1 JSON 基础

1.1 JSON 的定义

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。易于人阅读和编写，同时也易于机器解析和生成。它基于 JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999 的一个子集。 JSON 采用完全独立于语言的文本格式，但是也使用了类似于 C 语言家族的习惯（包括 C, C++, C#, Java, JavaScript, Perl, Python 等）。这些特性使 JSON 成为理想的数据交换语言。

1.2 JSON 的结构

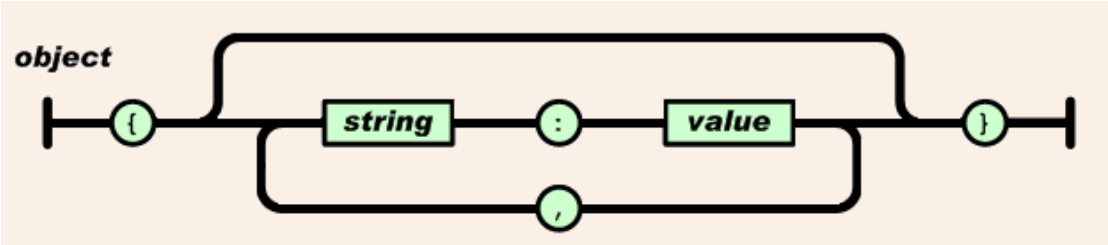
- “名称/值”对的集合(A collection of name/value pairs)。不同的语言中，它被理解为*对象(object)*、纪录（record）、结构(struct)、字典(dictionary)、哈希表(hash table)、有键列表(keyed list)或者关联数组 (associative array)。
- 值的有序列表(An ordered list of values)。在大部分语言中，它被理解为数组(array)。

这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。

1.3 JSON 的形式

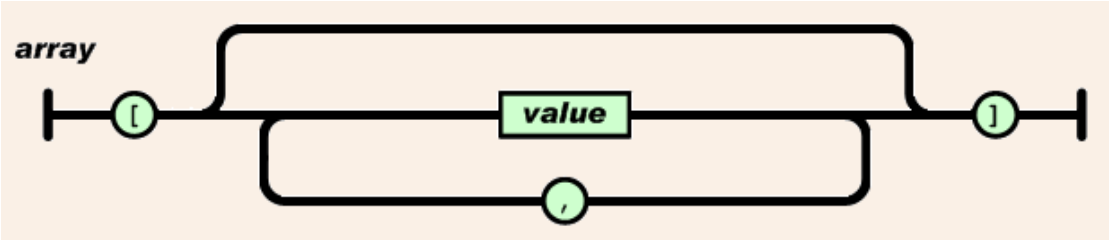
JSON 具有一下这些形式：

（1）对象是一个无序的“‘名称/值’对”集合。一个对象以“{”开始、“}”结束。每个“名称”后跟一个“：”，“‘名称/值’ 对”之间使用“，”分隔。



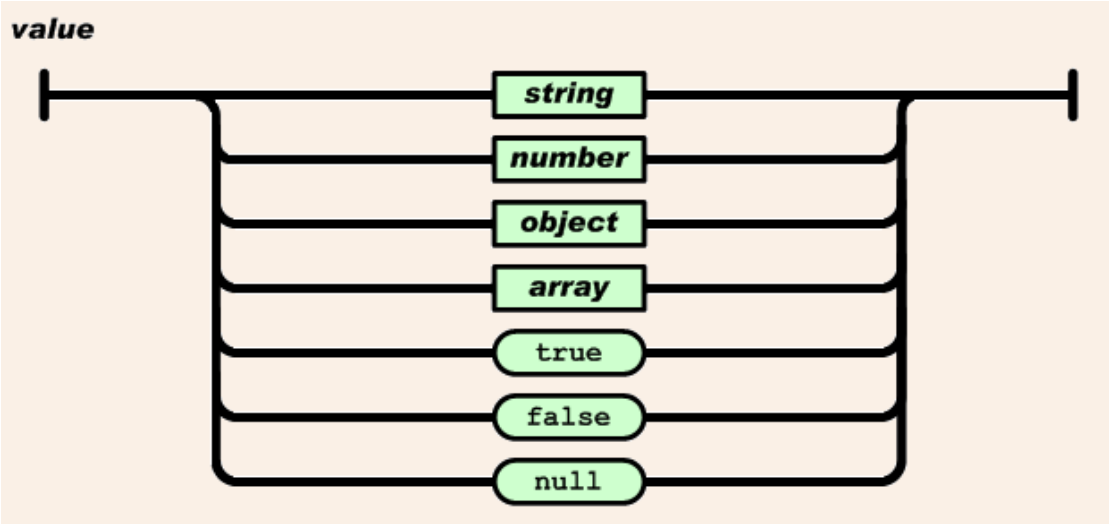
object
{}
{ *members* }
members
pair
pair , *members*
pair
string : *value*

（2）数组是值(value)的有序集合。一个数组以“[”开始、“]”结束。值之间使用“，”分隔。



array
　　[]
　　[*elements*]
elements
　　value
　　value , *elements*

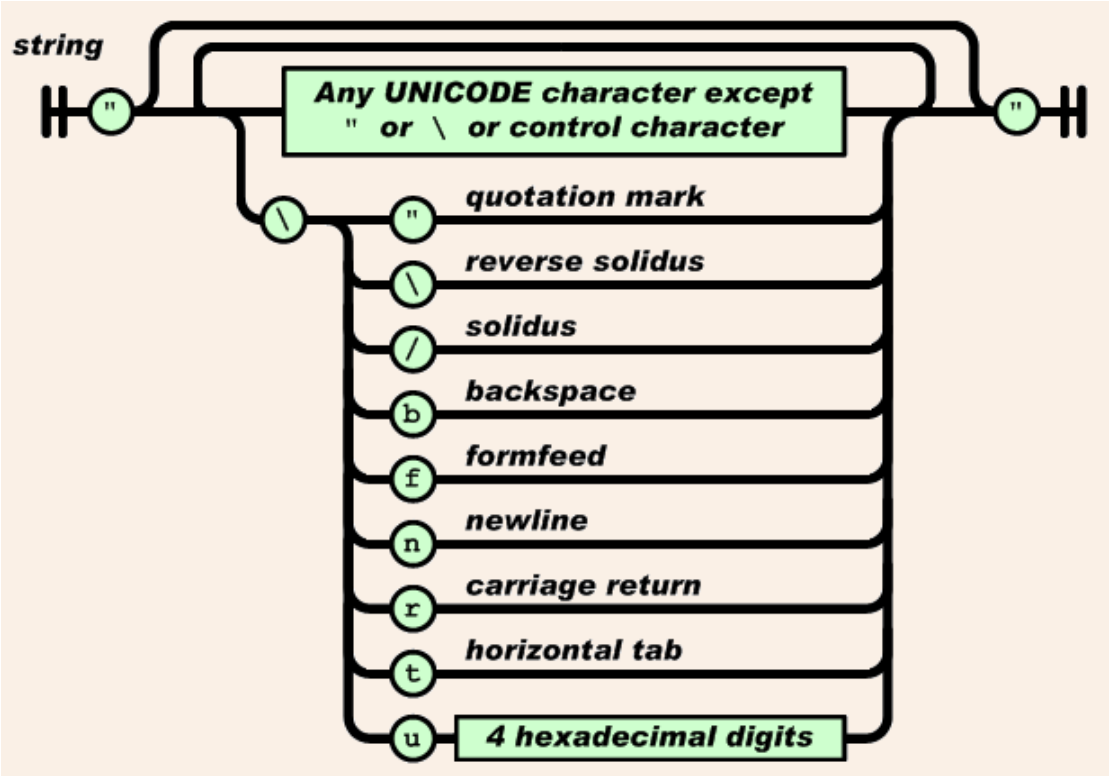
（3）值（*value*）可以是双引号括起来的字符串（*string*）、数值(number)、true、false、 null、对象（object）或者数组(array)。这些结构可以嵌套。



value
　　string
　　number
　　object
　　array
　　true
　　false
　　null

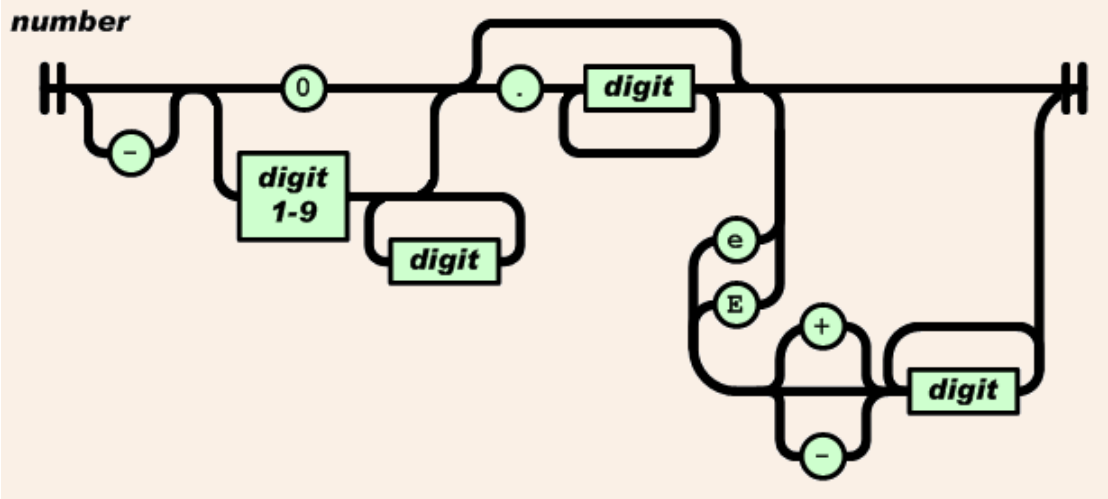
（4）字符串(*string*)是由双引号包围的任意数量 Unicode 字符的集合，使用反斜线转义。一个字符(character)即一个单独的字符串(character string)。

字符串(*string*)与 C 或者 Java 的字符串非常相似。



string
    ""
    " *chars* "
chars
    char
    char chars
char
    any-Unicode-character-
      except-"-or-\-or-
      control-character
    \"
    \\
    \/
    \b
    \f
    \n
    \r
    \t
    \u *four-hex-digits*

（5）数值(*number*)也与 C 或者 Java 的数值非常相似。除去未曾使用的八进制与十六进制格式，除去一些编码细节。



number
    int
    int frac
    int exp
    int frac exp
int
    digit
    digit1-9 digits
    - digit
    - digit1-9 digits
frac
    . digits
exp
    e digits
digits
    digit
    digit digits
e
    e
    e+
    e-
    E
    E+
    E-

空白可以加入到任何符号之间。

2 JSON 示例

2.1 示例一

```
var myJSONObject = {"bindings": [
{"ircEvent": "PRIVMSG", "method": "newURI", "regex": "^http://.*"},
{"ircEvent": "PRIVMSG", "method": "deleteURI", "regex": "^delete.*"},
{"ircEvent": "PRIVMSG", "method": "randomURI", "regex": "^random.*"}
]
};
```

在这个例子中，创建了一个对象，它只包含一个成员“bindings”。“bindings”中有一个包含了 3 个对象的数组，而这每个对象都有“ircEvent”、“method”、“regex”3 个成员。

```
myJSONObject.bindings[0].method 的值是 "newURI"
```

2.2 示例二

按照最简单的形式，可以用下面这样的 JSON 表示名称/值对：

```
{ "firstName": "Brett" }
```

这个示例非常基本，而且实际上比等效的纯文本名称/值对占用更多的空间：

```
firstName=Brett
```

但是，当将多个名称/值对串在一起时，JSON 就会体现出它的价值了。首先，可以创建包含多个名称/值对的*记录*，比如：

```
{ "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" }
```

从语法方面来看，这与名称/值对相比并没有很大的优势，但是在这种情况下，JSON 更容易使用，而且可读性更好。例如，它明确地表示以上三个值都是同一记录的一部分，花括号使这些值有了某种联系。

2.3 值的数组

当需要表示一组值时，JSON 不但能够提高可读性，而且可以减少复杂性。例如，假设你希望表示一个人名列表。在 XML 中，需要许多开始标记和结束标记；如果使用 JSON，就只需将多个花括号的记录分组在一起：

```
{ "people": [
  { "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },
  { "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },
  { "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }
]}
```

这不难理解，在这个示例中，只有一个名为 people 的变量，值是包含三个条目的数组，每个条目是一个人的记录，其中包含名、姓和电子邮件地址。上面的示例演示如何用括号将记录组合成一个值。当然，可以使用相同的语法表示多个值（每个值包含多个记录）：

```
{ "programmers": [
  { "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },
  { "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },
  { "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }
 ],
"authors": [
  { "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction" },
  { "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },
  { "firstName": "Frank", "lastName": "Peretti", "genre": "christian fiction" }
 ],
"musicians": [
  { "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
  { "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }
 ]
}
```

这里最值得注意的是，能够表示多个值，每个值进而包含多个值。但是还应该注意，在不同的主条目(programmers、authors 和 musicians)之间，记录中实际的名称/值对可以不一样。JSON 是完全动态的，允许在 JSON 结构的中间改变表示数据的方式。

在处理 JSON 格式的数据时，没有需要遵守的预定义的约束。所以，在同样的数据结构中，可以改变表示数据的方式，甚至可以以不同方式表示同一事物。

3 JSON 的使用

3.1 JSON 官方网站提供的一个开源的 JSON 解析器和字符串转换器：json.js

String.parseJSON()解析函数，它将 JSON 文本解析称 object 或者 array，可以抛出一个语法错误的异常；

Array.toJSONString(), boolean.toJSONString(), date.toJSONString(), number.toJSONString(), object.toJSONString(), string.toJSONString()这几个函数可以生成 JSON 文本。

下面是一些简单的解析和转换的例子：

```
var str='["0","1"]';
var obj = str.parseJSON();
document.write(obj[0]);
document.write(obj[1]);
```

结果是 01

```
var doc = new Array();
doc[0]='0';
doc[1]='1';
document.write(doc.toJSONString())
```

结果是[“0”,”1”]

3.2 在 JavaScript 中使用 JSON

JSON 以一种特定的字符串形式来表示 JavaScript 对象。如果将具有这样一种形式的字符串赋给任意一个 JavaScript 变量，那么该变量会变成一个对象引用，而这个对象就是字符串所构建出来的，好像有点拗口，我们还是用实例来说明。

3.2.1 将 JSON 数据赋值给变量

例如，可以创建一个新的 JavaScript 变量，然后将 JSON 格式的数据字符串直接赋值给它：

```
var people =
  { "programmers": [
    { "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },
    { "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },
    { "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }
   ],
  "authors": [
    { "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction" },
    { "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },
    { "firstName": "Frank", "lastName": "Peretti", "genre": "christian fiction" }
   ],
  "musicians": [
    { "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
    { "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }
   ]
  }
```

这非常简单。现在 people 包含前面看到的 JSON 格式的数据，但是，这还不够，因为访问数据的方式似乎还不明显。

3.2.2 访问数据

尽管看起来不明显，但是上面的长字符串实际上只是一个数组；将这个数组放进 JavaScript 变量之后，就可以很轻松地访问它。实际上，只需用点号表示法来表示数组元素。所以，要想访问 programmers 列表的第一个条目的姓氏，只需在 JavaScript 中使用下面这样的代码：

```
people.programmers[0].lastName;
```

注意，数组索引是从零开始的。所以，这行代码首先访问 people 变量中的数据；然后移动到称为 programmers 的条目，再移动到第一个记录([0])；最后，访问 lastName 键的值。结果是字符串值“McLaughlin”。

下面是使用同一变量的几个示例。

```
people.authors[1].genre                    // Value is "fantasy"

people.musicians[3].lastName               // Undefined. This refers to the fourth entry,
 and there isn't one

people.programmers.[2].firstName     // Value is "Elliotte"
```

利用这样的语法，可以处理任何 JSON 格式的数据，而不需要使用任何额外的 JavaScript 工具包或 API。

3.2.3 修改 JSON 数据

正如可以用点号和括号访问数据，也可以按照同一的方式轻松地修改数据：

```
people.musicians[1].lastName = "Rachmaninov";
```

在将字符串转换为 JavaScript 对象之后，就可以像这样修改变量中的数据。

3.2.4 转换回字符串（见 5.3 小节）

当然，如果不能轻松地将对象转换回文本格式，那么所有数据修改都没有太大的价值。在 JavaScript 中这种转换也很简单：

```
String newJSONtext = people.toJSONString();
```

这样就行了。现在就获得了一个可以在任何地方使用的文本字符串，例如，可以将它用作 Ajax 应用程序中的请求字符串。

更重要的是，可以将*任何* JavaScript 对象转换为 JSON 文本。并非只能处理原来用 JSON 字符串赋值的变量。为了对名为 myObject 的对象进行转换，只需执行相同形式的命令。

```
String myObjectInJSON = myObject.toJSONString();
```

如果使用 JSON，只需调用一个简单的函数，就可以获得经过格式化的数据，可以直接使用了。

最终结论是：如果要处理大量 JavaScript 对象，那么 JSON 几乎肯定是一个好选择，这样就可以轻松地将数据转换为可以在请求中发送给服务器端程序的格式。

3.3 将 JSON 发给服务器

将 JSON 发给服务器并不难，但却至关重要，而且还有一些重要的选择要做。但是，一旦决定使用 JSON，所要做的这些选择就会十分简单而且数量有限，所以你需要考虑的关注

的事情不多，重要的是能够将 JSON 字符串发送给服务器，而且最好能做到尽快和尽可能简单。

3.3.1 通过 GET 以名称/值对发送 JSON

将 JSON 数据发给服务器的最简单方法是将其转换成文本，然后以名称/值对的值的方式进行发送。请务必注意，JSON 格式的数据是相当长的一个对象，看起来可能会如清单 1 所示：

```
var people =  { "programmers": [    { "firstName": "Brett", "lastName":"McLaughlin",
"email": "brett@newInstance.com" },    { "firstName": "Jason", "lastName":"Hunter",
"email": "jason@servlets.com" },    { "firstName": "Elliotte", "lastName":"Harold",
"email": "elharo@macfaq.com" }   ],  "authors": [    { "firstName": "Isaac",
"lastName": "Asimov", "genre": "science fiction" },    { "firstName": "Tad",
"lastName": "Williams", "genre": "fantasy" },    { "firstName": "Frank",
"lastName": "Peretti", "genre": "christian fiction" }   ],  "musicians": [
{ "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
{ "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }   ]  }
```

清单 1. JSON 格式的简单 JavaScript 对象

如果要以名称/值对将其发送到服务器端，应该如下所示：

```
var url = "organizePeople.php?people=" + people.toJSONString();
xmlHttp.open("GET", url, true);
xmlHttp.onreadystatechange = updatePage;
xmlHttp.send(null);
```

这看起来不错，但却存在一个问题：在 JSON 数据中会有空格和各种字符，Web 浏览器往往要尝试对其继续编译。要确保这些字符不会在服务器上（或者在将数据发送给服务器的过程中）引起混乱，需要在 JavaScript escape()函数中做如下添加：

```
var url = "organizePeople.php?people=" + escape(people.toJSONString());
request.open("GET", url, true);
request.onreadystatechange = updatePage;
request.send(null);
```

该函数可以处理空格、斜线和其他任何可能影响浏览器的内容，并将它们转换成 Web 可用字符（比如，空格会被转换成%20，浏览器并不会将其视为空格处理，而是不做更改，将其直接传递到服务器）。之后，服务器会（通常自动）再把它们转换回它们传输后的本来“面目”。

这种做法的缺点有两个：

(1) 在使用 GET 请求发送大块数据时，对 URL 字符串有长度限制。虽然这个限制很宽泛，但对象的 JSON 字符串表示的长度可能超出您的想象，尤其是在使用极其复杂的对象时更是如此。

(2) 在跨网络以纯文本发送所有数据的时候，发送数据面临的不安全性超出了您的处理能力。

简言之，以上是 GET 请求的两个限制，而不是简单的两个与 JSON 数据相关的事情。在想要发送用户名和姓之外的更多内容，比如表单中的选择时，二者可能会需要多加注意。若要处理任何机密或极长的内容，可以使用 POST 请求。

3.3.2 利用 POST 请求发送 JSON 数据

当决定使用 POST 请求将 JSON 数据发送给服务器时，并不需要对代码进行大量更改，如下所示：

```
var url = "organizePeople.php?timeStamp=" + new Date().getTime();
request.open("POST", url, true);
request.onreadystatechange = updatePage;
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
request.send(people.toJSONString());
```

请求使用 POST 而非 GET 打开，而且 Content-Type 头被设置为让服务器预知它能得到何种数据。在这种情况下，即为 application/x-www-form-urlencoded，它让服务器知道现在发送的是文本，正如它从常规的 HTML 表单中得到的一样。

另一个简单提示是 URL 的末尾追加了时间。这就确保了请求不会在它第一次被发送后即缓存，而是会在此方法每次被调用后重新创建和重发；此 URL 会由于时间戳的不同而稍微有些不同。这种技巧常被用于确保到脚本的 POST 每次都会实际生成新请求且 Web 服务器不会尝试缓存来自服务器的响应。

3.3.3 JSON 就只是文本

不管使用 GET 还是 POST，关键之处在于 JSON 就只是文本。由于不需要特殊编码而且每个服务器端脚本都能处理文本数据，所以可以轻松利用 JSON 并将其应用到服务器。假如 JSON 是二进制格式的或是一些怪异的文本编码，情况就不这么简单了；幸好 JSON 只是常规的文本数据（正如脚本能从表单提交中所接收到的数据，在 POST 段和 Content-Type 头中亦可以看出），所以在将数据发送到服务器时无需太费心。

3.4 在服务器上解释 JSON

一旦你编写完客户端 JavaScript 代码、允许用户与 Web 表单和 Web 页的交互、收集发送给服务器端程序以做处理所需的信息，此时，服务器就成为了应用程序（如果调用了异步使用的服务器端程序，则可能是我们认为的所谓的“Ajax 应用程序”）中的主角。在此时，您在客户端所做的选择（比如使用 JavaScript 对象，然后将其转换成 JSON 字符串）必须要与服务器端的选择相匹配，比如使用哪个 API 解码 JSON 数据。

3.4.1 处理 JSON 的两个步骤

不管在服务器端使用何种语言，在服务器端处理 JSON 基本上就需要两个步骤。

（1） 针对编写服务器端程序所用的语言，找到相应的 JSON 解析器/工具箱/帮助器 API。

（2） 使用 JSON 解析器/工具箱/帮助器 API 取得来自客户机的请求数据并将数据转变成脚本能理解的东西。

以上差不多就是目前所应了解的大致内容了。接下来，我们对每个步骤进行较为详细的

介绍。

3.4.2 寻找 JSON 解析器

寻找 JSON 解析器或工具箱最好的资源是 JSON 站点。在这里，除了可以了解此格式本身的方方面面之外，还可以通过各种链接找到 JSON 的各种工具和解析器，从 ASP 到 Erlang，到 Pike，再到 Ruby，应有尽有。您只需针对自己编写脚本所用的语言下载相应的工具箱即可。为了让服务器端脚本和程序能够使用此工具箱，可以根据情况对其进行选择、扩展或安装（如果在服务器端使用的是 C#、PHP 或 Lisp，则可变性更大）。

例如，如果使用的是 PHP，可以简单将其升级至 PHP 5.2 并用它完成操作；在 PHP 这个最新版本默认包含了 JSON 扩展。实际上，那也是在使用 PHP 时处理 JSON 的最好方法。如果使用的是 Java servlet，json.org 上的 org.json 包显然就是个不错的选择。在这种情况下，可以从 JSON Web 站点下载 json.zip 并将其中包含的源文件添加到项目构建目录。编译完这些文件后，一切就就绪了。对于所支持的其他语言，同样可以使用相同的步骤；使用何种语言取决于您对该语言的精通程度，最好使用您所熟悉的语言。

3.4.3 使用 JSON 解析器

一旦获得了程序可用的资源，剩下的事就是找到合适的方法进行调用。比如，假设为 PHP 使用的是 JSON-PHP 模板：

```
// This is just a code fragment from a larger PHP server-side script
require_once('JSON.php');

$json = new Services_JSON();

// accept POST data and decode it
$value = $json->decode($GLOBALS['HTTP_RAW_POST_DATA']);

// Now work with value as raw PHP
```

通过该模板，可将获得的所有数据（数组格式的、多行的、单值的或 JSON 数据结构中的任何内容）转换成原生 PHP 格式，放在 $value 变量中。

如果在 servlet 中使用的是 org.json 包，则会使用如下代码：

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

  StringBuffer jb = new StringBuffer();
  String line = null;
  try {
    BufferedReader reader = request.getReader();
    while ((line = reader.readLine()) != null)
      jb.append(line);
  } catch (Exception e) { //report an error }

  try {
    JSONObject jsonObject = new JSONObject(jb.toString());
  } catch (ParseException e) {
    // crash and burn
    throw new IOException("Error parsing JSON request string");
  }

  // Work with the data using methods like...
  // int someInt = jsonObject.getInt("intParamName");
  // String someString = jsonObject.getString("stringParamName");
  // JSONObject nestedObj = jsonObject.getJSONObject("nestedObjName");
  // JSONArray arr = jsonObject.getJSONArray("arrayParamName");
  // etc...
}
```

4 JSON 的优点和不足

对于 JSON，首先要明白 JSON 和 XML 一样也是一种简单文本格式。相对于 XML，它更加易读、更便于肉眼检查。在语法的层面上，JSON 与其他格式的区别是在于分隔数据的字符，JSON 中的分隔符限于单引号、小括号、中括号、大括号、冒号和逗号。下图是一个 JSON 有效负载：

```
{"addressbook": {"name": "Mary Lebow",
"address": {
"street": "5 Main Street"
"city": "San Diego, CA",
"zip": 91912,
},
"phoneNumbers": [
"619 332-3452",
"664 223-4667"
]
}
}
```

将上面的 JSON 有效负载用 XML 改写，如下：

```
<addressbook>
<name>Mary Lebow</name>
<address>
<street>5 Main Street</street>
<city zip="91912"> San Diego, CA </city>
<phoneNumbers>
<phone>619 332-3452</phone>
<phone>664 223-4667</phone>
</phoneNumbers>
</address>
</addressbook>
```

是不是很相似？但它们并不相同。下面将详细阐述采用 JSON 句法的优点和不足。

4.1 优点

乍看上去，使用 JSON 的数据分隔符的优点可能并不那么明显，但存在一个根本性的缘由：它们简化了数据访问。使用这些数据分隔符时， JavaScript 引擎对数据结构（如字符串、数组、对象）的内部表示恰好与这些符号相同。

这将开创一条比 DOM 技术更为便捷的数据访问途径。下面列举几个 JavaScript 代码片段来说明这一过程，这些代码片段会访问先前的 JSON 代码片段中的信息：

（1）访问 JSON 中的名称： addressbook.name

（2）访问 JSON 中的地址： addressbook.address.street

（3）访问 JSON 中的电话号码第一位：addressbook.address.phoneNumbers[0]

JSON 的另一个优点是它的非冗长性。在 XML 中，打开和关闭标记是必需的，这样才能满足标记的依从性；而在 JSON 中，所有这些要求只需通过一个简单的括号即可满足。在包含有数以百计字段的数据交换中，传统的 XML 标记将会延长数据交换时间。目前还没有正式的研究表明 JSON 比 XML 有更高的线上传输效率；人们只是通过简单的字节数比较发现，对于等效的 JSON 和 XML 有效负载，前者总是小于后者。至于它们之间的差距有多大，特别是在新的 XML 压缩格式下它们的差距有多大，有待进一步的研究。

此外，还有一下一些优点：

（1）轻量级的数据交换格式

（2）人们读写更加容易

（3）易于机器的解析和生成

（4） 能够通过 JavaScript 中 eval()函数解析 JSON

（5）JSON 支持多语言。包括：ActionScript, C, C#, ColdFusion, E, Java, JavaScript, ML, Objective CAML, Perl, PHP, Python, Rebol, Ruby, and Lua.

4.2 不足

和许多好东西都具有两面性一样，JSON 的非冗长性也不例外，为此 JSON 丢失了 XML

具有的一些特性。命名空间允许不同上下文中的相同的信息段彼此混合，然而，显然在 JSON 中已经找不到了命名空间。JSON 与 XML 的另一个差别是属性的差异，由于 JSON 采用冒号赋值，这将导致当 XML 转化为 JSON 时，在标识符（XML CDATA）与实际属性值之间很难区分谁应该被当作文本考虑。

另外，JSON 片段的创建和验证过程比一般的 XML 稍显复杂。从这一点来看，XML 在开发工具方面领先于 JSON。

5 JSON 与 XML 的比较

5.1 可读性

JSON 和 XML 的可读性可谓不相上下，一边是建议的语法，一边是规范的标签形式，很难分出胜负。

5.2 可扩展性

XML 天生有很好的扩展性，JSON 当然也有，没有什么是 XML 能扩展，JSON 不能的。

5.3 编码难度

XML 有丰富的编码工具，比如 Dom4j、JDom 等，JSON 也有 json.org 提供的工具，但是 JSON 的编码明显比 XML 容易许多，即使不借助工具也能写出 JSON 的代码，可是要写好 XML 就不太容易了。

5.4 解码难度

XML 的解析得考虑子节点父节点，让人头昏眼花，而 JSON 的解析难度几乎为 0。

5.5 实例比较

XML 和 JSON 都使用结构化方法来标记数据，下面来做一个简单的比较。

现假设有一个用户数据包括：用户名、密码、所在部门、性别、年龄。

用 XML 表示如下：

```
<?xml version="1.0" encoding="utf-8"?>
<user>
<name>张三 </name>
<password>123456</password>
<department>技术部</department>
<sex>男</sex>
<old>30</old>
</user>
```

用 JSON 表示如下：

```
{
"name":"张三",
"password":"123456",
"department":"技术部",
"sex":"男",
"old":"28"
}
```

与 XML 一样，JSON 也是基于文本的，且它们都使用 Unicode 编码，同样具有可读性。XML 比较适合于标记文档，而 JSON 却更适合于时行数据交换处理。

6 JSON 在 js 中的应用示例

6.1 例一

```
1.  function showJSON() {
2.      var user =
3.      {
4.          "username":"andy",
5.          "age":20,
6.          "info": { "tel": "123456", "cellphone": "98765"},
7.          "address":
8.              [
9.                  {"city":"beijing","postcode":"222333"},
```

```
            {"city":"newyork","postcode":"555666"}
        ]
    }

    alert(user.username);
    alert(user.age);
    alert(user.info.cellphone);
    alert(user.address[0].city);
    alert(user.address[0].postcode);
}
```

这表示一个 user 对象，拥有 username, age, info, address 等属性。
同样也可以用 JSON 来简单的修改数据，修改上面的例子

6.2 例二

```
function showJSON() {
    var user =
    {
        "username":"andy",
        "age":20,
        "info": { "tel": "123456", "cellphone": "98765"},
        "address":
            [
                {"city":"beijing","postcode":"222333"},
                {"city":"newyork","postcode":"555666"}
            ]
    }

    alert(user.username);
    alert(user.age);
    alert(user.info.cellphone);
    alert(user.address[0].city);
    alert(user.address[0].postcode);

    user.username = "Tom";
    alert(user.username);
}
```

6.3 例三（json.js 包的使用还有些问题）

JSON 提供了 json.js 包，下载 http://www.json.org/json.js 后，将其引入然后就可以简单的使用 object.toJSONString()转换成 JSON 数据。

```
1.  function showCar() {
2.      var carr = new Car("Dodge", "Coronet R/T", 1968, "yellow");
3.      alert(carr.toJSONString());  //使用该方法时总是有“对象不支持此属性或方法”的错误，尚未找到问题所在；问题出在 json.js 中没有定义 toJSONString()方法
4.  }
5.
6.  function Car(make, model, year, color)      {
7.      this.make  =  make;
8.      this.model  =  model;
9.      this.year  =  year;
10.     this.color  =  color;
11. }
```

可以使用 eval 来转换 JSON 字符到 Object。

```
1.  function myEval() {
2.      var str = '{ "name": "Violet", "occupation": "character" }';
3.      var obj = eval('(' + str + ')');
4.      alert(obj.toJSONString());
5.  }
```

或者使用 parseJSON()方法

```
1.  function myEval() {
2.      var str = '{ "name": "Violet", "occupation": "character" }';
3.      var obj = str.parseJSON();
4.      alert(obj.toJSONString());
5.  }
```

解决上述问题的方法代码如下：

（1）

```
var people =
  { "programmers": [
      { "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },
      { "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },
      { "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }
    ],
  "authors": [
      { "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction" },
      { "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },
      { "firstName": "Frank", "lastName": "Peretti", "genre": "christian fiction" }
    ],
  "musicians": [
      { "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
```

```
    { "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }
  ],toJSONString:function(){return(this.programmers[0].firstName)}
};

function a(){
    alert(people.toJSONString());
}

a()
```

（2）

```
function myEval() {
    var str = { "name": "Violet", "occupation": "character" ,toJSONString:function(){return
this.name}};
    alert(str.toJSONString());
}
```

7 用 JQuery 处理 JSON

7.1 处理从服务器返回的 JSON 数据

1.页面部分

```
<%@ page language="java" contentType="text/html; charset=GBK" pageEncoding="GBK"%>
<%@ include file="/common/taglibs.jsp"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>用户列表</title>
        <%@ include file="/common/meta.jsp"%>
    </head>
    <body>
        <a href="add.do">新增用户</a>
        <br />
        <ul>
            <c:forEach items="${userList}" var="userinfo">
                <li id="user_info_view_${userinfo.id}">
                    id:${userinfo.id} name:${userinfo.name}
namedesc:${userinfo.userDesc}
                    <a href="show.do?id=${userinfo.id}">查看</a>
                    <a href="edit.do?id=${userinfo.id}">修改</a>
```

```
                        <a href="javascript:deleteUser(${userinfo.id})">删除</a>
                    </li>
                </c:forEach>
            </ul>
        </body>
<script type="text/javascript">
function deleteUser(id){
        $.getJSON("delete.do",{"id":id},function(json){
            alert(json.length);
           for(var i=0;i<json.length;i++){
            alert(json[i].id);
            alert(json[i].name);
            alert(json[i].userDesc);
           }
            $('#user_info_view_'+id).remove();
        });
}
</script>
</html>
```

2. 后台处理部分

```
List oneList=userInfoMgr.findAll();
net.sf.json.JSONArray jsonArray = net.sf.json.JSONArray.fromObject(oneList);
response.setCharacterEncoding("utf-8");    response.getWriter().print(jsonArray);
return null;
```

7.2 处理普通 JSON 数据

7.2.1 例一

1 js 代码

```
$.each( { name: "John", lang: "JS" },
        function(i, n){
            alert( "Name: " + i + ", Value: " + n );
        });
```

2.结果

```
Name:name, Value:John
Name:lang, Value:JS
```

7.2.3 例二

1 js 代码

```
$.each( [{ name: "John", lang: "JS" },{ name: "Johnd", lang: "JSd" }],
    function(i, n){
        alert( "Name: " + i + ", Value:name: " + n.name+", lang:"+n.lang );
        });
```

2 结果

```
Name:0, Value: name:John, lang:JS
Name:1, Value: name:Johnd, lang:JSd
```

7.2.4 例三

1 js 代码

```
$.each( { name: {firstName:"John", lastName:"ddd"}, lang: "JS" },
    function(i, n){
        alert( "Name: " + i + ", Value:" + n.firstName+", "+n.lastName );
        });
```

2 结果

```
Name:name, Value:John, ddd
Name:lang, Value:undefined, undefined
```

注：因为 lang 的值不是 JSON 数据

7.2.4 例四

Jquery 结合 Json 控制 Select 下拉框：

```
<html>
    <head>
        <title>jquery 操作 Select</title>
        <script type="text/javascript" src="jquery-1.2.6.min.js"></script>
        <script type="text/javascript">
            $(document).ready(function(){
                var oSheng = $("#sheng");
                var dSheng = {head:[
                    {text:"text - a",id:"a"},
                    {text:"text - b",id:"b"}      ]};
                oSheng.empty();//清空 select 下拉框
                for(var i=0;i<dSheng.head.length;i++){
                    $("<option value='" + dSheng.head[i].id + "'>" + dSheng.head[i].text +
"</option>").appendTo(oSheng)//动态添加 Option 子项      }
                    var dShi = {head:[
                        {id:"sub - a - 1",value:"a1",parentid:"a"},
                        {id:"sub - a - 2",value:"a2",parentid:"a"},
                        {id:"sub - b - 1",value:"b1",parentid:"b"},
                        {id:"sub - b - 2",value:"b2",parentid:"b"},
                        {id:"sub - b - 3",value:"b3",parentid:"b"}
```

```
                ]};
                var oShi = $("#shi");
                oSheng.change(function(){//添加 onchange 事件
                oShi.empty();//清空下级下拉框
                for(i=0;i<dShi.head.length;i++){
                    if (dShi.head[i].parentid==oSheng.val()){
                        $("<option value='" + dShi.head[i].value + "'>" +
dShi.head[i].id + "</option>").appendTo(oShi)
                    }
                }
            }
        )}})
    </script>
</head>
<body>
    <select id="sheng">
        <option value="a1">--------</option>
    </select>
    <select id="shi">
        <option value="b1">????????</option>
    </select>
</body>
</html>
```

7.2.5 例五

1 js 代码

```
jQuery.extend(
 {
  /**
   * @see  将 json 字符串转换为对象
   * @param    json 字符串
   * @return 返回 object,array,string 等对象
   */
  evalJSON : function (strJson)
  {
   return eval( "(" + strJson + ")");
  }
 });
 jQuery.extend(
 {
  /**
   * @see  将 javascript 数据类型转换为 json 字符串
   * @param 待转换对象,支持 object,array,string,function,number,boolean,regexp
```

```
 * @return 返回 json 字符串
 */
toJSON : function (object)
{
 var type = typeof object;
 if ('object' == type)
 {
  if (Array == object.constructor)
   type = 'array';
  else if (RegExp == object.constructor)
   type = 'regexp';
  else
   type = 'object';
 }
     switch(type)
 {
          case 'undefined':
       case 'unknown':
   return;
   break;
  case 'function':
       case 'boolean':
  case 'regexp':
   return object.toString();
   break;
  case 'number':
   return isFinite(object) ? object.toString() : 'null';
       break;
  case 'string':
   return "" + object.replace(/(\\|\")/g,"\\$1").replace(/\n|\r|\t/g,
       function(){
                   var a = arguments[0];
        return    (a == '\n') ? '\\n':
                         (a == '\r') ? '\\r':
                         (a == '\t') ? '\\t': ""
              }) + "";
   break;
  case 'object':
   if (object === null) return 'null';
       var results = [];
       for (var property in object) {
          var value = jQuery.toJSON(object[property]);
          if (value !== undefined)
             results.push(jQuery.toJSON(property) + ':' + value);
```

```
            }
            return '{' + results.join(',') + '}';
        break;
       case 'array':
        var results = [];
            for(var i = 0; i < object.length; i++)
        {
          var value = jQuery.toJSON(object[i]);
                if (value !== undefined) results.push(value);
        }
            return '[' + results.join(',') + ']';
        break;
          }
   }
 });


var obj = {
     name : "sean",
     friend : ["fans","bruce","wawa"],
     action : function(){alert("gogogog")},
     boy    : true,
     age : 26,
     reg : /\b([a-z]+) \1\b/gi,
     child : {
      name : "none",
      age : -1
     }
   };

   var json = $.toJSON(obj);
   alert(json);
   var objx = $.evalJSON(json);
   alert(objx);
```

2 结果

{name : "sean", friend : ["fans","bruce","wawa"], action : function(){alert("gogogog")}, boy : true, age : 26, reg : /\b([a-z]+) \1\b/gi, child : { name : "none", age : -1 } }

[object, Object]