Handling the Front-End Data

5

Fundamental Web Programming

Asst. Prof. Manop Phankokkruad, Ph.D.
School of Information Technology
King Mongkut's Institute of Technology Ladkrabang

KMITL

Outline

1. JavaScript Object Notation
2. Local Storage

KMITL

Introduction

- XML and JSON are the two most common formats for data interchange in the Web today.
- Although their purposes are not identical, they are frequently used to accomplish the same task, which is data interchange.
- Both have well-documented open standards on the Web (*JSON : RFC 7159, XML : RFC 4825*), and both are human and machine-readable. Neither one is superior to the other, as each is better suited for different use cases.

KMITL

What is the JSON?

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, ECMA-262 3rd Edition.

- JSON is a text format that is completely language independent.
- JSON is built on two structures:
  - A collection of name/value pairs.
  - An ordered list of values.

Advantages of Using JSON

- Provide support for all browsers
- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

JSON - Syntax

JSON syntax is basically considered as a subset of JavaScript syntax; it includes the following:

- Data is represented in name/value pairs.
- Curly braces hold objects, and each name is followed by **:** (colon), the name/value pairs are separated by **,** (comma).
- Square brackets hold arrays and values are separated by **,** (comma).

```
{
    "book": [
        {
            "id": "01",
            "language": "Java",
            "edition": "third",
            "author": "Herbert Schildt"
        },
        {
            "id": "07",
            "language": "C++",
            "edition": "second",
            "author": "E.Balagurusamy"
        }
    ]
}
```

JSON : Data Types

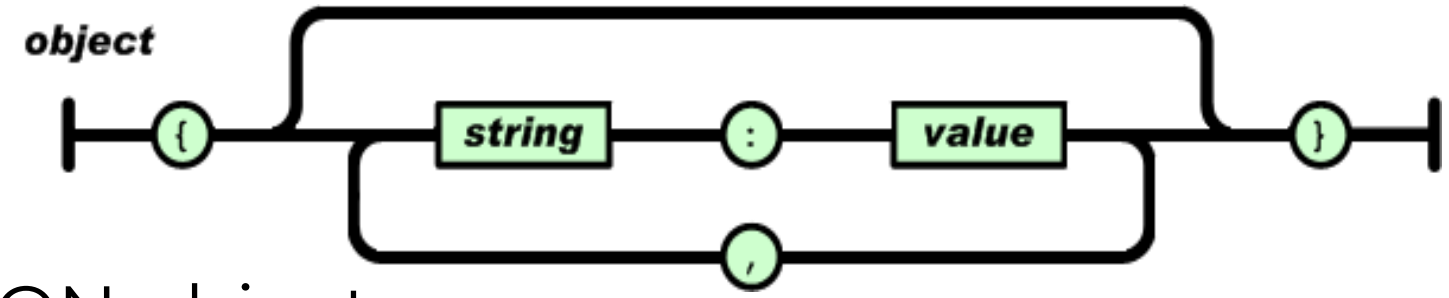
JSON format supports the following data types:

Type	Description
Number	double- precision floating-point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
Whitespace	can be used between any pair of tokens
null	empty

KMITL

JSON : object

An **object** is an unordered set of name/value pairs. An object begins with **{** and ends with **}**. Each name is followed by **:** and the name/value pairs are separated by **,** . In a JSON object the name(key) must be unique.



Example of a JSON object:

```
{ "name": "John Doe",
  "age": 30,
  "married": true
}
```

JSON : object

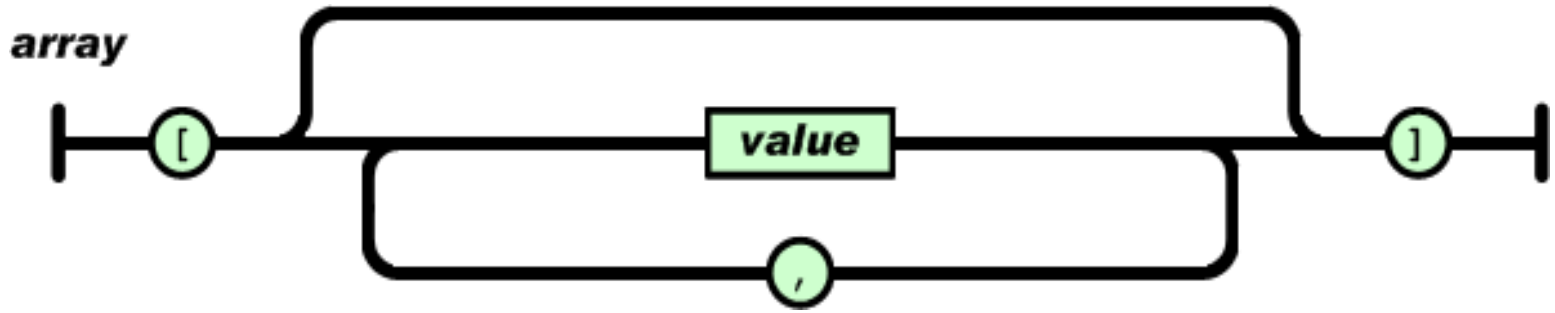
Example of a simple JSON object:

```
{
  "address" : {
    "line1" : "555 Any Street",
    "city" : "Denver",
    "stateOrProvince" : "CO",
    "zipOrPostalCode" : "80202",
    "country" : "USA"
  }
}
```

KMITL

JSON : array

An **array** is an ordered collection of values. An array begins with **[** and ends with **]** . Values are separated by **,** .



Example of Object with a nested Array

```
{ "name": "John Doe",
  "age": 30,
  "married": true,
  "siblings": ["John", "Mary", "Pat"]
}
```

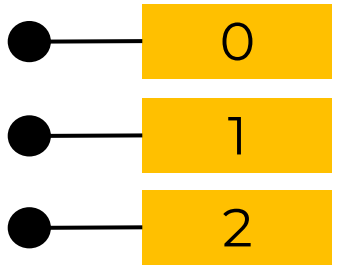
JSON : array

Syntax

[value0, value1, value2, ..., value n]

Example

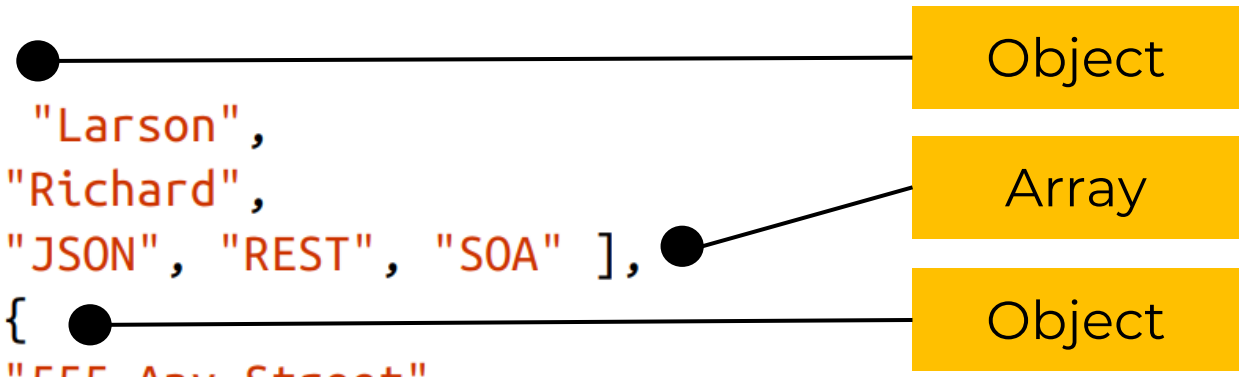
```
{
    "books": [
            { "language":"Java" , "edition":"second" },
            { "language":"C++" , "lastName":"fifth" },
            { "language":"C" , "lastName":"third" }
    ]
}
```



JSON : array

Example of Object that contains another Object

```
{
  "speaker" : {
    "firstName": "Larson",
    "lastName": "Richard",
    "topics": [ "JSON", "REST", "SOA" ],
    "address" : {
      "line1" : "555 Any Street",
      "city" : "Denver",
      "stateOrProvince" : "CO",
      "zipOrPostalCode" : "80202",
      "country" : "USA"
    }
  }
}
```



JSON : array of object

Each item in an array may be any of the seven JSON values(as described in the next slide).

Example of a JSON array of object

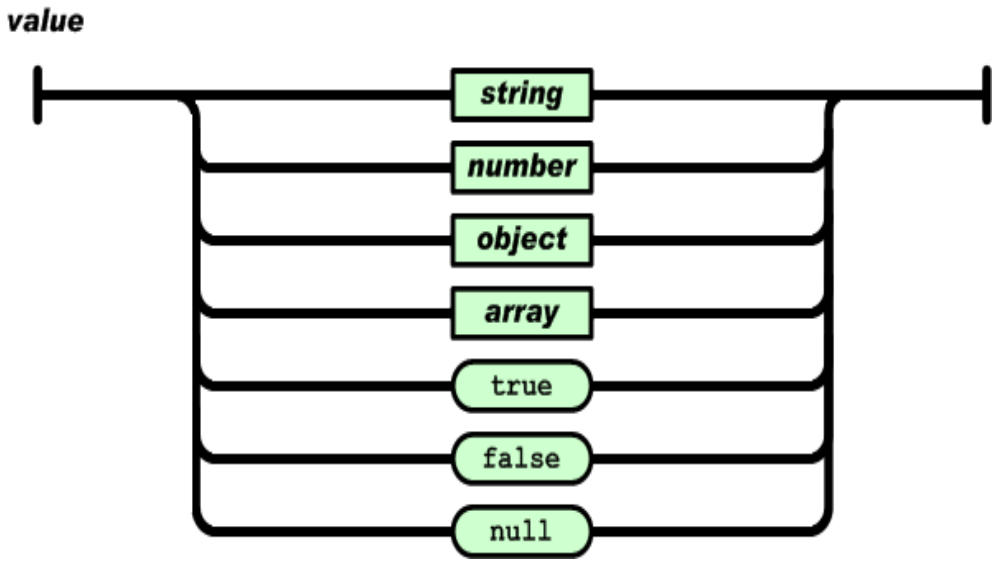
```
{
    "name": "John Doe",
    "age": 30,
    "married": true,
    "siblings": [
        {"name": "John", "age": 25},
        true,
        "Hello World"
    ]
}
```

The array contains 3 items. The first item is an object, the second item is a boolean, and the third item is a string.

KMITL

JSON : value

A **value** can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.



Syntax

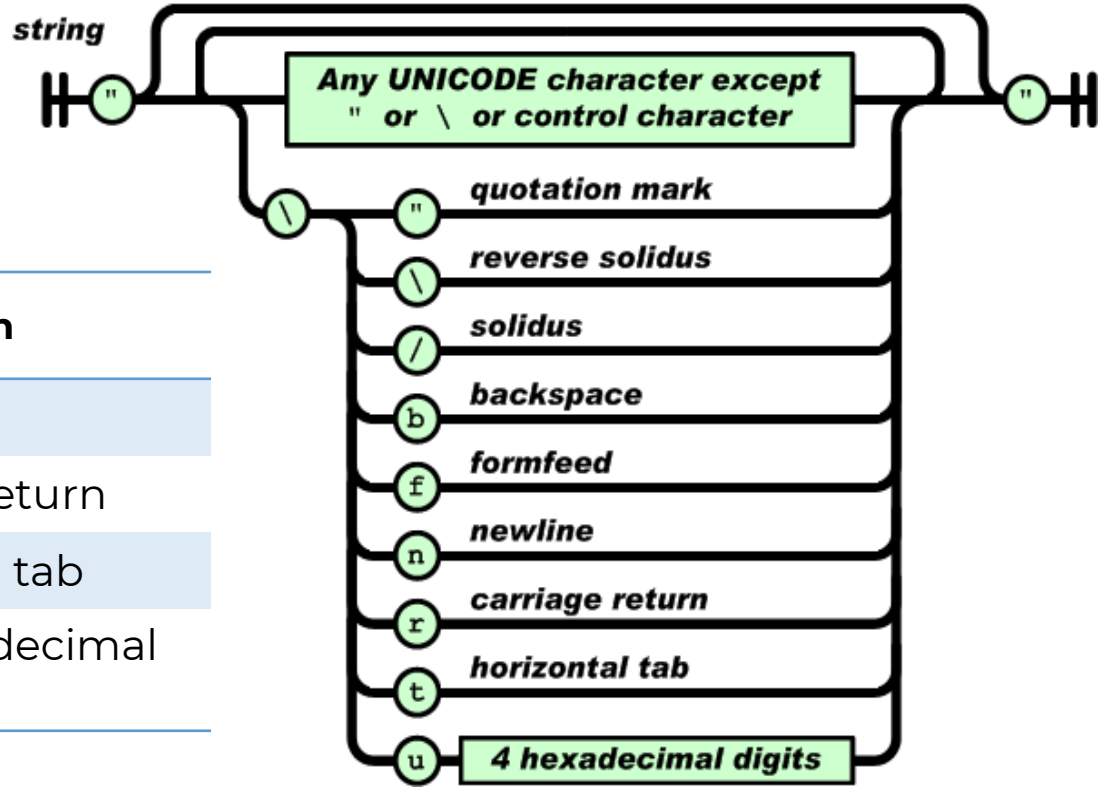
```
let object-name = { name : value, .....}
```

Example

```
let obj = {“price":  500, “product”: ”Shampoo” }
```

JSON : string

A **string** is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string.



Type	Description
\”	double quotation
\\	backslash
\/	forward slash
\b	backspace
\f	form feed

Type	Description
\n	New line
\r	carriage return
\t	horizontal tab
\u	four hexadecimal digits

JSON : number

A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.

Type	Description
Integer	Digits 1-9, 0 and positive or negative
Fraction	Fractions like .3, .7
Exponent	Exponent like e, e+, e-,E, E+, E-

```
{
  "age": 29,
  "cost": 299.99,
  "temperature": -10.5,
  "unitCost": 0.2,
  "speedOfLight": 1.23e11,
  "speedOfLight2": 1.23e+11,
  "avogadro": 6.023E23,
  "avogadro2": 6.023E+23,
  "oneHundredth": 10e-3,
  "oneTenth": 10E-2
}
```

KMITL

JSON

Example : Convert table to JSON format.

	Company	Contact	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Centro commercial	Francisco Chang	Mexico
3	Ernst Handel	Roland Mendel	Austria

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany

```
{
    "company": "Alfreds Futterkiste",
    "contact": "Maria Anders",
    "country": "Germany"
}
```

```
[
  {
    "company": "Alfreds Futterkiste",
    "contact": "Maria Anders",
    "country": "Germany"
  },
  {
    "company": "Centro Commercial",
    "contact": "Francisco Chang",
    "country": "Mexico"
  },
  {
    "company": "Ernst Handel",
    "contact": "Roland Mendel",
    "country": "Austria"
  }
]
```

JSON and JavaScript

The JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

JSON data is normally accessed in JavaScript through dot notation.

```
let person = { "name":“Brad", "age":31, “address":{“city”:“Boston“} };
document.write( person.name );              // Brad
document.write( person.address.city );      // Boston
document.write( person["name"] );           // Brad
```

Functions for Working with JSON

The **JSON.stringify()** function converts an object to a JSON string.

```
Let obj = {"first_name" : "Sammy", "last_name" : "Shark",
"location" : "Ocean"};
let s = JSON.stringify(obj);
```

We'll have the JSON available to us as a string rather than an object.

```
'{"first_name" : "Sammy", "last_name" : "Shark", "location" :
"Ocean"}'
```

KMITL

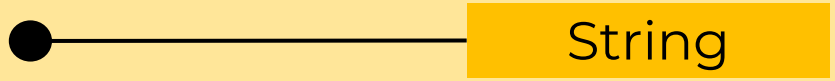
Functions for Working with JSON

The **JSON.parse()** method parses a JSON string, constructing the JavaScript value or object.

```
<script>
let str = '{"firstname" : "Sammy", "lastname" : "Shark", "location" : "Ocean"}';

let obj = JSON.parse(str);

document.write("Name: " + obj.firstname + " " +obj.lastname + "<br>");
Document.write("Location: " + obj.location );
</script>
```



Local Storage

2

HTML DOM Window localStorage is provided by Browser and it allows us to store data as key-value pairs in our web browser using an object. The localStorage is the read-only property of the window interface.

LocalStorage is a type of web storage for info. As a result, JavaScript websites and applications can store and access data without any time limits. The data will therefore always be preserved and never expire. Thus, the information saved in the browser will remain accessible even after the browser window is closed.

KMITL

Local Storage

- Data is stored as key-value pair and the keys are unique. The keys and the values are always in the UTF-16 DOM String format that is stored within localStorage.
- The main features of local storage are:

a) The storage is the origin(domain) bounded.
b) The data will not get deleted even if the browser is closed or even OS reboot and will be available until we manually clear the Local Storage of our Browser.

Local Storage

Properties and methods provided by the localStorage object:

- **setItem**(key , value): stores key/value pair
- **getItem**(key): returns the value in front of key
- **key**(index): get the key on a given index
- **length**: returns the number of stored items(data)
- **removeItem**(key): removes given key with its value
- **clear**(): deletes everything from the storage

KMITL

localStorage API

setItem(key, value) is used to store/add data to the local storage. It accepts two arguments: a key and a value. The key serves as an identifier for the data we want to store, while the value is the data itself.

```
<script>
    var a = [1, 2, 3]
    localStorage.setItem("array", a)
    var myArray = localStorage.getItem("array")
    console.log(myArray) // "1, 2, 3"
</script>
```

KMITL

localStorage API

getItem(key) retrieves stored data. If you pass a valid key to the method, it will return the data; if the key does not exist, null will be returned.

```
<script>
    localStorage.setItem("name", "Web Programming")
    var name = localStorage.getItem("name")
    console.log(name) // "Web Programming"

    var age = localStorage.getItem("age")
    console.log(age) // null
</script>
```

KMITL

localStorage API

removeItem(key) is used to remove data from localStorage. To use this method, pass in the key of the data you want to remove, and both the key and the value will be removed from localStorage.

```
<script>
    localStorage.setItem("name", "webdata1")
    localStorage.removeItem("name")

    var item = localStorage.getItem("name")
    console.log(name) // null
</script>
```

KMITL

localStorage API

key(index) is used to access the keys stored in localStorage. The index is used to get keys, and it ranges from 0 to the length of the keys.

```
<script>
    localStorage.setItem("First Name", "William")
    localStorage.setItem("Middle Name", "Henry")
    localStorage.setItem("Last Name", "Gates")

    console.log(localStorage.key(0))
    console.log(localStorage.key(1))
    console.log(localStorage.key(2))
</script>
```

KMITL

localStorage API

length retrieves the length or the number of items in localStorage.

clear() is used to clear out the data stored in localStorage. Calling this method deletes all keys and values from localStorage.

```
<script>
    localStorage.setItem("First Name", "William")
    localStorage.setItem("Middle Name", "Henry")
    localStorage.setItem("Last Name", "Gates")
    console.log(localStorage.length) // 3
    localStorage.clear()
</script>
```

KMITL

Local Storage

Example: The following snippet accesses the current domain's localStorage object.

```
<script>
    // Saving data as key/value pair
    localStorage.setItem("name", "FWP2023");
    localStorage.setItem("color", "green");
    // Updating data
    localStorage.setItem("name", "FWP2023(New)");
    localStorage.setItem("color", "Blue");
</script>
```

KMITL

Local Storage

Example: The following snippet accesses the current domain's localStorage object. (continue)

```
<script>
    // Get the data by key
    let name = localStorage.getItem("name");
    console.log("This is - ", name);
    let color = localStorage.getItem("color");
    console.log("Value of color is - ", color);
    // Get number of stored items
    let items = localStorage.length;
    console.log("Total number of items is ", items);
    // Remove key with its value
    localStorage.removeItem("color");
</script>
```

Storage Event

We can listen to storage changes on the browser. The listener function is passed in the event with the following properties:

- **newValue**: the value passed to setItem() when we create or update an item in storage.
- **oldValue**: the value of the item previously.
- **key**: the key of the item that is being changed.
- **url**: the URL for which the storage action was performed.
- **storageArea**: the storage object on which the action was performed.

KMITL

Storage Event

We can set storage event listeners either by using window.addEventListener("storage", func) or by using the onstorage attribute like so window.onstorage = func.

```
<script>
    window.onstorage = (event) => {
        if (event.key === "title") {
            alert(`your new title is ${event.newValue}
             the previous was ${event.oldValue}`)
        }
        alert('SAVED!!');
    }
    localStorage.setItem("title", "kelly")
</script>
```

KMITL

More Information

- Introducing JSON
  https://https://www.json.org
- What is JSON?
  https://www.w3schools.com/js/js_json_intro.asp
- JSON - Introduction
  https://www.w3schools.com/js/js_json_intro.asp
- JSON Tutorial
  https://www.tutorialspoint.com/json/index.htm
- HTML Web Storage API
  https://www.w3schools.com/html/html5_webstorage.asp

KMITL