Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 19**

**Инструкционно-технологическая карта**

Тема: JSON формат. Структура JSON-документа. Создание и применение.

Цель: Научиться использовать формат передачи данных JSON. Изучить структуру JSON документа, научиться создавать и применять JSON формат.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Краткие теоретические сведения

## Формат JSON, метод toJSON

Например, есть сложный объект, и нужно бы преобразовать его в строку, чтобы отправить по сети или просто вывести для логирования.

Естественно, такая строка должна включать в себя все важные свойства.

Можно бы реализовать преобразование следующим образом:

let user = {

    name: "John",

    age: 30,

    toString() {

        return `{name: "${this.name}", age: ${this.age}}`;

    }

};

alert(user); // {name: "John", age: 30}

Но в процессе разработки добавляются новые свойства, старые свойства переименовываются и удаляются. Обновление такого **toString** каждый раз может стать проблемой. Можно попытаться перебрать свойства в нём, но что, если объект сложный, и в его свойствах имеются вложенные объекты? Нужно осуществить их преобразование тоже.

К счастью, нет необходимости писать код для обработки всего этого. У задачи есть простое решение.

## JSON.stringify

JSON (JavaScript Object Notation) – это общий формат для представления значений и объектов. Его описание задокументировано в стандарте RFC 4627. Первоначально он был создан для JavaScript, но многие другие языки также имеют библиотеки, которые могут работать с ним. Таким образом, 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.

Например, здесь мы преобразуем через JSON.stringify данные студента:

let student = {

    name: 'John',

    age: 30,

    isAdmin: false,

    courses: ['html', 'css', 'js'],

    wife: null

};

let json = JSON.stringify(student);

alert(typeof json); // получили строку

alert(json);

  /\* выведет объект в формате JSON:

{

  "name": "John",

  "age": 30,

  "isAdmin": false,

  "courses": ["html", "css", "js"],

  "wife": null

}

\*/

Метод **JSON.stringify(student)** берёт объект и преобразует его в строку.

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

Следует обратить внимание, что 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.
* JSON.stringify может быть применён и к примитивам.

JSON поддерживает следующие типы данных:

* Объекты { ... }
* Массивы [ ... ]
* Примитивы: строки, числа, логические значения true/false, null.

Например:

// 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]

JSON является независимой от языка спецификацией для данных, поэтому **JSON.stringify** пропускает некоторые специфические свойства объектов JavaScript.

А именно:

* Свойства-функции (методы).
* Символьные свойства.
* Свойства, содержащие undefined.

let user = {

    sayHi() { // будет пропущено

        alert("Hello");

    },

    [Symbol("id")]: 123, // также будет пропущено

    something: undefined // как и это - пропущено

};

alert(JSON.stringify(user)); // {} (пустой объект)

Обычно это нормально. Если это не то, что устраивает, то скоро можно настроить этот процесс.

Самое замечательное, что вложенные объекты поддерживаются и конвертируются автоматически.

Например:

let meetup = {

    title: "Conference",

    room: {

        number: 23,

        participants: ["john", "ann"]

    }

};

alert(JSON.stringify(meetup));

  /\* вся структура преобразована в строку:

{

  "title":"Conference",

  "room":{"number":23,"participants":["john","ann"]},

}\*/

Важное ограничение: не должно быть циклических ссылок.

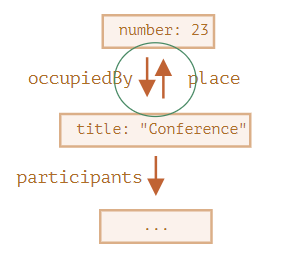
Например:

let room = {

    number: 23

};

let meetup = {

    title: "Conference",

    participants: ["john", "ann"]

};

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

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

JSON.stringify(meetup); // Ошибка: Преобразование цикличной структуры в JSON

Здесь преобразование завершается неудачно из-за циклической ссылки: **room.occupiedBy** ссылается на **meetup**, и **meetup.place** ссылается на **room**.

## Исключаем и преобразуем: replacer

Полный синтаксис **JSON.stringify**:

let json = JSON.stringify(value[, replacer, space])

**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 передадать ему массив свойств, будут закодированы только эти свойства.

Например:

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":[{},{}]}

Здесь, наверное, слишком строги. 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}

}

\*/

Теперь всё, кроме **occupiedBy**, **сериализовано**. Но список свойств довольно длинный.

К счастью, в качестве **replacer** можно использовать функцию, а не массив.

Функция будет вызываться для каждой пары (**key**, **value**), и она должна возвращать заменённое значение, которое будет использоваться вместо исходного. Или **undefined**, чтобы пропустить значение.

В нашем случае следует вернуть **value** «как есть» для всего, кроме **occupiedBy**. Чтобы игнорировать **occupiedBy**, код ниже возвращает **undefined**:

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, function replacer(key, value) {

    alert(`${key}: ${value}`);

    return (key == 'occupiedBy') ? undefined : value;

}));

/\* пары ключ:значение, которые приходят в replacer:

:             [object Object]

title:        Conference

participants: [object Object],[object Object]

0:            [object Object]

name:         John

1:            [object Object]

name:         Alice

place:        [object Object]

number:       23

\*/

Следует обратить внимание, что функция **replacer** получает каждую пару ключ/значение, включая вложенные объекты и элементы массива. И она применяется рекурсивно. Значение **this** внутри **replacer** – это объект, который содержит текущее свойство.

Первый вызов – особенный. Ему передаётся специальный «объект-обёртка»: **{"": meetup}**. Другими словами, первая (**key**, **value**) пара имеет пустой ключ, а значением является целевой объект в общем. Вот почему первая строка из примера выше будет **":[object Object]"**.

Идея состоит в том, чтобы дать как можно больше возможностей **replacer** – у него есть возможность проанализировать и заменить/пропустить даже весь объект целиком, если это необходимо.

## Форматирование: space

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 необходимо отправить объект по сети. Аргумент space используется исключительно для вывода в удобочитаемом виде.

Ниже **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

  }

}

\*/

/\* для JSON.stringify(user, null, 4) результат содержит больше отступов:

{

    "name": "John",

    "age": 25,

    "roles": {

        "isAdmin": false,

        "isEditor": true

    }

}\*/

Параметр **space** применяется для логирования и красивого вывода.

## Пользовательский «toJSON»

Как и **toString** для преобразования строк, объект может предоставлять метод **toJSON** для преобразования в **JSON**. **JSON.stringify** автоматически вызывает его, если он есть.

Например:

let room = {

    number: 23

};

let meetup = {

    title: "Conference",

    date: new Date(Date.UTC(2017, 0, 1)),

    room

};

alert(JSON.stringify(meetup));

  /\*

  {

    "title":"Conference",

    "date":"2017-01-01T00:00:00.000Z",  // (1)

    "room": {"number":23}               // (2)

  }

\*/

Видно, что **date (1)** стал строкой. Это потому, что все объекты типа **Date** имеют встроенный метод **toJSON**, который возвращает такую строку.

Теперь нужно добавить собственную реализацию метода **toJSON** в наш объект **room (2)**:

let room = {

    number: 23,

    toJSON() {

        return this.number;

    }

};

let meetup = {

    title: "Conference",

    room

};

alert(JSON.stringify(room)); // 23

alert(JSON.stringify(meetup));

  /\*

  {

    "title":"Conference",

    "room": 23

  }

\*/

Видно, что **toJSON** используется как при прямом вызове **JSON.stringify(room)**, так и когда **room** вложен в другой **сериализуемый** объект.

## JSON.parse

Чтобы декодировать JSON-строку, нужен другой метод с именем **JSON.parse**.

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.

Например:

// строковый массив

let numbers = "[0, 1, 2, 3]";

numbers = JSON.parse(numbers);

alert( numbers[1] ); // 1

Или для вложенных объектов:

let user = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';

user = JSON.parse(user);

alert( user.friends[1] ); // 1

JSON может быть настолько сложным, насколько это необходимо, объекты и массивы могут включать другие объекты и массивы. Но они должны быть в том же JSON-формате.

Вот типичные ошибки в написанном от руки JSON (иногда приходится писать его для отладки):

let json = `{

    name: "John",                     // Ошибка: имя свойства без кавычек

    "surname": 'Smith',               // Ошибка: одинарные кавычки в значении (должны быть двойными)

    'isAdmin': false                  // Ошибка: одинарные кавычки в ключе (должны быть двойными)

    "birthday": new Date(2000, 2, 3), // Ошибка: не допускается конструктор "new", только значения.

    "friends": [0,1,2,3]                     // Здесь всё в порядке

  }`;

Кроме того, JSON не поддерживает комментарии. Добавление комментария в JSON делает его недействительным.

Существует ещё один формат JSON5, который поддерживает ключи без кавычек, комментарии и т.д. Но это самостоятельная библиотека, а не спецификация языка.

Обычный JSON настолько строг не потому, что его разработчики ленивы, а потому, что позволяет легко, надёжно и очень быстро реализовывать алгоритм кодирования и чтения.

## Использование reviver

Представим, что получен объект **meetup** с сервера в виде строки данных.

Вот такой:

// title: (meetup title), date: (meetup date)

let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';

…А теперь нам нужно **десериализовать** её, т.е. снова превратить в объект **JavaScript**.

Сделаем это, вызвав **JSON.parse**:

let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';

let meetup = JSON.parse(str);

alert( meetup.date.getDate() ); // Error!

Oшибка!

Значением **meetup.date** является строка, а не **Date объект**. Как **JSON.parse** мог знать, что он должен был преобразовать эту строку в Date?

Передадим **JSON.parse** функцию восстановления вторым аргументом, которая возвращает все значения «как есть», но date станет **Date**:

let str = '{"title":"Conference","date":"2017-11-30T12:00:00.000Z"}';

let meetup = JSON.parse(str, function(key, value) {

  if (key == 'date') return new Date(value);

  return value;

});

alert( meetup.date.getDate() ); // 30 - теперь работает!

Это работает и для вложенных объектов:

let schedule = `{

    "meetups": [

      {"title":"Conference","date":"2017-11-30T12:00:00.000Z"},

      {"title":"Birthday","date":"2017-04-18T12:00:00.000Z"}

    ]

  }`;

schedule = JSON.parse(schedule, function (key, value) {

    if (key == 'date') return new Date(value);

    return value;

});

alert(schedule.meetups[1].date.getDate()); // 18 - отлично!

## Итого

JSON – это формат данных, который имеет собственный независимый стандарт и библиотеки для большинства языков программирования.

JSON поддерживает простые объекты, массивы, строки, числа, логические значения и **null**.

JavaScript предоставляет методы **JSON.stringify** для **сериализации** в JSON и **JSON.parse** для чтения из JSON.

Оба метода поддерживают функции преобразования для интеллектуального чтения/записи.

Если объект имеет метод **toJSON**, то он вызывается через **JSON.stringify**.

# Контрольные вопросы

1. Что такое JSON?
2. Какие объекты поддерживает JSON?
3. Какие методы существуют в JavaScript для сереализации и десериализации JSON?

# Задания

1. Преобразовать объект в JSON, а затем обратно в обычный объект.

Преобразовать **user** в JSON, затем прочитать этот JSON в другую переменную.

let user = {

    name: "Василий Иванович",

    age: 35

};

1. Исключить обратные ссылки

В простых случаях циклических ссылок можно исключить свойство, из-за которого они возникают, из сериализации по его имени.

Но иногда нельзя использовать имя, так как могут быть и другие, нужные, свойства с этим именем во вложенных объектах. Поэтому можно проверять свойство по значению.

Написать функцию replacer для JSON-преобразования, которая удалит свойства, ссылающиеся на **meetup**:

let room = {

    number: 23

};

let meetup = {

    title: "Совещание",

    occupiedBy: [{ name: "Иванов" }, { name: "Петров" }],

    place: room

};

// цикличные ссылки

room.occupiedBy = meetup;

meetup.self = meetup;

alert(JSON.stringify(meetup, function replacer(key, value) {

    /\* ваш код \*/

}));

/\* в результате должно быть:

{

  "title":"Совещание",

  "occupiedBy":[{"name":"Иванов"},{"name":"Петров"}],

  "place":{"number":23}

}\*/

1. Массив объектов, созданный в предыдущей лабораторной работе сиреализовать и десиреализовать при помощи JSON.

# Литература

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич