# Module – 5

- Managing State
  - The Problem of State in Web Applications
  - Passing Information via Query Strings
  - Passing Information via the URL Path
  - Cookies
  - Serialization
  - Session State
  - HTML5 Web Storage
  - Caching
- Advanced JavaScript and jQuery
  - JavaScript Pseudo-Classes
  - jQuery Foundations
  - AJAX
  - Asynchronous File Transmission
  - Animation
  - Backbone MVC Frameworks
- XML Processing and Web Services
  - XML Processing
  - JSON
  - Overview of Web Services.

Explain why managing state is a problem in web applications.

## The Problem of State in Web Applications

**HTTP is a stateless** protocol, which means that the connection between the browser and the server is lost once the transaction ends. Because a **stateless** protocol does not require the server to retain session information or status about each communications partner for the duration of multiple requests. A Web page is recreated every time it is posted back to the server. In traditional Web programming, this means that all the information within the page and information in the controls is lost with each round trip. So it is difficult for the server to maintain the client information. To overcome this, the information is to be send to the server using:

- Query strings
- Cookies

why is state is a problem in web applications explain.

**Passing Information via Query Strings :** A web page can pass query string information from the browser to the server using one of the two methods: a query string within the URL (GET) and a query string within the HTTP header (POST). Figure 13.4 reviews these two different approaches.

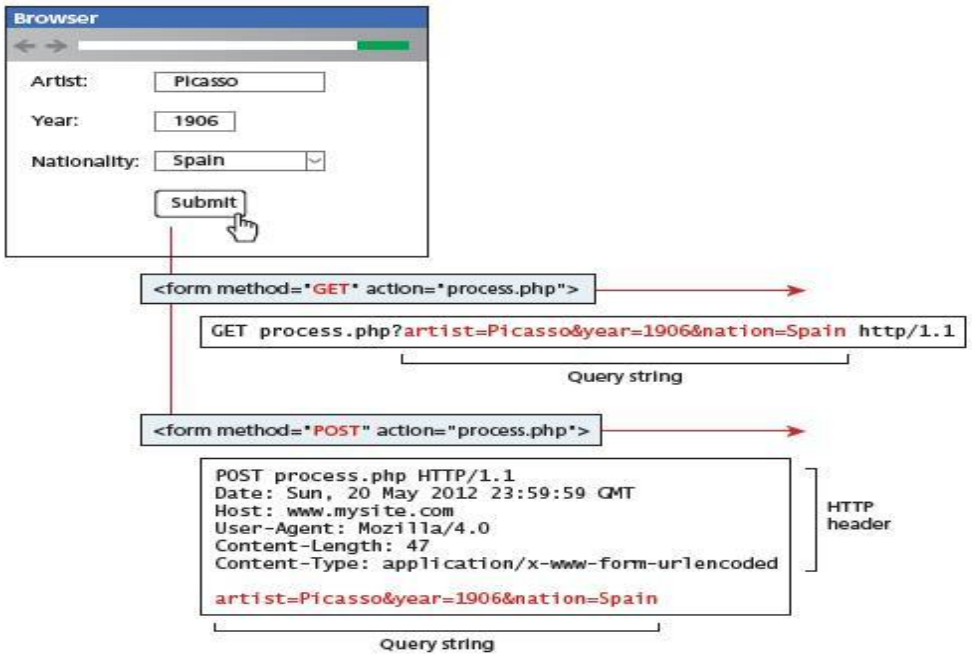



FIGURE 13.4 Recap of GET versus POST

## Passing Information via the URL Path

Passing information via URL is very useful in 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 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.**

URL rewriting is a way of appending data at the end of URL. It is a way to rewrite the dynamic URL into a static one.

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

**URL Rewriting in Apache and Linux**

what are http cookies ?how do u handle them in php?

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

## Cookies

With suitable PHP scripts, explain creating and reading cookies ---->imp

Cookies are a client-side approach for persisting state information. They are name=value pairs that are saved within one or more text files that are managed by the browser. 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 web-site 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.

## 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).

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).

**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 tele-visions that used a cookie-based shopping cart.

discuss
1. session cookies
2. persistent cookies
3. session state

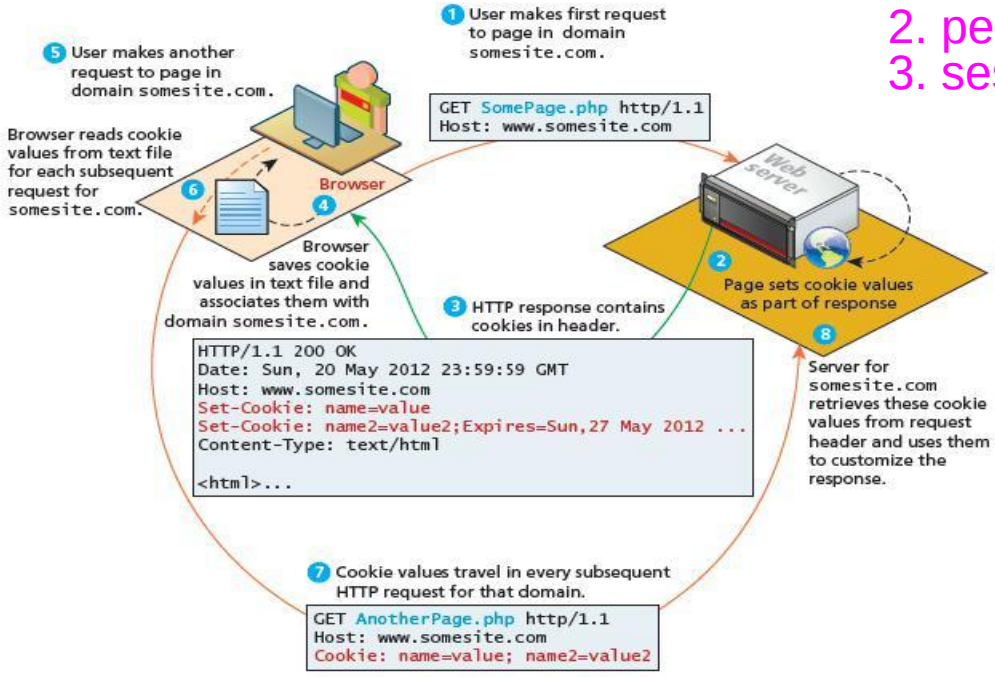



FIGURE 13.6 Cookies at work

The site placed not only the product 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.

## 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 .

```
<?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

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
   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

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.

8. Illustrate with an example how can we add methods to existing classes, like String or array?

## Serialization

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.

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.

Illustrate Serialization in PHP with suitable example

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

LISTING 13.3 The Serializable interface

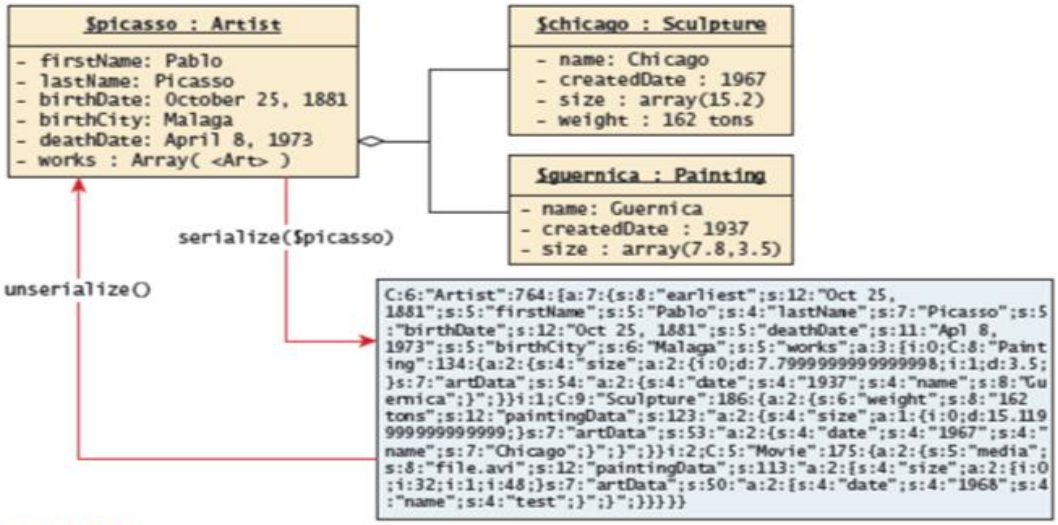



FIGURE 13.7 Serialization and deserialization

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.

```php
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
                      );
                );
    }
```

```php
    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:"Oct25,1881";s:5:"first Name"; s:5:"Pablo"; s:4:"lastName"; s:7:"Picasso"; s:5:"birthDate"; s:13:"Oct25, 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);
```

- **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.

## Session State

with suitable php scripts explain checking session existence and accessing session state.

**Session state** is a method to keep track of the user **session** during a series of HTTP requests. 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.

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 ashopping cart. While shopping carts could be implemented via cookies or query string parameters, it would be quite complex and cumbersome to do so.

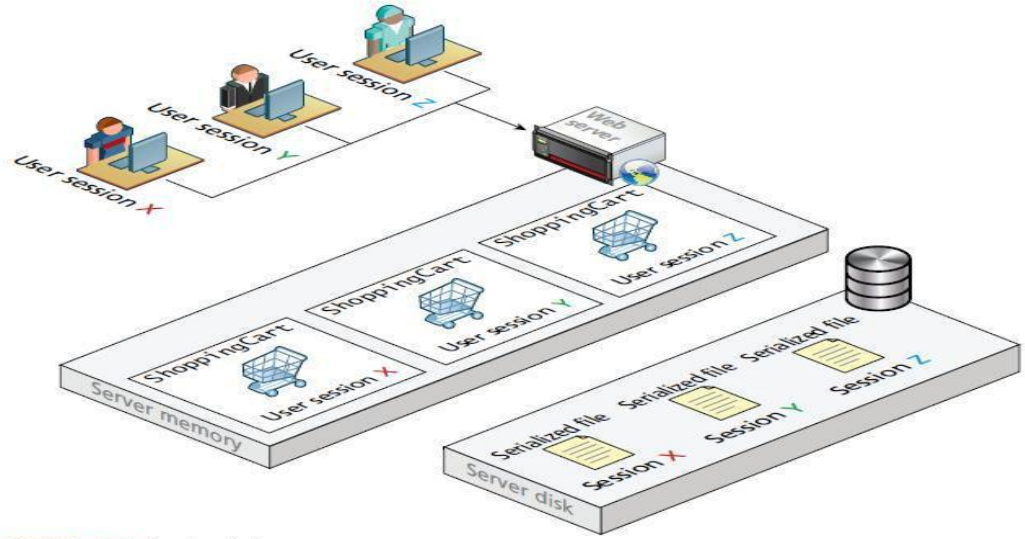



FIGURE 13.8 Session state

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. To use sessions in a script, you must call the session_start() function at the begin-ning 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
<?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.

```
<?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

In this example ShoppingCart is a user-defined class. Since PHP sessions are serialized into files, one must ensure that any classes stored into sessions can be serialized and deserialized, and that the class definitions are parsed before calling session_start().

**How Does Session State Work?**

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.

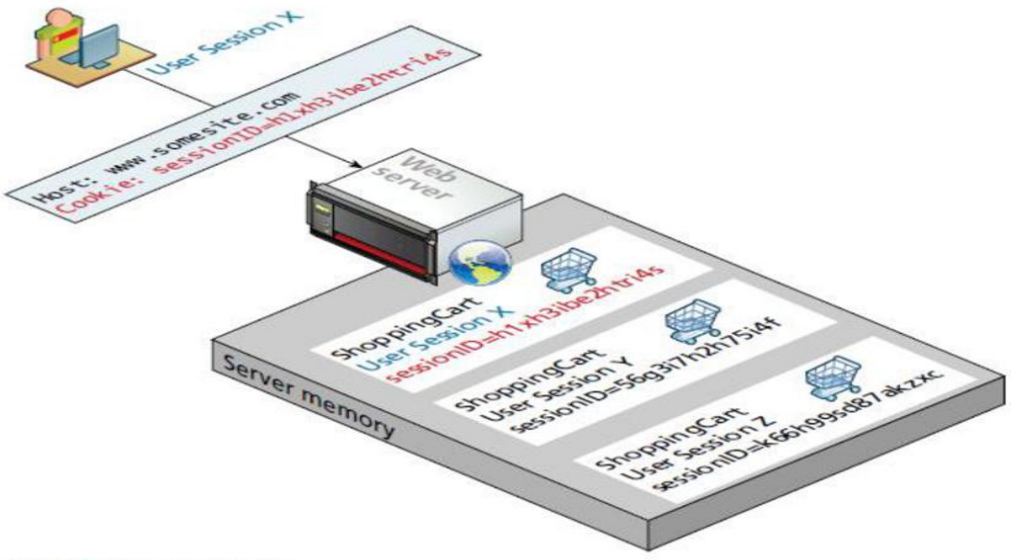



FIGURE 13.9 Session IDs

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.

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. 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.

**Session Storage and Configuration**

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.

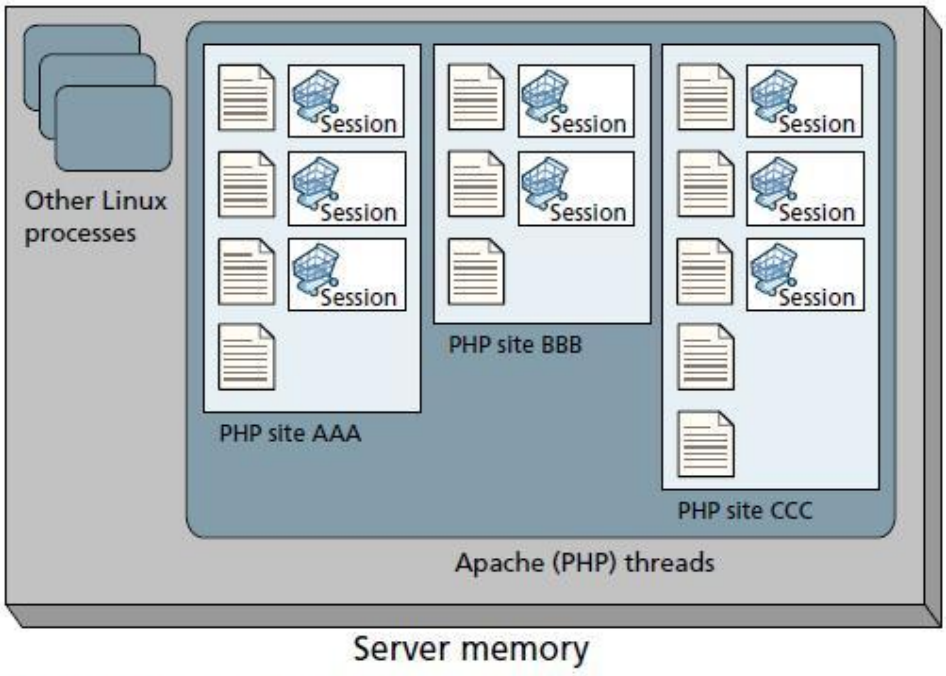



FIGURE 13.10 Applications and server memory

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 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 degradation in performance compared to memory storage, but the advantages, it was decided, out-weigh 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.

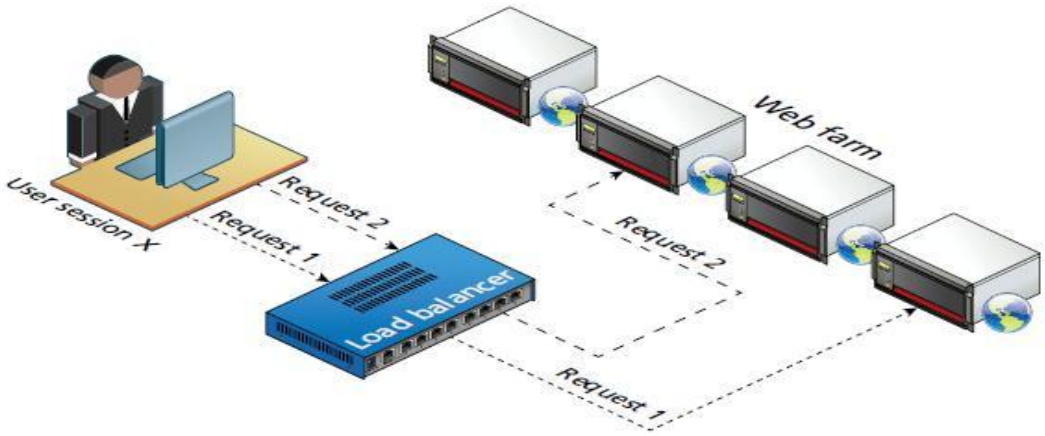



FIGURE 13.11 Web farm

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

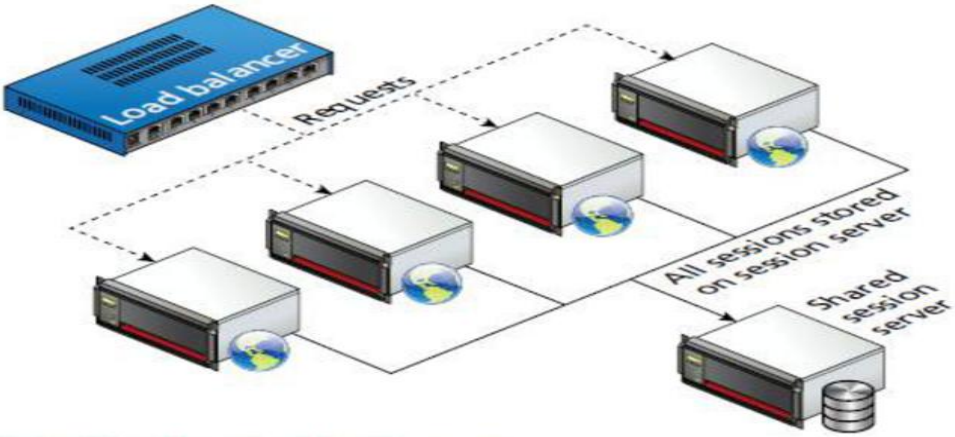



FIGURE 13.12 Shared session provider

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 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

6. What is web storage in HTML5? How does it differ from HTTP cookies?

## 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.

Just as there were two types of cookies, there are two types of global web stor-age 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.

### 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.

```
<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

## 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 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.

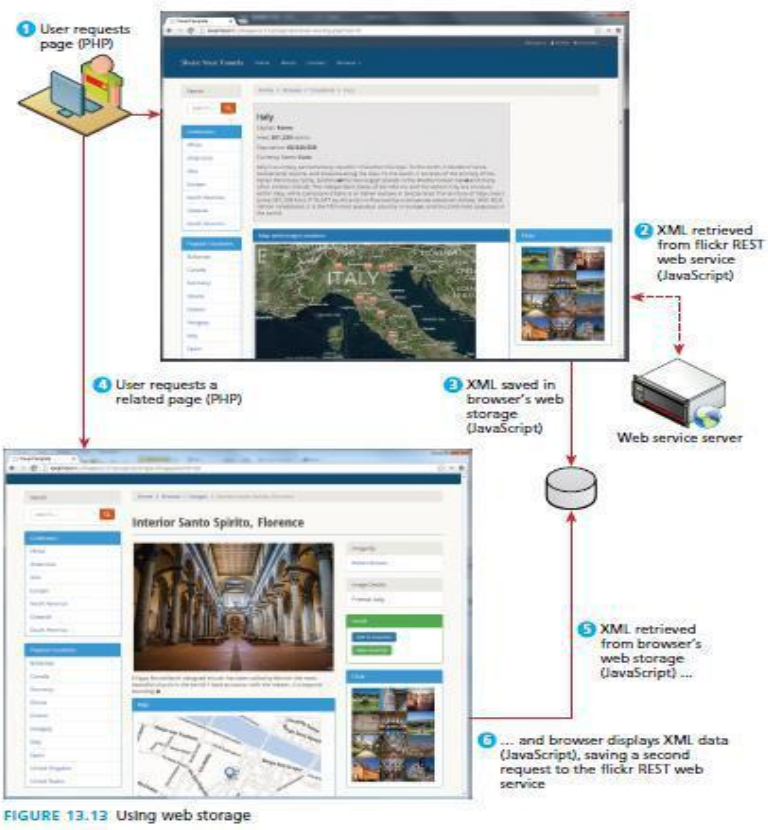



FIGURE 13.13 Using web storage

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.

## Caching

Discuss caching in the context of web applications.Explain the two types of caching. What benefit does it provide?

Caching is a vital way to improve the performance of web applications. 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.



explain different types od caching used to improve performance of web application

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.

- **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. 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.

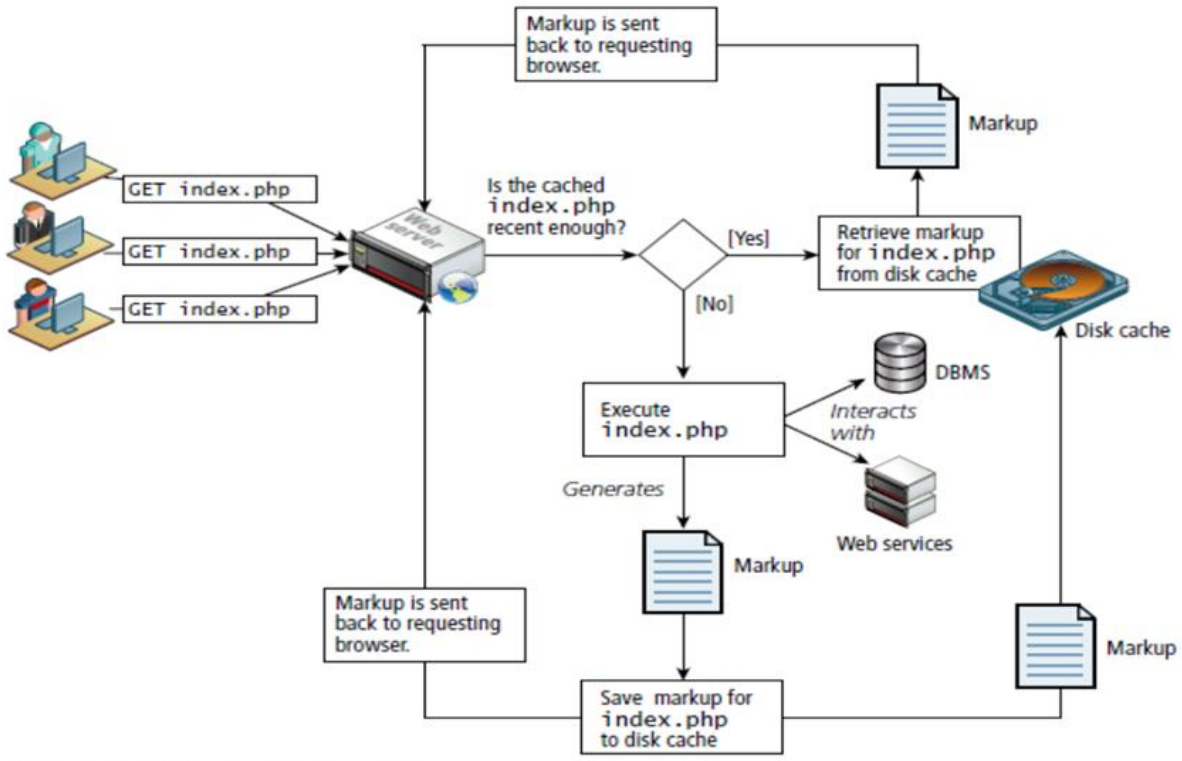



FIGURE 13.14 Page output caching

Page caching is not included in PHP by default, which has allowed a market-place 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.

- **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. 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. Listing 13.10 illustrates a typical use of memcache.

```php
<?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

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 per-formance 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.

# Advanced JavaScript and jQuery

## JavaScript Pseudo-Classes

explain javascript pseudo-classes with examples ---8M

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 meth-ods, 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.

- **Using Object Literals**

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:

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";

- **Emulate Classes through Functions**

```
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

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. 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**

8. Illustrate with an example how can we add methods to existing classes, like String or array?

```
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

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. 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.

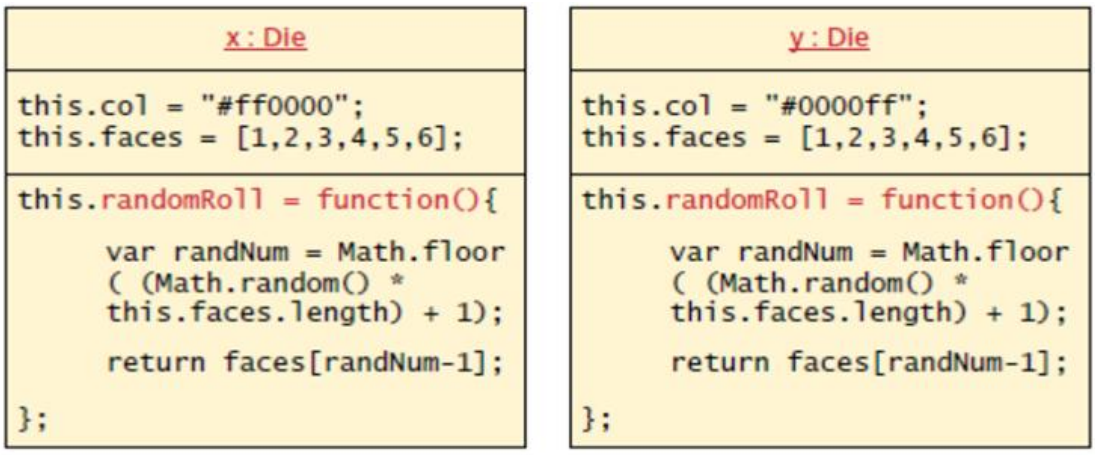



FIGURE 15.1 Illustrating duplicated method definition

.Write a short note on
1) prototypes in Javascript
2)Backbone.js MVC framework.

**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. This definition is better because it defines the method only once, no matter how many instances of Die are created.

```
// 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

**Prototype property** is basically an object (also known as Prototype object), where we can attach methods and properties in a prototype object, which enables all the other objects to inherit these methods and properties.

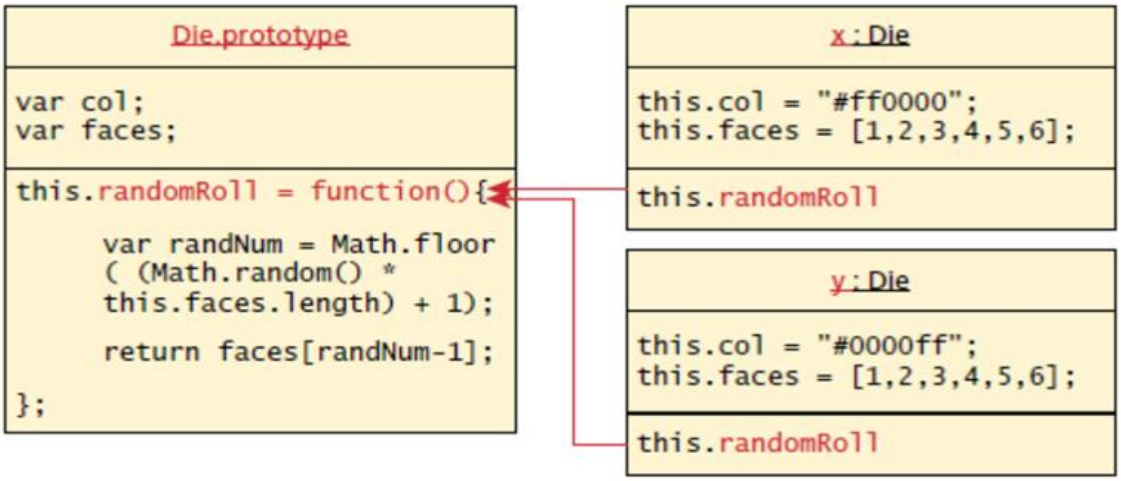



FIGURE 15.2 Illustration of JavaScript prototypes as pseudo-classes

## jQuery Foundations

**jQuery** is a lightweight, "write less, do more", JavaScript library. The purpose of **jQuery** is to make it much easier to use JavaScript on your website. **jQuery** takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code. It eliminates the need of programmers to write browser specific code.

A library or framework is software that you can utilize in your own software, which provides some common implementations of standard ideas. A web frame-work 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.

jQuery bills itself as the write less, do more framework. 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.

- **Including jQuery in Your Page**

```
<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

. List all jquery selectors,explain any two with suitable examples

**jQuery Selectors:** ---->explain any 4

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. 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.

1. **Basic Selectors :** The four basic selectors were defined back in Chapter 3, and include the universal selector, class selectors, id selectors, and elements selectors. To review:
   - $("*") Universal selector matches all elements (and is slow).
   - $("tag") Element selector matches all elements with the given element name
   - $(".class") Class selector matches all elements with the given CSS class.
   - $("#id") Id selector matches all elements with a given HTML id attribute.

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

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

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.

## 2. 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 browsers implemented it. jQuery overcomes those browser limitations, providing the ability to select elements by attribute.

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]).

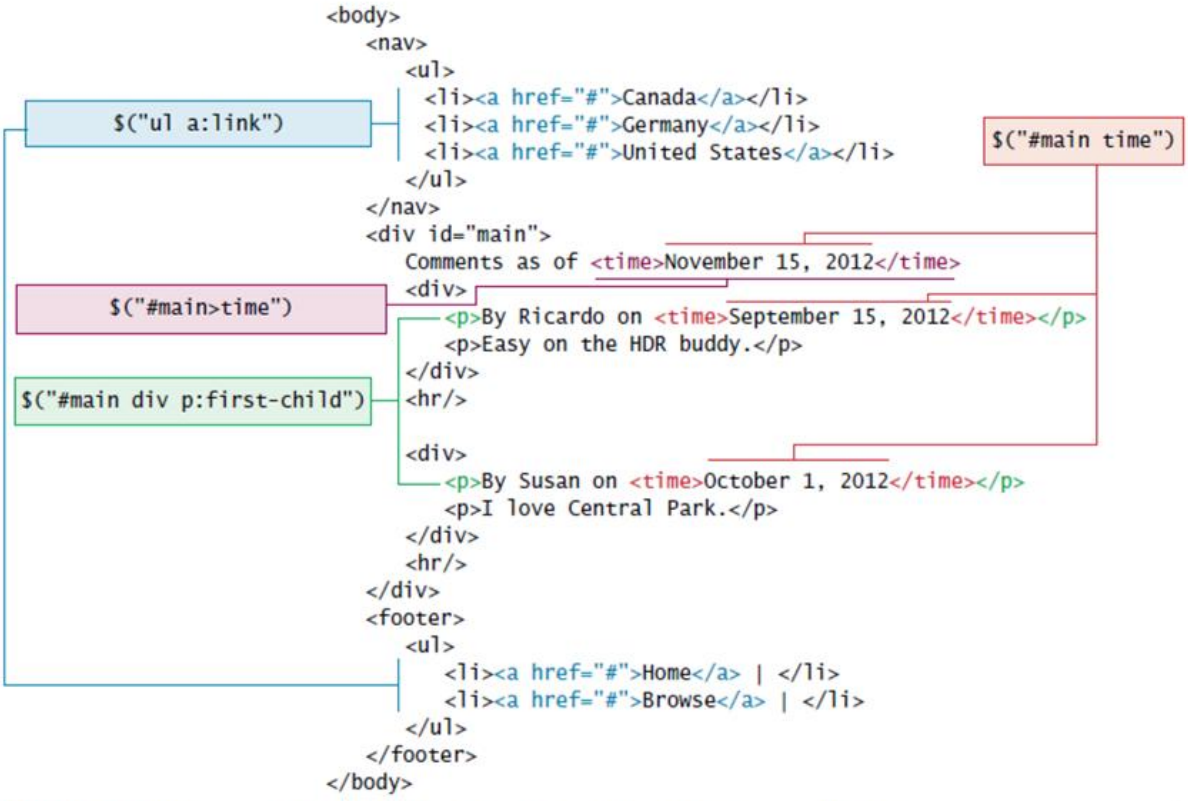



FIGURE 15.4 Illustration of some jQuery selectors and the HTML being selected

## 3. Pseudo-Element Selector

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");
```

4. **Contextual Selector**

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  elements you would write

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

5. **Content Filters**

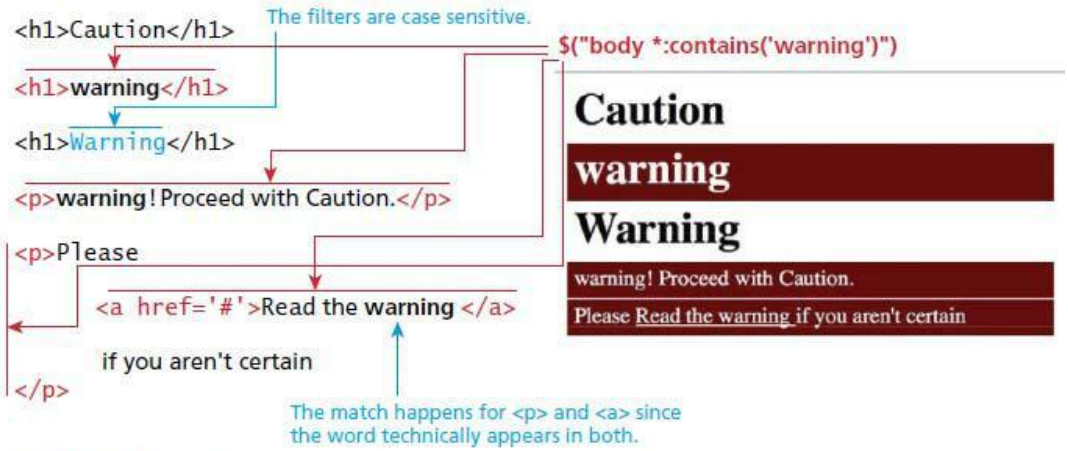



FIGURE 15.5 An illustration of jQuery's content filter selector

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.

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

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

6. **Form Selectors** : As depicted in table 15.1

what does $() short and stand for in jquery? explain any 3 jquery form selectors

| 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

## Query Attributes

**HTML Attributes**: The core set of attributes related to DOM elements are the ones specified in the HTML tags . You have by now integrated many of the key 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 :**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.
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 indi-vidually, have a look at the additional shortcut methods described in the jQuery documentation.
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.

## 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 :** when the page is fully downloaded it is parsed into its DOM representation

```
$(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);
});
});
```

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

**Listener Management:** 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.

```
$(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.

**Modifying the DOM:** jQuery comes with several useful methods to manipulate the DOM elements them-selves. 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:** 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

**A comparison of node creation in JS and jQuery**

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

## Prepending and Appending 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.

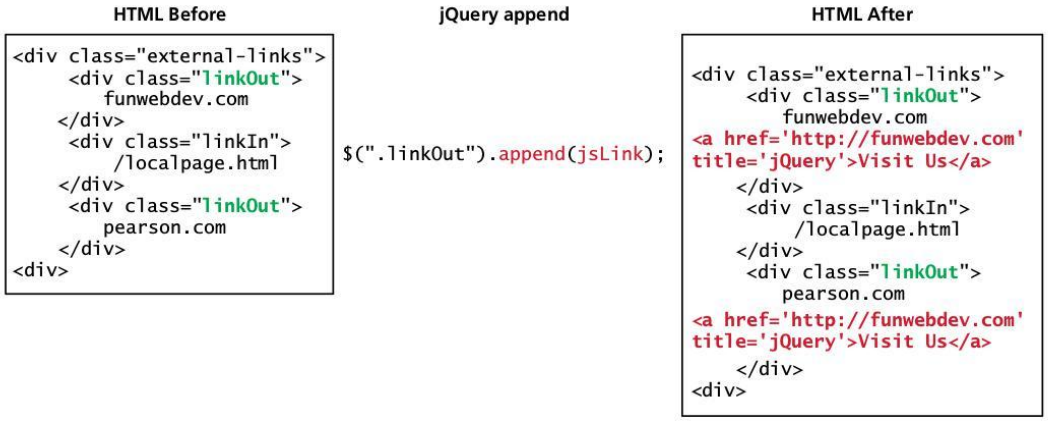



FIGURE 15.6 Illustration of where append adds a node

The appendTo() method is similar to append() but is used in the syntacti-cally 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().

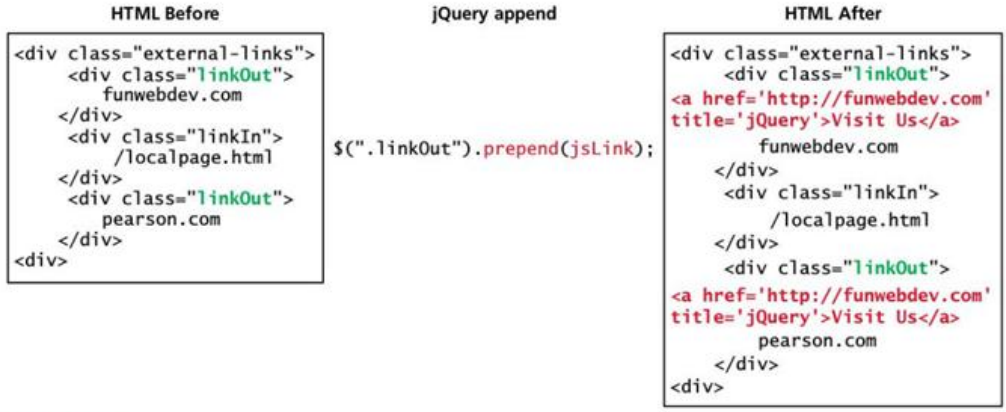



FIGURE 15.7 Illustration of prepend() adding a  node

## 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.

```
<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**

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(' ') which modifies the HTML to that shown in Listing 15.10.

```
<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  ele-ment that reflected the unique contents inside

```
$(".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).

```
<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 param-eters 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.

## AJAX

9. Explain AJAX get and post requests.
----->with suitable scripts

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.

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.
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).

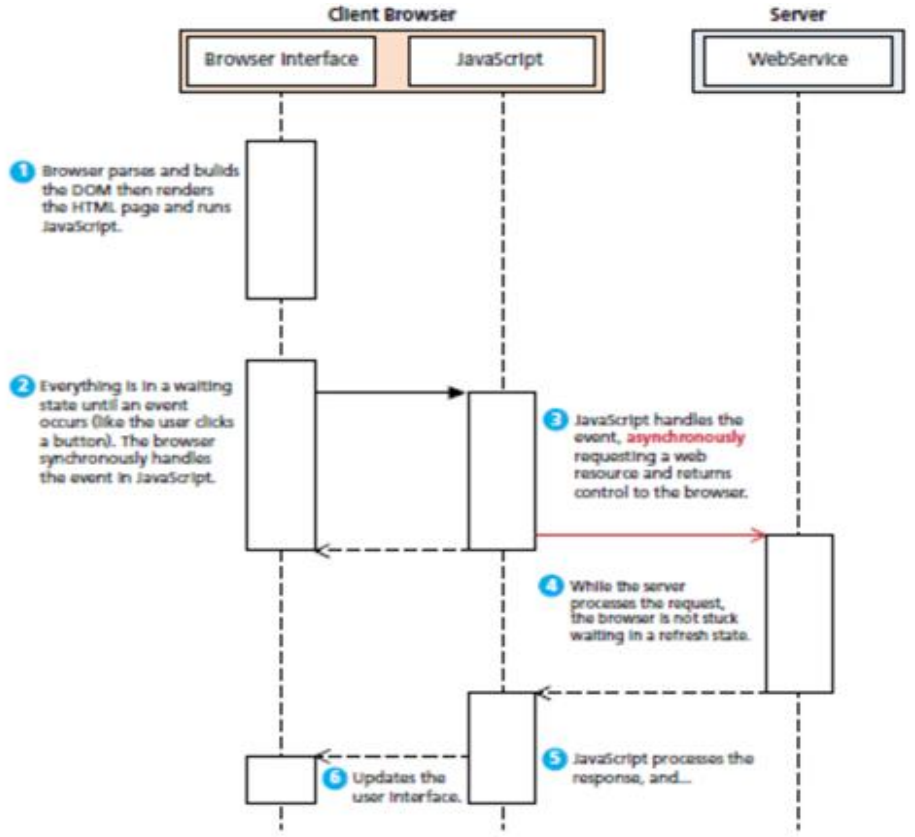



FIGURE 15.8 UML sequence diagram of an AJAX request

uml dig important

### Making Asynchronous Request

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. 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 asyn-chronously into the

 element, you could write:

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

**GET Request:**

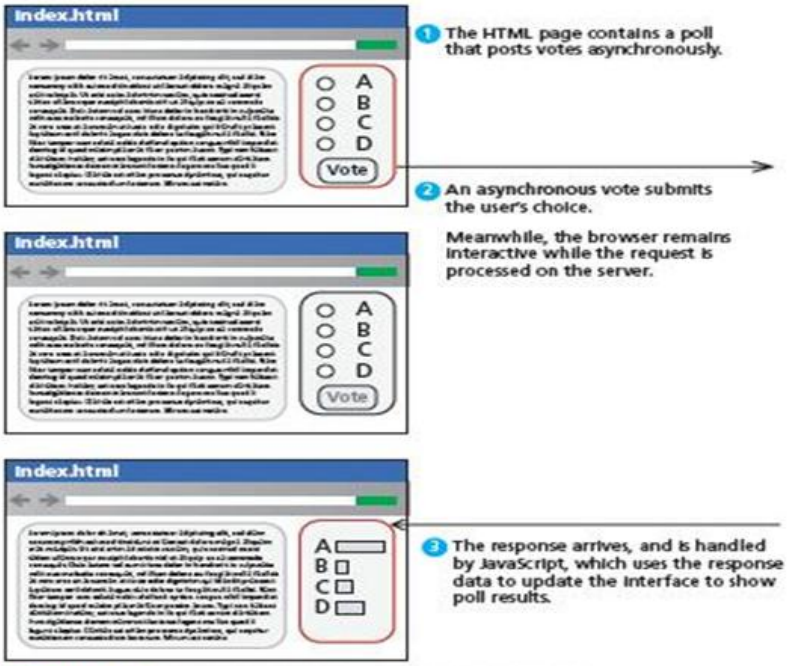



FIGURE 15.11 Illustration of a simple asynchronous web poll

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 asyn-chronously 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.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 complet-ing 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
- 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 success-ful 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**

## The jqXHR Object

The jqXHR (jQuery XMLHttpRequest) replaces the browser native XMLHttpRequest object. jQuery wraps the browser native XMLHttpRequest object with a superset API. The jQuery XMLHttpRequest (jqXHR) object is returned by the $.ajax() function. The jqXHR object simulates native XHR functionality where possible

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.

■ 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.
- 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.

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.

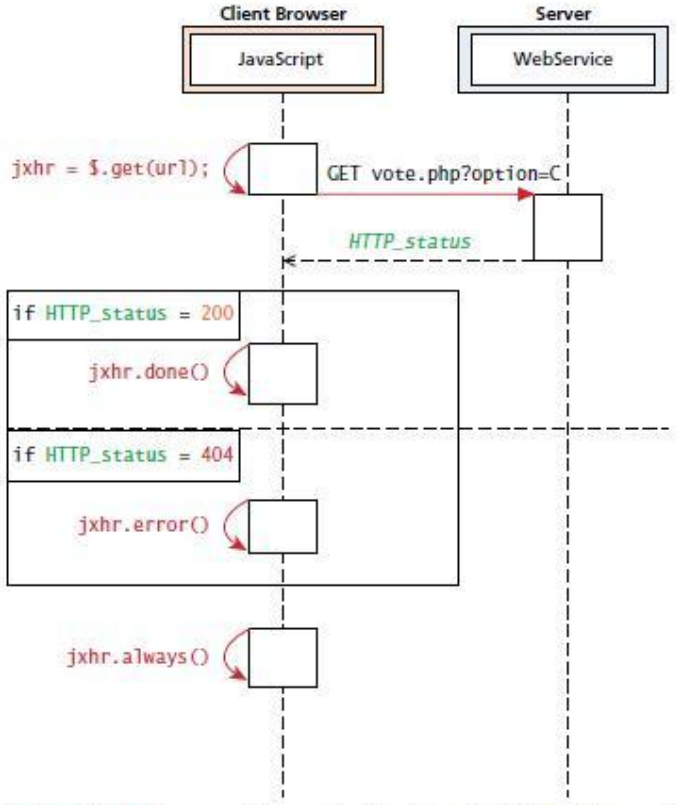



FIGURE 15.12 Sequence diagram depicting how the jqXHR object reacts to different response codes

```
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**

## 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.

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 ] )
```

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

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 vari-able), you can transmit that data through an asynchronous POST using the $.post() method as follows:

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

## 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 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 below:

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

the username and password fields. You can also modify headers using the header field, which brings us full circle to the HTTP protocol . 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", funwebdev.com"
}
```

**listing 15.16 Adding headers to an AJAX post in jQuery**

## 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 becomes harder. By default, JavaScript requests for images on images.funwebdev.com from the domain www.funwebdev.com will result in denied requests because subdo-mains 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.funweb-dev.com, allowing content to be shared between domains as needed.

**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 upload.

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**

**Old iframe Workarounds**

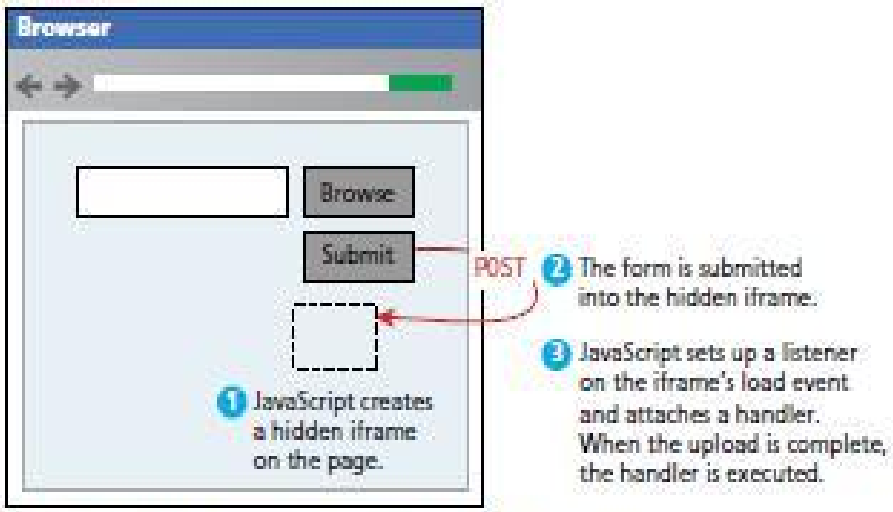



FIGURE 15.13 Illustration of posting to a hidden iframe

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 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.

```
$(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();
/       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) specifica-tion allows us to get access to file data and more through the FormData interface4 so we can post a file as illustrated in Figure 15.14.

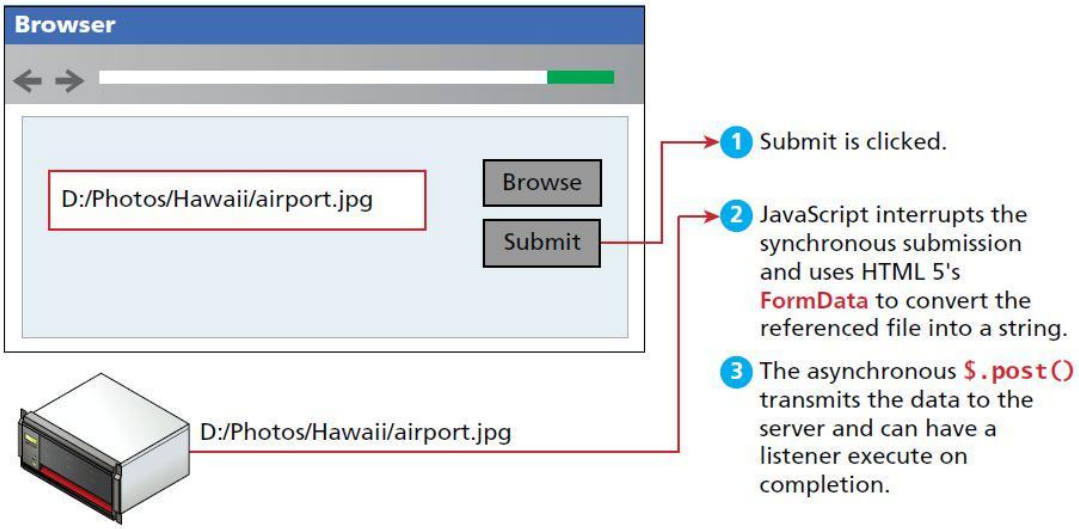



FIGURE 15.14 Posting a file using FormData

**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.

```
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**

**Appending Files to a POST:** 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 trans-mitted 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**

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.

## 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.

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 pro-vide 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 ani-mation (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.

```
<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];
        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.


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


FIGURE 15.16 Illustration of a fadeIn() animation

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 15.17 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.

**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 prepack-aged 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.

■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 devel-opment, 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 swing(t)52 12 cos(tπ)10.5

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

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

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 unre-lated 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 (3360). 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**

**Backbone MVC Frameworks:** In working with jQuery thus far we have seen how easily we can make great anima-tions 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 applica-tions, 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 design-ing a large-scale web application.

- **Getting Started with Backbone.js**

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. 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 .

**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.

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**

**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.

```
        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**

**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.

```
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 func-tions is available online.

```
/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**

# XML Overview

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 non web 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.
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).

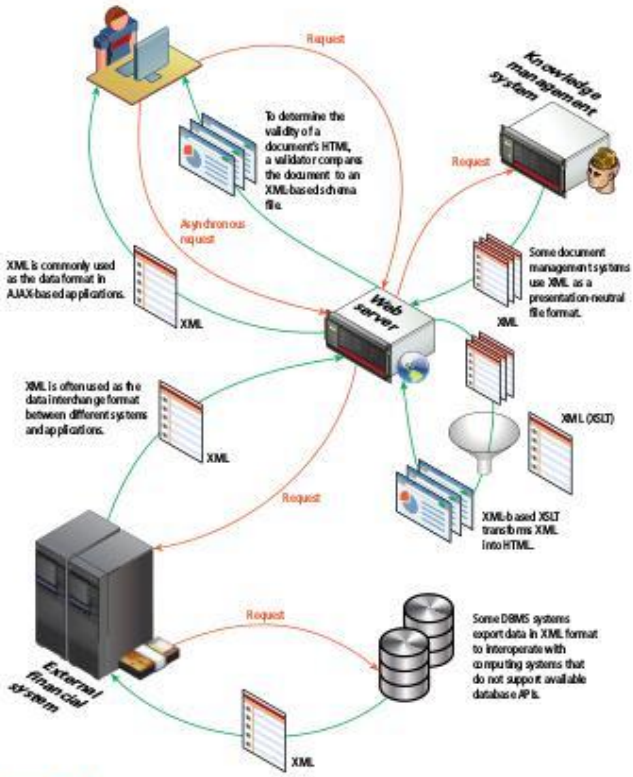



FIGURE 17.1 XML in the web context

**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.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

**DTD:**

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

**PCDATA** – Par sable Character Data

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>.

```
<?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

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 con-temporary browsers have built-in parsers, as do most web development environ-ments such as PHP and ASP.NET.

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 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.

imp question:

Construct DTD for the following XML code.

<?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>
</art>

```
<?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

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 vali-date the values of attributes or the textual content of elements.

# XSLT

XSLT stands for XML Style-sheet Transformations. XSLT is an XML-based programming language that is used for transforming XML into other document formats, as shown in Figure 17.2.

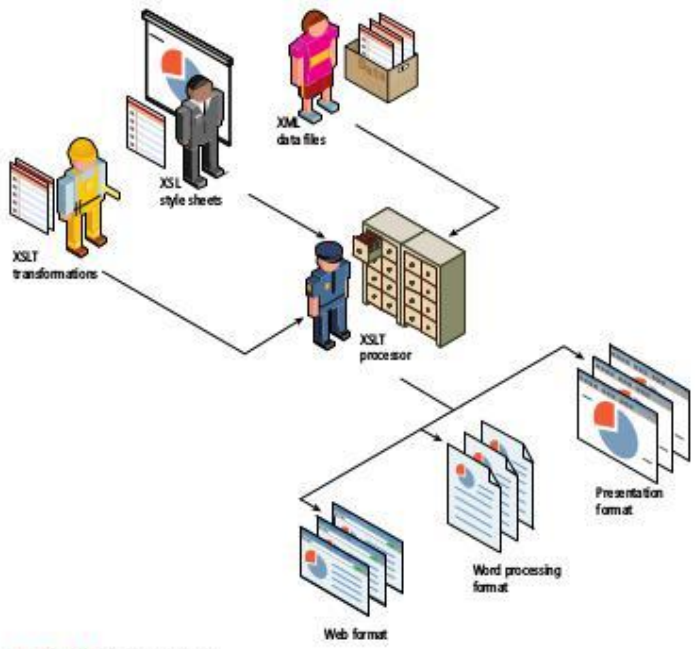



FIGURE 17.2 XSLT workflow

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.

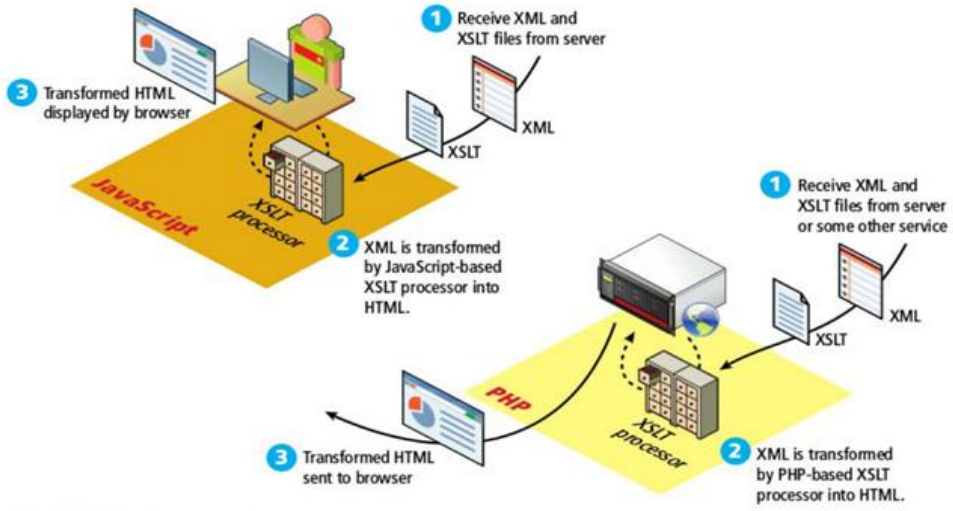



FIGURE 17.3 Usage of XSLT

The <xsl : for-each> element is the 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.

```
<?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


```
<?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

## 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. XPath is typically used as part of the programmatic manipulation of an XML document in PHP and other languages. XPath uses a syntax that is similar to the one used in most operating systems to access directories. 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 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 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.

**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.

## XML Processing in JavaScript

All modern browsers have a built-in XML parser and their JavaScript implementa-tions 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(), getElements ByTagName(), and createElement() to access and manipulate the data. For instance, Listing 17.5 shows the code necessary for loading an XML docu-ment 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.

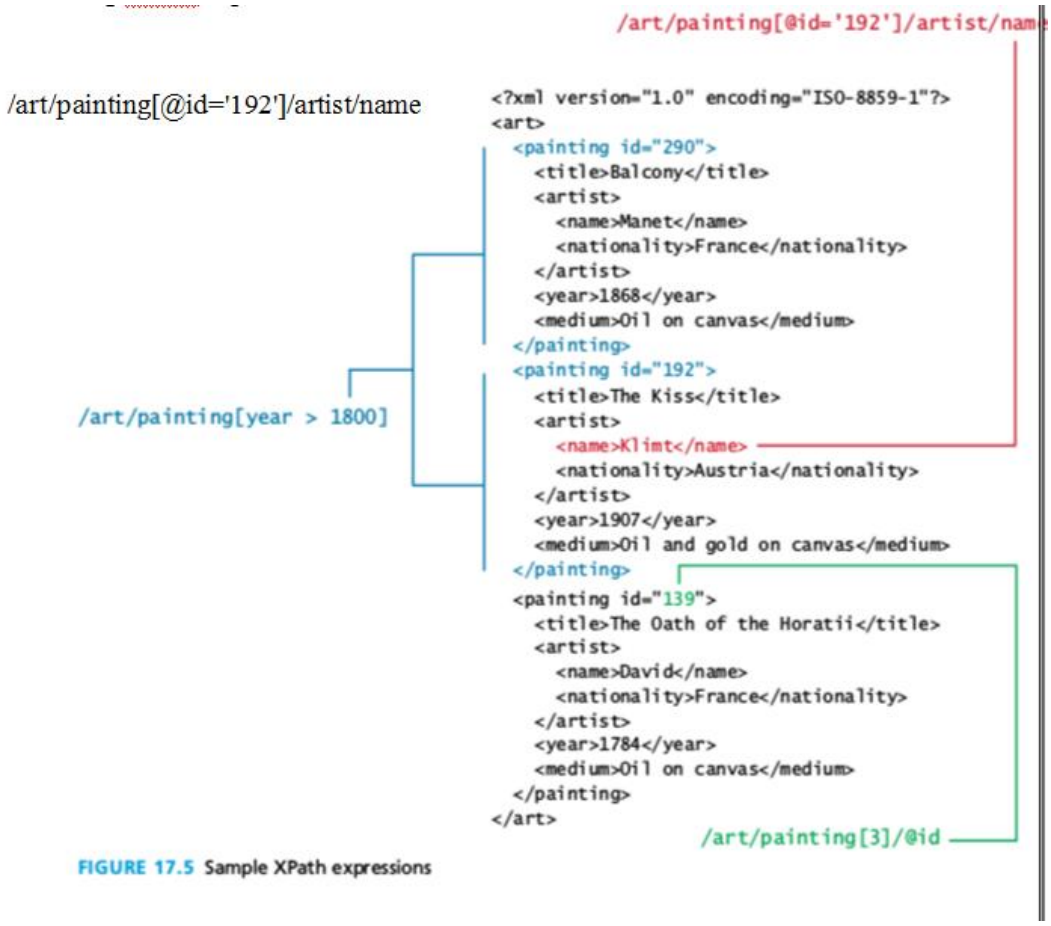



FIGURE 17.5 Sample XPath expressions

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.

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. 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  element within the HTML document.

```
<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");
}
```

```
// 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

```
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

```
<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

## XML Processing in PHP

PHP provides several extensions or APIs for working with XML:

■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.

■The Simple XML 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.

In general, the Simple XML 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 simple xml_load_file() function. The various elements in the XML document can then be manipulated using regular PHP object techniques.

```php
<?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

Power of XPath expressions can be used 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.

```php
$art = simplexml_load_file($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

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

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 XML Reader 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. For a multilevel XML file, the code can become quite complicated.

```
         $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

```
// 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

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.

## 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.

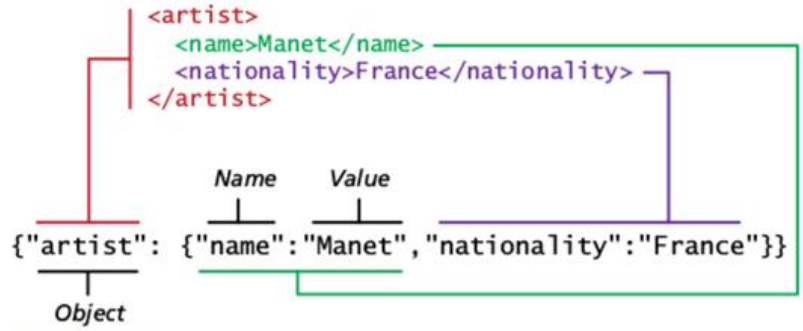



FIGURE 17.6 Sample JSON

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

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.

```
        {
            "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.

Explain converting a JSON string to JSON object in javascript and a PHP object in PHP

imp

## 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:

```
<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:

**Reading JSON in JavaScript**

```
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 brows-ers (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);
```

**Using 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. 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;
        }
?>
```

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);
```

## Overview of Web Services

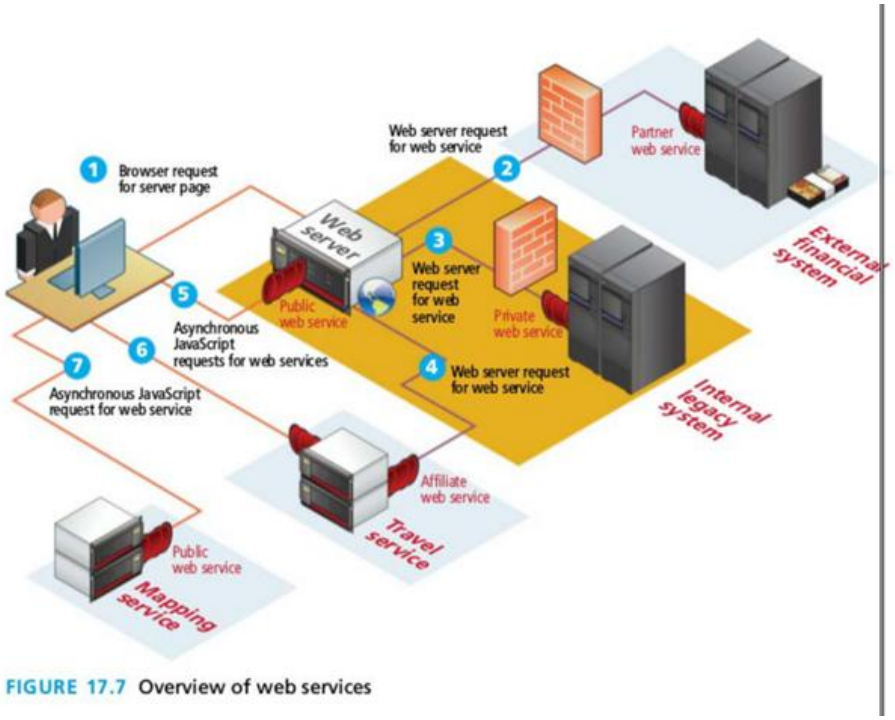



FIGURE 17.7 Overview of web services

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 Service Oriented Architecture (SOA). This type of software architecture aims to achieve very

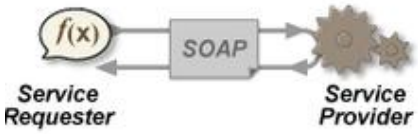
loose coupling among interacting software services. The rationale behind an SOA is one that is familiar to computing practitioners with some experience in the enter-prise: namely, how to best deal with the problem of application integration. Due to corporate mergers, longer-lived legacy applications, 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 differ-ent 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.

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 lan-guages as well as deal with the difficulty of enterprise-level application integration.

Explain SOAP and RESET web services with a neat diagram

**SOAP Services**

---------imp




**SOAP** (abbreviation for **Simple Object Access Protocol**) is a messaging protocol specification for exchanging structured information in the implementation of web services .The purpose is to provide extensibility, neutrality and independence.

It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP), although some legacy systems communicate over Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

SOAP allows developers to invoke processes running on disparate operating systems (such as Windows, macOS, and Linux) to authenticate, authorize, and communicate using Extensible Markup Language (XML).

Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.

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). No explicit detailed knowledge of the SOAP and WSDL specifications are required to create and consume SOAP-based services.

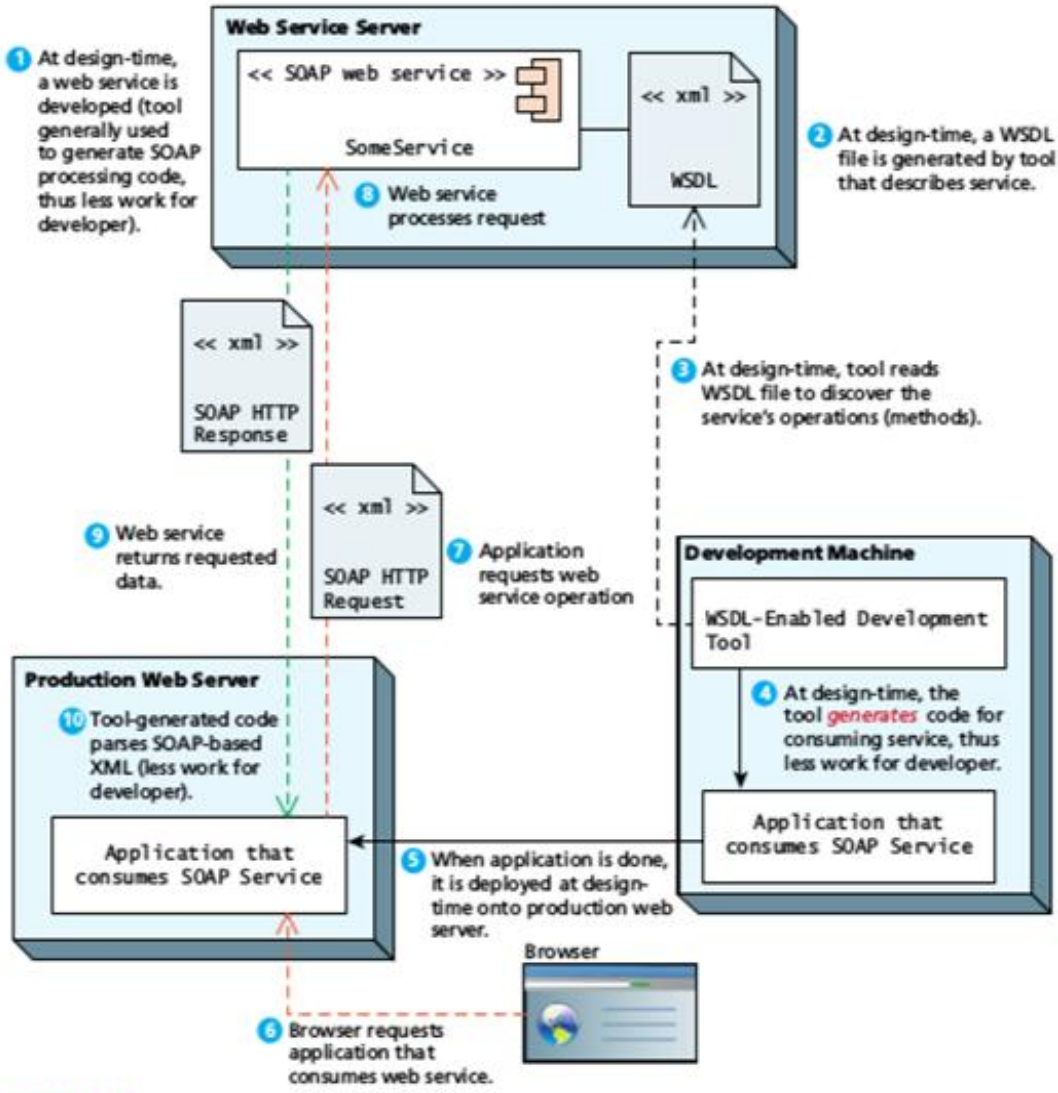



FIGURE 17.8 SOAP web services

## REST Services

**Representational state transfer** (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the Internet.

RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations.

By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running.

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.

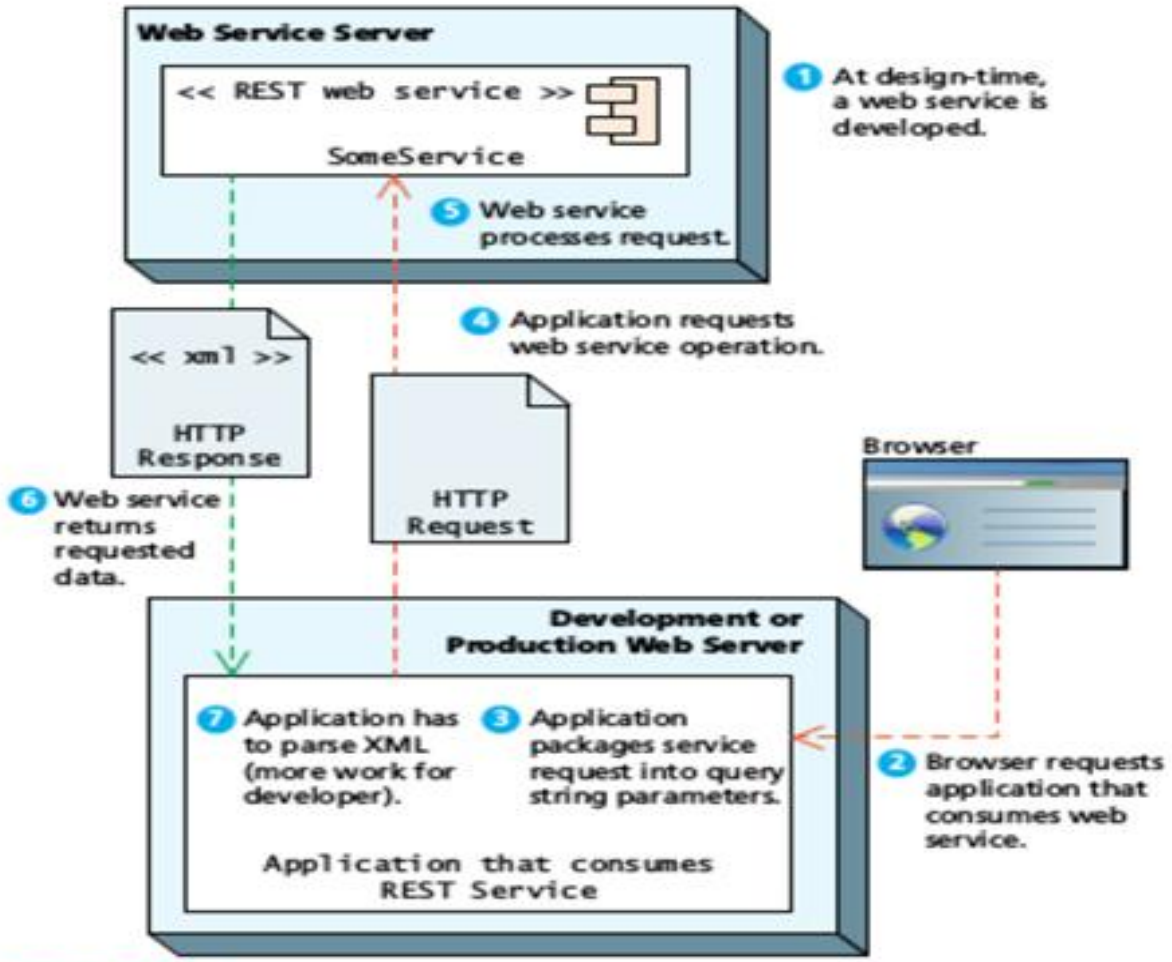



FIGURE 17.9 REST web services

REST services are used by 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.

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.

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 geo-code) and sensor (for whether the request comes from a device with a location sensor). So an example geocode request would look like the following:

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