目 录

[1 JSON 教程 2](#_Toc504570403)

[1.1 JSON 简介 3](#_Toc504570404)

[1.1.1 什么是 JSON 3](#_Toc504570405)

[1.1.2 JSON 的特点 3](#_Toc504570406)

[1.2 JSON 语法 5](#_Toc504570407)

[1.2.1 JSON 键/值对 5](#_Toc504570408)

[1.2.2 JSON 字符串 5](#_Toc504570409)

[1.2.3 JSON 数值 6](#_Toc504570410)

[1.2.4 JSON 对象 7](#_Toc504570411)

[1.2.5 JSON 数组 7](#_Toc504570412)

[1.2.6 JSON 布尔值 8](#_Toc504570413)

[1.2.7 JSON null 9](#_Toc504570414)

[1.2.8 JSON 文件 9](#_Toc504570415)

[1.3 JSON 对象 9](#_Toc504570416)

[1.3.1 访问对象的值 9](#_Toc504570417)

[1.3.2 修改对象的值 10](#_Toc504570418)

[1.3.3 删除对象属性 10](#_Toc504570419)

[1.4 JSON 数组 10](#_Toc504570420)

[1.4.1 遍历数组 10](#_Toc504570421)

[1.4.2 修改数组的值 11](#_Toc504570422)

[1.4.3 删除数组元素 11](#_Toc504570423)

[1.5 JSON 和 JS 对象互转 12](#_Toc504570424)

[1.5.1 JSON.parse() 12](#_Toc504570425)

[1.5.2 JSON.stringify() 14](#_Toc504570426)

[2 参考资料 19](#_Toc504570427)

# JSON 教程

JSON是JavaScript Object Notation的简称，中文含义为“JavaScript 对象表示法”。JSON 是存储和交换文本信息的一种语法，它类似于XML，却比 XML 更小、更快、更易解析。

本章内容：

* JSON简介
* JSON语法
* JSON对象
* JSON数组
* JSON和JS对象互转

## JSON 简介

### 什么是 JSON

JSON是JavaScript Object Notation的简称，中文含义为“JavaScript 对象表示法”，它是一种数据交换的文本格式，而不是一种编程语言。

JSON 是一种轻量级的数据交换格式，它基于 ECMAScript (w3c制定的js规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

JSON 使用 JavaScript 语法来描述数据对象，请看以下实例：

{

"social": [

{ "weibo": "https://weibo.com/leiqikui" },

{ "github": "https://github.com/leiqikui" }

]

}

以上这个 social 对象，用来描述作者的社交媒体信息，它是包含 2 个社交媒体对象的数组，一个社交媒体对象是 weibo ，另一个社交媒体对象是 github。

JSON之所以受欢迎，主要是因为它仍然使用 JavaScript 语法来描述数据对象，并没有改变开发人员的使用习惯，这更容易被开发人员接受。由于这种相似性，JavaScript 程序就无需解析器，便可以直接用 JSON 数据来生成原生的 JavaScript 对象。

### JSON 的特点

JSON 主要具有以下特性，这些特性使它成为理想的数据交换语言：

* JSON 是轻量级的文本数据交换格式
* JSON 具有自我描述性，更易理解
* JSON 采用完全独立于语言的文本格式：JSON 使用 JavaScript 语法来描述数据对象，但是 JSON 仍然独立于语言和平台。JSON 解析器和 JSON 库支持许多不同的编程语言。 目前常见的动态编程语言（PHP，JSP，.NET）都支持JSON。

JSON 是存储和交换文本信息的一种语法，它与XML具有相同的特性，是一种数据存储格式，却比 XML 更小、更快、 更易于人编写和阅读、更易于生成和解析。

类似于 XML 的特性：

* JSON 是纯文本
* JSON 具有“自我描述性”（人类可读）
* JSON 具有层级结构（值中存在值）
* JSON 可通过 JavaScript 进行解析
* JSON 数据可使用 AJAX 进行传输

相比 XML 的不同之处：

* 没有结束标签
* 更短
* 读写的速度更快
* 能够使用内建的 JavaScript eval() 方法进行解析
* 使用数组
* 不使用保留字

请看一个简单实例：

<html>

<body>

<p>

weibo: <span id="weibo"></span><br />

github: <span id="github"></span>

</p>

<script type="text/JavaScript">

var JSONObject= {

"weibo": "https://weibo.com/leiqikui" ,

"github": "https://github.com/leiqikui"

};

document.getElementById("weibo").innerHTML=JSONObject.weibo;

document.getElementById("github").innerHTML=JSONObject.github;

</script>

</body>

</html>

在浏览器中运行以上代码，得到的结果为：

weibo: https://weibo.com/leiqikui

github: https://github.com/leiqikui

## JSON 语法

在 JavaScript 语言中，一切皆对象。任何支持的类型，如字符串、数值、对象、数组等，都可以通过 JSON 来表示。

JSON 语法是 JavaScript 对象表示法的子集：

* 数据在键/值对中，键/值对可以嵌套
* 数据由逗号分隔
* 花括号保存对象，对象可以包含多个键/值对
* 方括号保存数组，数组可以包含多个对象

JSON不是JavaScript语句，对象的末尾没有分号，对象和数组的最后一个成员后面，也不能加逗号。

### JSON 键/值对

JSON键/值对由键和值组成，键必须是字符串，值可以是字符串（string）、数值(number) 、对象（object）、数组（array）、true、false、 null。如图 1‑1所示：



图 1‑1 JSON的值

在定义JSON键/值时，先是键名，后面写一个冒号，然后是值。如：

"github": "https://github.com/leiqikui"

这就等价于这条 JavaScript 语句：

github = "https://github.com/leiqikui"

### JSON 字符串

JSON 的字符串（string）跟C或者Java的字符串非常相似，是由双引号包围的任意数量Unicode字符的集合，使用反斜线转义。如图 1‑2所示：



图 1‑2 JSON字符串

由于JSON 字符串不能使用单引号，因此一个字符（character）就是一个单独的字符串（character string）。

### JSON 数值

JSON 数值（number）与C或者Java的数值非常相似，支持整型、浮点型，浮点型也可以用指数表示。如图 1‑3所示：



图 1‑3 JSON数值

JSON不支持八进制和十六进制， 数值必须以十进制表示，且不能使用NaN和Infinity。以下是合法的JSON数值：

{ "age": 30 }

### JSON 对象

JSON 对象是一个无序的key/value对的集合。一个对象以 '{' 开始，以 '}' 结束；每个key 和 value 中使用冒号分隔；每个 key/value 对之间使用逗号分隔。如图 1‑4所示：



图 1‑4 JSON对象

JSON 对象语法：

* JSON 对象在大括号({})中书写。
* 对象可以包含多个 key/value（键/值）对。
* key是字符串，必须放在双引号里面。value 可以是合法的 JSON 数据类型（字符串, 数值, 对象, 数组, 布尔值或 null）。
* 一个 key/value 对内部，key 和 value使用冒号(:)分隔。
* key/value 对之间，使用逗号(,)分隔。
* JSON 对象可以嵌套

下面就是一个合法的JSON 对象：

{

"weibo": "https://weibo.com/leiqikui" ,

"github": "https://github.com/leiqikui" ,

"qq": {

"number": "376601179",

"email": "376601179@qq.com"

}

}

### JSON 数组

数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。如图 1‑5所示：



图 1‑5 JSON数组

JSON 数组：

* JSON 数组在中括号中书写。
* JSON 中数组值必须是合法的 JSON 数据类型（字符串, 数值, 对象, 数组, 布尔值或 null），也可以是 JavaScript 的表达式，包括函数、日期、undefined。

对象的属性值也可以是一个JSON数组。如，以下对象的 sites 属性值是一个包含三个元素的数组：

{

"name": "网站",

"num": 3,

"sites": [ "Google.com", "Taobao.com", "Waibo.wang" ]

}

使用索引值来访问数组的元素。如：

x = myObj.sites[2]; // "Waibo.wang"

JSON数组的元素也可以是对象。如，以下对象 "employees" 是包含三个对象的数组，每个对象代表一条关于某人（有姓和名）的记录：

{

"employees": [

{ "firstName":"John" , "lastName":"Doe" },

{ "firstName":"Anna" , "lastName":"Smith" },

{ "firstName":"Peter" , "lastName":"Jones" }

]

}

### JSON 布尔值

JSON 布尔值可以是 true 或者 false。如：

{ "flag":true }

### JSON null

JSON 可以设置 null 值。如：

{ "zhihu":null }

### JSON 文件

* JSON 文件的文件类型是 ".json"
* JSON 文本的 MIME 类型是 "application/json"

## JSON 对象

### 访问对象的值

可以使用点号（.），或中括号（[]）来访问对象的值。

使用点号访问对象时，直接使用对象的 key 来访问， key 无须放在双引号中。如：

<script>

var myObj, x;

myObj = { "github": "https://github.com/leiqikui" };

x = myObj.github;

document.write(x);

</script>

在浏览器中运行，得到运行结果为：

https://github.com/leiqikui

使用中括号访问对象时，要把对象的 key 以字符串形式，放在中括号中。也就是说，key 要放在双引号中。如：

<script>

var myObj, x;

myObj = { "github": "https://github.com/leiqikui" };

x = myObj["github"];

document.write(x);

</script>

JSON 对象中，可以包含另外一个 JSON 对象。如，以下代码在 myObj 对象中，嵌套了一个 qq 对象：

{

"weibo": "https://weibo.com/leiqikui" ,

"github": "https://github.com/leiqikui" ,

"qq": {

"number": "376601179",

"email": "376601179@qq.com"

}

}

你可以使用点号(.)、或者中括号([])来访问嵌套的 JSON 对象：

document.write(myObj.qq.email);

或者：

document.write(myObj["qq"]["email"]);

document.write(myObj.qq["email"]);

上述三种方法的效果相同，在浏览器中的运行结果都是：

376601179@qq.com

### 修改对象的值

你可以使用点号(.)来修改 JSON 对象的值。如：

myObj.zhihu = "https://www.zhihu.com/people/leiqikui/";

你可以使用中括号([])来修改 JSON 对象的值。如：

myObj["zhihu"] = "https://www.zhihu.com/people/leiqikui/";

### 删除对象属性

你可以使用 **delete** 关键字来删除 JSON 对象的属性。如：

delete myObj.zhihu

你可以使用中括号([])来删除 JSON 对象的属性。如：

delete myObj["zhihu"];

## JSON 数组

### 遍历数组

你可以使用 for-in循环来遍历数组：

for (i in myObj.sites) {

x += myObj.sites[i] + "<br>";

}

你也可以直接使用 for 循环来遍历数组：

for (i = 0; i < myObj.sites.length; i++) {

x += myObj.sites[i] + "<br>";

}

JSON 对象中的数组也可以包含另外一个数组，或者另外一个 JSON 对象：

myObj = {

"name":"网站",

"num":3,

"sites": [

{ "name":"Google", "info":[ "Android", "Google 搜索", "Google 翻译" ] },

{ "name":"Waibo", "info":[ "歪脖教程", "歪脖工具", "歪脖微信" ] },

{ "name":"Taobao", "info":[ "淘宝", "网购" ] }

]

}

可以使用 for-in 来循环访问每个数组：

for (i in myObj.sites) {

x += "<h1>" + myObj.sites[i].name + "</h1>";

for (j in myObj.sites[i].info) {

x += myObj.sites[i].info[j] + "<br>";

}

}

### 修改数组的值

可以使用索引值来修改数组值。如：

myObj.sites[1] = "Github";

### 删除数组元素

你可以使用 **delete** 关键字来删除数组元素。如：

delete myObj.sites[1];

## JSON 和 JS 对象互转

JSON的本质是一个字符串，它之所以受欢迎，是因为可以把JSON字符串解析为有用的JavaScript对象。

ECMAScript5中，对解析JSON的行为进行了规范，并定义了一个全局对象JSON，该对象有两个方法：JSON.parse() 和 JSON.stringify()。

### JSON.parse()

JSON.parse() 方法，用于把JSON字符串转换为原生的 JavaScript 值或对象。如：

JSON.parse('1') // 1

JSON.parse('{}') // {}

JSON.parse('true') // true

JSON.parse('"foo"') // "foo"

JSON.parse('[1, 5, "false"]') // [1, 5, "false"]

JSON.parse('null') // null

var o = JSON.parse('{"name": "张三"}');

o.name // 张三

如果传递给JSON.parse()方法的字符串不是有效的JSON格式，JSON.parse()方法将报错。如：

JSON.parse("'String'")

//Uncaught SyntaxError: Unexpected token u in JSON at position 0(…)

JSON.parse("undefined")

//Uncaught SyntaxError: Unexpected token u in JSON at position 0(…)

除了直接进行字符串转换之外，JSON.parse()方法也可以接收一个函数参数，该函数被称为还原函数。如果提供还原函数，则对象的每个成员都会调用这个函数。

还原函数接收两个参数，一个键和一个值，并返回一个值。如果还原函数返回undefined，则表示要从结果中删除相应的键；如果返回其他值，则将该值插入到结果中。如：

var o = JSON.parse('{"a":1,"b":2}', function(key, value) {

if (key === ''){

return value;

}

if (key === 'a') {

return value + 10;

}

});

o.a // 11

o.b // undefined

JSON 不能存储 Date 对象。如果你需要存储 Date 对象，需要将其转换为字符串，之后再将字符串转换为 Date 对象。在将日期字符串转换为Date对象时，经常要用到还原函数。如：

var book = {

"title": "javascript",

"date": new Date(2018,1,21)

}

var jsonStr = JSON.stringify(book);

console.log(jsonStr)

//'{"title":"javascript","date":"2018-02-20T16:00:00.000Z"}'

var bookCopy = JSON.parse(jsonStr,function(key,value){

if(key == 'date'){

return new Date(value);

}

return value;

})

console.log(bookCopy.date.getFullYear()); //2018

**🕮 说明：**JSON.parse() 和 eval()

实际上，eval() 函数的功能类似于 JSON.parse()方法，也可以将json字符串转换为json对象。如：

eval('(' + '{"a":1}'+')').a; //1

JSON.parse('{"a":1}').a; //1

但是，eval() 函数可以执行不符合 JSON 格式的代码。如：

eval('(' + '{"a":alert(1)}'+')').a; //弹出1

JSON.parse('{"a":alert(1)}').a; //报错

在浏览器中运行上述代码，eval() 函数就会弹出 alert 框，而 JSON.parse()方法将会报错。由于 eval() 函数会为恶意代码提供可乘之机，因此应当尽量少使用 eval() 函数，而是使用更安全的 JSON.parse()方法。

### JSON.stringify()

JSON.stringify()方法用于把JavaScript值（对象或者数组）序列化为JSON字符串，并返回序列化后的JSON字符串。调用格式为：

JSON.stringify(value[, replacer[, space]])

参数：

value: 必需。将要序列化为JSON 字符串的值。

replacer: 可选。用于替换结果的函数、或仅包括指定的属性数组。

space: 可选。用于文本添加缩进、空格和换行符。

1. 序列化规则

序列化的本质，就是按照 JSON 的语法规则，把 JavaScript 值（对象或者数组），转换为 JSON 字符串，以方便存储和数据交换。

1）数值、布尔值、字符串的包装对象，会转换成对应的原始值。如：

JSON.stringify(1) // "1"

JSON.stringify(false) // "false"

JSON.stringify('abc') // ""abc""

JSON.stringify([1, "false", false]) // '[1,"false",false]'

2）键和值都被序列化，如果键名没有引号，会自动为它添加引号。如：

JSON.stringify({x: 5, y: 6}); // "{"x":5,"y":6}"

JSON.stringify({name: "张三"}) // '{"name":"张三"}'

3）数组，会转换成数组格式。如：

JSON.stringify([]) // "[]"

JSON.stringify([1, "false", false]) // '[1,"false",false]'

4）对象，会转换成对象格式。如：

JSON.stringify({}) // "{}"

JSON.stringify({x: 5, y: 6}); // "{"x":5,"y":6}"

5）正则表达式和数学对象，转换成空对象的字符串形式。如：

JSON.stringify(/foo/) // "{}"

JSON.stringify(Math) // "{}"

6）日期对象和包装对象，转换成字符串。如：

JSON.stringify(new Boolean(true)) //"true"

JSON.stringify(new String('123')) //""123""

JSON.stringify(new Number(1)) //"1"

JSON.stringify(new Date()) //""2016-09-20T02:26:38.294Z""

7） 如果非数组对象的成员是undefined、或任意的函数、或 symbol 值，这个成员会被省略。如果数组对象的成员是undefined、或任意的函数、或 symbol 值，则这些值被转成null。如：

JSON.stringify({

a: function(){},

b: undefined,

c: [ function(){}, undefined ]

});

// "{"c":[null,null]}"

8）不可枚举的属性，会被忽略。如：

var obj = {};

Object.defineProperties(obj, {

'foo': { value: 1, enumerable: true },

'bar': { alue: 2, enumerable: false }

});

JSON.stringify(obj); // "{"foo":1}"

9）所有以 symbol 为键的属性都会被完全忽略掉，即便 replacer 参数中强制指定包含了它们。如：

JSON.stringify(

{[Symbol.for("foo")]: "foo"},

function (k, v) {

if (typeof k === "symbol"){

return "a symbol";

}

}

);

// undefined

1. replacer参数

replacer参数可以是一个函数或者一个数组。

1）函数

如果replacer参数是一个函数，则对象的每个成员都会调用这个函数。该函数接收两个参数，一个是键（key），一个是值（value），执行序列化时，键和值都会被序列化。如：

function dobule(key, value){

if (typeof value === "number") {

value = 2 \* value;

}

return value;

}

console.log(JSON.stringify({a:1,b:2}, dobule())); // "{"a":2,"b":4}"

如果replacer函数返回的是一个对象，则该对象被递归地序列化成JSON字符串，并且允许键为空。如：

var jsonObj = {a: {b: 1}};

JSON.stringify(jsonObj, function(key, value) {

console.log("["+ key +"]:" + value);

return value;

});

上述代码中，对象 jsonObj 一共会被 function函数处理三次。第一次键名为空，键值是整个对象 jsonObj；第二次键名为 a，键值是 {b:1}；第三次键名为 b，键值为 1。运行结果为：

[]:[object Object]

[a]:[object Object]

[b]:1

如果replacer函数返回了undefined或没有返回值，那么相应的属性会被忽略，这种情况将不会被序列化成JSON字符串。如：

JSON.stringify({ a: "abc", b: 123 }, function (key, value) {

if (typeof(value) === "string") {

return undefined;

}

return value;

})

上述代码得到的结果为：

'{"b": 123}'

2）数组

如果replacer是一个数组，数组的值代表将被序列化成JSON字符串的属性名。执行序列化时，则只对数组中所列举的键执行序列化操作，就相当于实现一个过滤器的功能。如：

var jsonObj = {

"title":"javascript",

"group":{

"a":1,

"b":2

}

};

console.log(JSON.stringify(jsonObj,["group","b"]))

上述代码的运行结果为：

{"group":{"b":2}}

需要特别说明的是，过滤器功能只适用于对象，对数组无效。如，

var jsonObj = [1,2];

console.log(JSON.stringify(jsonObj,["0"])); // "[1,2]"

1. space参数

默认情况下，JSON.stringify() 输出的JSON字符串不包括任何空格字符或缩进。如：

var jsonObj = {

"title":"javascript",

"group":{

"name":"jia",

"tel":12345

}

};

JSON.stringify(jsonObj);

//{"title":"javascript","group":{"name":"jia","tel":12345}}

通过指定space参数，可以在JSON字符串中添加空格或字符串，用于增加返回的JSON字符串的可读性。space参数的值可以是数值或字符串。

1）数值

如果space参数的值是数值，则序列化的字符串中，每一级别会比上一级别缩进指定数目的空格（不超过10个空格）。如：

console.log(JSON.stringify({ p1: 1, p2: 2 }, null, 2));

上述代码得到的结果为：

{

"p1": 1,

"p2": 2

}

2）字符串

如果space参数的值是字符串（字符串不超过10个字符），则序列化的字符串中，每一级别会比上一级别的缩进，使用该字符串填充。如：

console.log(JSON.stringify({ p1:1, p2:2 }, null, '|-'));

上述代码得到的结果为：

{

|-"p1": 1,

|-"p2": 2

}

**🕮 注意：**

在序列化一个对象时，如果该对象拥有 toJSON() 方法，那么该 toJSON() 方法就会覆盖该对象默认的序列化行为: 被序列化的不是那个原始对象，而是调用 toJSON() 方法后返回的那个对象。如：

var obj = {

foo: 'foo',

toJSON: function () {

return 'bar';

}

};

JSON.stringify(obj); // '"bar"'

JSON.stringify({x: obj}); // '{"x":"bar"}'

如果 toJSON() 方法返回 undefined，此时如果包含它的对象嵌入在另一个对象中，会导致该对象的值变成 null。而如果包含它的对象是顶级对象，结果就是 undefined。

Date 对象拥有自己的 toJSON() 方法，它会自动将 Date 对象转换成日期字符串。如：

JSON.stringify(new Date("2017-12-29"))

// "2017-12-29T00:00:00.000Z"

# 参考资料

[1] http://www.w3school.com.cn/json/

[2] http://www.runoob.com/json/json-tutorial.html

[3] http://www.json.org/json-zh.html

[4] https://www.cnblogs.com/Arnold2015/p/5910338.html

[5] https://www.cnblogs.com/willperfect/p/4715133.html

[6] http://blog.csdn.net/meng\_xiangfeng/article/details/53131511

[7] http://www.runoob.com/json/json-parse.html

[8] http://jsonapi.org.cn/