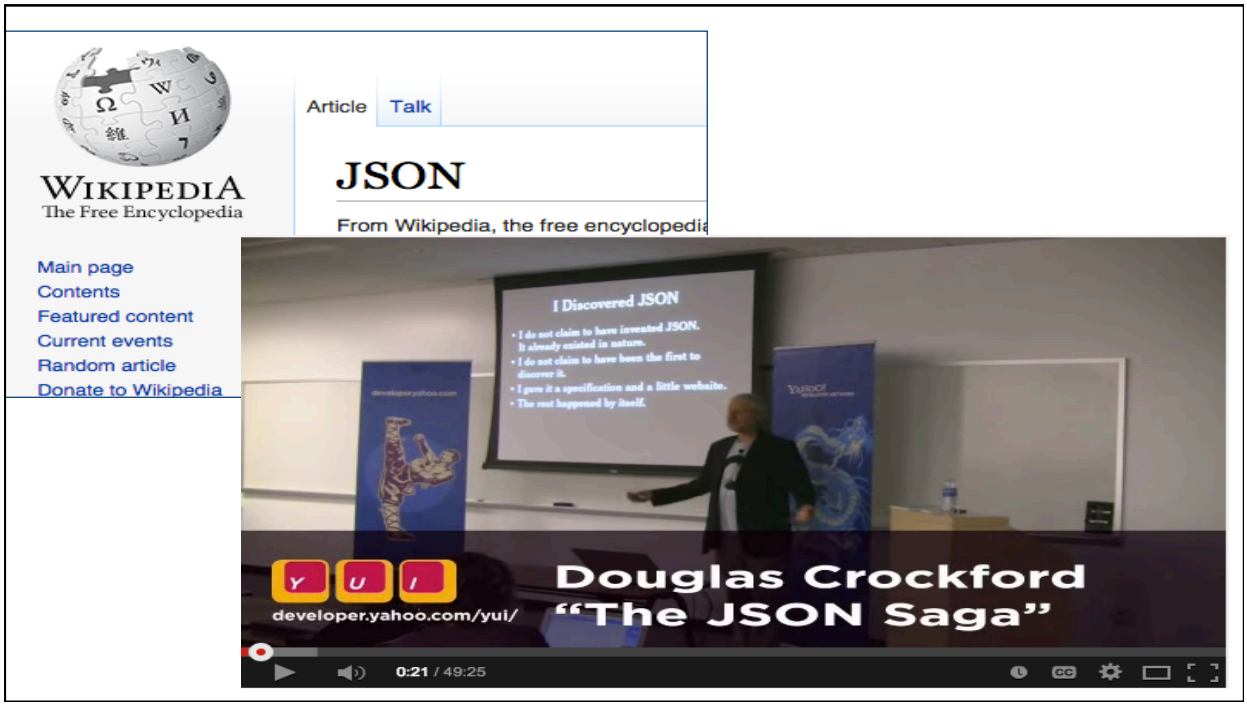
Ajax

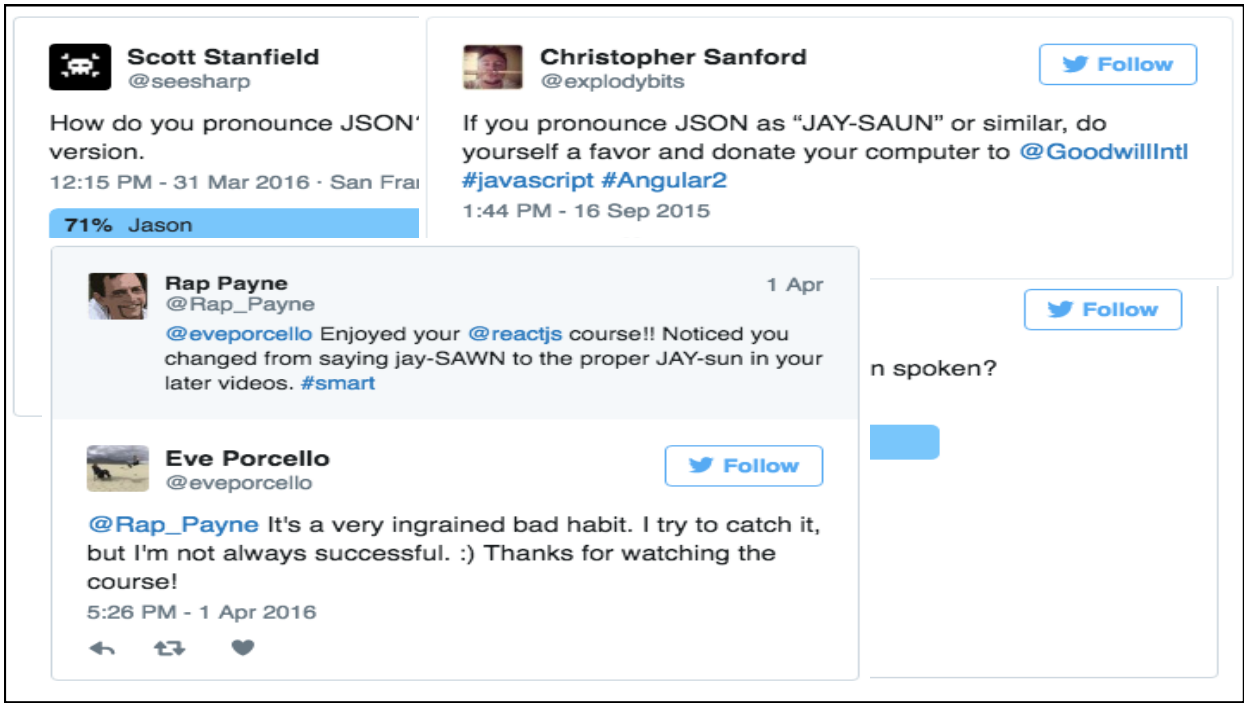
Talking with the server

tl;dr

- Ajax is transferring data between two computers
- JSON is the favorite format for Ajax
- XMLHttpRequest object is the heart of Ajax requests
- The fetch API is great for making Ajax requests because it is easy and supported by all modern browsers with no libraries

json


WIKIPEDIA
The Free Encyclopedia
Article
Talk
JSON
From Wikipedia, the free encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
I Discovered JSON
developer.yahoo.com/yui/
Douglas Crockford
"The JSON Saga"
0:21 / 49:25



JSON is simply a way to represent an object

- Like a person object whose
- First name is "Cal"
- Last name is "Naughton"
- Nickname is "Magic Man"
- Car number is 47

```
{
  firstname: "Cal",
  lastname: "Naughton",
  nickname: "Magic Man",
  carNumber: 47
}
```

Note the syntax:

- The object is bounded by curly braces
- Entries are comma-separated
- Keys & values are colon-separated

Sounds like XML. Why not just use XML?

```
<person>
  <firstname>
    Cal
  </firstname>
  <lastname>
    Naughton
  </lastname>
  <nickname>
    Magic Man
  </nickname>
  <carNumber>
    47
```

126 BYTES

```
{
  firstname: "Cal",
  lastname: "Naughton",
  nickname: "Magic Man",
  carNumber: 47
}
```

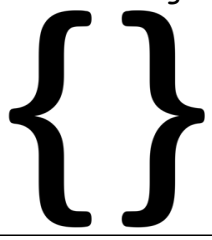
73 BYTES

String to Object

```
var driver = getDriver(123);   // This returns a string.
console.log(typeof driver);    // "string"
console.log(driver);           // '{"firstName":"Ricky"}'
console.log(driver.firstName); // undefined
```

- driver is a string, but we need to get JSON data!

```
var obj = JSON.parse(driver); // *Now* obj is an object
console.log(obj.firstName);   // "Ricky"
```



Object to String

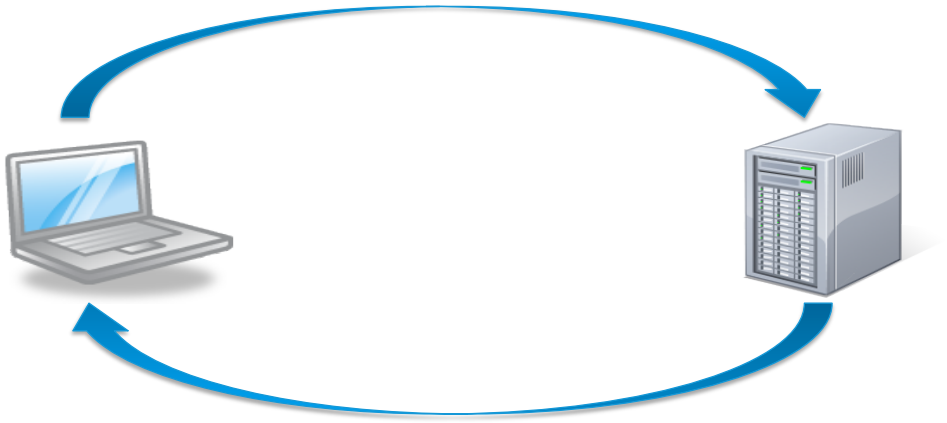
- Great for serializing an object.

```
console.log(typeof obj);      // 'object'
console.log(obj.firstName);   // 'Ricky'
var str = JSON.stringify(obj);
console.log(str); // '{"firstName":"Ricky"}'
```

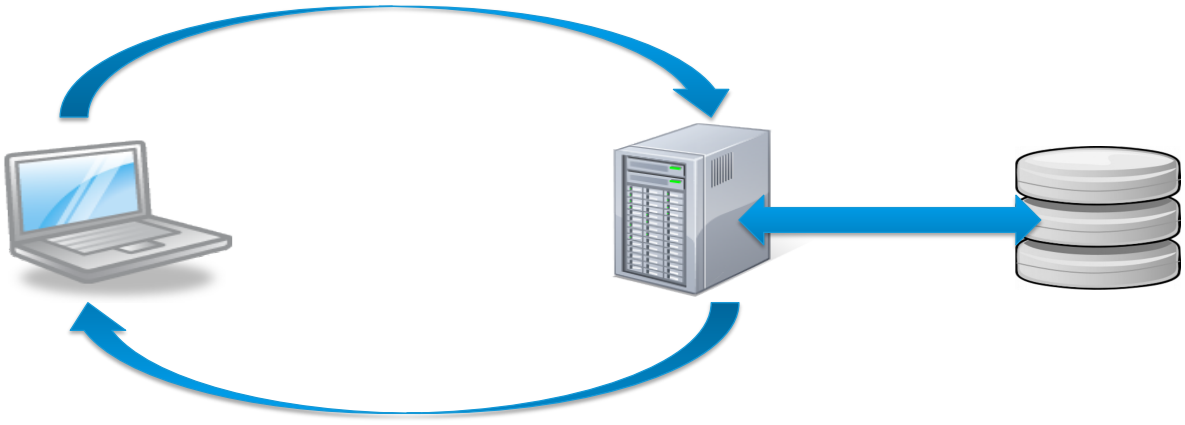
Introduction to Ajax

Making Ajax calls at the lowest possible level

Remember that the web is a thin-client technology



So we clearly need some way to tell the server we want to send data to it and a way to receive data from it.



If only we could send a request while the page remains loaded and displayed

- Precondition – a page is loaded

1. The page sends a request for some data
2. The page listens for a response while the user continues to work (asynchronous)
3. It receives a response
4. It runs a callback function which usually processes the data and injects it into the DOM

- Postcondition – the same page is loaded but has new data

So we use JavaScript to send an XML request asynchronously and process the XML response

We call this technology …

JXAA!

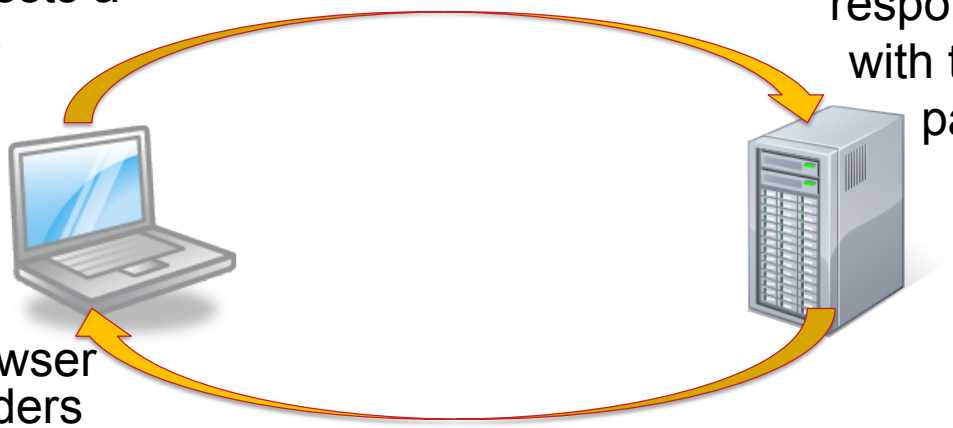
No, wait …

Ajax

Here's how Ajax works

1. Browser requests a page

2. Server responds with that page

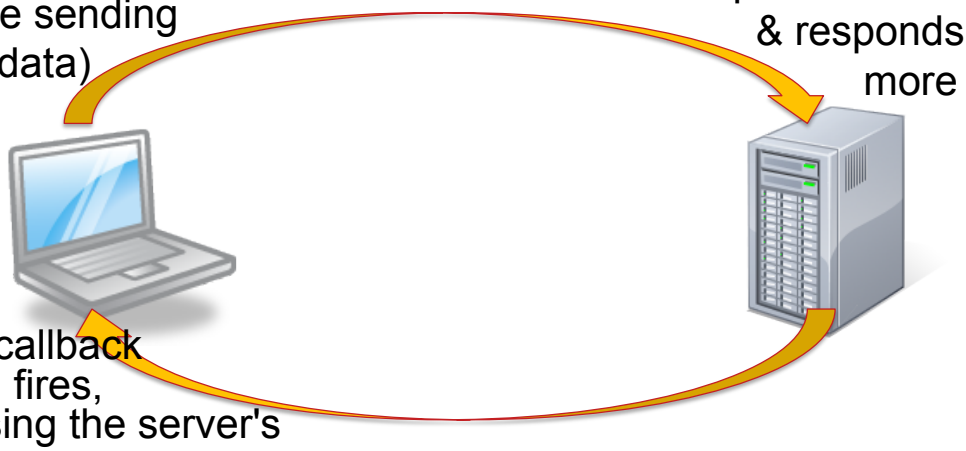


3. Browser renders the page

Here's how Ajax works

4. An Ajax call requests *data* (maybe sending some data)

5. Server process that data & responds with more *data*



6. Ajax callback function fires, processing the server's response

XMLHttpRequest

The OG of Ajax requests

The XMLHttpRequest* object's API

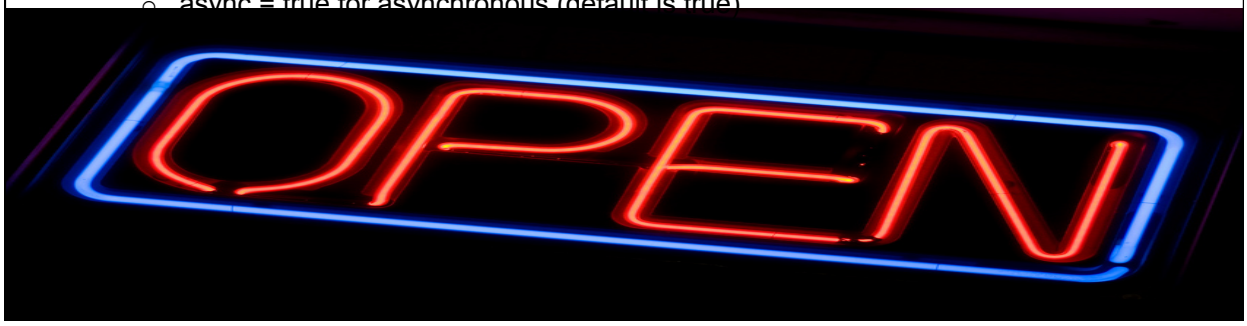
Methods	Properties	Events
open(httpMethod, url, async)	readyState	onreadystatechange
setRequestHeader(header, value)	status	
send(string)	statusText	
getResponseHeader(header)	responseText	
abort()	responseXML	

*The XMLHttpRequest object is often abbreviated "xhr"

The open() method prepares the Ajax call

```
xhr.open(httpMethod, url, async)
```

- where
  - httpMethod = "GET" or "POST" or "PUT" or "DELETE"
  - url = address we're hitting
  - async = true for asynchronous (default is true)



send() instructs the system to make the actual call

- send() usually has no arguments

```
xhr.send();
```

- You can pass in form/querystring data using send …

```
xhr.send('Artist=CeeLo');
```

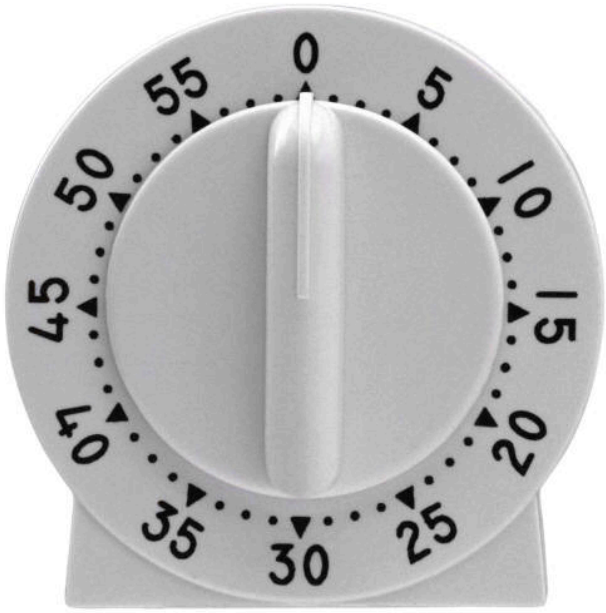
Are we ready? readyState knows!

- xhr.readyState holds a number between 0 and 4

Value	Meaning
0	Request not initialized
1	We are connected to the server
2	The request was received
3	The server is processing the request
4	Request is finished and the response has been sent back

The xhr.status is like the http response state

- 100 Informational only
- 200 OK
- 300 Moved
- 400 Bad request
- 500 Server error



So, when xhr.status is 200 and xhr.readyState is 4, we know that the request is ready

readystatechange points to a function that fires when the readyState changes

- We can use this to determine when a response comes back from the server.
- Then we test it to see if readyState=4 and status=200

Next, we process the response

xhr.responseText	xhr.responseXML
• For JSON	• For XML

But xhr.responseText is just a string and we need it to be an object. So ...

```
const data = JSON.parse(xhr.responseText);
```

First, let's create a page with all the hooks built in

```
<html>
<head>
  <script src='SayHello.js'></script>
</head>
<body>
  <input id="personId" />
  <button onclick='getPerson()'>Go!</button>
  <div id='pDiv'>
    <p>Hello there!</p>
  </div>
</body>
</html>
```

Next, we'll create the server-side program

```
<?php
$person= get_person($_GET['personId']);
echo($person);
?>
```

Then we write the call to the service

```
function getPerson() {
  const personId =
    document.getElementById('personId').value;
  xhr = new XMLHttpRequest();
  xhr.open('GET','GetPerson.php', true);
  xhr.addEventListener('readystatechange',
    changePersonInfo);
  xhr.send('personId=' + personId);
}
```

And we write the callback function

```
function changePersonInfo() {
 if (xhr.readyState===4 && xhr.status===200) {
   const p = JSON.parse(xhr.responseText);
   document.getElementById('pDiv')
     .innerHTML = `<p>Person: ${p.first}
                   ${p.last}</p>`;
  }
}
```

Piece of cake, right?

The fetch API

A more modern way to handle Ajax

fetch receives a URL and returns a promise

```
fetch("http://us.com/api/people/123").then(
  (res) => { /* do something *},
  (err) => { /* handle the error */}
);
```

That response object has an API also. It has ...

- status - 100 - 599
- statusText - "OK" or "Not found" or "No content"
- ok - shorthand for a 200-series return
- json() - Convert it from a json string to an object.
- blob() - Convert it to a raw object
- text() - Read it as a string.

A more full example

```
const out = document.getElementById('out');

fetch('/api/people/123')
 .then( res => res.json())
 .then( person => {
   out.textContent = `
     ${person.name.first}
     ${person.name.last}`;
});
```

What if it's not that simple?

```
const headers = new Headers();
headers.set("content-type","application/json");
headers.set('accept','application/json');
const payload = {
  headers: headers,
  body: '{"givenName":"Jose"}',
  method: "PATCH",
  mode: "CORS"
};
fetch(url, payload).then(
  // Handle as before
);
```

tl;dr

- Ajax is transferring data between two computers
- JSON is the favorite format for Ajax
- XMLHttpRequest object is the heart of Ajax requests
- The fetch API is great for making Ajax requests because it is easy and supported by all modern browsers with no libraries