

Lecture Sheet on “PHP+MySQL” Day-17: (Ajax)

Ajax	Time: 2 hours
-------------	----------------------

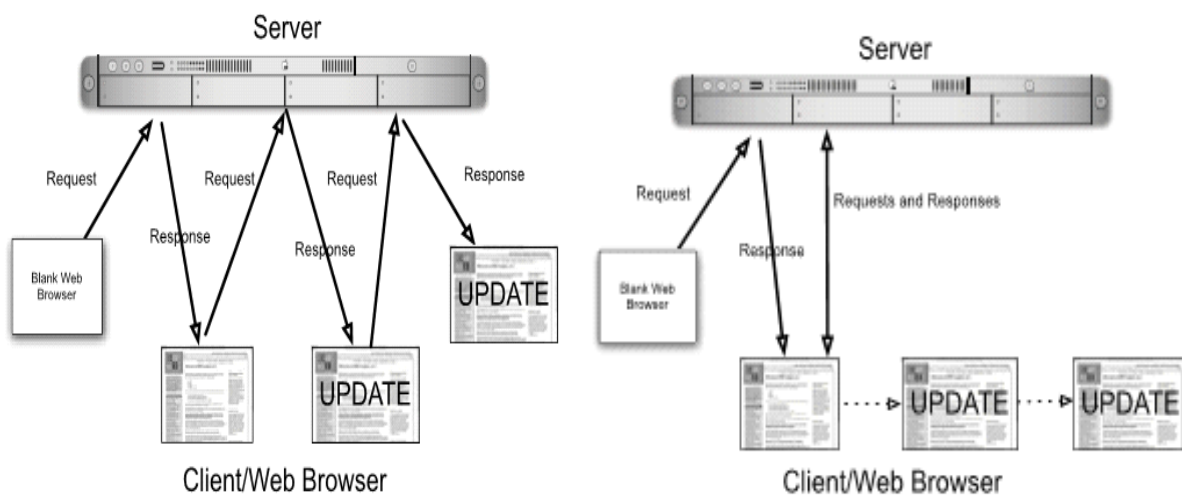
19. Ajax (Day-1)

Time: 3 hours

AJAX (Asynchronous Java and XML) has emerged as a powerful platform for building web applications with extensive client-side interactivity. Unlike older approaches, which require reloading of the entire page with every postback, AJAX uses the JavaScript DOM, the XMLHttpRequest object, XML, and CSS to download and display just the content that needs to change.

OBJECTIVES	TOPICS
<ul style="list-style-type: none"> ❖ Know the basics of Ajax ❖ Advantage and disadvantages ❖ Checking availabilities with ajax 	<ul style="list-style-type: none"> ❖ What is AJAX? ❖ Overview of Ajax ❖ Ajax advantage and disadvantages. ❖ Checking availability with Ajax

Of all the buzzwords to enter the computer lexicon in the past couple of years, Ajax may be the “buzziest.” **Ajax, which stands for Asynchronous JavaScript and XML**, changes the client/server relationship so that server interactions can take place without any apparent action on the part of the client. In truth, Ajax is just a label given to functionality that’s been present for years, but sometimes a good label helps, and when a powerhouse like Google uses Ajax(for Gmail, Google Suggest, and more), people pay attention.



In this chapter we provide an introduction to the Ajax concept, with PHP and MySQL as its back end. As Ajax is still comparatively new, there are new ideas and debates for each aspect of the technology, but in two different examples we can demonstrate the entire soup-to-nuts of what Ajax is about and, in the process, provide real-world code. Because the heart of

Ajax really is JavaScript and the **Document Object Model (DOM)**, the PHP in this chapter will be basic. As sophisticated JavaScript may be new territory for you, the chapter concludes with a discussion of debugging techniques, as resolving Ajax problems requires special approaches.

Introduction to Ajax

Before getting into the code, you should understand what, exactly, Ajax is and how it differs from what you currently do with PHP. As you know, PHP is primarily a Web-based technology. This means that PHP does its thing whenever a server request is made. A user goes to a Web page or submits a form; the request is made of the server; PHP handles that request and returns the result. Each execution of some PHP code requires an active request and a redrawing on the Web browser. For the end user, this means they see their browser leave the current page, access the new one, download the new page's content, and display the content, repeating as necessary.

The secret to Ajax is that it can make the server request happen behind the scenes while still changing what the user sees as if they actively made the server request. JavaScript is really the key technology here.

With an Ajax transaction, JavaScript, which does its thing within a Web browser, makes the request of the server and then handles that request.

The Web page can then be updated without ever seeming to leave the current page. In action, this might mean:

1. The end user goes to a Web page.
2. The user types something in a box, clicks a button, or drags something with their cursor.
3. Whatever the user does in Step 2 triggers JavaScript to request something from the server.
4. The server handles that request (using PHP in this chapter), returning some data.
5. JavaScript receives that data and uses it to update the Web page, without reloading it.

Looking at all the technologies involved, you start with (X)HTML, which is the foundation of all Web pages. Then there's JavaScript, which runs in the Web browser, asks for and receives data from the server, and manipulates the HTML by referring to the Document Object Model (DOM, an object-based representation of the elements of a Web page).

And finally, on the server, you have our friend PHP. The last technology commonly involved is XML (Extensible Markup Language), which can be used in the data transfer. I say “can be,” because you don’t have to use XML. In fact, this chapter has two examples—one simple, one complex—without any XML.

: Tip **Jesse James Garrett** coined the term Ajax in February 2005. He has since claimed that Ajax is not an acronym. It just, you know, seems like an acronym.

What Is Asynchronous?

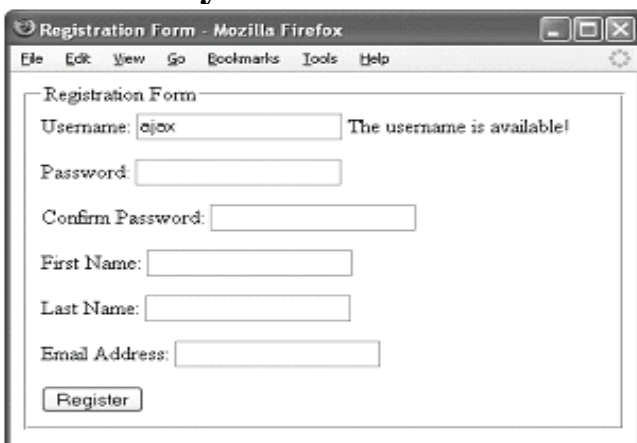


Figure 13.4 The availability of the requested username will be indicated prior to submitting the form to the server.

One of the key components of Ajax is its asynchronous behavior. This term, which literally means not synchronous, refers to a type of communication where one side (say, the client) does not have to wait for the other side (the server) before doing something else. In an asynchronous transaction, the user can do something that makes the JavaScript request data from the server. While the JavaScript is awaiting that data, which it’ll then display in the Web browser, the user is free to do other things, including other things that involve JavaScript. So the asynchronicity makes the experience more seamless. On the other hand, you can perform synchronous transactions, if you are in a situation where the user should wait until the data is in.

A Simple Example

One of the best uses of Ajax starts with a commonplace Web presence: a registration form. When using PHP to validate a registration form, there are many steps you’d take, from checking that all the required fields are filled out to validating the values in certain fields (e.g., a valid email address, a valid date of birth). If the registration process requires a

username for logging in, PHP would also check the database to see if that username is available. With a traditional model, this check couldn't take place until the form was submitted. With Ajax, this check can take place prior to form submission, saving the user from having to go back, think up a new username, and resubmit the form, repeating as necessary.

In developing this Ajax example, we'll work the application backward, which makes it easiest to understand the whole process (it also leaves the newer information toward the end). For the server-side stuff, it's really basic PHP and MySQL. We won't develop it fully, but you shouldn't have a problem fleshing it out when it comes time for you to implement this in a real site.

Create the users table

```
CREATE TABLE users (  
  user_id INT UNSIGNED NOT NULL  
  ? AUTO_INCREMENT,  
  username VARCHAR(20) NOT NULL,  
  userpass CHAR(40) NOT NULL,  
  first_name VARCHAR(20) NOT NULL,  
  last_name VARCHAR(40) NOT NULL,  
  email VARCHAR(60) NOT NULL,  
  PRIMARY KEY (user_id),  
  UNIQUE (username)  
);
```

Programming the PHP

In discussing this example, I state that the Ajax aspect of the application will be to check that a username is available prior to submitting the form to the server. This PHP script, which is the server side of the Ajax process, just needs to check the availability of a provided username.

The script will be written so that it accepts a username in the URL; the script is invoked using:

`http://hostname/checkusername.php?username=XXXX`

The PHP script then runs a query on the database to see if this username—XXXX—has already been registered. Finally the script should return a message indicating the availability of that username. This should all be quite easy for even a beginning PHP programmer.

One little quirk with this page: It will not use any HTML (gasp!). As this page won't be intended to be viewed directly in a Web browser, HTML is unnecessary.

```
1 <?php # Script Ajax - checkusername.php
```

```
2
3 /* This page checks a database to see if
4 * $_GET['username'] has already been registered.
5 * The page will be called by JavaScript.
6 * The page returns a simple text message.
7 * No HTML is required by this script!
8 */
9
10 // Validate that the page received $_GET['username']:
11 if (isset($_GET['username'])) {
12
13 // Connect to the database:
14 $dbc = @mysqli_connect ('localhost', 'username', 'password', 'test') OR die
('The
availability of this username will be confirmed upon form submission.');
```

```
15
16 // Define the query:
17 $q = sprintf("SELECT user_id FROM users WHERE username='%s'",
mysqli_real_escape_string($dbc,
trim($_GET['username'])));
18
19 // Execute the query:
20 $r = mysqli_query($dbc, $q);
21
22 // Report upon the results:
23 if (mysqli_num_rows($r) == 1) {
24 echo 'The username is unavailable!';
25 } else {
26 echo 'The username is available!';
27 }
28
29 mysqli_close($dbc);
30
31 } else { // No username supplied!
32
33 echo 'Please enter a username.';
34
35 }
36 ?>
```

Writing the JavaScript, part 1

For many PHP programmers the JavaScript will be the most foreign part of the Ajax process. It's also the most important, so we'll go through the thinking in detail. Let's start breaking the JavaScript code into two sections:

the creation of the XMLHttpRequest object and the use of said object. The second chunk of code will be application-specific.

To start, you'll need to create an object through which all the magic happens. For

this purpose, there is the XMLHttpRequest class. The functionality defined within this class is at the heart of all Ajax capabilities. If the user's browser supports the

XMLHttpRequest (which I'll show you how to test for in the next script), you'll want to make an object of this type:

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

Windows Internet Explorer 5.x and 6.x do not support XMLHttpRequest and instead

an ActiveXObject: need to use

```
var ajax = new ActiveXObject("Microsoft.XMLHTTP");
```

And that's really it to creating the object. Even if you've never done any JavaScript programming before, this is easy enough to follow. In this next script, I'll formally implement the two lines of code but wrap it in some conditionals and **try...catch** blocks.

```
1 // Script - ajax.js
2
3 /* This page creates an Ajax request object.
4 * This page is included by other pages that
5 * need to perform an XMLHttpRequest.
6 */
7
8 // Initialize the object:
9 var ajax = false;
10
11 // Create the object...
12
13 // Choose object type based upon what's supported:
14 if (window.XMLHttpRequest) {
15
16 // IE 7, Mozilla, Safari, Firefox,
17 // Opera, most browsers:
18 ajax = new XMLHttpRequest();
19
20 } else if (window.ActiveXObject) { //
21 // Older IE browsers
22
23 }
24
25 // Create type Msxml2.XMLHTTP, if
```


possible:

```
22try {
23ajax = new
ActiveXObject("Msxml2.XMLHTTP");
24} catch (e1) { // Create the older
type instead:
25try {
26ajax = new
ActiveXObject("Microsoft.XMLHTTP");
27} catch (e2) { }
28}
29
30}
31
32// Send an alert if the object wasn't
created.
33if (!ajax) {
34alert ('Some page functionality is
unavailable.');
```

To write some of the JavaScript:

1. Create a new JavaScript script in your text editor or IDE

// - ajax.js

2. Initialize a variable.

```
var ajax = false;
```

In JavaScript you should declare variables var. Here I declare the using the keyword variable (called ajax, but it could be called anything) and initially set its value to false.

3. Create the XMLHttpRequest object, if supported.

```
if (window.XMLHttpRequest) {ajax = new XMLHttpRequest();
```

If the browser supports the XMLHttpRequest object type, a new object of that typewill be created.

This will be the case for most browsers, including Internet Explorer 7 (but not earlier versions), Firefox, Mozilla, Safari, and Opera.

4. Check if ActiveXObject is supported.

```
} else if (window.ActiveXObject) {
```

If the browser does not support **XMLHttpRequest**, then an object type of ActiveXObject should be created.

This only applies to Internet Explorer versions prior to 7.

5. Try to create the ActiveXObject object.

```
try {  
    ajax = new  
    ? ActiveXObject("Msxml2.XMLHTTP");  
} catch (e1) {  
    try {  
        ajax = new  
        ? ActiveXObject("Microsoft.XMLHTTP");  
    } catch (e2) { }  
}
```

Microsoft has developed several different flavors of ActiveXObject XMLHTTP.

Ideally, you'd like to use the most current one. Since there's no way of testing what versions are supported by the current browser, two try...catch blocks will attempt to make the object. If an object of type Msxml2.XMLHTTP cannot be created, then an attempt is made to make one of type Microsoft.XMLHTTP.

Nothing is done with any caught exceptions, as a more general error handling later in the script will suffice.

6. Complete the main conditional.

```
}
```

This closes the else if clause begun in Step 4.

7. Send an alert if the object wasn't created.

```
if (!ajax) {  
    alert ('Some page functionality  
    ? is unavailable.');
```

```
}
```

If ajax is false, meaning that none of the attempts to declare it as an object succeeded, then the entire Ajax application won't work. In such a case, a simple message is displayed to the end user although you don't have to do that.

8. Save the file as ajax.js and place it in your Web directory.

This file should be put in the same directory as checkusername.php.

Common JavaScript Events

EVENT	OCCURS WHEN...
onfocus	An element gains focus.
onblur	An element loses focus.
onchange	A form element's value or state changes.
onreset	The form is reset.
onsubmit	The form is submitted.
onclick	A mouse is clicked on an element.
onload	The HTML page has completely loaded.

XMLHttpRequest readyState Values

Value	Meaning
0	initialized
1	loading
2	loaded
3	interactive
4	complete

```
1 // - checkusername.js
2
3 /* This page does all the magic for applying
4 * Ajax principles to a registration form.
5 * The users's chosen username is sent to a PHP
```

Common HTTP Status Codes	
CODE	MEANING
200	OK
204	No content
400	Bad request
401	Unauthorized
403	Forbidden
404	Not found
408	Timeout
500	Internal server error

6 * script which will confirm its

availability.

```
7*/
8
9 // Function that starts the Ajax process:
10function check_username(username) {
11
12// Confirm that the object is usable:
13if (ajax) {
14
15// Call the PHP script.
16// Use the GET method.
17// Pass the username in the URL.
18 ajax.open('get', 'checkusername.php?username=' +
encodeURIComponent(username));
19
20// Function that handles the response:
21ajax.onreadystatechange = handle_check;
22
23// Send the request:
24 ajax.send(null);
25
26} else { // Can't use Ajax!
27document.getElementById('username_label').innerHTML = 'The availability
of this username
will be confirmed upon form submission.';
28}
29
30} // End of check_username() function.
31
32// Function that handles the response from the PHP script:
```

```
33function handle_check() {  
34  
35// If everything's OK:  
36if ( (ajax.readyState == 4) && (ajax.status == 200) ) {  
37  
38// Assign the returned value to a document element:  
39document.getElementById('username_label').innerHTML =  
ajax.responseText;  
40  
41}  
42  
43} // End of handle_check() function.
```

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
3 <html xmlns="http://www.w3.org/1999/xhtml">  
4 <head>  
5 <title>Registration Form</title>  
6 <script src="ajax.js" type="text/javascript"  
language="javascript"></script>  
7 <script src="checkusername.js" type="text/javascript"  
language="javascript"></script>  
8 </head>  
9 <body>  
10<!-- register.html  
11/*This page has the HTML registration form.  
12*The JavaScript code is included in the HEAD.  
13*/  
14-->  
15<form action="checkusername.php" method="post">  
16<fieldset>  
17<legend>Registration Form</legend>  
18  
19<p>Username: <input name="username" type="text" size="20"  
maxlength="20"  
onchange="check_username(this.form.username.value)" /> <span  
id="username_label"></span></p>  
20  
21<p>Password: <input name="pass1" type="password" /></p>  
22  
23<p>Confirm Password: <input name="pass2" type="password" /></p>  
24  
25<p>First Name: <input name="first_name" type="text" size="20"  
maxlength="20" /></p>  
26  
27<p>Last Name: <input name="last_name" type="text" size="20"  
maxlength="20" /></p>
```



28

29<p>Email Address: <input name="email" type="text" size="20"
maxlength="60" /></p>

30

31<input name="submit" type="submit" value="Register" />

32

33</form>

34</fieldset>

35</body>

36</html>