FUTURE VISION BIE

One Stop for All Study Materials

& Lab Programs

Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page

Or

Visit : https://hemanthrajhemu.github.io

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

Join Telegram to get Instant Updates: https://bit.ly/VTU_TELEGRAM

Contact: MAIL: futurevisionbie@gmail.com

INSTAGRAM: www.instagram.com/hemanthraj_hemu/

INSTAGRAM: www.instagram.com/futurevisionbie/

WHATSAPP SHARE: https://bit.ly/FVBIESHARE

FUNDAMENTALS of WEB DEVELOPMENT

RANDY CONNOLLY • RICARDO HOAR

https://hemanthrajhemu.github.io

https://hemanthrajhemu.github.io

https://hemanthrajhemu.github.io

https://hemanthrajhemu.github.io

https://hemanthrajhemu.github.io

https://hemanthrajhemu.github.io

Managing State 13

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- Why state is a problem in web application development
- What cookies are and how to use them
- What HTML5 web storage is and how to use it
- What session state is and what are its typical uses and limitations
- What server cache is and why it is important in real-world websites

This chapter examines one of the most important questions in the web development world, namely, how does one page pass information to another page? This question is sometimes also referred to as the problem of state management in web applications. State management is essential to any web application because every web application has information that needs to be preserved from request to request. This chapter begins by examining the problem of state in web applications and the solutions that are available in HTTP. It then examines the state management features that are available in PHP.

https://hemanthrajhemu.github.io

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.

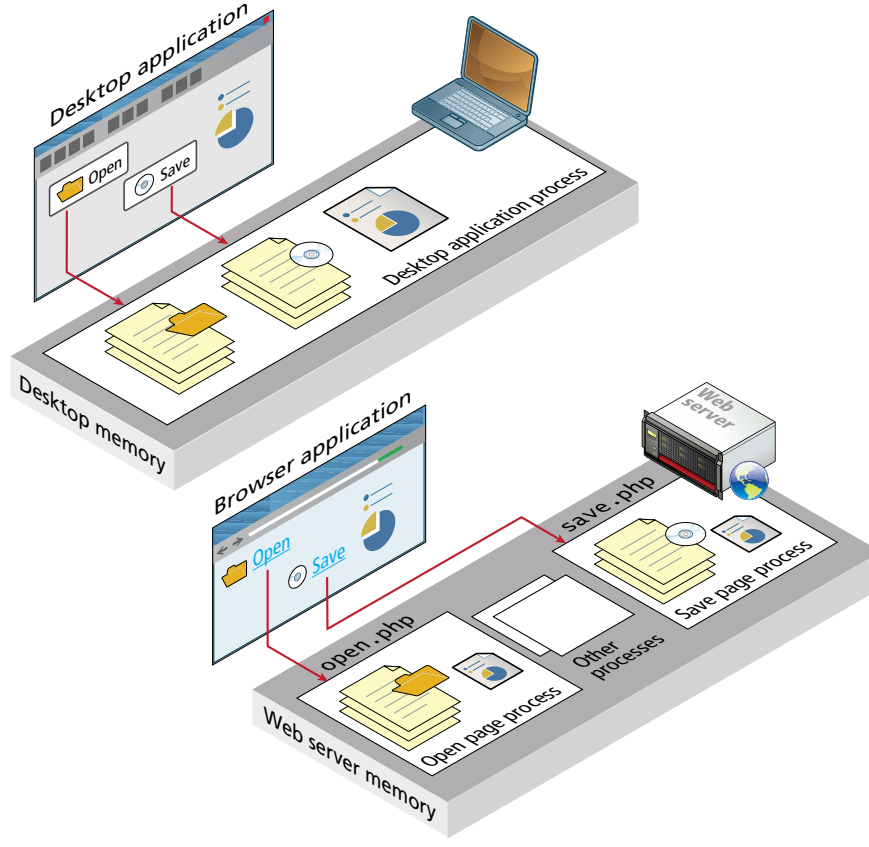


FIGURE 13.1 Desktop applications versus web applications

https://hemanthrajhemu.github.io

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.

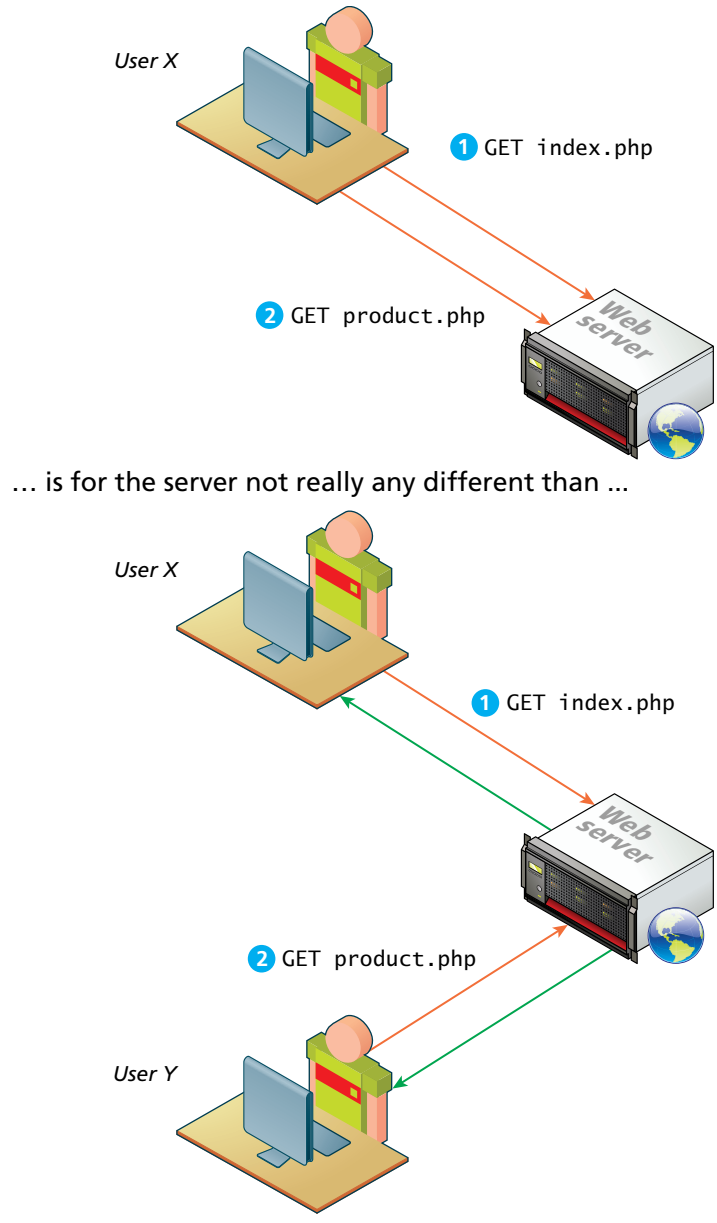


FIGURE 13.2 What the web server sees

https://hemanthrajhemu.github.io

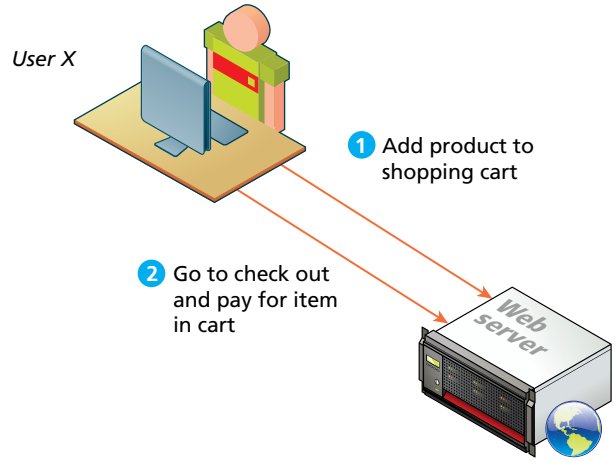


FIGURE 13.3 What the user wants the server to see

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

13.2 Passing Information via Query Strings

As you will recall from earlier chapters, a web page can pass query string information from the browser to the server using one of the two methods: a query string within

https://hemanthrajhemu.github.io

the URL (GET) and a query string within the HTTP header (POST). Figure 13.4 reviews these two different approaches.

NOTE

Remember as well that HTML links and forms using the GET method do the same thing: they make HTTP requests using the GET method.

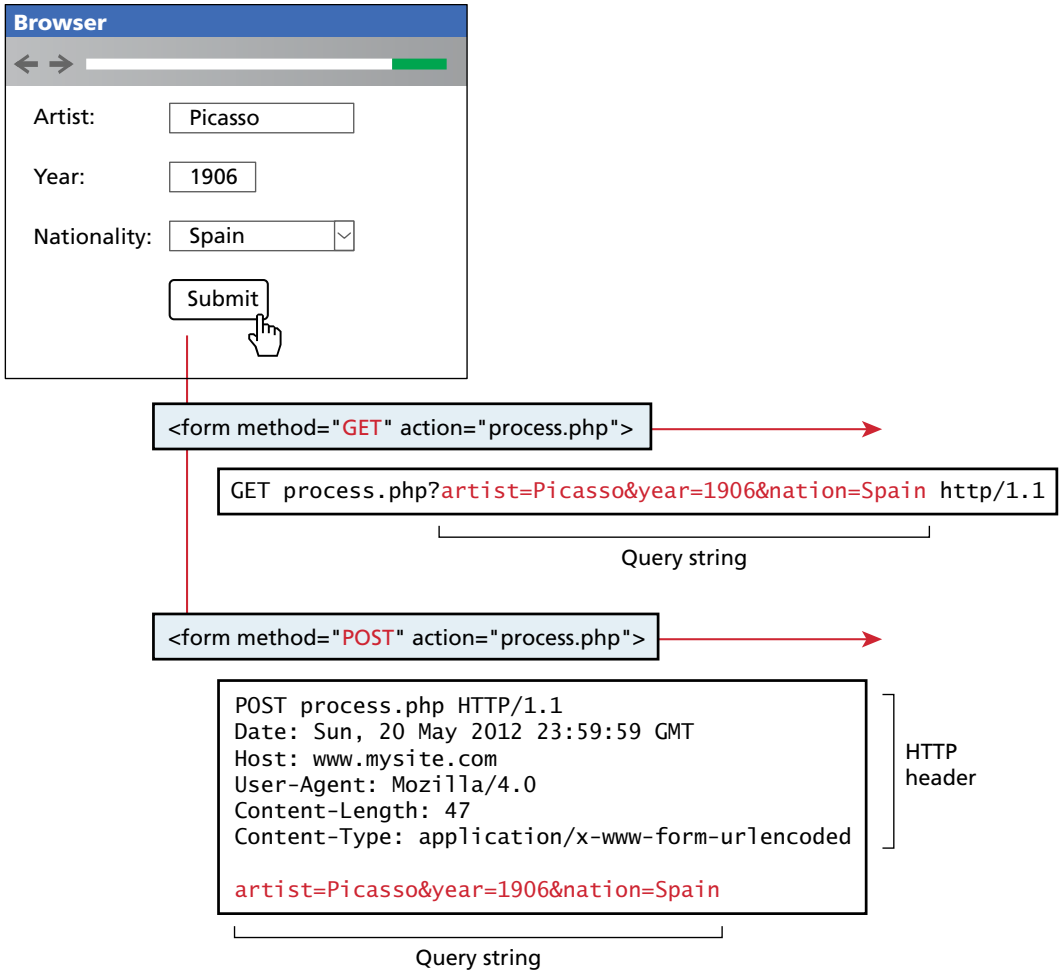


FIGURE 13.4 Recap of GET versus POST

https://hemanthrajhemu.github.io

13.3 Passing Information via the URL Path

While query strings are a vital way to pass information from one page to another, they do have a drawback. The URLs that result can be long and complicated. While for many users this is not that important, many feel that for one particular type of user, query strings are not ideal. Which type of user? Perhaps the single most important user: search engines.

While there is some dispute about whether dynamic URLs (i.e., ones with query string parameters) or static URLs are better from a search engine result optimization (or SEO for search engine optimization) perspective, the consensus is that static URLs do provide some benefits with search engine result rankings. Many factors affect a page's ranking in a search engine, as you will see in Chapter 20, but the appearance of search terms within the URL does seem to improve its relative position. Another benefit to static URLs is that users tend to prefer them.

As we have seen, dynamic URLs (i.e., query string parameters) are a pretty essential part of web application development. How can we do without them? The answer is to rewrite the dynamic URL into a static one (and vice versa). This process is commonly called **URL rewriting**.

For instance, in Figure 13.5, the top four commerce-related results for the search term "reproductions Raphael portrait la donna velata" are shown along with their URLs. Notice how the top three do not use query string parameters but instead put the relevant information within the folder path or the file name.

You might notice as well that the extension for the first three results is **.html**. This doesn't mean that these sites are serving static HTML files (in fact two of them are using PHP); rather the file name extension is also being rewritten to make the URL friendlier.

We can try doing our own rewriting. Let us begin with the following URL with its query string information:

```
www.somedomain.com/DisplayArtist.php?artist=16
```

One typical alternate approach would be to rewrite the URL to:

```
www.somedomain.com/artists/16.php
```

Notice that the query string name and value have been turned into path names. One could improve this to make it more SEO friendly using the following:

```
www.somedomain.com/artists/Mary-Cassatt
```

13.3.1 URL Rewriting in Apache and Linux

Depending on your web development platform, there are different ways to implement URL rewriting. On web servers running Apache, the solution typically involves using the `mod_rewrite` module in Apache along with the `.htaccess` file.

https://hemanthrajhemu.github.io

FIGURE 13.5 URLs within a search engine result page

The `mod_rewrite` module uses a rule-based rewriting engine that utilizes Perl-compatible regular expressions to change the URLs so that the requested URL can be mapped or redirected to another URL internally.

URL rewriting requires knowledge of the Apache web server, so the details of URL rewriting are covered in Section 19.3.12 of Chapter 19, after some more background on Apache has been presented.

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

https://hemanthrajhemu.github.io

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

https://hemanthrajhemu.github.io

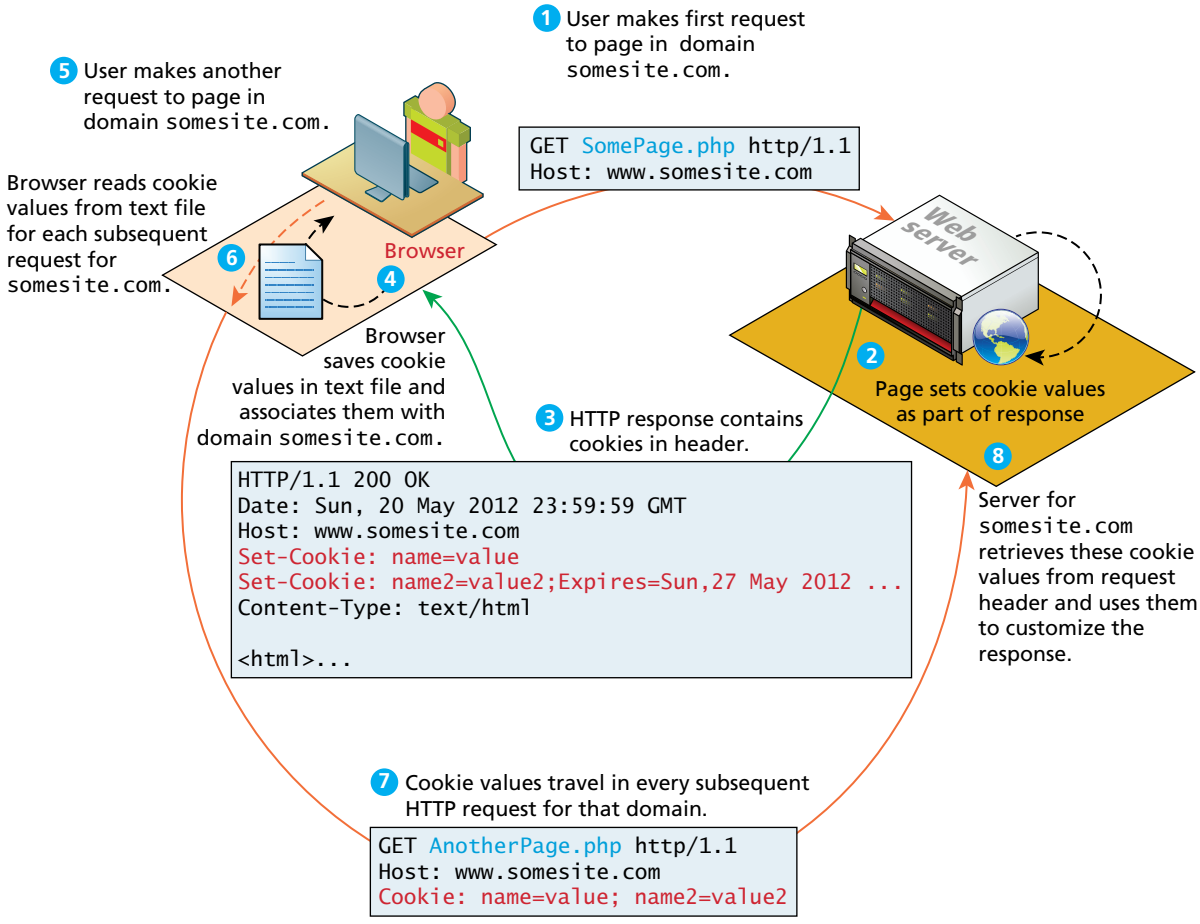


FIGURE 13.6 Cookies at work

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.

https://hemanthrajhemu.github.io

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
   // 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:

```
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

https://hemanthrajhemu.github.io

```
<?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.

PRO TIP

All requests/responses to/from a domain will include all cookies for that domain. This includes not just requests/responses for web pages, but for static components as well, such as image files, CSS files, etc. For a site that makes use of many static components, cookie overhead will increase the network traffic load for the site unnecessarily. For this reason, most large websites that make use of cookies will host those static elements on a completely different domain that does not use cookies. For instance, ebay.com hosts its images on ebaystatic.com and amazon.com hosts its images on images-amazon.com.

https://hemanthrajhemu.github.io

13.5 Serialization

HANDS-ON EXERCISES

LAB 13 EXERCISE
Serialize Your Objects

Serialization is the process of taking a complicated object and reducing it down to zeros and ones for either storage or transmission. Later that sequence of zeros and ones can be reconstituted into the original object as illustrated in Figure 13.7.

In PHP objects can easily be reduced down to a binary string using the `serialize()` function. The resulting string is a binary representation of the object and therefore may contain unprintable characters. The string can be reconstituted back into an object using the `unserialize()` method.[2]

While arrays, strings, and other primitive types will be serializable by default, classes of our own creation must implement the `Serializable` interface shown in Listing 13.3, which requires adding implementations for `serialize()` and `unserialize()` to any class that implements this interface.

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

LISTING 13.3 The Serializable interface

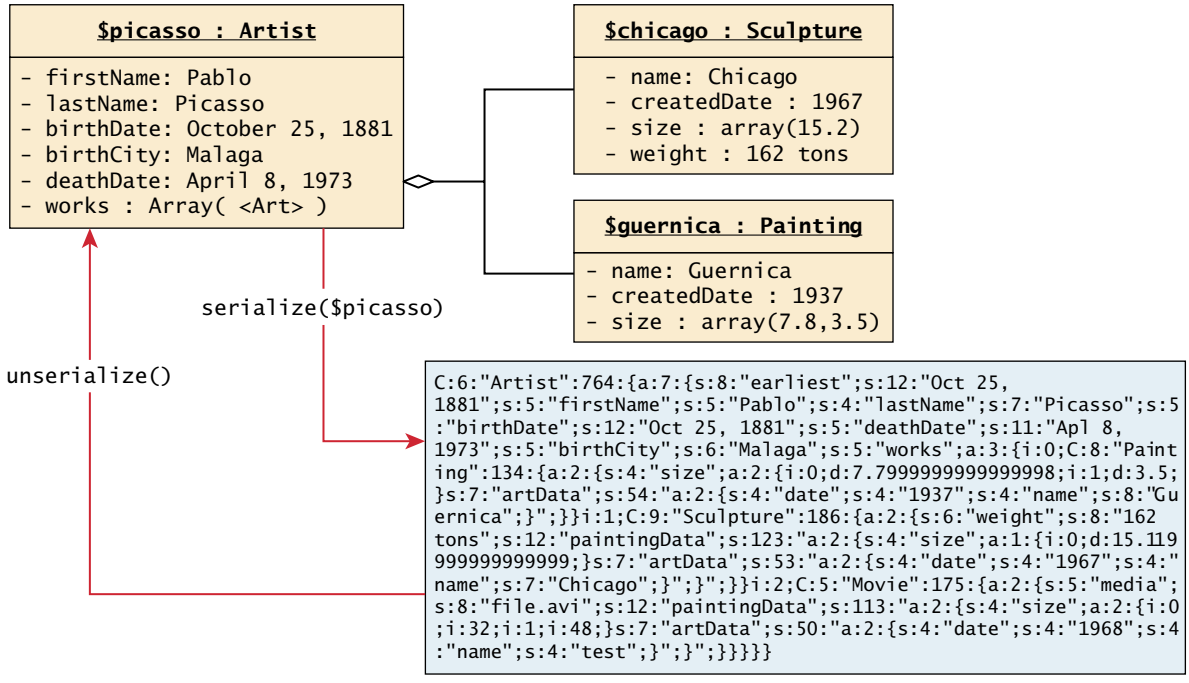


FIGURE 13.7 Serialization and deserialization

https://hemanthrajhemu.github.io

Listing 13.4 shows how the `Artist` class must be modified to implement the `Serializable` interface by adding the `implements` keyword to the class definition and adding implementations for the two methods.

```
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'];
    }
    //...
}
```

LISTING 13.4 Artist class modified to implement the Serializable interface

Note that in order for our `Artist` class to save successfully, the `Art`, `Painting`, and other classes must also implement the `Serializable` interface (not shown here). It should be noted that references to other objects stored at the same time will be preserved while references to objects not serialized in this operation will be lost. This will influence how we use serialization, since if we want to store an object model, we must store all associated objects at once.

The output of calling `serialize($picasso)` is:

```
C:6:"Artist":764:{a:7:{s:8:"earliest";s:13:"Oct 25, 1881";s:5:"first
Name";s:5:"Pablo";s:4:"lastName";s:7:"Picasso";s:5:"birthDate";s:13:
```

https://hemanthrajhemu.github.io

```
"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:"artData";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.avi";s:13:"paintingData";s:1
13:"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";}";}";}}}}}
```

Although nearly unreadable to most people, this data can easily be used to reconstitute the object by passing it to `unserialize()`. If the data above is assigned to `$data`, then the following line will instantiate a new object identical to the original:

```
$picassoClone = unserialize($data);
```

NOTE

Where are serialized objects stored? They are stored in the same directory that the page is executing from.

13.5.1 Application of Serialization

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.

In the next section, you will encounter session state, and will discover that PHP serializes objects for you in its implementation of session state.

13.6 Session State

HANDS-ON EXERCISES

LAB 13 EXERCISE
Using Sessions

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. That is, each browser session has its own session state stored as a serialized file on the server, which is deserialized and loaded into memory as needed for each request, as shown in Figure 13.8.

https://hemanthrajhemu.github.io

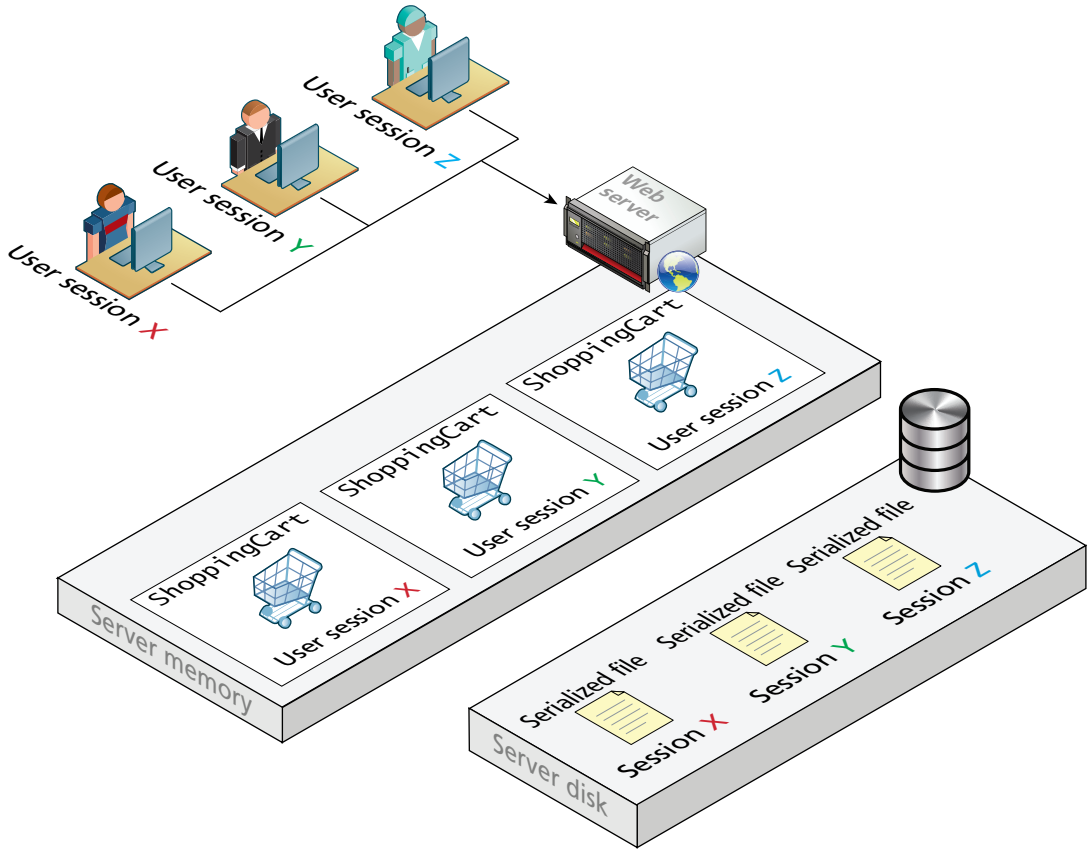


FIGURE 13.8 Session state

Because server storage is a finite resource, objects loaded into memory are released when the request completes, making room for other requests and their session objects. This means there can be more active sessions on disk than in memory at any one time.

Session state is ideal for storing more complex (but not too complex . . . more on that later) objects or data structures that are associated with a user session. The classic example is a shopping cart. While shopping carts could be implemented via cookies or query string parameters, it would be quite complex and cumbersome to do so.

In PHP, session state is available to the developer as a superglobal associative array, much like the `$_GET`, `$_POST`, and `$_COOKIE` arrays.[3] It can be accessed via the `$_SESSION` variable, but unlike the other superglobals, you have to take additional steps in your own code in order to use the `$_SESSION` superglobal.

https://hemanthrajhemu.github.io

To use sessions in a script, you must call the `session_start()` function at the beginning of the script as shown in Listing 13.5. In this example, we differentiate a logged-in user from a guest by checking for the existence of the `$_SESSION['user']` variable.

```
<?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 is typically used for storing information that needs to be preserved across multiple requests by the same user. Since each user session has its own session state collection, it should not be used to store large amounts of information because this will consume very large amounts of server memory as the number of active sessions increase.

As well, since session information does eventually time out, one should always check if an item retrieved from session state still exists before using the retrieved object. If the session object does not yet exist (either because it is the first time the user has requested it or because the session has timed out), one might generate an error, redirect to another page, or create the required object using the lazy initialization approach as shown in Listing 13.6. In this example `ShoppingCart` is a user-defined class. Since PHP sessions are serialized into files, one must ensure that any

```
<?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

https://hemanthrajhemu.github.io

classes stored into sessions can be serialized and deserialized, and that the class definitions are parsed before calling `session_start()`.

13.6.1 How Does Session State Work?

Typically when our students learn about session state, their first reaction is to say "Why didn't we learn this first? This solves all our problems!" Indeed because modern development environments such as ASP.NET and PHP make session state remarkably easy to work with, it is tempting to see session state as a one-stop solution to all web state needs. However, if we take a closer look at how session state works, we will see that session state has the same limitations and issues as the other state mechanisms examined in this chapter.

The first thing to know about session state is that it works within the same HTTP context as any web request. The server needs to be able to identify a given HTTP request with a specific user request. Since HTTP is stateless, some type of user/session identification system is needed. Sessions in PHP (and ASP.NET) are identified with a unique session ID. In PHP, this is a unique 32-byte string that is by default transmitted back and forth between the user and the server via a session cookie (see Section 13.4.1 above), as shown in Figure 13.9.

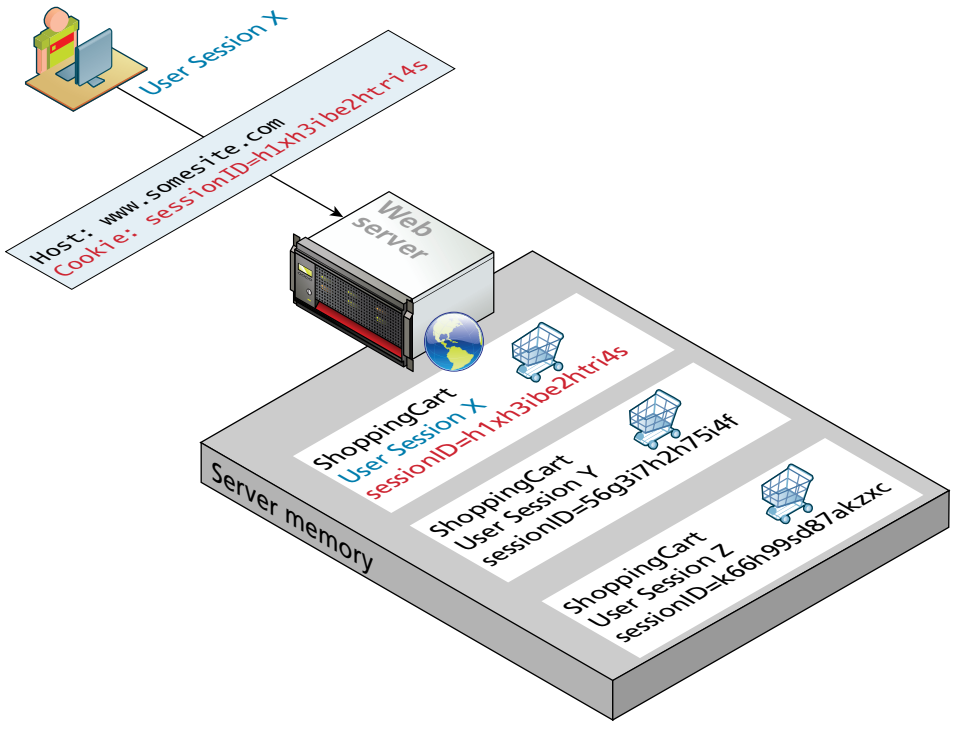


FIGURE 13.9 Session IDs

https://hemanthrajhemu.github.io

As we learned earlier in the section on cookies, users may disable cookie support in their browser; for that reason, PHP can be configured (in the php.ini file) to instead send the session ID within the URL path.

REMEMBER

Session state relies on session IDs that are transmitted via cookies or via embedding in the URL path.

So what happens besides the generating or obtaining of a session ID after a new session is started? For a brand new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session. When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider (discussed in next section). Finally, when a new request is received for an already existing session, the session's dictionary collection is filled with the previously saved session data from the session state provider.

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

https://hemanthrajhemu.github.io

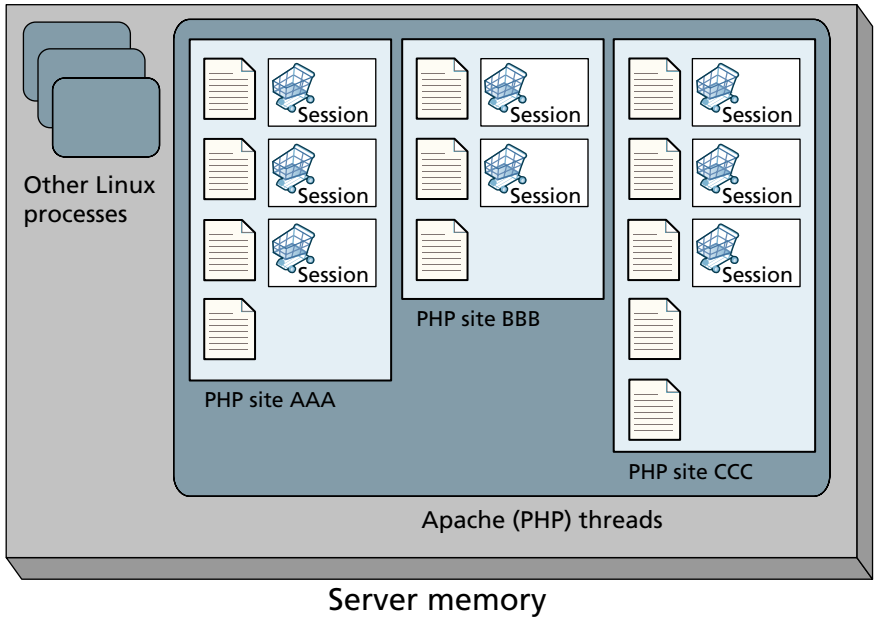


FIGURE 13.10 Applications and server memory

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.

https://hemanthrajhemu.github.io

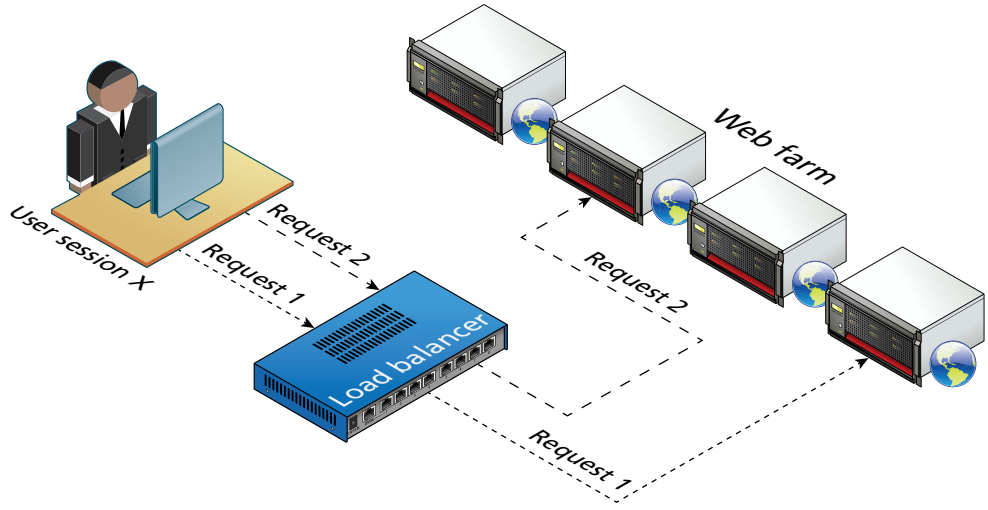


FIGURE 13.11 Web farm

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

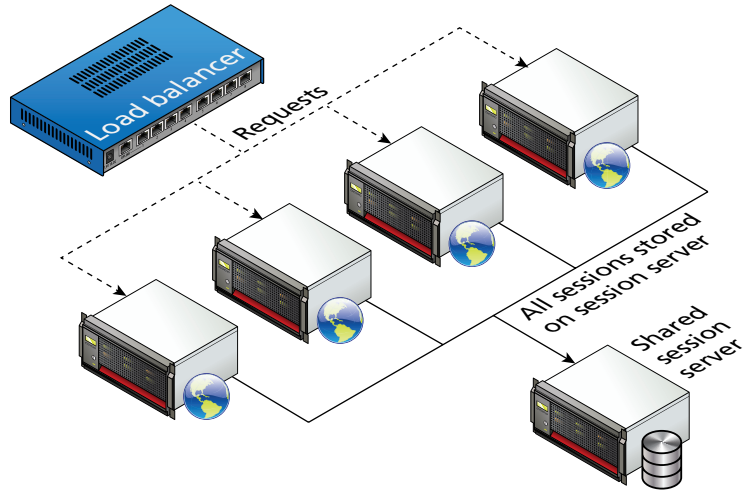


FIGURE 13.12 Shared session provider

https://hemanthrajhemu.github.io

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

13.7 HTML5 Web Storage

Web storage is a new JavaScript-only API introduced in HTML5.[4] It is meant to be a replacement (or perhaps supplement) to cookies, in that web storage is managed by the browser; unlike cookies, web storage data is not transported to and from the server with every request and response. In addition, web storage is not limited to the 4K size barrier of cookies; the W3C recommends a limit of 5MB but browsers are allowed to store more per domain. Currently web storage is supported by current versions of the major browsers, including IE8 and above. However, since JavaScript, like cookies, can be disabled on a user's browser, web storage should not be used for mission-critical application functions.

HANDS-ON EXERCISES

LAB 13 EXERCISE
HTML5 Web Storage

Just as there were two types of cookies, there are two types of global web storage objects: `localStorage` and `sessionStorage`. The `localStorage` object is for saving information that will persist between browser sessions. The `sessionStorage` object is for information that will be lost once the browser session is finished.

These two objects are essentially key-value collections with the same interface (i.e., the same JavaScript properties and functions).

13.7.1 Using Web Storage

Listing 13.8 illustrates the JavaScript code for writing information to web storage. Do note that it is *not* PHP code that interacts with the web storage mechanism but JavaScript. As demonstrated in the listing, there are two ways to store values in web storage: using the `setItem()` function, or using the property shortcut (e.g., `sessionStorage.FavoriteArtist`).

Listing 13.9 demonstrates that the process of reading from web storage is equally straightforward. The difference between `sessionStorage` and `localStorage` in this example is that if you close the browser after writing and then run the code in Listing 13.8, only the `localStorage` item will still contain a value.

https://hemanthrajhemu.github.io

```
<form ... >
   <h1>Web Storage Writer</h1>
   <script language="javascript" type="text/javascript">

        if (typeof (localStorage) === "undefined" ||
               typeof (sessionStorage) === "undefined") {
            alert("Web Storage is not supported on this browser...");
        }
        else {
            sessionStorage.setItem("TodaysDate", new Date());
            sessionStorage.FavoriteArtist = "Matisse";

            localStorage.UserName = "Ricardo";
            document.write("web storage modified");
        }
   </script>
   <p><a href="WebStorageReader.php">Go to web storage reader</a></p>
</form>
```

LISTING 13.8 Writing web storage

```
<form id="form1" runat="server">
   <h1>Web Storage Reader</h1>
   <script language="javascript" type="text/javascript">

        if (typeof (localStorage) === "undefined" ||
               typeof (sessionStorage) === "undefined") {
            alert("Web Storage is not supported on this browser...");
        }
        else {
            var today = sessionStorage.getItem("TodaysDate");
            var artist = sessionStorage.FavoriteArtist;

            var user = localStorage.UserName;
            document.write("date saved=" + today);
            document.write("<br/>favorite artist=" + artist);
            document.write("<br/>user name = " + user);
        }
   </script>
</form>
```

LISTING 13.9 Reading web storage

https://hemanthrajhemu.github.io

13.7.2 Why Would We Use Web Storage?

Looking at the two previous listings you might wonder why we would want to use web storage. Cookies have the disadvantage of being limited in size, potentially disabled by the user, vulnerable to XSS and other security attacks, and being sent in every single request and response to and from a given domain. On the other hand, the fact that cookies are sent with every request and response is also their main advantage: namely, that it is easy to implement data sharing between the client browser and the server. Unfortunately with web storage, transporting the information within web storage back to the server is a relatively complicated affair involving the construction of a web service on the server (see Chapter 17) and then using asynchronous communication via JavaScript to push the information to the server.

A better way to think about web storage is not as a cookie replacement but as a local cache for relatively static items available to JavaScript. One practical use of web storage is to store static content downloaded asynchronously such as XML or JSON from a web service in web storage, thus reducing server load for subsequent requests by the session.

Figure 13.13 illustrates an example of how web storage could be used as a mechanism for reducing server data requests, thereby speeding up the display of the page on the browser, as well as reducing load on the server.

13.8 Caching

Caching is a vital way to improve the performance of web applications. Your browser uses caching to speed up the user experience by using locally stored versions of images and other files rather than re-requesting the files from the server. While important, from a server-side perspective, a server-side developer only has limited control over browser caching (see Pro Tip).

PRO TIP

In the HTTP protocol there are headers defined that relate exclusively to caching. These include the `Expires`, `Cache-Control`, and `Last-Modified` headers. In PHP one can set any HTTP header explicitly using the `header()` function, but to ensure consistency, additional functions have been provided, which manage headers related to caching.

The function `session_cache_limiter()` allows you to set the cache. The function `session_cache_expire()` provides control over the default expiry time (180 seconds by default). By using these two functions one can determine how and when the browser caches pages locally.

https://hemanthrajhemu.github.io

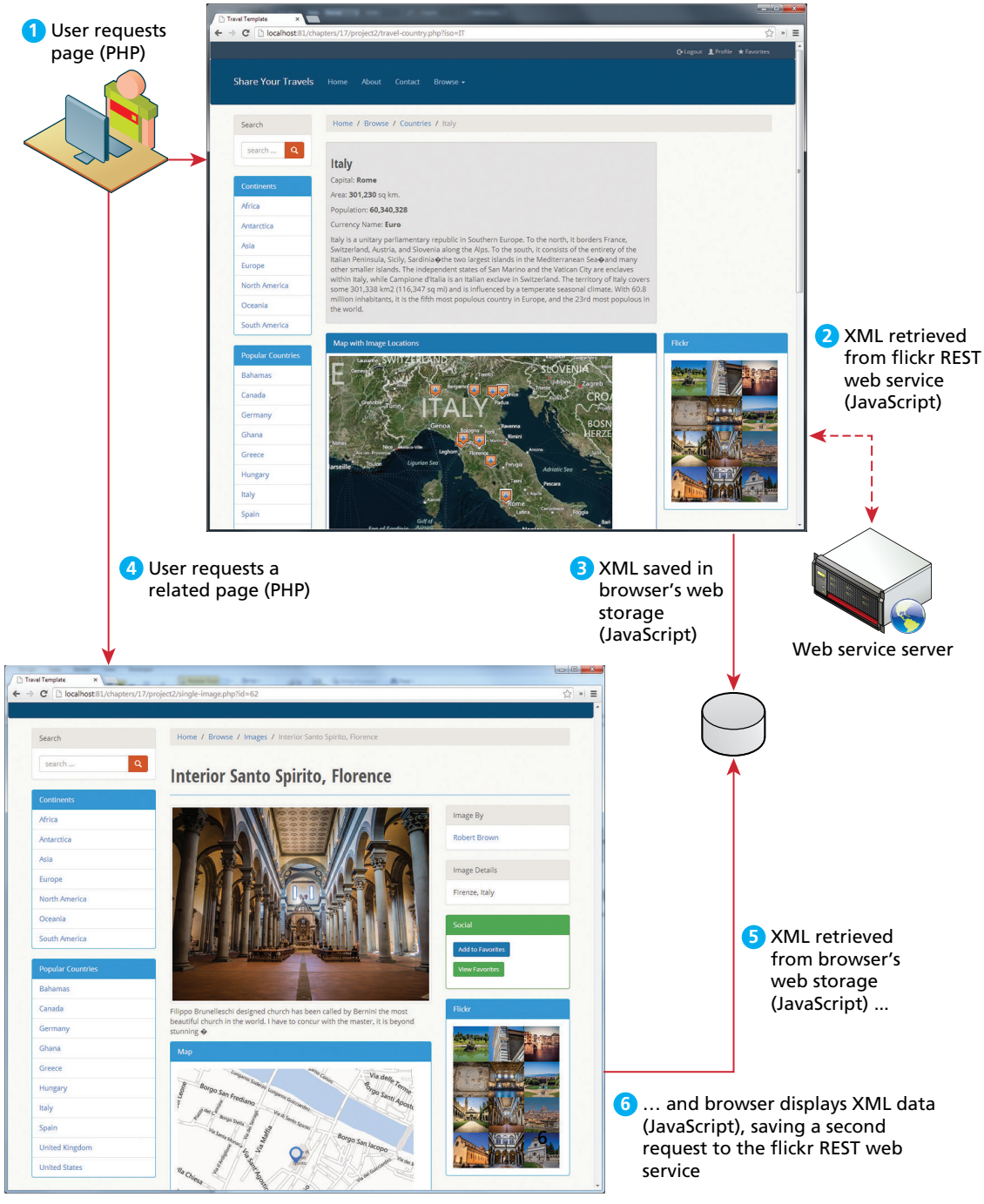


FIGURE 13.13 Using web storage

https://hemanthrajhemu.github.io

There is a way, however, to integrate caching on the server side. Why is this necessary? Remember that every time a PHP page is requested, it must be fetched, parsed, and executed by the PHP engine, and the end result is HTML that is sent back to the requestor. For the typical PHP page, this might also involve numerous database queries and processing to build. If this page is being served thousands of times per second, the dynamic generation of that page may become unsustainable.

One way to address this problem is to cache the generated markup in server memory so that subsequent requests can be served from memory rather than from the execution of the page.

There are two basic strategies to caching web applications. The first is page output caching, which saves the rendered output of a page or user control and reuses the output instead of reprocessing the page when a user requests the page again. The second is application data caching, which allows the developer to programmatically cache data.

13.8.1 Page Output Caching

In this type of caching, the contents of the rendered PHP page (or just parts of it) are written to disk for fast retrieval. This can be particularly helpful because it allows PHP to send a page response to a client without going through the entire page processing life cycle again (see Figure 13.14). Page output caching is especially useful for pages whose content does not change frequently but which require significant processing to create.

HANDS-ON EXERCISES

LAB 13 EXERCISE
Cache a Page

There are two models for page caching: full page caching and partial page caching. In full page caching, the entire contents of a page are cached. In partial page caching, only specific parts of a page are cached while the other parts are dynamically generated in the normal manner.

Page caching is not included in PHP by default, which has allowed a marketplace for free and commercial third-party cache add-ons such as Alternative PHP Cache (open source) and Zend (commercial) to flourish. However, one can easily create basic caching functionality simply by making use of the output buffering and time functions. The mod_cache module that comes with the Apache web server engine is the most common way websites implement page caching. This separates server tuning from your application code, simplifying development, and leaving cache control up to the web server rather than the application developer. The details of configuring that Apache cache are described in Section 19.4.6 of Chapter 19.

It should be stressed that it makes no sense to apply page output caching to every page in a site. However, performance improvements can be gained (i.e., reducing server loads) by caching the page output of especially busy pages in which the content is the same for all users.

13.8.2 Application Data Caching

One of the biggest drawbacks with page output caching is that performance gains will only be had if the entire cached page is the same for numerous requests.

https://hemanthrajhemu.github.io

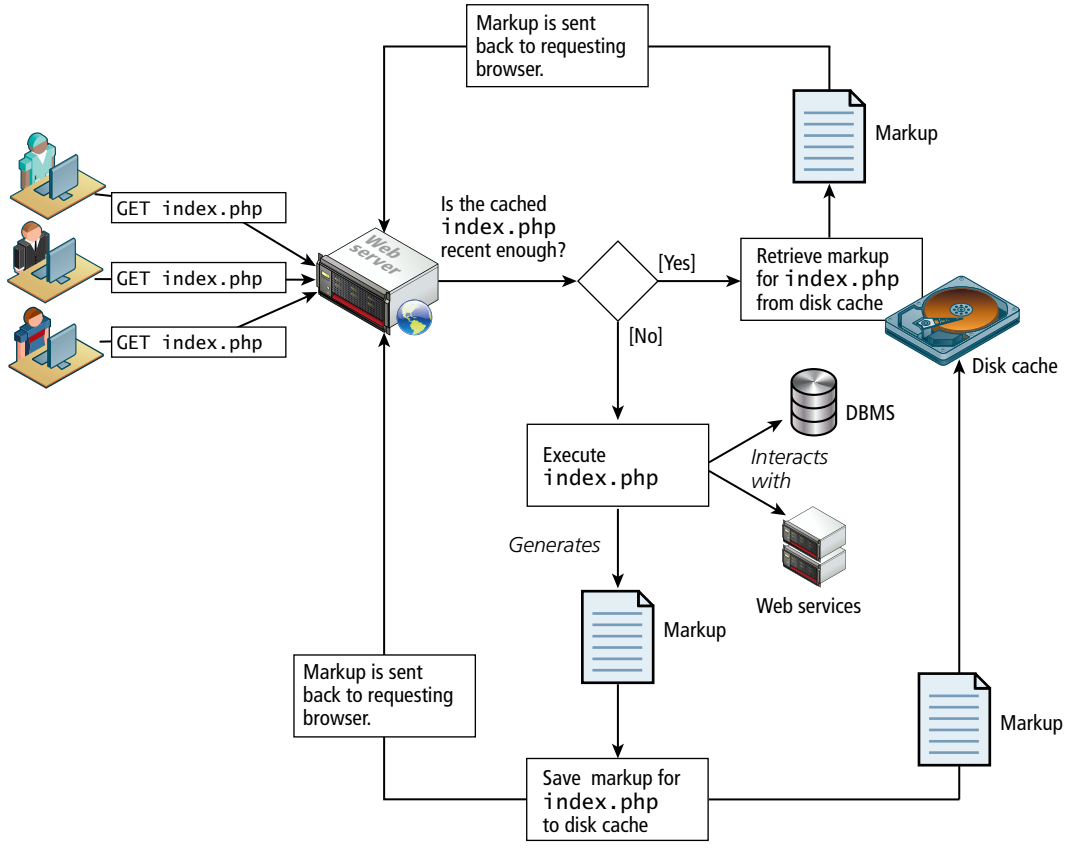


FIGURE 13.14 Page output caching

However, many sites customize the content on each page for each user, so full or partial page caching may not always be possible.

An alternate strategy is to use application data caching in which a page will programmatically place commonly used collections of data that require time-intensive queries from the database or web server into cache memory, and then other pages that also need that same data can use the cache version rather than re-retrieve it from its original location.

While the default installation of PHP does not come with an application caching ability, a widely available free PECL extension called memcache is widely used to provide this ability.[5] Listing 13.10 illustrates a typical use of memcache.

It should be stressed that memcache should not be used to store large collections. The size of the memory cache is limited, and if too many things are placed in it, its performance advantages will be lost as items get paged in and out. Instead, it should be used for relatively small collections of data that are frequently accessed on multiple pages.

https://hemanthrajhemu.github.io

```
<?php

// create connection to memory cache
$memcache = new Memcache;
$memcache->connect('localhost', 11211)
   or die ("Could not connect to memcache server");

$cacheKey = 'topCountries';
/* If cached data exists retrieve it, otherwise generate and cache
   it for next time */
   if ( ! isset($countries = $memcache->get($cacheKey)) ) {

   // since every page displays list of top countries as links
   // we will cache the collection

   // first get collection from database
   $cgate = new CountryTableGateway($dbAdapter);
   $countries = cgate->getMostPopular();

   // now store data in the cache (data will expire in 240 seconds)
   $memcache->set($cacheKey, $countries, false, 240)
       or die ("Failed to save cache data at the server");
}
// now use the country collection
displayCountryList($countries);

?>
```

LISTING 13.10 Using memcache

13.9 Chapter Summary

Most websites larger than a few pages will eventually require some manner of persisting information on one page (generally referred to as "state"), so that it is available to other pages in the site. This chapter examined the options for managing state using what is available to us in HTTP (query strings, the URL, and cookies), as well as those for managing state on the server (session state). The chapter finished with caching, an important technique for optimizing real-world web applications.

13.9.1 Key Terms

application data caching
cache
cookies
HttpOnly cookie
page output caching
persistent cookies
serialization
session cookie
session state
URL rewriting
web storage

https://hemanthrajhemu.github.io

13.9.2 Review Questions

1. Why is state a problem for web applications?
2. What are HTTP cookies? What is their purpose?
3. Describe exactly how cookies work.
4. What is the difference between session cookies and persistent cookies? How does the browser know which type of cookie to create?
5. Describe best practices for using persistent cookies.
6. What is web storage in HTML5? How does it differ from HTTP cookies?
7. What is session state?
8. Describe how session state works.
9. In PHP, how are sessions stored between requests?
10. How does object serialization relate to stored sessions in PHP?
11. What is a web farm? What issues do they create for session state management?
12. What is caching in the context of web applications? What benefit does it provide?
13. What is the difference between page output caching and application data caching?

13.9.3 Hands-On Practice

PROJECT 1: Book Rep Customer Relations Management

DIFFICULTY LEVEL: Intermediate

HANDS-ON EXERCISES

PROJECT 13.1

Overview

Demonstrate your ability to work with PHP by converting the login page we have been developing over the last few chapters into a functioning authentication system that remembers a valid login using sessions and allows users to log out.

Instructions

1. You may begin by using your login page started in Chapter 8. You may recall that PHP did some simple field validation in PHP.
2. Create a new SQL table to store user credentials. Store at least one auto-generated `UserID`, as well as a *username* and *password* combination they will use to log in (we will return to this in later projects).
3. Modify the PHP script to handle the submitted form by validating the passed credentials against the database users. Upon a successful login, display a welcome message; otherwise, display the login page with an error message. Note: Remember to sanitize your user inputs.
4. Add the `session_start()` to your existing PHP script to add session functionality. Note: If you are using multiple files, be sure to include `session_start()` in each one that has to make use of the session variables.

https://hemanthrajhemu.github.io

5. When the user login is successful, set a new `$_SESSION['UserID']` variable to hold a value associated with the session. For the sake of simplicity, store the user's unique `UserID` from our database schema.
6. Modify the PHP code that generates the login page. Have it first check the `$_SESSION` variable to see if a user ID has been set. If the session is not set, it displays a login page like before; otherwise, it should output a Welcome Page, with a link to log out as illustrated in Figure 13.15.
7. Finally, create a page named `logout.php`, which calls the `session_start()` function, and then resets all the `$_SESSION` variables, calls `session_destroy()`, and then redirects back to the page it was clicked from (using the `$_SERVER['REFERER']` value).

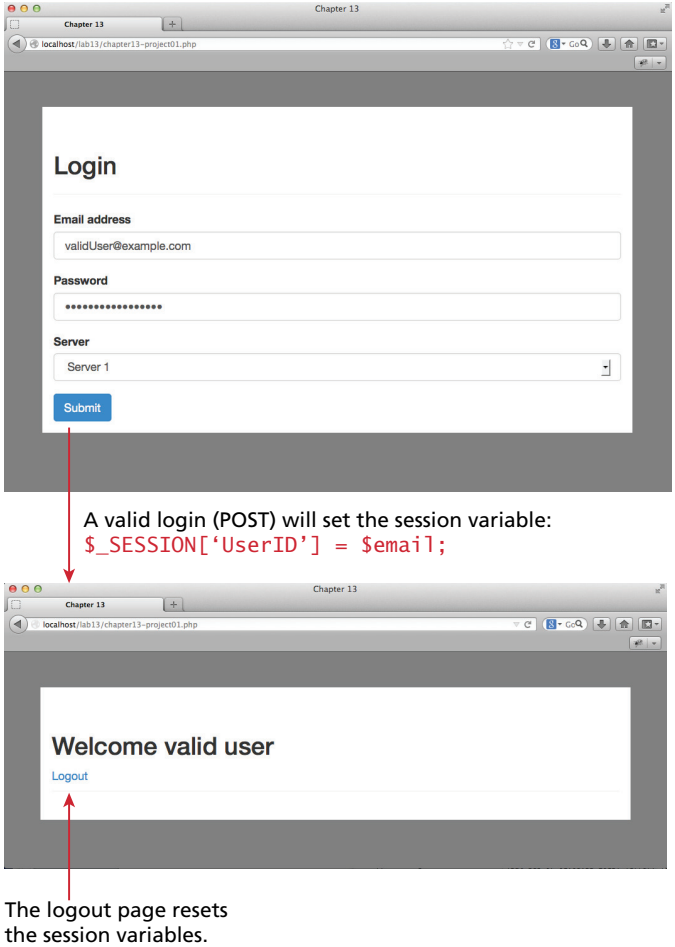


FIGURE 13.15 Completed Project 1

https://hemanthrajhemu.github.io

Test

1. Create one user in the database for testing purposes.
2. Test the page by logging in with correct credentials. You should see the Welcome screen and the Logout link.
3. Try navigating away from the page and coming back to the login page. You should see the Welcome message (i.e., the session is being saved between visits).
4. Verify that clicking logout results in the session ending and starting back at the login page.

PROJECT 2: Art Store

DIFFICULTY LEVEL: Intermediate

HANDS-ON EXERCISES

PROJECT 13.2

Overview

Building on the HTML and PHP pages already created in earlier chapters, you will add the functionality to manage a simple shopping cart, within a session variable.

Instructions

1. Begin by finding the project folder you have created for the Art Store. Session integration requires adding the `session_start()` function call to all pages that will participate in the session integration.
2. Create a new blank page, addToCart.php, which will handle a `GET` request to add something to the shopping cart.
3. Update display-art-work.php to include a link for the addToShoppingCart button in the format:

   ```
   addToCart.php?artworkID=122
   ```

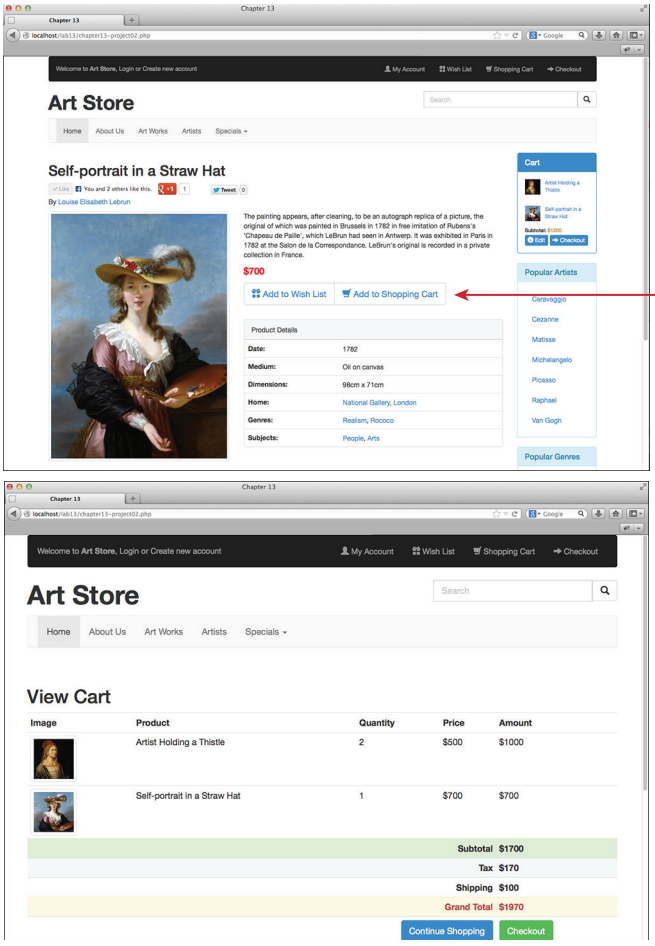
4. addToCart.php should process the ID of the desired item and update the $_SESSION['ShoppingCart'] variable as an array appropriately.

Create another PHP page in your project based on the HTML page from Chapter 8 that generates a dynamic shopping cart page based on the items in the session variable as illustrated in Figure 13.16. This same session should populate a small widget on the display-art-work.php page as well.

Test

1. Test the page by starting to add items to the list.
2. Notice that even without logging in, the session is able to track multiple users. Surf to the site on two computers at the same time and note how the wish lists are distinct on each.

https://hemanthrajhemu.github.io



The mini cart will be built based on the session.

When "Add to Shopping Cart" is clicked, the ID of the current artwork is added to the session variable.

Use the values in the session to build the shopping cart page.

FIGURE 13.16 Completed Project 2

PROJECT 3: Art Store

DIFFICULTY LEVEL: Intermediate

Overview

This project utilizes page caching to improve the performance of your Art Store project.

Instructions

1. Download and install the PECL extension, which supports memcache.
2. For the Art gallery pages, which require several JOINs in order to create the desired SQL result set, we will add caching of pages.

HANDS-ON EXERCISES
PROJECT 13.3

https://hemanthrajhemu.github.io

3. Write code that either retrieves from or stores to the cache when the Art Gallery pages are requested. Refer to Listing 13.10 for an example of this logic.

Instructions

1. Test the page by visiting a gallery, which you assume will generate a cached page.
2. Turn off your database server, or temporarily rename the table with the gallery records, to break any queries. Revisit the page you just visited, and it should display the cached copy.
3. Wait the amount of time you specified in cache, and revisit the page. If the SQL database is still offline, you should see an error (which will be cached!).
4. Turn the SQL server back on and confirm that everything is running as expected.

13.10 References

1. PHP, "setcookie." [Online]. http://www.php.net/manual/en/function.setcookie.php.
2. PHP, "Object Serialization." [Online]. http://php.net/manual/en/language.oop5.serialization.php.
3. PHP, "Session Handling." [Online]. http://ca1.php.net/manual/en/book.session.php.
4. W3C, "Web Storage." [Online]. http://www.w3.org/TR/webstorage/.
5. PECL, "PECL PHP Extensions." [Online]. http://pecl.php.net/.

https://hemanthrajhemu.github.io

Advanced JavaScript & jQuery 15

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- About JavaScript pseudo-classes, prototypes, and object-oriented design
- About JavaScript frameworks such as jQuery
- How to post files asynchronously with JavaScript
- How jQuery can be used to animate page elements

Now that you have learned the fundamentals of JavaScript (Chapter 6) and server-side development (Chapters 8–14), you are ready to learn advanced client-side scripting, which will allow you to design and build more efficient and maintainable JavaScript code. This chapter also examines two JavaScript frameworks (jQuery and Backbone), which facilitate the creation of engaging and interactive user experiences by simplifying the listener and AJAX mechanisms. These frameworks remove many of the headaches associated with dealing with multiple browser differences, and allow the developer to focus on core features and logic rather than nitty-gritty details. Finally, this chapter provides instructions in the design and implementation of AJAX web pages.

https://hemanthrajhemu.github.io

15.1 JavaScript Pseudo-Classes

Although JavaScript has no formal class mechanism, it does support objects (such as the DOM). While most object-oriented languages that support objects also support classes formally, JavaScript does not. Instead, you define pseudo-classes through a variety of interesting and nonintuitive syntax constructs. Many common features of object-oriented programming, such as inheritance and even simple methods, must be arrived at through these nonintuitive means. Despite this challenge, the benefits of using object-oriented design in your JavaScript include increased code reuse, better memory management, and easier maintenance. From a practical perspective, almost all modern frameworks (such as jQuery and the Google Maps API) use prototypes to simulate classes, so understanding the mechanism is essential to apply those APIs in your applications.

This section will demonstrate how you mimic class features through the creation of a simple prototype to represent a single die object (die, the singular for dice) which could be used in a game of some sort. This process will begin with the simplest mechanisms and introduce new syntactic constructs until we arrive at the best way to create and use pseudo-classes (prototypes) in JavaScript.

15.1.1 Using Object Literals

Recall that an array in JavaScript can be instantiated with elements in the following way:

```
var daysofWeek = ["sun","mon","tue","wed","thu","fri","sat"];
```

An object can be instantiated using the similar concept of object literals: that is, an object represented by the list of key-value pairs with colons between the key and value with commas separating key-value pairs.

A dice object, with a string to hold the color and an array containing the values representing each side (face), could be defined all at once using object literals as follows:

PRO TIP

Object literals are also known as **Plain Objects** in jQuery. Plain Objects are also commonly used to encapsulate data for asynchronous post requests to the server rather than using URL encoded query strings as done for a `GET` request. Object literals are also used in Chapter 17 on web services and Chapter 21 on social network integration.

https://hemanthrajhemu.github.io

```
var oneDie = { color : "FF0000", faces : [1,2,3,4,5,6] };
```

Once defined, these elements can be accessed using dot notation. For instance, one could change the color to blue by writing:

```
oneDie.color="0000FF";
```

15.1.2 Emulate Classes through Functions

Although a formal *class* mechanism is not available to us in JavaScript, it is possible to get close by using functions to encapsulate variables and methods together, as shown in Listing 15.1.

HANDS-ON EXERCISES

LAB 15 EXERCISE
Define a Class

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}
```

LISTING 15.1 Very simple Die pseudo-class definition as a function

The `this` keyword inside of a function refers to the instance, so that every reference to internal properties or methods manages its own variables, as is the case with PHP. One can create an instance of the object as follows, very similar to PHP.

```
var oneDie = new Die("0000FF");
```

Developers familiar with using objects in Java or PHP typically use a constructor to instantiate objects. In JavaScript, there is no need for an explicit constructor since the function definition acts as both the definition of the pseudo-class and its constructor.

Adding Methods to the Object

One of the most common features one expects from a class is the ability to define behaviors with methods. In JavaScript this is relatively easy to do syntactically.

To define a method in an object's function one can either define it internally, or use a reference to a function defined outside the class. External definitions can quickly cause namespace conflict issues, since all method names must remain conflict free with all other methods for other classes. For this reason, one technique for adding a method inside of a class definition is by assigning an anonymous function to a variable, as shown in Listing 15.2.

With this method so defined, all dice objects can call the `randomRoll` function, which will return one of the six faces defined in the Die constructor.

https://hemanthrajhemu.github.io

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];

   // define method randomRoll as an anonymous function
   this.randomRoll = function() {
      var randNum = Math.floor((Math.random() * this.faces.length)+ 1);
      return faces[randNum-1];
   };
}
```

LISTING 15.2 Die pseudo-class with an internally defined method

```
var oneDie = new Die("0000FF");
console.log(oneDie.randomRoll() + " was rolled");
```

Although this mechanism for methods is effective, it is not a memory-efficient approach because each inline method is redefined for each new object. Unlike a PHP or Java class, an anonymous function in JavaScript is not defined once. Figure 15.1 illustrates how two instances of a `Die` object define two (identical) definitions of the `randomRoll` method.

Just imagine if you had 100 `Die` objects created; you would be redefining every method 100 times, which could have a noticeable effect on client execution speeds and browser responsiveness. To prevent this needless waste of memory, a better approach is to define the method just once using a *prototype* of the class.

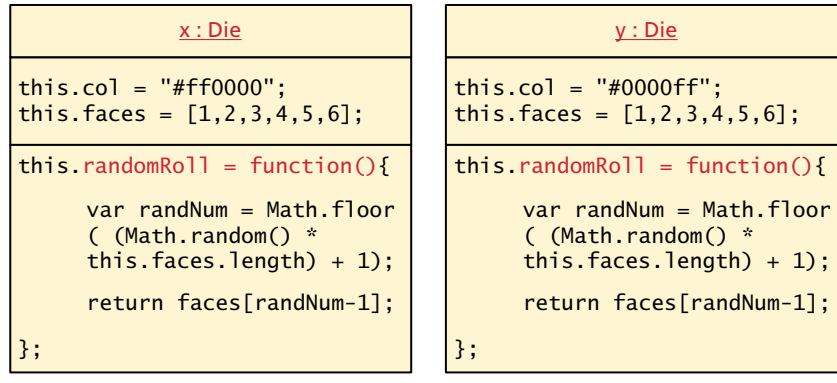


FIGURE 15.1 Illustrating duplicated method definition

https://hemanthrajhemu.github.io

15.1.3 Using Prototypes

Prototypes are an essential syntax mechanism in JavaScript, and are used to make JavaScript behave more like an object-oriented language. The `prototype` properties and methods are defined *once* for all instances of an *object*. So now you can modify the definition of the `randomRoll()` method once again, by changing our *Die* in Listing 15.2 to that in Listing 15.3 by moving the `randomRoll()` method into the prototype.

```
// Start Die Class
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}

Die.prototype.randomRoll = function() {
   var randNum = Math.floor((Math.random() * this.faces.length) + 1);
   return faces[randNum-1];
};
// End Die Class
```

LISTING 15.3 The Die pseudo-class using the prototype object to define methods

This definition is better because it defines the method only once, no matter how many instances of `Die` are created. In contrast to the duplicated code in Figure 15.1, Figure 15.2 shows how the `prototype` object (not class) is updated to contain the method so that subsequent instantiations (`x` and `y`) reference that one-method

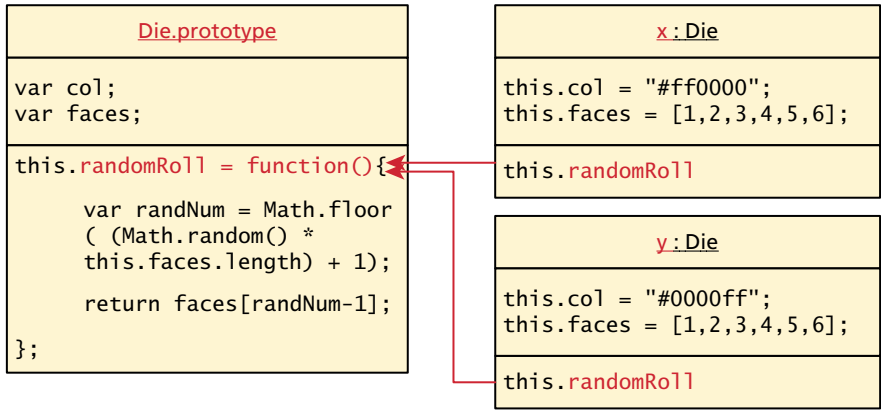


FIGURE 15.2 Illustration of JavaScript prototypes as pseudo-classes

https://hemanthrajhemu.github.io

definition. Since all instances of a `Die` share the same prototype object, the function declaration only happens one time and is shared with all `Die` instances.

More about Prototypes

HANDS-ON EXERCISES

LAB 15 EXERCISE
Prototype Function

Even experienced JavaScript programmers sometimes struggle with the prototype concept. It should be known that every object (and method) in JavaScript has a prototype.

A prototype is an object from which other objects inherit.

The above definition sounds almost like a class in an object-oriented language, except that a prototype is itself an *object*, whereas in other oriented-oriented languages a class is an abstraction, not an object. Despite this distinction, you can make use of a function's prototype object, and assign properties or methods to it that are then available to any new objects that are created.

In addition to the obvious application of prototypes to our own pseudo-classes, prototypes enable you to *extend* existing classes by adding to their prototypes! Imagine a method added to the `String` object, which allows you to count instances of a character. Listing 15.4 defines just such a method, named `countChars`, that takes a character as a parameter.

```
String.prototype.countChars = function (c) {
   var count=0;
   for (var i=0;i<this.length;i++) {
      if (this.charAt(i) == c)
         count++;
   }
   return count;
}
```

LISTING 15.4 Adding a method named countChars to the String class

Now any new instances of `String` will have this method available to them (created using the `new` keyword), while existing strings will not. You could use the new method on any strings instantiated after the prototype definition was added. For instance the following example will output `Hello World has 3 letter l's`.

```
var hel = "Hello World";
console.log(hel + "has" + hel.countChars("l") + " letter l's");
```

This technique is also useful to assign properties to a pseudo-class that you want available to all instances. Imagine an array of all the *valid* characters

https://hemanthrajhemu.github.io

attached to some custom string class. Again using prototype you could define such a list.

```
CustomString.prototype.validChars = ["A","B","C"];
```

Prototypes are certainly one of the hardest syntactic mechanisms to learn in JavaScript and are a poor choice for teaching object-oriented design to students. You must, however, understand and make use of them: even helpful frameworks like jQuery make extensive use of prototypes.

15.2 jQuery Foundations

A library or framework is software that you can utilize in your own software, which provides some common implementations of standard ideas. A web framework can be expected to have features related to the web including HTTP headers, AJAX, authentication, DOM manipulation, cross-browser implementations, and more.

jQuery's beginnings date back to August 2005, when jQuery founder John Resig was looking into how to better combine CSS selectors with succinct JavaScript notation.[1] Within a year, AJAX and animations were added, and the project has been improving ever since. Additional modules (like the popular jQuery UI extension and recent additions for mobile device support) have considerably extended jQuery's abilities. Many developers find that once they start using a framework like jQuery, there's no going back to "pure" JavaScript because the framework offers so many useful shortcuts and succinct ways of doing things. jQuery is now the most popular JavaScript library currently in use as supported by the statistics in Figure 15.3.

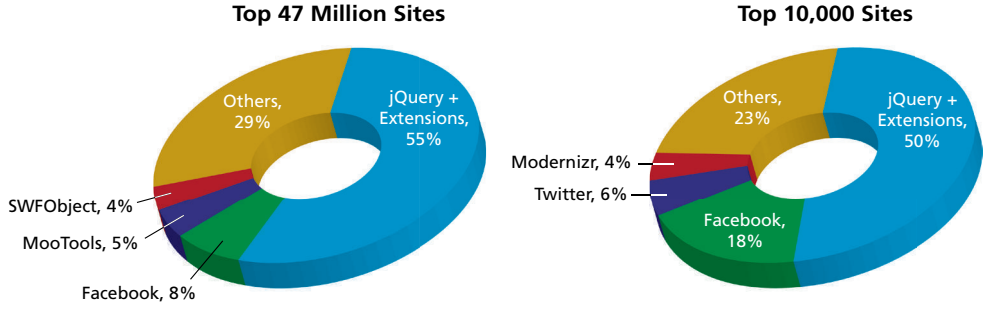


FIGURE 15.3 Comparison of the most popular JavaScript frameworks (data courtesy of BuiltWith.com)

https://hemanthrajhemu.github.io

jQuery bills itself as the *write less, do more* framework.[2] According to its website

> *jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.*

To really benefit from jQuery, you must understand how and why it replaces some JavaScript techniques and code regarding selectors, attributes, and AJAX with more succinct syntax that also includes improvements and enhancements. It should be noted that ideas and syntax learned in Chapter 6 will be used since jQuery is still JavaScript and must make use of the loops, conditionals, variables, and prototypes of that language.

15.2.1 Including jQuery in Your Page

HANDS-ON EXERCISES

LAB 15 EXERCISE
Set Up jQuery

Since the entire library exists as a source JavaScript file, importing jQuery for use in your application is as easy as including a link to a file in the <head> section of your HTML page. You must either link to a locally hosted version of the library or use an approved third-party host, such as Google, Microsoft, or jQuery itself.

Using a third-party content delivery network (CDN) is advantageous for several reasons. Firstly, the bandwidth of the file is offloaded to reduce the demand on your servers. Secondly, the user may already have cached the third-party file and thus not have to download it again, thereby reducing the total loading time. This probability is increased when using a CDN like Google rather than a developer-focused CDN like jQuery.

A disadvantage to the third-party CDN is that your jQuery will fail if the third-party host fails, although that is unlikely given the mission-critical demands of large companies like Google and Microsoft.

To achieve the benefits of the CDN and increase reliability on the rare occasion it might be down, you can write a small piece of code to check if the first attempt to load jQuery was successful. If not, you can load the locally hosted version. This setup should be included in the <head> section of your HTML page as shown in Listing 15.5.

```
<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
<script type="text/javascript">
window.jQuery ||
document.write('<script src="/jquery-1.9.1.min.js"><\/script>');
</script>
```

LISTING 15.5 jQuery loading using a CDN and a local fail-safe if the CDN is offline

https://hemanthrajhemu.github.io

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:

HANDS-ON EXERCISES

LAB 15 EXERCISE
Basic Selectors

- `$("*")` **Universal selector** matches all elements (and is slow).
- `$("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");
```

https://hemanthrajhemu.github.io

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

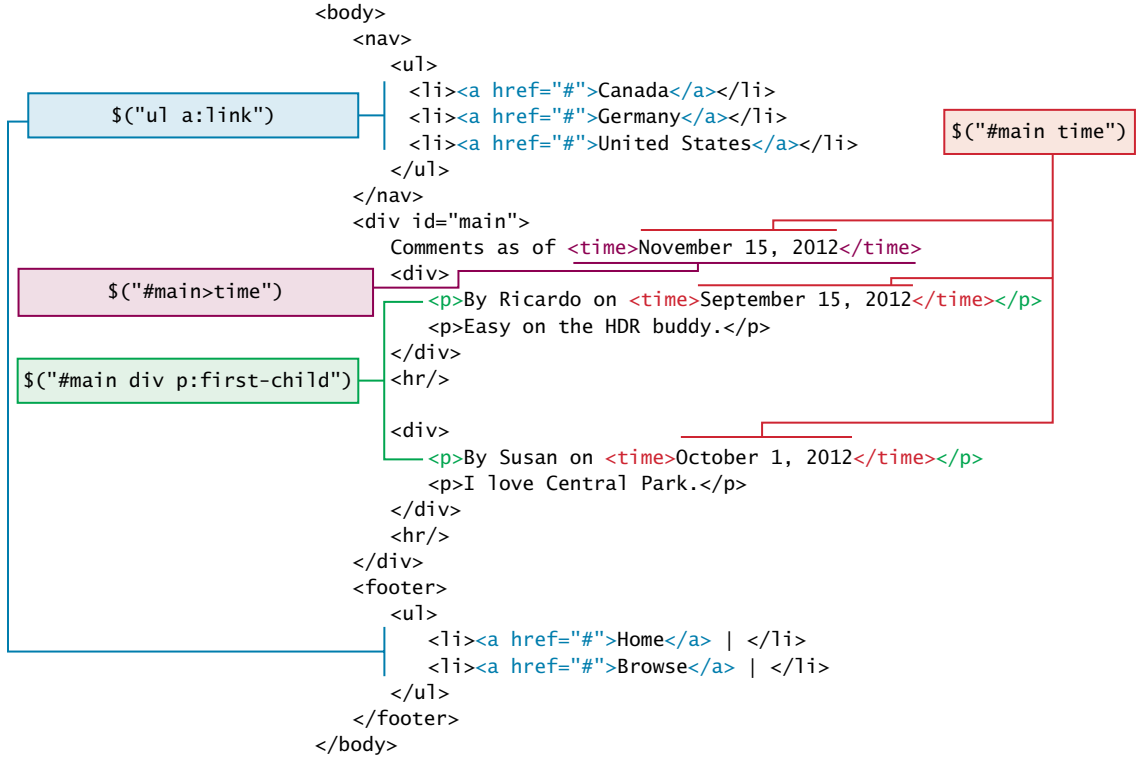


FIGURE 15.4 Illustration of some jQuery selectors and the HTML being selected

https://hemanthrajhemu.github.io

browsers implemented it. jQuery overcomes those browser limitations, providing the ability to select elements by attribute. A list of sample CSS attribute selectors was given in Chapter 3 (Table 3.4), but to jog your memory with an example, consider a selector to grab all `<img>` elements with an `src` attribute beginning with `/artist/` as:

```
var artistImages = $("img[src^='/artist/']");
```

Recall that you can select by attribute with square brackets (`[attribute]`), specify a value with an equals sign (`[attribute=value]`) and search for a particular value in the beginning, end, or anywhere inside a string with ^, $, and * symbols respectively (`[attribute^=value]`, `[attribute$=value]`, `[attribute*=value]`).

Pseudo-Element Selector

Pseudo-elements are special elements, which are special cases of regular ones. As you may recall from Chapter 3, these **pseudo-element selectors** allow you to append to any selector using the colon and one of `:link`, `:visited`, `:focus`, `:hover`, `:active`, `:checked`, `:first-child`, `:first-line`, and `:first-letter`.

These selectors can be used in combination with the selectors presented above, or alone. Selecting all links that have been visited, for example, would be specified with:

```
var visitedLinks = $("a:visited");
```

Since this chapter reviews and builds on CSS selectors, you are hopefully remembering some of the selectors you have used earlier and are making associations between those selectors and the ones in jQuery. As you already know from Chapter 6, once you have the ability to select an element, you can do many things to manipulate that element from changing its content or style all the way to removing it.

Contextual Selector

Another powerful CSS selector included in jQuery's selection mechanism is the **contextual selectors** introduced in Chapter 3. These selectors allowed you to specify elements with certain relationships to one another in your CSS. These relationships included descendant (space), child (>), adjacent sibling (+), and general sibling (~).

To select all `<p>` elements inside of `<div>` elements you would write

```
var para = $("div p");
```

HANDS-ON EXERCISES

LAB 15 EXERCISE
Advanced Selectors

Content Filters

The **content filter** is the only jQuery selector that allows you to append filters to all of the selectors you've used thus far and match a particular pattern. You can select

https://hemanthrajhemu.github.io

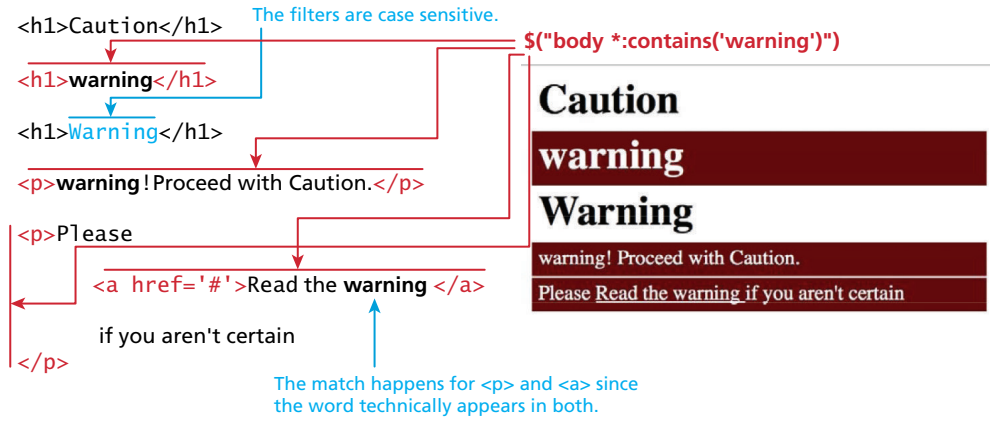


FIGURE 15.5 An illustration of jQuery's content filter selector

elements that have a particular child using `:has()`, have no children using `:empty`, or match a particular piece of text with `:contains()`. Consider the following example:

```
var allWarningText = $("body *:contains('warning')");
```

It will return a list of all the DOM elements with the word *warning* inside of them. You might imagine how we may want to highlight those DOM elements by coloring the background red as shown in Figure 15.5 with one line of code:

```
$("body *:contains('warning')").css("background-color", "#aa0000");
```

Form Selectors

Since form HTML elements are well known and frequently used to collect and transmit data, there are jQuery selectors written especially for them. These selectors, listed in Table 15.1, allow for quick access to certain types of field as well as fields in certain states.

15.2.3 jQuery Attributes

With all of the selectors described in this chapter, you can select any set of elements that you want from a web page. In order to understand how to fully manipulate the elements you now have access to, one must understand an element's *attributes* and *properties*.

HTML Attributes

The core set of attributes related to DOM elements are the ones specified in the HTML tags described in Chapter 2. You have by now integrated many of the key

https://hemanthrajhemu.github.io

Selector	CSS Equivalent	Description
$(:button)	$("button, input[type='button']")	Selects all *buttons.*
$(:checkbox)	$('[type=checkbox]')	Selects all *checkboxes.*
$(:checked)	No equivalent	Selects elements that are checked. This includes radio buttons and checkboxes.
$(:disabled)	No equivalent	Selects form elements that are disabled. These could include <button>, <input>, <optgroup>, <option>, <select>, and <textarea>
$(:enabled)	No equivalent	Opposite of :disabled. It returns all elements where the disabled attribute=false as well as form elements with no disabled attribute.
$(:file)	$('[type=file]')	Selects all elements of type file.
$(:focus)	$(document.activeElement)	The element with focus.
$(:image)	$('[type=image]')	Selects all elements of type image.
$(:input)	No equivalent	Selects all <input>, <textarea>, <select>, and <button> elements.
$(:password)	$('[type=password]')	Selects all password fields.
$(:radio)	$('[type=radio]')	Selects all radio elements.
$(:reset)	$('[type=reset]')	Selects all the reset buttons.
$(:selected)	No equivalent	Selects all the elements that are currently selected of type <option>. It does not include checkboxes or radio buttons.
$(:submit)	$('[type=submit]')	Selects all submit input elements.
$(:text)	No equivalent	Selects all input elements of type text. $('[type=text]') is almost the same, except that $(:text) includes <input>. fields with no type specified.

TABLE 15.1 jQuery form selectors and their CSS equivalents when applicable

https://hemanthrajhemu.github.io

PRO TIP

The selectors that use CSS syntax do not benefit from the presence of the `querySelectorAll()` function included in newer versions of native JavaScript. When speed is an important consideration, use "pure" CSS selector syntax rather than these shortcuts. This means `$('[type=password]')` will be faster than `$(:password)` in many situations, although `querySelectorAll()` is still more flexible.

Another speed consideration for these selectors is that they by default are implied to search the entire DOM tree. This means `$(":focus")` is equivalent to `$("*:focus")`. To improve the script efficiency, be as specific as possible in your selectors to reduce the amount of DOM traversal.

attributes like the `href` attribute of an `<a>` tag, the `src` attribute of an `<img>`, or the `class` attribute of most elements.

In jQuery we can both set and get an attribute value by using the `attr() method` on any element from a selector. This function takes a parameter to specify which attribute, and the optional second parameter is the value to set it to. If no second parameter is passed, then the return value of the call is the current value of the attribute. Some example usages are:

```
// var link is assigned the href attribute of the first <a> tag
var link = $("a").attr("href");

// change all links in the page to http://funwebdev.com
$("a").attr("href","http://funwebdev.com");

// change the class for all images on the page to fancy
$("img").attr("class","fancy");
```

HTML Properties

HANDS-ON EXERCISES

LAB 15 EXERCISE
Properties and Attributes

Many HTML tags include properties as well as attributes, the most common being the *checked* property of a radio button or checkbox. In early versions of jQuery, HTML properties could be set using the `attr()` method. However, since properties are not technically attributes, this resulted in odd behavior. The `prop()` method is now the preferred way to retrieve and set the value of a property although, `attr()` may return some (less useful) values.

To illustrate this subtle difference, consider a DOM element defined by

```
<input class ="meh" type="checkbox" checked="checked">
```

The value of the `attr()` and `prop()` functions on that element differ as shown below.

https://hemanthrajhemu.github.io

```
var theBox = $(".meh");
theBox.prop("checked") // evaluates to TRUE
theBox.attr("checked") // evaluates to "checked"
```

Changing CSS

Changing a CSS style is syntactically very similar to changing attributes. jQuery provides the extremely intuitive `css()` methods. There are two versions of this method (with two different method signatures), one to get the value and another to set it. The first version takes a single parameter containing the CSS attribute whose value you want and returns the current value.

```
$color = $("#colourBox").css("background-color"); // get the color
```

To modify a CSS attribute you use the second version of `css()`, which takes two parameters: the first being the CSS attribute, and the second the value.

```
// set color to red
$("#colourBox").css("background-color", "#FF0000");
```

If you want to use classes instead of overriding particular CSS attributes individually, have a look at the additional shortcut methods described in the jQuery documentation.

Shortcut Methods

HANDS-ON EXERCISES

LAB 15 EXERCISE
Change the Styles

jQuery allows the programmer to rely on foundational HTML attributes and properties exclusively as described above. However, as with selectors, there are additional functions that provide easier access to common operations such as changing an object's class or the text within an HTML tag.

The `html()` method is used to get the HTML contents of an element (the part between the <> and </> tags associated with the `innerHTML` property in JavaScript). If passed with a parameter, it updates the HTML of that element.

The `html()` method should be used with caution since the inner HTML of a DOM element can itself contain nested HTML elements! When replacing DOM with text, you may inadvertently introduce DOM errors since no validation is done on the new content (the browser wouldn't want to presume).

You can enforce the DOM by manipulating `textNode` objects and adding them as children to an element in the DOM tree rather than use `html()`. While this enforces the DOM structure, it does complicate code. To illustrate, consider that you could replace the content of every `<p>` element with "jQuery is fun," with the one line of code:

```
$("p").html("jQuery is fun");
```

https://hemanthrajhemu.github.io

The shortcut methods `addClass(className)` / `removeClass(className)` add or remove a CSS class to the element being worked on. The `className` used for these functions can contain a space-separated list of classnames to be added or removed.

The `hasClass(classname)` method returns `true` if the element has the `className` currently assigned. False, otherwise. The `toggleClass(className)` method will add or remove the class `className`, depending on whether it is currently present in the list of classes. The `val()` method returns the value of the element. This is typically used to retrieve values from input and select fields.

15.2.4 jQuery Listeners

Just like JavaScript, jQuery supports creation and management of listeners/handlers for JavaScript events. The usage of these events is conceptually the same as with JavaScript with some minor syntactic differences.

Set Up after Page Load

In JavaScript, you learned why having your listeners set up inside of the `window.onload()` event was a good practice. Namely, it ensured the entire page and all DOM elements are loaded before trying to attach listeners to them. With jQuery we do the same thing but use the `$(document).ready()` event as shown in Listing 15.6.

```
$(document).ready(function(){
 //set up listeners on the change event for the file items.
 $("input[type=file]").change(function(){
     console.log("The file to upload is "+ this.value);
  });
});
```

LISTING 15.6 jQuery code to listen for file inputs changing, all inside the document's ready event

What is really happening is we are attaching our code to the handler for the `document.ready` event, which triggers when the page is fully downloaded and parsed into its DOM representation.

Listener Management

HANDS-ON EXERCISES

LAB 15 EXERCISE
jQuery Listeners

Setting up listeners for particular events is done in much the same way as JavaScript. While pure JavaScript uses the `addEventListener()` method, jQuery has `on()` and `off()` methods as well as shortcut methods to attach events. Modifying the code in Listing 15.6 to use listeners rather than one handler yields the more modular code in Listing 15.7. Note that the shortcut `:file` selector is used in place of the equivalent `input[type=file]`.

https://hemanthrajhemu.github.io

```
$(document).ready(function(){
   $(":file").on("change",alertFileName); // add listener
});
// handler function using this
function alertFileName() {
   console.log("The file selected is: "+this.value);
}
```

LISTING 15.7 Using the listener technique in jQuery with on and off methods

Listeners in jQuery become especially necessary once we start using AJAX since the advanced handling of those requests and responses can get quite complicated, and well-structured code using listeners will help us better manage that complexity.

15.2.5 Modifying the DOM

jQuery comes with several useful methods to manipulate the DOM elements themselves. We have already seen how the `html()` function can be used to manipulate the inner contents of a DOM element and how `attr()` and `css()` methods can modify the internal attributes and styles of an existing DOM element.

Creating DOM and textNodes

If you decide to think about your page as a DOM object, then you will want to manipulate the tree structure rather than merely manipulate strings. Thankfully, jQuery is able to convert strings containing valid DOM syntax into DOM objects automatically.

Recall that the basic act of creating a DOM node in JavaScript uses the `createElement()` method:

```
var element = document.createElement('div'); //create a new DOM node
```

However, since the jQuery methods to manipulate the DOM take an HTML string, jQuery objects, or DOM objects as parameters, you might prefer to define your element as

```
var element = $("<div></div>"); //create new DOM node based on html
```

This way you can apply all the jQuery functions to the object, rather than rely on pure JavaScript, which has fewer shortcuts. If we consider creation of a simple `<a>` element with multiple attributes, you can see the comparison of the JavaScript and jQuery techniques in Listing 15.8.

https://hemanthrajhemu.github.io

```
 // pure JavaScript way
var jsLink = document.createElement("a");
jsLink.href = "http://www.funwebdev.com";
jsLink.innerHTML = "Visit Us";
jsLink.title = "JS";

// jQuery way
var jQueryLink = $("<a href='http://funwebdev.com'
                       title = 'jQuery'>Visit Us</a>");

// jQuery long-form way
var jQueryVerboseLink = $("<a></a>");
jQueryVerboseLink.attr("href",'http://funwebdev.com');
jQueryVerboseLink.attr("title","jQuery verbose");
jQueryVerboseLink.html("Visit Us");
```

LISTING 15.8 A comparison of node creation in JS and jQuery

Prepending and Appending DOM Elements

HANDS-ON EXERCISES

LAB 15 EXERCISE
Inserting DOM Elements

When an element is defined in any of the ways described above, it must be inserted into the existing DOM tree. You can also insert the element into several places at once if you desire, since selectors can return an array of DOM elements.

The `append()` method takes as a parameter an HTML string, a DOM object, or a jQuery object. That object is then added as the last child to the element(s) being selected. In Figure 15.6 we can see the effect of an `append()` method call. Each element with a class of `linkOut` has the `jsLink` element defined in Listing 15.8 appended to it.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

```
$(".linkOut").append(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
<div>
```

FIGURE 15.6 Illustration of where append adds a node

https://hemanthrajhemu.github.io

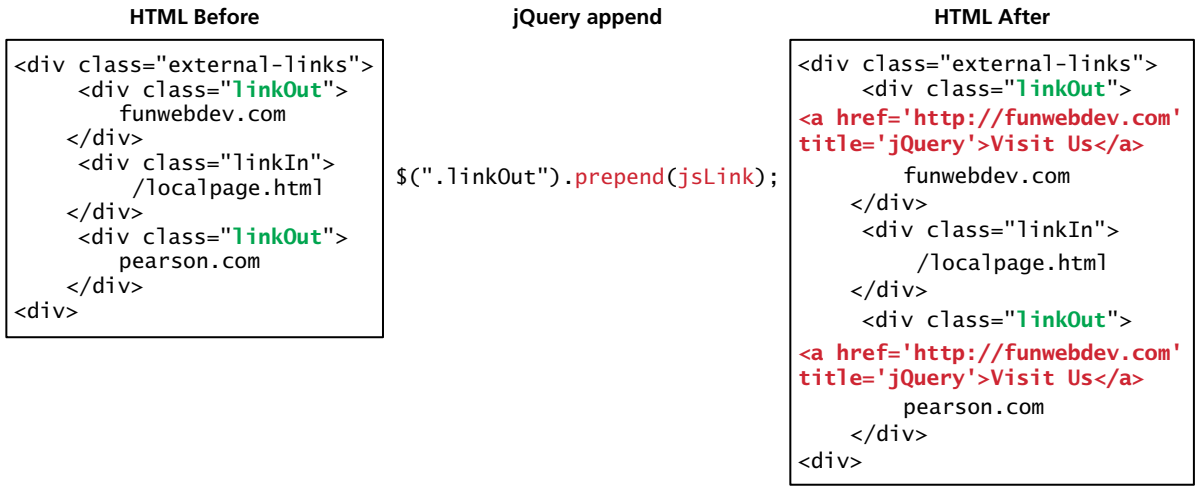


FIGURE 15.7 Illustration of prepend() adding a node

The `appendTo()` method is similar to `append()` but is used in the syntactically converse way. If we were to use `appendTo()`, we would have to switch the object making the call and the parameter to have the same effect as the previous code:

```
jsLink.appendTo($(".linkOut"));
```

The `prepend()` and `prependTo()` methods operate in a similar manner except that they add the new element as the first child rather than the last. See Figure 15.7 for an illustration of what happens with `prepend()`.

Wrapping Existing DOM in New Tags

One of the most common ways you can enhance a website that supports JavaScript is to add new HTML tags as needed to support some jQuery functions. Imagine for illustration purposes our art galleries being listed alongside some external links as described by the HTML in Listing 15.9.

```
<div class="external-links">
  <div class="gallery">Uffuzi Museum</div>
  <div class="gallery">National Gallery</div>
  <div class="link-out">funwebdev.com</div>
</div>
```

LISTING 15.9 HTML to illustrate DOM manipulation

https://hemanthrajhemu.github.io

If we wanted to wrap all the gallery items in the whole page inside, another (perhaps because we wish to programmatically manipulate these items later) with class galleryLink we could write:

```
$(".gallery").wrap('<div class="galleryLink"/>');
```

which modifies the HTML to that shown in Listing 15.10. Note how each and every link is wrapped in the correct opening and closing and uses the galleryLink class.

```
<div class="external-links">
  <div class="galleryLink">
      <div class="gallery">Uffuzi Museum</div>
  </div>
   <div class="galleryLink">
      <div class="gallery">National Gallery</div>
  </div>
   <div class="link-out">funwebdev.com</div>
</div>
```

LISTING 15.10 HTML from Listing 15.9 modified by executing the wrap statement above

In a related demonstration of how succinctly jQuery can manipulate HTML, consider the situation where you wanted to add a title element to each element that reflected the unique contents inside. To achieve this more sophisticated manipulation, you must pass a function as a parameter rather than a tag to the wrap() method, and that function will return a dynamically created element as shown in Listing 15.11.

```
$(".contact").wrap(function(){
  return "<div class='galleryLink' title='Visit " + $(this).html() +
         "'></div>";
});
```

LISTING 15.11 Using wrap() with a callback to create a unique div for every matched element

The wrap() method is a callback function, which is called for each element in a set (often an array). Each element then becomes this for the duration of one of the wrap() function's executions, allowing the unique title attributes as shown in Listing 15.12.

https://hemanthrajhemu.github.io

```
<div class="external-links">
  <div class="galleryLink" title="Visit Uffuzi Museum">
     <div class="gallery">Uffuzi Museum</div>
  </div>
  <div class="galleryLink" title="Visit National Gallery">
     <div class="gallery">National Gallery</div>
  </div>
  <div class="link-out">funwebdev.com</div>
</div>
```

LISTING 15.12 The modified HTML from Listing 15.9 after executing using wrap code from Listing 15.11

As with almost everything in jQuery, there is an inverse method to accomplish the opposite task. In this case, `unwrap()` is a method that does not take any parameters and whereas `wrap()` *added* a parent to the selected element(s), `unwrap()` *removes* the selected item's parent.

Other methods such as `wrapAll()` and `wrapInner()` provide additional controls over wrapping DOM elements. The details of those methods can be found in the online jQuery documentation.[3]

15.3 AJAX

Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser. This makes use of a browser's multi-threaded design and lets one thread handle the browser and interactions while other threads wait for responses to asynchronous requests.

NOTE

Chapter 6 briefly introduced AJAX in Section 6.1.2 with a high-level overview of how AJAX can improve the website experience for end users. You may want to go back to that section before moving forward.

Figure 15.8 annotates a UML sequence diagram where the white activity bars illustrate where computation is taking place. Between the request being sent and the response being received, the system can continue to process other requests from the client, so it does not appear to be waiting in a loading state.

https://hemanthrajhemu.github.io

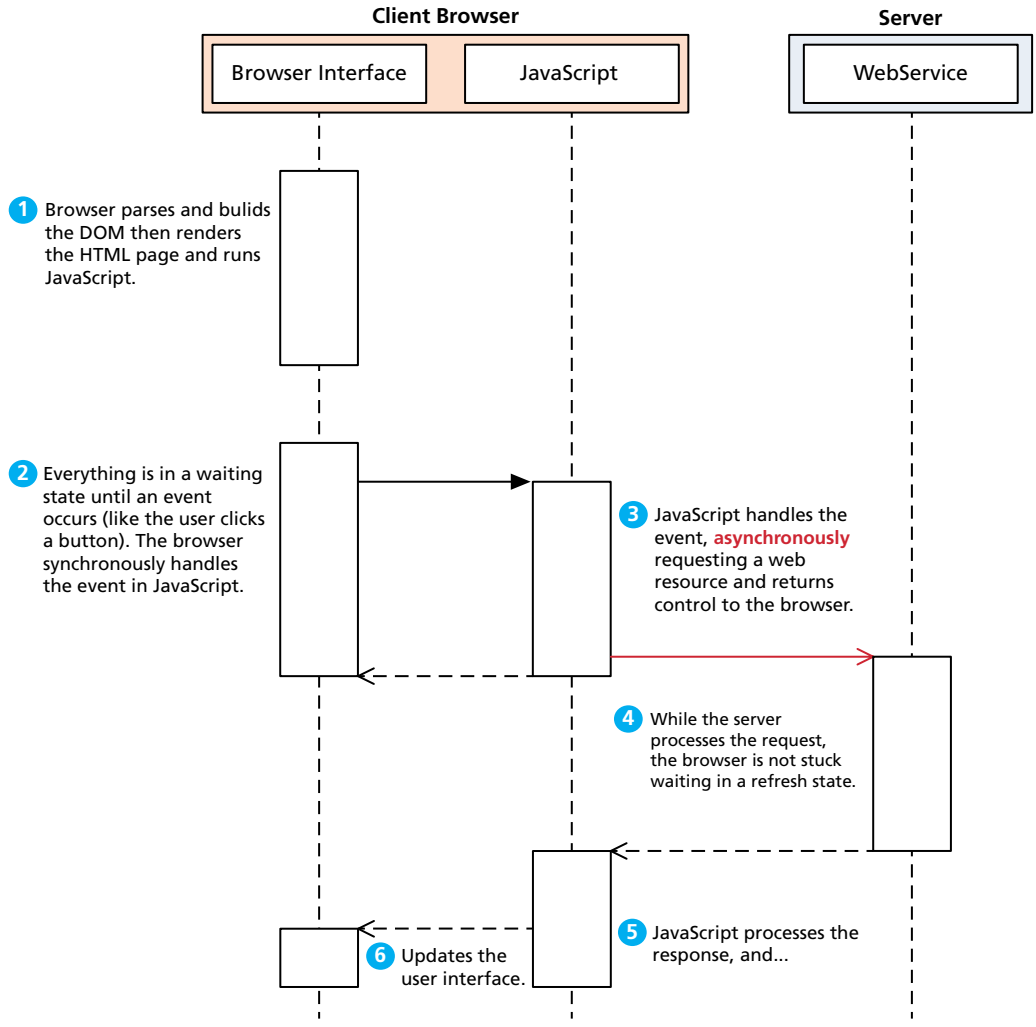


FIGURE 15.8 UML sequence diagram of an AJAX request

Responses to asynchronous requests are caught in JavaScript as events. The events can subsequently trigger changes in the user interface or make additional requests. This differs from the typical synchronous requests we have seen thus far, which require the entire web page to refresh in response to a request.

Another way to contrast AJAX and synchronous JavaScript is to consider a web page that displays the current server time as illustrated in Figure 15.9. If implemented synchronously, the entire page has to be refreshed from the server just to

https://hemanthrajhemu.github.io

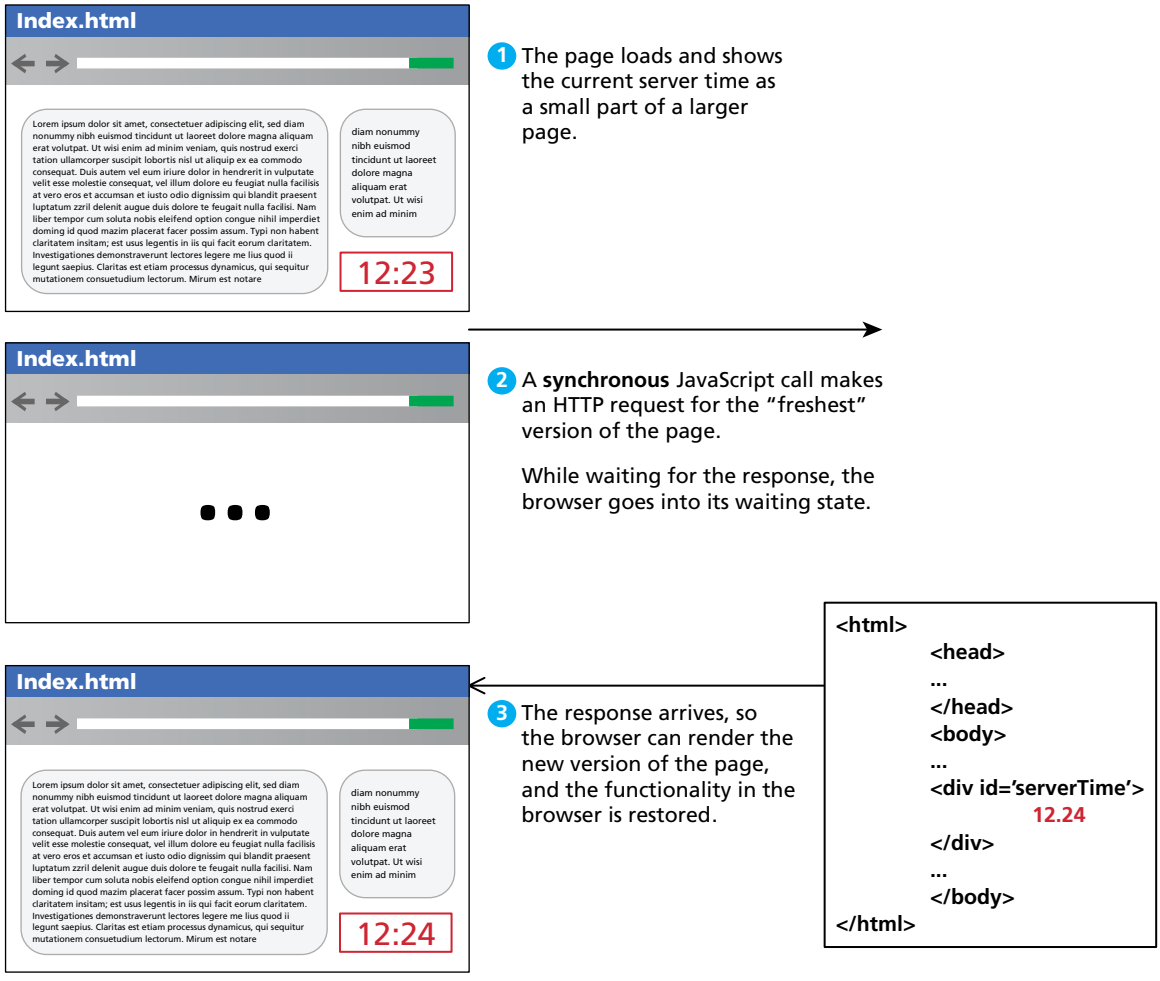


FIGURE 15.9 Illustration of a synchronous implementation of the server time web page.

update the displayed time. During that refresh, the browser enters a waiting state, so the user experience is interrupted (yes, you could implement a refreshing time using pure JavaScript, but for illustrative purposes, imagine it's essential to see the server's time).

In contrast, consider the very simple asynchronous implementation of the server time, where an AJAX request updates the server time in the background as illustrated in Figure 15.10.

In pure JavaScript it is possible to make asynchronous requests, but it's tricky and it differs greatly between browsers with Mozilla's `XMLHttpRequest` object and

https://hemanthrajhemu.github.io

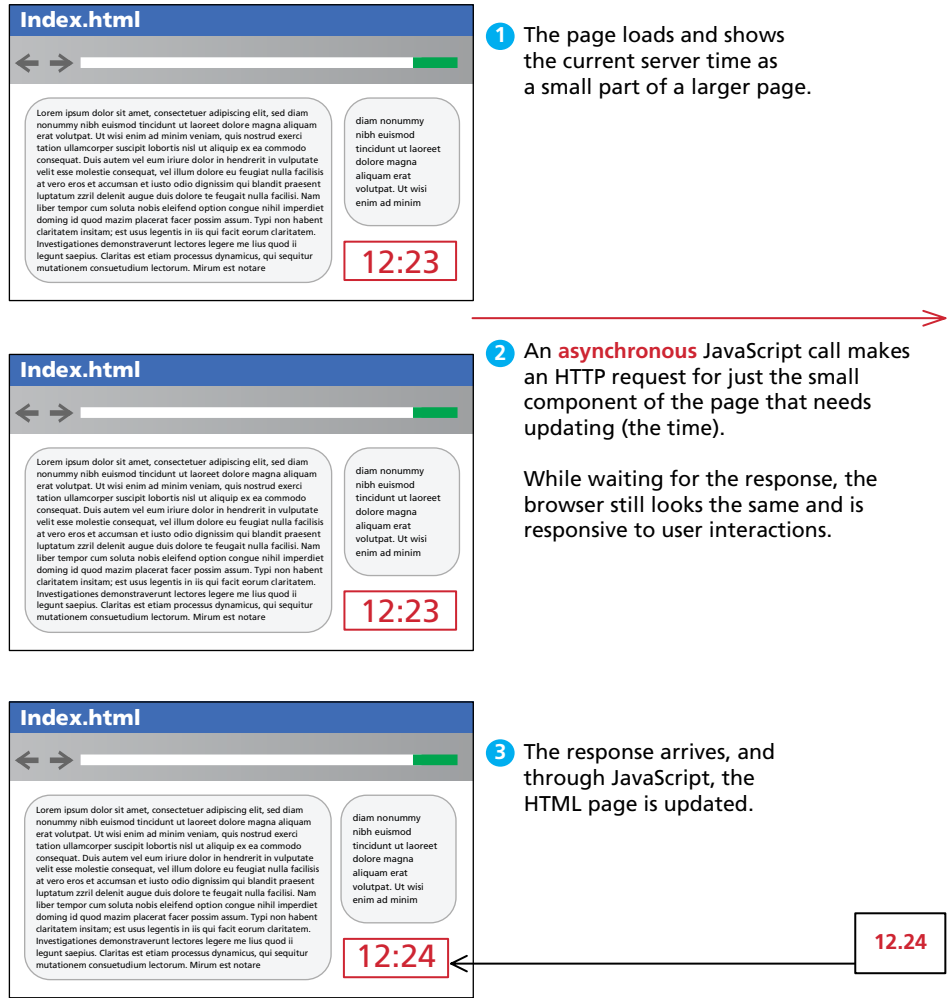


FIGURE 15.10 Illustration of an AJAX implementation of the server time widget

Internet Explorer's `ActiveX` wrapper. jQuery simplifies making asynchronous requests in different browsers by defining high-level methods that can work on any browser (and hiding the implementation details from the developer).

15.3.1 Making Asynchronous Requests

jQuery provides a family of methods to make asynchronous requests. We will start with the simplest `GET` requests, and work our way up to the more complex usage of AJAX where all variety of control can be exerted.

https://hemanthrajhemu.github.io

Consider for instance the very simple server time page described above. If the URL `currentTime.php` returns a single string and you want to load that value asynchronously into the `<div id="timeDiv">` element, you could write:

```
$("#timeDiv").load("currentTime.php");
```

GET Requests

To illustrate the more powerful features of jQuery and AJAX, consider the more complicated scenario of a web poll where the user must choose one of the four options as illustrated in Figure 15.11.

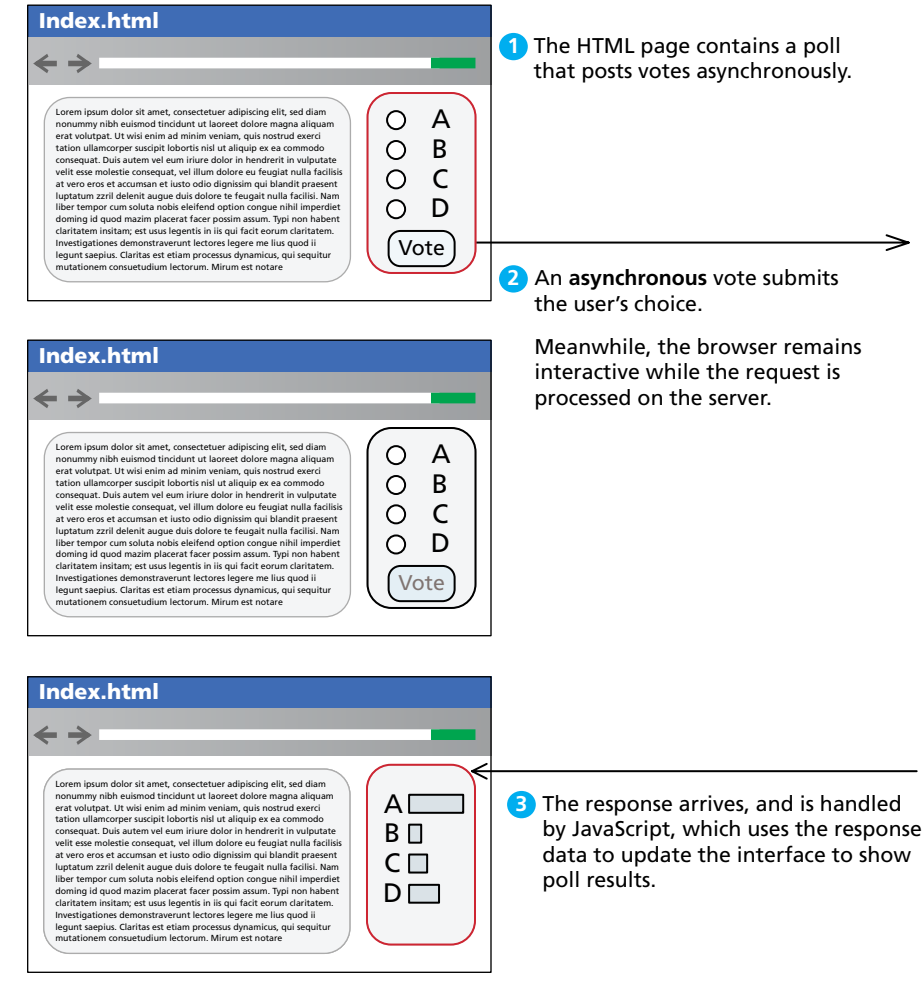


FIGURE 15.11 Illustration of a simple asynchronous web poll

HANDS-ON EXERCISES

LAB 15 EXERCISE
Asynchronous GET

https://hemanthrajhemu.github.io

Making a request to vote for option C in a poll could easily be encoded as a URL request `GET /vote.php?option=C`. However, rather than submit the whole page just to vote in the poll, jQuery's `$.get()` method sends that `GET` request asynchronously as follows:

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

Note that the $ symbol is followed by a dot. Recall that since $ is actually shorthand for `jQuery()`, the above method call is equivalent to

```
jQuery().("/vote.php?option=C");
```

Attaching that function call to the form's submit event allows the form's default behavior to be replaced with an asynchronous `GET` request.

NOTE

Although a `GET` request passes information in the URL, you can split the request into URL and data components by passing the query string as the **data** parameter and let the jQuery engine build the complete URL (with `?` added).

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

can therefore be rewritten as

```
$.get("/vote.php", "option=C");
```

This allows you to easily change between `GET` and `POST` requests for debugging and modularize your request calls.

Although a `get()` method can request a resource very easily, handling the response from the request requires that we revisit the notion of the handler and listener.

The event handlers used in jQuery are no different than those we've seen in JavaScript, except that they are attached to the event triggered by a request completing rather than a mouse move or key press. The formal definition of the `get()` method lists one required parameter `url` and three optional ones: `data`, a callback to a `success()` method, and a `dataType`.

```
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. Callbacks are the programming term

https://hemanthrajhemu.github.io

given to placeholders for functions so that a function can be passed into another function and then called from there (called back). This callback function can take three optional parameters

  - `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. By default jQuery makes an intelligent guess between **xml**, **json**, **script,** or **html**.

In Listing 15.13, the callback function is passed as the second parameter to the `get()` method and uses the `textStatus` parameter to distinguish between a successful post and an error. The `data` parameter contains plain text and is echoed out to the user in an alert. Passing a function as a parameter can be an odd syntax for newcomers to jQuery.

```
$.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

Unfortunately, if the page requested (`vote.php`, in this case) does not exist on the server, then the callback function does not execute at all, so the code announcing an error will never be reached. To address this we can make use of the `jqXHR` object to build a more complete solution.

The jqXHR Object

All of the `$.get()` requests made by jQuery return a jqXHR object to encapsulate the response from the server. In practice that means the `data` being referred to in the callback from Listing 15.13 is actually an object with backward compatibility with `XMLHttpRequest`. The following properties and methods are provided to conform to the `XMLHttpRequest` definition.

HANDS-ON EXERCISES

LAB 15 EXERCISE
jqXHR handling

- `abort()` stops execution and prevents any callback or handlers from receiving the trigger to execute.
- `getResponseHeader()` takes a parameter and gets the current value of that header.

https://hemanthrajhemu.github.io

- `readyState` is an integer from 1 to 4 representing the state of the request. The values include 1: sending, 3: response being processed, and 4: completed.
- `responseXML` and/or `responseText` the main response to the request.
- `setRequestHeader(name, value)` when used before actually instantiating the request allows headers to be changed for the request.
- `status` is the HTTP request status codes described back in Chapter 1. (200 = ok)
- `statusText` is the associated description of the status code.

`jqXHR` objects have methods, `done()`, `fail()`, and `always()`, which allow us to structure our code in a more modular way than the inline callback. Figure 15.12 shows a representation of the various paths a request could take, and which methods are called.

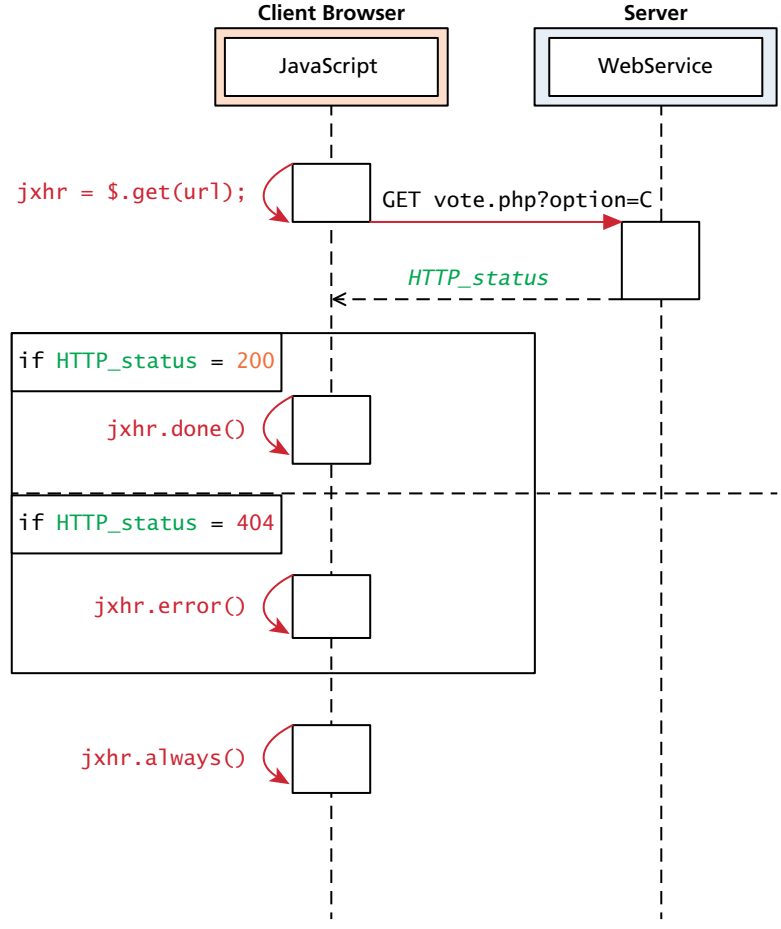


FIGURE 15.12 Sequence diagram depicting how the jqXHR object reacts to different response codes

https://hemanthrajhemu.github.io

By using these methods, the messy and incomplete code from Listing 15.13 becomes the more modular code in Listing 15.14, which also happens to work if the file was missing from the server.

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

jqxhr.done(function(data) { console.log(data);});
jqxhr.fail(function(jqXHR) { console.log("Error: "+jqXHR.status); })
jqxhr.always(function() { console.log("all done"); });
```

LISTING 15.14 Modular jQuery code using the jqXHR object

As we progress with AJAX in jQuery you will see that the jqXHR object is used extensively and that knowledge of it will help you develop more effective, complete code.

NOTE

Code written in versions of jQuery earlier than 1.8 will use methods `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` rather than `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()`.

POST Requests

`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. Finally, with `POST` it is possible to transmit files, something which is not possible with `GET`.

Although the differences between a `GET` and `POST` request are relatively minor, the HTTP 1.1 definition describes `GET` as a **safe method** meaning that they should not change anything, and should only read data. `POST`s on the other hand are not safe, and should be used whenever we are changing the state of our system (like casting a vote). Although Listing 15.13 used a `GET` request to send our vote, it really should have used a `POST` to adhere to the notion of *safe* `GET` requests.

jQuery handles `POST` almost as easily as `GET`, with the need for an added field to hold our data. The formal definition of a jQuery `post()` request is identical to the `get()` request, aside from the method name.

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

https://hemanthrajhemu.github.io

The main difference between a POST and a GET http request is where the data is transmitted. The data parameter, if present in the function call, will be put into the body of the request. Interestingly, it can be passed as a string (with each name=value pair separated with a "&" character) like a GET request or as a Plain Object, as with the get() method.

If we were to convert our vote casting code from Listing 15.14 to a POST request, it would simply change the first line from

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

to

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

Since jQuery can be used to submit a form, you may be interested in the shortcut method serialize(), which can be called on any form object to return its current key-value pairing as an & separated string, suitable for use with post().

Consider our simple vote-casting example. Since the poll's form has a single field, it's easy to understand the ease of creating a short query string on the fly. However, as forms increase in size this becomes more difficult, which is why jQuery includes a helper function to serialize an entire form in one step. The serialize() method can be called on a DOM form element as follows:

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

With the form's data now encoded into a query string (in the postData variable), you can transmit that data through an asynchronous POST using the $.post() method as follows:

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

NOTE

You may have noticed that both $.get() and $.post() methods perform asynchronous transmission. This default behavior in jQuery makes your code more succinct (so long as you want asynchronous transmission). To transit synchronously, you must use the $.ajax() method.

15.3.2 Complete Control over AJAX

It turns out both the $.get() and $.post() methods are actually shorthand forms for the jQuery().ajax() method, which allows fine-grained control over HTTP

https://hemanthrajhemu.github.io

requests. This method allows us to control many more aspects of our asynchronous JavaScript requests including the modification of headers and use of cache controls.

The `ajax()` method has two versions. In the first it takes two parameters: a URL and a Plain Object (also known as an object literal), containing any of over 30 fields. A second version with only one parameter is more commonly used, where the URL is but one of the key-value pairs in the Plain Object. The one line required to post our form using `get()` becomes the more verbose code in Listing 15.15.

```
$.ajax({ url: "vote.php",
         data: $("#voteForm").serialize(),
         async: true,
         type: post
});
```

LISTING 15.15 A raw AJAX method code to make a post

A complete listing of the 33 options available to you would require a chapter in itself. Some of the more interesting things you can do are send login credentials with the username and password fields. You can also modify headers using the header field, which brings us full circle to the HTTP protocol first explored in Chapter 1.

HANDS-ON EXERCISES

LAB 15 EXERCISE
Serialize forms

To pass HTTP headers to the `ajax()` method, you enclose as many as you would like in a Plain Object. To illustrate how you could override `User-Agent` and `Referer` headers in the `POST`, see Listing 15.16.

```
$.ajax({ url: "vote.php",
  data: $("#voteForm").serialize(),
  async: true,
  type: post,
  headers: {"User-Agent" : "Homebrew JavaScript Vote Engine agent",
            "Referer": "http://funwebdev.com"
           }
});
```

LISTING 15.16 Adding headers to an AJAX post in jQuery

15.3.3 Cross-Origin Resource Sharing (CORS)

As you will see when we get to Chapter 16 on security, cross-origin resource sharing (also known as cross-origin scripting) is a way by which some malicious software can gain access to the content of other web pages you are surfing despite the scripts being hosted on another domain. Since modern browsers prevent cross-origin requests by default (which is good for security), sharing content legitimately between two domains

https://hemanthrajhemu.github.io

becomes harder. By default, JavaScript requests for images on images.funwebdev.com from the domain www.funwebdev.com will result in denied requests because subdomains are considered different origins.

Cross-origin resource sharing (CORS) uses new headers in the HTML5 standard implemented in most new browsers. If a site wants to allow any domain to access its content through JavaScript, it would add the following header to all of its responses.

```
Access-Control-Allow-Origin: *
```

The browser, seeing the header, permits any cross-origin request to proceed (since * is a wildcard) thus allowing requests that would be denied otherwise (by default).

A better usage is to specify specific domains that are allowed, rather than cast the gates open to each and every domain. In our example the more precise header

```
Access-Control-Allow-Origin: www.funwebdev.com
```

will prevent all cross-site requests, except those originating from www.funwebdev.com, allowing content to be shared between domains as needed.

15.4 Asynchronous File Transmission

Asynchronous file transmission is one of the most powerful tools for modern web applications. In the days of old, transmitting a large file could require your user to wait idly by while the file uploaded, unable to do anything within the web interface. Since file upload speeds are almost always slower than download speeds, these transmissions can take minutes or even hours, destroying the feeling of a "real" application. Unfortunately jQuery alone does not permit asynchronous file uploads! However, using clever tricks and HTML5 additions, you too can use asynchronous file uploads.

For the following examples consider a simple file-uploading HTML form defined in Listing 15.17.

```
<form name="fileUpload" id="fileUpload" enctype="multipart/form-data"
      method="post" action="upload.php">
<input name="images" id="images" type="file" multiple />
<input type="submit" name="submit" value="Upload files!"/>
</form>
```

LISTING 15.17 Simple file upload form

HANDS-ON EXERCISES

LAB 15 EXERCISE
iFrame file upload

15.4.1 Old iframe Workarounds

The original workaround to allow the asynchronous posting of files was to use a hidden <iframe> element to receive the posted files. Given that jQuery still does not

https://hemanthrajhemu.github.io

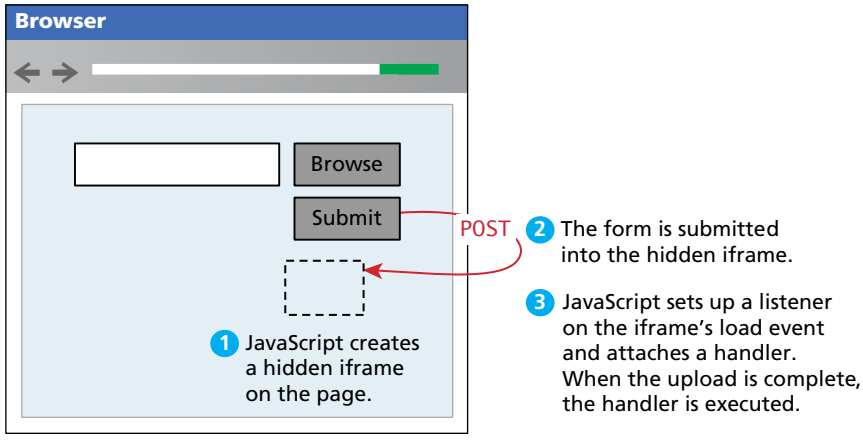


FIGURE 15.13 Illustration of posting to a hidden iframe

natively support the asynchronous uploading of files, this technique persists to this day and may be found in older code you have to maintain. As illustrated in Figure 15.13 and Listing 15.18, a hidden `<iframe>` allows one to post synchronously to another URL in another window. If JavaScript is enabled, you can also hide the upload button and use the change event instead to trigger a file upload. You then use the `<iframe>` element's `onload` event to trigger an action when it is done loading. When the window is done loading, the file has been received and we use the return message to update our interface much like we do with AJAX normally.

```
$(document).ready(function() {
  // set up listener when the file changes
  $(":file").on("change",uploadFile);
  // hide the submit buttons
  $("input[type=submit]").css("display","none");
});

// function called when the file being chosen changes
function uploadFile () {
  // create a hidden iframe
  var hidName = "hiddenIFrame";
  $("#fileUpload").append("<iframe id='"+hidName+"' name='"+hidName+"'
  style='display:none' src='#' ></iframe>");

  // set form's target to iframe
  $("#fileUpload").prop("target",hidName);
  // submit the form, now that an image is in it.
  $("#fileUpload").submit();
```

(*continued*)

https://hemanthrajhemu.github.io

```
    // Now register the load event of the iframe to give feedback
    $('#'+hidName).load(function() {
    var link = $(this).contents().find('body')[0].innerHTML;
    // add an image dynamically to the page from the file just uploaded
    $("#fileUpload").append("<img src='"+link+"' />");
    });
}
```

LISTING 15.18 Hidden iFrame technique to upload files

This technique exploits the fact that browsers treat each `<iframe>` element as a separate window with its own thread. By forcing the post to be handled in another window, we don't lose control of our user interface while the file is uploading.

Although it works, it's a workaround using the fact that every browser can post a file synchronously. A more modular and "pure" technique would be to somehow serialize the data in the file being uploaded with JavaScript and then post it in the body of a post request asynchronously.

Thankfully, the newly redefined XMLHttpRequest Level 2 (XHR2) specification allows us to get access to file data and more through the `FormData` interface[4] so we can post a file as illustrated in Figure 15.14.

15.4.2 The FormData Interface

Using the FormData interface and File API, which is part of HTML5, you no longer have to trick the browser into posting your file data asynchronously. However, you are limited to modern browsers that implement the new specification.

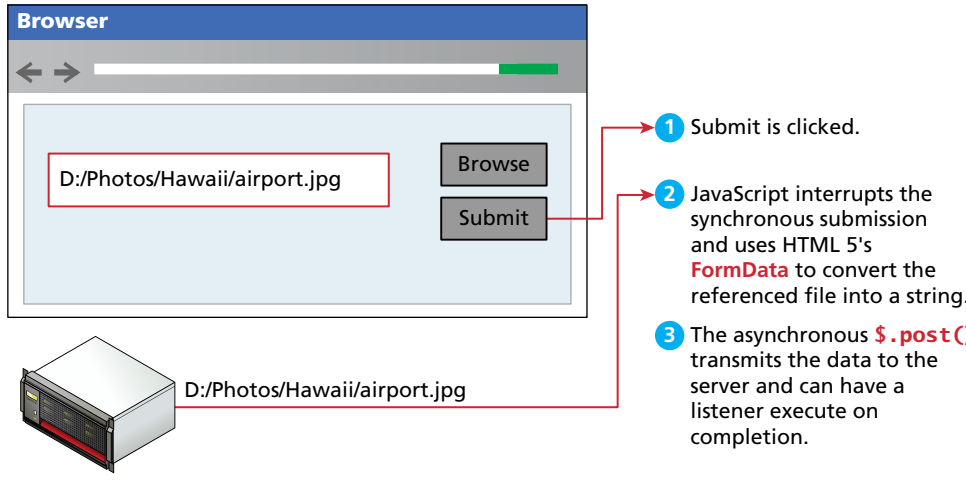


FIGURE 15.14 Posting a file using FormData

https://hemanthrajhemu.github.io

The FormData interface provides a mechanism for JavaScript to read a file from the user's computer (once they choose the file) and encode it for upload. You can use this mechanism to upload a file asynchronously. Intuitively the browser is already able to do this, since it can access file data for transmission in synchronous posts. The FormData interface simply exposes this functionality to the developer, so you can turn a file into a string when you need to.

The <iframe> method uploadFile() from Listing 15.18 can be replaced with the more elegant and straightforward code in Listing 15.19. In this pure AJAX technique the form object is passed to a FormData constructor, which is then used in the call to send() the XHR2 object. This code attaches listeners for various events that may occur.

```
function uploadFile () {
  // get the file as a string
  var formData = new FormData($("#fileUpload")[0]);

  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", transferComplete, false);
  xhr.addEventListener("error", transferFailed, false);
  xhr.addEventListener("abort", transferCanceled, false);

  xhr.open('POST', 'upload.php', true);
  xhr.send(formData);                 // actually send the form data

  function transferComplete(evt) {    // stylized upload complete
     $("#progress").css("width","100%");
     $("#progress").html("100%");
  }

  function transferFailed(evt) {
     alert("An error occurred while transferring the file.");
  }

  function transferCanceled(evt) {
     alert("The transfer has been canceled by the user.");
  }
}
```

LISTING 15.19 Using the new FormData interface from the XHR2 Specification to post files asychronously

While the code in Listing 15.19 works whenever the browser supports the specification, it always posts the entire form.

https://hemanthrajhemu.github.io

15.4.3 Appending Files to a POST

HANDS-ON EXERCISES

LAB 15 EXERCISE
Posting with FormData

When we consider uploading multiple files, you may want to upload a single file, rather than the entire form every time. To support that pattern, you can access a single file and post it by appending the raw file to a `FormData` object as shown in Listing 15.20. The advantage of this technique is that you submit each file to the server asynchronously as the user changes it; and it allows multiple files to be transmitted at once.

```
var xhr = new XMLHttpRequest();
// reference to the 1st file input field
var theFile = $(":file")[0].files[0];
var formData = new FormData();
formData.append('images', theFile);
```

LISTING 15.20 Posting a single file from a form

It should be noted that back in Listing 15.17 the file input is marked as **multiple**, and so, if supported by the browser, the user can select many files to upload at once. To support uploading multiple files in our JavaScript code, we must loop through all the files rather than only hard-code the first one. Listing 15.21 shows a better script than Listing 15.20, since it handles multiple files being selected and uploaded at once.

```
var allFiles =  $(":file")[0].files;
for (var i=0;i<allFiles.length;i++) {
  formData.append('images[]', allFiles[i]);
}
```

LISTING 15.21 Looping through multiple files in a file input and appending the data for posting

The main challenge of asynchronous file upload is that your implementation must consider the range of browsers being used by your users. While the new XHR2 specification and `FormData` interfaces are "pure" and easy to use, they are not widely supported yet across multiple platforms and browsers, making reliance on them bad practice. Conversely the `<iframe>` workaround works well on more browsers, but simply feels inelegant and perhaps not worthy of your support and investment of time.

Recall, from Chapter 6, the principles of *graceful degradation* and *progressive enhancement*. These strategies guide how you design your site and regard JavaScript. How you implement features like asynchronous file upload will depend on the particular strategy you've adopted for your website.

https://hemanthrajhemu.github.io

15.5 Animation

When developers first learn to use jQuery, they are often initially attracted to the easy-to-use animation features. When used appropriately, these animation features can make your web applications appear more professional and engaging.

15.5.1 Animation Shortcuts

By now you've seen how jQuery provides complex (and complete) methods as well as shortcuts. Animation is no different with a raw `animate()` method and many more easy-to-use shortcuts like `fadeIn()/fadeOut()`, `slideUp()/slideDown()`. We introduce jQuery animation using the shortcuts first, then we learn about `animate()` afterward.

One of the common things done in a dynamic web page is to show and hide an element. Modifying the visibility of an element can be done using `css()`, but that causes an element to change instantaneously, which can be visually jarring. To provide a more natural transition from hiding to showing, the `hide()` and `show()` methods allow developers to easily hide elements gradually, rather than through an immediate change.

The `hide()` and `show()` methods can be called with no arguments to perform a default animation. Another version allows two parameters: the duration of the animation (in milliseconds) and a callback method to execute on completion. Using the callback is a great way to chain animations together, or just ensure elements are fully visible before changing their contents.

HANDS-ON EXERCISES

LAB 15 EXERCISE
Simple jQuery animation

Listing 15.22 describes a simple contact form and script that builds and shows a clickable email link when you click the email icon. Hiding an email link is a common way to avoid being targeted by spam bots that search for `mailto:` links in your `<a>` tags.

```
<div class="contact">
  <p>Randy Connolly</p>
  <div class="email">Show email</div>
</div>
<div class="contact">
  <p>Ricardo Hoar</p>
  <div class="email">Show email</div>
</div>
<script type='text/javascript'>
$(".email").click(function() {
     // Build email from 1st letter of first name + lastname
     // @ mtroyal.ca
     var fullName = $(this).prev().html();
     var firstName = fullName.split(" ")[0];
```

(*continued*)

https://hemanthrajhemu.github.io

```
        var address = firstName.charAt(0) + fullName.split(" ")[1] +
                      "@mtroyal.ca";

        $(this).hide();         // hide the clicked icon.
        $(this).html("<a href='mailto:"+address+"'>Mail Us</a>");
        $(this).show(1000);     // slowly show the email address.
    });
    </script>
```

LISTING 15.22 jQuery code to build an email link based on page content and animate its appearance

A visualization of the `show()` method is illustrated in Figure 15.15.[5] Note that both the size and opacity are changing during the animation. Although using the very straightforward `hide()` and `show()` methods works, you should be aware of some more advanced shortcuts that give you more control.

fadeIn()/fadeOut()

The `fadeIn()` and `fadeOut()` shortcut methods control the opacity of an element. The parameters passed are the duration and the callback, just like `hide()` and `show()`. Unlike `hide()` and `show()`, there is no scaling of the element, just strictly control over the transparency. Figure 15.16 shows a span during its animation using `fadeIn()`.

It should be noted that there is another method, `fadeTo()`, that takes two parameters: a duration in milliseconds and the opacity to fade to (between 0 and 1).

slideDown()/slideUp()

The final shortcut methods we will talk about are `slideUp()` and `slideDown()`. These methods do not touch the opacity of an element, but rather gradually change its height. Figure 15.17 shows a `slideDown()` animation using an email icon from http://openiconlibrary.sourceforge.net.

FIGURE 15.15 Illustration of the show() animation using the icon from openiconlibrary.sourceforge.net

FIGURE 15.16 Illustration of a fadeIn() animation

https://hemanthrajhemu.github.io

FIGURE 15.17 Illustration of the slideDown() animation

You should note at this point that `hide()` and `show()` are in fact a combination of both the fade and slide animations. However, different browsers may interpret these animations in slightly different ways.

Toggle Methods

As you may have seen, the shortcut methods come in pairs, which make them ideal for toggling between a shown and hidden state. jQuery has gone ahead and written multiple toggle methods to facilitate exactly that. For instance, to toggle between the visible and hidden states (i.e., between using the `hide()` and `show()` methods), you can use the `toggle()` methods. To toggle between fading in and fading out, use the `fadeToggle()` method; toggling between the two sliding states can be achieved using the `slideToggle()` method.

Using a toggle method means you don't have to check the current state and then conditionally call one of the two methods; the toggle methods handle those aspects of the logic for you.

15.5.2 Raw Animation

Just like `$.get()` and `$.post()` methods are shortcuts for the complete `$.ajax()` method, the animations shown this far are all specific versions of the generic `animate()` method. When you want to do animation that differs from the prepackaged animations, you will need to make use of `animate`.

The `animate()` method has several versions, but the one we will look at has the following form:

```
.animate( properties, options );
```

The `properties` parameter contains a Plain Object with all the CSS styles of the final state of the animation. The `options` parameter contains another Plain Object with any of the options below set.

- `always` is the function to be called when the animation completes or stops with a fail condition. This function will always be called (hence the name).
- `done` is a function to be called when the animation completes.
- `duration` is a number controlling the duration of the animation.
- `fail` is the function called if the animation does not complete.
- `progress` is a function to be called after each step of the animation.

https://hemanthrajhemu.github.io

- `queue` is a Boolean value telling the animation whether to wait in the queue of animations or not. If false, the animation begins immediately.
- `step` is a function you can define that will be called periodically while the animation is still going. It takes two parameters: a `now` element, with the current numerical value of a CSS property, and an `fx` object, which is a temporary object with useful properties like the CSS attribute it represents (called *tween* in jQuery). See Listing 15.23 for example usage to do rotation.
- Advanced options called easing and `specialEasing` allow for advanced control over the speed of animation.

Movement rarely occurs in a linear fashion in nature. A ball thrown in the air slows down as it reaches the apex then accelerates toward the ground. In web development, **easing functions** are used to simulate that natural type of movement. They are mathematical equations that describe how fast or slow the transitions occur at various points during the animation.

Included in jQuery are `linear` and `swing` easing functions. Linear is a straight line and so animation occurs at the same rate throughout while swing starts slowly and ends slowly. Figure 15.18 shows graphs for both the `linear` and `swing` easing functions.

Easing functions are just mathematical definitions. For example, the function defining `swing` for values of time t between 0 and 1 is

$$\text{swing}(t) = -\frac{1}{2}\cos(t\pi) + 0.5$$

The jQuery UI extension provides over 30 easing functions, including cubic functions and bouncing effects, so you should not have to define your own.

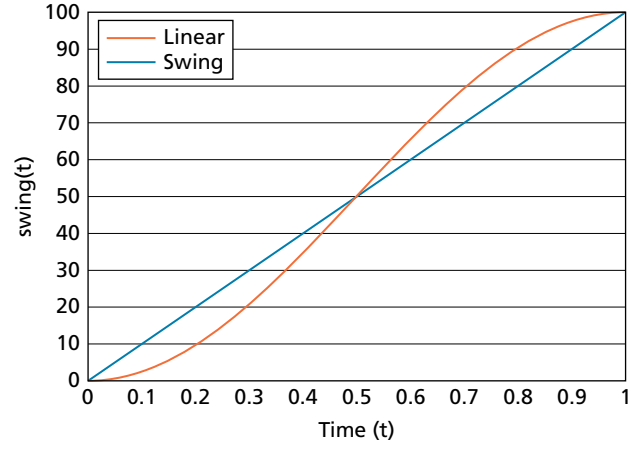


FIGURE 15.18 Visualization of the linear and swing easing functions

https://hemanthrajhemu.github.io

FIGURE 15.19 Illustration of rotation using the animate() method in Listing 15.23

An example usage of `animate()` is shown in Listing 15.23 where we apply several transformations (changes in CSS properties), including one for the text size, opacity, and a CSS3 style rotation, resulting in the animation illustrated in Figure 15.19.

HANDS-ON EXERCISES

LAB 15 EXERCISE
Easing functions

It should be noted that `step()` callbacks are only made for CSS values that are numerical. This is why you often see a dummy CSS value used to control an unrelated CSS option like rotation (which has string values, not numeric values). In Listing 15.23 we add a `margin-right` CSS attribute with a value of 100 so that whenever the callback for that CSS property occurs, we can figure what percentage of the animation we are on by dividing `now/100`. We then use that percentage to apply the appropriate rotation (×360). If we had added the `transform` elements as CSS attributes, no automatic values would be calculated, no animated rotation would occur. Figure 15.20 illustrates the step function callbacks for our example with two calls to step functions shown for illustrative purposes. The actual number of calls to step will depend on your hardware and software configuration.

```
$(this).animate(
  // parameter one: Plain Object with CSS options.

  {opacity:"show","fontSize":"120%","marginRight":"100px"},
  // parameter 2: Plain Object with other options including a
  // step function
  {step: function(now, fx) {
        // if the method was called for the margin property
        if (fx.prop=="marginRight") {
           var angle=(now/100)*360; //percentage of a full circle
           // Multiple rotation methods to work in multiple browsers
           $(this).css("transform","rotate("+angle+"deg)");
           $(this).css("-webkit-transform","rotate("+angle+"deg)");
           $(this).css("-ms-transform","rotate("+angle+"deg)");
        }
     },
     duration:5000, "easing":"linear"
  }
);
```

LISTING 15.23 Use of animate() with a step function to do CSS3 rotation

https://hemanthrajhemu.github.io

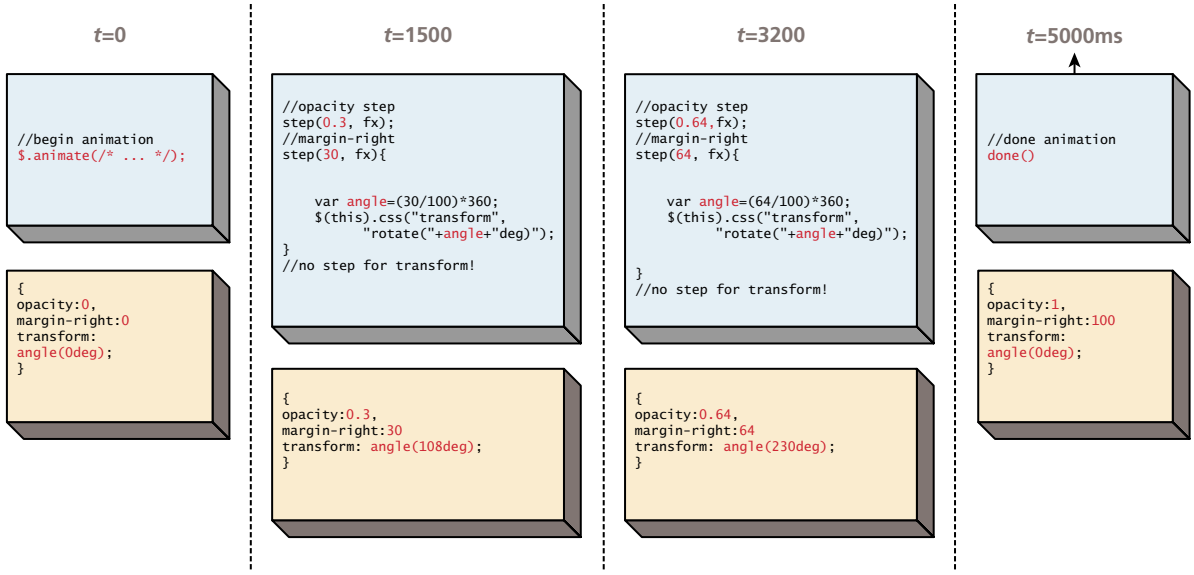


FIGURE 15.20 Illustration of an animation with step calls for numeric CSS properties over time *t*

You may ask, why use a `margin-right` property instead of the opacity that already goes from 0 to 1? The answer is that by attaching the rotation animation to another CSS property you care about, the two become coupled, so that modifying the opacity would modify the rotation. Decoupling the rotation from the opacity ensures both are independently controlled.

15.6 Backbone MVC Frameworks

HANDS-ON EXERCISES

LAB 15 EXERCISE
Backbone Walkthrough

In working with jQuery thus far we have seen how easily we can make great animations and modularize our UI into asynchronous components. As our model gets more and more complex, it becomes challenging to keep our data decoupled from our view and the DOM. We end up with many listeners, callbacks, and server-side scripts, and it can be challenging to coordinate all the different parts together. In response, frameworks have been created that enforce a strict MVC pattern, and if used correctly can speed up development and result in maintainable modular code.

MVC frameworks (and frameworks in general) are not silver bullets that solve all your development challenges. These frameworks are overkill for small applications, where a small amount of jQuery would suffice. You will learn about the basics of Backbone so that you can consider this library or one like it before designing a large-scale web application.

https://hemanthrajhemu.github.io

15.6.1 Getting Started with Backbone.js

Backbone is an MVC framework that further abstracts JavaScript with libraries intended to adhere more closely to the MVC model as described in Chapter 14. This library is available from http://backbonejs.org and relies on the underscore library, available from http://underscorejs.org/.

In Backbone, you build your client scripts around the concept of **models**. These models are often related to rows in the site's database and can be loaded, updated, and eventually saved back to the database using a REST interface, described in Chapter 17. Rather than writing the code to connect listeners and event handlers, Backbone allows user interface components to be notified of changes so they can update themselves just by setting everything up correctly.

You must download the source for these libraries to your server, and reference them just as we've done with jQuery. Remember that the underscore library is also required, so a basic inclusion will look like:

```
<script src="underscore-min.js"></script>
<script src="backbone-min.js"></script>
```

To illustrate the application of Backbone, consider our travel website discussed throughout earlier chapters. In particular consider the management of albums, and an interface to select and publish particular albums.

The HTML shown in Listing 15.24 will serve as the basis for the example, with a form to create new albums and an unordered list to display them.

```
<form id="publishAlbums" method="post" action="publish.php">
  <h1>Publish Albums</h1>
  <ul id="albums">
       <!-- The albums will appear here -->
  </ul>
  <p id="totalAlbums">Count: <span>0</span></p>
  <input type="submit" id="publish" value="Publish" />
</form>
```

LISTING 15.24 HTML for an album publishing interface

The MVC pattern in Backbone will use a `Model` object to represent the `TravelAlbum`, a `Collection` object to manage multiple albums, and a `View` to render the HTML for the model, and instantiate and render the entire application as illustrated in the class diagram in Figure 15.21.

15.6.2 Backbone Models

The term models can be a challenging one to apply, since authors of several frameworks and software engineering patterns already use the term.

https://hemanthrajhemu.github.io

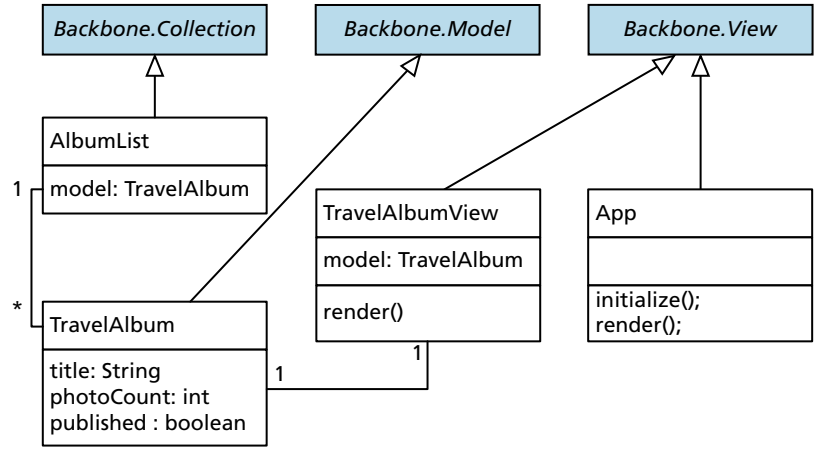


FIGURE 15.21 Illustration of Backbone Model, Collections, and Views for a Photo Album example

Backbone.js defines **models** as

> *the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control.*

When using Backbone, you therefore begin by abstracting the elements you want to create models for. In our case, `TravelAlbum` will consist of a `title`, an image `photoCount`, and a Boolean value controlling whether it is published or not. The `Models` you define using Backbone must *extend* `Backbone.Model`, adding methods in the process as shown in Listing 15.25.

```
// Create a model for the albums
var TravelAlbum = Backbone.Model.extend({
  defaults:{
           title: 'NewAlbum',
           photoCount: 0,
           published: false
      },

      // Function to publish/unpublish
      toggle: function(){
          this.set('checked', !this.get('checked'));
      }
  });
```

LISTING 15.25 A PhotoAlbum Model extending from Backbone.Model

https://hemanthrajhemu.github.io

15.6.3 Collections

In addition to models, Backbone introduces the concept of Collections, which are normally used to contain lists of `Model` objects. These collections have advanced features and like a database can have indexes to improve search performance. In Listing 15.26, a collection of `Albums`, `AlbumList`, is defined by extending from Backbone's `Collection` object. In addition an initial list of `TravelAlbums`, named `albums`, is instantiated to illustrate the creation of some model objects inside a `Collection`.

```
// Create a collection of albums
var AlbumList = Backbone.Collection.extend({

  // Set the model type for objects in this Collection
  model: TravelAlbum,

  // Return an array only with the published albums
  GetChecked: function(){
     return this.where({checked:true});
  }
});

// Prefill the collection with some albums.
var albums = new AlbumList([
  new TravelAlbum({ title: 'Banff, Canada', photoCount: 42}),
  new TravelAlbum({ title: 'Santorini, Greece', photoCount: 102}),
]);
```

LISTING 15.26 Demonstration of a Backbone.js Collection defined to hold PhotoAlbums

Although we now have the capacity to model albums and a collection of albums, we still have not rendered anything to the user! To facilitate this, Backbone adheres closely to a pure MVC pattern, and requires that you define a `View` for any DOM element you want to be auto-refreshed on certain occurrences.

15.6.4 Views

Views allow you to translate your models into the HTML that is seen by the users. They attach themselves to methods and properties of the `Collection` and define methods that will be called whenever Backbone determines the view needs refreshing.

For our example we extend a `View` as shown in Listing 15.27. In that code we attach our view to a particular `tagName` (in our case the `<li>` element) and then

https://hemanthrajhemu.github.io

```
var TravelAlbumView = Backbone.View.extend({
  TagName: 'li',

  events:{
     'click': 'toggleAlbum'
  },

  initialize: function(){
      // Set up event listeners attached to change
     this.listenTo(this.model, 'change', this.render);
  },

  render: function(){
  // Create the HTML
  this.$el.html('<input type="checkbox" value="1" name="' +
                this.model.get('title') + '" /> ' +
                this.model.get('title') + '<span> ' +
                this.model.get('photoCount') + ' images</span>');
  this.$('input').prop('checked', this.model.get('checked'));

  // Returning the object is a good practice
     return this;
  },

  toggleAlbum: function() {
     this.model.toggle();
  }
});
```

LISTING 15.27 Deriving custom View objects for our model and Collection

associate the click event with a new method named `toggleAlbum()`. You must always override the `render()` method since it defines the HTML that is output.

Finally, to make this code work you must also override the render of the main application. In our case we will base it initially on the entire `<body>` tag, and output our content based entirely on the models in our collection as shown in Listing 15.28.

Notice that you are making use of several methods, `_.each()`, `elem.get()`, and `this.total.text()`, that have not yet been defined. Some of these methods replace jQuery functionality, while others have no analog in that framework. The `_` is defined by the underscore library in much the same way as `$` is defined for jQuery. If you are interested in learning more about Backbone, a complete listing of functions is available online.[6]

These models also allow us to save data temporarily on the client's machine as JavaScript and then post it back to the server when needed. You can leverage the

https://hemanthrajhemu.github.io

```
// The main view of the entire Backbone application
var App = Backbone.View.extend({
  // Base the view on an existing element
  el: $('body'),

  initialize: function() {
     // Define required selectors
     this.total = $('#totalAlbums span');
     this.list = $('#albums');

     // Listen for the change event on the collection.
     this.listenTo(albums, 'change', this. render);

     // Create views for every one of the albums in the collection
     albums.each(function(album) {
        var view = new TravelAlbumView({ model: album });
              this.list.append(view.render().el);
        }, this);   // "this" is the context in the callback
     },

     render: function(){

         // Calculate the count of published albums and photos
            var total = 0; var photos = 0;

           _.each(albums.getChecked(), function(elem) {
              total++;
              photos+= elem.get("photoCount");
           });

        // Update the total price
        this.total.text(total+' Albums ('+photos+' images)');
        return this;
     }
});

new App(); // create the main app
```

LISTING 15.28 Defining the main app's view and making use of the Collections and models defined earlier

`jQuery().ajax()` method already described since Backbone is designed to be used alongside jQuery. With Backbone's structured models and collections coupled with jQuery's visual flourishes and helper function, you have all the tools to build advanced client-side scripts.

https://hemanthrajhemu.github.io

15.7 Chapter Summary

This chapter introduced the advanced JavaScript concept of prototypes, which are used extensively by several frameworks. The jQuery framework was introduced with its filters, selectors, and listeners illustrated to show how to use the powerful framework. The concepts of content delivery network and cross-site resource sharing were covered in the context of AJAX design. jQuery animation was described along with easing functions that control the rate of animation. Finally, the Backbone.js framework was described, which provides structural support for your client-side scripts.

15.7.1 Key Terms

animation
attribute selectors
collections
content delivery network (CDN)
content filters
contextual selectors
cross-origin resource sharing (CORS)
easing function
FormData
handler
jQuery
jqXHR
listener
models
object literals
prototypes
pseudo-classes
pseudo-element selectors
safe method

15.7.2 Review Questions

1. Why are prototypes more efficient than other techniques for creating classes in JavaScript?
2. How can we add methods to existing classes, like `String` or `array`?
3. What does `$()` shorthand stand for in jQuery?
4. Write a jQuery selector to get all the `<p>` that contain the word "hello."
5. jQuery extends the CSS syntax for selectors. Explain what that means.
6. How can we ensure jQuery loads, even if the CDN is down?
7. How would you change the text color of all the `<a>` tags in jQuery (one line)?
8. What is the difference between the `append()` and `appendTo()` methods?
9. What are the advantages of using asynchronous requests over traditional synchronous ones?
10. What are the two techniques for AJAX file upload?
11. What are the commonly used animations in jQuery?
12. What is the base method on which all jQuery animations rely?
13. What do MVC frameworks accomplish?

https://hemanthrajhemu.github.io

15.7.3 Hands-On Practice

PROJECT 1: Share Your Travel Photos

DIFFICULTY LEVEL: Easy

HANDS-ON EXERCISES
PROJECT 15.1

Overview

Use jQuery to submit a form asynchronously. In this case we will continue building on the image upload form for our travel site.

Instructions

1. Open lab15-project01.html in the editor of your choice, so you can start making changes.
2. Import jQuery in the `<head>` of the page.
3. Import your own script lab15-project01.js. In that file define a listener to listen to the form's submit event.
4. Use `preventDefault()` to stop the event from submitting synchronously. Now build the `$.post()` request manually using one of the techniques covered in this chapter.
5. Set up the request to trigger a small user interface change on success, to indicate to the user that the image upload was successful.

Testing

1. To test, select an image and try to submit the form. Using Firebug (or equivalent in Chrome), track that the post is indeed being transmitted, and the page does not get stuck in a refreshing mode. You should see a confirmation when the image is done uploading.
2. For an intermediate challenge, handle multiple image uploads using the `FormData` interface.

PROJECT 2: Any Web Page

DIFFICULTY LEVEL: Intermediate

HANDS-ON EXERCISES
PROJECT 15.2

Overview

Designing user interfaces is by no means easy, nor are best practices static. To assist in getting data about how users interact with an interface, you will create a script that captures all the movements, which can be used to show a focus map of where the user interacts and clicks. A script like this could be used to evaluate alternate designs, or to test if a certain area was being noticed by users.

Instructions

1. Open lab15-project02.html in the editor of your choice, so you can start making changes.

https://hemanthrajhemu.github.io

2. Attach a listener to every DOM element. Each element will capture events for mouse in, mouse out, and mouse clicks and transmit them to the script provided in lab15-processData.php, which writes to a database defined in lab15-data.sql.
3. Using the data collected in the database, create another script that colors the DOM elements based on how visible/popular they were as shown in Figure 15.22 on an early prototype for the Art store. Hint: You will need the total time on the site, and the time spent on each element to get a relative weighting.

Testing

1. To test, integrate a GET parameter into your page, which switches between the collection mode and the display mode.
2. Reset the database and surf to the page with the data capture script enabled. Focus your mouse movements and clicks on one area of the page
3. Stop data collection and visit the second version of the page, which should show a concentration of focus in that area of the page.

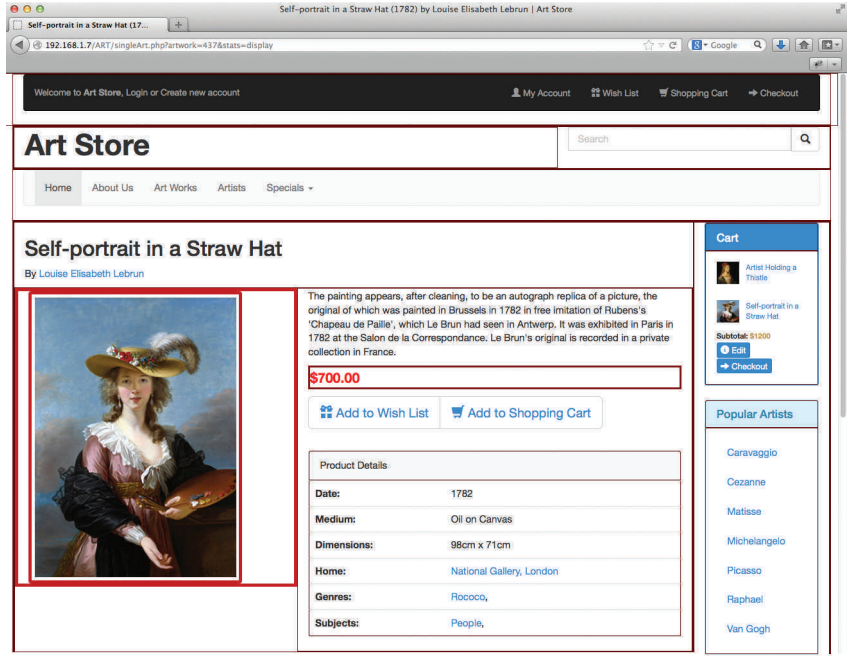


FIGURE 15.22 Areas of user focus around the picture and price are highlighted with thicker red borders.

https://hemanthrajhemu.github.io

PROJECT 3: Share Your Travel Photos

DIFFICULTY LEVEL: Advanced

Overview

This project will build your own gallery using jQuery for our travel photo sharing site as shown in Figure 15.23.

HANDS-ON EXERCISES
PROJECT 15.3

Instructions

1. This project builds on that from Project 11.2 where you defined a gallery for your images (browse-images.php).
2. Modify your PHP script so that each `<img>` uses the class `galleryImage`.
3. Inside the framework include jQuery from the CDN and include your own jQuery file as well to hold your code for this project.
4. Write code to create extra `<div>` elements: two for navigation and three to hold the images in the gallery. Give the image `<div>` elements a class name `galleryPic`, and the navigation divs unique IDs. Style them appropriately.

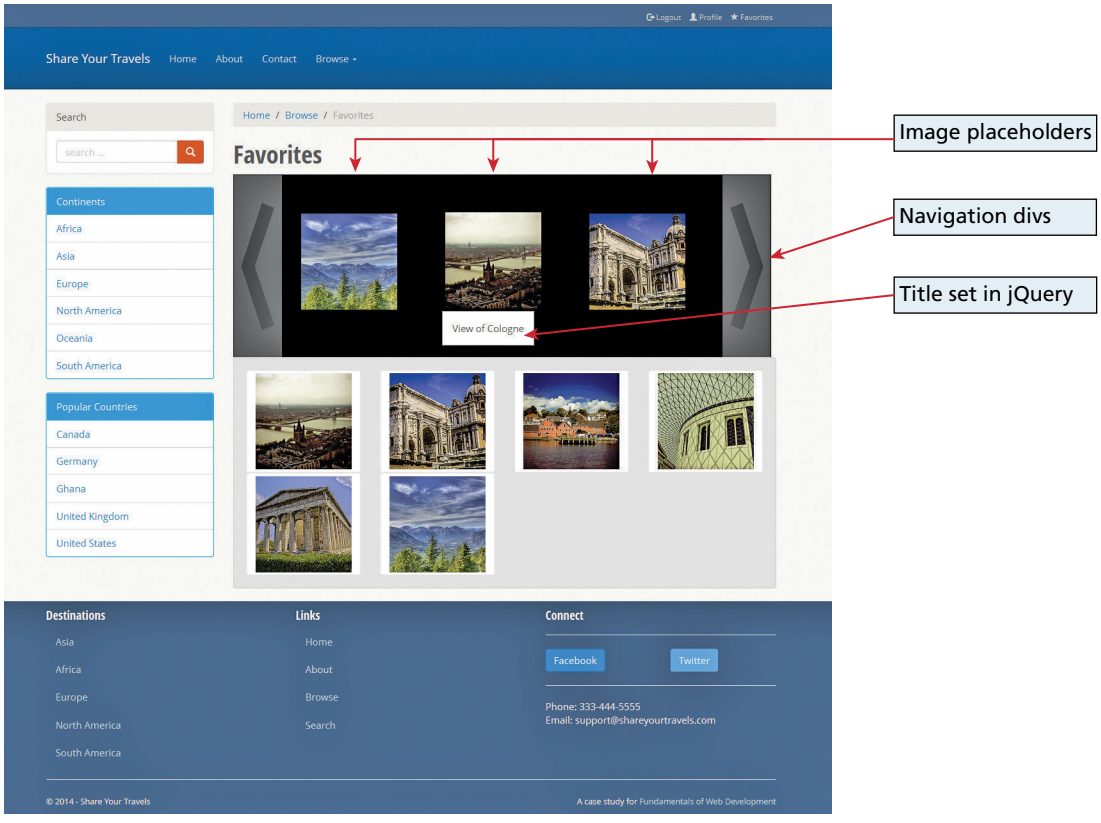


FIGURE 15.23 Screenshot of the image gallery

https://hemanthrajhemu.github.io

5. Upon the page loading, create a script that selects using jQuery all the `<img>` elements with class `galleryImage`. Using the `title` and `src` details, the newly created `<div>` elements can have images and titles loaded.
6. Add a listener so that when the navigation `<div>` elements are clicked, you respond by animating all the images moving left or right. Upon completion of the animation, change the images and reset their location (left, middle, and right).

Testing

1. Cycle all the way through the images in both directions.
2. If interested, modify the script and see if you can easily build a gallery on the Art store using the same code.

15.7.4 References

1. J. Resig, "Selectors in JavaScript," August 2005. [Online]. http://ejohn.org/blog/selectors-in-javascript/.
2. jQuery Foundation, "jQuery API Documentation." [Online]. http://api.jquery.com/.
3. jQuery, "Dom Insertion, Around." [Online]. http://api.jquery.com/category/manipulation/dom-insertion-around/.
4. W3C, "XMLHttpRequest." [Online]. https://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html.
5. Mail icon, mail-mark-unread-new.png, http://openiconlibrary.sourceforge.net/gallery2/open_icon_library-full/icons/png/256x256/actions/mail-mark-unread-new.png.
6. backbone.js. [Online]. http://backbonejs.org.

https://hemanthrajhemu.github.io

17 XML Processing and Web Services

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- What XML is and what role it plays in software systems
- How to process an XML file in JavaScript and PHP
- What the JSON data form is and how to process it in JavaScript and PHP
- About web services and their role in web development
- How to consume web services in JavaScript and PHP
- How to create web services in PHP

This chapter covers XML processing along with one of the most common uses of XML in the web context: the consumption and creation of web services. The chapter begins by describing the XML data interchange format, as well as techniques for creating XML files and processing them in PHP. It also covers JSON, which is another data interchange format that is commonly used in web applications. The chapter then moves on to web services and how they facilitate data exchange and asynchronous applications. The chapter provides guidance along with sample code for consuming as well as creating XML and JSON web services.

https://hemanthrajhemu.github.io

17.1 XML Overview

Back in Chapter 2, you learned that like HTML, XML is a markup language, but unlike HTML, XML can be used to mark up any type of data. XML is used not only for web development but is also used as a file format in many nonweb applications. One of the key 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 and understandable as well. Back in Chapter 7, you also encountered XML in the SVG (Scalable Vector Graphics) file format. XML is also used in the web context as a format for moving information between different systems. As can be seen in Figure 17.1, XML is not only used on the web server and to communicate asynchronously with the browser, but is also used as a data interchange format for moving information between systems (in this diagram, with a knowledge management system and a finance system).

17.1.1 Well-Formed XML

For a document to be well-formed XML, it must follow the syntax rules for XML.[1] These rules are quite straightforward:

- 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.

Listing 17.1 illustrates a sample XML document. Notice that it begins with an XML declaration, which is analogous to the `DOCTYPE` of an HTML document. In this example, the root element is called `<art>`.

Some type of XML parser is required to verify that an XML document is well formed. A parser not only checks the document for syntax errors; it also typically converts the XML document into some type of internal memory structure. All contemporary browsers have built-in parsers, as do most web development environments such as PHP and ASP.NET.

https://hemanthrajhemu.github.io

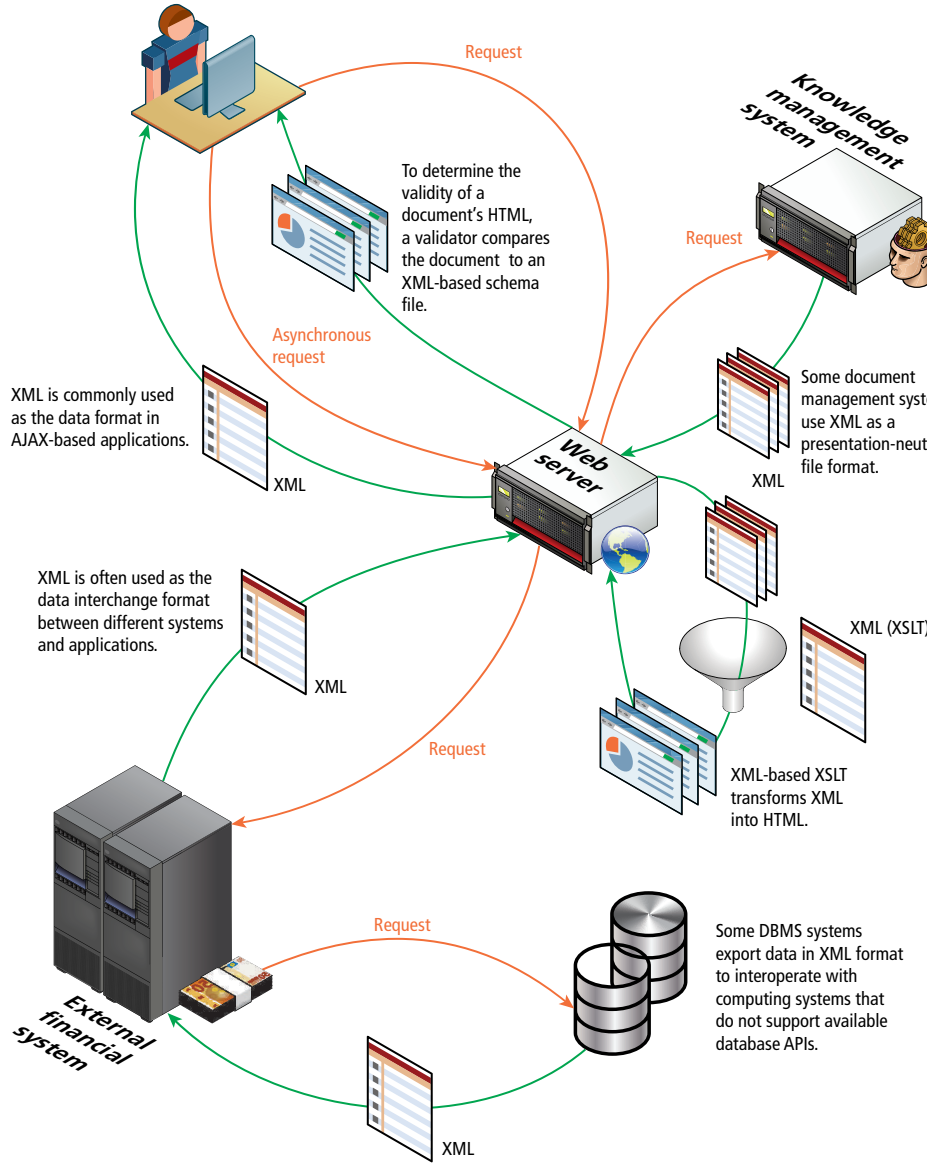


FIGURE 17.1 XML in the web context

17.1.2 Valid XML

A valid XML document is one that is well formed and whose element and content conform to the rules of either its document type definition (DTD) or its schema.[2] DTDs were the original way for an XML parser to check an XML document for

https://hemanthrajhemu.github.io

```
<?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>
```

LISTING 17.1 Sample XML document

validity. They tell the XML parser which elements and attributes to expect in the document as well as the order and nesting of those elements. A DTD can be defined within an XML document or within an external file. Listing 17.2 contains the DTD for the XML file from Listing 17.1.

The main drawback with DTDs is that they can only validate the existence and ordering of elements (and the existence of attributes). They provide no way to validate the values of attributes or the textual content of elements. For this type of validation, one must instead use XML schemas, which have the added advantage of using XML syntax. Unfortunately, schemas have the corresponding disadvantage of being long-winded and harder for humans to read and comprehend; for this reason, they are typically created with tools. An explanation of XML schemas and DTDs is considerably beyond the scope of this book. Listing 17.3 illustrates a sample XML schema for the XML document in Listing 17.1.

https://hemanthrajhemu.github.io

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE art [
<!ELEMENT art (painting*)>
<!ELEMENT painting (title,artist,year,medium)>
<!ATTLIST painting id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (name,nationality)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT medium (#PCDATA)>
]>
<art>
...
</art>
```

LISTING 17.2 Example DTD

```
<xs:schema attributeFormDefault="unqualified"
     elementFormDefault="qualified"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="art">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="painting" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="title"/>
              <xs:element name="artist">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="name"/>
                    <xs:element type="xs:string" name="nationality"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element type="xs:short" name="year" />
              <xs:element type="xs:string" name="medium"/>
            </xs:sequence>
            <xs:attribute type="xs:short" name="id" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

LISTING 17.3 Example schema

https://hemanthrajhemu.github.io

17.1.3 XSLT

There are two other XML technologies that are sometimes used in a web context. The first of these is XSLT, which stands for XML Stylesheet Transformations.[3] XSLT is an XML-based programming language that is used for transforming XML into other document formats, as shown in Figure 17.2.

Perhaps the most common translation is the conversion of XML to HTML. All of the modern browsers support XSLT, though XSLT is also used on the server side and within JavaScript, as shown in Figure 17.3.

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Using XSLT

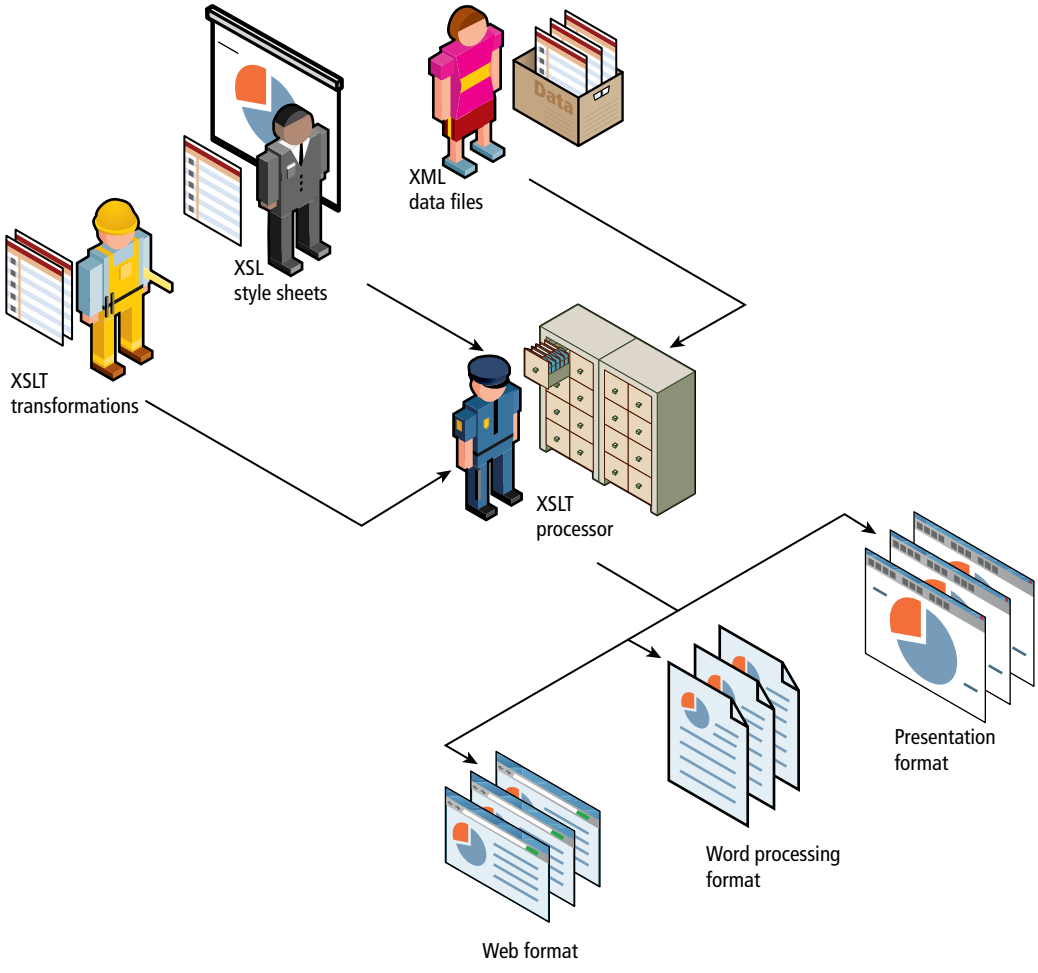


FIGURE 17.2 XSLT workflow

https://hemanthrajhemu.github.io

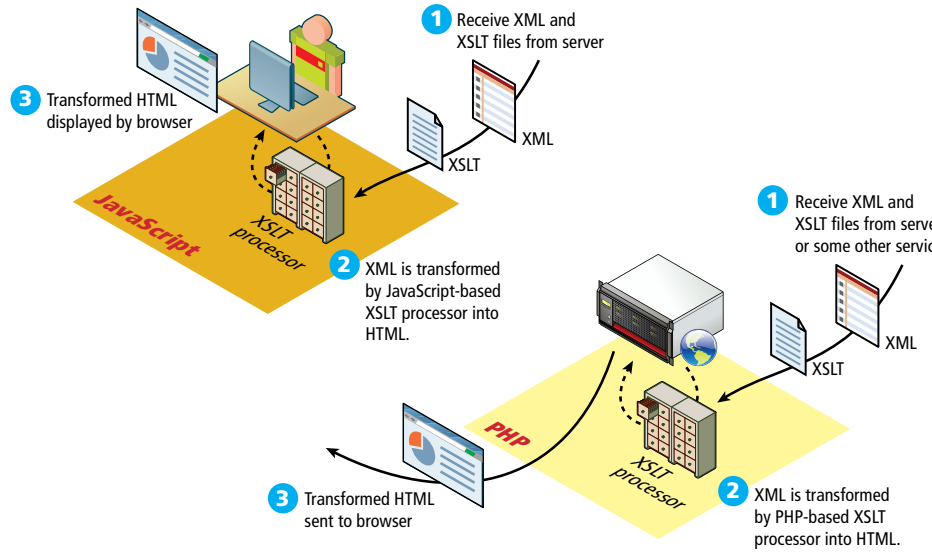


FIGURE 17.3 Usage of XSLT

Listing 17.4 shows an example XSLT document that would convert the XML shown in Listing 17.1 into an HTML list. Notice the strings within the `select` attribute: these are XPath expressions, which are used for selecting specific elements

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/1999/xhtml">
<body>
   <h1>Catalog</h1>
   <ul>
    <xsl:for-each select="/art/painting">
      <li>
        <h2><xsl:value-of select="title"/></h2>
        <p>By: <xsl:value-of select="artist/name"/><br/>
           Year: <xsl:value-of select="year"/>
           [<xsl:value-of select="medium"/>]</p>
      </li>
    </xsl:for-each>
   </ul>
</body>
</html>
```

LISTING 17.4 An example XSLT document

https://hemanthrajhemu.github.io

within the XML source document. The `<xsl:for-each>` element is one of the iteration constructs within XSLT. In this example, it iterates through each of the `<painting>` elements.

An XML parser is still needed to perform the actual transformation. The result of the transformation is shown in Figure 17.4. It is beyond the scope of this book to cover the details of the XSLT programming language.

17.1.4 XPath

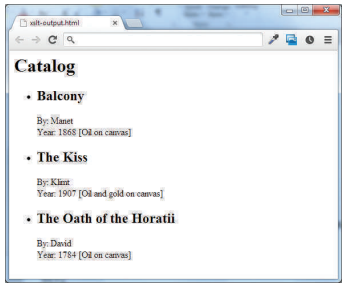
The other commonly used XML technology in the web context is XPath, which is a standardized syntax for searching an XML document and for navigating to elements within the XML document.[4] XPath is typically used as part of the programmatic manipulation of an XML document in PHP and other languages.

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Using XPath

XPath uses a syntax that is similar to the one used in most operating systems to access directories. For instance, to select all the `painting` elements in the XML file in Listing 17.1, you would use the XPath expression: `/art/painting`. Just as with operating system paths, the forward slash is used to separate elements contained within other elements; as well, an XPath expression beginning with a forward slash is an absolute path beginning with the start of the document.

In XPath terminology, an XPath expression returns zero, one, or many XML nodes. In XPath, a node generally refers to an XML element. From a node, you can




```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<h1>Catalog</h1>
<ul>
  <li>
    <h2>Balcony</h2>
    <p>By: Manet<br/>
    Year: 1868 [Oil on canvas]</p>
  </li>
  <li>
    <h2>The Kiss</h2>
    <p>By: Klimt<br/>
    Year: 1907 [Oil and gold on canvas]</p>
  </li>
  <li>
    <h2>The Oath of the Horatii</h2>
    <p>By: David<br/>Year: 1784 [Oil on canvas]</p>
  </li>
</ul>
</body>
</html>
```

FIGURE 17.4 Result of XSLT

https://hemanthrajhemu.github.io

examine and extract its attributes, textual content, and child nodes. XPath also comes with a sophisticated vocabulary for specifying search criteria. For instance, let us examine the following XPath expression:

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

It selects the <name> element within the <artist> element for the <painting> element with an id attribute of 192, as shown in Figure 17.5 (which also illustrates several additional XPath expressions). As can be seen in the figure, square brackets are used to specify a criteria expression at the current path node, which in the above

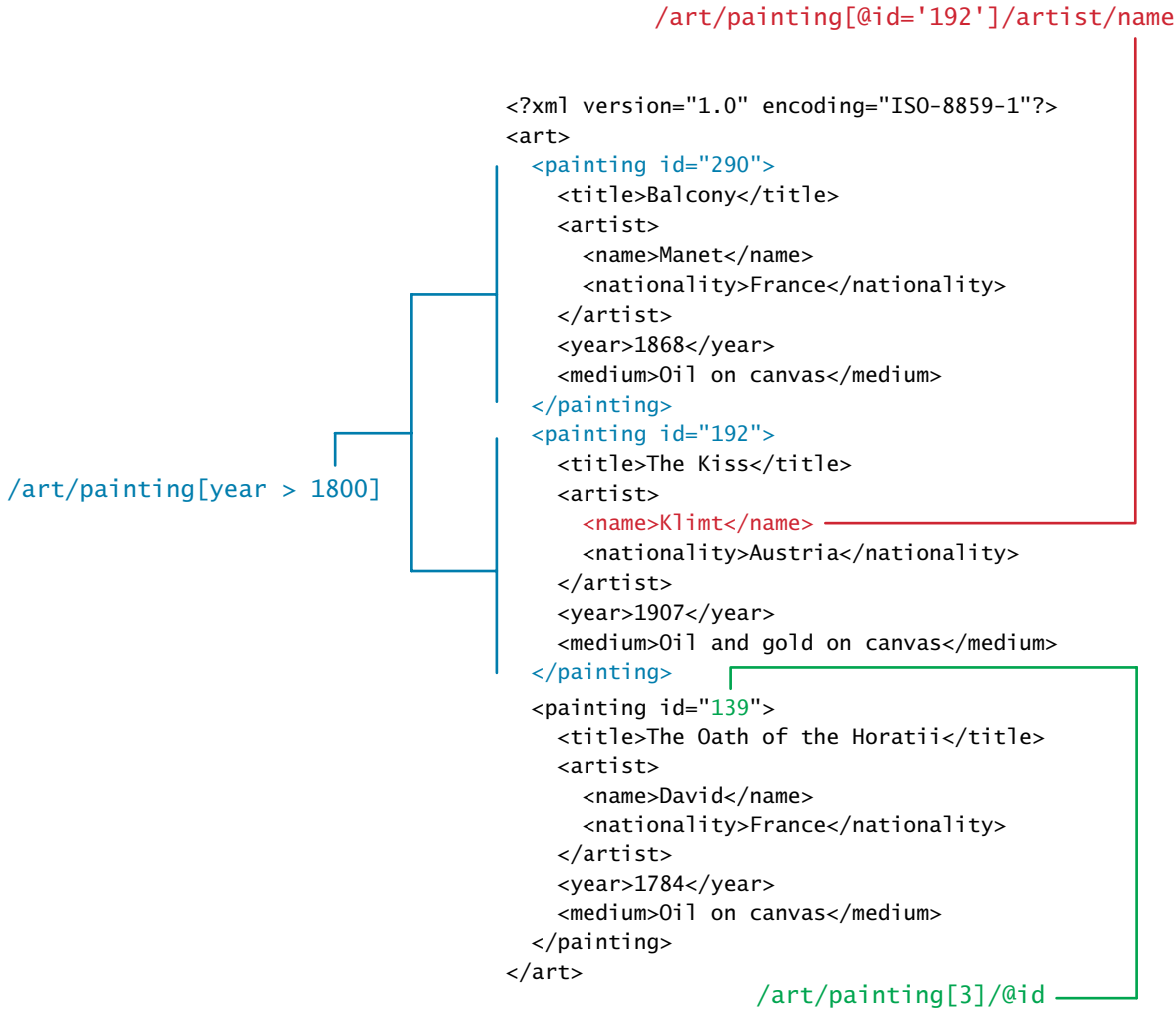


FIGURE 17.5 Sample XPath expressions

https://hemanthrajhemu.github.io

example is `/art/painting` (i.e., each `painting` node is examined to see if its `id` attribute is equal to the value 192). Notice that when referencing a node using an index expression (e.g., `painting[3]`), XPath expressions begin with one and not zero. As well, you will notice that attributes are identified in XPath expressions by being prefaced by the `@` character.

We will be using XPath in later examples in the chapter when we process XML-based web services.

17.2 XML Processing

XML processing in PHP, JavaScript, and other modern development environments is divided into two basic styles:

- The **in-memory approach**, which involves reading the entire XML file into memory into some type of data structure with functions for accessing and manipulating the data.
- The **event or pull approach**, which lets you pull in just a few elements or lines at a time, thereby avoiding the memory load of large XML files.

17.2.1 XML Processing in JavaScript

All modern browsers have a built-in XML parser and their JavaScript implementations support an in-memory XML DOM API, which loads the entire document into memory where it is transformed into a hierarchical tree data structure. You can then use the already familiar DOM functions such as `getElementById()`, `getElementsByTagName()`, and `createElement()` to access and manipulate the data.

HANDS-ON EXERCISES

LAB 17 EXCERCISE
JavaScript XML Processing

For instance, Listing 17.5 shows the code necessary for loading an XML document into an XML DOM object, and it displays the `id` attributes of the `<painting>` elements as well as the content of each painting's `<title>` element.

```
<script>
if (window.XMLHttpRequest)  {
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}
else  {
  // code for old versions of IE (optional you might just decide to
  // ignore these)
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
```

(*continued*)

https://hemanthrajhemu.github.io

```
// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");
if (paintings) {
   // loop through each painting element
   for (var i = 0; i < paintings.length; i++)
   {
      // display its id attribute
      alert("id="+paintings[i].getAttribute("id"));

      // find its <title> element
      title = paintings[i].getElementsByTagName("title");
      if (title) {
         // display the text content of the <title> element
         alert("title="+title[0].textContent);
      }
   }
}
</script>
```

LISTING 17.5 Loading and processing an XML document via JavaScript

NOTE

For security reasons, both the web page and the XML file it tries to load via JavaScript must be located on the same domain/server.

JavaScript can also manipulate XML that is contained within a string rather than in an external file. The technique for doing so differs in Internet Explorer, so the code would look similar to the following:

```
art = '<?xml version="1.0" encoding="ISO-8859-1"?>';
art += '<art><painting id="290"><title>Balcony … </art>';
if (window.DOMParser)  {
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(art,"text/xml");
}
else {
  // Internet Explorer
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false;
  xmlDoc.loadXML(art);
}
```

https://hemanthrajhemu.github.io

As can be seen in Listing 17.5, JavaScript supports a variety of node traversal functions as well as properties for accessing information within an XML node.

jQuery provides an alternate way to process XML that handles the cross-browser support for you.[5] Listing 17.6 illustrates the use of jQuery that performs the exact same processing as shown in Listing 17.5, except the XML is loaded from a string.

```
art = '<?xml version="1.0" encoding="ISO-8859-1"?>';
art += '<art><painting id="290"><title>Balcony … </art>';

// use jQuery parseXML() function to create the DOM object
xmlDoc = $.parseXML( art );
// convert DOM object to jQuery object
$xml = $( xmlDoc );

// find all the painting elements
$paintings = $xml.find( "painting" );
// loop through each painting element
$paintings.each(function() {
   // display its id
   alert($(this).attr("id"));
   // find the title element within the current painting element
   $title = $(this).find( "title" );
   // and display its content
   alert( $title.text() );
});
```

LISTING 17.6 XML processing using jQuery

While using the `alert()` function to display XML content is fine for learning purposes, a real example would likely display the XML data as HTML content. Listing 17.7 expands on the previous listing to insert the XML content into a `<div>` element within the HTML document.

Later in the chapter, we will use these techniques to asynchronously request an XML file and then update HTML elements to display the XML content.

17.2.2 XML Processing in PHP

PHP provides several extensions or APIs for working with XML:[6]

- The DOM extension, which loads the entire document into memory where it is transformed into a hierarchical tree data structure. This DOM approach is relatively standardized, in that many other development environments and languages implement relatively similarly named functions/methods for accessing and manipulating the data.

https://hemanthrajhemu.github.io

- The SimpleXML extension, which loads the data into an object that allows the developer to access the data via array properties and modifying the data via methods.
- The XML parser is an event-based XML extension. This is sometimes referred to as a SAX-style API, which for PHP developers confusingly stands for Simple API for XML, which was the original package for processing XML in the Java environment. This is generally a complicated approach that requires defining handlers for each XML type (e.g., element, attribute, etc.).
- The XMLReader is a read-only pull-type extension that uses a cursor-like approach similar to that used with database processing. The XMLWriter provides an analogous approach for creating XML files.

```
<body>
…
<div id="container"></div>

<script>
art = '<?xml version="1.0" encoding="ISO-8859-1"?>';
art += '<art><painting id="290"><title>Balcony … </art>';

xmlDoc = $.parseXML( art );
$xml = $( xmlDoc );

$paintings = $xml.find( "painting" );
$paintings.each(function() {
   // add XML content to <div> element
   $( "#container" ).append( $(this).attr("id") + " - ");
   $( "#container" ).append( $(this).find( "title" ).text() + "<br/>");
});

</script>
```

LISTING 17.7 Using jQuery to inject XML data into an HTML element

In general, the SimpleXML and the XMLReader extensions provide the easiest ways to read and process XML content. Let us begin with the SimpleXML approach, which reads the entire XML file into memory and transforms into a complex object. Like the DOM extension, the SimpleXML extension is not a sensible solution for processing very large XML files because it reads the entire file into server memory; however, since the file is in memory, it offers fast performance.

Listing 17.8 shows how our XML file is transformed into an object using the `simplexml_load_file()` function. The various elements in the XML document can then be manipulated using regular PHP object techniques.

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Reading XML in PHP Using SimpleXML

https://hemanthrajhemu.github.io

```
<?php

$filename = 'art.xml';
if (file_exists($filename)) {
   $art = simplexml_load_file($filename);

   // access a single element
   $painting = $art->painting[0];
   echo '<h2>' . $painting->title . '</h2>';
   echo '<p>By ' . $painting->artist->name . '</p>';
   // display id attribute
   echo '<p>id=' . $painting["id"] . '</p>';

   // loop through all the paintings
   echo "<ul>";
   foreach ($art->painting as $p)
   {
      echo '<li>' . $p->title . '</li>';
   }
   echo '</ul>';
} else {
   exit('Failed to open ' . $filename);
}

?>
```

LISTING 17.8 Using SimpleXML

You can also use the power of XPath expressions with SimpleXML to make it very easy to find and filter content in an XML file. Any object in the object tree can access the `xpath()` method; Listing 17.9 demonstrates some sample usages of this method.

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Reading XML in PHP Using XMLReader

```
$art = simplexml_load_file($filename);

$titles = $art->xpath('/art/painting/title');
foreach ($titles as $t) {
   echo $t . '<br/>';
}

$names = $art->xpath('/art/painting[year>1800]/artist/name');
foreach ($names as $n) {
   echo $n . '<br/>';
}
```

LISTING 17.9 Using XPath with SimpleXML

https://hemanthrajhemu.github.io

NOTE

While XML element names can contain the hyphen character, PHP does not allow hyphens in variable names. So if your XML file contains elements with hyphens, you will have to use an alternative approach.

For instance, consider the following XML file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <book>
    <copyright-year>2014</copyright-year>
    ...
  </book>
  ...
</catalog>
```

To access the elements with hyphens, we would need to encapsulate the element name within braces and the apostrophe:

```
$catalog = simplexml_load_file($filename);
echo $catalog->book[0]->{'copyright-year'};
```

While the SimpleXML extension is indeed very straightforward to use, it is not a sensible choice for reading very large XML files. In such a case, the XMLReader is a better choice. The XMLReader is sometimes referred to as a pull processor, in that it reads a single node at a time, and then the program has to determine what to do with that node. As can be seen in Listing 17.10, the code for this processing is more difficult; indeed, for a multilevel XML file, the code can become quite complicated.

```
$filename = 'art.xml';
if (file_exists($filename)) {

   // create and open the reader
   $reader = new XMLReader();
   $reader->open($filename);

   // loop through the XML file
   while ( $reader->read() ) {
     $nodeName = $reader->name;

     // since all sorts of different XML nodes we must check
     // node type
     if ($reader->nodeType == XMLREADER::ELEMENT
         && $nodeName == 'painting') {
```

https://hemanthrajhemu.github.io

```
            $id = $reader->getAttribute('id');
            echo '<p>id=' . $id . '</p>';
        }

        if ($reader->nodeType == XMLREADER::ELEMENT
            && $nodeName =='title') {
            // read the next node to get at the text node
            $reader->read();
            echo '<p>' . $reader->value . '</p>';
        }
    }
} else {
    exit('Failed to open ' . $filename);
}
```

LISTING 17.10 Using XMLReader

One way to simplify the use of XMLReader is to combine it with SimpleXML. We will use the XMLReader to read in a <painting> element at a time (perhaps in the real XML file, there are thousands of <painting> elements, so we don't want to read them all into memory). We can then pass on the element to SimpleXML and let it convert that single element into an object to simplify our programming. Listing 17.11 demonstrates how these two extensions can be combined to get the memory advantages of the XMLReader along with the programming simplicity of SimpleXML.

```
// create and open the reader
$reader = new XMLReader();
$reader->open($filename);

// loop through the XML file
while($reader->read()) {
  $nodeName = $reader->name;
  if ($reader->nodeType == XMLREADER::ELEMENT
      && $nodeName =='painting') {
      // create a SimpleXML object from the current painting node
      $doc = new DOMDocument('1.0', 'UTF-8');
      $painting = simplexml_import_dom($doc->importNode(
                    $reader->expand(),true));
      // now have a single painting as an object so can output it
      echo '<h2>' . $painting->title . '</h2>';
      echo '<p>By ' . $painting->artist->name . '</p>';
  }
}
```

LISTING 17.11 Combining XMLReader and SimpleXML

https://hemanthrajhemu.github.io

17.3 JSON

Like XML, JSON is a data serialization format. That is, it is used to represent object data in a text format so that it can be transmitted from one computer to another. Many REST web services encode their returned data in the JSON data format instead of XML. While JSON stands for JavaScript Object Notation, its use is not limited to JavaScript. It provides a more concise format than XML to represent data. It was originally designed to provide a lightweight serialization format to represent objects in JavaScript. While it doesn't have the validation and readability of XML, it has the advantage of generally requiring significantly fewer bytes to represent data than XML, which in the web context is quite significant. Figure 17.6 shows an example of how an XML data element would be represented in JSON.

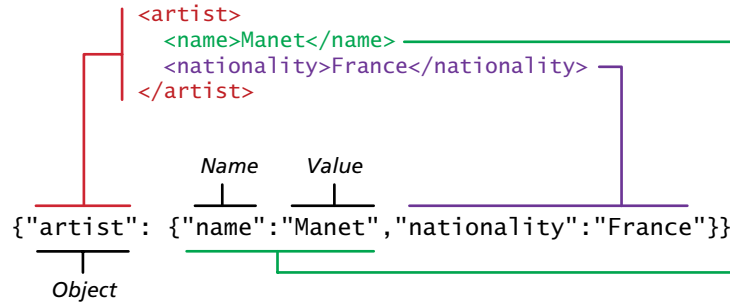


FIGURE 17.6 Sample JSON

Just like XML, JSON data can be nested to represent objects within objects. Listing 17.12 demonstrates how the data in Listing 17.1 could be represented in JSON. While Listing 17.12 uses spacing and line breaks to make the structure more readable, in general JSON data will have all white space removed to reduce the number of bytes traveling across the network.

```
{
   "paintings": [
      {
         "id":290,
         "title":"Balcony",
         "artist":{
            "name":"Manet",
            "nationality":"France"
         },
         "year":1868,
         "medium":"Oil on canvas"
      },
```

https://hemanthrajhemu.github.io

```
        {
           "id":192,
           "title":"The Kiss",
           "artist":{
              "name":"Klimt",
              "nationality":"Austria"
           },
           "year":1907,
           "medium":"Oil and gold on canvas"
        },
        {
           "id":139,
           "title":"The Oath of the Horatii",
           "artist":{
              "name":"David",
              "nationality":"France"
           },
           "year":1784,
           "medium":"Oil on canvas"
        }
     ]
}
```

LISTING 17.12 JSON representation of XML data from Listing 17.1

Notice how this example uses square brackets to contain the three painting object definitions: this is the JSON syntax for defining an array.

17.3.1 Using JSON in JavaScript

Since the syntax of JSON is the same used for creating objects in JavaScript, it is easy to make use of the JSON format in JavaScript:

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Reading JSON in JavaScript

```
<script>
  var a = {"artist": {"name":"Manet","nationality":"France"}};
  alert(a.artist.name + " " + a.artist.nationality);
</script>
```

While this is indeed quite straightforward, generally speaking you will not often hard-code JSON objects like that shown above. Instead, you will either programmatically construct them or download them from an external web service. In either case, the JSON information will be contained within a string, and the `JSON.parse()` function can be used to transform the string containing the JSON data into a JavaScript object:

https://hemanthrajhemu.github.io

```
var text = '{"artist": {"name":"Manet","nationality":"France"}}';
var a = JSON.parse(text);
alert(a.artist.nationality);
```

The jQuery library also provides a JSON parser that will work with all browsers (the `JSON.parse()` function is not available on older browsers):

```
var artist = jQuery.parseJSON(text);
```

JavaScript also provides a mechanism to translate a JavaScript object into a JSON string:

```
var text = JSON.stringify(artist);
```

17.3.2 Using JSON in PHP

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Reading JSON in PHP

PHP comes with a JSON extension and as of version 5.2 of PHP, the JSON extension is bundled and compiled into PHP by default.[7] Converting a JSON string into a PHP object is quite straightforward:

```
<?php
  // convert JSON string into PHP object
  $text = '{"artist": {"name":"Manet","nationality":"France"}}';
  $anObject = json_decode($text);
  echo $anObject->artist->nationality;

  // convert JSON string into PHP associative array
  $anArray = json_decode($text, true);
  echo $anArray['artist']['nationality'];
?>
```

Notice that the `json_decode()` function can return either a PHP object or an associative array. Since JSON data is often coming from an external source, one should always check for parse errors before using it, which can be done via the `json_last_error()` function:

```
<?php
  // convert JSON string into PHP object
  $text = '{"artist": {"name":"Manet","nationality":"France"}}';
  $anObject = json_decode($text);
  // check for parse errors
  if (json_last_error() == JSON_ERROR_NONE) {
    echo $anObject->artist->nationality;
  }
?>
```

https://hemanthrajhemu.github.io

To go the other direction (i.e., to convert a PHP object into a JSON string), you can use the `json_encode()` function.

```
// convert PHP object into a JSON string
$text = json_encode($anObject);
```

In the next three sections we will be making more use of JSON in PHP and JavaScript.

17.4 Overview of Web Services

Web services are the most common example of a computing paradigm commonly referred to as **service-oriented computing** (SOC), which utilizes something called "services" as a key element in the development and operation of software applications.

A **service** is a piece of software with a platform-independent interface that can be dynamically located and invoked. **Web services** are a relatively standardized mechanism by which one software application can connect to and communicate with another software application using web protocols. Web services make use of the HTTP protocol so that they can be used by any computer with Internet connectivity. As well, web services typically use XML or JSON (which will be covered shortly) to encode data within HTTP transmissions so that almost any platform should be able to encode or retrieve the data contained within a web service.

The benefit of web services is that they potentially provide interoperability between different software applications running on different platforms. Because web services use common and universally supported standards (HTTP and XML/JSON), they are supported on a wide variety of platforms. Another key benefit of web services is that they can be used to implement something called a **service-oriented architecture** (SOA). This type of software architecture aims to achieve very loose coupling among interacting software services. The rationale behind an SOA is one that is familiar to computing practitioners with some experience in the enterprise: namely, how to best deal with the problem of application integration. Due to corporate mergers, longer-lived legacy applications, and the need to integrate with the Internet, getting different software applications to work together has become a major priority of IT organizations. SOA provides a very palatable potential solution to application integration issues. Because services are independent software entities, they can be offered by different systems within an organization as well as by different organizations. As such, web services can provide a computing infrastructure for application integration and collaboration within and between organizations, as shown in Figure 17.7.

https://hemanthrajhemu.github.io

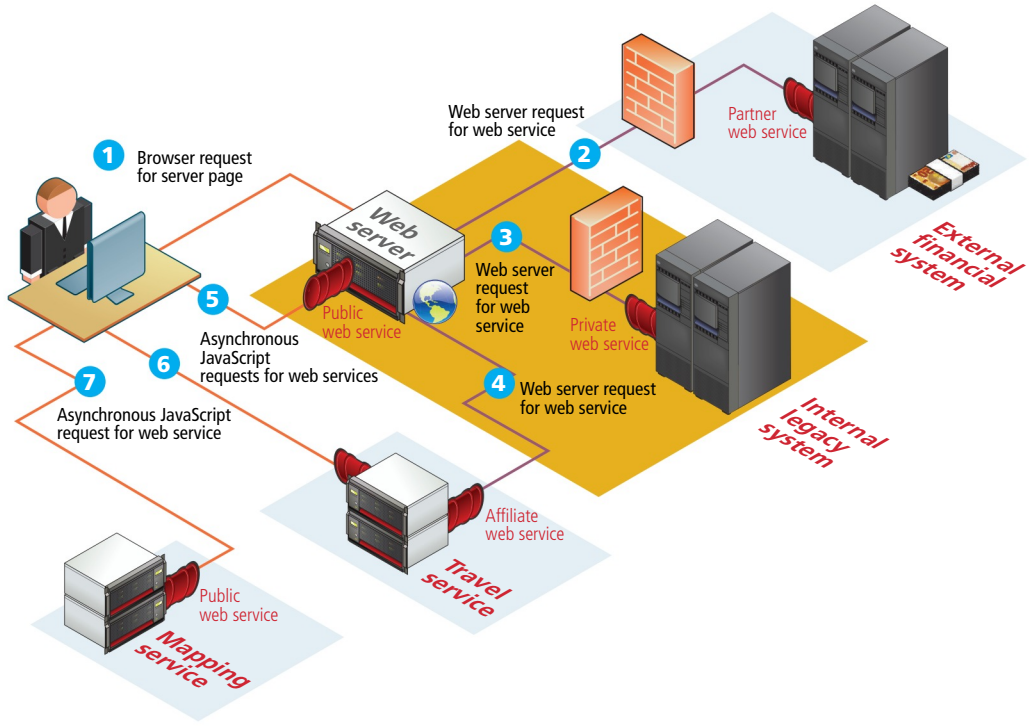


FIGURE 17.7 Overview of web services

In the first few years of the 2000s, there was a great deal of enthusiasm for service-oriented computing in general and web services in particular. The hope was that development in which an application's functional capability was externalized into services would finally realize the reusability promised by object-oriented languages as well as deal with the difficulty of enterprise-level application integration.

17.4.1 SOAP Services

In the first iteration of web services fever, the attention was on a series of related XML vocabularies: WSDL, SOAP, and the so-called WS-protocol stack (WS-Security, WS-Addressing, etc.). In this model, WSDL is used to describe the operations and data types provided by the service. SOAP is the message protocol used to encode the service invocations and their return values via XML within the HTTP header, as can be seen in Figure 17.8.

While SOAP and WSDL are complex XML schemas, this now relatively mature standard is well supported in the .NET and Java environments (perhaps a little less so with PHP). From the authors' professional and teaching experience, it is not necessary to have detailed knowledge of the SOAP and WSDL specifications to create and consume SOAP-based services. Using SOAP-based services is somewhat

https://hemanthrajhemu.github.io

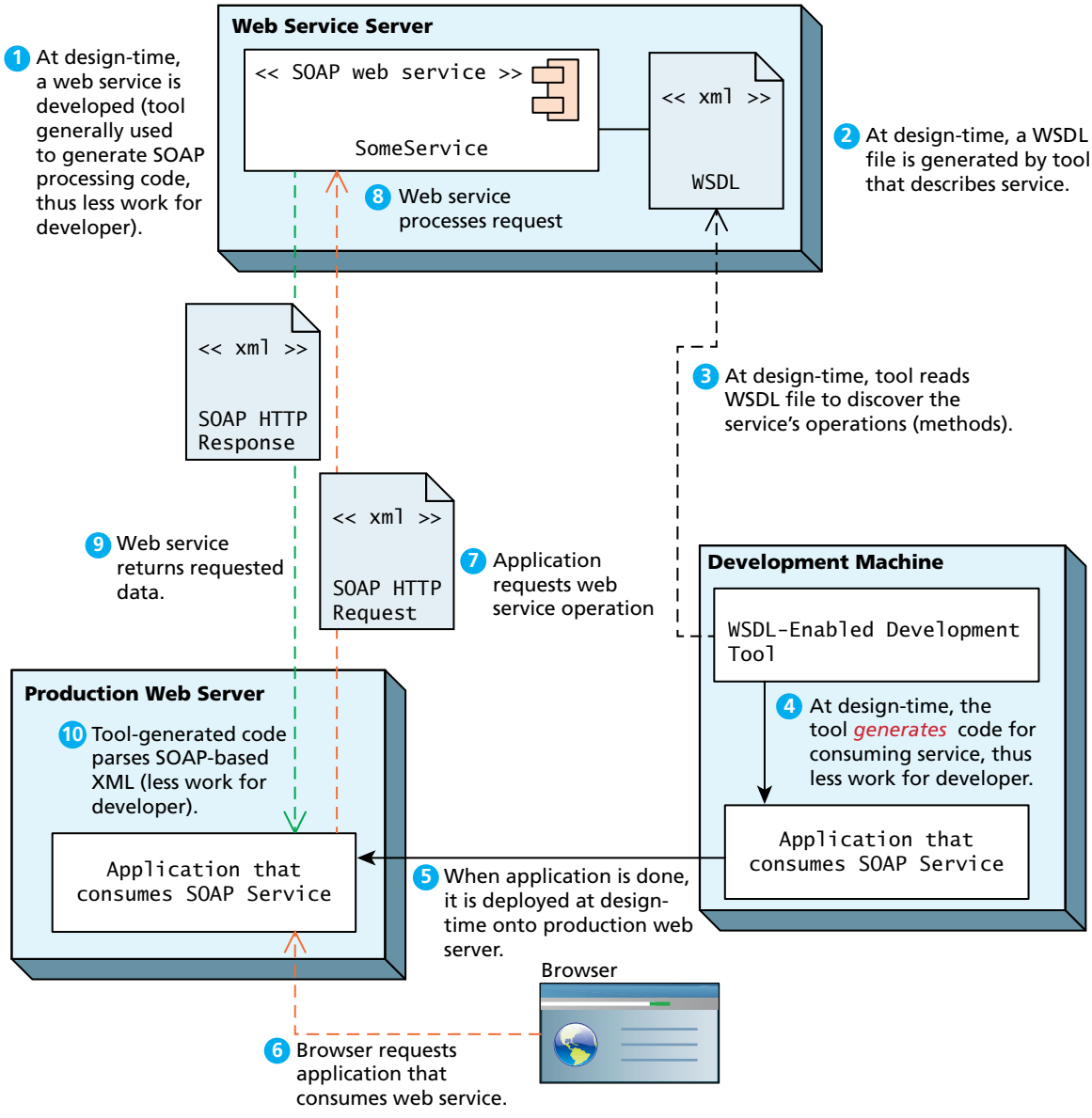


FIGURE 17.8 SOAP web services

akin to using a compiler: its output may be complicated to understand, but it certainly makes life easier for most programmers. Yet, despite the superb tool support in these two environments, by the middle years of the 2000s, the enthusiasm for SOAP-based web services had cooled.

https://hemanthrajhemu.github.io

17.4.2 REST Services

By the end of the decade, the enthusiasm for web services was back, thanks to the significantly simpler REST-based web service standard. **REST** stands for Representational State Transfer. A RESTful web service does away with the service description layer as well as doing away with the need for a separate protocol for encoding message requests and responses. Instead it simply uses HTTP URLs for requesting a resource/object (and for encoding input parameters). The serialized representation of this object, usually an XML or JSON stream, is then returned to the requestor as a normal HTTP response. No special steps are needed to deploy a REST-based service, no special tools (other than a browser) are generally needed to test a RESTful service, and it is easier to scale for a large number of clients using well-established practices and experience with caching, clustering, and load-balancing traditional dynamic HTTP websites.

With the broad interest in the asynchronous consumption of server data at the browser using JavaScript (generally referred to as AJAX) in the latter half of this decade, the lightweight nature of REST made it significantly easier to consume in JavaScript than SOAP. Indeed, if an object is serialized via JSON, it can be turned into a complex JavaScript object in one simple line of JavaScript. However, since many REST web services use XML as the data format, manual XML parsing and processing is required in order to deserialize a REST response back into a usable object, as shown in Figure 17.9. (With the SOAP approach, in contrast, tools can use the WSDL document to automatically generate proxy classes at development time, which in turn obviates the necessity of writing the serialize/deserialize code yourself.)

REST appears to have almost completely displaced SOAP services. For instance, in July 2013, the **programmableweb.com** API directory had 2030 indexed SOAP services in comparison to 6088 REST services. While some of the most popular services, such as those from Amazon, eBay, and Flickr, support both formats, others, such as Facebook, Google, YouTube, and Wikipedia, have either discontinued SOAP support or have never offered it. For this reason, this chapter will only cover the consumption and creation of REST-based services.

The relatively easy availability of a wide range of RESTful services has given rise to a new style of web development, often referred to as a **mashup**, which generally refers to a website that combines and integrates data from a variety of different sources. Even websites that are not overtly mashups nonetheless often make use of some external data via the consumption of REST services. The proliferation of maps, externalized search, Amazon widgets, and so on, on a wide variety of sites are examples of the commonality of the consumption of REST services.

17.4.3 An Example Web Service

Perhaps the best way to understand RESTful web service would be to examine a sample one. In this section we will look at the Google Geocoding API. The term

https://hemanthrajhemu.github.io

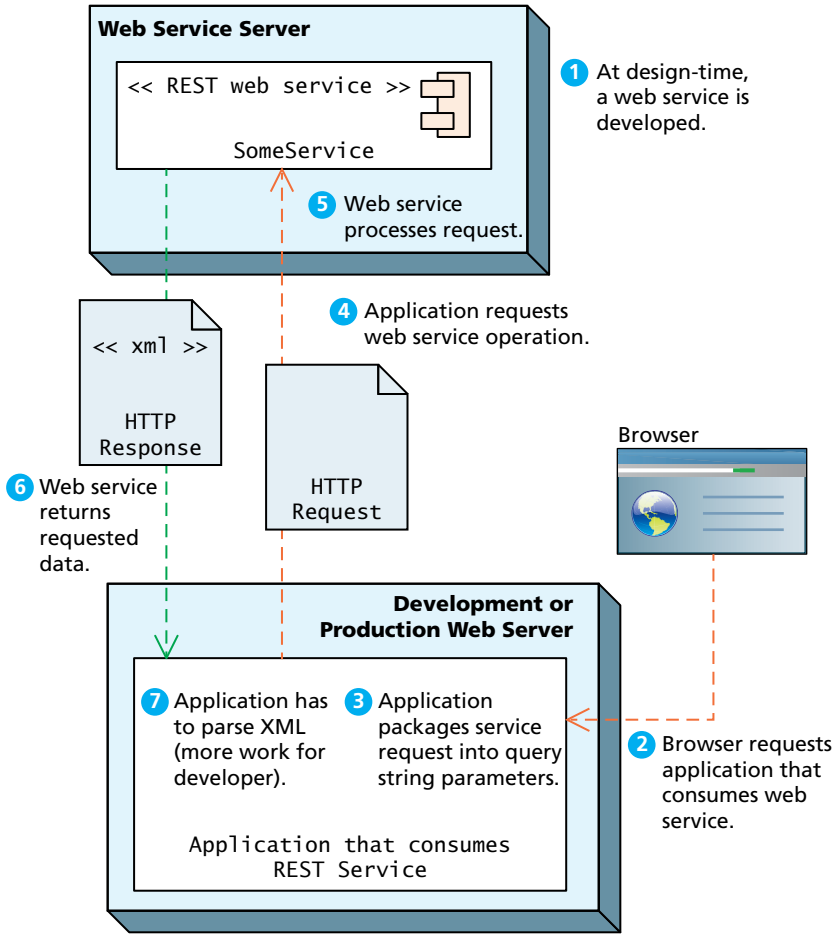


FIGURE 17.9 REST web services

geocoding typically refers to the process of turning a real-world address (such as British Museum, Great Russell Street, London, WC1B 3DG) into geographic coordinates, which are usually latitude and longitude values (such as 51.5179231, -0.1271022). Reverse geocoding is the process of converting geographic coordinates into a human-readable address.

The Google Geocoding API provides a way to perform geocoding operations via an HTTP GET request, and thus is an especially useful example of a RESTful web service.

https://hemanthrajhemu.github.io

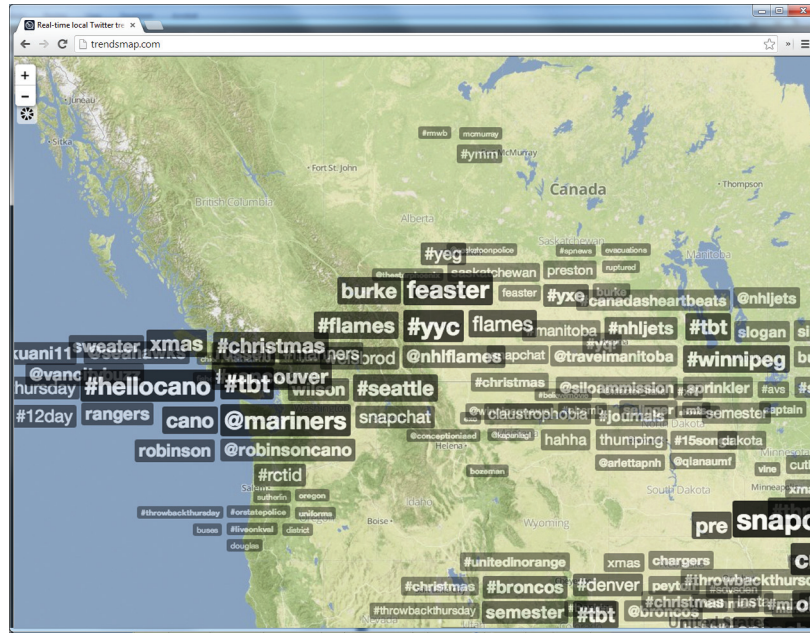


FIGURE 17.10 Example mashup combining Google Maps and Twitter (taken from TrendsMap.com)

NOTE

The Geocoding API may only be used in conjunction with a Google Map; performing a geocoding without displaying it on a map is prohibited by the Maps API Terms of Service License. In this example, we are using the service simply to illustrate a typical web service. In a real-world example, we would plot the returned latitude and longitude values on a Google Map.

Like all of the REST web services we will be examining in this chapter, using a web service begins with an HTTP request. In this case the request will take the following form:

```
http://maps.googleapis.com/maps/api/geocode/xml?parameters
```

The parameters in this case are `address` (for the real-world address to geocode) and `sensor` (for whether the request comes from a device with a location sensor).

https://hemanthrajhemu.github.io

So an example geocode request would look like the following:

```
http://maps.googleapis.com/maps/api/geocode/xml?address=British%20
Museum,+Great+Russell+Street,+London,+WC1B+3DG&sensor=false
```

Notice that a REST request, like all HTTP requests, must URL encode special characters such as spaces. If the request is well formed and the service is working, it will return an HTTP response similar to that shown in Listing 17.13 (with some omissions and indenting spaces added for readability).

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Date: Fri, 19 Jul 2013 19:15:54 GMT
Expires: Sat, 20 Jul 2013 19:15:54 GMT
Cache-Control: public, max-age=86400
Vary: Accept-Language
Content-Encoding: gzip
Server: mafe
Content-Length: 512
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
   <status>OK</status>
   <result>
      <type>route</type>
      <formatted_address>
         Great Russell Street, London Borough of Camden, London, UK
      </formatted_address>
      <address_component>
         <long_name>Great Russell Street</long_name>
         <short_name>Great Russell St</short_name>
         <type>route</type>
      </address_component>
      <address_component>
         <long_name>London</long_name>
         <short_name>London</short_name>
         <type>locality</type>
         <type>political</type>
      </address_component>
      ...
      <geometry>
         <location>
            <lat>51.5179231</lat>
            <lng>-0.1271022</lng>
         </location>
```

(*continued*)

https://hemanthrajhemu.github.io

```
            <location_type>GEOMETRIC_CENTER</location_type>
            ...
        </geometry>
    </result>
</GeocodeResponse>
```

LISTING 17.13 HTTP response from web service

After receiving this response, our program would then presumably need some type of XML processing in order to extract the latitude and longitude values (perhaps the simplest way to do so would be via XPath).

17.4.4 Identifying and Authenticating Service Requests

The previous section illustrated a sample request to a REST-based web service and its XML response. That particular service was openly available to any request (though its term of service license limited how the response data could be used). Most web services are not open in the same way. Instead, they typically employ one of the following techniques:

- Identity. Each web service request must identify who is making the request.
- Authentication. Each web service request must provide additional evidence that they are who they say they are.

Many web services are not providing information that is especially private or proprietary. For instance, the Flickr web service, which provides URLs to publicly available photos on their site in response to search criteria, is in some ways simply an XML version of the main site's already existing search facility. Since no private user data is being requested, it only expects each web service request to include one or more API keys to identity who is making the request.

This typically is done not only for internal record-keeping, but more importantly to keep service request volume at a manageable level. Most external web service APIs limit the number of web service requests that can be made, generally either per second, per hour, or per day. For instance, Panoramio limits requests to 100,000 per day while Google Maps and Microsoft Bing Maps allow 50,000 geocoding requests per day; Instagram allows 5000 requests per hour but Twitter allows just 100 to 400 requests per hour (it can vary); Amazon and last.fm limit requests to just one per second. Other services such as Flickr, NileGuide, and YouTube have no documented request limits.

Web services that make use of an API key typically require the user (i.e., the developer) to register online with the service for an API key. This API key is then added to the `GET` request as a query string parameter. For instance, a geocoding

https://hemanthrajhemu.github.io

request to the Microsoft Bing Maps web service will look like the following (in this particular case, the actual Bing API key is a 64-character string):

```
http://dev.virtualearth.net/REST/v1/Locations?o=xml&query=British%20
Museum,+Great+Russell+Street,+London,+WC1B+3DG,+UK&key=[BING API KEY
HERE]
```

NOTE

In the examples that follow in the rest of this chapter (and in the associated lab exercises), it will be assumed that the reader has registered for the relevant services and has the necessary API key.

While some web services are simply providing information already available on their website, other web services are providing private/proprietary information or are involving financial transactions. In this case, these services not only may require an API key, but they also require some type of user name and password in order to perform an authorization.

In such a case, user credential information is almost never sent via `GET` query string parameters due to the security risk. Instead this information is sent within the HTTP or HTTPS `Authorization` header as discussed in the previous chapter on Security. This could use HTTP basic authentication; many of the most well-known web services instead make use of the OAuth standard (covered in Chapter 16) since it eliminates the need to transmit passwords in service requests.

17.5 Consuming Web Services in PHP

Now that we understand REST web services and know how to process both XML and JSON, we are ready to consume some web services in PHP. There are three usual approaches in PHP for making a REST request:

- Using the `file_get_contents()` function.
- Using functions contained within the `curl` library.
- Using a custom library for the specific web service. Many of the most popular web services have free and proprietary PHP libraries available.

The `file_get_contents()` function is simple but doesn't allow `POST` requests, so services that require authentication will have to use the `curl` extension library, which allows significantly more control over requests. Unfortunately, not all PHP

https://hemanthrajhemu.github.io

servers allow usage of `curl`. To test if your installation supports `curl`, create a simple page with the following code and then run it:

```
<?php
    echo phpinfo();
?>
```

This will display information about your PHP installation. About a quarter of the way down the listing, if `curl` is installed, you will find information about its support. If you are using XAMPP then `curl` support should be enabled.

17.5.1 Consuming an XML Web Service

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Consuming an XML Web Service in PHP

The Flickr web service (documentation available at http://www.flickr.com/services/api/) provides a comprehensive set of web services for interacting with its vast library of user-supplied photos. Perhaps its most commonly used service method is its photo search facility. The basic format for this service method is:

```
http://api.flickr.com/services/rest/?method=flickr.photos.search&api_
key=[enter your flickr api key here]&tags=[search values
here]&format=rest
```

Notice that this service request has a specific URL, which can be discovered by examining the web service API documentation. As well, various query string parameters indicate which service method we are requesting (in this case, `method=flickr.photos.search`). As well, we need to supply our own API key, our search tags, and specify whether we want the service to return its results as XML (REST) or as JSON. The documentation for the service describes other parameters that can be specified.

The service will return its standard XML photo list, which is shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
  <photos page="1" pages="9" perpage="10" total="82">
    <photo id="8711739266" owner="31790027@N04" secret="0f29a86417"
       server="8560" farm="9" title="Back end of the Parthenon"
       ispublic="1" isfriend="0" isfamily="0" />
    <photo id="8710493439" owner="31790027@N04" secret="66b58d04a7"
       server="8406" farm="9" title="Me at the Agora" ispublic="1"
       isfriend="0" isfamily="0" />
    ...
  </photos>
</rsp>
```

https://hemanthrajhemu.github.io

We can turn the id, server, farm, and secret attributes of the returned <photo> elements into URLs using the following format:

```
http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}_
[mstzb].jpg
```

In this case, the mstzb refers to the size (m = small, s = small square, t = thumbnail, z = medium, or b = large). For instance, to use the data from the first <photo> element in the above example into a request for a small square version of the photo, you would use:

```
http://farm9.staticflickr.com/8560/8711739266_0f29a86417_s.jpg
```

Now that we have covered how the API works, let's write the PHP to make the request. To begin, we will encapsulate the creation of the search request in a PHP function that is shown in Listing 17.14.

```
<?php

function constructFlickrSearchRequest($search)
{
   $serviceDomain = 'http://api.flickr.com/services/rest/?';
   $method = 'method=flickr.photos.search';
   $api_key = 'api_key=' . 'your Flickr api key here';
   $searchFor = 'tags=' . $search;
   $format = 'format=rest';
   // only 12 results for now
   $options = 'per_page=12';
   // due to copyright, we will use only the author's Flickr images
   $options .= '&user_id=31790027%40N04';

   return $serviceDomain . $method . '&' . $api_key .'&'
       . $searchFor . '&' . $format . '&' . $options;
}

?>
```

LISTING 17.14 Function to construct Flickr search request

With the service request function created, we can now simply make the request, examine the response for errors, and for now, simply display the XML (which will need to be HTML encoded due to the angle brackets in the returned XML), as shown below. Notice that this example has a hard-coded search string. Of course, we could easily generalize the example to instead use a value from a database or a user input form.

```
<?php
// for now just hard-code the search
```

https://hemanthrajhemu.github.io

```
$request = constructFlickrSearchRequest('Athens');
$response = file_get_contents($request);

// Retrieve HTTP status code
$statusLine = explode(' ',$http_response_header[0], 3);
$status_code = $statusLine[1];

if ($status_code == 200) {
   // for debugging output response
   echo htmlspecialchars($response);
}
else {
   die("Your call to web service failed -- code=" . $status_code);
}
?>
```

One can achieve the same functionality using the `curl` extension; it requires a little more code but provides more control and allows `POST` requests as well. Listing 17.15 demonstrates how the `curl` extension is used to make a web service request. It also makes use of the XML processing techniques from earlier in the chapter to display thumbnail versions of the images as shown in Figure 17.11.

```
$request = constructFlickrSearchRequest('Athens');
echo '<p><small>' . $request . '</small></p>';

$http = curl_init($request);
// set curl options
curl_setopt($http, CURLOPT_HEADER, false);
curl_setopt($http, CURLOPT_RETURNTRANSFER, true);
// make the request
$response = curl_exec($http);
// get the status code
$status_code = curl_getinfo($http, CURLINFO_HTTP_CODE);
// close the curl session
curl_close($http);

if ($status_code == 200) {
   // create simpleXML object by loading string
   $xml = simplexml_load_string($response);
   // iterate through each <photo> element
   foreach ($xml->photos->photo as $p) {
      // construct URLs for image and for link
      $pageURL = "http://www.flickr.com/photos/" . $p['owner'] . "/"
                 . $p['id'];
```

https://hemanthrajhemu.github.io

```
        $imgURL = "http://farm" .$p["farm"] . ".staticflickr.com/"
          . $p["server"] . "/" . $p["id"] . "_" . $p["secret"] . "_q.jpg";
        // output links and image tags
        echo "<a href='" . $pageURL . "'>";
        echo "<img src='" . $imgURL . "' />";
        echo "</a>";
    }
}
else {
    die("Your call to web service failed -- code=" . $status_code);
}
```

LISTING 17.15 Querying web service and processing the results

Earlier in the chapter, we used the SimpleXML extension to load an XML file. In this case, the XML is contained within a string, and as a result it cannot use the `simplexml_load_file()` function. Instead it uses the `simplexml_load_string()` function.

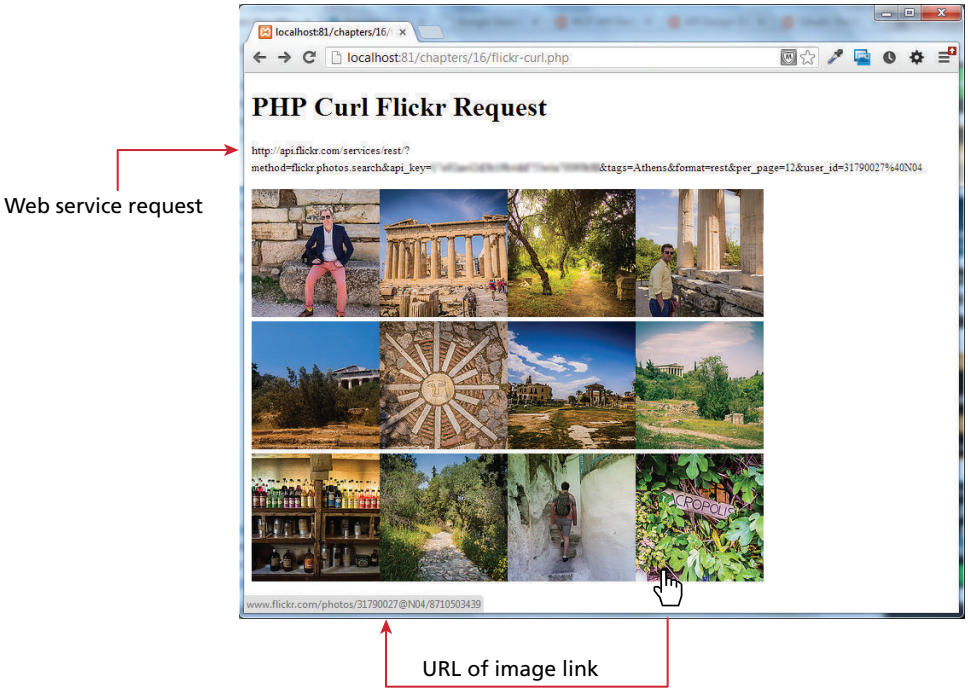


FIGURE 17.11 Result of Listing 17.15 in the browser

https://hemanthrajhemu.github.io

17.5.2 Consuming a JSON Web Service

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Consuming a JSON Web Service in PHP

Consuming a JSON web service requires almost the same type of PHP coding as consuming an XML web service. But rather than using SimpleXML to extract the information one needs, one instead uses the `json_decode()` function.

To illustrate, we will have a more involved example that makes use of two different web services. The first of these is the Microsoft Bing Maps web service (http://msdn.microsoft.com/en-us/library/ff701702.aspx). It will be used to geocode a client's address. With the returned latitude and longitude we will then use the second web service: the GeoNames web service (http://www.geonames.org/), which provides access to a database of over 10 million geographical names. We will use the service to find nearby amenities to the address. Finally, the Microsoft Bing Maps web service will be used to generate a static map image that displays the client's location along with nearby amenities. Both of these services require that you register to get the relevant API key. Figure 17.12 illustrates the process flow of this example.

By examining the web service's API documentation, you can see that our geocoding request must take the following form:

```
http://dev.virtualearth.net/REST/v1/Locations?query=address&key=
api-key
```

The `address` parameter will contain the customer's address, city, region, and country separated by commas and each will have to be URL encoded. It will return a JSON object with quite a lot of information in it; the relevant part is the latitude and longitude, which are shown in Listing 17.16 (with unneeded information omitted).

```
{ …
  "resourceSets":[
   {  …
      "resources":[
      { …
        "point":{ …
           "coordinates":[
               43.6520004, -79.4082336
           ]
        }, …
```

LISTING 17.16 Example JSON returned from geocoding request

https://hemanthrajhemu.github.io

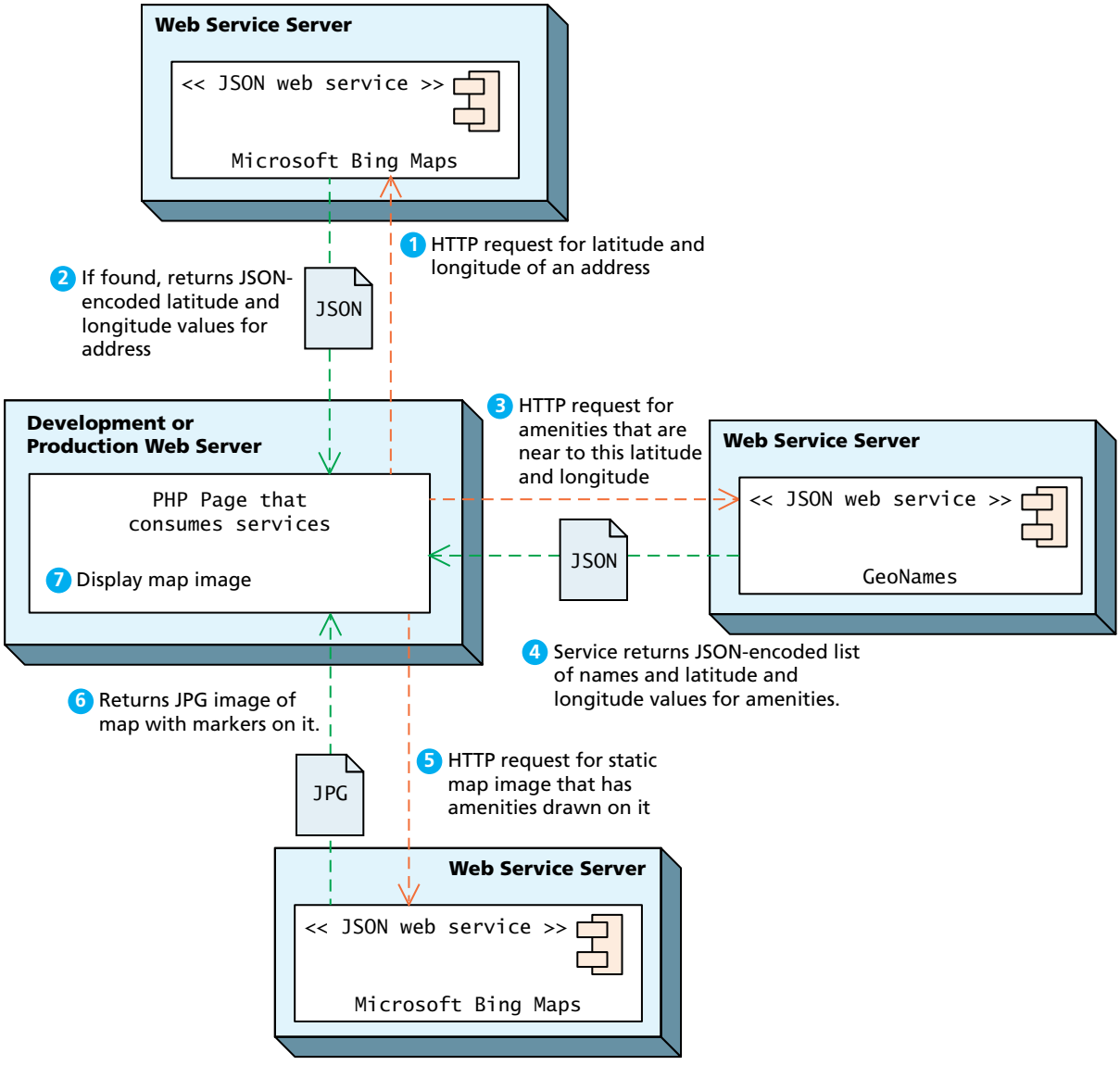


FIGURE 17.12 JSON example process

https://hemanthrajhemu.github.io

To extract the latitude and longitude from the JSON string returned from the mapping web service, you would need code similar to the following:

```
// decode JSON and extract latitude and longitude
$json = json_decode($response);
if (json_last_error() == JSON_ERROR_NONE) {
   $lat = $json->resourceSets[0]->resources[0]->point
        ->coordinates[0];
   $long = $json->resourceSets[0]->resources[0]->point
         ->coordinates[1];
}
```

Once our program has retrieved the latitude and longitude of the contact's address, the program then will use the GeoNames web service's Find NearBy Points of Interest method. This request will take the following form:

```
http://api.geonames.org/findNearbyPOIsOSMJSON?lat=43.6520004&lng=
-79.4082336&username=your-username-here
```

Notice that this request to GeoNames uses the latitude and longitude values retrieved from the previous geocoding request (i.e., from the Bing Maps service). If successful, this request will return a list of amenities as shown in Listing 17.17 (again with unneeded information omitted).

```
{
   "poi":[
      {
         "typeName":"pharmacy",
         "distance":"0.05",
         "name":"…",
         "lng":"-79.4085317",
         "typeClass":"amenity",
         "lat":"43.6517321"
      },
      …
}
```

LISTING 17.17 Example JSON returned from GeoNames request

Once these two web services requests are finished, our program can finally display the static map with a marker for the customer location and other markers for the amenity locations. For this example, we will again use the Microsoft Bing Map service. Rather than return XML or JSON, this request will return the URL of a JPG image (shown in Figure 17.13); this will simply be the `src` attribute value for an `<img>` element.

Listing 17.18 lists the PHP code used for this mapping page. Figure 17.14 illustrates what the page will look like in the browser.

https://hemanthrajhemu.github.io

URL of service request for static road map image

Zoom level (between 1 and 21)

```
http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/43.6516321,-79.4085317/16?
key=[your api key]
&mapSize=600,400
&pp=43.6516321,-79.4085317;66
&pp=43.6520854,-79.4061892;34
&pp=43.6516601,-79.4095859;34
```

Width and height of map in pixels

Location (latitude and longitude) of center of map

Location of marker (marker 66 = blue circle)

Location of other markers (amenities) with marker 34 = orange circle

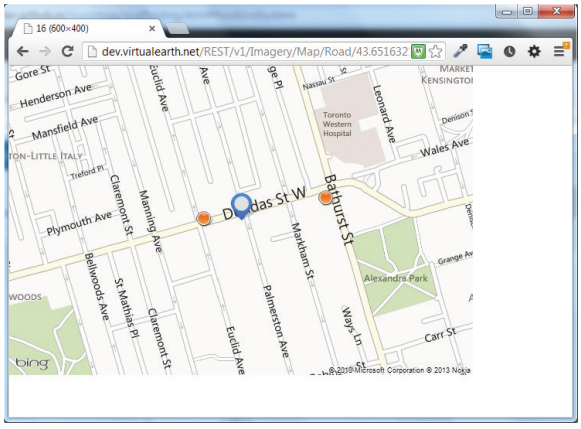


FIGURE 17.13 Map request format

```php
<?PHP

// First define api key constants - you will replace these values
define("BING_API_KEY",'[your api key here]');
define("GEONAMES_API_USERNAME", '[your username here]');

//
// Constructs the URL to retrieve lat/long for a real-world
// address. It is passed a customer object
//
function constructBingSearchRequest($customer)
{
   $serviceDomain = 'http://dev.virtualearth.net/REST/v1/Locations?';
   $api_key = 'key=' . BING_API_KEY;
   $query = 'query=' . urlencode($customer->address) . ','
      . urlencode($customer->city) . ',' . $customer->region . ','
      . $customer->country;

   return $serviceDomain . $api_key . '&' . $query;
}
```

(*continued*)

https://hemanthrajhemu.github.io

```
//
// Constructs the URL to retrieve nearby amenities to a location
//
function constructGeoNameSearchRequest($lat, $long)
{
   $serviceDomain = 'http://api.geonames.org/findNearbyPOIsOSMJSON?';
   $api_key = 'username=' . GEONAMES_API_USERNAME;
   $query = 'lat=' . $lat . '&lng=' . $long;
   return $serviceDomain . $api_key . '&' . $query;
}
//
// Constructs the URL for static map with main location and amenities
//
function constructBingMapRequest($zoom, $width, $length, $lat,
                                 $long, $amenities)
{
   $serviceDomain = 'http://dev.virtualearth.net/REST/v1/Imagery/
                     Map/Road/';
   $api_key = 'key=' . BING_API_KEY;

   $request = $serviceDomain . $lat . ',' . $long . '/' . $zoom;
   $request .= '?mapSize=' . $width . ',' . $length . '&' . $api_key;
   $request .= '&pp=' . $lat . ',' . $long . ';66';
   foreach ($amenities as $amenity)
   {
      $request .= '&pp=' . $amenity->lat . ',' . $amenity->lng . ';34';
   }
   return $request;
}
//
// Invokes/requests a web service and returns its response.
// For simplicity's sake, if problem with service it simply dies.
// For real-world site, would need better error handling.
//
function invokeWebService($request)
{
   $http = curl_init($request);
   curl_setopt($http, CURLOPT_HEADER, false);
   curl_setopt($http, CURLOPT_RETURNTRANSFER, true);
   $response = curl_exec($http);
   $status_code = curl_getinfo($http, CURLINFO_HTTP_CODE);
   curl_close($http);

   if ($status_code == 200) {
      return $response;
   }
   else {
```

https://hemanthrajhemu.github.io

```
        die("Your call to web service failed -- code=" . $status_code);
    }
}
//
// Code that implements algorithm from Figure 17.12. Notice that it
// returns the populated image tag for the map image
//
function getCustomerMapImage($customer)
{
    // call web service
    $request = constructBingSearchRequest($customer);
    $response = invokeWebService($request);

    // now decode JSON and extract latitude and longitude
    $json = json_decode($response);
    if (json_last_error() == JSON_ERROR_NONE) {
        $lat = $json->resourceSets[0]->resources[0]->point
               ->coordinates[0];
        $long = $json->resourceSets[0]->resources[0]->point
                ->coordinates[1];

        // with this lat/long, get list of amenities
        $request = constructGeoNameSearchRequest($lat, $long);
        $response = invokeWebService($request);

        $json = json_decode($response);
        if (json_last_error() == JSON_ERROR_NONE) {
            // now get map image with location and amenity markers
            $mapImageURL = constructBingMapRequest(16, 600, 400, $lat,
                                $long, $json->poi);
            $img = '<img src="' . $mapImageURL . '" alt="map here" />';
            return $img;
        }
    }
}

// Somewhere in your page, you will have to get the customer object
$customer = getCustomer();

// And then somewhere on the page there will be this call, which
// displays the map image.

echo getCustomerMapImage($customer);

?>
```

LISTING 17.18 PHP used in the mapping page

https://hemanthrajhemu.github.io

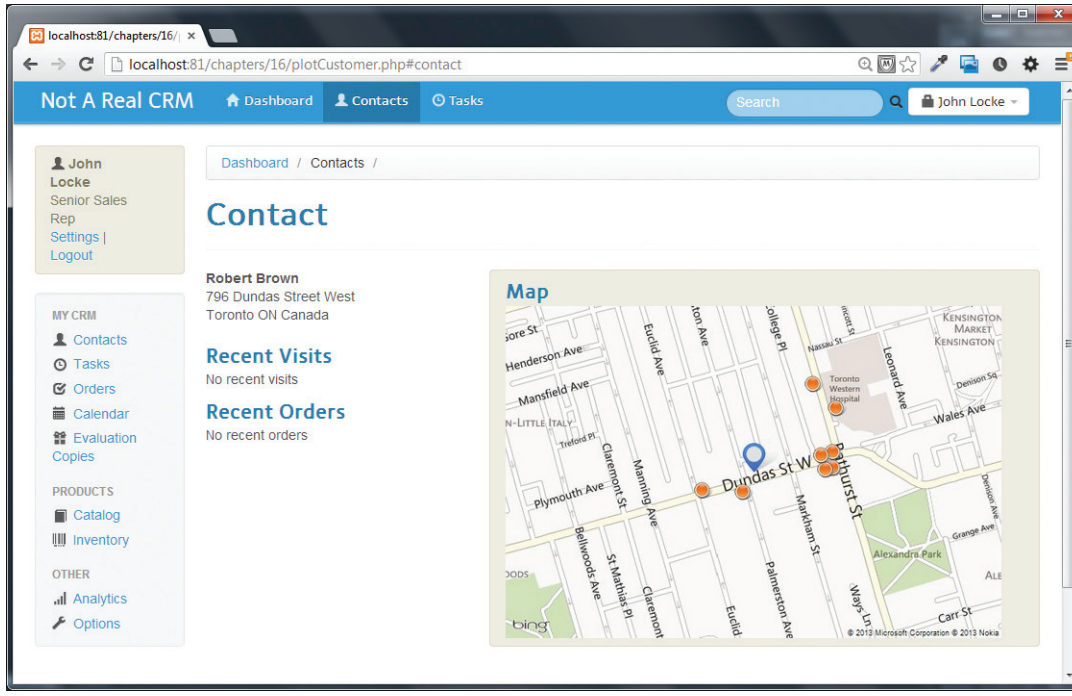


FIGURE 17.14 Finished page with map

NOTE

You may be wondering if it is possible to have a dynamic map (i.e., one in which the user can zoom and pan) instead of a static map. Dynamic maps require the interaction of JavaScript with external web services. In the last section of this chapter (which is on consuming web services asynchronously), we will use the Google Maps API to create a dynamic map that interacts with web services that we will create in the next section.

17.6 Creating Web Services

One of the significant advantages of REST web services in comparison to SOAP web services is that creating web services is relatively straightforward. Since REST services simply respond to HTTP requests, creating a PHP web service is only a matter of creating a page that responds to query string parameters and instead of returning HTML, it returns XML or JSON (or indeed any other format). As well, since a web service does not return HTML, our PHP page must also modify the `Content-type`

https://hemanthrajhemu.github.io

response header. A real-world web service would most likely also perform some type of identification or authentication.

17.6.1 Creating an XML Web Service

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Creating an XML Web Service in PHP

The first service we will create will be one that returns data from our Book Customer Relations Management database. You may think that there is not likely to be much public interest in such a web service (and you are probably correct in thinking so). However, it is important to recognize that not all web services are intended to be used by external clients. Many web services are intended to be consumed asynchronously by their own web pages via JavaScript. Indeed, in the next section we will be demonstrating precisely that functionality.

To begin, we should determine the methods our service will support and the format of the requests. This service will take the following format:

```
crmServiceSearchBooks.php?criteria=yyy&look=zzz
```

The `criteria` parameter will be used to specify what type of criteria (i.e., which field) we will use for the book search. This exercise will only support four values: `imprint`, `category`, `look`, and `subcategory`. The `look` parameter will be used to specify the actual value to search. For instance, if we had the following request:

```
crmServiceSearchBooks.php?criteria=subcategory&look=finance
```

It would be equivalent to the SQL search:

```
SELECT * FROM Books WHERE SubCategoryID=5
```

We will make use of the class infrastructure from Chapter 14 so that this example can focus on the creation of the web service. Listing 17.19 shows the XML structure that will be returned from the `crmServiceSearchBooks` service. It contains a number of `<book>` elements, each of which contains select information from the Books and Authors tables.

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book id="696">
    <isbns>
      <isbn10>0133140512</isbn10>
      <isbn13>9780133140514</isbn13>
    </isbns>
    <title>Entrepreneurial Finance</title>
    <authors>
      <author>
```

(*continued*)

https://hemanthrajhemu.github.io

```
        <lastname>Adelman</lastname>
        <firstname>Philip</firstname>
        <institution>DeVry University</institution>
      </author>
      <author>
        <lastname>Marks</lastname>
        <firstname>Alan</firstname>
        <institution>DeVry University</institution>
      </author>
    </authors>
    <category>Business</category>
    <subcategory>Finance</subcategory>
    <year>2014</year>
    <imprint>Prentice Hall</imprint>
    <pagecount>448</pagecount>
    <description>For courses in ...</description>
  </book>
  <book>...</book>
  ...
</books>
```

LISTING 17.19 XML to be returned from crmServiceSearchBooks service

The main algorithm for the service is quite straightforward. Since PHP already responds to HTTP requests, the main difference between developing a web service and a regular web page is that the web service doesn't return HTML. The algorithm (indeed the complete listing of `crmServiceSearchBooks.php`) is shown in Listing 17.20. Notice that there is no `<html>`; instead it contains just PHP code.

The most important thing to note in Listing 17.20 is the one emphasized line, which outputs the HTTP `Content-type` header. The `Content-type` header is used to

```
<?php
require_once('includes/setup.inc.php');
require_once('includes/funcSearchBooks.inc.php');

// array to be used for query string validation and extraction
$acceptedCriteria = array('imprint','category','subcategory');
// parallel array to be used for constructing appropriate SQL
// criteria
$whereClause = array('Imprint=?','CategoryName=?','SubcategoryName=?');

// tell the browser to expect XML rather than HTML
// NOTE: comment this line out when debugging
header('Content-type: text/xml');
```

https://hemanthrajhemu.github.io

```
// check query string parameters and either output XML or error
// message (in XML)
if ( isCorrectQueryStringInfo($acceptedCriteria) ) {
   outputXML($dbAdapter, $acceptedCriteria, $whereClause);
}
else {
   echo '<errorResult>Error: incorrect query string values</errorResult>';
}
?>
```

LISTING 17.20 The crmServiceSearchBooks.php service

specify the type of content that the browser will be receiving. The default MIME value for PHP pages is `text/html`. However, since the service is returning XML, we need to change this value to `text/xml`. This change does have ramifications for the developer, which are described in the nearby note. Most of the rest of the code for this example is shown in Listing 17.21.

NOTE

Changing the `Content-Type` header from its default `text/html` to `text/xml` can create some frustrating moments for the developer. If your PHP's error reporting settings are such that you expect to see PHP's error and warning messages, then these will cause some unusual output due to the `text/xml` header setting. Since PHP's warning and error messages are HTML, depending on the browser you use, you may see nothing (or only a very cryptic browser message) when one of these PHP's messages is sent. As the comment in Listing 17.20 indicates, the solution is to temporarily comment out the line that changes the `Content-Type` header, or refer directly to the log files of Apache, where the errors will still be readable.

```
<?php
/*
   Algorithm for outputting the XML for the books
*/
function outputXML($dbAdapter, $acceptedCriteria, $whereClause) {
   // get query string values and set up search criteria
   $criteria = $_GET['criteria'];
   $look = $_GET['look'];
   $index = array_search($criteria, $acceptedCriteria);

   // get the data from the database
```

(*continued*)

https://hemanthrajhemu.github.io

```
    $bookGate = new BookTableGateway($dbAdapter);
    $results = $bookGate->findByFromJoins( $whereClause[$index],
                                           Array($look) );

    // output the XML for the retrieved book data
    echo createXMLforBooks($results, $dbAdapter);

    $dbAdapter->closeConnection();
}


/*
   Checks if valid query string information was passed in GET or POST
*/
function isCorrectQueryStringInfo($acceptedCriteria) {
    if ( isCriteraPresent($acceptedCriteria) && isLookPresent()) {
       return true;
    }
    return false;
}

/*
  Checks for query string info that specifies which criteria to use
*/
function isCriteraPresent($acceptedCriteria)  {
    if ($_SERVER['REQUEST_METHOD'] == 'GET'
        && isset($_GET['criteria'])) {

       // now check criteria values are correct
       if ( in_array($_GET['criteria'],$acceptedCriteria) )
          return true;
       else
          return false;
    }
    return false;
}

/*
  Checks for query string info that specifies which criteria to use
*/
function isLookPresent()  {
    if ($_SERVER['REQUEST_METHOD'] == 'GET' && !empty($_GET['look']))
       return true;
    return false;
}
```

https://hemanthrajhemu.github.io

```
/*
   Return a string containing XML for book
*/
function createXMLforBooks($bookResults, $dbAdapter) {
  // will implement this function shortly
}
```

LISTING 17.21 The functions in the funcSearchBooks.inc.php file

Since we are using the class infrastructure from Chapter 14, the code in Listing 17.21 mainly consists of comments and query string validation. The function `createXMLforBooks()`, which will actually output the XML, was left unimplemented in this listing. Let us turn now to this function.

There are different ways to output XML in PHP. One approach would be to simply echo XML within string literals to the response stream:

```
echo '<?xml version="1.0" encoding="UTF-8"?>';
echo '<books>';
...
```

While this approach has the merit of familiarity, it will be up to the programmer to ensure that our page outputs well-formed and valid XML. The alternate approach would be to use one of PHP's XML extensions that were covered back in Section 17.2.2. This example will use the `XMLWriter` object, which is shown in Listing 17.22.

```
/*
   Return a string containing XML for book
*/
function createXMLforBooks($bookResults, $dbAdapter) {
   // first set up the XML writer
   $writer = new XMLWriter();
   $writer->openMemory();
   $writer->startDocument('1.0','UTF-8');
   $writer->setIndent(true);

   // create the root element
   $writer->startElement("books");

   // now loop through each book object in our collection and
   // write the appropriate XML for it
```

(*continued*)

https://hemanthrajhemu.github.io

```
    foreach ($bookResults as $book) {
       writeSingleBookXML($writer, $book, $dbAdapter);
    }

    // close root element
    $writer->endElement();
    // finish up writer
    $writer->endDocument();
    // return a string representation of the XML writer
    return $writer->outputMemory(true);
}
/*
 Writes XML for a single book
*/
function writeSingleBookXML($writer, $book, $dbAdapter) {
    $writer->startElement("book");
    $writer->writeAttribute("id", $book->ID);
    // write XML for the ISBN numbers
    writeIsbnsXML($writer, $book);

    $writer->startElement("title");
    $writer->text(htmlentities($book->Title));
    $writer->endElement();

    // write XML for the authors
    writeAuthorXML($writer, $book, $dbAdapter);

    $writer->startElement("category");
    $writer->text($book->Category);
    $writer->endElement();

    $writer->startElement("subcategory");
    $writer->text($book->Subcategory);
    $writer->endElement();

    $writer->startElement("year");
    $writer->text($book->CopyrightYear);
    $writer->endElement();

    $writer->startElement("imprint");
    $writer->text($book->Imprint);
    $writer->endElement();

    $writer->startElement("pagecount");
    $writer->text($book->PageCountsEditorialEst);
    $writer->endElement();
```

https://hemanthrajhemu.github.io

```
    $writer->startElement("description");
    $writer->text(htmlentities($book->Description));
    $writer->endElement();

    $writer->endElement();
}

/*
   Using the XML Writer, add information for a book's ISBN numbers
*/
function writeIsbnsXML($writer, $book) {
    $writer->startElement("isbns");
    $writer->startElement("isbn10");
    $writer->text($book->ISBN10);
    $writer->endElement();

    $writer->startElement("isbn13");
    $writer->text($book->ISBN13);
    $writer->endElement();
    $writer->endElement();
}

/*
  writes XML for a book's authors
*/
function writeAuthorXML($writer, $book, $dbAdapter) {
  // will implement this shortly
}
```

LISTING 17.22 Implementing createXMLforBooks using the XMLWriter

While a bit verbose, the `XMLWriter` is quite straightforward to use and has the advantage that it will generate well-formed XML. To make the code more readable and maintainable, the code for writing the various XML child elements is delegated to single-purpose functions such as `writeSingleBookXML()`, `writeIsbnsXML()`, and `writeAuthorXML()`.

You will notice that `writeAuthorXML()` has not yet been implemented. This one is slightly more complicated since the information needed for this element is not actually in the `Book` object; instead, we will need to retrieve the relevant authors from the `Authors` table as shown in Listing 17.23.

The web service is now complete. To test it, simply open a browser and request the `crmServiceSearchBooks.php` page with the appropriate query string parameters, as shown in Figure 17.15.

https://hemanthrajhemu.github.io

```
/*
   Using the XML Writer, add information for a book's authors
*/
function writeAuthorXML($writer, $book, $dbAdapter) {
   // retrieve authors for the current book
   $authorGate = new AuthorTableGateway($dbAdapter);
   $authorResults = $authorGate->getForBookId($book->ID);

   // now write <authors> collection
   $writer->startElement("authors");
   // loop through each author in the collection and output it
   foreach ($authorResults as $author) {
      writeSingleAuthorXML($writer, $author);
   }
   $writer->endElement();
}
/*
 Writes XML for a single author
*/
function writeSingleAuthorXML($writer, $author) {
   $writer->startElement("author");

   $writer->startElement("lastname");
   $writer->text($author->LastName);
   $writer->endElement();

   $writer->startElement("firstname");
   $writer->text($author->FirstName);
   $writer->endElement();

   $writer->startElement("institution");
   $writer->text($author->Institution);
   $writer->endElement();

   $writer->endElement();
}
```

LISTING 17.23 The writeAuthorXML() function

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Creating a JSON Web Service in PHP

17.6.2 Creating a JSON Web Service

Creating a JSON web service rather than an XML service is simply a matter of creating a JSON representation of an object, setting the `Content-type` header to indicate the content will be JSON, and then outputting the JSON object.

One potential problem is that there is not really a standard MIME type for JSON data. The standard `Content-type` is `application/json`. Unfortunately, some

https://hemanthrajhemu.github.io

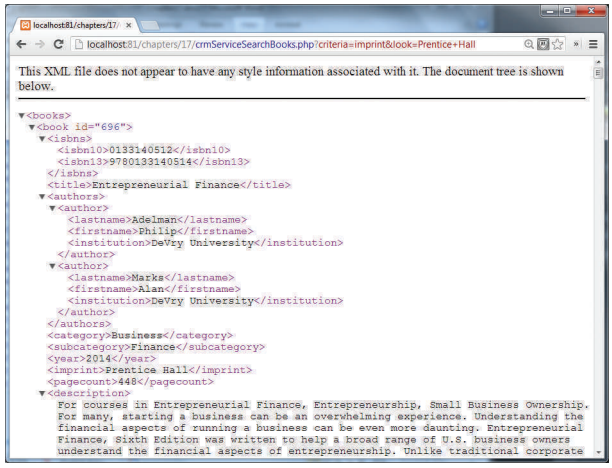


FIGURE 17.15 Testing the crmServiceSearchBooks.php service in the browser

firewalls may block content with the `application` prefix; as well, some versions of IE may display the file download dialog when requested within an `<iframe>` element. Thus, some developers will use the `text/plain` MIME type.

In this section, we will create another service, this time one that returns JSON and which returns title matches for books whose title begins with the same characters as the specified search string. In the last section of the chapter, we will asynchronously consume this service to create an autosuggest search box (i.e., a textbox that displays matches as the user enters text into the search box), as can be seen in Figure 17.17.

Since the built-in PHP `json_encode()` function does most of the work for us, our JSON service is simpler than the XML web service from the last section, as can be seen in Listing 17.24.

```
<?php
require_once('includes/setup.inc.php');
require_once('includes/funcFindTitles.inc.php');

// Tell the browser to expect JSON rather than HTML
header('Content-type: application/json');

if ( isCorrectQueryStringInfo() ) {
   outputJSON($dbAdapter);
}
else {
```

https://hemanthrajhemu.github.io

```
    // put error message in JSON format
    echo '{"error": {"message":"Incorrect query string values"}}';
}

function outputJSON($dbAdapter) {
    // get query string values and set up search criteria
    $whereClause = 'Title Like ?';
    $look = $_GET['term'] . '%';

    // get the data from the database
    $bookGate = new BookTableGateway($dbAdapter);
    $results = $bookGate->findByFromJoins($whereClause, Array($look) );

    // output the JSON for the retrieved book data
    echo json_encode($results);

    $dbAdapter->closeConnection();
}
```

LISTING 17.24 JSON crmServiceFindTitleMatches service

Unfortunately, if we request this service, it will return an empty JSON document. Why is this the case?

The problem resides in the fact that we are passing an array of custom objects to the `json_encode()` function. This function does not "know" how to create the JSON representation of a custom object. For this function to work, the class of the custom object being converted must provide its own implementation of the `JsonSerializable` interface. This interface contains only the single method `jsonSerialize()`. In this web service, we are outputting JSON for objects of the `Book` class, so this class will need to implement this method, as shown in Listing 17.25. We've chosen to use the key of **value** for the title, so that it will work with our jQuery plug-in in the next section.

```
class Book extends DomainObject implements JsonSerializable
{
    ...
    /*
     This method is called by the json_encode() function that is
     part of PHP
    */
    public function jsonSerialize() {
        return ['id' => $this->ID, 'value' => $this->Title];
    }
}
```

LISTING 17.25 Adding jsonSerializable() to Book class

https://hemanthrajhemu.github.io

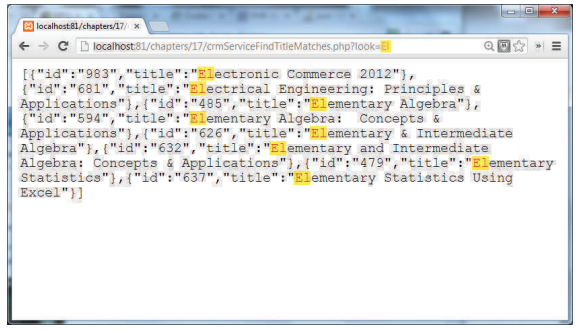


FIGURE 17.16 Testing the crmServiceFindTitleMatches.php service in the browser

Now the web service should work correctly, and the output can be seen in Figure 17.16.

17.7 Interacting Asynchronously with Web Services

Although it's possible to consume web services in PHP, it's far more common to consume those services asynchronously using JavaScript. With JavaScript and jQuery's parsing libraries, it's easy to parse XML and JSON replies and then update the user interface asynchronously.

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Consuming a Web Service in JavaScript

As you might guess, the details on how to consume services depend on whether they are XML or JSON encoded (or not encoded at all), since you will have to make use of different jQuery functions in each case. This section will cover consumption of a simple JSON object for the autosuggest feature you just created for autocomplete and then move on to an example with asynchronous consumption of location services inside Google Maps.

When using client-side requests for third-party services, there's also the advantage of distributing requests to each client rather then making all requests from your own server's IP address. Although API keys are still sometimes required, often you can achieve more requests per day, because the requests from clients count toward their IP address's total, not your server's.

17.7.1 Consuming Your Own Service

To achieve the nice dropdown autocomplete box illustrated in Figure 17.17, you must not only have your own web service in PHP, but associated JavaScript code to request data from your web service and display it correctly.

https://hemanthrajhemu.github.io

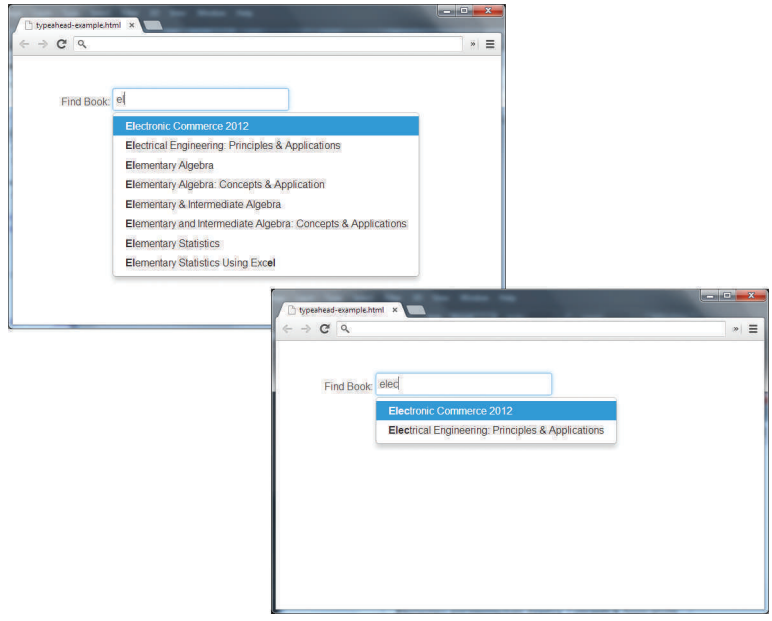


FIGURE 17.17 Example auto suggest text box

The code to connect the front-end client page to the web service you built is shown in Listing 17.26. It listens for changes to an input box with id *search*. With each change the code makes an asynchronous get request to the *source* URL, which in this case is the script in Listing 17.24 that returns JSON results. Those results are then used by autocomplete to display nicely underneath the input box. This takes advantage of the autocomplete jQuery extension, which may have to be included separately in the head of the page.

```
$("#search").autocomplete({
    // the URL of service, with the search text transmitted in the
    // term= field
    source:"crmServiceFindTitleMatches.php?",
    minlength:1,    //how many characters required before querying
    delay:1         //delay to prevent multiple events
});
```

LISTING 17.26 Autocomplete jQuery plug-in refreshes the list of suggestions to choose from

The biggest advantage of using your own web service is that you can change it to meet your needs. In this case, the jQuery plug-in requires that the query string to the web service contain the *key* **term** associated with the *value* of the search box.

https://hemanthrajhemu.github.io

```
crmServiceFindTitleMatches.php?term=el
```

Since you wrote the web service, your script already does that!

17.7.2 Using Google Maps

While you might be able to define some pretty good web services yourself, there are many services out there that provide not only web services to consume but platforms to consume them into. Google Maps is the industry standard for web-mapping applications, and provides some very easy-to-use APIs to work with. With Google Maps, you can leverage users' experiences with those tools to build an impressive application in little time.

HANDS-ON EXERCISES

LAB 17 EXCERCISE
Displaying a Google Map Using JavaScript and PHP

To demonstrate using Google Maps with our own web service, consider our photo-sharing website. We will show you how to build a map view that plots user photos onto a map using the location information associated with the image.

PRO TIP

The EXIF data embedded in many image formats allows us to extract the latitude and longitude from the image directly. In PHP we can easily check for embedded data using `exif_read_data` as follows:

```
//extract the lat/lng in degrees minutes and seconds
$exif=exif_read_data($filename);

//extract the lat/lng in degrees minutes and seconds
$gps['LatDegree']=exif['GPSLatitude'][0];
$gps['LatMinutes']=exif['GPSLatitude'][1];
$gps['LatSeconds']=exif['GPSLatitude'][2];
$gps['LongDegree']=exif['GPSLongitude'][0];
$gps['LongMinutes']=exif['GPSLongitude'][1];
$gps['LongSeconds']=exif['GPSLongitude'][2];
```

To begin using Google Maps, you must do three things

1. Include the Google Maps libraries in the `<head>` section of your page.
2. Define `<div>` elements that will contain the maps.
3. Initialize instances of `google.maps.Map` (we will call it `Map`) in JavaScript and associate them with the `<div>` elements.

https://hemanthrajhemu.github.io

Listing 17.27 defines a function in PHP that creates a and then initializes it using the passed-in latitude and longitude. When called as in the code, you get a map centered on Mount Royal University in Calgary as shown in Figure 17.18. The size and shape of the map are controlled through CSS while the options are all controlled at initialization.

```
<!DOCTYPE html>
<html>
<head>
   <script src="https://maps.googleapis.com/maps/api/js?v=3.
      exp&sensor=false"></script>
   <script src="http://code.jquery.com/jquery.js"></script>
</head>
<body>

<?php

function getGoogleMap($imageID, $latitude, $longitude) {
   return "<script>
          $(document).ready(function() {
             var map$imageID;
             var mapOptions = {
                 zoom:14,
                 center:new google.maps.LatLng($latitude,$longitude),
                 mapTypeId:google.maps.MapTypeId.ROADMAP
             };
             map$imageID = new google.maps.Map(
                             document.getElementById
                                ('map-canvas$imageID'),mapOptions);
          });
          </script>
          <div style='width: 400px; height: 400px;'
          class='map-canvas' id='map-canvas$imageID'></div>";
}

echo getGoogleMap(1, 51.011179,-114.132866);
?>

</body>
</html>
```

LISTING 17.27 Web page to output one map centered on Mount Royal University

Note that the Map object's constructor takes a MapOptions object. While beyond the scope of this chapter, there are dozens of options you can control

https://hemanthrajhemu.github.io

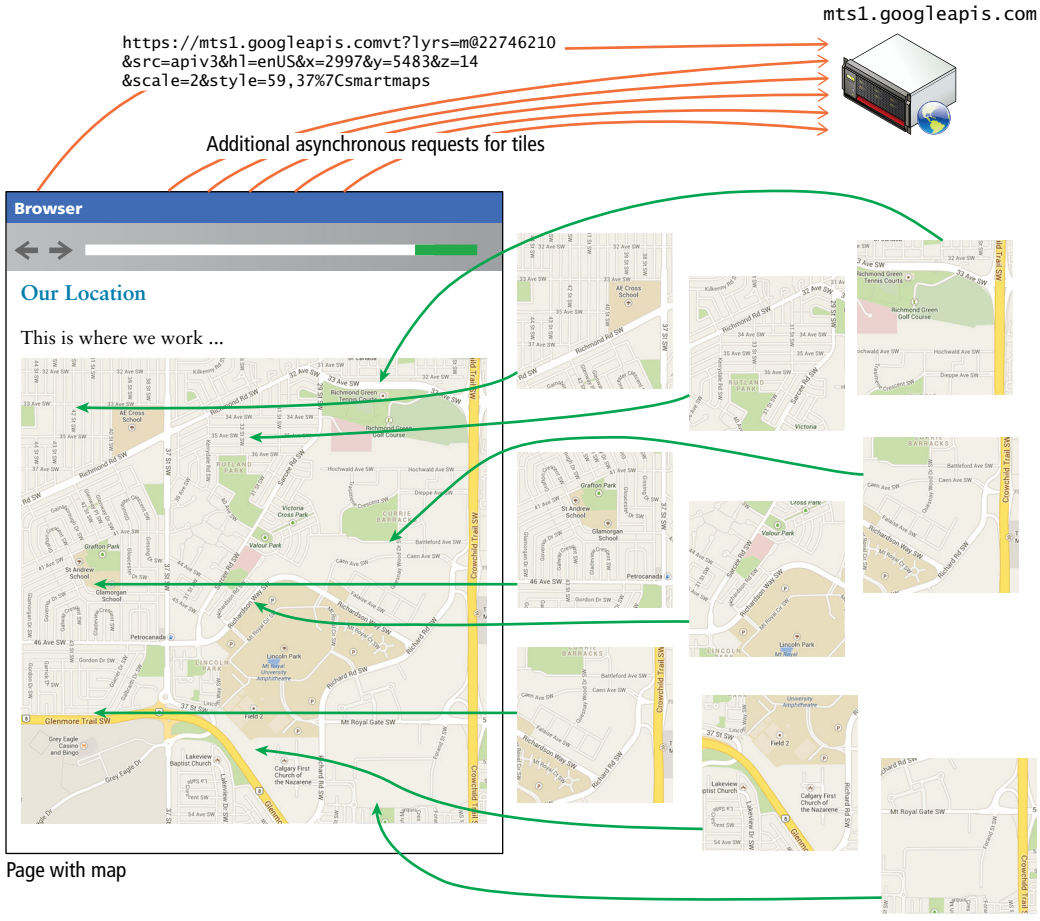


FIGURE 17.18 Visualization of the asynchronous requests for tiles made by Google Maps

about the map through the `MapOptions` object including whether it's draggable, has keyboard control, satellite imagery, and more. You make these decisions up front at initialization time, and do not change them while the map is loaded.

What's interesting in terms of web services is that this basic page with just a simple map is actually using asynchronous web services in the background to load the tiles that make up the background of the map. That means whenever the map's view changes (or first loads), those image requests also go out to Google as illustrated in Figure 17.18.

To demonstrate a more advanced usage of Google Maps, consider a page such as the one in Figure 17.19 for our photo-sharing site, which shows all photos you've uploaded as markers on a map. Since you might have thousands of photos uploaded, it wouldn't

https://hemanthrajhemu.github.io

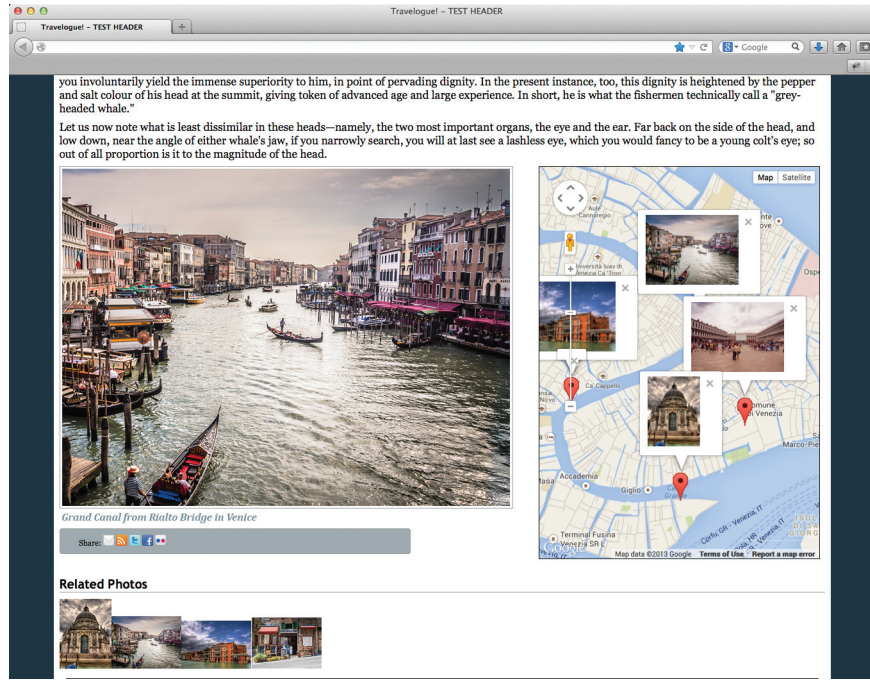


FIGURE 17.19 Screenshot of a mashup with locations of images plotted on the map

be efficient to load markers for images you can't even see. Instead our application makes use of an internal web service to return JSON containing all the images within view.

Every time the view changes (because the user drags the map or changes zoom level), your application must ask your web service for the list of images in the new view range. The JavaScript, shown in Listing 17.28, must attach a listener to the event that occurs when the view changes (`bounds_changed`) and trigger an asynchronous `GET` request to our web service (`images.php`). While implementation of that service is left as an exercise, the resulting JSON object might look like the one shown below.

```
{"images":[{"Title":"Venice","Latitude":"45.435124","Long
itude":"12.328055","ImageURL":"\/UPLOADS\/travel-images\/
medium\/9494470337.jpg"}, {"Title":"Santa Maria della Salute,
Venice","Latitude":"45.431003", "Longitude":"12.334766","ImageURL":
"\/UPLOADS\/travel-images\/medium\/9494472443.jpg"},{"Title":"Venice
View From Ponte di Rialto", "Latitude":"45.437975",
"Longitude":"12.335871", "ImageURL":"\/UPLOADS\/travel-images\/
medium\/9494464567.jpg"}, {"Title":"St Marks Square in Venice",
"Latitude":"45.434233", "Longitude":"12.338693","ImageURL":"\/
UPLOADS\/travel-images\/medium\/9494475161.jpg"}]}
```

https://hemanthrajhemu.github.io

We must then process the JSON from the server by clearing all markers and then creating a marker on the map for each image returned from the request.

```
 var markersArray = [];          //array to track markers on map

//function to create marker and info window and add to map
function createMarker(map, lat, lon, src, title){
   var pt = new google.maps.LatLng(lat,lon);     //latlng object

   //create an info window (the thing that pops up).
   var infowindow = new google.maps.InfoWindow({
      content: '<img src=\'http://funwebdev.com/'+src+'\'
                  height=100px/>'
   });

   //define a marker based on lat, lng
   var marker=new google.maps.Marker({
        position: pt,
        map: map,
        title: title
   });

   // Attach a listener to the click of each marker
   google.maps.event.addListener(marker, 'click', function() {
      infowindow.open(map,marker);
   });
   markersArray.push(marker);
}

// Deletes all markers in the array by removing references to them
function deleteOverlays() {
   if (markersArray) {
        for (i in markersArray) {
           markersArray[i].setMap(null);
        }
        markersArray.length = 0;
   }
}

$(document).ready(function(){
   var map;
   var mapOptions = {
       zoom: 14,
       center: new google.maps.LatLng($latitutde, $longitude),
       mapTypeId: google.maps.MapTypeId.ROADMAP
```

(*continued*)

https://hemanthrajhemu.github.io

```
    };
    map$ = new google.maps.Map(
       document.getElementById('map-canvas'), mapOptions);
    // Attach a listener to the bounds_changed event of the map
    google.maps.event.addListener(map$imageID, 'bounds_changed',
       function(){
       $.get('images.php?ne='+ map.getBounds().getNorthEast()
          +'&sw='+map$imageID.getBounds().getSouthWest(),
            function(data) {
            deleteOverlays();             //delete all old markers.
            var json = jQuery.parseJSON(data);
            for (var i=0; i < json.images.length;i++){
               createMarker(map,
                  json.images[i].Latitude,
                  json.images[i].Longitude,
                  json.images[i].ImageURL
                  json.images[i].Title);
            }
       }); //END $.get
}); //END addListener
});
```

LISTING 17.28 Code to define the dynamic map from Figure 17.19

PRO TIP

Marker management is an advanced mapping topic that you should consider if you are going to have lots of markers on a map. Our simplistic code that deletes all overlays is inefficient and re-creates the same markers over and over again. Instead the new markers should be compared with the old before new ones are created.

17.8 Chapter Summary

In this chapter we have covered the creation, consumption, and techniques of web services. From XML through JSON, you saw how markup allows data to be transferred between machines in a standardized way. PHP and JavaScript libraries allow for easy server- or client-side service consumption, giving you choices in how you want to implement your application. Finally we consumed our web services together with Google Maps services in a simple mashup that illustrated how web services can work together.

https://hemanthrajhemu.github.io

17.8.1 Key Terms

authentication
DOM extension
event or pull approach
geocoding
identity
in-memory approach
JSON
mashup
node
REST
reverse geocoding
root element
service
service-oriented architecture
service-oriented computing
SimpleXML
valid XML
web services
well-formed XML
XML declaration
XML parser
XMLReader
XPath
XSLT

17.8.2 Review Questions

1. What is well-formedness and validity in the context of XML? How do they differ?
2. What is XSLT? How can it be used in web development?
3. Using the XML document shown in Figure 17.5, what would be the XPath expressions for selecting artists from France? For selecting paintings whose artists are from France?
4. What are the in-memory and the event approaches to XML processing? How do they differ? What are some examples of each approach in PHP?
5. Imagine that you are asked to provide advice on implementing web services for a site. Discuss the merits and drawbacks of SOAP- and REST-based web services and for XML versus JSON as a REST data format.

17.8.3 Hands-On Practice

PROJECT 1: Book Rep Customer Relations Management

DIFFICULTY LEVEL: Basic

HANDS-ON EXERCISES
PROJECT 17.1

Overview

Demonstrate your ability to read in and display an XML file in PHP along with the ability to filter that XML data using XPath expressions.

Instructions

1. You have been provided with an XML file named employees.xml. Examine this file.
2. Alter filter-employees.php so that it reads in employees.xml using whichever method you wish (you will find that SimpleXML is the easiest) and displays some of its information in a table as shown in Figure 17.20.
3. Add a simple form that allows the user to enter in an XPath expression that filters the XML data using XPath as shown in Figure 17.20.

Test

1. Test with a variety of XPath expressions.

https://hemanthrajhemu.github.io

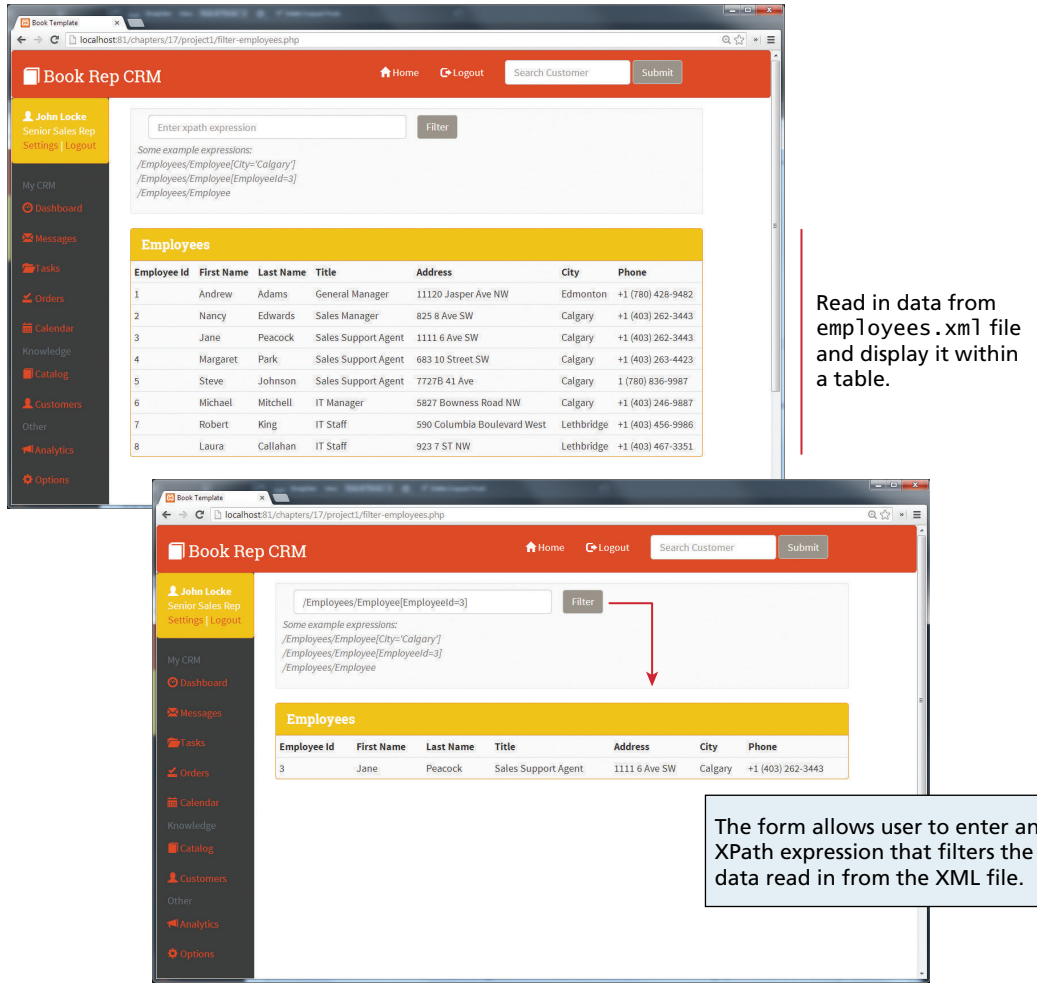


FIGURE 17.20 Completed Project 1

https://hemanthrajhemu.github.io

PROJECT 2: Share Your Travel Photos

DIFFICULTY LEVEL: Intermediate

Overview

Demonstrate your ability to consume XML- and JSON-based services. You will be modifying two files: single-image.php and travel-country.php.

HANDS-ON EXERCISES
PROJECT 17.2

Instructions

1. Add a panel to single-image.php that displays related Flickr images using the Flickr web service. As in the example from Section 17.5.1, you should use the `flickr.photos.search` method. The search term will be the country name of the `TravelImage` record being displayed by the page (see Figure 17.21).
2. Add a panel to single-image.php that displays a static road map with the latitude and longitude of the current `TravelImage` record indicated using a marker (or pushpins in Bing terminology) using the Bing Web Service.
3. Modify the travel-country.php page so that it displays the country information as shown in Figure 17.21. Notice the specific country is indicated by the `ISO` query string parameter.
4. Add a panel that displays the `TravelImages` for the current country.
5. Add a panel that displays a static map for the country using the Bing Web Service. This map will contain markers (or pushpins in Bing terminology) for the locations of each `TravelImage` for the current country. Unlike the example from Section 17.5.2, you will not be displaying a road map for a specific latitude and longitude. Instead, you will display a map with pushpins (one for each `TravelImage` in the country) that does not specify a center point. See the Bing Maps API documentation at http://msdn.microsoft.com/en-us/library/ff701724.aspx for additional guidance.
6. For additional credit, try replacing the static Bing Maps with a dynamic Google map.

Test

1. You can use browse-images.php to help find travel images to test. The country links on the left side of this and other pages can be used to test various countries.

PROJECT 3: Art Store

DIFFICULTY LEVEL: Advanced

Overview

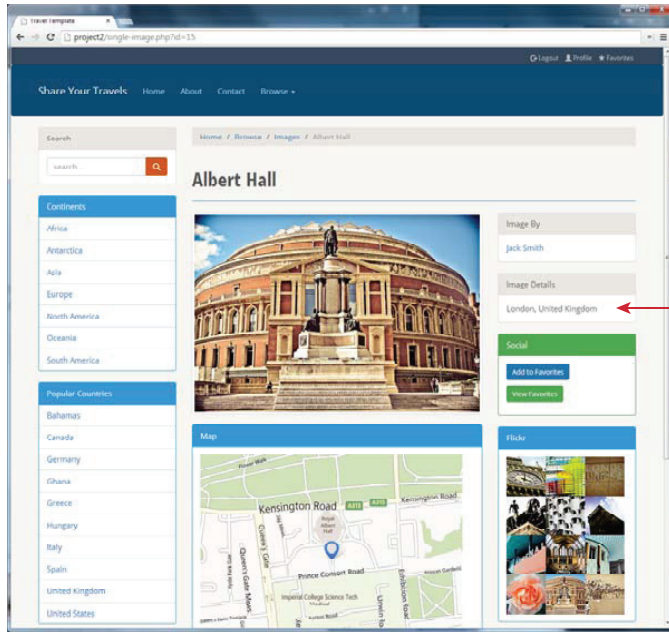
Demonstrate your ability to asynchronously consume XML- and JSON-based services. You will be modifying the file display-art-work.php.

HANDS-ON EXERCISES
PROJECT 17.3

Instructions

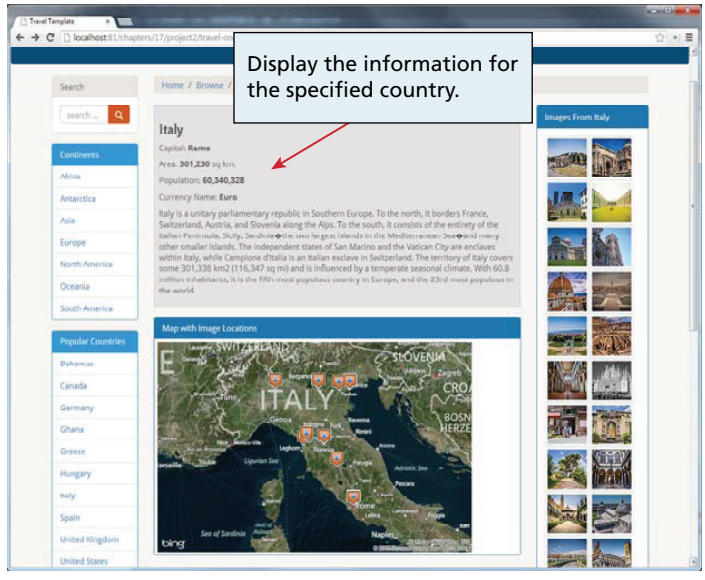
1. Create a web service that returns matching art work titles similar to that shown in Section 17.6.2 (except it is performing searches on the `ArtWorks` table).
2. Add autosuggest capability to the search text box. It should asynchronously make use of the search art titles web service created in step 1.

https://hemanthrajhemu.github.io



Display 12 items from the Flickr web service. Search on the city name of the travel image.

Use the Bing Maps service to display a static map using the latitude and longitude field values for the travel image.



Display travel images for the specified country.

Use the Bing Maps service to display a static map for the selected country. Add markers to the map for the locations for each travel image for the country.

FIGURE 17.21 Completed Project 2

https://hemanthrajhemu.github.io

3. Display the museum information from the `Gallery` table in the Museum Details collapsible panel. You will need to add the appropriate classes (`Gallery` and `GalleryTableGateway`) to the model to do this step.
4. Display the location of the gallery using Google Maps (see Figure 17.22). Add a marker to the map that shows the exact location of the gallery.
5. Use the accordion functionality within collapse.js to perform an asynchronous fetch of the map when the user expands the map panel. Information about collapse.js can be found within the Bootstrap documentation.

Test

1. Verify it works with a variety of artworks.

FIGURE 17.22 Completed Project 3

https://hemanthrajhemu.github.io

17.8.4 References

1. W3C. [Online]. http://www.w3.org/XML/.
2. W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)." [Online]. http://www.w3.org/TR/REC-xml/.
3. W3C, "The Extensible Stylesheet Language Family (XSL)." [Online]. http://www.w3.org/Style/XSL/.
4. W3C, "XML Path Language (XPath)," 16 November 1999. [Online]. http://www.w3.org/TR/xpath/.
5. jQuery, "jQuery.parseXML()." [Online]. http://api.jquery.com/jQuery.parseXML/.
6. PHP, "XML Parser." [Online]. http://php.net/manual/en/book.xml.php.
7. PHP, "JavaScript Object Notation." [Online]. http://php.net/manual/en/book.json.php.

https://hemanthrajhemu.github.io