# Лабораторная работа № 11 Взаимодействие с сервером. Технология AJAX

**Цель работы:** изучить технологию AJAX, научиться получать данные, используя REST API.

**Краткие теоретические сведения:**

Ajax представляет технологию для отправки запросов к серверу из клиентского кода JavaScript без перезагрузки страницы. Сам термин расшифровывается как Asynchronous JavaScript And XML. То есть изначально AJAX предполагал асинхронное взаимодействие клиента и сервера посредством данных в формате XML. В настоящее время основным форматом данных для обмена данных между клиентом и сервером явлется формат JSON.

**Объект XMLHttpRequest**

Для создания приложений, использующих Ajax, применяются различные способы. Но самым распространенным способом является использование объекта XMLHttpRequest:

const request = new XMLHttpRequest();

После создания объекта XMLHttpRequest можно отправлять запросы к серверу. Но для начала надо вызвать метод open() для инициализации:

request.open("GET", "http://localhost/test.txt", false);

Метод open() принимает три параметра: тип запроса (GET, POST, HEAD, PUT), адрес запроса и третий необязательный параметр – логическое значение true или false, указывающее, будет ли запрос осуществляться в асинхронном режиме. По умолчанию, если третий параметр не используется, то запрос отправляется в асинхронном режиме, что позволяет параллельно с выполнением запроса выполнять также и другой код JavaScript.

Кроме того, метод open() может принимать еще два параметра: логин и пароль пользователя, если для выполнения запроса нужна аутентификация:

request.open("GET", "http://localhost/home.php", true, "login", "password");

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

request.send();

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

* **status**: содержит статусный код ответа HTTP, который пришел от сервера. С помощью статусного кода можно судить об успешности запроса или об ошибках, которые могли бы возникнуть при его выполнении. Например, статусный код 200 указывает на то, что запрос прошел успешно. Код 403 говорит о необходимости авторизации для выполнения запроса, а код 404 сообщает, что ресурс не найден и так далее.
* **statusText**: возвращает текст статуса ответа, например, "200 OK"
* **responseType**: возвращает тип ответа.
* **response**: возвращает ответ сервера.
* **responseText**: возвращает текст ответа сервера.
* **responseXML**: возвращает xml, если ответ от сервера в формате xml.

При асинхронном запросе объект XMLHttpRequest использует свойство **readyState** для хранения состояния запроса. Состояние запроса представляет собой число:

0: объект XMLHttpRequest создан, но метод open() еще не был вызван для инициализации объекта

* 1: метод open() был вызван, но запрос еще не был отправлен методом send()
* 2: запрос был отправлен, заголовки и статус ответа получены и готовы к использованию
* 3: ответ получен от сервера
* 4: выполнение запроса полностью завершено (даже если получен код ошибки, например, 404)

Событие readystatechange возникает каждый раз, когда изменяется значение свойства readyState.

**Отправка запросов**

GET-запрос характеризуется тем, что данные могут отправляться в строке запроса:

<script>

// объект для отправки

var user = {

name: "Alex",

age: 21

};

const request = new XMLHttpRequest();

function reqReadyStateChange() {

if (request.readyState == 4) {

var status = request.status;

if (status == 200) {

document.getElementById("output").innerHTML=request.responseText;

}

}

}

// строка с параметрами для отправки

var body = "name=" + user.name + "&age="+user.age;

request.open("GET", "http://localhost:8080/postdata.php?"+body);

request.onreadystatechange = reqReadyStateChange;

request.send();

</script>

Для отправки берем свойства объекта user и формируем из их значений сроку с параметрами: "name=" + user.name + "&age=" + user.age. Затем эта строка добавляется к строке запроса в методе open("GET", "http://localhost:8080/postdata.php?"+body)

Для отправки данных методом POST надо установить заголовок Content-Type с помощью метода **setRequestHeader()**. В данном случае заголовок имеет значение application/x-www-form-urlencoded. Данные встраиваются в тело запроса при посылке запроса.

var user = {

name: "Tom",

age: 23

};

const request = new XMLHttpRequest();

function reqReadyStateChange() {

if (request.readyState == 4 && request.status == 200)

document.getElementById("output").innerHTML=request.responseText;

}

let body = "name=" + user.name + "&age="+user.age;

request.open("POST", "http://localhost:8080/postdata.php");

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

request.onreadystatechange = reqReadyStateChange;

request.send(body);

**Promise в Ajax-запросах**

Для более удобного способа организации асинхронного кода в современном JavaScript используется объект Promise, который обертывает асинхронную операцию в один объект и позволяет определить действия, выполняющиеся при успешном или неудачном выполнении этой операции.

Promise – это специальный объект, который содержит своё состояние. Вначале pending («ожидание»), затем – одно из: fulfilled («выполнено успешно») или rejected («выполнено с ошибкой»).

Синтаксис создания Promise:

const promise = new Promise(function(resolve, reject) {

// Эта функция будет вызвана автоматически

// В ней можно делать любые асинхронные операции,

// А когда они завершатся — нужно вызвать одно из:

// resolve(результат) при успешном выполнении

// reject(ошибка) при ошибке

})

Для обработки результата объекта Promise вызывается метод then(), который принимает два параметра: функцию, вызываемую при успешном выполнении запроса, и функцию, которая вызывается при неудачном выполнении запроса. Метод then() также возвращает объект Promise. Поэтому при необходимости мы можем применить к его результату цепочки вызовов метода then: promise.then().then().

Рассмотрим пример использования промиса с использованием промисификация – это когда берут асинхронную функциональность и делают для неё обёртку, возвращающую промис.

После промисификации использование функциональности зачастую становится гораздо удобнее. Например:

function httpGet(url) {

return new Promise(function(resolve, reject) {

var xhr = new XMLHttpRequest();

xhr.open('GET', url, true);

xhr.onload = function() {

if (this.status == 200) {

resolve(this.response);

} else {

var error = new Error(this.statusText);

error.code = this.status;

reject(error);

}

};

xhr.onerror = function() {

reject(new Error("Network Error"));

};

xhr.send();

});

}

httpGet("/article/promise/user.json")

.then(

response => alert(`Fulfilled: ${response}`),

error => alert(`Rejected: ${error}`)

);

**Метод Fetch**

Современные браузеры поддерживает метод **fetch** – новый встроенный метод для AJAX-запросов, призванный заменить XMLHttpRequest. Он гораздо мощнее, чем приведенная выше функция httpGet.

Типичный запрос с помощью fetch состоит из двух операторов await:

let response = await fetch(url, options); // завершается с заголовками ответа

let result = await response.json(); // читать тело ответа в формате JSON

Или, без await:

fetch(url, options)

.then(response => response.json())

.then(result => /\* обрабатываем результат \*/)

Метод fetch следующие опции для формирования запроса:

* **method** – HTTP-метод;
* **headers** – объект с запрашиваемыми заголовками;
* **body** – данные для отправки (тело запроса) в виде текста, FormData, BufferSource, Blob или UrlSearchParams.

Для получения тела ответа используются следующие методы:

* **response.text()** – возвращает ответ как обычный текст;
* **response.json()** – преобразовывает ответ в JSON-объект;
* **response.formData()** – возвращает ответ как объект FormData (кодировка form/multipart, см. следующую главу);
* **response.blob()** – возвращает объект как Blob (бинарные данные с типом);
* **response.arrayBuffer()** – возвращает ответ как ArrayBuffer (низкоуровневые бинарные данные).

**Практическая часть:**

**Практическая часть:**

**Задание 1.** Написать скрипт, в котором предусмотрена возможность получения данных с сайта Национального банка Республики Беларусь***,*** используя службу **Web API (**документация **-**<http://www.nbrb.by/APIHelp/ExRates>***).***

На странице предусмотреть получение валюты страны со своим вариантом. Информацию отобразить в виде таблицы, указав флаг страны, название страны, название валюты по-английски и по-белорусски, курс валюты к белорусскому рублю на текущую дату. Для получения данных использовать объект **XMLHttpRequest*.***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Номер варианта** | **Страна** | **Номер варианта** | **Страна** | **Номер варианта** | **Страна** |
| 1 | Аргентина | 11 | Ирландия | 21 | Турция |
| 2 | Алжир | 12 | Израиль | 22 | Великобритания |
| 3 | Канада | 13 | Южная Корея | 23 | Россия |
| 4 | Чили | 14 | Мексика | 24 | Таджикистан |
| 5 | Китай | 15 | Монголия | 25 | Армения |
| 6 | Дания | 16 | Польша | 26 | Казахстан |
| 7 | Финляндия | 17 | Афганистан | 27 | Азербайджан |
| 8 | Грузия | 18 | Швеция | 28 | Малайзия |
| 9 | Гонконг | 19 | Украина | 29 | Индия |
| 10 | Исландия | 20 | Сингапур | 30 | Норвегия |

**Задание 2.** Получить данные для отображения динамики курса валюты за указанный период времени. Период времени задается в полях ввода. Информацию отобразить в виде таблицы. Для получения данных использовать **промисификацию*.***

**Задание 3.** Отобразить в виде таблицы курс конвертации валюты страны в соответствии с вариантом к валютам стран предыдущего и последующего вариантов. Информацию отобразить в виде таблицы. Для получения данных использовать метод **fetch** и специальный синтаксис для работы с промисами **«async/await».**

**Контрольные вопросы для защиты:**

1. Как осуществляется обмен данными между клиентом и сервером?
2. Что такое технология AJAX?
3. Что такое REST API?
4. Опишите свойства объекта XMLHttpRequest?
5. Как получить данные с сервера, используя GET запрос?
6. Как получить данные с сервера, используя POST запрос?
7. Что такое промис?
8. Основные действия при работе с промисами?
9. Как обрабатываются ошибки при использовании промисов?
10. Для чего предназначен метод fetch?