поддержания постоянного соединения с сервером и, таким образом, невозможность удобного построения систем обмена данными в реальном времени
* Невозможность осуществления двунаправленного обмена данными.

## **Протокол WebSocket**

Для решения задач эффективного фонового обмена данными в режиме реального времени разработан протокол WebSocket. В 2009 году поддержка этого протокола впервые была реализована в браузере Google Chromium. Протокол WebSocket работает поверх TCP-соединения и реализует двунаправленный обмен данными между клиентом и сервером. Протокол WebSocket может использоваться не только в браузерах. Для работы с WebSocket в браузерах реализован стандартный объект WebSocket. Свойства и методы объекта WebSocket приведены в Приложении 3. Действующий стандарт протокола WebSocket доступен по адресу — <https://tools.ietf.org/html/rfc6455>. Для реализации обмена по протоколу WebSocket со стороны сервера существуют библиотеки для разных языков (Java, javaScript, PHP) и разных платформ, например, Node.js. Первоначальное соединение с сервером осуществляется по протоколу HTTP, путем отправки специального запроса. Если сервер поддерживает обмен по протоколу WebSocket, он отправляет подтверждающий HTTP-ответ, после чего обмен данными уже идет только по протоколу WebSocket.

# Настройка и использование AJAX

Для работы AJAX необходим запущенный web-сервер. Для этого из ПО лабораторной работы можно запустить пакет Denwer или OpenServer. Рекомендуется использовать OpenServer, как более современный. Перед запуском OpenServer в папке C:\OpenServer\OSPanel\domains нужно создать папку с будущим сайтом вида: номер\_студбилета.ru. Тогда после старта сервера этот домен будет доступен через браузер. Затем в папке сайта можно создавать файлы, которые будут генерировать запросы («клиенты») и файлы, которые будут возвращаться по этим запросам («ресурсы» или «серверы»). Если в адресе запроса присутствует статический файл: htm, html, txt, xml и т.д., то возвращается сам этот файл («ресурсы»). Если в адресе запроса присутствует динамический файл: php, asp, pl и т.д., то возвращается результат работы этого файла («сервер»). При обращении к динамическим файлам с помощью AJAX возможно использование параметров, для управления результатом, возвращаемым динамическим файлом. Для практического освоения работы AJAX изучите и запустите на сервере код раздела «Пример применения AJAX».

Для работы с кодом можно использовать текстовый редактор Far или IDE NetBeance. Во втором случае создайте проект типа PHP или HTML5/JavaScript (см. рис. 1). При выборе папки с исходным кодом проекта — выберите на диске папку с вашим сайтом, созданным в Denwer или OpenServer (см. рис. 2). На третьем шаге укажите URL вашего сайта (см. рис. 3). Четвертый и пятый шаги настройщика пройдите без изменений.

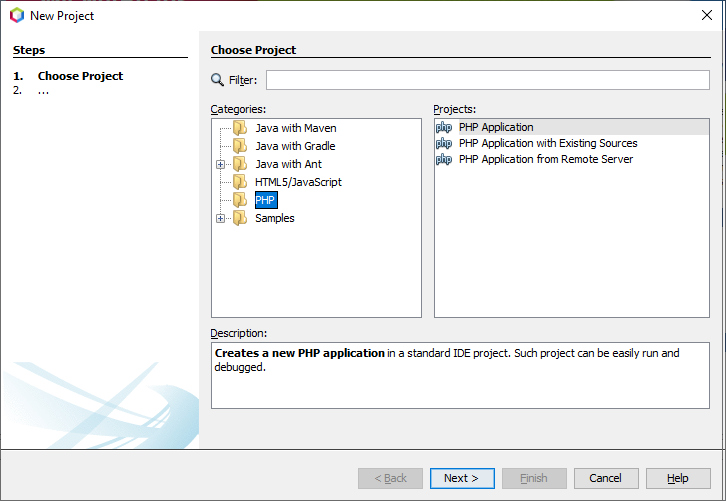


Рис. 1. Выбор типа проекта.

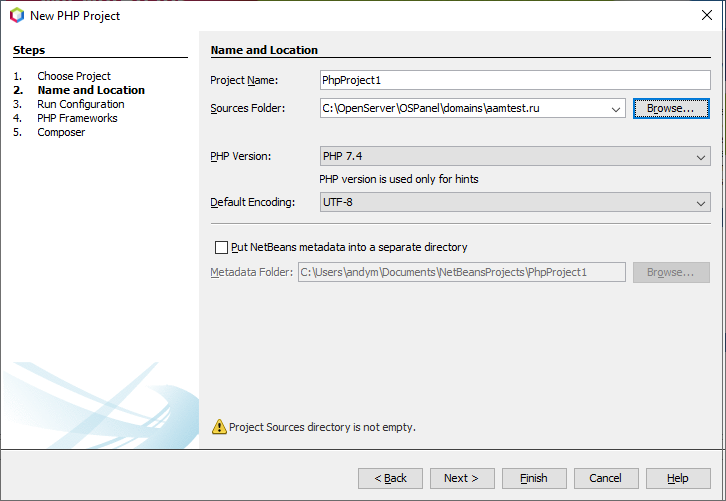


Рис. 2. Выбор папки проекта.

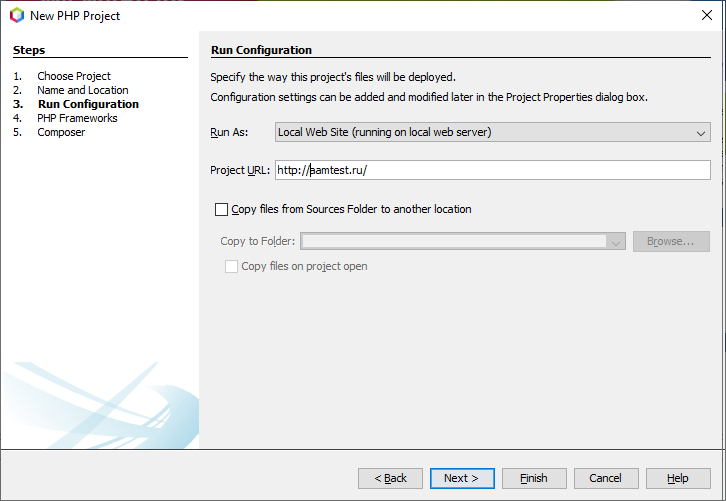


Рис 3. Настройка URL проекта.

# Настройка и использование WebSocket

По умолчанию базой для клиента WebSocket является браузер, в котором поддержка WbSocket встроена по умолчанию и реализуется с помощью объекта WebSocket языка JavaScript. Серверная часть WebSocket может быть реализована на различных языках программирования и серверных платформах. В лабораторной работе платформой для сервера WebSocket является Node.js (установлена версия 11.7.0). Для работы с WebSocket на Node.js потребуется установить пакет ws с помощью пакетного менеджера npm (установлена версия 6.5.0). Для установки требуется из консоли выполнить команду npm install -g ws.

Для работы с WebSocket в IDE NetBeance следует создать проект типа HTML5/JavaScript и указать в качестве источника кода папку вашего сайта. Затем разместите в папке сайта файлы из раздела «Пример применения WebSocket» и запустите сервер из NetBeance (открыть файл main.js и запустить «зеленой стрелкой» в панели инструментов).  
  
У объекта WebSocket браузера всего 4 события:

1. open – соединение установлено (метод onopen),
2. message – получены данные (метод onmessage),
3. error – ошибка (метод onerror),
4. close – соединение закрыто (метод onclose)

В Листинге 4 ко всем этим событиям добавлены обработчики.

# Пример применения AJAX

В примере проводится проверка авторизации клиента по передаваемым данным на сервер (файл ajax\_server.php) и возвращение ответа в контейнер с id=”Test”.

**Листинг 1.** Клиент - файл ajax\_client.php.   
Функция getContent\_putNode в браузере использует метод ajax библиотеки jQuery для отправки запроса на сервер и помещения ответа методом html в заданный селектором Param.selector контейнер страницы.

<!doctype html>

<head>

<meta name='viewport' content='width=device-width, initial-scale=1'>

<script

src="https://code.jquery.com/jquery-3.3.1.min.js"

integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="

crossorigin="anonymous"></script>

<script>

//Объект с параметрами запроса, который будет отправляться с помощью XMLHTTPRequest

var Params = {

url: "http://ip-spintex.ru/ajax\_server.php", //Адрес файла, которому будет отправлен запрос

async: true, //Включение асинхронной передачи

data: { login: "", password: 0 }, //Параметры, которые будут отправлены в запросе

selector: "#Test" //id контейнера в который будет выведен результат

};

function getContent\_putNode(Params)

{

//Получаем контент и размещаем его в заданном контейнере

var Result = "";

//Отправка ajax запроса с помощью метода ajax библиотеки jQuery

$.ajax({

type: "GET", //тип запроса — Get. Параметры будут переданы в URL

url: Params.url,

data: Params.data,

async: Params.async,

success: function(msg) {

Result = msg;

if( Params.selector != "" && Params.selector != undefined )

{ $(Params.selector).html($(Params.selector).html()+msg); }

},

error: function(jqXHR, textStatus, errorThrown)

{

Result = jqXHR.getAllResponseHeaders()+"<hr>"+textStatus+"<hr>"+errorThrown;

if( Params.selector != "" && Params.selector != undefined )

{ $(Params.selector).html(Result); }

}

});

return Result;

}

function Authorisation() {

Params.data.login = document.getElementById('login').value;

Params.data.password = document.getElementById('pass').value;

getContent\_putNode(Params);

}

</script>

</head>

<body>

Логин <input type="text" name="login" id="login"><br>

Пароль <input type="password" name="pass" id="pass"><br>

<div id="Test" style="width:50%; height:50px; border:1px #787878 solid; padding:5px;"></div>

<input type="button" value="Получить AJAX ответ" onclick="Authorisation()">

</body>

</html>

**Листинг 2.** Сервер - файл ajax\_server.php.

<?

//Получаем данные из входных массивов, очищая от не желательных символов

$login = strip\_tags ($\_REQUEST["login"]);

$pass = strip\_tags ($\_REQUEST["password"]);

if( $login == "Iam" && $pass == 12345 ) {

echo "Успешная авторизация"; //Возвращаем сообщение простым выводом в файл

} else {

echo "Ошибка авторизации";

}

?>

# Пример применения WebSocket

В примере демонстрируется создание простого чата (Листинг 3 - клиент, Листинг 4 — сервер). Сервер чата реализован на базе сервера Node.js (<https://nodejs.org/ru/> ) для которой с помощью менеджера пакетов npm (<https://www.npmjs.com/> ) установлена библиотека ws (<https://www.npmjs.com/package/ws> ).

При первом подключении клиента, сервер присваивает ему уникальный идентификатор (id) и возвращает его клиенту. Этот идентификатор сохраняется в объекте client, на стороне клиента и указывается при последующих подключениях к серверу. Таким образом сервер всегда знает какой конкретный клиент написал сообщение. После выполнения операции подключения и обмена идентификаторами, последующие сообщения сервер рассылает всем подключившимся клиентам с указанием того, какой клиент написал сообщение в общий чат. Основной обмен данными между клиентом и сервером осуществляется с помощью пересылки сериализованных JSON-объектов ( <https://learn.javascript.ru/json> )

**Листинг 3.** Сервер - файл main.js

/\* Загружаем модули \*/

const ws = require('ws');

// Обьект для хранения подключённых клиентов

var clients = {};

// Запускаем WebSocket-сервер на порту 3000

var webSocketServer = new ws.Server({

port: 3000

});

// Добавляем в сервер слушателя на событие подключения

webSocketServer.on('connection', function(ws) {

var id = Math.random();

clients[id] = ws; //Каждому клиенту присваиваем ссылку на объект сервера

console.log("новое соединение " + id);

//Отправляем клиенту id соединения, чтобы потом индивидуально отправлять сообщения

clients[id].send(JSON.stringify({'id\_first':id}));

//Обработка входящих сообщений

ws.on('message', function(message) {

console.log('получено сообщение ' + message);

let p\_ms = JSON.parse(message);

let answer = {

from\_id: "",

from\_name: "",

message: ""

};

if( p\_ms ) {

if( p\_ms.first\_name !== undefined ) {

/\*

\*Если это первое подключение и клиент прислал свое имя для чата

\*Проверяем имена других клиентов и изменяем исходное так, чтобы не повторялось

\*(пока не сделано)

\*/

//Полученное имя передаем обратно конкретному клиенту

answer.name\_first = p\_ms.first\_name;

clients[p\_ms.id].send(JSON.stringify(answer));

} else {

answer.message = p\_ms.message;

answer.from\_id = p\_ms.connect.id;

answer.from\_name = p\_ms.connect.name;

//Рассылка сообщений всем клиентам

for (var key in clients) {

//Отправляем сообщение каждому клиенту

clients[key].send(JSON.stringify(answer));

}

}

}

});

});

**Листинг 4.** Клиент - файл ws\_test.htm   
В заголовке head файла содержится импорт инструментов фреймворка bootstrap и библиотеки jQuery.

<!doctype html>

<head>

<meta name='viewport' content='width=device-width, initial-scale=1'>

<link rel="stylesheet" href="/css/bootstrap.min.css">

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://unpkg.com/popper.js@1.16.1/dist/umd/popper.js"></script>

<script src="/js/bootstrap.min.js"></script>

<style>

body{ padding: 25px; }

#messages, #log { width:100%; height:200px; border:1px #787878 solid; padding:5px; overflow:auto; }

#messages { height: 400px; }

</style>

</head>

<body>

<script>

var socket; //Переменная, хранящая сокет подключения

var client = { //Переменная, хранящая свойства клиента

connect: {

id: "",

name: ""

},

message: ""

};

//Устанавливаем коннект с сервером и определяем реакцию на события

function SetConnect() {

//Подключаемся к серверу

socket = new WebSocket("ws://localhost:3000");

//Метод onopen открывает соединение с сервером

socket.onopen = function(e) {

if( socket.OPEN == 1) {

toLog("[open] Соединение установлено");

//Блокируем кнопку установки соединения и закрываем блок с интерфейсом коннекта к серверу

$("#ConnectOpen").attr("disabled", true);

$("#ConnectBlock").css("display", "none");

//Разблокируем кнопку отправки сообщения и открываем блок с интерфейсом отправки сообщения

$("#BSubmit").attr("disabled", false);

$("#MessageBlock").css("display", "");

//Отправляем имя для отображения в списке чата

client.connect.name = $("#name").val();

//socket.send(JSON.stringify(client));

} else {

toLog("Соединение не установлено");

}

};

/\*

\* Метод onmessage определяет обработчик входящих сообщений, который

\* срабатывает каждый раз когда сервер присылает сообщение

\*/

socket.onmessage = function(event) {

toLog(`[message] Данные получены с сервера: ${event.data}`);

var dataJSON = JSON.parse(event.data);

if( dataJSON ) {

if( dataJSON.id\_first !== undefined ) {

//Если получено сообщение с идентификатором сервера

client.connect.id = dataJSON.id\_first;

socket.send(JSON.stringify({"id":client.connect.id, "first\_name":client.connect.name}));

}

if( dataJSON.name\_first !== undefined ) {

//Если получено сообщение с подтвержденным именем

client.connect.name = dataJSON.name\_first;

$("#MyName").html(client.connect.name);

showMessage(client.connect.name+": Вошел в чат");

}

if( dataJSON.message !== undefined && dataJSON.message !== "" ) {

showMessage(dataJSON.from\_name+": "+dataJSON.message);

}

}

};

/\*

\* Метод onclose срабатывает при потере коннекта с сервером

\*/

socket.onclose = function(event) {

if (event.wasClean) {

toLog(`[close] Соединение закрыто чисто, код=${event.code} причина=${event.reason}`);

} else {

toLog('[close] Соединение прервано по причине '+event.code, true);

}

//Блокируем кнопку отправки сообщения и открываем блок с интерфейсом отправки сообщения

$("#BSubmit").attr("disabled", true);

$("#MessageBlock").css("display", "none");

//Разблокируем кнопку установки соединения и открываем блок с интерфейсом коннекта к серверу

$("#ConnectOpen").attr("disabled", false);

$("#ConnectBlock").css("display", "");

};

socket.onerror = function(error) {

alert(`[error] ${error.message}`);

toLog(`[error] ${error.message}`);

};

}

//Отправляем сообщение на сервер

function SendMsg(f) {

client.message = f.message.value;

socket.send(JSON.stringify(client));

return false;

}

// показать сообщение в div#message

function showMessage(message) {

var messageElem = document.createElement('div');

messageElem.appendChild(document.createTextNode(message));

document.getElementById('messages').appendChild(messageElem);

}

// запись сообщения в лог#log

function toLog(msg, show) {

var messageElem = document.createElement('div');

let d = new Date();

//Форматируем дату и время

dt = ('0'+d.getDate()).slice(-2)+'.'+('0'+(d.getMonth()+1)).slice(-2)+'.'+d.getFullYear();

dt += " "+d.getHours()+":"+d.getMinutes()+":"+d.getSeconds()+":"+d.getMilliseconds();

if(show) { alert(msg); }

messageElem.appendChild(document.createTextNode(dt));

messageElem.appendChild(document.createElement('br'));

messageElem.appendChild(document.createTextNode(msg));

document.getElementById('log').appendChild(messageElem);

}

//Определяем код, срабатывающий после загрузки всех элементов страницы и построения DOM

window.onload = function() {

//Привязываем вызов функции SetConnect к событию клика по кнопке ConnectOpen

ConnectOpen.onclick = SetConnect;

document.forms.publish.onsubmit = function() {return SendMsg(this);};

};

</script>

<h1>Пример чата</h1>

<hr>

<form name="publish">

<div id="ConnectBlock">

Имя в чате: <input type="text" name="name" id="name"> <input id="ConnectOpen" type="button" value="Открыть соединение">

</div>

<div id="MessageBlock" style="display:none;">

Вы подключились как <b><span id="MyName"></span></b><br>

Сообщение: <input type="text" name="message" id="message" style="width:30%;">

<input type="submit" id="BSubmit" value="Отправить" disabled>

</div>

</form>

<!-- div с сообщениями -->

Сообщения

<div id="messages"></div>

<!-- log с сервера -->

Лог

<div id="log"></div>

</body>

</html>

**Задание на лабораторную работу**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Баллы | % |
| [1](https://learn.javascript.ru/websocket) | Запустить примеры применения AJAX и WebSocket  Изучить описание AJAX по адресу <https://javascript.ru/ajax/intro>  Изучить описание WebSocket по адресу <https://learn.javascript.ru/websocket> | 1 | 20 |
| 2 | Выполнить по собственному выбору одно из заданий:  С помощью AJAX реализовать подгрузку городов (3-5 шт) к списку стран (3-5 шт). Хранение стран и городов на сервере выполняется способом на выбор студента: статические файлы (xml, json, txt), динамические (js, php). В случае выбора файлов js можно взять за основу веб-сервер из Приложения 2.  Для WebSocket чата реализовать отключение конкретного клиента от чата с помощью кнопки в интерфейсе клиента  Для WebSocket чата реализовать алгоритм подбора уникального имени так, чтобы участники чата всегда писали с уникальными именами, даже если при подключении случайно ввели одинаковые. Если клиент не указал имя для чата, установить его автоматически.  Для WebSocket чата реализовать автоматическое отключение клиентов, в зависимости от заданного таймаута.  Для WebSocket чата реализовать хранение истории сообщений ограниченной по времени и количеству так, чтобы вновь подключенный клиент мог запросить историю сообщений за определенное время, например 15 минут, час или с начала первого коннекта к серверу чтобы быть в курсе обсуждения, которое велось до его подключения. | 3 | 60 |
| 3 | Ответить на контрольные вопросы | 1 | 20 |
|  |  | 5 | 100 |

**Контрольные вопросы:**

1. Как работает AJAX?
2. Что нужно иметь для работы с AJAX на клиенте и сервере?
3. Как локально развернуть окружение для разработки с использованием Ajax?
4. Как передать параметры в AJAX?
5. Что такое XMLHttpRequest?
6. Какие методы и свойства доступны для работы с XMLHttpRequest?
7. Как работает WebSocket?
8. Что нужно иметь для работы с WebSocket на клиенте и сервере?
9. Как локально развернуть окружение для разработки с использованием WebSocket?
10. Что такое Node.js?
11. Что такое npm и как использовать npm?
12. Какие события доступны для работы с WebSocket на клиенте?
13. Какие события доступны для работы с WebSocket на сервере?
14. Как передать параметры в WebSocket?
15. Расскажите о целесообразности применения Ajax
16. Расскажите о целесообразности применения WebSocket
17. Расскажите об устройстве протокола WebSocket

**Литература и ресурсы для обучения:**

1. XMLHttpRequest  
   <https://learn.javascript.ru/xmlhttprequest>
2. Введение в Ajax  
   <https://javascript.ru/ajax/intro>
3. Документация по методу ajax для jQuery, реализующему обмен по XMLHttpRequest  
   <https://api.jquery.com/jquery.ajax/>
4. Документация по Node.js  
   <https://nodejs.org/ru/docs/>
5. Учебник по Node.js (с информацией о npm)  
   <https://yadi.sk/i/0MeMmtWFtA8vig>
6. События в Node.js (метод on)  
   <https://xsltdev.ru/nodejs/tutorial/event-emitter/>
7. Скринкасты по Node.js

[https://learn.javascript.ru/screencast/nodejs#nodejs-modules-module](https://learn.javascript.ru/screencast/nodejs" \l "nodejs-modules-module)

1. Использование WebSocket API в веб-приложении (для NetBeance)  
   <https://netbeans.org/kb/docs/javaee/maven-websocketapi_ru.html>
2. Введение в WebSocket  
   <https://learn.javascript.ru/websocket>
3. Формат JSON  
   <https://learn.javascript.ru/json>
4. Пакетный менеджер npm  
   <http://prgssr.ru/development/vvedenie-v-paketnyj-menedzher-npm-dlya-nachinayushih.html>
5. Стандарт протокола WebSocket  
    <https://tools.ietf.org/html/rfc6455>

**Приложение 1**

# **Объекты и методы класса XMLHttpRequest**

|  |  |
| --- | --- |
| Метод | Описание |
| abort() | Отменяет текущий запрос, удаляет все заголовки, ставит текст ответа сервера в null. |
| getAllResponseHeaders() | Возвращает полный список HTTP-заголовков в виде строки. Заголовки разделяются знаками переноса (CR+LF). Если флаг ошибки равен true, возвращает пустую строку. Если статус 0 или 1, вызывает ошибку INVALID\_STATE\_ERR. |
| getResponseHeader(headerName) | Возвращает значение указанного заголовка. Если флаг ошибки равен true, возвращает null. Если заголовок не найден, возвращает null. Если статус 0 или 1, вызывает ошибку INVALID\_STATE\_ERR. |
| open(method, URL, async, userName, password) | Определяет метод, URL и другие опциональные параметры запроса; параметр async определяет, происходит ли работа в асинхронном режиме. Последние два параметра необязательны. |
| send(content) | Отправляет запрос на сервер. |
| setRequestHeader(label, value) | Добавляет HTTP-заголовок к запросу. |
| overrideMimeType(mimeType) | Позволяет указать mime-type документа, если сервер его не передал или передал неправильно. **Внимание**: метод отсутствует в Internet Explorer! |

|  |  |  |
| --- | --- | --- |
| Свойство | Тип | Описание |
| onreadystatechange | EventListener | Обработчик события, которое происходит при каждой смене состояния объекта. Имя должно быть записано в нижнем регистре. |
| readyState | unsigned short | Текущее состояние объекта (0 — не инициализирован, 1 — открыт, 2 — отправка данных, 3 — получение данных и 4 — данные загружены) |
| responseText | DOMString | Текст ответа на запрос. Если состояние не 3 или 4, возвращает пустую строку. |
| responseXML | Document | Текст ответа на запрос в виде XML, который затем может быть обработан посредством [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model). Если состояние не 4, возвращает null. |
| status | unsigned short | HTTP-статус в виде числа ([404](https://ru.wikipedia.org/wiki/HTTP_404) — «Not Found», [200](https://ru.wikipedia.org/wiki/HTTP_200) — «OK» и т. д.) |
| statusText | DOMString | Статус в виде строки («Not Found», «OK» и т. д.). Если статус не распознан, браузер пользователя должен вызвать ошибку INVALID\_STATE\_ERR. |

|  |  |  |
| --- | --- | --- |
| Название | Код | Описание |
| SECURITY\_ERR | 18 | Вызывается при попытке совершить запрос, запрещённый настройками безопасности в браузере пользователя. |
| NETWORK\_ERR | 101 | Вызывается при ошибке сети (во время синхронного запроса). |
| ABORT\_ERR | 102 | Вызывается при прерывании пользователем запроса (во время синхронного запроса). |

# Приложение 2.

Код http сервера на JavaScript для запуска под Node.js с ресурса <https://www.webdraftt.com/tutorial/nodejs/start>. Сохраните этот код как файл main.js и запустите сервер командой node main.js. В случае верного запуска в консоль будет выведена строка Server listens http://127.0.0.1:7000, им можно будет из браузера посылать на сервер запросы вида: http://localhost:7000?param1=value1&param2=value2 или http://localhost:7000/home?param1=value1&param2=value2 и т.д.

const http = require('http');

const host = '127.0.0.1';

const port = 7000;

function notFound(res){

res.statusCode = 404;

res.setHeader('Content-Type', 'text/plain');

res.end('Not found\n');

}

const server = http.createServer((req, res) => {

switch(req.method){

case 'GET': {

switch(req.url){

case '/home': {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Home page\n');

break;

}

case '/about': {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('About page\n');

break;

}

default: {

notFound(res);

break;

}

}

break;

}

case 'POST': {

switch(req.url){

case '/api/admin': {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Create admin request\n');

break;

}

case '/api/user': {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Create user request\n');

break;

}

default: {

notFound(res);

break;

}

}

break;

}

default: {

notFound(res);

break;

}

}

});

server.listen(port, host, () => {

console.log(`Server listens http://${host}:${port}`);

});