T. JOHN INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

THIRD INTERNAL QUESTION BANK– 6th Semester

Subject Code : 18CS63 Subject Name: Web Technology and It's Applications

1. Explain $_GET and $_POST superglobal arrays with appropriate diagrams.
2. Explain the use of $_FILES array.
3. How do you achieve data encapsulation in PHP? Give example.
4. How to access methods and properties of class in PHP. Explain the use of static members
5. Explain the support for inheritance in PHP with UML class diagram
6. How is array defined in PHP? Explain the array operations of PHP.
7. Explain procedural error handling and object oriented exception handing with suitable code segments.
8. How do you read or write a file on the server from PHP? Give example.
9. Explain three approaches to restrict the file size in file upload with suitable code example.
10. Define constructor and discuss the concept of inheritance, polymorphism and object interface with respect to OOP.
11. Explain the $_SERVER array
12. Explain Serialization? What are its applications?
13. Explain the working of cookie with diagram
14. What are cookies? What is the purpose of it? Demonstrate cookies with PHP program
15. With suitable example explain AJAX GET and POST request.
16. Why is state is a problem for web applications? Explain.
17. Write short notes on i) XML ii) XPath
18. Discuss jQuery selectors in detail.
19. Discuss the following:- i) Session Cookies ii) Persistent Cookies iii) session state
20. What is session storage. how is it configured?

9.3 $_SERVER Array

HANDS-ON EXERCISES

LAB 9 EXERCISE
Using the $_SERVER Superglobal

11

The $_SERVER associative array contains a variety of information. It contains some of the information contained within HTTP request headers sent by the client. It also contains many configuration options for PHP itself, as shown in Figure 9.11.

To use the $_SERVER array, you simply refer to the relevant case-sensitive key name:

```
echo $_SERVER["SERVER_NAME"] . "<br/>";
echo $_SERVER["SERVER_SOFTWARE"] . "<br/>";
echo $_SERVER["REMOTE_ADDR"] . "<br/>";
```

It is worth noting that because the entries in this array are created by the web server, not every key listed in the PHP documentation will necessarily be available. A complete list of keys contained within this array is listed in the online PHP documentation, but we will cover some of the critical ones here. They can be classified into keys containing request header information and keys with information about the server settings (which is often configured in the php.ini file).

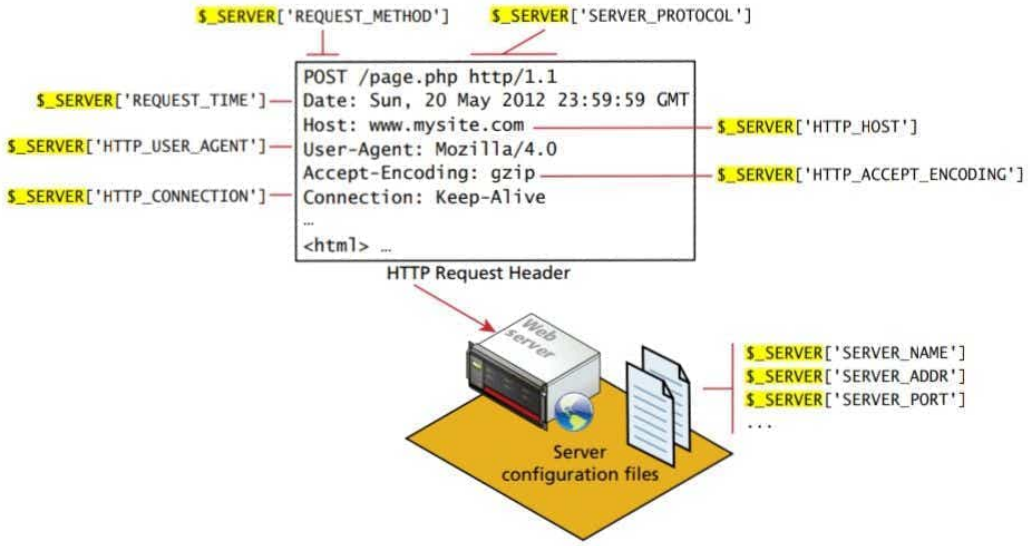


FIGURE 9.11 Relationship between request headers, the server, and the $_SERVER array

9.3.1 Server Information Keys

SERVER_NAME is a key in the $_SERVER array that contains the name of the site that was requested. If you are running multiple hosts on the same code base, this can be a useful piece of information. SERVER_ADDR is a complementary key telling us the IP of the server. Either of these keys can be used in a conditional to output extra HTML to identify a development server, for example.

DOCUMENT_ROOT tells us the file location from which you are currently running your script. Since you are often moving code from development to production, this key can be used to great effect to create scripts that do not rely on a particular location to run correctly. This key complements the SCRIPT_NAME key that identifies the actual script being executed.

9.3.2 Request Header Information Keys

Recall that the web server responds to HTTP requests, and that each request contains a request header. These keys provide programmatic access to the data in the request header.

The REQUEST_METHOD key returns the request method that was used to access the page: that is, GET, HEAD, POST, PUT.

The REMOTE_ADDR key returns the IP address of the requestor, which can be a useful value to use in your web applications. In real-world sites these IP addresses are often stored to provide an audit trail of which IP made which requests, especially on sensitive matters like finance and personal information. In an online poll, for example, you might limit each IP address to a single vote. Although these can be forged, the technical competence required is high, thus in practice one can usually assume that this field is accurate.

One of the most commonly used request headers is the user-agent header, which contains the operating system and browser that the client is using. This header value can be accessed using the key HTTP_USER_AGENT. The user-agent string as posted in the header is cryptic, containing information that is semicolon-delimited and may be hard to decipher. PHP has included a comprehensive (but slow) method to help you debug these headers into useful information. Listing 9.10 illustrates a script that accesses and echoes the user-agent header information.

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];

$browser = get_browser($_SERVER['HTTP_USER_AGENT'], true);
print_r($browser);
?>
```

LISTING 9.10 Accessing the user-agent string in the HTTP headers

One can use user-agent information to redirect to an alternative site, or to include a particular style sheet. User-agent strings are also almost always used for analytic purposes to allow us to track which types of users are visiting our site, but this technique is captured in later chapters.

PRO TIP

In order for get_browser() to work, your php.ini file must point the browscap setting to the correct location of the browscap.ini file on your system. A current browscap.ini file can be downloaded from php.net.[3] Also, this function is very complete, but slow. More simplistic string comparisons are often used when only one or two aspects of the user-agent string are important.

HTTP_REFERER is an especially useful header. Its value contains the address of the page that referred us to this one (if any) through a link. Like HTTP_USER_AGENT, it is commonly used in analytics to determine which pages are linking to our site.

Listing 9.11 shows an example of context-dependent output that outputs a message to clients that came to this page from the search page, a message that is not shown to clients that came from any other link. This allows us to output a link back to the search page, but only when the user arrived from the search page.

```
$previousPage = $_SERVER['HTTP_REFERER'];
// Check to see if referer was our search page
if (strpos("search.php",$previousPage) != 0) {
   echo "<a href='search.php'>Back to search</a>";
}
// Rest of HTML output
```

LISTING 9.11 Using the HTTP_REFERER header to provide context-dependent output

SECURITY

12

•**Serialization** is the process of taking a complicated object and reducing it down to zeros and ones for either storage or transmission.

•In PHP objects can easily be reduced down to a binary string using the **serialize()** function.

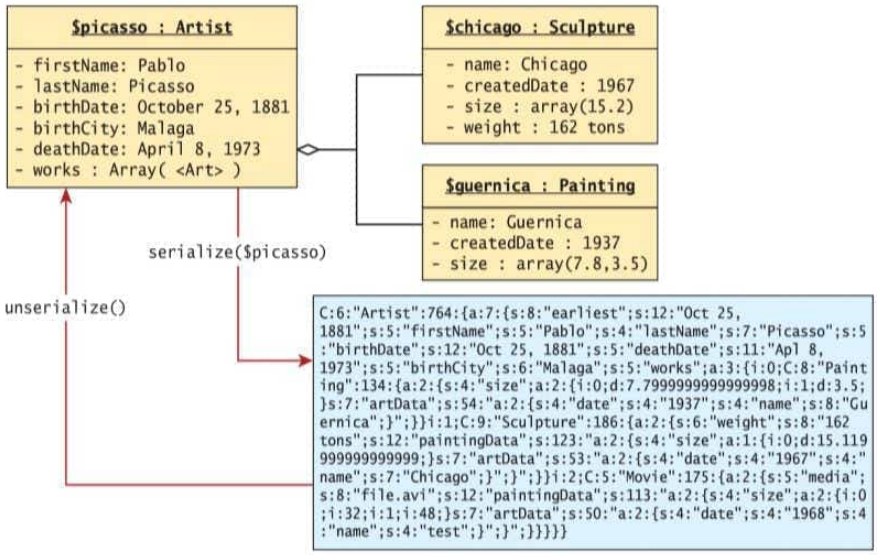
•The string can be reconstituted back into an object using the **unserialize()** method

```
interface Serializable {
   /* Methods */
   public function serialize();
   public function unserialize($serialized);
}
```

LISTING 13.3 The Serializable interface

Serialization

Serialization and deserialization



Serialization

```
class Artist implements Serializable {
   //...
   // Implement the Serializable interface methods
   public function serialize() {
       // use the built-in PHP serialize function
       return serialize(
              array("earliest" =>self::$earliestDate,
                    "first" => $this->firstName,
                    "last" => $this->lastName,
                    "bdate" => $this->birthDate,
                    "ddate" => $this->deathDate,
                    "bcity" => $this->birthCity,
                    "works" => $this->artworks
                    );
              );
    }
    public function unserialize($data) {
       // use the built-in PHP unserialize function
       $data = unserialize($data);
       self::$earliestDate = $data['earliest'];
       $this->firstName = $data['first'];
       $this->lastName = $data['last'];
       $this->birthDate = $data['bdate'];
       $this->deathDate = $data['ddate'];
       $this->birthCity = $data['bcity'];
       $this->artworks = $data['works'];
    }
    //...
}
```

Serialization

Consider our Artist class

The output of calling serialize($picasso) is:

C:6:"Artist":764:{a:7:{s:8:"earliest";s:13:"Oct 25,
1881";s:5:"firstName";s:5:"Pablo";s:4:"lastName";s:7:"Picasso";s:5:"birthDate";s:13:"Oct
25, 1881";s:5:"deathDate";s:11:"Apl 8, 1973";s:5:"birthCity";s:6:"Malaga";s:5:"works";
a:3:{i:0;C:8:"Painting":134:{a:2:{s:4:"size";a:2:{i:0;d:7.7999999999999998;i:1;d:3.5;}s:7:"a
rtData";s:54:"a:2:{s:4:"date";s:4:"1937";s:4:"name";s:8:"Guernica";}";}}i:1;C:9:"Sculpture"
:186:{a:2:{s:6:"weight";s:8:"162 tons";s:13:"paintingData";
s:133:"a:2:{s:4:"size";a:1:{i:0;d:15.119999999999999;}s:7:"artData";s:53:"a:2:{s:4:"date";s
:4:"1967";s:4:"name";s:7:"Chicago";}";}";}}i:2;C:5:"Movie":175:{a:2:{s:5:"media";s:8:"file.a
vi";s:13:"paintingData";s:113:"a:2:{s:4:"size";a:2:{i:0;i:32;i:1;i:48;}s:7:"artData";s:50:"a:2:{
s:4:"date";s:4:"1968";s:4:"name";s:4:"test";}";}";}}}}}

If the data above is assigned to $data, then the following line will instantiate a new object identical to the original:

$picassoClone = unserialize($data);

Application of Serialization

Remember our state problem

➢Since each request from the user requires objects to be reconstituted, using serialization to store and retrieve objects can be a rapid way to maintain state between requests.

➢At the end of a request you store the state in a serialized form, and then the next request would begin by deserializing it to reestablish the previous state.

13.4 Cookies

There are few things in the world of web development so reviled and misunderstood as the HTTP cookie. Cookies are a client-side approach for persisting state information. They are name=value pairs that are saved within one or more text

13

files that are managed by the browser. These pairs accompany both server requests and responses within the HTTP header. While cookies cannot contain viruses, third-party tracking cookies have been a source of concern for privacy advocates.

Cookies were intended to be a long-term state mechanism. They provide website authors with a mechanism for persisting user-related information that can be stored on the user's computer and be managed by the user's browser.

Cookies are not associated with a specific page but with the page's domain, so the browser and server will exchange cookie information no matter what page the user requests from the site. The browser manages the cookies for the different domains so that one domain's cookies are not transported to a different domain.

While cookies can be used for any state-related purpose, they are principally used as a way of maintaining continuity over time in a web application. One typical use of cookies in a website is to "remember" the visitor, so that the server can customize the site for the user. Some sites will use cookies as part of their shopping cart implementation so that items added to the cart will remain there even if the user leaves the site and then comes back later. Cookies are also frequently used to keep track of whether a user has logged into a site.

13.4.1 How Do Cookies Work?

While cookie information is stored and retrieved by the browser, the information in a cookie travels within the HTTP header. Figure 13.6 illustrates how cookies work.

There are limitations to the amount of information that can be stored in a cookie (around 4K) and to the number of cookies for a domain (for instance, Internet Explorer 6 limited a domain to 20 cookies).

Like their similarly named chocolate chip brethren beloved by children worldwide, HTTP cookies can also expire. That is, the browser will delete cookies that are beyond their expiry date (which is a configurable property of a cookie). If a cookie does not have an expiry date specified, the browser will delete it when the browser closes (or the next time it accesses the site). For this reason, some commentators will say that there are two types of cookies: session cookies and persistent cookies. A session cookie has no expiry stated and thus will be deleted at the end of the user browsing session. Persistent cookies have an expiry date specified; they will persist in the browser's cookie file until the expiry date occurs, after which they are deleted.

The most important limitation of cookies is that the browser may be configured to refuse them. As a consequence, sites that use cookies should not depend on their availability for critical features. Similarly, the user can also delete cookies or even tamper with the cookies, which may lead to some serious problems if not handled. Several years ago, there was an instructive case of a website selling stereos and televisions that used a cookie-based shopping cart. The site placed not only the product

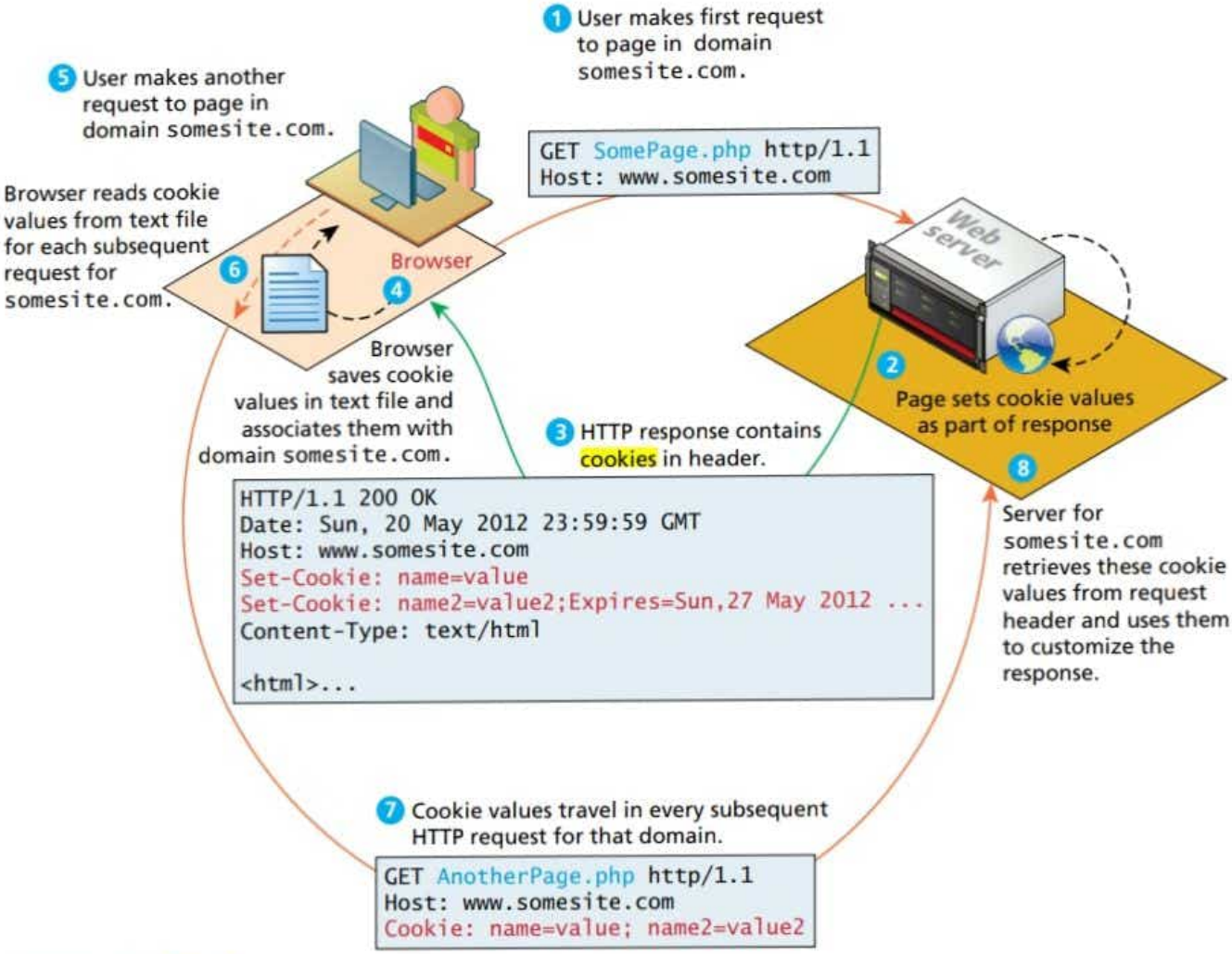


FIGURE 13.6 Cookies at work

identifier but also the product price in the cart. Unfortunately, the site then used the price in the cookie in the checkout. Several curious shoppers edited the price in the cookie stored on their computers, and then purchased some big-screen televisions for only a few cents!

NOTE

Remember that a user's browser may refuse to save cookies. Ideally your site should still work even in such a case.

13.4.2 Using Cookies

HANDS-ON EXERCISES
LAB 13 EXERCISE
Using Cookies

Like any other web development technology, PHP provides mechanisms for writing and reading cookies. Cookies in PHP are *created* using the `setcookie()` function and are *retrieved* using the `$_COOKIES` superglobal associative array, which works like the other superglobals covered in Chapter 9.

Listing 13.1 illustrates the writing of a persistent cookie in PHP. **It is important to note that cookies must be written before any other page output.**

```php
<?php
   // add 1 day to the current time for expiry time
   $expiryTime = time()+60*60*24;

   // create a persistent cookie
   $name = "Username";
   $value = "Ricardo";
   setcookie($name, $value, $expiryTime);
?>
```

LISTING 13.1 Writing a cookie

The `setcookie()` function also supports several more parameters, which further customize the new cookie. You can examine the online official PHP documentation for more information.[1]

Listing 13.2 illustrates the reading of cookie values. Notice that when we read a cookie, we must also check to ensure that the cookie exists. In PHP, if the cookie has expired (or never existed in the first place), then the client's browser would not send anything, and so the `$_COOKIE` array would be blank.

PRO TIP

Almost all browsers now support the HttpOnly cookie. This is a cookie that has the `HttpOnly` flag set in the HTTP header. Using this flag can mitigate some of the security risks with cookies (e.g., cross-site scripting or XSS). This flag instructs the browser to not make this cookie available to JavaScript. In PHP, you can set the cookie's `HttpOnly` property to `true` when setting the cookie:

```php
setcookie($name, $value, $expiry, null, null, null, true);
```

13.4.3 Persistent Cookie Best Practices

So what kinds of things should a site store in a persistent cookie? Due to the limitations of cookies (both in terms of size and reliability), your site's correct operation


```php
<?php
   if( !isset($_COOKIE['Username']) ) {
      //no valid cookie found
   }
   else {
      echo "The username retrieved from the cookie is:";
      echo $_COOKIE['Username'];
   }
?>
```

LISTING 13.2 Reading a cookie

should not be dependent upon cookies. Nonetheless, the user's experience might be improved with the judicious use of cookies.

Many sites provide a "Remember Me" checkbox on login forms, which relies on the use of a persistent cookie. This login cookie would contain the user's username but not the password. Instead, the login cookie would contain a random token; this random token would be stored along with the username in the site's back-end database. Every time the user logs in, a new token would be generated and stored in the database and cookie.

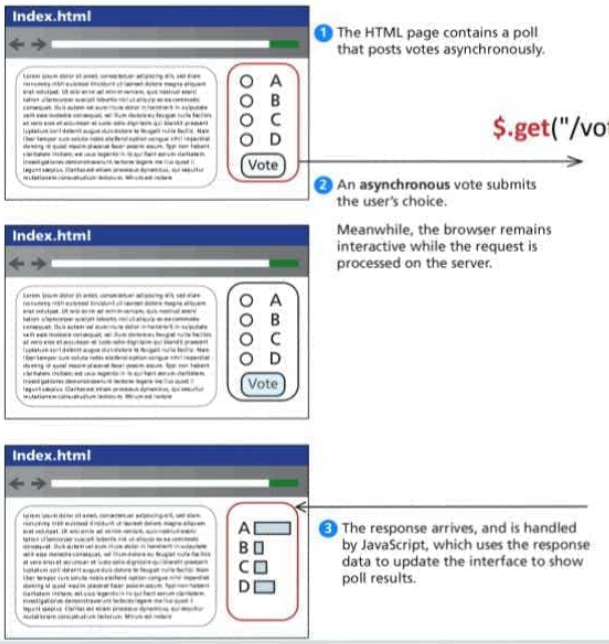
Another common, nonessential use of cookies would be to use them to store user preferences. For instance, some sites allow the user to choose their preferred site color scheme or their country of origin or site language. In these cases, saving the user's preferences in a cookie will make for a more contented user, but if the user's browser does not accept cookies, then the site will still work just fine; at worst the user will simply have to reselect his or her preferences again.

Another common use of cookies is to track a user's browsing behavior on a site. Some sites will store a pointer to the last requested page in a cookie; this information can be used by the site administrator as an analytic tool to help understand how users navigate through the site.

AJAX

GET Requests

15



AJAX

GET Requests – formal definition

jQuery.get (**url [, data] [, success(data, textStatus, jqXHR)] [, dataType]**)

- **url** is a string that holds the location to send the request.
- **data** is an optional parameter that is a query string or a *Plain Object*.
- **success(data,textStatus,jqXHR)** is an optional *callback* function that executes when the response is received.
  - **data** holding the body of the response as a string.
  - **textStatus** holding the status of the request (i.e., "success").
  - **jqXHR** holding a jqXHR object, described shortly.
- **dataType** is an optional parameter to hold the type of data expected from the server.

AJAX

GET Requests – an example

```
$.get("/vote.php?option=C", function(data,textStatus,jsXHR) {
  if (textStatus=="success") {
     console.log("success! response is:" + data);
  }
  else {
     console.log("There was an error code"+jsXHR.status);
  }
  console.log("all done");
});
```

LISTING 15.13 jQuery to asynchronously get a URL and outputs when the response arrives

POST requests

Via jQuery AJAX

➢POST requests are often preferred to GET requests because one can post an unlimited amount of data, and because they do not generate viewable URLs for each action.

➢GET requests are typically not used when we have forms because of the messy URLs and that limitation on how much data we can transmit.

➢With POST it is possible to transmit files, something which is not possible with GET.

POST requests

Via jQuery AJAX

•The HTTP 1.1 definition describes GET as a **safe method** meaning that they should not change anything, and should only read data.

•POSTs on the other hand are not safe, and should be used whenever we are changing the state of our system (like casting a vote). get() method.

•POST syntax is almost identical to GET.

•jQuery.post (url [, data] [, success(data, textStatus, jqXHR)] [, dataType])

POST requests

Via jQuery AJAX

If we were to convert our vote casting code it would simply change the first line from

```
var jqxhr = $.get("/vote.php?option=C");
```

to

```
var jqxhr = $.post("/vote.php", "option=C");
```

POST requests

Via jQuery AJAX

If we were to convert our vote casting code it would simply change the first line from

```
var jqxhr = $.get("/vote.php?option=C");
```

to

```
var jqxhr = $.post("/vote.php", "option=C");
```

POST requests

Serialize() will seriously help

serialize() can be called on any form object to return its current key-value pairing as an & separated string, suitable for use with post().

```
var postData = $("#voteForm").serialize();

$.post("vote.php", postData);
```

16

13.1 The Problem of State in Web Applications

Much of the programming in the previous several chapters has analogies to most typical nonweb application programming. Almost all applications need to process user inputs, output information, and read and write from databases or other storage media. But in this chapter we will be examining a development problem that is unique to the world of web development: how can one request share information with another request?

At first glance this problem does not seem especially formidable. Single-user desktop applications do not have this challenge at all because the program information for the user is stored in memory (or in external storage) and can thus be easily accessed throughout the application. Yet one must always remember that web applications differ from desktop applications in a fundamental way. Unlike the unified single process that is the typical desktop application, a web application consists of a series of disconnected HTTP requests to a web server where each request for a server page is essentially a request to run a separate program, as shown in Figure 13.1.

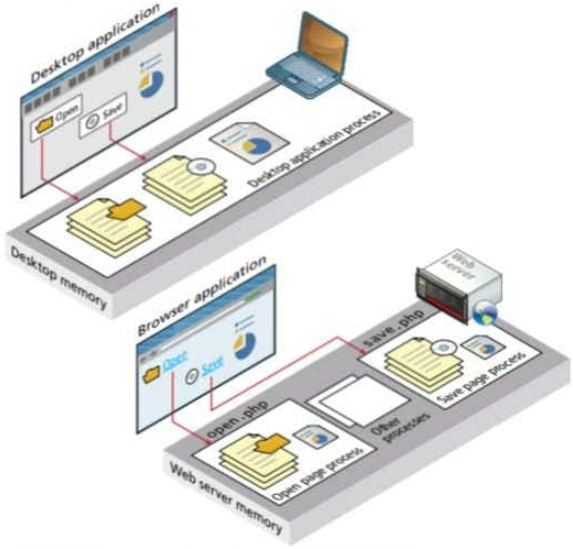
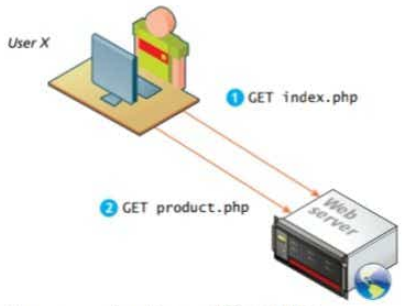


FIGURE 13.1 Desktop applications versus web applications

Furthermore, the web server sees only requests. The HTTP protocol does not, without programming intervention, distinguish two requests by one source from two requests from two different sources, as shown in Figure 13.2.



... is for the server not really any different than ...

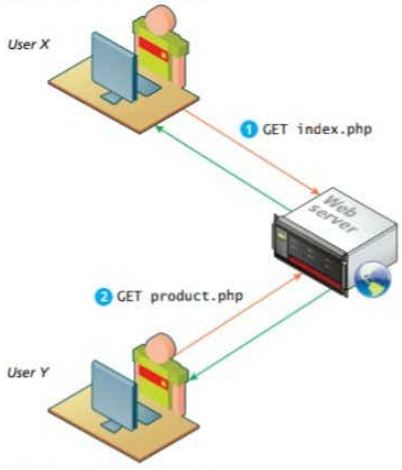


FIGURE 13.2 What the web server sees

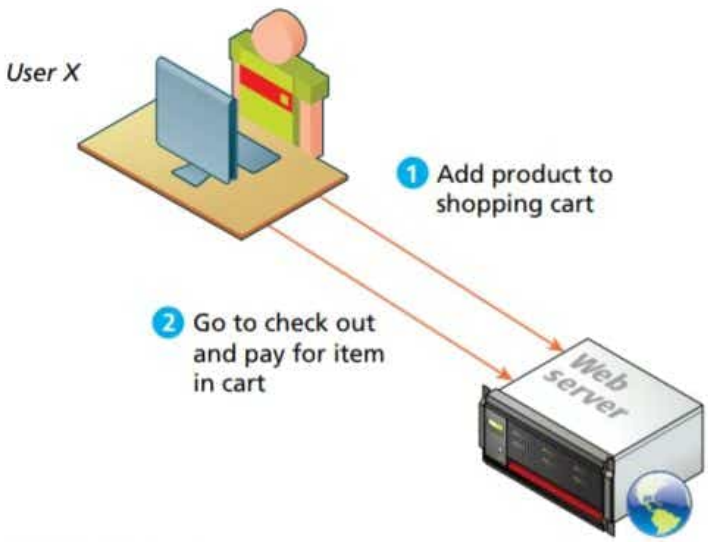


FIGURE 13.3 What the user wants the server to see

While the HTTP protocol disconnects the user's identity from his or her requests, there are many occasions when we want the web server to connect requests together. Consider the scenario of a web shopping cart, as shown in Figure 13.3. In such a case, the user (and the website owner) most certainly wants the server to recognize that the request to add an item to the cart and the subsequent request to check out and pay for the item in the cart are connected to the same individual.

The rest of this chapter will explain how web programmers and web development environments work together through the constraints of HTTP to solve this particular problem. As we will see, there is no single "perfect" solution, but a variety of different ones each with their own unique strengths and weaknesses.

The starting point will be to examine the somewhat simpler problem of how does one web page pass information to another page? That is, what mechanisms are available within HTTP to pass information to the server in our requests? As we have already seen in Chapters 1, 4, and 9, what we can do to pass information is constrained by the basic request-response interaction of the HTTP protocol. In HTTP, we can pass information using:

- Query strings
- Cookies

17

❑XML is a markup language, but unlike HTML, XML can be used to mark up any type of data.

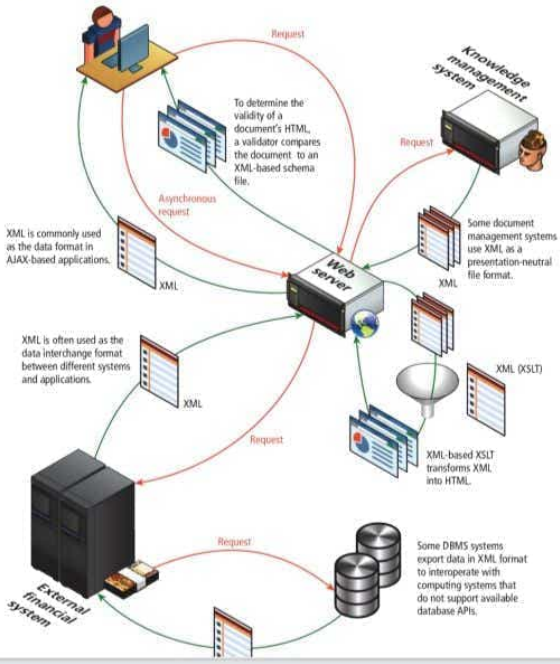
❑Benefits of XML data is that as plain text, it can be read and transferred between applications and different operating systems as well as being human-readable.

❑XML is used on the web server to communicate asynchronously with the browser

❑Used as a data interchange format for moving information between systems

XML Overview

Used in many systems



Well Formed XML

For a document to be **well-formed XML**, it must follow the syntax rules for XML:

- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with a number.
- There must be a single-root element. A **root element** is one that contains all the other elements; for instance, in an HTML document, the root element is <html>.
- All elements must have a closing element (or be self-closing).
- Elements must be properly nested.
- Elements can contain attributes.
- Attribute values must always be within quotes.
- Element and attribute names are case sensitive.

XPath

Another XML Technology

❖**XPath** is a standardized syntax for searching an XML document and for navigating to elements within the XML document

❖XPath is typically used as part of the programmatic manipulation of an XML document in PHP and other languages

❖XPath uses a syntax that is similar to the one used in most operating systems to access directories.

XPath

Learn through example

/art/painting[@id='192']/artist/name

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
  <painting id="192">
    <title>The Kiss</title>
    <artist>
      <name>Klimt</name>
      <nationality>Austria</nationality>
    </artist>
    <year>1907</year>
    <medium>Oil and gold on canvas</medium>
  </painting>
  <painting id="139">
    <title>The Oath of the Horatii</title>
    <artist>
      <name>David</name>
      <nationality>France</nationality>
    </artist>
    <year>1784</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

/art/painting[year > 1800]

/art/painting[3]/@id

This used when you want to look for particular data, like just the artist's name for a particular painting

18

jQuery Selectors

Should ring a bell

➢When discussing basic JavaScript we introduced the **getElementByID()** and **querySelector()** selector functions in JavaScript.

➢Although the advanced **querySelector()** methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers

➢jQuery introduces its own way to select an element, which under the hood supports a myriad of older browsers for you!

jQuery Selectors

The easiest way to select an element yet

✓The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework,

jQuery()

✓Is reserved for selecting elements from the DOM.

✓Because it is used so frequently, it has a shortcut notation and can be written as

$()

15.2.2 jQuery Selectors

Selectors were first covered in Chapter 6, when we introduced the `getElementByID()` and `querySelector()` functions in JavaScript (they were also covered back in Chapter 3, when CSS was introduced). Selectors offer the developer a way of accessing and modifying a DOM object from an HTML page in a simple way. Although the advanced `querySelector()` methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers. To address this issue jQuery introduces its own way to select an element, which under the hood supports a myriad of older browsers for you! jQuery builds on the CSS selectors and adds its own to let you access elements as you would in CSS or using new shortcut methods.

The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework, `jQuery()`, lets programmers easily access DOM objects using selectors passed as parameters. Because it is used so frequently, it has a shortcut notation and can be written as `$()`. This `$()` syntax can be confusing to PHP developers at first, since in PHP the `$` symbol indicates a variable. Nonetheless jQuery uses this shorthand frequently, and we will use this shorthand notation throughout this book.

You can combine CSS selectors with the `$()` notation to select DOM objects that match CSS attributes. Pass in the string of a CSS selector to `$()` and the result will be the set of DOM objects matching the selector. You can use the basic selector syntax from CSS, as well as some additional ones defined within jQuery.

The selectors always return arrays of results, rather than a single object. This is easy to miss since we can apply operations to the set of DOM objects matched by the selector. For instance, sometimes in the examples you will see the *0*th element referenced using the familiar [0] syntax. This will access the first DOM object that matches the selector, which we can then drill down into to access other attributes and properties.

Basic Selectors

The four basic selectors were defined back in Chapter 3, and include the universal selector, class selectors, id selectors, and elements selectors. To review:

- `$("*")` **Universal selector** matches all elements (and is slow).
- `$("tag")` **Element selector** matches all elements with the given element name.
- `$(".class")` **Class selector** matches all elements with the given CSS class.
- `$("#id")` **Id selector** matches all elements with a given HTML id attribute.

For example, to select the single `<div>` element with `id="grab"` you would write:

```
var singleElement = $("#grab");
```

HANDS-O
EXERCISE
LAB 15 EX
Basic Select

To get a set of all the <a> elements the selector would be:

```
var allAs = $("a");
```

These selectors are powerful enough that they can replace the use of `getElementById()` entirely.

The implementation of selectors in jQuery purposefully mirrors the CSS specification, which is especially helpful since CSS is something you have learned and used throughout this book.

In addition to these basic selectors, you can use the other CSS selectors that were covered in Chapter 3: attribute selectors, pseudo-element selectors, and contextual selectors as illustrated in Figure 15.4. The remainder of this section reviews some of these selectors and how they are used with jQuery.

Attribute Selector

An attribute selector provides a way to select elements by either the presence of an element attribute or by the value of an attribute. Chapter 3 mentioned that not all

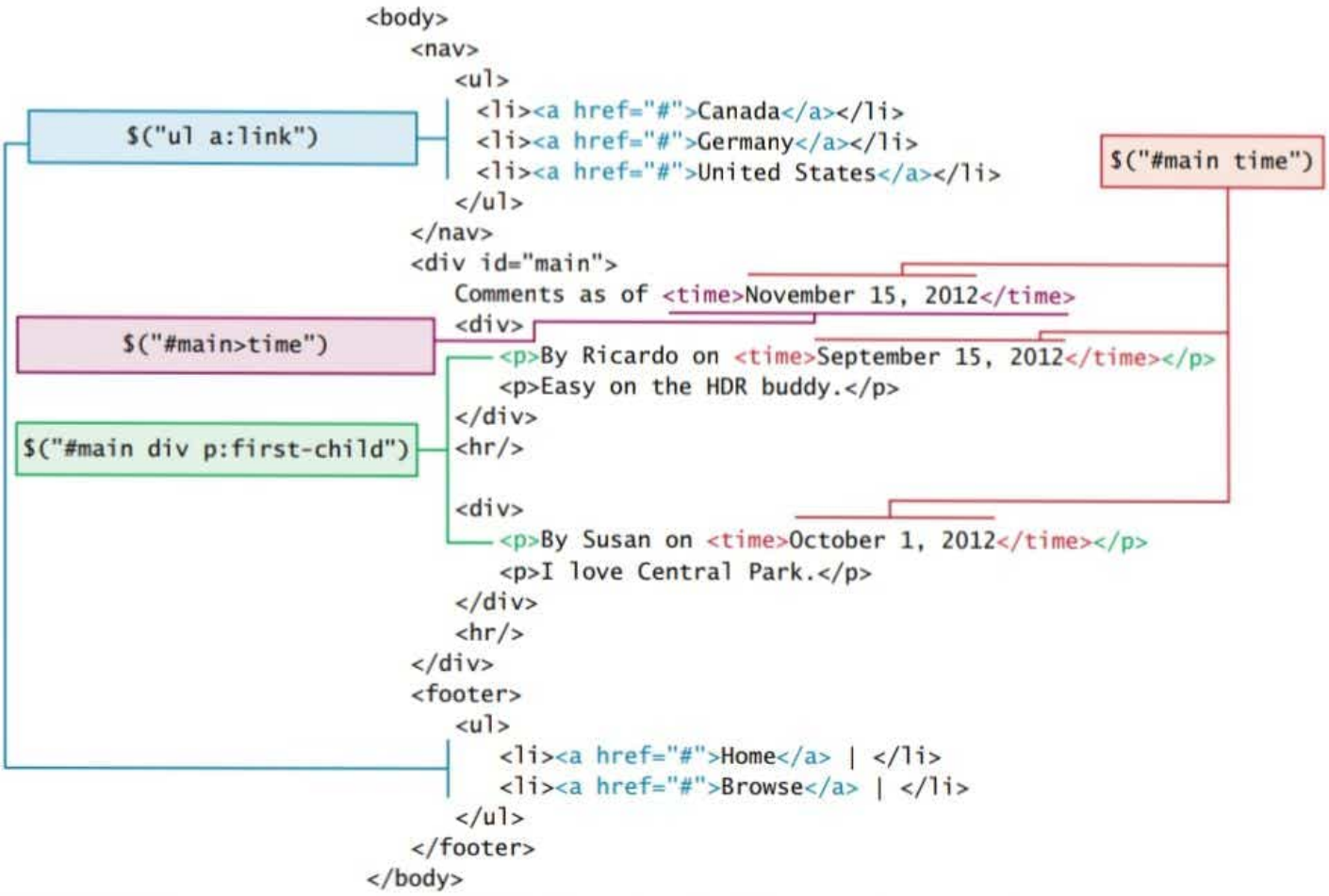


FIGURE 15.4 Illustration of some jQuery selectors and the HTML being selected

19

Section 4 of 8

COOKIES

Cookies

mmmm

➢**Cookies** are a client-side approach for persisting state information.

➢They are name=value pairs that are saved within one or more text files that are managed by the browser.

Cookies

Chocolate and peanut butter

Two kinds of Cookie

- A **session cookie** has no expiry stated and thus will be deleted at the end of the user browsing session.
- **Persistent cookies** have an expiry date specified;

Session State

❖All modern web development environments provide some type of session state mechanism.

❖**Session state** is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session.

❖Session state is ideal for storing more complex objects or data structures that are associated with a user session.

❖In PHP, session state is available to the via the $_SESSION variable

❖Must use session_start() to enable sessions.

Session State

Accessing State

```php
<?php

session_start();

if ( isset($_SESSION['user']) ) {
   // User is logged in
}
else {
   // No one is logged in (guest)
}
?>
```

LISTING 13.5 Accessing session state

Session State

Checking Session existance

```php
<?php
include_once("ShoppingCart.class.php");

session_start();

// always check for existence of session object before accessing it
if ( !isset($_SESSION["Cart"]) ) {
   //session variables can be strings, arrays, or objects, but
   // smaller is better
   $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

LISTING 13.6 Checking session existence

20

13.6.2 Session Storage and Configuration

You may have wondered why session state providers are necessary. In the example shown in Figure 13.8, each user's session information is kept in serialized files, one per session (in ASP.NET, session information is by default not stored in files, but in memory). It is possible to configure many aspects of sessions including where the session files are saved. For a complete listing refer to the session configuration options in php.ini.

The decision to save sessions to files rather than in memory (like ASP.NET) addresses the issue of memory usage that can occur on shared hosts as well as persistence between restarts. Many sites run in commercial hosting environments that are also hosting many other sites. For instance, one of the book author's personal sites (randyconnolly.com, which is hosted by discountasp.net) is, according to a Reverse IP Domain Check, on a server that was hosting 68 other sites when this chapter was being written. Inexpensive web hosts may sometimes stuff hundreds or even thousands of sites on each machine. In such an environment, the server memory that is allotted per web application will be quite limited. And remember that for each application, server memory may be storing not only session information, but pages being executed, and caching information, as shown in Figure 13.10.

On a busy server hosting multiple sites, it is not uncommon for the Apache application process to be restarted on occasion. If the sessions were stored in

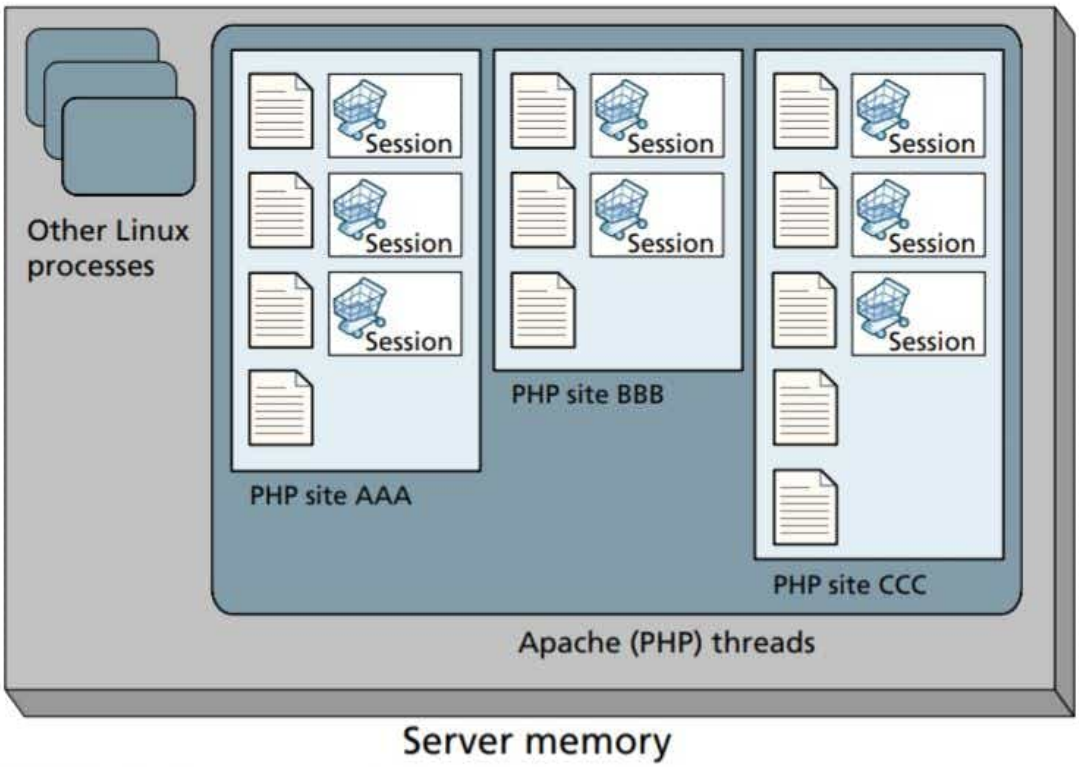


FIGURE 13.10 Applications and server memory

memory, the sessions would all expire, but as they are stored into files, they can be instantly recovered as though nothing happened. This can be an issue in environments where sessions are stored in memory (like ASP.NET), or a custom session handler is involved. One downside to storing the sessions in files is a degradation in performance compared to memory storage, but the advantages, it was decided, outweigh those challenges.

Higher-volume web applications often run in an environment in which multiple web servers (also called a web farm) are servicing requests. Each incoming request is forwarded by a load balancer to any one of the available servers in the farm. In such a situation the in-process session state will not work, since one server may service one request for a particular session, and then a completely different server may service the next request for that session, as shown in Figure 13.11.

There are a number of different ways of managing session state in such a web farm situation, some of which can be purchased from third parties. There are effectively two categories of solution to this problem.

1. Configure the load balancer to be "session aware" and relate all requests using a session to the same server.

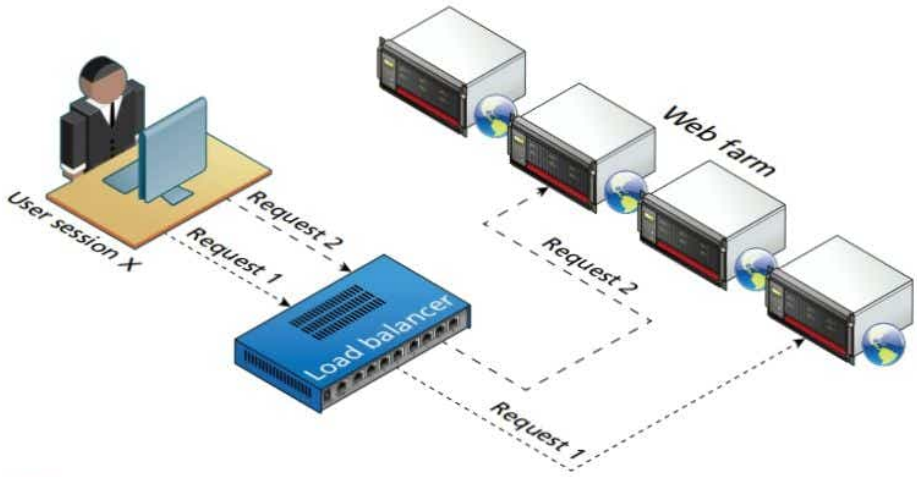


FIGURE 13.11 Web farm

2. Use a shared location to store sessions, either in a database, memcache (covered in the next section), or some other shared session state mechanism as seen in Figure 13.12.

Using a database to store sessions is something that can be done programmatically, but requires a rethinking of how sessions are used. Code that was written to work on a single server will have to be changed to work with sessions in a shared database, and therefore is cumbersome. The other alternative is to configure PHP to

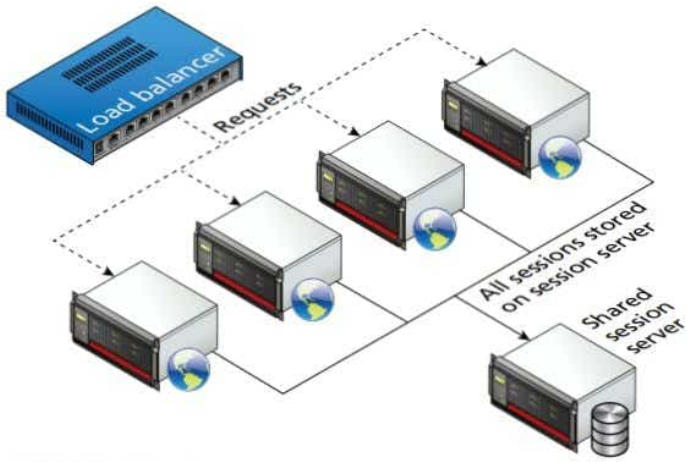


FIGURE 13.12 Shared session provider

use memcache on a shared server (covered in Section 13.8). To do this you must have PHP compiled with memcache enabled; if not, you may need to install the module. Once installed, you must change the php.ini on all servers to utilize a shared location, rather than local files as shown in Listing 13.7.

```
[Session]
; Handler used to store/retrieve data.
session.save_handler = memcache
session.save_path = "tcp://sessionServer:11211"
```

LISTING 13.7 Configuration in php.ini to use a shared location for sessions