**JSON (JavaScript Object Notation) :**

* is mostly used ***data format for data exchange*** *on the* ***web***.
* The ***data interchange*** can happen **between two computers applications** at **different geographical locations** or running within **same hardware machine**.
* A good thing is that JSON is **human readable** and **machine-readable format**.
* While ***applications/libraries*** *can* ***parse*** *the* ***JSON data****-****humans*** *can also* ***look*** *at* ***data*** and derive meaning from it.
* JSON document may contains text, curly braces, square brackets, colons, commas, double quotes and may be a few other characters.

Primarily *JSON is* ***built*** *on* ***two structures***:

* A Collection of name/value pairs, in various languages, this is realized as an object, record, struct, dictionary, hast table, keyed list or associative array.
* An order list of values, in most languages, this realized as an array, vector, list or sequence.

JSON Data Types:

1. String – any sequence of characters inserted between double quotes.
2. Number – digits(0-9), negative number(-4), fraction(2.5), exponent(1.0e+2)
3. Boolean – true or false
4. null/empty – no value, treated as null.
5. Object – An unordered set of name-vale pairs inserted in **{ }** curly braces. Multiple name/values pairs are separated by a **,** (comma).
6. Array – An ordered collection of values in square brackets separated by a **,** (comma).

|  |  |
| --- | --- |
| **JSON Object**: key-value data format rendered in curly braces | **JSON Array**: data can be nested within the JSON using arrays in square brackets. JSPN arrays are ordered collection and contains values of different data types. |
| **{**  **"employee": {**  **"id":           1,**  **"name":         "Admin",**  **"location":     "USA"**  **}**  **}** | **{**  **"employees": [**  **{**  **"id":           1,**  **"name":         "Admin",**  **"location":     "USA"**  **},**  **{**  **"id":           2,**  **"name":         "User",**  **"location":     "USA"**  **},**  **{**  **"id":           3,**  **"name":         "User2",**  **"location":     "USA"**  **}**  **]**  **}** |
| **{**  **"color" : "Purple",**  **"id" : "210",**  **"composition" : {**  **"R" : 70,**  **"G" : 39,**  **"B" : 89**  **}**  **}** | **{**  **"colors" :**  **[**  **{**  **"color" : "Purple",**  **"id" : "210"**  **},**  **{**  **"color" : "Blue",**  **"id" : "211"**  **},**  **{**  **"color" : "Black",**  **"id" : "212"**  **}**  **]**  **}** |

**JSON Schema**:

It’s often necessary for application to validate JSON objects, to ensure that required properties are present and that additional constraints are met.

Validation is typically performed in the context of JSON Schema.

JSON Schema is grammar language for defining the structure, context and semantics of JSON object. It let us specify metadata (means data about data) and what an object’s properties mean and what values are valid for those properties.

It provides complete structural validation, which is useful for automated testing and validating client- submitted data.

**Accessing object values:** two ways (using dot(.) or bracket ([ ]) )

{

    "name" : "Admin",

    "age" : 36,

    "rights" : [ "admin", "editor", "contributor" ]

}

**Eg:**

String str= {

    "name" : "Admin",

    "age" : 36,

    "rights" : [ "admin", "editor", "contributor" ]

}

System.out.println(str.name);

Or

System.out.println(str[“name”]);

**Looping objects values:**

List<String> li= {

    "name" : "Admin",

    "age" : 36,

    "rights" : [ "admin", "editor", "contributor" ]

}

for(String s: li){

System.out.println( s + “ -- ” +li[s]);

}

**Modifying object values:**

str.name=”John”;

Or

str[“name”]=”John”;

**Deleting object values:**

delete.str.namel}

**Accessing object values:** can be accessed by using index number.

{

    "name" : "Admin",

    "age" : 36,

    "rights" : [ "admin", "editor", "contributor" ]

}

Eg.

**Get value at specific index location in array:**

x = myObj.rights[0];

**Looping through array values:**

for (String s: myObj.rights) {

    System.out.println(myObj.rights[s];

}

**Modifying array value at index location:**

myObj.rights[1] = "blogger";

**Delete array value at index location:**

delete myObj.rights[1];

**Multi-Dimensional array:**

var myObj = {

    "name"   : "Joe",

    "rights" : [

                    [ "admin",  [1,2,3] ],

                    [ "editor", [4,5,6] ]

               ]

}

**Iterate over multi-dimensional array:**

for (i in myObj.rights) {

    for (j in myObj.rights[i]) {

        x = myObj.rights[i][j];

        console.log(x);

    }

}

**JSON JSONPath:**

* is a ***query language*** to ***extract*** ***just*** ***bits of JSON document*** that our ***application needs***.

<dependency>

    <groupId>com.jayway.jsonpath</groupId>

    <artifactId>json-path</artifactId>

    <version>2.0.0</version>

</dependency>

**JSONPath Syntax:**

**$** -refers to the root object or element.

**@** - refers to current object or element.

**.** – refers to child element of the current element.

**[]** – refers to child element of the current element (by name or index).

**\*** - is a wildcard, returning all objects or elements regardless of their names.

**,** - union operator, which returns the union of children or indexes indicated.

**:** - array slice operator, can slice collections using [start:end:step] to return a subcollection of collection.

**( )** – lets to pass a script expression in the underlying implementation’s script language.

**? ( )** – to query all items that meet a certain criteria.