[JSON](https://ru.wikipedia.org/wiki/JSON) (JavaScript Object Notation) – это общий формат для представления значений и объектов. Его описание задокументировано в стандарте [RFC 4627](https://tools.ietf.org/html/rfc4627). 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); // мы 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. Мы можем отправить его по сети или поместить в обычное хранилище данных.

Обратите 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 ключи и значения.
* 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, чего мы хотим, то скоро мы увидим, как можно настроить этот процесс.

вложенные объекты поддерживаются и конвертируются автоматически.

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](https://learn.javascript.ru/json" \l "isklyuchaem-i-preobrazuem-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.

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.

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 мы, наверное, слишком строги. Список свойств применяется ко всей структуре объекта. Так что внутри participants – пустые объекты, потому что name нет в списке.

Давайте 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 мы можем 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 мы можем 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;

}));

Обратите 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.

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](https://learn.javascript.ru/json" \l "formatirovanie-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 мы хотим отправить объект по сети. Аргумент space используется исключительно для вывода в удобочитаемом виде.

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

alert(JSON.stringify(user, null, 2));

Третьим аргументом также может быть строка. В этом случае строка будет использоваться для отступа вместо ряда пробелов.

Параметр space применяется исключительно для логирования и красивого вывода.

## [Пользовательский «toJSON»](https://learn.javascript.ru/json" \l "polzovatelskiy-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 meetup = {

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

};

alert( JSON.stringify(meetup) );

/\*

{

"date":"2017-01-01T00:00:00.000Z",

}

\*/

Как видим, date стал строкой. Это потому, что все объекты типа Date имеют встроенный метод toJSON, который возвращает такую строку.

Например добавим собственную реализацию метода toJSON в объект room:

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

}

\*/

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

**[JSON.parse](https://learn.javascript.ru/json" \l "json-parse)**

Чтобы декодировать JSON-строку, нам нужен другой метод с именем [JSON.parse](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/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.

// строковый массив

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

numbers = JSON.parse(numbers);

alert( numbers[1] ); // 1

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

user = JSON.parse(user);

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

 JSON не поддерживает комментарии. Добавление комментария в JSON делает его недействительным.

Существует ещё один формат [JSON5](https://json5.org/), 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](https://learn.javascript.ru/json" \l "ispolzovanie-reviver)

Например мы получили объект meetup с сервера в виде строки данных.

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

Нужно *десериализовать* её, т.е. снова превратить в объект JavaScript.

Вызовем JSON.parse:

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

let meetup = JSON.parse(str);

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

Ой, ошибка!

Значением meetup.date является строка, а не Date объект.

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](https://learn.javascript.ru/json" \l "itogo)

* 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](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify) для сериализации в JSON и [JSON.parse](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/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.