JSON and AJAX

Dynamic Content

Shadi Lahham - Programmazione web - Frontend - Javascript

JSON

Javascript Object Notation

- JSON is a lightweight data-interchange format.
  - Easy for humans to read and write.
  - Easy for machines to parse and generate.
- JSON is has two structures:
  - A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.
- JSON vs Javascript objects
  - Keys must be stored with quotes
  - Values can be of these types: number, string, boolean, array, object, null
- Use JSONLint - The JSON Validator to validate your JSON files

Storing data in JSON

```
{
 "firstName": "Jane",
 "lastName": "Smith",
 "address": {
   "streetAddress": "425 2nd Street",
   "city": "San Francisco",
   "state": "CA",
   "postalCode": 94107
 },
 "phoneNumbers": [
   "212 732-1234",
   "646 123-4567" ]
}
```

JSON vs a Javascript object

```
{
 "firstName": "Jane",
 "lastName": "Smith",
 "address": {
   "streetAddress": "425 2nd Street",
   "city": "San Francisco",
   "state": "CA",
   "postalCode": 94107
 },
 "phoneNumbers": [
   "212 732-1234",
   "646 123-4567" ]
}
```

```
let person = {
  firstName: 'Jane',
  lastName: 'Smith',
  address: {
    streetAddress: '425 2nd Street',
    city: 'San Francisco',
    state: 'CA',
    postalCode: 94107
  },
  phoneNumbers: [ '212 732-1234', '646
123-4567' ]
};
```

Common JSON mistakes

```
{
  name: "John",                     // mistake: property name without quotes
  "surname": 'Smith',               // mistake: single quotes in value (must be double)
  'isAdmin': false                  // mistake: single quotes in key (must be double)
  "birthday": new Date(2000, 2, 3), // mistake: no "new" is allowed, only bare values
  "friends": [0,1,2,3]              // here all fine
}
```

JSON array

```
[
 {
   "name": "Bidhan Chatterjee",
   "email": "bidhan@example.com"
 },
 {
   "name": "Rameshwar Ghosh",
   "email": "datasoftonline@example.com"
 }
]
```

Writing a JSON to the DOM

```
let myProfile = {
  "firstName": "Liz",
  "lastName": "Howard",
  "cats": [ "Tribbles", "Jean Claws" ]
};

let p = document.createElement('p');
p.innerHTML = 'My name is ' + myProfile.firstName + ' ' + myProfile.lastName + '.';
p.innerHTML += 'My cats are ' + myProfile.cats.join(', ') + '.';
```

JSON methods

JavaScript provides these two methods:

- JSON.stringify to convert objects into JSON.
- JSON.parse to convert JSON back into an object.

JSON.stringify();

```
let student = {
  name: 'John',
  age: 30,
  isAdmin: false,
  courses: ['html', 'css', 'js'],
  wife: null
};
let json = JSON.stringify(student);
console.log(typeof json); // we've got a
string!
console.log(json);
```

Resultant JSON-encoded object

```
{
  "name": "John",
  "age": 30,
  "isAdmin": false,
  "courses": ["html", "css", "js"],
  "wife": null
}
```

JSON.parse();

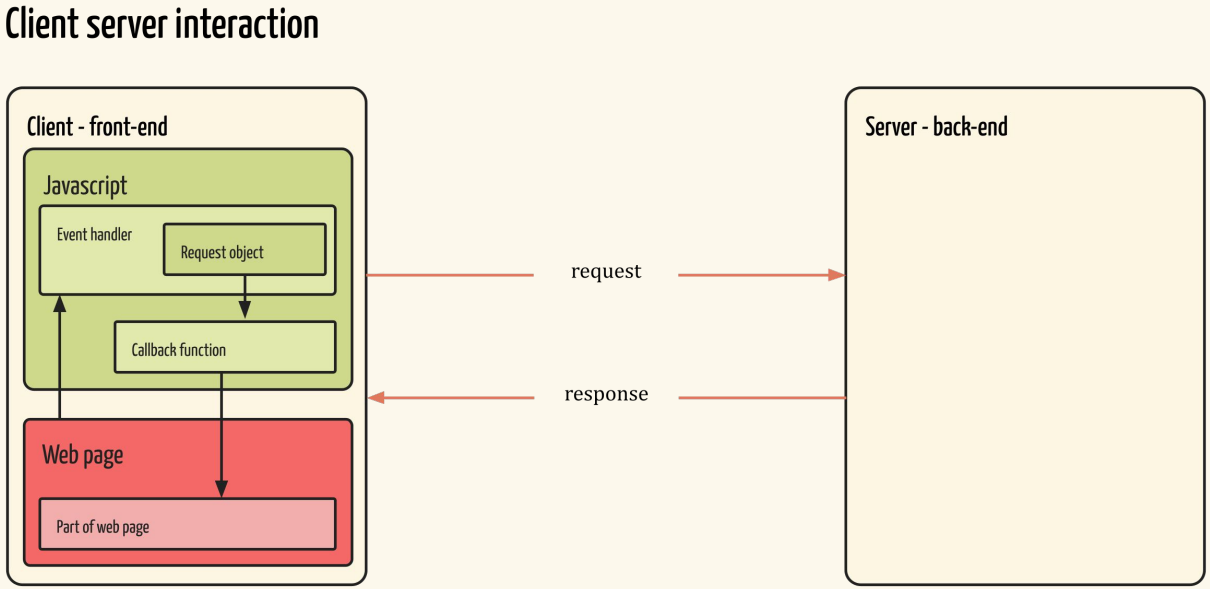
```
let user = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';

user = JSON.parse(user);

console.log(user.friends[1]); // 1
```

AJAX

AJAX - Asynchronous JavaScript and "XML"

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

AJAX - Asynchronous JavaScript and "XML"



An XMLHttpRequest

```
// instantiate a new request
let request = new XMLHttpRequest();

// add event listeners
request.addEventListener('load', function() {
  // transform a string into a usable object
  console.log(JSON.parse(request.responseText));
});

// prepare the request
request.open('get', '/path/to/api', true); // third parameter async
request.setRequestHeader('Content-type', 'application/json');

// send the request
request.send();
```

HTTP verbs and CRUD

HTTP Verb	Action	CRUD
GET	Requests data from a specified resource	Read
POST	Submits data to be processed to a specified resource	Create
PUT	Uploads a representation of the specified URI	Update
DELETE	Deletes the specified resource	Delete

There are other verbs. Here is the complete list:
GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, and PATCH

XMLHttpRequest events

loadstart
fires when the process of loading data has begun. This event always fires first

progress
fires multiple times as data is being loaded, giving access to intermediate data

error
fires when loading has failed

abort
fires when data loading has been canceled by calling abort()

load
fires only when all data has been successfully read

XMLHttpRequest events

loadend
fires when the object has finished transferring data
always fires and will always fire after error, abort, or load

timeout
fires when progression is terminated due to preset time expiring

readystatechange
fires when the readyState attribute of a document has changed

An XMLHttpRequest using onLoad

```
let request = new XMLHttpRequest();

request.open('GET', 'myservice/username?id=some-unique-id');

request.onload = function() {
  if (request.status === 200) {
    console.log("User's name is " + request.responseText);
  } else {
    console.log('Request failed.  Returned status of ' + request.status);
  }
};

request.send();
```

An XMLHttpRequest using readyState

```
let request = new XMLHttpRequest(),
let method = 'GET',
let url = 'https://developer.mozilla.org/';

request.open(method, url, true);

request.onreadystatechange = function() {
  if (request.readyState === XMLHttpRequest.DONE && request.status === 200) {
    console.log(request.responseText);
  }
};

request.send();
```

XMLHttpRequest.readyState values

0	UNSENT	Client has been created. open() not called yet.
1	OPENED	open() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; responseText holds partial data.
4	DONE	The operation is complete.

You can use them as constants e.g. `XMLHttpRequest.DONE`

HTTP Status Messages

To check the status use `XMLHttpRequest.status` **and** `XMLHttpRequest.statusText`

Categories
1xx: Information
2xx: Successful
3xx: Redirection
4xx: Client Error
5xx: Server Error

Most common
200 OK
403 Forbidden
404 Not Found
500 Internal Server Error

Complete lists: HTTP Messages , HTTP Status Codes

A PUT request example

```
let request = new XMLHttpRequest();
request.open('PUT', 'myservice/user/1234');
request.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
request.onload = function() {
  if (request.status === 200) {
    let userInfo = JSON.parse(request.responseText);
  }
};
request.send(
  JSON.stringify({
    name: 'John Smith',
    age: 34
  })
);
```

What would we need to change if this were a POST request?

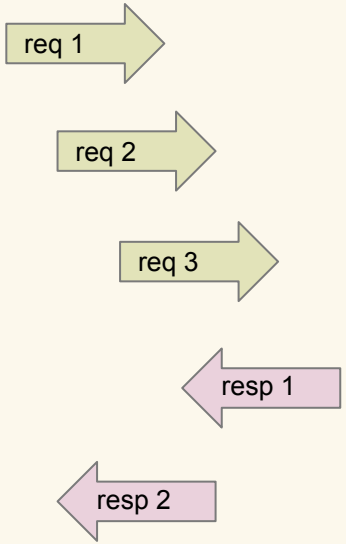
This request sends information about the encoding in the request header (the extra charset=UTF-8 part)

Multiple AJAX requests don't return in order

Client

Sends
request 1
request 2
request 3

response 3 never arrives



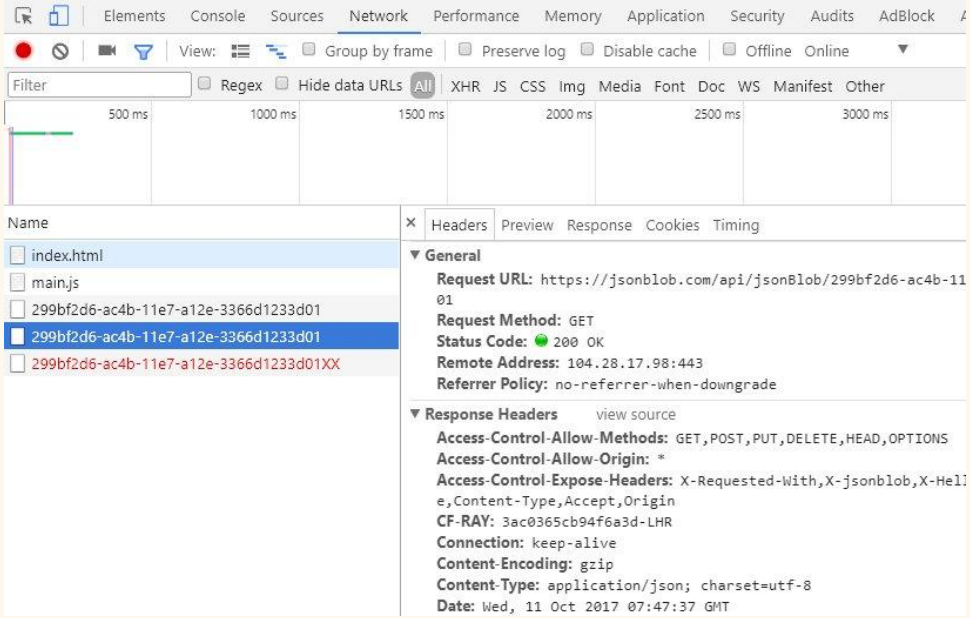
Server

Receives
request 1
request 2
request 3

Sends
response 2
response 1
.
breaks and doesn’t send other responses

Debugging AJAX Requests

Use the Dev Tools in Chrome to debug all network activity. "XHR" filters only XMLHttpRequests.



Cross Domain Policy

Cross domain requests are restricted.
Outcome for requests originating from: **http://store.company.com/dir/page.html**

URL	Outcome	Reason
http://store.company.com/**dir2**/other.html	success	
http://store.company.com/**dir/inner**/other.html	success	
http**s**://store.company.com/secure.html	failure	Different protocol
http://store.company.com**:81**/dir/other.html	failure	Different port
http://**news**.company.com/dir/other.html	failure	Different host

Cross-Origin Resource Sharing | MDN
Cross-origin resource sharing

Status and error handling

Show loader or message: Check the status and always give users feedback
Handle errors: No guarantee that HTTP requests will always succeed

```
let request = new XMLHttpRequest();
request.open('GET', url);
request.onload = function() {
  if (request.status === 200) {
    // do something useful with request
  } else {
    console.error("Request didn't load successfully. Error code:", request.statusText);
  }
};
request.onerror = function() {
  console.error('Network error');
};
request.send();
```

Simulating the backend

JSON blob allows you to create JSON objects online and access them them as endpoint via HTTPS requests.

https://jsonblob.com/api#endpoints

Important: Read the documentation and experiment with creating a JSON to make sure you understand how it works.

Simulating the backend

JSON blob is a free service, so sometimes it doesn't work.

There are many alternatives. Search and you will find many.

Here are a few:

https://my-json-server.typicode.com/

https://github.com/typicode/json-server (requires node)

https://jsonbin.io/

Your turn

1.Factory

- Write car.json, a JSON that represents a car object
- Try to make your object as complete as possible, having at least one property of the following types:
  - Number, String, Boolean, Array, Object, Null
- Write a factory.json that represents a car factory. Follow the same rules above.
- Transform car.json into cars.json with 5 cars
- Cars should belong to a factory. Write two variants of factory.json:
  - One that has cars directly embedded in the factory JSON structure.
  - Another that uses cars referring to their IDs

2.DOM Factory

- Write your cars and factory objects as JSON strings in a variable.
- Parse them with JSON.parse();
- Write each of them to the DOM in a list
    - You should use a CSS styled list with no bullets
    - Don’t use <table>

3.Remote factory

- Use jsonblob to store JSON data about cars and a car factory
- You can use as many blobs as you need. Decide the structure in a way to reduce the amount of data you modify with HTTP requests
- Write an application that displays a factory with a list of cars
- Clicking on each car should display a collapsible panel with more information about the car
- It should be possible to edit the car details
- Save the modified data to jsonblob with an HTTP request
- Whenever data is modified you should reload the new data from jsonblob once the writing has finished

Continues on next page >>>

3.Remote factory

- You should handle all error cases in your application. If an HTTP request fails, you should display a message to the user
- Your project should include a folder called ‘json’ with all the initial json files that you upload to jsonblob (the initial state of your DB)
- Your readme (markdown) should include links to all the jsonblobs that you are using as well as a list of their IDs

Bonus

4.Parallel factory

- Create another version of the factory that uses the same jsonblobs that you created for the previous exercise
- Make sure that each car information is stored in a different jsonblob
- The page should display the list of cars with detailed information about each car directly visible without a collapsible panel
- Make sure that you request all jsonblobs in parallel (at the same time) not in sequence (one after another)
- Show a loader or a loading message while loading and show the list only when data has returned from all jsonblobs and all requests finished
- Make sure that your code handles all **errors**

References

JSON

JSON.org

JSON Syntax

JSON.stringify()

JSON.parse()

References

XMLHttpRequest

XMLHttpRequest.open()

XMLHttpRequest.send()

Using XMLHttpRequest

References

HTTP Methods GET vs POST

An introduction to HTTP verbs – Roblog, the blog of Rob Miller

HTTP request methods