Ajax & Promises

or how to create ~~callback~~ promise hell

Frontend Junior Program - 2022

AJAX = Asynchronous JavaScript And XML

but what is XML?

(XMLHttpRequest)

extra cassette, extra resolution, extensible language

XML was born at the end of a long era when it was absolute mandatory to add an "X" to almost everything to state its superior and divine nature.

Unfortunately, XML infected the HTML (hence the XHTML was born), throwing back the evolution of the web for ten years.

A short history of why we had to work endless nights on fighting with the Internet Explorer

*"XHTML is by far the silliest puff of hothouse XMLing-for-its-own-sake to have wafted out of the W3C. It is bad engineering from beginning to end, and as such, constitutes a prime example of how text/xml - as the means to deliver XML applications - could become an unworkable mess through dogmatic advocacy of *bad* designs."*

- XHTML Considered Harmful

1969: GML

Charles F. Goldfarb, Ed Mosher, Ray Lorie:

Generalized Markup Language

- *was named after the authors*

Internet Engineering Task Force (IETF):

Transmission Control Protocol / Internet Protocol **- TCP/IP**

1980: SGML

Charles F. Goldfarb

Standard GML

1990: HTML

Tim Berners-Lee:

- ***HiperText Markup Language***
  (a dumbed-down version of SGML)
- *the first web server + browser*

1995: JavaScript

Brendan Eich

Scripting Language for the Netscape Navigator

1996: XML

W3C:

- ***Extensible Markup Language***
  (a dumbed-down version of SGML)
- *extensible and strict*

IE 6

2001:

IE 8

1998: XHTML

W3C

- ***Extensible HyperText Markup Language***
- *HTML in XML syntax*

2004-: HTML5

WHATWG:

- *not SGML based*

IE 9

2006:

Tim Berners-Lee

apologies for screwing up the HTML (well, almost)

1996-2006:

The Dark Age *of the Web Development*

- *by forcing XML as a base of the web, any development in HTML was essentially stopped, enabling the outdated IE versions to rule the market*
- *the consequences are still here, slowly fading out*

XML is not that forgiving standard

And as such, it violates the Robustness principle, despite that error handling and correction is part all the data transmission and communication standards.

Except the XML, which simply gives up on any error - but the behavior could depend on the client (i.e., some of them ignore the standard).

HTML5 was built with error handling in mind, as it specifies the methods how a page should be processed despite its problems – this way, the behavior can be consistent across browsers.

"In general, an implementation should be conservative in its sending behavior, and liberal in its receiving behavior."

Postel's law

just a slash is missing there

```
<?xml version="1.0" encoding="UTF-8"?>
<body>
  <shrek>There's a lot more to ogres than people think<shrek>
</body>
```

XML Parsing Error: mismatched tag. Expected: </shrek>.
Location: file:///Users/home/shrek/src/shrek.xml
Line Number 4, Column 3:

```
</body>
--^
```

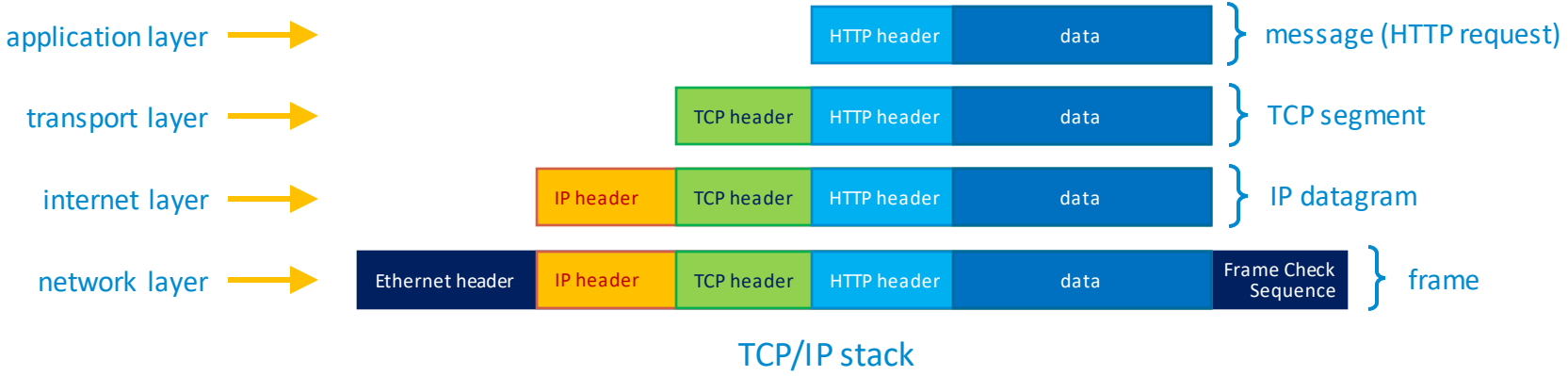
Fortunately, the internet itself if more robust than XML

Thanks to Jon Postel and the way the data is transferred, the internet is stable and usable.

The internet communication is based on layers: basically, when sending, additional information will be packed around the data as it goes through these layers.

Different protocols exist on the application layer, such as http, ftp, smtp, telnet, etc.

the internet is like an ogre, it has layers!



So, what does the *XML* have to do with *AJAX* and *XMLHttpRequest*?

Very little. XMLHttpRequest works with any type of data.

"This was the good-old-days when critical features were crammed in just days before a release…I realized that the MSXML library shipped with IE and I had some good contacts over in the XML team who would probably help out—I got in touch with Jean Paoli who was running that team at the time and we pretty quickly struck a deal to ship the thing as part of the MSXML library.

Which is the real explanation of where the name XMLHTTP comes from—the thing is mostly about HTTP and doesn't have any specific tie to XML other than that was the easiest excuse for shipping it so I needed to cram XML into the name."

— Alex Hopmann *The story of XMLHTTP*

XMLHttpRequest basic usage

create an instance of XMLHttpRequest

add the required event listeners

setup a request (open)

send

this is the *this* in the callback, and yes, there is bit of magic here with the *this*

responseText

```
> function requestCallback () {
    console.log(this);
    console.log(this.responseText);
  }

  let request = new XMLHttpRequest();

  request.onload = requestCallback;

  request.open("GET", 'https://api.spacexdata.com/v4/company');

  request.send();
<· undefined
                                                                    VM2707:2
▶ XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials:
  false, upload: XMLHttpRequestUpload, …}
                                                                    VM2707:3
{"headquarters":{"address":"Rocket
Road","city":"Hawthorne","state":"California"},"links":{"website":"https://www.spacex.
com/","flickr":"https://www.flickr.com/photos/spacex/","twitter":"https://twitter.com/
SpaceX","elon_twitter":"https://twitter.com/elonmusk"},"name":"SpaceX","founder":"Elon
Musk","founded":2002,"employees":9500,"vehicles":4,"launch_sites":3,"test_sites":3,"ce
o":"Elon Musk","cto":"Elon Musk","coo":"Gwynne Shotwell","cto_propulsion":"Tom
Mueller","valuation":74000000000,"summary":"SpaceX designs, manufactures and launches
advanced rockets and spacecraft. The company was founded in 2002 to revolutionize
space technology, with the ultimate goal of enabling people to live on other
planets.","id":"5eb75edc42fea42237d7f3ed"}
```

a real API

XMLHttpRequest API

it operates with event handlers

response is not always text, it depends on the *responseType*

HTTP status code

200: Ok
401: Unauthorized
404: Not Found
500: Internal Server Error
(aka: ping your BE colleague)

```
▼ XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials:
  false, upload: XMLHttpRequestUpload, …} i
    onabort: null
    onerror: null
  ▶ onload: ƒ ()
    onloadend: null
    onloadstart: null
    onprogress: null
    onreadystatechange: null
    ontimeout: null
    readyState: 4
    response: "{\"heat_shield\":{\"material\":\"PICA-X\",\"size_meters\":3.6,\"temp_degr…
    responseText: "{\"heat_shield\":{\"material\":\"PICA-X\",\"size_meters\":3.6,\"temp_…
    responseType: ""
    responseURL: "https://api.spacexdata.com/v4/dragons/5e9d058759b1ff74a7ad5f8f"
    responseXML: null
    status: 200
    statusText: ""
    timeout: 0
  ▶ upload: XMLHttpRequestUpload {onloadstart: null, onprogress: null, onabort: null, on…
    withCredentials: false
```

XMLHttpRequest versus *this*

When calling a function as a *function* (in contrast as a method) the *this* should be global object:

```
> function callback () {
    console.log(this);
  }

  function Constructor() {
      return {
          callFunction(callback) {
              callback();
          }
      }
  }

  let instance = new Constructor();

  instance.callFunction(callback);
                                                    VM1507:2
  ▶ Window {window: Window, self: Window, document:
    document, name: "", location: Location, …}
```

this === global object

```
> function callback () {
    console.log(this);
  }

  function Constructor() {
      return {
          callFunction(callback) {
              callback.call(this);
          }
      }
  }

  let instance = new Constructor();

  instance.callFunction(callback);
  ▶ {callFunction: f}                               VM1896:2
```

we pass the *this*

this === object instance

There is public API, serving the data of the SpaceX launches: https://github.com/r-spacex/SpaceX-API, please feel free to play with it!

Name

dragons
api.spacexdata.com/v4

1 requests 2.0 kB transferred 3.5 kB r

× Headers Preview Response Initiator Timing

General

Request URL: `https://api.spacexdata.com/v4/dragons`

Request Method: `GET`

Status Code: 200

Remote Address: `108.61.204.151:443`

Referrer Policy: `strict-origin-when-cross-origin`

Response Headers

access-control-allow-origin: `*`

access-control-expose-headers: `spacex-api-cache,spacex-api-response-time`

alt-svc: `h3-29=":443"; ma=2592000,h3-34=":443"; ma=2592000,h3-32=":443"; ma=2592000`

cache-control: `max-age=86400`

content-encoding: `gzip`

content-security-policy: `default-src 'self';base-uri 'self';block-all-mixed-content;font-src 'self' https: data:;frame-ancestors 'self';img-src 'self' data:;object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self' https: 'unsafe-inline';upgrade-insecure-requests`

content-type: `application/json; charset=utf-8`

the response is a JSON data

Name

dragons
api.spacexdata.com/v4

1 requests 2.0 kB transferred 3.5 kB r

× Headers Preview Response Initiator Timing


```
▼ [{heat_shield: {material: "PICA-X", size_meters: 3.6, temp_degrees: 3000, dev_partner: "NASA"},…},…]
  ▶ 0: {heat_shield: {material: "PICA-X", size_meters: 3.6, temp_degrees: 3000, dev_partner: "NASA"},…}
  ▼ 1: {heat_shield: {material: "PICA-X", size_meters: 3.6, temp_degrees: 3000, dev_partner: "NASA"},…}
      active: true
      crew_capacity: 7
      description: "Dragon 2 (also Crew Dragon, Dragon V2, or formerly DragonRider) is the second version of
    ▶ diameter: {meters: 3.7, feet: 12}
      dry_mass_kg: 6350
      dry_mass_lb: 14000
      first_flight: "2019-03-02"
    ▶ flickr_images: ["https://farm8.staticflickr.com/7647/16581815487_6d56cb32e1_b.jpg",…]
    ▶ heat_shield: {material: "PICA-X", size_meters: 3.6, temp_degrees: 3000, dev_partner: "NASA"}
    ▶ height_w_trunk: {meters: 7.2, feet: 23.6}
      id: "5e9d058859b1ffd8e2ad5f90"
    ▶ launch_payload_mass: {kg: 6000, lb: 13228}
    ▶ launch_payload_vol: {cubic_meters: 25, cubic_feet: 883}
      name: "Dragon 2"
      orbit_duration_yr: 2
    ▶ pressurized_capsule: {payload_volume: {cubic_meters: 11, cubic_feet: 388}}
    ▶ return_payload_mass: {kg: 3000, lb: 6614}
    ▶ return_payload_vol: {cubic_meters: 11, cubic_feet: 388}
      sidewall_angle_deg: 15
```


actually, it is an array, so we should parse it

```
> let request = new XMLHttpRequest();

  request.onload = function() {
    const dragons = JSON.parse(this.responseText);
    console.log(dragons);
  };

  request.open("GET", 'https://api.spacexdata.com/v4/dragons');
  request.send();
<· undefined
```

VM562:5

```
▼(2) [{…}, {…}]
  ▶0: {heat_shield: {…}, launch_payload_mass: {…}, launch_payload_vol: {…}, return_payload_mass: {…}, return_payload_vol: {…}, …}
  ▼1:
     active: true
     crew_capacity: 7
     description: "Dragon 2 (also Crew Dragon, Dragon V2, or formerly DragonRider) is the second version of the SpaceX Dragon spacec…
    ▶diameter: {meters: 3.7, feet: 12}
     dry_mass_kg: 6350
     dry_mass_lb: 14000
     first_flight: "2019-03-02"
    ▶flickr_images: (3) ["https://farm8.staticflickr.com/7647/16581815487_6d56cb32e1_b.jpg", "https://farm1.staticflickr.com/780/211…
    ▶heat_shield: {material: "PICA-X", size_meters: 3.6, temp_degrees: 3000, dev_partner: "NASA"}
    ▶height_w_trunk: {meters: 7.2, feet: 23.6}
     id: "5e9d058859b1ffd8e2ad5f90"
    ▶launch_payload_mass: {kg: 6000, lb: 13228}
    ▶launch_payload_vol: {cubic_meters: 25, cubic_feet: 883}
     name: "Dragon 2"
     orbit_duration_yr: 2
    ▶pressurized_capsule: {payload_volume: {…}}
    ▶return_payload_mass: {kg: 3000, lb: 6614}
    ▶return_payload_vol: {cubic_meters: 11, cubic_feet: 388}
     sidewall_angle_deg: 15
    ▶thrusters: (2) [{…}, {…}]
    ▶trunk: {trunk_volume: {…}, cargo: {…}}
     type: "capsule"
     wikipedia: "https://en.wikipedia.org/wiki/Dragon_2"
    ▶__proto__: Object
   length: 2
  ▶__proto__: Array(0)
```

We have the dragons now!

But that's too much: let's query first the *id*, and then get the *dragon*!

Consecutive requests could be called inside the callback

However, usually we have to send many requests – based on the previous request results, so we could find ourself in a bit of inception like situation soon: we have a callback inside a callback, inside a callback…

This is called: *the callback hell*.

```
> let dragonsXhr = new XMLHttpRequest();

  dragonsXhr.onload = function() {
    const dragons = JSON.parse(this.responseText);
    const id = dragons[0].id;
    console.log(id);

    let xhr = new XMLHttpRequest();
    xhr.onload = function() {
      console.log(JSON.parse(this.responseText));
    };
    xhr.open("GET", 'https://api.spacexdata.com/v4/dragons/' + id);
    xhr.send();
  };

  dragonsXhr.open("GET", 'https://api.spacexdata.com/v4/dragons');
  dragonsXhr.send();
<· undefined
  5e9d058759b1ff74a7ad5f8f                                  VM1135:6
                                                            VM1135:10
  ▼ {heat_shield: {…}, launch_payload_mass: {…}, launch_payload_vol:
    {…}, return_payload_mass: {…}, return_payload_vol: {…}, …} i
      active: true
      crew_capacity: 0
      description: "Dragon is a reusable spacecraft developed by Spac…
    ▶ diameter: {meters: 3.7, feet: 12}
      dry_mass_kg: 4200
```

With ES6, however, we have fetch API

Fetch API replaces the XMLHttpRequest with a simpler, cleaner API *(but most importantly, it does not have XML in the name).*

The fetch API, however, utilizes promises, instead of event handlers.

much simpler API

```
> fetch('https://api.spacexdata.com/v4/dragons')
    .then(response => response.json())
    .then(data => console.log(data));
<· ▶ Promise {<pending>}
                                                    VM1600:3
  ▼ (2) [{…}, {…}]
    ▼ 0:
        active: true
        crew_capacity: 0
        description: "Dragon is a reusable spacecraft…
      ▶ diameter: {meters: 3.7, feet: 12}
        dry_mass_kg: 4200
        dry_mass_lb: 9300
        first_flight: "2010-12-08"
      ▶ flickr_images: (4) ["https://i.imgur.com/9fWd…
      ▶ heat_shield: {material: "PICA-X", size_meters…
      ▶ height_w_trunk: {meters: 7.2, feet: 23.6}
        id: "5e9d058759b1ff74a7ad5f8f"
```

everything is a promise:
fetch() returns a promise
response.json() returns a promise
then returns a promise

```
> typeof fetch('https://api.spacexdata.com/v4/dragons');
<· "object"
> fetch('https://api.spacexdata.com/v4/dragons').toString();
<· "[object Promise]"
> fetch('https://api.spacexdata.com/v4/dragons')
    .then(response => {
      console.log(response.json().toString());
      console.log(response);
    }
  );

<· ▶ Promise {<pending>}
  [object Promise]                                          VM2341:3
                                                            VM2341:4
  ▼ Response {type: "cors", url: "https://api.spacexdata.com/v4/dragons",
    redirected: false, status: 200, ok: true, …}
    ▶ body: ReadableStream
      bodyUsed: true
    ▶ headers: Headers {}
      ok: true
      redirected: false
      status: 200
      statusText: ""
      type: "cors"
      url: "https://api.spacexdata.com/v4/dragons"
    ▶ __proto__: Response
```

response is an object

body is definitely not a text,

the body of the data can be handled with different Response methods (actually, Body methods, because Response implements the Body object)

What is a promise then?

Promise means simply that it will run the callbacks in the *.then()* chain asynchronously, when the the promise has been *fulfilled* (e.g., the API call response arrived).

Comparing to XMLHttpRequest, it still uses callbacks, but there are 2 major differences:

1. the structure is flattened now
2. the callbacks will be called by the promise directly, and not through the event loop

```
xhr_1.onload = function() {
  xhr_2.onload = function() {
    xhr_3.onload = function() {
      ...
    }
  }
}
```

```
fetch()
  .then(() => {})
  .then(() => {})
  .then(() => {});
```

resolve or *reject*

```
> const asyncJob = new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log('Shrek: No!');
      reject({
        place: 'Road'
      });
    });
  });

  asyncJob
    .then((data) => {
      console.log(data);
      console.log("Donkey: Oh, finally! Wow!");
    })
    .catch((data) => {
      console.log(data);
      console.log("Donkey: This is why nobody likes ogres.");
    });
<· ▶ Promise {<pending>}
  Shrek: No!                                          VM3078:3
  ▶ {place: "Road"}                                   VM3078:16
  Donkey: This is why nobody likes ogres.             VM3078:17
```

an *async* job

we *reject* now

catch can be used in the rejected cases

not there yet...

Promises are also useful for handling other asynchronous situations, not just API calls

We can create our own promise instances as well (promise is an object).

```
const asyncJob = new Promise((resolve, reject) => {
  setTimeout(() => {
    console.log('Fiona: Yes!');
    resolve({
      place: 'Castle'
    });
  });
});

asyncJob.then((data) => {
    console.log(data);
    console.log("Donkey: Oh, finally! Wow!");
});
```

now we *resolve*

```
▶ Promise {<pending>}
Fiona: Yes!                                   VM3072:3
▶ {place: "Castle"}                           VM3072:11
Donkey: Oh, finally! Wow!                     VM3072:12
```

the resolved data is available in the callbacks (both in *resolved* and *rejected* cases)

yes!

Data can (and must) be channeling though

Passing the data from *.then()* to *.then()* is the responsibility of the callback.

we return *undefined*

so the data will be *undefined* here

but we can return new data

```
> const asyncJob = new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log('Fiona: Yes!');
      resolve({
        place: 'Road'
      });
    });
  });

  asyncJob
    .then((data) => {
      console.log(data);
      console.log("Donkey: Oh, finally! Wow!");
    })
    .then((data) => {
      console.log(data);
      console.log("Donkey: It's going to be champagne wishes and caviar dreams from now on.");

      return {
        place: 'City'
      };
    })
    .then((data) => {
      console.log(data);
      console.log("Donkey: Hey, good-looking! We'll be back to pick you up later!");
    });
<· ▶ Promise {<pending>}
  Fiona: Yes!                                                          VM3106:3
  ▶ {place: "Road"}                                                    VM3106:12
  Donkey: Oh, finally! Wow!                                            VM3106:13
  undefined                                                            VM3106:16
  Donkey: It's going to be champagne wishes and caviar dreams from now on.  VM3106:17
  ▶ {place: "City"}                                                    VM3106:24
  Donkey: Hey, good-looking! We'll be back to pick you up later!       VM3106:25
```

original data

undefined

new data

.then() can return a *new promise* as well

And usually, this is the case. The whole purpose of the next *.then()* element in the chain is to start a new async job (e.g., a new *fetch* call), and wait for its result.

we return a new promise

```
> const asyncJob = new Promise((resolve) => {
    setTimeout(() => {
      console.log('Fiona: Yes!');
      resolve({
        place: 'Road'
      });
    }, 500);
  });

  asyncJob
    .then((data) => {
      console.log(data);
      console.log("Donkey: Oh, finally! Wow!");
    })
    .then(() => {
      return new Promise((resolve) => {
        setTimeout(() => {
          console.log("Donkey: It's going to be champagne wishes and caviar dreams from now on.");
          resolve({
            place: 'City'
          });
        }, 10000);
      });
    })
    .then((data) => {
      console.log(data);
      console.log("Donkey: Hey, good-looking! We'll be back to pick you up later!");
    });
<· ▶Promise {<pending>}
  Fiona: Yes!                                                                    VM3626:3
  ▶{place: "Road"}                                                               VM3626:12
  Donkey: Oh, finally! Wow!                                                      VM3626:13
  Donkey: It's going to be champagne wishes and caviar dreams from now on.       VM3626:18
  ▶{place: "City"}                                                               VM3626:26
  Donkey: Hey, good-looking! We'll be back to pick you up later!                 VM3626:27
```

everything is as expected

Q&A

edu_hu@epam.com