第20章

JSON

本章内容

- 理解 JSON 语法
- 解析 JSON
- 序列化 JSON

曾经有一段时间，XML 是互联网上传输结构化数据的事实标准。Web 服务的第一次浪潮很大程度上都是建立在 XML 之上的，突出的特点是服务器与服务器间通信。然而，业界一直不乏质疑 XML 的声音。不少人认为 XML 过于烦琐、冗长。为解决这个问题，也涌现了一些方案。不过，Web 的发展方向已经改变了。

2006 年，Douglas Crockford 把 JSON（JavaScript Object Notation，JavaScript 对象表示法）作为 IETF RFC 4627 提交给 IETF，而 JSON 的应用早在 2001 年就已经开始了。JSON 是 JavaScript 的一个严格的子集，利用了 JavaScript 中的一些模式来表示结构化数据。Crockford 认为与 XML 相比，JSON 是在 JavaScript 中读写结构化数据的更好的方式。因为可以把 JSON 直接传给 `eval()`，而且不必创建 DOM 对象。

关于 JSON，最重要的是要理解它是一种数据格式，不是一种编程语言。虽然具有相同的语法形式，但 JSON 并不从属于 JavaScript。而且，并不是只有 JavaScript 才使用 JSON，毕竟 JSON 只是一种数据格式。很多编程语言都有针对 JSON 的解析器和序列化器。

20.1 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，表示的是一组有序的键值对儿。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。

20.1.1 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。

20.1.2 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）。再说一遍，对象的属性必须加双引号，这在 JSON 中是必需的。属性的值可以是简单值，也可以是复杂类型值，因此可以像下面这样在对象中嵌入对象：

```
{
    "name": "Nicholas",
    "age": 29,
    "school": {
        "name": "Merrimack College",
        "location": "North Andover, MA"
    }
}
```

这个例子在顶级对象中嵌入了学校（"school"）信息。虽然有两个"name"属性，但由于它们分别属于不同的对象，因此这样完全没有问题。不过，同一个对象中绝对不应该出现两个同名属性。

与 JavaScript 不同，JSON 中对象的属性名任何时候都必须加双引号。手工编写 JSON 时，忘了给对象属性名加双引号或者把双引号写成单引号都是常见的错误。

20.1.3 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
        }
    ]
```

这个数组中包含一些表示图书的对象。每个对象都有几个属性，其中一个属性是"authors"，这个属性的值又是一个数组。对象和数组通常是 JSON 数据结构的最外层形式（当然，这不是强制规定的），利用它们能够创造出各种各样的数据结构。

20.2 解析与序列化

JSON 之所以流行，拥有与 JavaScript 类似的语法并不是全部原因。更重要的一个原因是，可以把 JSON 数据结构解析为有用的 JavaScript 对象。与 XML 数据结构要解析成 DOM 文档而且从中提取数据极为麻烦相比，JSON 可以解析为 JavaScript 对象的优势极其明显。就以上一节中包含一组图书的 JSON 数据结构为例，在解析为 JavaScript 对象后，只需要下面一行简单的代码就可以取得第三本书的书名：

```
books[2].title
```

当然，这里是假设把解析后 JSON 数据结构得到的对象保存到了变量 books 中。再看看下面在 DOM 结构中查找数据的代码：

```
doc.getElementsByTagName("book")[2].getAttribute("title")
```

看看这些多余的方法调用，就不难理解为什么 JSON 能得到 JavaScript 开发人员的热烈欢迎了。从此以后，JSON 就成了 Web 服务开发中交换数据的事实标准。

20.2.1 JSON 对象

早期的 JSON 解析器基本上就是使用 JavaScript 的 eval()函数。由于 JSON 是 JavaScript 语法的子集，因此 eval()函数可以解析、解释并返回 JavaScript 对象和数组。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+。对于较早版本的浏览器，可以使用一个 shim：https://github.com/douglascrockford/JSON-js。在旧版本的浏览器中，使用 eval()对 JSON 数据结构求值存在风险，因为可能会执行一些恶意代码。对于不能原生支持 JSON 解析的浏览器，使用这个 shim 是最佳选择。

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。例如：

```
var book = {
                title: "Professional JavaScript",
                authors: [
                   "Nicholas C. Zakas"
                ],
                edition: 3,
                year: 2011
            };

var jsonText = JSON.stringify(book);
```

JSONStringifyExample01.htm

这个例子使用 JSON.stringify()把一个 JavaScript 对象序列化为一个 JSON 字符串，然后将它保存在变量 jsonText 中。默认情况下，JSON.stringify()输出的 JSON 字符串不包含任何空格字符或缩进，因此保存在 jsonText 中的字符串如下所示：

```
{"title":"Professional JavaScript","authors":["Nicholas C. Zakas"],"edition":3,
"year":2011}
```

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。

20.2.2 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。

1. 过滤结果

如果过滤器参数是数组，那么 JSON.stringify()的结果中将只包含数组中列出的属性。来看下面的例子。

```
var book = {
                "title": "Professional JavaScript",
                 "authors": [
                    "Nicholas C. Zakas"
                 ],
                 edition: 3,
                 year: 2011
            };

var jsonText = JSON.stringify(book, ["title", "edition"]);
```

JSONStringifyExample01.htm

JSON.stringify()的第二个参数是一个数组，其中包含两个字符串："title"和"edition"。这两个属性与将要序列化的对象中的属性是对应的，因此在返回的结果字符串中，就只会包含这两个属性：

```
{"title":"Professional JavaScript","edition":3}
```

如果第二个参数是函数，行为会稍有不同。传入的函数接收两个参数，属性（键）名和属性值。根据属性（键）名可以知道应该如何处理要序列化的对象中的属性。属性名只能是字符串，而在值并非键值对儿结构的值时，键名可以是空字符串。

为了改变序列化对象的结果，函数返回的值就是相应键的值。不过要注意，如果函数返回了

undefined，那么相应的属性会被忽略。还是看一个例子吧。

```
var book = {
                "title": "Professional JavaScript",
                "authors": [
                    "Nicholas C. Zakas"
                ],
                edition: 3,
                year: 2011
            };

var jsonText = JSON.stringify(book, function(key, value){
    switch(key){
        case "authors":
            return value.join(",")

        case "year":
            return 5000;

        case "edition":
            return undefi ned;

        default:
            return value;
    }
});
```

JSONStringifyExample02.htm

这里，函数过滤器根据传入的键来决定结果。如果键为"authors"，就将数组连接为一个字符串；如果键为"year"，则将其值设置为 5000；如果键为"edition"，通过返回 undefined 删除该属性。最后，一定要提供 default 项，此时返回传入的值，以便其他值都能正常出现在结果中。实际上，第一次调用这个函数过滤器，传入的键是一个空字符串，而值就是 book 对象。序列化后的 JSON 字符串如下所示：

```
{"title":"Professional JavaScript","authors":"Nicholas C. Zakas","year":5000}
```

要序列化的对象中的每一个对象都要经过过滤器，因此数组中的每个带有这些属性的对象经过过滤之后，每个对象都只会包含"title"、"authors"和"year"属性。

Firefox 3.5 和 3.6 对 JSON.stringify()的实现有一个 bug，在将函数作为该方法的第二个参数时这个 bug 就会出现，即这个函数只能作为过滤器：返回 undefined 意味着要跳过某个属性，而返回其他任何值都会在结果中包含相应的属性。Firefox 4 修复了这个 bug。

2. 字符串缩进

JSON.stringify()方法的第三个参数用于控制结果中的缩进和空白符。如果这个参数是一个数值，那它表示的是每个级别缩进的空格数。例如，要在每个级别缩进 4 个空格，可以这样写代码：

```
var book = {
                "title": "Professional JavaScript",
                "authors": [
                    "Nicholas C. Zakas"
                ],
                edition: 3,
                year: 2011
```

```
};

var jsonText = JSON.stringify(book, null, 4);
```

JSONStringifyExample03.htm

保存在 jsonText 中的字符串如下所示：

```
{
    "title": "Professional JavaScript",
    "authors": [
        "Nicholas C. Zakas"
    ],
    "edition": 3,
    "year": 2011
}
```

不知道读者注意到没有，JSON.stringify()也在结果字符串中插入了换行符以提高可读性。只要传入有效的控制缩进的参数值，结果字符串就会包含换行符。（只缩进而不换行意义不大。）最大缩进空格数为 10，所有大于 10 的值都会自动转换为 10。

如果缩进参数是一个字符串而非数值，则这个字符串将在 JSON 字符串中被用作缩进字符（不再使用空格）。在使用字符串的情况下，可以将缩进字符设置为制表符，或者两个短划线之类的任意字符。

```
var jsonText = JSON.stringify(book, null, "--");
```

这样，jsonText 中的字符串将变成如下所示：

```
{
--"title": "Professional JavaScript",
--"authors": [
----"Nicholas C. Zakas"
--],
--"edition": 3,
--"year": 2011
}
```

缩进字符串最长不能超过 10 个字符长。如果字符串长度超过了 10 个，结果中将只出现前 10 个字符。

3. toJSON()方法

有时候，JSON.stringify()还是不能满足对某些对象进行自定义序列化的需求。在这些情况下，可以通过对象上调用 toJSON()方法，返回其自身的 JSON 数据格式。原生 Date 对象有一个 toJSON()方法，能够将 JavaScript 的 Date 对象自动转换成 ISO 8601 日期字符串（与在 Date 对象上调用 toISOString()的结果完全一样）。

可以为任何对象添加 toJSON()方法，比如：

```
var book = {
            "title": "Professional JavaScript",
            "authors": [
                "Nicholas C. Zakas"
            ],
            edition: 3,
            year: 2011,
            toJSON: function(){
```

```
            return this.title;
        }
    };

var jsonText = JSON.stringify(book);
```

JSONStringifyExample05.htm

以上代码在 book 对象上定义了一个 toJSON()方法，该方法返回图书的书名。与 Date 对象类似，这个对象也将被序列化为一个简单的字符串而非对象。可以让 toJSON()方法返回任何序列化的值，它都能正常工作。也可以让这个方法返回 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。

toJSON()可以作为函数过滤器的补充，因此理解序列化的内部顺序十分重要。假设把一个对象传入 JSON.stringify()，序列化该对象的顺序如下。

(1) 如果存在 toJSON()方法而且能通过它取得有效的值，则调用该方法。否则，按默认顺序执行序列化。

(2) 如果提供了第二个参数，应用这个函数过滤器。传入函数过滤器的值是第(1)步返回的值。

(3) 对第(2)步返回的每个值进行相应的序列化。

(4) 如果提供了第三个参数，执行相应的格式化。

无论是考虑定义 toJSON()方法，还是考虑使用函数过滤器，亦或需要同时使用两者，理解这个顺序都是至关重要的。

20.2.3 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。例如：

```
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());
```

JSONParseExample02.htm

以上代码先是为 book 对象新增了一个 releaseDate 属性，该属性保存着一个 Date 对象。这个对象在经过序列化之后变成了有效的 JSON 字符串，然后经过解析又在 bookCopy 中还原为一个 Date 对象。还原函数在遇到"releaseDate"键时，会基于相应的值创建一个新的 Date 对象。结果就是 bookCopy.releaseDate 属性中会保存一个 Date 对象。正因为如此，才能基于这个对象调用 getFullYear()方法。

20.3 小结

JSON 是一个轻量级的数据格式，可以简化表示复杂数据结构的工作量。JSON 使用 JavaScript 语法的子集表示对象、数组、字符串、数值、布尔值和 null。即使 XML 也能表示同样复杂的数据结果，但 JSON 没有那么烦琐，而且在 JavaScript 中使用更便利。

ECMAScript 5 定义了一个原生的 JSON 对象，可以用来将对象序列化为 JSON 字符串或者将 JSON 数据解析为 JavaScript 对象。JSON.stringify()和 JSON.parse()方法分别用来实现上述两项功能。这两个方法都有一些选项，通过它们可以改变过滤的方式，或者改变序列化的过程。

原生的 JSON 对象也得到了很多浏览器的支持，比如 IE8+、Firefox 3.5+、Safari 4+、Opera 10.5 和 Chrome。