javaScript系列 [09]-javaScript和JSON (拓展)

> 本文输出JSON搜索和JSON转换相关的内容，是对前两篇文章的补充。

JSON搜索

在特定的开发场景中，如果服务器端返回的JSON数据异常复杂(可能超过上万行)，那么必然就有对JSON文档进行搜索的需求。在对JSON文档进行搜索的时候，建议使用专业的JSON搜索类库和工具来实现，这可以极大的简化JSON文档搜索的工作并降低工作难度。

JSON搜索的具体适用场景：对于某次API调用，我们只需要其中的部分数据，这种情况我们就可以根据某个标准来对返回的JSON内容进行搜索和过滤。

本文将会先后介绍多款处理JSON搜索的类库(工具)，包括但不限于 `JSONPath` 、 `JSON Pointer` 、 `jq` 等，在对各种方案进行介绍的时候将会从**方案的优缺点**、**具体的使用方式**等角度切入，开发中可以根据实际的开发场景和各工具自身的特点来进行选择。

工具001 → jq

`jq` 是一个提供了命令行界面的JSON搜索工具。

`功能` 可使用自身特定的查询语法来过滤JSON和截取数组，类似于JSON中的sed。

`生态` 除命令行外，拥有优秀的基于Web的jq测试器，甚至Node社区还以npm模块形式发布了教程。

`优势`

- 提供丰富的搜索和过滤功能。
- 大多数编程语言都对jq提供良好的支持。
- jq相关文档质量较高，且拥有友好的交互式教程。
- 拥有优秀的在线测试工具，能够对查询提供快速反馈。
- 在命令行中能够很好的与cURL以及管道操作等协同工作。
- 使用C语言编写的，没有运行时依赖，可运行在Linux，OS X和Windows等平台。

`资料` 官网 、Github仓库 、 Ubuntu-jq手册 、 Hyperpolyglot JSON工具、Jackson类库、Ruby-jq gem

`语法`

```
======基本语法======
.       输出所有的文档内容。
|       管道符，传递数据流。
.key    输出指定key对应的部分文档。
.[ ]    输出数组中指定索引对应的元素。

======查询语法示例========
.person[0]        获取JSON文档person数组中的第一个元素。
.person[-1]       获取JSON文档person数组中的最后一个元素。
```

```
.person[0:3]     获取JSON文档person数组中的前面三个元素。
.person[] | select (.age>20 ) 获取JSON文档中满足要求(age > 20)的所有数据。
```

安装

- OSX系统 建议使用Homebrew来安装，具体命令为： `$ brew install jq`
- windows系统 建议使用Chocolatey NuGet来安装，具体命令为： `$ chocolatey install jq`
- 当然也可以通过 `git clone` 仓库源码来进行安装，具体细节以及其它系统处理请参考Download jq

这里给出OSX系统中通过命令行安装 `jq` 的具体细节和示例。

BASH

```
wendingding$ brew install jq
Updating Homebrew...
==> Installing dependencies for jq: oniguruma
==> Installing jq dependency: oniguruma
==> Downloading https://homebrew.bintray.com/bottles/oniguruma-6.8.2.high_sierra
######################################################################## 100.0%
==> Pouring oniguruma-6.8.2.high_sierra.bottle.tar.gz
🍺  /usr/local/Cellar/oniguruma/6.8.2: 17 files, 1.2MB
==> Installing jq
==> Downloading https://homebrew.bintray.com/bottles/jq-1.5_3.high_sierra.bottle
######################################################################## 100.0%
==> Pouring jq-1.5_3.high_sierra.bottle.tar.gz
🍺  /usr/local/Cellar/jq/1.5_3: 19 files, 946.6KB

wendingding$ jq --version
jq-1.5
```

> ! 在安装jq的时候，如果命令行报Error: Failure while executing: git config –local –replace-all homebrew.private true错误，可以尝试先执行 `$ xcode-select --install` 命令然后重新安装。在安装的时候如果总是卡在Updating Homebrew...可以 `control + C` 停止更新。

工具(jq-tutorial)

jq-tutorial是node社区以npm模块的形式发布的jq教程，是学习jq使用的一个比较好用的工具，这里简单列出该模块的安装和使用示例，并对命令进行简单的说明。

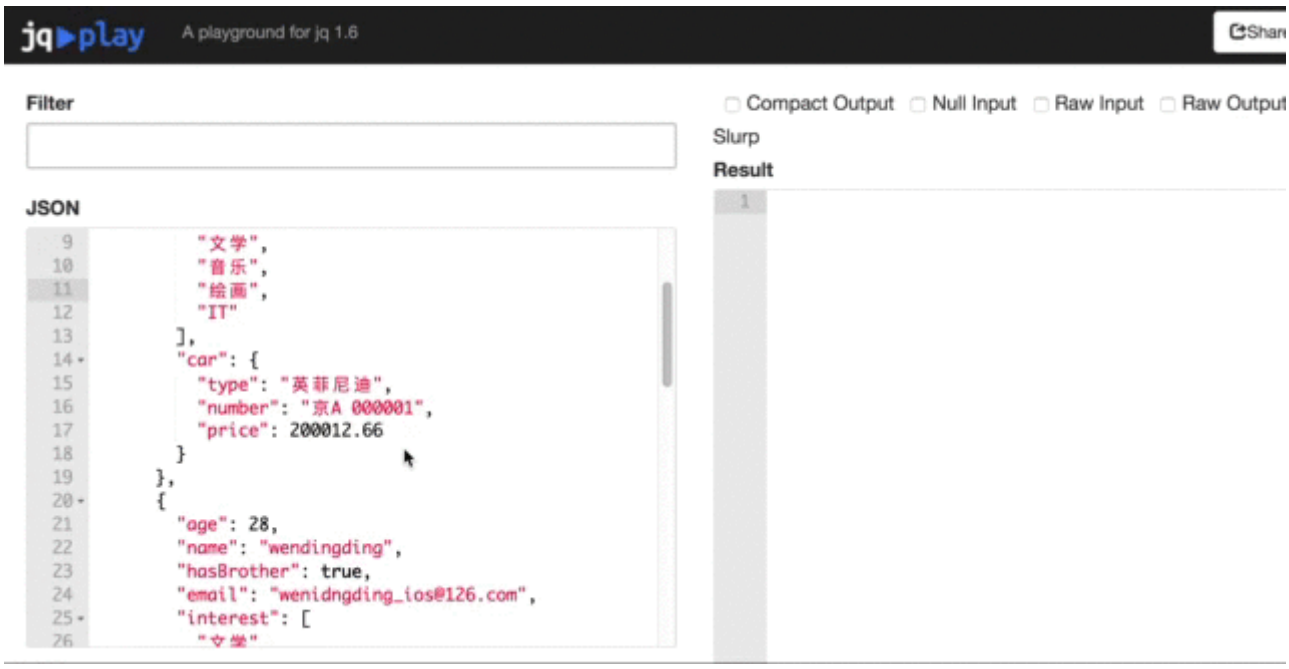
BASH

```
wendingding$ npm search jq-tutorial
NAME                      | DESCRIPTION          | AUTHOR          | DATE
jq-tutorial               | Exercises for…       | =rjz            | 2016-09-29

wendingding$ npm install -g jq-tutorial
npm WARN notice [SECURITY] lodash has the following vulnerability: 1 low.
Go here for more details: https://nodesecurity.io/advisories?search=lodash version=2.4.2
-Run `npm i npm@latest -g` to upgrade npm version, and then `npm audit` to get more info.
/usr/local/bin/jq-tutorial -> /usr/local/lib/node_modules/jq-tutorial/bin/jq-tutorial
+ jq-tutorial@0.0.5
added 4 packages in 20.012s

wendingding$ jq-tutorial
```

```
wendingding$ jq tutorial
Run jq-tutorial with one of the following:
  * pick
  * objects
  * mapping
  * filtering
  * output
  * reduce

wendingding$ jq-tutorial pick
Pick
=======================================
### Pick fields from an object
`jq` retrieves named properties from objects by using `.` syntax:
    $ echo '{"foo": { "bar": "a value" }}' | jq .foo
Nested values are accessible as well:
    $ echo '{"foo": { "bar": "a value" }}' | jq .foo.bar
### Pick elements from an array:
Elements in an array may be extracted by index:
    $ echo '["snap","crackle","pop"]' | jq .[1]

More than one index? No problem!
    $ echo '["snap","crackle","pop"]' | jq .[1, 2]
We can even extract *all* elements at once by omitting the indices:
    $ echo '["snap","crackle","pop"]' | jq .[]
type "data?" to see dataset or "help?" for more options
---------------------------------

Given:    'product' (type "data?" to view)
Challenge: Select the entire item (hint: don't overthink this!):
>
```

工具(jqPlay)

jqPlay是一个基于web的jq在线测试游乐场，它提供了对JSON数据进行jq查询的基本功能，而且提供了简单的jq查询语法示例，能够对查询进行快速反馈。

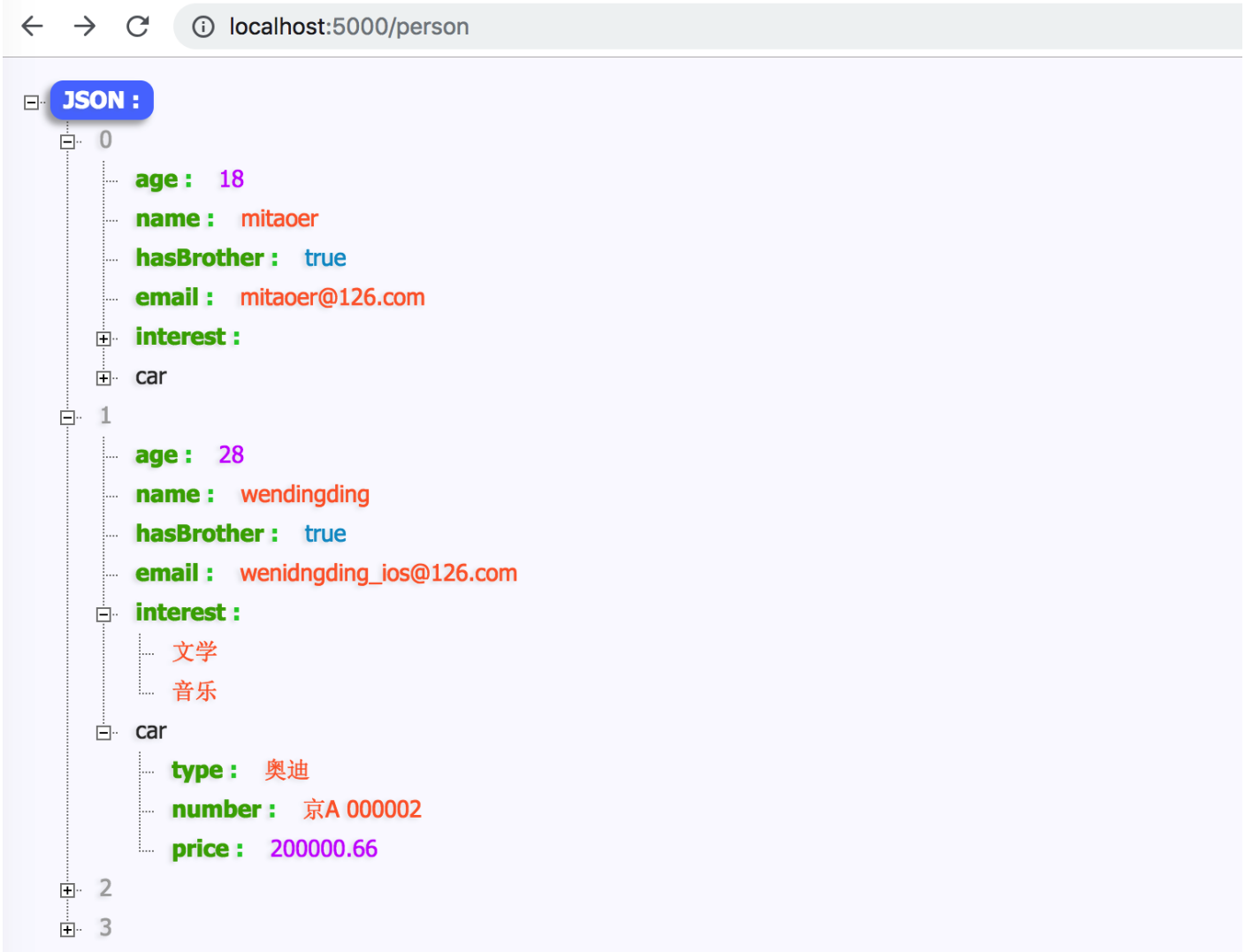


基本用法演示

这里我先提供一个稍复杂的JSON数据,数据保存在 `/JSON-Demo/data.json路径` (您可以点击此链接获取该数据)。为了演示方便，这里我将会把该文档的数据部署为RESTful API，从而创建一个模拟的API服务。在具体的处理中，将使用到名为json-server的Node模块，下面列出具体的细节。

```bash
wendingding$ pwd
/Users/文顶顶/Desktop/JSON-Demo

wendingding$ npm install -g json-server
/usr/local/bin/json-server -> /usr/local/lib/node_modules/json-server/bin/index.js
+ json-server@0.14.0
added 223 packages in 23.03s

wendingding$ json-server -p 5000 ./data.json

  \{^_^}/ hi!

  Loading ./data.json
  Done

  Resources
  http://localhost:5000/person

  Home
  http://localhost:5000

  Type s + enter at any time to create a snapshot of the database
GET /person 304 16.355 ms - -
```

执行 `$json-server -p 5000 ./data.json` 命令之后，我们可以在浏览器中通过http://localhost:5000/person地址来访问JSON文档中的数据，下面是显示结果。



备注：如果经常需要通过浏览器访问和显示JSON数据，建议安装相应的JSON扩展插件，我电脑Chrome安装的是JSON-handle，同类型的还有JSONView。

下面列出jq命令行工具的使用示例以及主要命令行的解读说明。

BASH

```
wendingding$ curl http://localhost:5000/person | jq '.'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   169k      0 --:--:-- --:--:-- --:--:--  255k
[
  {
    "age": 18,
    "name": "mitaoer",
    "hasBrother": true,
    "email": "mitaoer@126.com",
    "interest": [
      "文学",
      "音乐",
      "绘画",
      "IT"
    ],
    "car": {
      "type": "英菲尼迪",
      "number": "京A 000001",
      "price": 200012.66
    }
```

```
  },
  {
    "age": 28,
    "name": "wendingding",
    "hasBrother": true,
    "email": "wenidngding_ios@126.com",
    "interest": [
      "文学",
      "音乐"
    ],
    "car": {
      "type": "奥迪",
      "number": "京A 000002",
      "price": 200000.66
    }
  },
  {
    "age": 23,
    "name": "xiaxiaoxia",
    "hasBrother": true,
    "email": "mitaoer@126.com",
    "interest": [
      "文学",
      "IT"
    ],
    "car": null
  },
  {

    "age": 24,
    "name": "LiuY",
    "hasBrother": true,
    "email": "LiuY@126.com",
    "interest": [
      "文学",
      "音乐",
      "绘画",
      "IT",
      "阅读",
      "健身"
    ],
    "car": {
      "type": "ATS",
      "number": "京A 000003",
      "price": 888888.66
    }
  }
]

wendingding$ curl http://localhost:5000/person | jq '.[1]'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   140k      0 --:--:-- --:--:-- --:--:--  255k
{
  "age": 28,
  "name": "wendingding",
  "hasBrother": true,
  "email": "wenidngding_ios@126.com",
  "interest": [
```

```
   "interest": [
     "文学",
     "音乐"
   ],
   "car": {
     "type": "奥迪",
     "number": "京A 000002",
     "price": 200000.66
   }
}

wendingding$ curl http://localhost:5000/person | jq '.[1].name'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   171k      0 --:--:-- --:--:-- --:--:--  255k
"wendingding"

wendingding$ curl http://localhost:5000/person | jq '.[1].email'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   106k      0 --:--:-- --:--:-- --:--:--  127k
"wenidngding_ios@126.com"

wendingding$ curl http://localhost:5000/person | jq '.[1] | {name,age}'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   161k      0 --:--:-- --:--:-- --:--:--  204k
{
  "name": "wendingding",
  "age": 28
}

wendingding$ curl http://localhost:5000/person | jq '.[1] | {newName:.name,newAge:.age}'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   153k      0 --:--:-- --:--:-- --:--:--  204k
{
  "newName": "wendingding",
  "newAge": 28
}

wendingding$ curl http://localhost:5000/person | jq '.[] | select (.age >=24)'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   167k      0 --:--:-- --:--:-- --:--:--  255k
{
  "age": 28,
  "name": "wendingding",
  "hasBrother": true,
  "email": "wenidngding_ios@126.com",
  "interest": [
    "文学",
    "音乐"
  ],
  "car": {
    "type": "奥迪",
    "number": "京A 000002",
    "price": 200000.66
  }
```

```
  }
}
{
  "age": 24,
  "name": "LiuY",
  "hasBrother": true,
  "email": "LiuY@126.com",
  "interest": [
    "文学",
    "音乐",
    "绘画",
    "IT",
    "阅读",
    "健身"
  ],
  "car": {
    "type": "ATS",
    "number": "京A 000003",
    "price": 888888.66
  }
}

wendingding$ curl http://localhost:5000/person | jq '.[1,3] | {name,email}'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   162k      0 --:--:-- --:--:-- --:--:--  255k
{
  "name": "wendingding",
  "email": "wenidngding_ios@126.com"
}
{
  "name": "LiuY",
  "email": "LiuY@126.com"
}

wendingding$ curl http://localhost:5000/person | jq '.[1,3] | [{name,email}]'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   165k      0 --:--:-- --:--:-- --:--:--  255k
[
  {
    "name": "wendingding",
    "email": "wenidngding_ios@126.com"
  }
]
[
  {
    "name": "LiuY",
    "email": "LiuY@126.com"
  }
]
wendingding$ touch test.json

wendingding$ curl http://localhost:5000/person | jq '.[1,3] | [{name,email}]' > test.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1047  100  1047    0     0   143k      0 --:--:-- --:--:-- --:--:--  204k
wendingding$ cat test.json
[
```

```
L
  {
    "name": "wendingding",
    "email": "wenidngding_ios@126.com"
  }
]
[
  {
    "name": "LiuY",
    "email": "LiuY@126.com"
  }
]
```

主要命令说明[为方便阅读，命令行中的xxx均代表的是http://localhost:5000/person路径]

`$ curl xxx | jq '.'` 获取API返回的所有JSON数据。
`$ curl xxx | jq '.[1]'` 获取JSON文档中person数组的第2个元素内容。
`$ curl xxx | jq '.[1].name'` 获取JSON文档中person数组的第2个元素(对象)中的name属性值。
`$ curl xxx | jq '.[1] | {name,age}'` 获取数组第2个元素(对象)的name和age键值对组成新对象。
`$ curl xxx | jq '.[1,3] | {name,email}'` 获取数组第2和第4个元素中的name和email值组成对象。
`$ curl xxx | jq '.[] | select (.age >=24)'` 获取JSON文档中所有age属性值>=24的对象元素集合。

补充 如果需要在Node中使用jq,那么可以安装并使用require导入node-jq模块。

工具002 → JSONPath

定位 是一款可用于对JSON文档进行搜索和数据提取操作类库。
历史 由Stefan Goessner于2007年开发，最开始的版本使用JavaScript实现。
功能 能够对JSON文档进行搜索和数据提取，它的查询语法基于XPath实现。
生态 JSONPath没有提供命令行操作实现，但提供了优秀的在线测试工具和Node模块。
优势

- 具有丰富的查询语法。
- 查询语句可以返回文档中的多个元素。
- 大多数的主流平台都对JSONPath提供支持。
- 拥有很高的社区使用率，优秀的在线测试工具对开发者更友好。

资料 jsonpath Node模块 、在线测试网站 、Github仓库、Stefan Goessner主页

语法	描述
`$`	根节点
`@`	当前节点的筛选器属性处理
`*`	通配符，匹配任何属性名称
`..`	通配符，匹配任意层次的节点
`[]`	迭代器标示，同数组索引
`[,]`	迭代器标示，迭代器多选

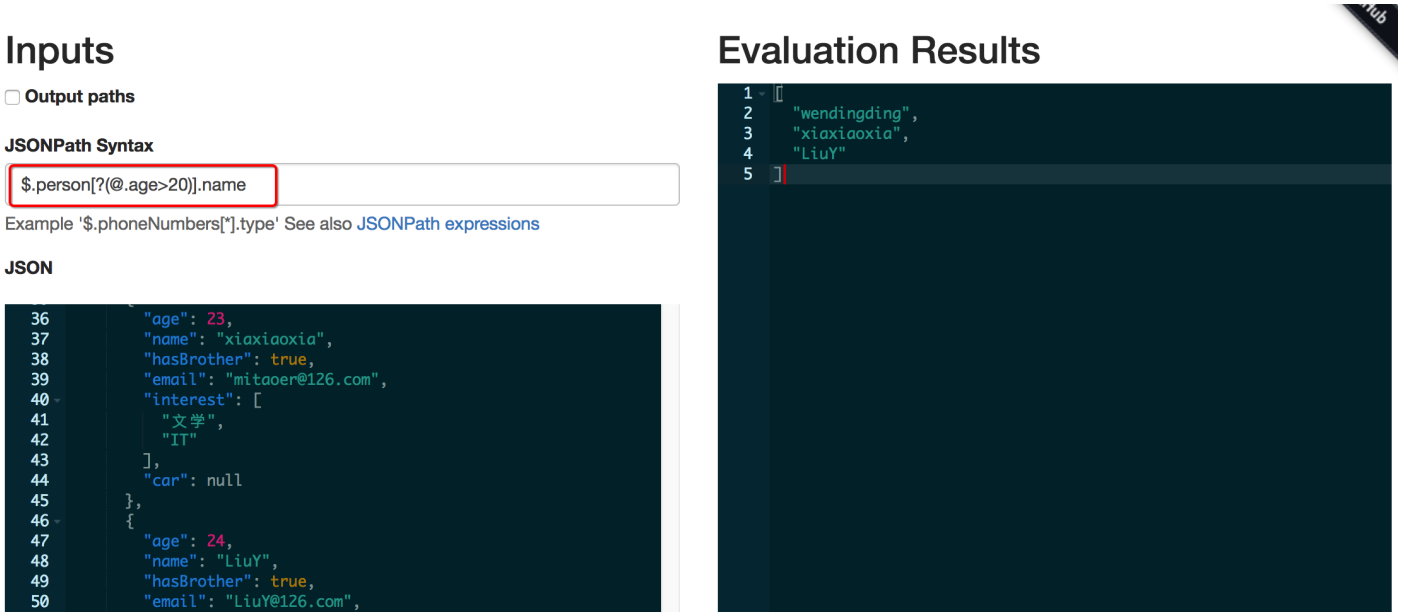
语法	描述
[start:end]	数组切片运算符
[?(<expression>)]	过滤表达式，求值必须为布尔值

查询语法示例

```HTML
//备注：参考的JSON数据为前文使用的data.json文件
$                          获取整个JSON文档的内容
$.person                   获取JSON文档中person数组的内容
$.person.length            获取JSON文档中person数组的长度(元素个数)

$.person[0]                获取JSON文档中person数组第一个元素的内容
$.person[:1]               获取JSON文档中person数组第一个元素的内容
$.person[-1:]              获取JSON文档中person数组最后一个元素的内容
$.person[(@.length-1)]     获取JSON文档中person数组最后一个元素的内容

$.person[:2]               获取JSON文档中person数组前两个元素的内容
$.person[0,3]              获取JSON文档中person数组第1和4第个元素的内容
$.person[0::2]             获取JSON文档中person数组指定元素的内容(隔一个元素抽取)
$.person[:2].name          获取JSON文档中person数组前两个元素中的name值

$..name                    获取JSON文档中所有的name子元素内容
$.person[?(@.age >23)]     获取JSON文档中age值大于23的所有元素
$.person[?(@.age >23)].age   获取JSON文档中age值大于23的所有元素中的age值信息
$.person[?(@.age >20 && @.interest.length == 2)].name 满足多个条件的筛选
```

工具(jsonpath在线测试网站)

jsonpath提供了对应的在线测试网站，给指定JSON文档输入对应的jsonpath查询语句能够快速看到最终效果。使用该测试工具来可以”重量级的”复杂JSON数据进行快速的筛选，输入查询语句后马上就能够在右侧看到查询后的结果。如果开发者原本不了解查询语法，那也可以通过该工具来快速的学习，下面给出简单的图示。



> **注意** jsonpath在线测试工具在使用的时候，总是会把查询的结果保存到[ ]的结构中。

工具(node模块jsonpath)

JSONPath本身没有命令行工具，除了上面介绍的在线测试网站之外，我们还能在代码中使用node社区发布的jsonpath模块实现JSON的搜索任务。下面给出一个简单的单元测试示例(列出源码和执行情况)。

001 先列出单元测试相关的代码

```
//jsonpath-test.js文件内容
var unirest  = require("unirest");
var jsonPath = require("jsonpath");
var expect   = require("chai").expect;

describe("wendingding-test",function(){
    var request ;
    beforeEach(function(){
        request = unirest.get("http://localhost:5000/person")
        .header("Accept","application/json");
    })

    it("return 200 状态码",function(done){
        request.end(function(response){
            expect(response.statusCode).to.eql(200);
            expect(response.headers["content-type"])
            .to.eql("application/json; charset=utf-8");
            done();
        })
    })

    it("return 所有的JOSN数据",function(done){
        request.end(function(response){
            expect(response.body.length).to.eql(4);
            done();
        })
    })

    it("return 所有的JOSN数据中第一个元素       -- $[1]",function(done){
        request.end(function(response){
            var resultData = jsonPath.query(response.body,"$[1]");
            expect(resultData[0].name).to.eql("wendingding");
            done();
        })
    })

    it("return 所有的JOSN数据中最后一个元素[1] -- $[-1:]",function(done){
        request.end(function(response){
            var resultData = jsonPath.query(response.body,"$[-1:]");
            expect(resultData[0].email).to.eql("LiuY@126.com");
            done();
        })
    })

    it("return 所有的JOSN数据中最后一个元素[2] -- $[(@.length-1)]",function(done){
```

```
        request.end(function(response){
            var resultData = jsonPath.query(response.body,"$[(@.length-1)]");
            expect(resultData[0].email).to.eql("LiuY@126.com");
            done();
        })
    })

    it("return 所有的JOSN数据中满足条件元素     -- $[?(@.age >= 23)]",function(done){
        request.end(function(response){
            var data = response.body;
            var resultData = jsonPath.query(data,"$[?(@.age >= 23)]");
            //console.log(resultData)
            expect(resultData.length).to.eql(3);
            for(var i = 0 ;i<resultData.length;i++)
            {
                expect(resultData[i].age).to.be.at.least(23);
            }
            done();
        })
    })
})
```

002 列出代码的具体执行细节

```BASH
wendingding$ pwd
/Users/文顶顶/Desktop/jsonPath-demo
wendingding$ npm test

> jsonpath-demo@1.0.0 test /Users/文顶顶/Desktop/jsonPath-demo
> mocha



  wendingding-test
    ✓ return 200 状态码
    ✓ return 所有的JOSN数据
    ✓ return 所有的JOSN数据中第一个元素       -- $[1]
    ✓ return 所有的JOSN数据中最后一个元素[1] -- $[-1:]
    ✓ return 所有的JOSN数据中最后一个元素[2] -- $[(@.length-1)]
    ✓ return 所有的JOSN数据中满足条件元素     -- $[?(@.age >= 23)]


  6 passing (92ms)
```

003 代码说明

〇 示例代码中使用了Mocha、Unirest测试框架，jsonpath查询模块以及Chai模块中的断言结构。
① 示例代码中的每一个 `it` 就代表着一个测试用例。
② 示例代码中我们在Mocha的 `beforeEach()方法` 中对请求信息进行了配置。
③ 示例代码中在describe语句定义的范围内，每次执行测试用例之前都会先运行一次beforeEach方法。

004 执行备注

这里简单说明上面代码的执行环境和处理过程。

BASH

[1] 在电脑中指定的路径创建文件夹，并通过命令行进入到该路径。

```
$ mkdir JSON-TEST
$ cd JSON-TEST/
$ pwd
    /Users/文顶顶/Desktop/JSON-TEST
```

[2] 初始化并安装必要的node模块。

```
$ npm init  //默认回车即可
$ npm install -g mocha
$ npm install --save-dev mocha
$ npm install --save-dev unirest
$ npm install -save-dev jsonpath
$ npm install --save-dev chai
```

[3] 修改package.json文件中的scripts项为"test": "mocha"。

```
wendingding$ cat package.json
{
  "name": "json-test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "mocha"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "chai": "^4.2.0",
    "jsonpath": "^1.0.0",
    "mocha": "^5.2.0",
    "unirest": "^0.5.1"
  }
}
```

[4] 在当前目录下创建test文件夹，并在该文件夹中创建测试文件(此处命名为json-test.js)。

```
$ mkdir test
$ cd test/
$ touch json-test.js
```

[5] 编辑json-test.js文件的内容(前文已经给出),列出文件目录。

```
.
├── node_modules
|   ├── JSONSelect
|   ...省略
|   ├── chai
|   ├── jsonpath
|   ├── mocha
|   ├── unirest
|   └── xtend
├── package-lock.json
├── package.json
└── test
    └── json-test.js
```

[6] 执行测试。

```
$ npm test
```

工具003 → JSON Pointer

JSON Pointer本身是一项用于获取JSON文档中特定值的标准。JSON Pointer设计的主要目标在于支持JSON Schema标准中的$ref（请参考JSON进阶一文）。

目前，大多数的主流平台(包括Node\Ruby on Rails\Ptyhon\Java等)都已经包含JSON Pointer相关的类库。Java，Jackson已支持JSON Pointer查询语法，javaEE 8将提供原生支持；Node中则可以在社区中找到并使用json-pointer模块来对JSON文档进行处理。

语法	描述
/data	获取JSON文档中某个key对应的所有数据
/data/index	获取指定索引对应的数据
/data/index/key	获取指定索引对应的数据并通过key来取值

JSON Pointer的查询语法简洁高效，其工作机制是以 / 来表示路径分隔，以 索引|下标 来获取指定的内容，索引总是从0开始。下面给出简短的查询语法示例：

```HTML
//备注：参考的JSON数据为前文使用的data.json文件
/person                 获取JSON文档中person的内容
/person/0               获取JSON文档中person数组的第一项内容
/person/0/name          获取JSON文档中person数组的第一项内容中的name值
```

> **注意** JSON Pointer标准中，查询操作的结果总只包含数据值而不会包含相关的键名。

这里将使用node社区的json-pointer模块简单演示node平台中对JSON文档的处理。下面列出核心单元测试代码(在测试的时候需要先安装对应的模块)。

```
var expect = require("chai").expect;
var pointer = require("json-pointer");
var unirest = require("unirest");

describe("wendingding-test",function(){
    var request ;
    beforeEach(function(){
        request = unirest.get("http://localhost:5000/person")
        .header("Accept","application/json");
    })

    it("return 200 状态码",function(done){
        request.end(function(response){
            expect(response.statusCode).to.eql(200);
            expect(response.headers["content-type"])
```

```
            .to.eql("application/json; charset=utf-8");
            done();
        })
    })

    it("return 所有的JOSN数据",function(done){
        request.end(function(response){
            expect(response.body.length).to.eql(4);
            done();
        })
    })

    it("return Person数组中的第一个元素 /person/0",function(done){
        request.end(function(response){
            var resultData = pointer.get(response.body,"/0");
            console.log("\n",resultData,"\n");
            expect(resultData.name).to.eql("mitaoer");
            done();
        })
    })


    it("return Person数组中的第三个元素中interest的值 /person/2/interest",function(done){
        request.end(function(response){
            var resultData = pointer.get(response.body,"/2/interest");
            console.log("\t",resultData);
            expect(resultData.length).to.eql(2);
            done();
        })
    })
})
```

简单列出执行情况：

BASH

```
wendingding$ npm test
> json-pointer-demo@1.0.0 test /Users/文顶顶/Desktop/JSON-Pointer
> mocha

  wendingding-test
    ✓ return 200 状态码
    ✓ return 所有的JOSN数据

      { age: 18,
        name: 'mitaoer',
        hasBrother: true,
        email: 'mitaoer@126.com',
        interest: [ '文学', '音乐', '绘画', 'IT' ],
        car: { type: '英菲尼迪', number: '京A 000001', price: 200012.66 }
      }
    ✓ return Person数组中的第一个元素 /person/0
            [ '文学', 'IT' ]
    ✓ return Person数组中的第三个元素中interest的值 /person/2/interest

  4 passing (66ms)
```

JSON转换

JSON → HTML结构(渲染)

将JSON数据转换(处理)为HTML的操作我们应该都很熟悉，在前端开发和移动端开发领域中，这一部分的操作通常和网络请求紧密联系，业务流程基本都是先发请求获取服务器端返回的(JSON)数据，然后通过序列化的方法来对数据进行解析，也就是反序列化处理(通常是将JSON数据转换为编程语言中对应的数据结构比如数组或者是对象)，最终再根据得到的数据来更新UI。

现在前端开发中这都属于基本操作，甚至像Vue这样类似的框架中 数据绑定 已经是其最最基础的一部分了。虽然如此，为了文章的完整性，这里还是会简单介绍 Mustache 和 Handlebars 两个类库在JSON转换中的运用。

Mustache

简介 Mustache使用声明式模板来转换数据格。

资料 Mustache、Mustache Github、Mustache 5说明文档

优势 使用模板可以从代码中抽取具体的数据信息，并将数据保存在外部的文件中，实现关注点分离。

接下来我将通过一个简短的示例来说明Mustache的语法以及其使用方式，您可以点击该链接获取完整的项目内容。为了对介绍Mustache的使用，这里我列出项目中的部分内容并做简要说明。

001 列出模板核心内容

```HTML
//备注：../Tem-TEST/src/index.mustache文件的核心内容
<div id="app">
    <table id="tb">
        <tr>
            <th>name</th>
            <th>age</th>
            <th>email</th>
            <th>interest</th>
            <th>car-type</th>
            <th>car-number</th>
        </tr>
        {{#person}}
        <tr>
            <td>{{name}}</td>
            <td>{{age}}</td>
            <td>{{email}}</td>
            <td>{{interest.1}}</td>
            {{#car}}
                <td>{{type}}</td>
                <td>{{number}}</td>
            {{/car}}
        </tr>
        {{/person}}
    </table>
</div>
```

002 列出单元测试代码

```
//备注: ../Tem-TEST/test/mustache-test.js文件内容
var fs = require("fs");
var expect = require("chai").expect;
var jsonfile = require("jsonfile");
var mustache = require("mustache");

describe("wendingding-mustache-test",function(){

    //文件的目录结构: json文件路径 + 模板文件路径 + 目标文件路径
    var jsonFileFullPath     = __dirname + "/../src/data.json";
    var templateFileFullPath = __dirname + "/../src/index.mustache";
    var targetFileFunllPath  = __dirname + "/../src/index.html";

    it("JSON -> HTML",function(done){
        jsonfile.readFile(jsonFileFullPath,function(readJsonFileError,jsonData){
            if(!readJsonFileError)
            {
                fs.readFile(templateFileFullPath,"utf8",function(readTemplateFileError,template
                    if(!readTemplateFileError)
                    {
                        var template = templateData.toString();
                        var html = mustache.render(template,jsonData);

                        fs.writeFile(targetFileFunllPath, html,  function(errorStatus) {
                            if (!errorStatus) {
                                console.log("转换成功并保存为HTML文件! ");
                                done();
                            }else
                            {
                                done(errorStatus);
                            }
                         });
                    }else
                    {
                        done(readTemplateFileError)
                    }
                })
            }else
            {
                done(readJsonFileError)
            }
        })
    })
})
```

通过 `$ npm test` 执行单元测试代码，将会执行JSON数组到HTML的转换，处理完毕后结果保存到index.html文件中，下面贴出该页面的效果图。

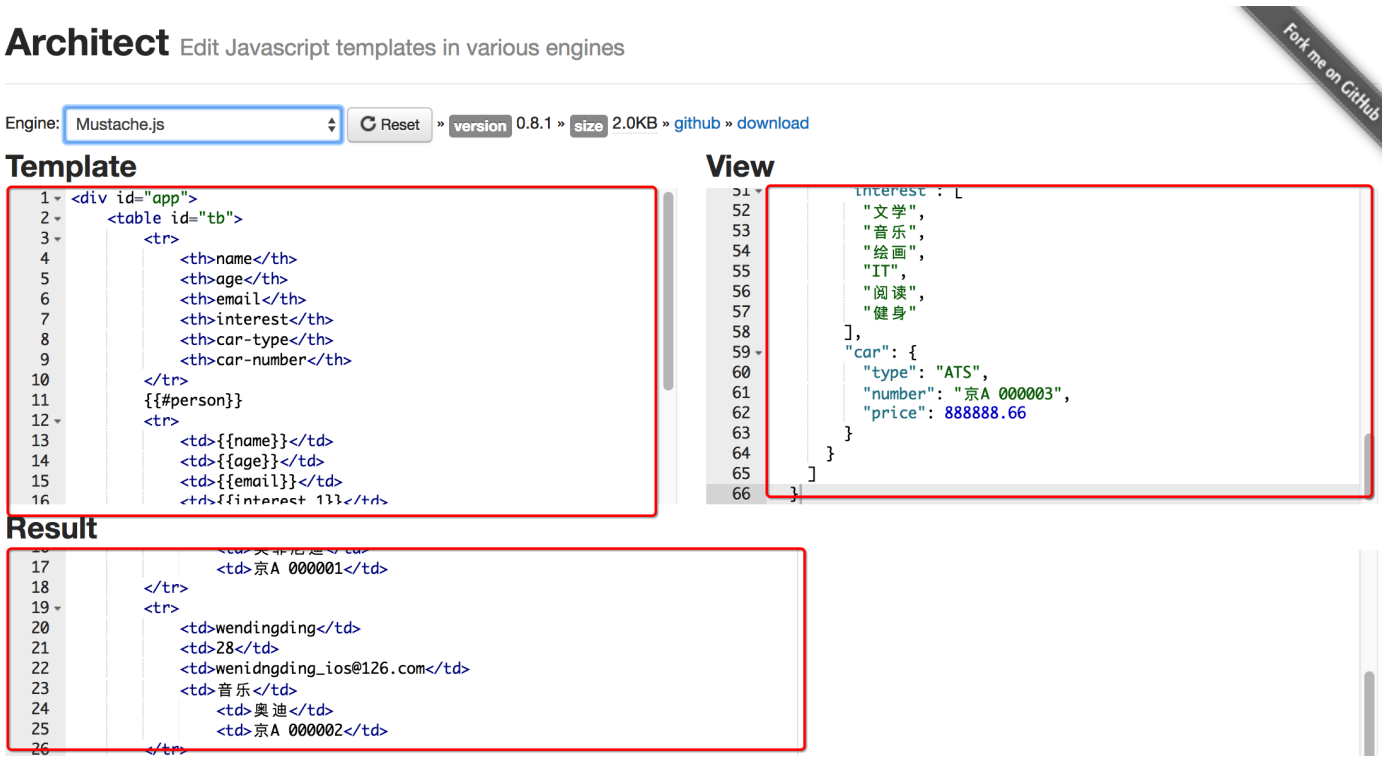
name	age	email	interest	car-type	car-number
mitaoer	18	mitaoer@126.com	音乐	英菲尼迪	京A 000001
wendingding	28	wenidngding_ios@126.com	音乐	奥迪	京A 000002
xiaxiaoxia	23	mitaoer@126.com	IT		
LiuY	24	LiuY@126.com	音乐	ATS	京A 000003

> **Mustache模板工作机制**
>
> ❒ 模板基于HTML, Mustache使用JSON数据来解析标签。
>
> ❒ 模板中的标签可以表示单个字段，使用 双大括号 的形式来包裹。
>
> ❒ 模板中的区块都需要由 开始标签和结尾标签 组成，例如上例中的person。
>
> ❒ 模板中的区块对应JSON数据中的数组或者是对象，例如上例中的person和car。
>
> ❒ 模板中的区块可以为内部的标签定义上下文，比如car区块内部的type和number。

Mustache工具(命令行 && 在线网站)

Mustache除上面演示的使用方式之外，还能直接在命令行中使用，下面给出简短的使用示例。

```bash
(1) 全局安装Mustache模块。
$ npm install -g mustache

(2) 执行mustache命令转换。
$ mustache /Users/文顶顶/Desktop/Tem-TEST/src/data.json /Users/文顶顶/Desktop/Tem-TEST/src/index.

$ open target.html
```



这里再推荐一款好用的在线模板编辑器Architect，使用该工具可以有效的简化测试和开发模板的工作，当修改模板的时候，可以实时的看到渲染的结果。该网站支持多款主流模板引擎(包括**doT.js、Dustjs、EJS、Handlebars.js、Hogan.js、Jade、Mustache、Nunjucks和Underscore.js等**)的编辑和渲染，可以有效的加速开发和调试工作。

Handlebars

简介 Handlebars是Mustache的扩展，使用hash或对象来渲染模板中的标签。

资料 Handlebars、Handlebars Github、Architect、在线测试网站、Handlebars的Node模块

说明 Handlebars与Mustache高度兼容，相对而言Handlebars自身增加了一些特性来增强转换操作。它们的差异主要在于Handlebars提供了 `if` 和 `unless` 等内联的辅助语句且允许开发者通过注册自定义辅助语义的方式来进行扩展，功能更加强大。

优势

- 模板语言丰富，能满足大多数的转换需求。
- 拥有优秀的在线编辑和测试工具用起来更加方便。
- 采用声明式，但也支持在自定义辅助指令中编写逻辑代码。
- 因为拥有内置的条件逻辑，所以在渲染的时候几乎可以不用编写额外的处理代码。
- 跨平台的支持度很好，支持 `JavaScript` 、 `Node.js` 、 `Java` 和 `Ruby on Rails` 等平台。

这里将简单介绍Handlebars在Node中的使用，并提供node和命令行两种执行示例供参考，更多的细节请自行参考其官网文档。

001 列出模板文件的核心代码

HTML

```
//备注(1)：/Users/文顶顶/Desktop/Handlebars-Test/index.hbs文件的核心内容

//备注(2)：转换过程中使用的json数据为前文中的data.json文件
<div id="app">
    <table id="tb">
        <tr>
            <th>name</th>
            <th>age</th>
            <th>email</th>
            <th>car-type</th>
            <th>car-Other</th>
        </tr>
        {{#each person}}
        {{#if car}}
        <tr>
            <td>{{name}}</td>
            <td>{{age}}</td>
            <td>{{email}}</td>
            <td>{{car.type}}</td>
            <td>号码：{{car.number}}  |  价格：{{car.price}}</td>
        </tr>
        {{/if}}
        {{/each}}
    </table>
</div>
```

002 列出命令行执行的细节

```BASH
(1) 先通过命令行全局安装hb-interpolate模块
$ npm install -g hb-interpolate

(2) 执行渲染命令。
$ hb-interpolate -j /Users/文顶顶/Desktop/Handlebars-Test/data.json -t  /Users/文顶顶/Desktop/Han
```

说明 上面的命令行中 `-j` 表示后面跟的是json文件， `-t` 表示后面跟的是模板文件，转换后的结果被输出并保存到target.html文件中。在模板文件中的 `#each` 表示遍历数组， `#if` 是逻辑控制指令，在渲染的时候过滤了car为null的数据情况，浏览器打开target.html文件可以看到下面的显示结果。

name	age	email	car-type	car-Other
mitaoer	18	mitaoer@126.com	英菲尼迪	号码：京A 000001 | 价格：200012.66
wendingding	28	wenidngding_ios@126.com	奥迪	号码：京A 000002 | 价格：200000.66
LiuY	24	LiuY@126.com	ATS	号码：京A 000003 | 价格：888888.66

003 Handlebars在Node中的使用

```
//备注：handlebars-test.js文件的内容
//001 导入node模板
var fs          = require("fs");
var jsonfile    = require("jsonfile");
var handlebars  = require("handlebars");

//002 处理文件路径
var jsonFullPath     = __dirname + "/data.json";
var templateFullPath = __dirname + "/index.hbs";
var outPutFullPath   = __dirname + "/output.html";

//003 读取JSON文件的内容
jsonfile.readFile(jsonFullPath,function(readJsonError,jsonData){
    if(!readJsonError)
    {
        //004 读取模板文件的内容
        fs.readFile(templateFullPath,"utf8",function(readTemplateError,templateData){
            if(!readTemplateError)
            {
                //005 JSON数据 + 模板 = > 渲染
                var template = handlebars.compile(templateData);
                var html = template(jsonData);

                //006 把渲染后的结果保存到指定文件中
                fs.writeFile(outPutFullPath, html,  function(errorStatus) {
                    if(! errorStatus)
                    {
                        console.log("渲染成功！请打开"+outPutFullPath+"查看结果!");
                    }
                })
            }
        })
```

```
    }
})
```

说明 在指定文件目录中(我这里是 `/Users/文顶顶/Desktop/Handlebars-Test`)创建handlebars-test.js文件，并编写上述对应的代码。通过命令行安装必要的Node模块，执行即可得到前文所示的图片结果。下面给出命令行执行的细节：

```BASH
(1) 切换到当前目录
$ cd Handlebars-Test/
$ pwd
/Users/文顶顶/Desktop/Handlebars-Test

(2) 使用npm初始化并安装必要的Node模块。
$ npm init
$ npm install --save-dev jsonfile
$ npm install --save-dev handlebars

(3) 执行。
$ node handlebars-test.js
```

补充 Mustache和Handlebars除用来把JSON转换为HTML(渲染)之外，还能够对JSON数据本身的格式进行转换工作(主要是对JSON数据进行二次处理，譬如删减或结构调整等)，但Mustache在具体进行格式化的时候因为无法确定当前所处理的元素是否为数组或对象的最后一个元素，所以可能存在“无谓逗号”的问题。Handlebars可以通过使用 `#unless 和@last` 的形式对”无谓逗号”的问题进行规避，具体的细节请参考其官方文档说明。另外，JSON格式转换的工具在Node环境中推荐使用Json2Json和jsonapter，也可以参考JSON Patch和JSON-T的实现，这里不再展开。

JSON数据 ←→ XML文档

最后简单介绍JSON数据和XML数据之间的相互转换，虽然这种场景通常可能很少出现(因为开发中常见的场景一般是对JSON或XML数据进行序列化或反序列化处理，JSON和XML两种数据格式之间直接相互转换的情况真的很少见)。

XML全称 `Extensible Markup Language` (可扩展标记语言)，主要流行于(1998~2008年)。同JSON类似，XML也能用于表示和传输数据，现在很多大公司的API都提供XML和JSON两种格式的数据响应。

XML数据和JSON数据的转换难易程度主要看XML文档的结构，如果文档中所有的数据都以XML元素和文本的方式保存那么转换为JSON数据是比较简单的，如果XML文档的元素上存在这大量的属性节点(早年的时候很多XML的设计人员把数据保存在XML的属性节点上，这样有助于减少文件体积和简化多平台之间的转换工作)，那么这种转换就会比较困难。不过，好在我们可以使用很多现成的工具(譬如：Parker和JsonML以及Badgerfish等)来完成这种具体的转换工作。

对于上面这些工具的具体使用情况，大家可以自行了解。需要说明的是即便如此，这些工具仍然存在很大的局限性(譬如 `文档不全` 、 `缺乏跨平台的支持` 和 `完整实现` 以及 `有损转换` 等等)。所以，在实际的使用过程中其实可以考虑先把JSON|XML转换为当前编程语言中的数据结构形式，然后再转换成XML|JSON)。下面以JavaScript(Node)平台为例加以简单说明。

XML文档 → JSON数据

- 先把XML数据解析为JavaScript中的对象|数组(xml2js模块)。
- 把JavaScript的对象|数组序列化为JSON格式的数据(JSON.stringify方法)。

列出核心示例代码

```
var xmlFullPath      = __dirname + "/data.xml";
fs.readFile(xmlFullPath,"utf8",function(readFileError,xmlData){
    var parser = new xml2js.Parser();
    parser.parseString(xmlData,function(error,xmlObj){
        console.log(JSON.stringify(xmlObj,null,2));
    })
})
```

JSON数据 → XML文档

- 先将JSON数据解析(反序列化)为JavaScript中的对象|数组数据(eval函数或者是JSON.parse方法)。
- 根据JavaScript数据来生成(marshaling)对应的XML文档(xml2js模块)

这里列出json数据转换为xml数据的代码示例。

```
//001 导入node模板
var fs          = require("fs");
var xml2js      = require("xml2js");
var jsonfile    = require("jsonfile");

//002 处理文件路径
var jsonFullPath     = __dirname + "/data.json";
var xmlFullPath      = __dirname + "/data.xml";

//003 读取JSON文件的内容并解析为JavaScript对象(jsonData)
jsonfile.readFile(jsonFullPath,function(readJsonError,jsonData){
    if(!readJsonError)
    {
        //004 创建并返回bulider实例对象
        var bulider = new xml2js.Builder();
        //005 使用bulider对象将jsonData转换为xml格式的字符串
        var xml = bulider.buildObject(jsonData);
        //006 写文件操作(把最终的数据保存到指定的文件中)
        fs.writeFile(xmlFullPath, xml,  function(errorStatus) {
            if(! errorStatus)
            {
                console.log("json->XML 转换成功");
                console.log(xml);
            }
        })
    }
})
```

列出用于转换的初始json数据。

JSON

```
{
    "person": [
      {
```

```
        {
          "age": 18,
          "name": "mitaoer",
          "hasBrother": true,
          "email": "mitaoer@126.com",
          "car": {
            "type": "英菲尼迪",
            "number": "京A 000001",
            "price": 200012.66
          }
        },
        {
          "age": 28,
          "name": "wendingding",
          "hasBrother": true,
          "email": "wenidngding_ios@126.com",
          "car": {
            "type": "奥迪",
            "number": "京A 000002",
            "price": 200000.66
          }
        }
    ]
}
```

列出最终输出的xml数据内容。

XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<person>
  <age>18</age>
  <name>mitaoer</name>
  <hasBrother>true</hasBrother>
  <email>mitaoer@126.com</email>
  <car>
    <type>英菲尼迪</type>
    <number>京A 000001</number>
    <price>200012.66</price>
  </car>
  <age>28</age>
  <name>wendingding</name>
  <hasBrother>true</hasBrother>
  <email>wenidngding_ios@126.com</email>
  <car>
    <type>奥迪</type>
    <number>京A 000002</number>
    <price>200000.66</price>
  </car>
</person>
```

! 注意：如果JSON数据中存在数组这种结构那么使用xml2js模块的处理其实不甚理想，更新信息请参考官方文档说明。

> **后记**！ MD，这篇文章写了我好久中间一度放弃，玩了两天农药。好在，我终于完成了。