JS

JavaScript Fundamentals

By:Ahmed ElMahdy

Cookies

What is a Cookie

- A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer.
- They are typically used for keeping track of information such as user preferences that the site can retrieve to personalize the page when user visits the website next time.
- Cookies are an old client-side storage mechanism that was originally designed for use by server-side scripting languages such as PHP, ASP, etc.
- cookies can also be created, accessed, and modified directly using JavaScript, but the process is little bit complicated and messy.

Tip: A cookie can be up to 4 KB, including its name and values, cookies that exceed this length are trimmed to fit. Also, each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.

Warning: Don't store sensitive data such as a password or credit card information in cookies since it could potentially be manipulated by the malicious user.

Cookies

Creating a Cookie in JavaScript

- In JavaScript, you can create, read, and delete cookies with the document.cookie property. This property represents all the cookies associated with a document.
- To create or store a new cookie, assign a name=value string to this property, like this:

```
document.cookie = "firstName=Christopher";
```

- A cookie value cannot contain semicolons, commas, or spaces. For this reason, you will need to use the JavaScript's built-in function encodeURIComponent() to encode the values containing these characters before storing it in the cookie.
- Likewise, you'll need to use the corresponding decodeURIComponent() function when you read the cookie value.

```
document.cookie = "name=" + encodeURIComponent("Christopher Columbus");
```

- the lifetime of a cookie is the current browser session, which means it is lost when the user exits the browser.
- For a cookie to persist beyond the current browser session, you will need to specify its lifetime (in seconds) with a max-age attribute.
- This attribute determine how long a cookie can be remain on the user's system before it is deleted, e.g., following cookie will live for 30 days.

```
document.cookie = "firstName=Christopher; max-age=" + 30*24*60*60;
```

Cookies

Creating a Cookie in JavaScript

- You can also specify the lifetime of a cookie with the expires attribute.
- This attribute takes an exact date (in GMT/UTC format) when the cookie should expire, rather than an offset in seconds.

```
document.cookie = "firstName=Christopher; expires=Thu, 31 Dec 2099 23:59:59 GMT";
```

- Here's a function that sets a cookie with an optional max-age attribute. You can also use the same function to delete a cookie by passing the value 0 for daysToLive parameter.
- a cookie is available to all web pages in the same directory or any subdirectories of that directory. However, if you specify a path the cookie is available to all web pages in the specified path and to all web pages in all subdirectories of that path.
- For example, if the path is set to / the cookie is available throughout a website, regardless of which page creates the cookie.

```
document.cookie = "firstName=Christopher; path=/";
```

Cookies

Creating a Cookie in JavaScript

- you can use the domain attribute if you want a cookie to be available across subdomains.
- cookies are available only to the pages in the domain they were set in.
- If a cookie created by a page on blog.example.com sets its path attribute to / and its domain attribute to example.com, that cookie is also available to all web pages on backend.example.com, portal.example.com.
- However, you cannot share cookies outside of a domain.

```
document.cookie = "firstName=Christopher; path=/; domain=example.com";
```

- There is also a boolean attribute named secure. If this attribute is specified, the cookie will be only be transmitted over a secure (i.e. encrypted) connection such as HTTPS.

```
document.cookie = "firstName=Christopher; path=/; domain=example.com; secure";
```

Cookies

Reading a Cookie

- Reading a cookie is a slightly more complex because the document.cookie property simply returns a string containing a semicolon and a space separated list of all cookies (i.e. name=value pairs, for example, firstName=John; lastName=Doe;).
- This string doesn't contain the attributes such as expires, path, domain, etc. that may have been set for the cookie.
- In order to get the individual cookie from this list, you need to make use of split() method to break it into individual name=value pairs, and search for the specific name.

Updating a Cookie

- The only way to update or modify a cookie is to create another cookie with the same name and path as an existing one.
- Creating a cookie with the same name but with a different path then that of an existing one will add an additional cookie.

Cookies

Deleting a Cookie

- To delete a cookie, just set it once again using the same name, specifying an empty or arbitrary value, and setting its max-age attribute to 0.
- Remember that if you've specified a path, and domain attribute for the cookie, you'll also need to include them when deleting it.

Ajax

What is Ajax?

- Ajax stands for Asynchronous Javascript And Xml.
- Ajax is just a means of loading data from the server and selectively updating parts of a web page without reloading the whole page.
- Basically, what Ajax does is make use of the browser's built-in XMLHttpRequest (XHR) object to send and receive information to and from a web server asynchronously, in the background, without blocking the page or interfering with the user's experience.
- Ajax has become so popular that you hardly find an application that doesn't use Ajax to some extent.
- The example of some large-scale Ajaxdriven online applications are: Gmail, Google Maps, Google Docs, YouTube, Facebook, Flickr, and so many other applications.

Note: Ajax is not a new technology, in fact, Ajax is not even really a technology at all. Ajax is just a term to describe the process of exchanging data from a web server asynchronously through JavaScript, without refreshing the page.

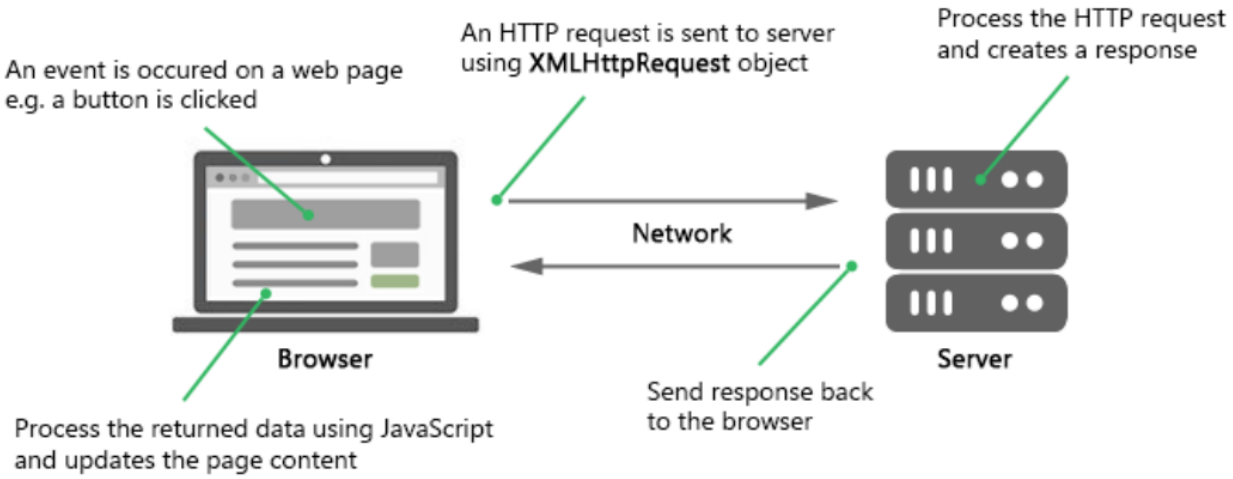
Tip: Don't get confused by the term X (i.e. XML) in AJAX. It is only there for historical reasons. Other data exchange format such as JSON, HTML, or plain text can be used instead of XML.

Ajax

Understanding How Ajax Works

To perform Ajax communication JavaScript uses a special object built into the browser—an XMLHttpRequest (XHR) object—to make HTTP requests to the server and receive data in response. All modern browsers (Chrome, Firefox, IE7+, Safari, Opera) support the XMLHttpRequest object.

The following illustrations demonstrate how Ajax communication works:



Ajax

Sending Request and Retrieving the Response

- Before you perform Ajax communication between client and server, the first thing you must do is to instantiate an XMLHttpRequest object, as shown below:
  var request = new XMLHttpRequest();
- Now, the next step in sending the request to the server is to instantiating the newly-created request object using the open() method of the XMLHttpRequest object.
- The open() method typically accepts two parameters— the HTTP request method to use, such as "GET", "POST", etc., and the URL to send the request to, like this:
  request.open("GET", "info.txt"); -Or- request.open("POST", "adduser.php");

Tip: The file can be of any kind, like .txt or .xml, or server-side scripting files, like .php or .asp, which can perform some actions on the server (e.g. inserting or reading data from database) before sending the response back to the client

- send the request to the server using the send() method of the XMLHttpRequest object.
  request.send(); -Or- request.send(body);

Note: The send() method accepts an optional body parameter which allow us to specify the request's body. This is primarily used for HTTP POST requests, since the HTTP GET request doesn't have a request body, just request headers.

Ajax

Sending Request and Retrieving the Response

- The GET method is generally used to send small amount of data to the server.
- The POST method is used to send large amount of data, such as form data.
- In GET method, the data is sent as URL parameters.
- In POST method, the data is sent to the server as a part of the HTTP request body.
- Data sent through POST method will not visible in the URL.

In the following section we'll take a closer look at how Ajax requests actually work

Ajax

Performing an Ajax GET Request

- The GET request is typically used to get or retrieve some kind of information from the server that doesn't require any manipulation or change in database, for example, fetching search results based on a term, fetching user details based on their id or name, and so on.
- When the request is asynchronous, the send() method returns immediately after sending the request. You must check where the response currently stands in its lifecycle before processing it using the readyState property of the XMLHttpRequest object.
- The readyState is an integer that specifies the status of an HTTP request.
- The function assigned to the onreadystatechange event handler called every time the readyState property changes.
- The possible values of the readyState property are summarized below.

Ajax

Performing an Ajax GET Request

Value	State	Description
0	UNSENT	An `XMLHttpRequest` object has been created, but the `open()` method hasn't been c (i.e. request not initialized).
1	OPENED	The `open()` method has been called (i.e. server connection established).
2	HEADERS_RECEIVED	The `send()` method has been called (i.e. server has received the request).
3	LOADING	The server is processing the request.
4	DONE	The request has been processed and the response is ready.

Note: Theoretically, the readystatechange event should be triggered every time the readyState property changes. But, most browsers do not fire this event when readyState changes to 0 or 1. However, all browsers fire this event when readyState changes to 4 .

- The status property returns the numerical HTTP status code of the XMLHttpRequest's response. Some of the common HTTP status codes are listed below:

✓ 200 — OK. The server successfully processed the request.
✓ 404 — Not Found. The server can't find the requested page.
✓ 503 — Service Unavailable. The server is temporarily unavailable.

Please check out the HTTP status codes reference for a complete list of response codes.

Ajax

Performing an Ajax POST Request

- The POST method is mainly used to submit a form data to the web server.
- If you are not using the FormData object to send form data, for example, if you're sending the form data to the server in the query string format, i.e. request.send(key1=value1&key2=value2) then you need to explicitly set the request header using setRequestHeader() method, like this: request.setRequestHeader("Content-type", "application/x-www-formurlencoded");
- The setRequestHeader() method, must be called after calling open(), but before calling send().
- Some common request headers are: application/x-www-formurlencoded, multipart/formdata, application/json, application/xml, text/plain, text/html, and so on

Note: The FormData object provides an easy way to construct a set of key/value pairs representing form fields and their values which can be sent using XMLHttpRequest.send() method. The transmitted data is in the same format that the form's submit() method would use to send the data if the form's encoding type were set to multipart/form-data.

Ajax

Performing an Ajax POST Request

- For security reasons, browsers do not allow you to make cross-domain Ajax requests. This means you can only make Ajax requests to URLs from the same domain as the original page, for example, if your application is running on the domain "mysite.com", you cannot make Ajax request to "othersite.com" or any other domain.
- This is commonly known as same origin policy.
- You can load images, style sheets, JS files, and other resources from any domain.

Tip: Check out the jQuery Ajax methods for quick and seamless Ajax implementation. The jQuery framework provides very convenient methods to implement Ajax functionality.