CHAPTER 8

AJAX & JSON

WHAT IS AJAX?

Ajax lets you…

1

Request data from a server

2

Load it without refreshing the entire page

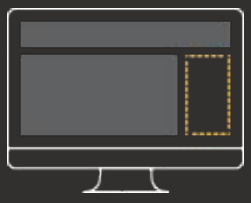
It uses an **asynchronous** processing model.

(Users can do other things while the data is loading.)

1

THE REQUEST

The browser requests information from the server



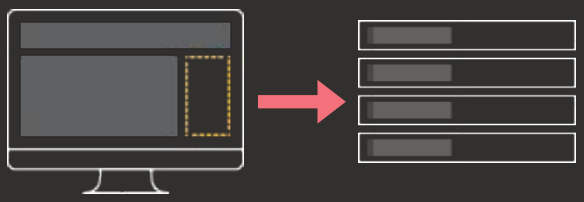
1

THE REQUEST

The browser requests information from the server

ON THE SERVER

The server responds with data (usually HTML, XML, or JSON)



1

THE REQUEST

The browser requests information from the server

ON THE SERVER

The server responds with data (usually HTML, XML, or JSON)

2

THE RESPONSE

The browser processes the content and adds it to the page

DATA FORMATS: HTML

HTML is the simplest way to get data into a page:

```
<div class="event">
  <img src="img/map-ny.png"
       alt="New York, NY" />
  <p><b>New York, NY</b>
  <br>May 30</p>
</div>
```

The browser renders this HTML like any other HTML - no extra work required.

DATA FORMATS: XML

XML looks like HTML but the tags contain different words:

```
<event>
  <location>New York, NY</location>
  <date>May 15</date>
  <map>img/map-ny.png</map>
</event>
```

You need to write JavaScript to convert the XML data into HTML so it can be displayed.

DATA FORMATS: JSON

JSON looks like object literal syntax but it is just data, not an object:

```
{
  "location": "New York, NY",
  "date": "May 30",
  "map": "img/map-ny.png"
}
```

You need to write JavaScript to convert the JSON into HTML so it can be displayed.

JSON data is made up of **keys** and **values**:

```
{
  "location": "New York, NY",
  "date": "May 30",
  "map": "img/map-ny.png"
}
```

JSON data is made up of **keys** and **values**:

```
{
  "location": "New York, NY",
  "date": "May 30",
  "map": "img/map-ny.png"
}
```

The value can be a string, number, Boolean, array, **object** or null.

You can nest objects.

```
{
  "events": [
    {
      "location": "Austin, TX",
      "date": "May 15",
      "map": "img/map-tx.png"
    },
    {
      "location": "New York, NY",
      "date": "May 30",
      "map": "img/map-ny.png"
    }
  ]
}
```


```
{
  "events": [
    {
      "location": "Austin, TX",
      "date": "May 15",
      "map": "img/map-tx.png"
    },
    {
      "location": "New York, NY",
      "date": "May 30",
      "map": "img/map-ny.png"
    }
  ]
}
```


```
{
  "events": [
    {
      "location": "Austin, TX",
      "date": "May 15",
      "map": "img/map-tx.png"
    },
    {
      "location": "New York, NY",
      "date": "May 30",
      "map": "img/map-ny.png"
    }
  ]
}
```

```
{
  "events": [
    {
      "location": "Austin, TX",
      "date": "May 15",
      "map": "img/map-tx.png"
    },
    {
      "location": "New York, NY",
      "date": "May 30",
      "map": "img/map-ny.png"
    }
  ]
}
```

JavaScript has a `JSON` object with two important methods:

1: Convert a JavaScript object to a string:

```
JSON.stringify();
```

2: Convert a string to a JavaScript object:

```
JSON.parse();
```

JSONP

Ajax only works with data from the same domain. One way to get around this is to use **JSONP**.

First, a function is included in the HTML page to process the JSON data and display it on the page:

```
<script>
  function showEvents(data) {
    // code to process & display data
  }
</script>
```

Next, a `<script>` element calls the JSON data from a remote server:

```
<script>
  function showEvents(data) {
    // code to process & display data
  }
</script>

<script
  src="http://example.org/jsonp">
</script>
```

The script then calls the function that was in the browser and passes the data to it as an argument:

```
showEvents({
  "events": [
    {
      "location": "New York, NY",
      "date": "May 30",
      "map": "img/map-ny.png"
    }...
  ]
});
```

JQUERY & AJAX

jQuery provides methods to handle Ajax requests / responses:

WORKS ON SELECTION	GLOBAL METHODS OF `jQuery` OBJECT

`.load()`	`$.get()`
	`$.post()`
	`$.getJSON()`
	`$.getScript()`
	`$.ajax()`

The `.load()` method returns the content into the jQuery selection:

```
$('#text').load('ajax.html #text');
```

jQuery provides four shorthand methods to handle specific types of Ajax requests.

`url`	where the data is fetched from
`data`	extra information for the server
`callback`	function to call when data returned
`type`	type of data to expect from server

`url`	where the data is fetched from
`data`	extra information for the server
`callback`	function to call when data returned
`type`	type of data to expect from server


```
$.get(url[, data][, callback][, type])
```

`url`	where the data is fetched from
`data`	extra information for the server
`callback`	function to call when data returned
`type`	type of data to expect from server

```
$.get(url[, data][, callback][, type])

$.post(url[, data][, callback][, type])
```

`url`	where the data is fetched from
`data`	extra information for the server
`callback`	function to call when data returned
`type`	type of data to expect from server

```
$.get(url[, data][, callback][, type])

$.post(url[, data][, callback][, type])
```


```
$.getJSON(url[, data][, callback])
```

`url`	where the data is fetched from
`data`	extra information for the server
`callback`	function to call when data returned
`type`	type of data to expect from server

```
$.get(url[, data][, callback][, type])
$.post(url[, data][, callback][, type])
$.getJSON(url[, data][, callback])
$.getScript(url[, callback])
```

There are also methods that help you deal with an Ajax response if it fails:

`.done()`	when request complete
`.fail()`	when request fails
`.always()`	complete / fail