COURSE TECHNOLOGY
CENGAGE Learning

JavaScript, Sixth Edition

Chapter 11
Updating Web Pages with Ajax

Objectives

When you complete this chapter, you will be able to:

- Describe the steps involved in using Ajax to update data
- Create an HTTP request and interpret an HTTP response
- Request and receive server data using the `XMLHttpRequest` object
- Process data received from a web service and add it to the DOM
- Update app data using JSON-P

Introduction to Ajax

- Allows client web pages to quickly interact and exchange data with a web server
  - Without reloading entire web page
- Relies on
  - Programming language such as JavaScript
  - Data interchange format such as JSON or XML

Introduction to Ajax (cont'd.)

- `XMLHttpRequest` object (XHR object)
  - Uses HTTP to exchange data between a client computer and a web server
  - Can be used to request and receive data without reloading a web page

Introduction to Ajax (cont'd.)

- Combining `XMLHttpRequest` with DHTML
  - Allows update and modification to individual portions of a web page
    - With data received from a web server
- Google search suggestions
  - One of the first commercial Ajax applications

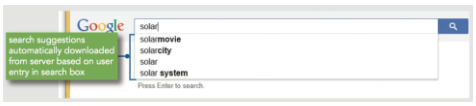

Google
search suggestions automatically downloaded from server based on user entry in search box
solar|
solarmovie
solarcity
solar
solar system
Press Enter to search.

Figure 11-1 Google search suggestions provided using Ajax

Introduction to Ajax (cont'd.)

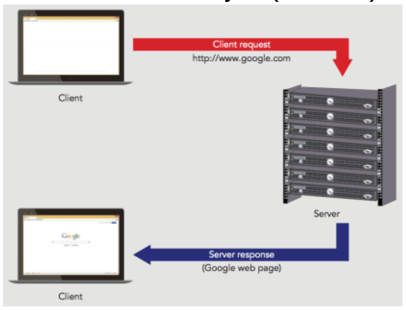


Figure 11-2 Standard HTTP request

Introduction to Ajax (cont'd.)

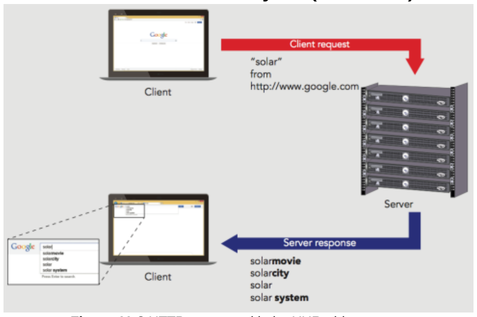


Figure 11-3 HTTP request with the XHR object

Understanding the Limitations of Ajax

- Data requested can be located on a third-party server
  - Not all web servers or browsers support this
- Can use a server-side script as a proxy to access data from another domain
- Proxy
  - Server that acts for or performs requests for other clients and servers

Accessing Content on a Separate Domain

- Web service
  - Data source made available on one domain for use on other domains across web
  - Does not contain graphical user interface or command-line interface
  - Simply provides services and data in response to requests
    - Up to the client to provide an implementation for a program calling a web service
  - Often requires API key
    - Unique identifier assigned by service to each person/organization that wants access

Accessing Content on a Separate Domain (cont'd.)

- Proxy scripts often written in PHP
  - Language specifically designed to run on web servers

Running Ajax from a Web Server

- Must open Ajax files from a web server
  - With the HTTP (http://) or HTTPS (https://) protocol
- Can install server software on any computer
- Popular web server software
  - Apache HTTP Server
  - Nginx ("engine-ex")
  - Microsoft Internet Information Services (IIS)

Working with HTTP

- Using Ajax to update data involves 4 steps:
  1. Instantiate an `XMLHttpRequest` object for the web browser where the script will run.
  2. Use the `XMLHttpRequest` object to send a request to the server.
  3. Receive the response from the server containing the requested data.
  4. Process the data returned from the server, and incorporate the data into the app.

Working with HTTP (cont'd.)

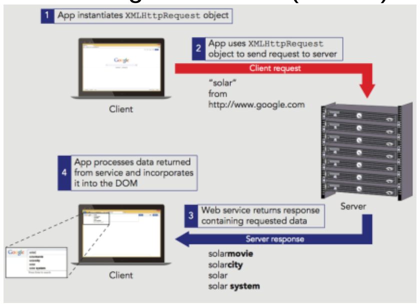

1 App instantiates XMLHttpRequest object
2 App uses XMLHttpRequest object to send request to server
Client request
"solar" from http://www.google.com
Client
Server
3 Web service returns response containing requested data
Server response
solarmovie
solarcity
solar
solar system
4 App processes data returned from service and incorporates it into the DOM
Client

Figure 11-6 Using Ajax to update data

Working with HTTP (cont'd.)

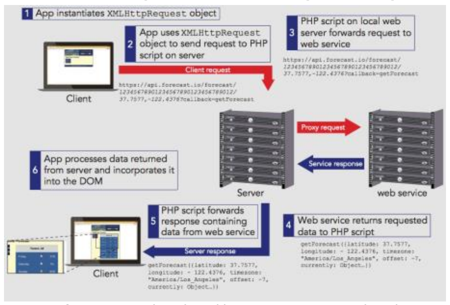

1 App instantiates XMLHttpRequest object
2 App uses XMLHttpRequest object to send request to PHP script on server
Client request
https://api.forecast.io/forecast/123456789012345678901234567890l2/37.7577,-122.4376?callback=getForecast
Client
3 PHP script on local web server forwards request to web service
https://api.forecast.io/forecast/123456789012345678901234567890l2/37.7577,-122.4376?callback=getForecast
Proxy request
Service response
Server
web service
4 Web service returns requested data to PHP script
getForecast({latitude: 37.7577, longitude: - 122.4376, timezone: "America/Los_Angeles", offset: -7, currently: Object…})
5 PHP script forwards response containing data from web service
Server response
getForecast({latitude: 37.7577, longitude: - 122.4376, timezone: "America/Los_Angeles", offset: -7, currently: Object…})
6 App processes data returned from server and incorporates it into the DOM
Client

Figure 11-7 Using Ajax with a proxy server to update data

Working with HTTP (cont'd.)

- Request
  - Process of asking for a web page from a web server
- Response
  - Web server's reply
- Uniform Resource Locator (URL)
  - A web page's unique address
  - Consists of two parts
    - Protocol (usually HTTP)
    - Web server's domain name or a web server's Internet Protocol address

Working with HTTP (cont'd.)

- Hypertext Transfer Protocol (HTTP)
  - Set of rules defining how requests made by an HTTP client to an HTTP server
  - Defines how responses returned from an HTTP server to an HTTP client
- HTTP client
  - Refers to an application (web browser) making the request
- HTTP server (another name for a web server)
  - Refers to a computer that receives HTTP requests and returns responses to HTTP clients

Working with HTTP (cont'd.)

- Host
  - Computer system being accessed by a remote computer
- W3C and Internet Engineering Task Force jointly develop HTTP
  - Version 1.1: most recent version of HTTP commonly used today
  - Version 2.0: in development
    - Modern browser already support some features

Understanding HTTP Messages

- HTTP messages
  - HTTP client requests and server responses
- HTTP client opens a connection to the server
  - Submits a request message
  - Web server returns a response message appropriate to the request type
- Format:

  Start line (request method or status returned)
  Header lines (zero or more)
  Blank line

  Message body (optional)

Understanding HTTP Messages (cont’d.)

- Headers
  - Define information about the request or response message and about the contents of the message body
- 47 HTTP 1.1 headers
  - generic headers used in request or response messages
  - headers specific to a request, response, or message body
- Format for using a header

  header: *value*

Understanding HTTP Messages (cont’d.)

- `Cache-Control` header
  - Specifies how a web browser should cache any server content it receives
- Caching
  - Temporary storage of data for faster access
  - Web browser attempts to locate any necessary data in its cache
    - Before making a request from a web server
  - Goes against the reason for using Ajax
    - Include `Cache-Control: no-cache`

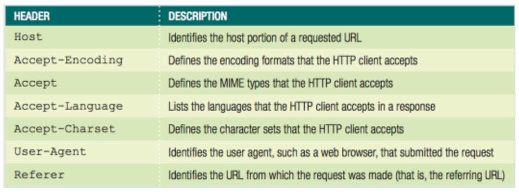
Understanding HTTP Messages (cont’d.)

- Blank line always follows last header line
  - Optional: message body can follow the blank line in the messages
- Most common types of HTTP requests
  - `GET` and `POST`
- Other HTTP request methods
  - `HEAD`, `DELETE`, `OPTIONS`, `PUT`, and `TRACE`
- Can use browser tools to examine HTTP headers

Sending HTTP Requests

- `GET` method
  - Used for standard web page requests
  - Can have a query string or form data appended to the URL
- `POST` request
  - Similar to a `GET` request except that any submitted data is included in the message body
    - Immediately following the blank line after the last header

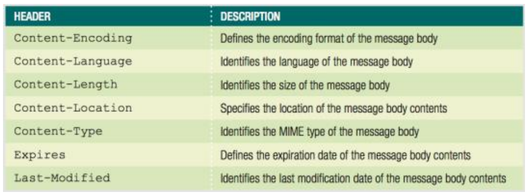
Sending HTTP Requests (cont’d.)



HEADER	DESCRIPTION
`Host`	Identifies the host portion of a requested URL
`Accept-Encoding`	Defines the encoding formats that the HTTP client accepts
`Accept`	Defines the MIME types that the HTTP client accepts
`Accept-Language`	Lists the languages that the HTTP client accepts in a response
`Accept-Charset`	Defines the character sets that the HTTP client accepts
`User-Agent`	Identifies the user agent, such as a web browser, that submitted the request
`Referer`	Identifies the URL from which the request was made (that is, the referring URL)

Table 11-1 Common request headers

Sending HTTP Requests (cont’d.)



HEADER	DESCRIPTION
`Content-Encoding`	Defines the encoding format of the message body
`Content-Language`	Identifies the language of the message body
`Content-Length`	Identifies the size of the message body
`Content-Location`	Specifies the location of the message body contents
`Content-Type`	Identifies the MIME type of the message body
`Expires`	Defines the expiration date of the message body contents
`Last-Modified`	Identifies the last modification date of the message body contents

Table 11-2 Common message body headers

Receiving HTTP Responses

- HTTP response messages
  - Take the same format as request messages
  - Return protocol and version of the HTTP server
    - Along with a status code and descriptive text
- Status codes format
  - 1*xx*: (informational) - Request received
  - 2*xx*: (success) - Request successful
  - 3*xx*: (redirection) - Request cannot be completed without further action
  - 4*xx*: (client error) - Request cannot be fulfilled due to a client error

Receiving HTTP Responses (cont'd.)

- 5*xx*: (server error) - Request cannot be fulfilled due to a server error

CODE	TEXT	DESCRIPTION
200	OK	The request was successful.
301	Moved Permanently	The requested URL has been permanently moved.
302	Moved Temporarily	The requested URL has been temporarily moved.
304	Not Modified	The client already has the current version of the requested content.
404	Not Found	The requested URL was not found.
500	Internal Server Error	The request could not be completed due to an internal server error.

Table 11-3 Common response codes

Receiving HTTP Responses (cont'd.)

- Zero or more response headers follow the status line
- Response returned from a server
  - Can be much more involved than original request that generated it

HEADER	DESCRIPTION
`Vary`	Determines whether the server can respond to subsequent requests with the same response
`Server`	Returns information about the server software that processed the request
`Location`	Redirects clients to a different URI

Table 11-4 Common response headers

Requesting Server Data

- `XMLHttpRequest` object
  - Key to incorporating Ajax in JavaScript code
  - Allows use of use JavaScript and HTTP to exchange data between a web browser and a web server

Requesting Server Data (cont'd.)

METHOD	DESCRIPTION
`abort()`	Cancels the current HTTP request
`getAllResponseHeaders()`	Returns a text string containing all of the headers that were returned with a response in *header: value* format, separated by line breaks
`getResponseHeader(`*`header_name`*`)`	Returns a text string containing the value assigned to the specified header
`open(`*`method,URL[,async,user,password]`*`)`	Specifies the method and URL for an HTTP request; assigning a value of `true` to the *`async`* argument performs the request asynchronously, while a value of `false` performs the request synchronously; the default is `true`
`send([`*`content`*`])`	Submits an HTTP request using the information assigned with the `open()` method; the optional *`content`* argument contains the message body
`setRequestHeader(`*`header_name,value`*`)`	Creates an HTTP header using the *`header_name`* and *`value`* arguments

Table 11-5 `XMLHttpRequest` object methods

Requesting Server Data (cont'd.)

PROPERTY	DESCRIPTION
`onreadystatechange`	Specifies the name of the event handler function that executes whenever the `readyState` property value changes
`readyState`	Contains one of the following values, which represent the state of the HTTP request: 0 (uninitialized), 1 (open), 2 (sent), 3 (receiving), or 4 (loaded)
`responseText`	Contains the HTTP response as a text string, such as a JSON string
`responseXML`	Contains the HTTP response as an XML document
`status`	Contains the HTTP status code (such as 200 for "OK" or 404 for "Not Found") that was returned with the response
`statusText`	Contains the HTTP status text (such as "OK" or "Not Found") that was returned with the response

Table 11-6 `XMLHttpRequest` object properties

Instantiating an `XMLHttpRequest` Object

- Use the `XMLHttpRequest` constructor

```
var httpRequest = new XMLHttpRequest();
```

- Originally created specifically to request XML data
  - Name hasn't changed, but now capable of more
- Most JavaScript programmers use a series of nested `try/catch` statements
- Opening and closing HTTP connections is a bottleneck in page loading
  - HTTP/1.1 automatically keeps the client-server connection open unless it is specifically closed
- Can make Ajax programs faster by reusing an instantiated `XMLHttpRequest` object

Instantiating an `XMLHttpRequest` Object (cont'd.)

```
var curRequest = false;
var httpRequest;
function getRequestObject() {
  try {
    httpRequest = new XMLHttpRequest();
  }
  catch (requestError) {
    document.getElementById("main").innerHTML = "Your↵
      browser does not support this content";
    return false;
  }
  return httpRequest;
}
if (!curRequest) {
  curRequest = getRequestObject();
}
```

Opening and Sending a Request

- Use the `open()` method with the instantiated `XMLHttpRequest` object
  - To specify the `request` method (`GET` or `POST`) and URL
- `open()` method accepts three optional arguments
  - `async`, *`username`*, *`password`*
- `abort()` method
  - Used to cancel any existing HTTP requests before beginning a new one

Opening and Sending a Request (cont'd.)

- `send()` method
  - Submit the request to the server
  - Accepts a single argument containing the message body
- POST requests more involved
  - Must manually build name-value pairs to submit
  - Must submit at least `Content-Type` header before `send()` method
  - Also should submit `Content-Length` header and `Connection` header

Receiving Server Data

- `responseXML` property
  - Contains the HTTP response as an XML document
  - Only if server response includes the `Content-Type` header with a MIME type value of `text/xml`
- `responseText` property
  - Contains the HTTP response as a text string

Processing XML Data in a Response

- Assign property values to document nodes
  - Assign value of `responseXML` property to a variable
  - Use `innerHTML` and node properties to assign values of XML document stored in variable to appropriate elements

Processing Text Data in a Response

- `responseText` value almost always a JSON string
  - First use `JSON.parse()` to convert to object
  - Then access property values of new object and add to DOM elements

Sending and Receiving Synchronous Requests and Responses

- Synchronous request
  - Stops the processing of the JavaScript code until a response returned from the server
- Check `XMLHttpRequest` object's `status` property value
  - Ensure response received successfully

Sending and Receiving Synchronous Requests and Responses (cont'd.)

- Synchronous responses
  - Easier to handle
- Drawback
  - Script will not continue processing until the response is received
- Use asynchronous requests with the `send()` method

Sending and Receiving Asynchronous Requests and Responses

- Asynchronous request
  - Allows JavaScript to continue processing while it waits for a server response
- Create an asynchronous request
  - Pass a value of `true` as the third argument of the `open()` method
    - Or omit the argument altogether
- Receive a response
  - Use the `XMLHttpRequest` object's `readyState` property and `onreadystatechange` event

Sending and Receiving Asynchronous Requests and Responses (cont'd.)

- Example:

```
stockRequest.abort();
stockRequest.open("get","StockCheck.php?" + "checkQuote=" +↵
  tickerSymbol, true);
stockRequest.send(null);

stockRequest.onreadystatechange = fillStockInfo;
```

Sending and Receiving Asynchronous Requests and Responses (cont'd.)

- Value assigned to the `readyState` property
  - Updated automatically
    - According to the current statement of the HTTP request
- If property assigned a value of 4
  - Response finished loading

Sending and Receiving Asynchronous Requests and Responses (cont'd.)

- Example:

```
function fillStockInfo() {
  if (stockRequest.readyState === 4 && stockRequest.status↵
    === 200) {
    var stockValues = stockRequest.responseText;
    document.getElementById("ticker").innerHTML =↵
      stockValues.ticker;
    ...
  }
}
```

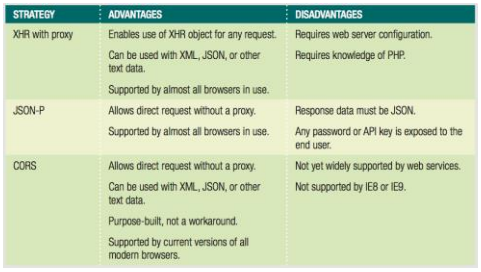
Refreshing Server Data Automatically

- Automatically refresh data obtained from an HTTP server
  - Use JavaScript's `setTimeout()` or `setInterval()` methods
    - Send request to the server
    - Read and process the data returned from the server

Creating Cross-Domain Requests Without a Proxy Server

- Two alternatives to proxies for working around same-origin policy
  - JSON-P (JSON with padding)
    - Requests JSON content using a `script` element rather than an XHR object
  - CORS (Cross-Origin Resource Sharing)
    - Server sends special response header that indicates data may be used on other domains

Creating Cross-Domain Requests Without a Proxy Server (cont'd.)



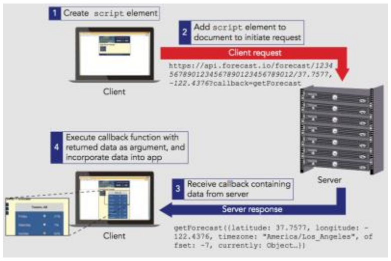
STRATEGY	ADVANTAGES	DISADVANTAGES
XHR with proxy	Enables use of XHR object for any request. Can be used with XML, JSON, or other text data. Supported by almost all browsers in use.	Requires web server configuration. Requires knowledge of PHP.
JSON-P	Allows direct request without a proxy. Supported by almost all browsers in use.	Response data must be JSON. Any password or API key is exposed to the end user.
CORS	Allows direct request without a proxy. Can be used with XML, JSON, or other text data. Purpose-built, not a workaround. Supported by current versions of all modern browsers.	Not yet widely supported by web services. Not supported by IE8 or IE9.

Table 11-7 Comparison of XHR proxy, JSON-P, and CORS

Updating Content with JSON-P

- `script` element not subject to same-origin policy
- Program running on web server returns content
  - JSON object treated as parameter for function call
  - Called function processes JSON object

Updating Content with JSON-P (cont'd.)


1 Create script element
2 Add script element to document to initiate request
Client request
Client
Server
4 Execute callback function with returned data as argument, and incorporate data into app
3 Receive callback containing data from server
Server response
Client

```
https://api.forecast.io/forecast/1234
5678901234567890123456789012/37.7577,
-122.4376?callback=getForecast
```

```
getForecast({latitude: 37.7577, longitude: -
122.4376, timezone: "America/Los_Angeles", of
fset: -7, currently: Object…})
```

Figure 11-14 Using JSON-P to update data

Updating Content with JSON-P (cont'd.)

- JSON-P URL generally consists of 2 parts:
  - Request information
    - URL of service, parameters
  - Callback query string
    - Keyword (usually "callback") & name of function to call

Updating Content with JSON-P (cont'd.)

- JSON-P opens a security hole in your website
  - If data source compromised, content you receive is a potential attack route on your site
  - Use JSON-P only with web service you trust
- JSON-P exposes API key or password to end users
  - Use only with trusted users, such as employees

Updating Content with CORS

- Cross-domain request within an XHR object
- Part of XMLHttpRequest2 specification
  - Additional properties, methods, and events for XHR object
- Enables content provider to convey permission
  - `Access-Control-Allow-Origin` HTTP response header
    - Value includes requesting domain
  - `XDomainRequest` object (Microsoft)
    - Must check first if browser defines this object

Summary

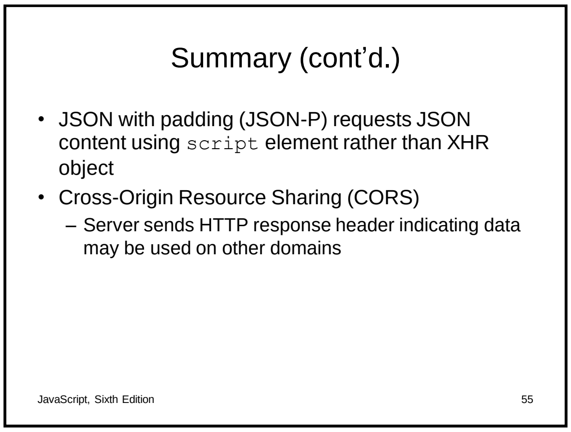
- Ajax allows data exchange with web server without reloading page
- `XMLHttpRequest` object
  - Uses HTTP to exchange data between a client computer and a web server
- Proxy is common technique for working around same-origin policy with Ajax
- HTTP defines rules for requests and responses between client and server

Summary (cont’d.)

- Use methods and properties of an instantiated `XMLHttpRequest` object with JavaScript
- First step to exchange data between an HTTP client and a web server
  - Instantiate an `XMLHttpRequest` object
- To improve performance
  - Call the `abort()` method of the `XMLHttpRequest` object
- Use the `send()` method with the instantiated `XMLHttpRequest` object to submit the request to the server

Summary (cont’d.)

- Server response assigned to `responseXML` or `responseText` property
- Synchronous request stops the processing of the JavaScript code until a response returned
- Asynchronous request allows JavaScript to continue processing while waiting for a server response
- `readystatechange` event fires when value assigned to `readyState` property changes



Summary (cont'd.)

- JSON with padding (JSON-P) requests JSON content using `script` element rather than XHR object
- Cross-Origin Resource Sharing (CORS)
  - Server sends HTTP response header indicating data may be used on other domains

JavaScript, Sixth Edition 55