FREE eBook

LEARNING JSON

JSON HAND BOOK FOR BEGINNERS

#json

Table of Contents

Chapter 1: Getting started with JSON

Remarks

JSON (JavaScript Object Notation) is one of the most popular and widely accepted data exchange format originally specified by Douglas Crockford.

It is currently described by two competing standards, RFC 71592 and ECMA-404. The ECMA standard is minimal, describing only the allowed grammar syntax, whereas the RFC also provides some semantic and security considerations.

- JSON is widely accepted in the softwares that includes client-server architecture for exchanging data between client and server.
- JSON is easy to use and purely text-based, lightweight, and human- readable format and people often misunderstand as replacement of XML.
- Although the abbreviation starts with JavaScript, JSON is not a language or have any language literals it just a specification for notation of data.
- It is platform and language independent and inbuilt supported by almost all of the front line languages/frameworks like and support for the JSON data format is available in all the popular languages, some of which are C#, PHP, Java, C++, Python, Ruby and many more.
- The official Internet media type for JSON is application/json.
- The JSON file name extension is .json.

Versions

Since JSON haven't got any updates, there is only 1 version of JSON, the link below redirects to the original RFC document, which is RFC 4627.

Version	Release Date
Original	2006-07-28

Examples

JSON Syntax Rules

JSON (JavaScript Object Notation) syntax is based on a subset of JavaScript (see also json.org).

A valid JSON expression can be one of the following data types

- simple data types: String, Number, Boolean, Null
- composite data types: Value, Object, Array

Simple data types

A JSON string has to be enclosed in double quotes and may contain zero or more Unicode characters; backslash escapes are allowed. Accepted JSON numbers are in E notation. Boolean is one of `true`, `false`. Null is the reserved keyword `null`.

Data type	Examples of valid JSON
### String	`"apple"`
	`"□□"`
	`"\u00c4pfel\n"`
	`""`
### Number	`3`
	`1.4`
	`-1.5e3`
### Boolean	`true`
	`false`
### Null	`null`

Composite data types

Value

A JSON Value can be one of: String, Number, Boolean, Null, Object, Array.

Object

A JSON Object is an comma-separated unordered collection of name:value pairs enclosed in curly brackets where name is a String and value a JSON value.

Array

A JSON Array is an ordered collection of JSON values.

Example of a JSON array:

```
["home", "wooden"]
```

Examples of JSON objects:

```
{
    "id": 1,
    "name": "A wooden door",
    "price": 12.50,
    "tags": ["home", "wooden"]
```

```
}
```

```
[
  1,
  2,
  [3, 4, 5, 6],
  {
    "id": 1,
    "name": "A wooden door",
    "price": 12.50,
    "tags": ["home", "wooden"]
  }
]
```

Online tools for validating and formatting JSON data:

- http://jsonlint.com/
- http://www.freeformatter.com/json-validator.html
- http://jsonviewer.stack.hu/
- http://json.parser.online.fr/

JSON Object

A JSON Object is surrounded by curly braces and contains key-value pairs.

```
{ "key1": "value1", "key2": "value2", ... }
```

Keys must be valid strings, thus surrounded by double quotation marks. Values can be JSON objects, numbers, strings, arrays, or one of the following 'literal names': `false`, `null`, or `true`. In a key-value pair, the key is seperated from the value by a colon. Multiple key-value-pairs are separated by commas.

Order in objects is not important. The following JSON object is thus equivalent to the above:

```
{ "key2": "value2", "key1": "value1", ... }
```

To sum it all up, this is an example of a valid JSON Object :

```
{
  "image": {
    "width":  800,
    "height": 600,
    "title":  "View from 15th Floor",
    "thumbnail": {
      "url": "http://www.example.com/image/481989943",
      "height": 125,
      "width":  100
    },
    "visible": true,
    "ids": [116, 943, 234, 38793]
  }
```

```
}
```

Common examples of JSON objects, with related (Java) object counterparts

Throughout this example it is assumed that the 'root' object that is being serialized to JSON is an instance of the following class :

```
public class MyJson {
}
```

Example 1 : An example of an instance of `MyJson`, as is:

```
{}
```

i.e. since the class has no fields, only curly brackets are serialized. **Curly brackets are the common delimiters to represent an object**. Notice also how the root object is not serialized as a key-value pair. This is also true for simple types (String, numbers, arrays) when they are not fields of an (outer) object.

Example 2 : Let's add some fields to `MyJson`, and initialize them with some values:

```
// another class, useful to show how objects are serialized when inside other objects
public class MyOtherJson {}

// an enriched version of our test class
public class MyJson {
  String myString = "my string";
  int myInt = 5;
  double[] myArrayOfDoubles = new double[] { 3.14, 2.72 };
  MyOtherJson objectInObject = new MyOtherJson();
}
```

This is the related JSON representation:

```
{
  "myString" : "my string",
  "myInt" : 5,
  "myArrayOfDoubles" : [ 3.14, 2.72 ],
  "objectInObject" : {}
}
```

Notice how all the fields are serialized in a key-value structure, where the key is the name of the field holding the value. The common delimiters for arrays are square brackets. Notice also that each key-value pair is followed by a comma, except for the last pair.

JSON Array

A JSON Array is an ordered collection of values. It is surrounded by square braces i.e `[]`, and values are comma-delimited:

```
{ "colors" : [ "red", "green", "blue" ] }
```

JSON Arrays can also contain any valid JSON element, including objects, as in this example of an array with 2 objects (taken from the RFC document):

```
[
  {
     "precision": "zip",
     "Latitude":  37.7668,
     "Longitude": -122.3959,
     "Address":   "",
     "City":      "SAN FRANCISCO",
     "State":     "CA",
     "Zip":       "94107",
     "Country":   "US"
  },
  {
     "precision": "zip",
     "Latitude":  37.371991,
     "Longitude": -122.026020,
     "Address":   "",
     "City":      "SUNNYVALE",
     "State":     "CA",
     "Zip":       "94085",
     "Country":   "US"
  }
]
```

They can also contain elements with mixed types, for example:

```
[
    "red",
    51,
    true,
    null,
    {
        "state": "complete"
    }
]
```

A common mistake when writing JSON arrays (and objects) is to leave a trailing comma after the last element. This is common pattern in many languages, but unfortunately isn't valid in JSON. For example, the following array is invalid:

```
[
    1,
    2,
]
```

To make this valid, you would need to remove the comma after the last element, turning it into:

```
[
    1,
    2
]
```

Editing JSON by Hand

JSON is a very strict format (see http://json.org). That makes it easy to parse and write for machines but surprises humans when an inconspicuous mistakes breaks the document.

Common Problems

Trailing Comma

Unlike most programming languages your are not allowed to add a trailing comma:

```
{
  a: 1,
  b: 2,
  c: 3
}
```

Adding a comma after `3` will produce a synax error.

The same issue exists for arrays:

```
[
  1,
  2
]
```

You must take extra care if you need to reorder the items.

Missing Comma

```
{
  a: 1,
  b: 2,
  c: 3
  d: 4
}
```

Since trailing commas are not allowed, it is easy to forget to append one before adding a new value (in this case after `3`).

Comments

JSON does not allow comments as it is a data-interchange format. This is still a hot topic though with no clear answers other than not to use them.

There are several workarounds:

- Use C style comments, then strip them out before passing it to the parser
- Embed comments into the data

```
{
  "//": "comment",
  "data": 1
}
```

- Embed comments and overwrite them with data

```
{
  "data": "comment",
  "data": 1
}
```

The second `data` entry will overwrite the comment *in most parsers*.

Solutions

To make it easier to write JSON use an IDE that checks for syntax errors and provides syntax coloring. Plugins are available for most editors.

When you develop applications and tools, use JSON internally and as a protocol but try not to expose it in places where a human would likely be required to edit it by hand (except for debugging).

Evaluate other formats that are better suited for this use, like:

- Hjson, can be seamlessly converted to and from JSON
- TOML, similar to INI files
- YAML, more features but at the cost of added complexity

Rationale for Array vs Object (i.e. when to use what)

JSON arrays represent a collection of objects. In JS, theres a bunch of collection functions off of them such as `slice`, `pop`, `push`. Objects have just more raw data.

A **JSONArray** is an *ordered* sequence of *values*. Its external text form is a string wrapped in square brackets with commas separating the values.

A **JSONObject** is an *unordered* collection of *name/value* pairs. Its external form is a string wrapped in curly braces with colons between the names and values, and commas between the values and names.

Object - key and value, Array - numerals, strings, booleans. When do you use this or that?

You can think of Arrays as "is a/an" and Objects as "has a". Lets use "Fruit" as example. Every item in fruit array is a type of fruit.

```
array fruit : [orange, mango, banana]
```

Arrays can contain objects,strings, numbers, arrays, but lets deal with only objects and arrays.

```
array fruit : [orange:[], mango:{}, banana:{}]
```

. You can see that orange is an array too. It implies any item that goes int orange is a type of orange, say: bitter_orange, mandarin, sweet_orange.

for fruit object, any item in it is an attribute of fruit. thus the fruit has a

```
object fruit :{seed:{}, endocarp:{},flesh:{}}
```

This also implies that anything within the seed object should be property of seed, say: colour, ..

JSON is primarily a language that allows serializing javascript objects into strings. So upon deserializing a JSON string you should get a javascript object structure. If your json deserializes into an object that stores 100 objects called object1 to object100 then that's going to be very inconvenient. Most deserializers will expect you to have known objects and arrays of known objects so that they can convert the strings into the actual object structure in the language you're using. Also this is a question that the philosophy of object oriented design would answer you.

credits to all particiapnts What are the differences between using JSON arrays vs JSON objects?

Chapter 2: Parsing JSON string

Examples

Parse JSON string using com.google.gson library in Java

`com.google.gson` library needs to be added to use this code.

Here is the example string:

```
String companyDetails = {"companyName":"abcd","address":"abcdefg"}
```

JSON strings can be parsed using below syntax in Java:

```
JsonParser parser =  new JsonParser();
JsonElement jsonElement = parser.parse(companyDetails);
JsonObject jsonObj = jsonElement.getAsJsonObject();
String comapnyName = jsonObj.get("companyName").getAsString();
```

Parse JSON string in JavaScript

In JavaScript, the `JSON` object is used to parse a JSON string. This method is only available in modern browsers (IE8+, Firefox 3.5+, etc).

When a valid JSON string is parsed, the result is a JavaScript object, array or other value.

```
JSON.parse('"bar of foo"')
// "bar of foo" (type string)
JSON.parse("true")
// true (type boolean)
JSON.parse("1")
// 1 (type number)
JSON.parse("[1,2,3]")
// [1, 2, 3] (type array)
JSON.parse('{"foo":"bar"}')
// {foo: "bar"} (type object)
JSON.parse("null")
// null (type object)
```

Invalid strings will throw a JavaScript error

```
JSON.parse('{foo:"bar"}')
// Uncaught SyntaxError: Unexpected token f in JSON at position 1
JSON.parse("[1,2,3,]")
// Uncaught SyntaxError: Unexpected token ] in JSON at position 7
JSON.parse("undefined")
// Uncaught SyntaxError: Unexpected token u in JSON at position 0
```

The `JSON.parse` method includes an optional reviver function which can limit or modify the parse

result

```
JSON.parse("[1,2,3,4,5,6]", function(key, value) {
  return value > 3 ? '' : value;
})
// [1, 2, 3, "", "", ""]

var x = {};
JSON.parse('{"a":1,"b":2,"c":3,"d":4,"e":5,"f":6}', function(key, value) {
  if (value > 3) { x[key] = value; }
})
// x = {d: 4, e: 5, f: 6}
```

In the last example, the `JSON.parse` returns an `undefined` value. To prevent that, return the `value` inside the reviver function.

Parse JSON file with Groovy

Suppose we have the following JSON data:

```
{
    "TESTS":
    [
         {
              "YEAR": "2017",
              "MONTH": "June",
              "DATE": "28"
         }
    ]
}
```

import groovy.json.JsonSlurper

class JSONUtils {

```
private def data;
private def fileName = System.getProperty("jsonFileName")

public static void main(String[] args)
{
    JSONUtils jutils = new JSONUtils()
    def month = jutils.get("MONTH");
}
```

Below is the parser:

```
private parseJSON(String fileName = "data.json")
{
    def jsonSlurper = new JsonSlurper()
    def reader

    if(this.fileName?.trim())
    {
        fileName = this.fileName
    }
```

```
    reader = new BufferedReader(new InputStreamReader(new FileInputStream(fileName),"UTF-8"));
    data = jsonSlurper.parse(reader);
    return data
}

def get(String item)
{
    def result = new ArrayList<String>();
    data = parseJSON()
    data.TESTS.each{result.add(it."${item}")}
    return  result
}
```

}

Chapter 3: Stringify - Convert JSON to string

Parameters

Param	Details
Object	(Object) The JSON object
Replacer	(Function | Array<string | number> - optiotinal) filter Function | Array
Space	(Number | string - optional) Amount of white space in the JSON

Examples

Convert simple JSON object to simple string

```
var JSONObject = {
   stringProp: 'stringProp',
   booleanProp: false,
   intProp: 8
}

var JSONString = JSON.stringify(JSONObject);
console.log(JSONString);
/* output
 * {"stringProp":"stringProp","booleanProp":false,"intProp":8}
 */
```

Stringify with filter

```
var JSONObject = {
   stringProp: 'stringProp',
   booleanProp: false,
   intProp: 8
}

var JSONString = JSON.stringify(JSONObject, ['intProp']);
console.log(JSONString);
/* output
 * {"intProp":8}
 */
```

Stringify with white-space

```
var JSONObject = {
   stringProp: 'stringProp',
   booleanProp: false,
   intProp: 8
}
```

```
var JSONString = JSON.stringify(JSONObject, null, 2);
console.log(JSONString);
/* output:
 *  {
 *    "stringProp": "stringProp",
 *    "booleanProp": false,
 *    "intProp": 8
 *   }
 */
```