JSON

# 语法

JSON 的语法可以表示以下三种类型的值。

* 简单值：使用与JavaScript 相同的语法，可以在JSON 中表示字符串、数值、布尔值和null。但JSON 不支持JavaScript 中的特殊值undefined。
* 对象：对象作为一种复杂数据类型，表示的是一组无序的键值对儿。而每个键值对儿中的值可以是简单值，也可以是复杂数据类型的值。
* 数组：数组也是一种复杂数据类型，表示一组有序的值的列表，可以通过数值索引来访问其中的值。数组的值也可以是任意类型——简单值、对象或数组。

JSON 不支持变量、函数或对象实例，它就是一种表示结构化数据的格式，虽然与JavaScript 中表示数据的某些语法相同，但它并不局限于JavaScript 的范畴。

## 简单值

最简单的 JSON 数据形式就是简单值。例如，下面这个值是有效的 JSON 数据：

5这是 JSON 表示数值 5 的方式。类似地，下面是 JSON 表示字符串的方式：

"Hello world!"JavaScript 字符串与 JSON 字符串的最大区别在于， JSON 字符串必须使用双引号（单引号会导致语法错误）。布尔值和 null 也是有效的 JSON 形式。但是，在实际应用中， JSON 更多地用来表示更复杂的数据

结构，而简单值只是整个数据结构中的一部分。

## 对象

JSON 中的对象与JavaScript 字面量稍微有一些不同。下面是一个JavaScript 中的对象字面量：

var person = {

name: "Nicholas",

age: 29

};

这虽然是开发人员在JavaScript 中创建对象字面量的标准方式，但JSON 中的对象要求给属性加引号。实际上，在JavaScript 中，前面的对象字面量完全可以写成下面这样：

var object = {

**"name"**: "Nicholas",

**"age"**: 29

};

JSON 表示上述对象的方式如下：

{

"name": "Nicholas",

"age": 29

}

与JavaScript的对象字面量相比，JSON 对象有两个地方不一样。首先，没有声明变量（JSON 中没有变量的概念）。其次，没有末尾的分号（因为这不是JavaScript 语句，所以不需要分号）。

## 数组

JSON 中的第二种复杂数据类型是数组。JSON 数组采用的就是JavaScript 中的数组字面量形式。例如，下面是JavaScript 中的数组字面量：

var values = [25, "hi", true];

在JSON 中，可以采用同样的语法表示同一个数组：[25, "hi", true]

同样要注意，JSON 数组也没有变量和分号。把数组和对象结合起来，可以构成更复杂的数据集合，例如：

|  |
| --- |
| [{  "title" : "Professional JavaScript",  "authors" : [  "Nicholas C. Zakas"  ],  edition : 3,  year : 2011  }, {  "title" : "Professional JavaScript",  "authors" : [  "Nicholas C. Zakas"  ],  edition : 2,  year : 2009  }, {  "title" : "Professional Ajax",  "authors" : [  "Nicholas C. Zakas",  "Jeremy McPeak",  "Joe Fawcett"  ],  edition : 2,  year : 2008  }, {  "title" : "Professional Ajax",  "authors" : [  "Nicholas C. Zakas",  "Jeremy McPeak",  "Joe Fawcett"  ],  edition : 1,  year : 2007  }, {  "title" : "Professional JavaScript",  "authors" : [  "Nicholas C. Zakas"  ],  edition : 1,  year : 2006  }  ] |

# 解析与序列化

## JSON对象

ECMAScript 5 对解析JSON 的行为进行规范，定义了全局对象JSON。支持这个对象的浏览器有IE 8+、Firefox 3.5+、Safari 4+、Chrome和Opera 10.5+。JSON 对象有两个方法：stringify()和parse()。在最简单的情况下，这两个方法分别用于把JavaScript 对象序列化为JSON 字符串和把JSON 字符串解析为原生JavaScript 值。

在序列化JavaScript 对象时，所有函数及原型成员都会被有意忽略，不体现在结果中。此外，值为undefined 的任何属性也都会被跳过。结果中最终都是值为有效JSON 数据类型的实例属性。

将JSON 字符串直接传递给JSON.parse()就可以得到相应的JavaScript 值。例如，使用下列代码就可以创建与book 类似的对象：

var bookCopy = JSON.parse(jsonText);

注意，虽然book 与bookCopy 具有相同的属性，但它们是两个独立的、没有任何关系的对象。如果传给JSON.parse()的字符串不是有效的JSON，该方法会抛出错误。

|  |
| --- |
| var book = {  title: "Professional JavaScript",  authors: [  "Nicholas C. Zakas"  ],  edition: 3,  year: 2011  };  var jsonText = JSON.stringify(book);  console.log(jsonText);//{"title":"Professional JavaScript","authors":["Nicholas C. Zakas"],"edition":3, "year":2011}  var bookCopy = JSON.parse(jsonText);  console.log(bookCopy); |

## 序列化选项

实际上，JSON.stringify()除了要序列化的JavaScript 对象外，还可以接收另外两个参数，这两个参数用于指定以不同的方式序列化JavaScript 对象。第一个参数是个过滤器，可以是一个数组，也可以是一个函数；第二个参数是一个选项，表示是否在JSON 字符串中保留缩进。单独或组合使用这两个参数，可以更全面深入地控制JSON 的序列化。

## 解析选项

JSON.parse()方法也可以接收另一个参数，该参数是一个函数，将在每个键值对儿上调用。为了区别JSON.stringify()接收的替换（过滤）函数（replacer），这个函数被称为还原函数（reviver），但实际上这两个函数的签名是相同的——它们都接收两个参数，一个键和一个值，而且都需要返回一个值。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。在将日期字符串转换为Date 对象时，经常要用到还原函数。

|  |
| --- |
| var book = {  "title": "Professional JavaScript",  "authors": [  "Nicholas C. Zakas"  ],  edition: 3,  year: 2011,  releaseDate: new Date(2011, 11, 1)  };  var jsonText = JSON.stringify(book);  var bookCopy = JSON.parse(jsonText, function(key, value){  if (key == "releaseDate"){  return new Date(value);  } else {  return value;  }  });  alert(bookCopy.releaseDate.getFullYear()); |