Главное управление образования Гомельского облисполкома
Учреждение образования
«Гомельский государственный машиностроительный колледж»

Учебная дисциплина
«Веб-программирование для мобильных устройств»

Инструкция
по выполнению лабораторной работы №19
«Обработка данных в формате JSON на стороне клиента»

Гомель

2021

Составитель: _____ Гаврилова В.М.

Обсуждено и одобрено на заседании цикловой комиссии «Программируемые мобильные системы»

Протокол от «27» августа 2021 № 1

_____ В.М. Гаврилова

Лабораторная работа №19

Тема работы: «Обработка данных в формате JSON на стороне клиента»

1. Цель работы

Обучиться обработке данных в формате JSON на стороне клиента.

2. Задание

Выполнить обработку данных в формате JSON на стороне клиента используя данные с интернет-ресурса jsonplaceholder.typicode.com в соответствии с вариантом:
Вариант 1, 7 /posts
Вариант 2, 8 /comments
Вариант 3, 9 /albums
Вариант 4, 10 /photos
Вариант 5, 11 /todos
Вариант 6, 12 /users

3. Оснащение работы

Технические средства обучения:

- IBM – совместимый компьютер;

Электронные средства обучения:

- веб-браузер (Opera, Google Chrome, Mozilla Firefox и др.);
- html-редакторы (Notepad++, Sublime Text и др.).

4. Основные теоретические сведения

AJAX (аббревиатура от «Asynchronous Javascript And Xml») – технология обращения к серверу без перезагрузки страницы.

Технически, с помощью AJAX можно обмениваться любыми данными с сервером.

Обычно используются форматы:

JSON – для отправки и получения структурированных данных, объектов.

XML – если сервер почему-то работает в формате XML, то можно использовать и его, есть средства.

HTML/текст – можно и просто загрузить с сервера код HTML или текст для показа на странице.

Бинарные данные, файлы – гораздо реже, в современных браузерах есть удобные средства для них.

JSON (JavaScript Object Notation) – это общий формат для представления значений и объектов. Его описание задокументировано в стандарте RFC 4627. JSON легко использовать для обмена данными, когда клиент использует JavaScript, а сервер написан на Ruby/PHP/Java или любом другом языке.

JavaScript предоставляет методы:

JSON.stringify для преобразования объектов в JSON;

JSON.parse для преобразования JSON обратно в объект.

Пример. Данный метод преобразует объект в строку.

```
let student = {
  name: 'John',
  age: 30,
  isAdmin: false,
  courses: ['html', 'css', 'js'],
  wife: null
};

let json = JSON.stringify(student);

alert(typeof json); // мы получили строку!

alert(json);
/* выведет объект в формате JSON:
{
  "name": "John",
  "age": 30,
  "isAdmin": false,
  "courses": ["html", "css", "js"],
  "wife": null
}
*/
```

Полученная строка json называется JSON-форматированным или сериализованным объектом. Можно отправить его по сети или поместить в обычное хранилище данных.

Объект в формате JSON имеет несколько важных отличий от объектного литерала:

строки используют двойные кавычки. Никаких одинарных кавычек или обратных кавычек в JSON. Так 'John' становится "John";

имена свойств объекта также заключаются в двойные кавычки. Это обязательно. Так age:30 становится "age":30.

Метод JSON.stringify может быть применён к примитивам.

JSON поддерживает следующие типы данных:

Объекты { ... }

Массивы [...]

Примитивы:

строки,

числа,

логические значения true/false,

null.

Пример.

```
// число в JSON остаётся числом
alert( JSON.stringify(1) ) // 1

// строка в JSON по-прежнему остаётся строкой, но в двойных кавычках
alert( JSON.stringify('test') ) // "test"

alert( JSON.stringify(true) ); // true

alert( JSON.stringify([1, 2, 3]) ); // [1,2,3]
```

Полный синтаксис JSON.stringify:

value - значение для кодирования;

replacer - массив свойств для кодирования или функция соответствия function(key, value);

space - дополнительное пространство (отступы), используемое для форматирования.

В большинстве случаев JSON.stringify используется только с первым аргументом. Но если нам нужно настроить процесс замены, например, отфильтровать циклические ссылки, то можно использовать второй аргумент JSON.stringify.

Если мы передадим ему массив свойств, будут закодированы только эти свойства.

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: [{name: "John"}, {name: "Alice"}],
  place: room // meetup ссылается на room
};

room.occupiedBy = meetup; // room ссылается на meetup

alert( JSON.stringify(meetup, ['title', 'participants']) );
// {"title":"Conference","participants":[{},{}]}
```

В приведенном выше примере список свойств применяется ко всей структуре объекта. Так что внутри participants – пустые объекты, потому что name нет в списке.

Включим в список все свойства, кроме room.occupiedBy, из-за которого появляется цикличная ссылка:

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  participants: [{name: "John"}, {name: "Alice"}],
  place: room // meetup ссылается на room
};

room.occupiedBy = meetup; // room ссылается на meetup

alert( JSON.stringify(meetup, ['title', 'participants', 'place', 'name', 'number']) );
/*
{
  "title":"Conference",
  "participants":[{"name":"John"},{"name":"Alice"}],
  "place":{"number":23}
}
*/
```

Третий аргумент в JSON.stringify(value, replacer, space) – это количество пробелов, используемых для удобного форматирования.

Ранее все JSON-форматированные объекты не имели отступов и лишних пробелов. Это нормально, если мы хотим отправить объект по сети. Аргумент space используется исключительно для вывода в удобочитаемом виде.

Пример. space = 2 указывает JavaScript отображать вложенные объекты в несколько строк с отступом в 2 пробела внутри объекта

```
let user = {
  name: "John",
  age: 25,
  roles: {
    isAdmin: false,
    isEditor: true
  }
};

alert(JSON.stringify(user, null, 2));
/* отступ в 2 пробела:
{
  "name": "John",
  "age": 25,
  "roles": {
    "isAdmin": false,
    "isEditor": true
  }
}
*/
```

Метод JSON.parse используется, чтобы декодировать JSON-строку.

Синтаксис:

```
let value = JSON.parse(str, [reviver]);
```

str - JSON для преобразования в объект;

reviver - необязательная функция, которая будет вызываться для каждой пары (ключ, значение) и может преобразовывать значение.

XMLHttpRequest – это встроенный в браузер объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы.

На сегодняшний день не обязательно использовать XMLHttpRequest, так как существует другой, более современный метод fetch.

В современной веб-разработке XMLHttpRequest используется по трём причинам:

по историческим причинам: существует много кода, использующего XMLHttpRequest, который нужно поддерживать;

необходимость поддерживать старые браузеры и нежелание использовать полифилы (например, чтобы уменьшить количество кода);

потребность в функциональности, которую fetch пока что не может предоставить, к примеру, отслеживание прогресса отправки на сервер.

XMLHttpRequest имеет два режима работы: синхронный и асинхронный.

Асинхронный режим используется в большинстве случаев.

Чтобы сделать запрос, нужно выполнить три шага:

1 создать XMLHttpRequest.

```
let xhr = new XMLHttpRequest(); // у конструктора нет аргументов
```

2. Инициализировать его.

```
xhr.open(method, URL, [async, user, password])
```

Этот метод обычно вызывается сразу после new XMLHttpRequest. В него передаются основные параметры запроса:

method – HTTP-метод. Обычно это "GET" или "POST".

URL – URL, куда отправляется запрос: строка, может быть и объект URL.

async – если указать false, тогда запрос будет выполнен синхронно, это мы рассмотрим чуть позже.

user, password – логин и пароль для базовой HTTP-авторизации (если требуется).

3. Послать запрос.

```
xhr.send([body])
```

Этот метод устанавливает соединение и отсылает запрос к серверу. Необязательный параметр body содержит тело запроса.

Некоторые типы запросов, такие как GET, не имеют тела. А некоторые, как, например, POST, используют body, чтобы отправлять данные на сервер.

4. Слушать события на xhr, чтобы получить ответ.

Три наиболее используемых события:

load – происходит, когда получен какой-либо ответ, включая ответы с HTTP-ошибкой, например 404.

error – когда запрос не может быть выполнен, например, нет соединения или невалидный URL.

progress – происходит периодически во время загрузки ответа, сообщает о прогрессе.

```
xhr.onload = function() {
  alert(`Загружено: ${xhr.status} ${xhr.response}`);
};

xhr.onerror = function() { // происходит, только когда запрос совсем не получилось выпол
  alert(`Ошибка соединения`);
};

xhr.onprogress = function(event) { // запускается периодически
  // event.loaded - количество загруженных байт
  // event.lengthComputable = равно true, если сервер присылает заголовок Content-Length
  // event.total - количество байт всего (только если lengthComputable равно true)
  alert(`Загружено ${event.loaded} из ${event.total}`);
};
```

5. Порядок выполнения работы

1. Создайте сценарий в соответствии с заданием.
2. Подключите сценарий к html-файлу.
3. Просмотрите с помощью web-браузера созданный документ.

6. Форма отчета о работе

Номер учебной группы _____
Фамилия, инициалы обучающегося _____
Дата выполнения работы _____
Тема работы: _____
Цель работы: _____
Задание: _____
Оснащение работы: _____
Результат выполнения работы: _____

7. Контрольные вопросы и задания

1. Дайте понятие AJAX?
2. Дайте понятие JSON?
3. Дайте понятие XMLHttpRequest

Рекомендуемая литература

1 Васильев, А.Н. JavaScript в примерах и задачах / А.Н. Васильев. – М.: Эксмо, 2018.

2 Вахтуров, В.В. JavaScript. Освой на примерах / В.В. Вахтуров. – СПб.:БХВ-Петербург, 2007.

3 Макфарланд, Д. JavaScript и jQuery. Исчерпывающее руководство / Д. Макфарланд. - М.: Эксмо, 2012.

4 Интернет-ресурс https://learn.javascript.ru/