# JSON学习

**从结构上看，所有的数据（data）最终都可以分解成三种类型**：

任何疑问都可以咨询本人：148334733

第一种类型是**标量**（scalar），也就是一个单独的字符串（string）或数字（numbers），比如"北京"这个单独的词。

第二种类型是**序列**（sequence），也就是若干个相关的数据按照一定顺序并列在一起，又叫做数组（array）或列表（List），比如"北京，上海"。

第三种类型是**映射**（mapping），也就是一个名/值对（Name/value），即数据有一个名称，还有一个与之相对应的值，这又称作散列（hash）或字典（dictionary），比如"首都：北京"。

1） 并列的数据之间用逗号（", "）分隔。

2） 映射用冒号（": "）表示。

3） 并列数据的集合（数组）用方括号("[]")表示。

4） 映射的集合（对象）用大括号（"{}"）表示。

[  
　　{"城市":"北京","面积":16800,"人口":1600},  
　　{"城市":"上海","面积":6400,"人口":1800}  
]

成组的键/值数组对，用[]

[ 数组

{ 对象 },

{ 对象 },

{ 对象 },

]，

[数组

......

]

JSON具有以下这些形式：

**对象（*object*）** 是一个无序的“‘名称/值’对”集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间使用“,”（逗号）分隔。



**数组（*array*）** 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。



**值（*value*）** 可以是双引号括起来的字符串（*string*）、数值(number)、true、false、 null、对象（object）或者数组（array）。这些结构可以嵌套。



**字符串（*string*）** 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。一个字符（character）即一个单独的字符串（character string）。

JSON的字符串（*string*）与C或者Java的字符串非常相似。



**数值（*number*）** 也与C或者Java的数值非常相似。只是JSON的数值没有使用八进制与十六进制格式。



# JS中的应用

早期的JSON解析器基本上就是使用JavaScript的eval()函数。由于JSON是JavaScript语法的自己，因此eval()函数可以解析、解释并返回JavaScript的对象和数组。   
ECMAScript 5对解析JSON的行为进行了规范，定义了全局对象JSON。   
  
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。例如：   
新建一个HTML：代码如下：

复制代码代码如下:

<html>   
<head>   
<title></title>   
<script type="text/javascript">   
function init()   
{   
var book={   
title:"JavaScript高级程序设计",   
authors:[   
"Nicholas C. Zakas"   
],   
edition:3,   
year:2011   
};   
var jsonBook=JSON.stringify(book);   
var objectBook=JSON.parse(jsonBook);   
var title=objectBook.title;} </script> </head> <body> <input type="button" onclick="init()" value="测试" /> </body> </html>

默认情况下，JSON.stringify()输出的JSON字符串不包含任何空字符或缩进，因此保存在jsonBook中的字符串如下所示：

复制代码代码如下:

{"title":"JavaScript高级程序设计","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。   
  
注意，虽然book与objectBook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。

注：ie8(兼容模式),ie7和ie6没有JSON对象，推荐采用JSON官方的方式，引入json2.js。原先的都使用eval()，但现在确实不推荐这么用了。

上一节我们讲解了JSON.stringify()的基本用法，这一节我们来重点探讨一下序列化。   
JSON.stringify()除了要序列化的js对象外，还可以接收另外两个参数，这两个参数用于指定不同方式序列化js对象。第一个参数是过滤器，可以使一个数组，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。如：

复制代码代码如下:

<html>   
<head>   
<title></title>   
<script type="text/javascript">   
function init()   
{   
var student={   
name:"Bill",   
age:12,   
grade:3,   
id:"0802020114"   
};   
var jsonText=JSON.stringify(student,["name","id"]);   
}   
</script>   
</head>   
<body>   
<input type="button" onclick="init()" value="测试" />   
</body>   
</html>

jsonText的值为   
{"name":"Bill","id":"802020114"}   
如果第二个参数是函数，行为会有一点不同。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。   
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。如：

复制代码代码如下:

<html>   
<head>   
<title></title>   
<script type="text/javascript">   
function init()   
{   
var student={   
name:"Bill",   
age:12,   
grade:3,   
id:"0802020114",   
subject:["math","Chinese","English"]   
};   
var jsonText=JSON.stringify(student,jsonConvert);   
}   
function jsonConvert(key,value)   
{   
switch (key)   
{   
case "name":   
return "Lily";   
case "grade":   
return undefined;   
case "subject":   
return value.join(",");   
default :   
return value;   
}   
}   
</script>   
</head>   
<body>   
<input type="button" onclick="init()" value="测试" />   
</body>   
</html>

这里函数过滤器根据传入的键来决定结果。如果键为name，就将其值设置为Lily；如果为grade就返回undefined来删除该属性；如果为subject，它是一个数组，就将它通过数组方法join()转化为以逗号连接的字符串。最后一定要提供default项，使其他的值都能够正常出现在结果中，不然就会出错，没有结果。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student对象。上面的jsonText的值如下：   
{"name":"Lily","age":12,"id":"0802020114","subject":"math,Chinese,English"}   
在内部是按顺序遍历每一个对象每一个属性，所以公开key,value函数方法需要注意你的函数只有着两个参数才能起效，要理解函数的目的，就是在内部机制遍历每一个属性的时候让你来修改部分结果，并且是一次遍历每一个对象，这样在序列化对象中每一个对象都要经过过滤器。   
**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，那么它表示的是每个级别缩进的空格数。如：

复制代码代码如下:

<html>   
<head>   
<title></title>   
<script type="text/javascript">   
function init()   
{   
var student={   
name:"Bill",   
age:12,   
grade:3,   
id:"0802020114",   
subject:["math","Chinese","English"]   
};   
var jsonText=JSON.stringify(student,null,8);   
}   
</script>   
</head>   
<body>   
<input type="button" onclick="init()" value="测试" />   
</body>   
</html>

保存在jsonText中的字符串为：

复制代码代码如下:

{   
"name": "Bill",   
"age": 12,   
"grade": 3,   
"id": "0802020114",   
"subject": [   
"math",   
"Chinese",   
"English"   
]   
}

JSON.stringify()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）。最大缩进空格数位10，所有大于10的值都会自定转换为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）。如可以实现如下的效果：

复制代码代码如下:

{   
\*"name": "Bill",   
\*"age": 12,   
\*"grade": 3,   
\*"id": "0802020114",   
\*"subject": [   
\*\*"math",   
\*\*"Chinese",   
\*\*"English"   
\*]   
}

同样字符串最长不能超过10个字符长。如果超过，结果中将只出现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。如：

复制代码代码如下:

<html>   
<head>   
<title></title>   
<script type="text/javascript">   
function init()   
{   
var student={   
name:"Bill",   
age:12,   
grade:3,   
id:"0802020114",   
subject:["math","Chinese","English"],   
toJSON:function(){   
return this.name+"\_"+this.id;   
}   
};   
var jsonText=JSON.stringify(student);   
}   
</script>   
</head>   
<body>   
<input type="button" onclick="init()" value="测试" />   
</body>   
</html>

以上代码在student对象上定义了一个toJSON()方法，该方法返回name和id的组合。最后jsonText的值如下：   
"Bill\_0802020114"   
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是：   
（1）如果存在toJSON()方法而且能够通过它取得有效值，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。   
无论是考虑滴定toJSON()方法，还是考虑使用函数过滤器，或者需要同时使用两者，理解这个顺序都是至关重要的。

这一节我们主要讨论json的解析。   
JSON.parse()方法也可以接收另一个参数，该参数是一个函数，将早每个键值对上调用。为了区别JSON.stringify()接收的替换（过滤）函数（replacer），这个函数被称作还原函数（reviver），但实际上这两个函数的签名是相同的——它们都接收连个参数，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。如：

复制代码代码如下:

<html>   
<head>   
<title></title>   
<script type="text/javascript">   
function init()   
{   
var student={   
name:"Bill",   
birthDate:new Date(1990,8,4)   
};   
var jsonText=JSON.stringify(student);   
var studentObject=JSON.parse(jsonText,function(key,value){   
if(key=="birthDate")   
{   
return new Date(value);   
}   
else   
{   
return value;   
}   
});   
}   
</script>   
</head>   
<body>   
<input type="button" onclick="init()" value="测试" />   
</body>   
</html>

以上代码先是为student添加了出生日期birthDate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地JSON字符串，然后经过解析又在studentObject中还原为一个Date对象。

以上所有的知识点都只说明了两个函数的使用，

JSON.stringify( ) javascript对象或者数组 ===》 JSON字符串

对象的原型、函数方法忽略

JSON.parse( ) JSON字符串 ===》 javascript对象或者数组

以上的方法是原生的JS当中提供的两个方法

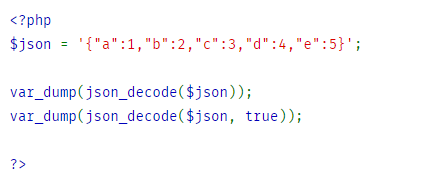
这两个方法只有ES5中提供了JSON这个对象，对IE678的低版本浏览器是不支持的，可以调用JSON2.js文件，它支持上述的方法。

# PHP中的应用

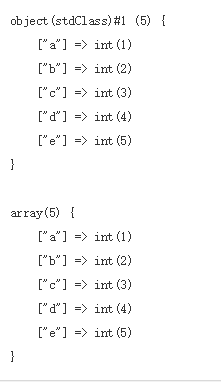
PHP中，

json\_decode JSON字符串 ===> PHP对象(数组)

需要注意的是，如果制定了第二个参数为true,则转化为数组.



另外，json只能用来表示对象（object）和数组（array），如果对一个字符串或数值使用json\_decode()，将会返回null。这里，主要关注的是JSON字符串的正确格式，千万别写错了，还有类型一定要对，不能是除去对象和数组的另外的数据类型。这就是需要注意的几点问题。

数组用下标访问，对象是用->来访问的。

Json\_encode PHP对象(数组) ===> JSON字符串

$arr = array ('a'=>1,'b'=>2,'c'=>3,'d'=>4,'e'=>5);

　　echo json\_encode($arr);

结果为

　　{"a":1,"b":2,"c":3,"d":4,"e":5}

再看一个对象转换的例子：

　　$obj->body = 'another post';

　　$obj->id = 21;

　　$obj->approved = true;

　　$obj->favorite\_count = 1;

　　$obj->status = NULL;

　　echo json\_encode($obj);

结果为

　　{

　　　　"body":"another post",

　　　　"id":21,

　　　　"approved":true,

　　　　"favorite\_count":1,

　　　　"status":null

　　}

由于json只接受utf-8编码的字符，所以json\_encode()的参数必须是utf-8编码，否则会得到空字符或者null。当中文使用GB2312编码，或者外文使用ISO-8859-1编码的时候，这一点要特别注意。怎么才能正确解析中文呢？

一定要仔细的检查，是否是uft8的编码，保证编码的正确性。

# jquery的应用

实际的，jquery中的应用：

1，一般情况下，数据都是通过ajax传递过来的。

2，不是ajax的情况下解析的。

来看一下第一种情况：

你也可以简单一点使用$.getJSON,实际上还是调用的dataype:JSON 这种格式的，一样的。

$.ajax({

dataType: "json",

url: url,

data: data,

success: success

});

简单的设置的话也可以这么来：

## jQuery.getJSON( url [, data ] [, success(data, textStatus, jqXHR) ] )

如果URL包含字符串“callback=?”（或类似的参数，取决于服务器端 API 是如何定义的），这个请求被视为JSONP形式请求。 这个是针对$.getJSONd的，不是针对所有的$.ajax请求，$.ajax有JSONP的专用写法

这样解析出来的就已经是javascript对象了，已经完成了JSON字符串的解析了。自动完成的，剩下的工作只是遍历而已了。

$.getJSON('ajax/test.json', function(data) {

var items = [];

$.each(data, function(key, val) {

items.push('<li id="' + key + '">' + val + '</li>');

});

$('<ul/>', {

'class': 'my-new-list',

html: items.join('')

}).appendTo('body');

});

$.each就是对这个对象或者数组进行遍历的一个操作，适合比较复杂的JSON格式。

第二种情况：在默认情况下，jquery是如何完成javascript对象和JSON字符串之间的转化的

$.parseJSON JSON 字符串 ===> javascript对象

# 数据较大的情况

$.ajax 提交的数据太大，或者是非常多的情况下怎么弄？

我们上边只考虑了接收数据，那么，传递的数据过多怎么办？

Type:POST,

Data:{

Name:’lizhiyuan’,

Age:20,

}

$\_POST[‘name’] $\_POST[‘age’] 服务器端接收就完了

这里我们考虑下如果这个对象确实比较复杂，嵌套比较多的情况下该如何传值？

1. **var** persons = [{
2. id: "001",
3. name: "zhangsan",
4. age: "20",
5. gender: **true**,
6. city: "shanghai"
7. },
8. {
9. id: "002",
10. name: "lisi",
11. age: "21",
12. gender: **false**,
13. city: "beijing"
14. }
15. ];

Persons[0].id // person[1].name 这么来访问

其实感觉JS对象或者数组跟JSON的区别不是特别的明显

1. **var** option = {
2. url: '/test/ReadPersons',
3. type: 'POST',
4. data: JSON.stringify(persons),
5. dataType: 'html',
6. contentType: 'application/json',
7. success: **function** (result) { alert(result); }
8. };

这样就完成了复杂的传值，具体的思路其实很简单，利用stringify()转化为JSON字符串，

服务器接收到这个JSON字符串之后，使用json\_decode转化为PHP的对象或者数组，需要遍历就遍历，需要取值就取值，就这么简单！