What Is JavaScript and What Can It Do?

- Larry Ullman, in his *Modern JavaScript: Develop and Design* (Peachpit Press, 2012), has an especially succinct definition of JavaScript: it is an object-oriented, dynamically typed, scripting language.
- Although it contains the word *Java*, JavaScript and Java are vastly different programming languages with different uses. Java is a full-fledged compiled, objectoriented language, popular for its ability to run on any platform with a Java Virtual Machine installed. Conversely, JavaScript is one of the world's most popular languages, with fewer of the object-oriented features of Java, and runs directly inside the browser, without the need for the JVM. Although there are some syntactic similarities, the two languages are not interchangeable and should not be confused with one another.
- JavaScript is object oriented in that almost everything in the language is an object. For instance, variables are objects in that they have constructors, properties, and methods. Unlike more familiar object oriented languages such as Java, C#, and Visual Basic, functions in JavaScript are also objects. Given that JavaScript approaches objects far differently than other languages, and does not have a formal class mechanism nor inheritance syntax, we might say that it is a *strange* object-oriented language. JavaScript is dynamically typed (also called weakly typed) in that variables can be easily (or implicitly) converted from one data type to another.
- In a programming language such as Java, variables are statically typed, in that the data type of a variable is defined by the programmer (e.g., int abc) and enforced by the compiler. With JavaScript, the type of data a variable can hold is assigned at runtime and can change during run time as well.

Client-Side Scripting

- The idea of **client-side scripting** is an important one in web development. It refers to the client machine (i.e., the browser) running code locally rather than relying on the server to execute code and return the result.
- There are many client-side languages that have come into use over the past decade including Flash, VBScript, Java, and JavaScript. Some of these technologies only work in certain browsers, while others require plug-ins to function.

- Figure 6.1 illustrates how a client machine downloads and executes JavaScript code. There are many advantages of client-side scripting:
    - Processing can be offloaded from the server to client machines, thereby reducing the load on the server.
    - The browser can respond more rapidly to user events than a request to a remote server ever could, which improves the user experience.
    - JavaScript can interact with the downloaded HTML in a way that the server cannot, creating a user experience more like desktop software than simpleHTML ever could.
    - The disadvantages of client-side scripting are mostly related to how programmers use JavaScript in their applications. Some of these include:
    - There is no guarantee that the client has JavaScript enabled, meaning any required functionality must be housed on the server, despite the possibility that it could be offloaded.
    - The idiosyncrasies between various browsers and operating systems make it difficult to test for all potential client configurations. What works in one browser, may generate an error in another.
    - JavaScript-heavy web applications can be complicated to debug and maintain. JavaScript has often been used through inline HTML hooks that are embedded into the HTML of a web page.

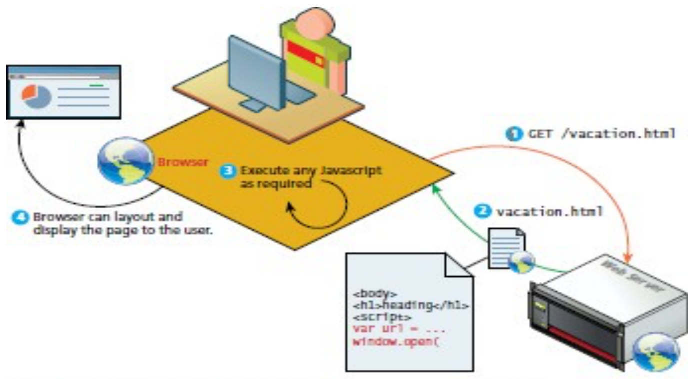


FIGURE 6.1 Downloading and executing a client-side JavaScript script

- There are two other noteworthy client-side approaches to web programming. Perhaps the most familiar of these alternatives is **Adobe Flash**, which is a vectorbased drawing and animation program, a video file format, and a software platform that has its own JavaScript-like programming language called **ActionScript**.
- Flash is often used for animated advertisements and online games, and can also be used to construct web interfaces. It is worth understanding how Flash works in the

browser. Flash objects (not videos) are in a format called SWF (Shockwave Flash) and are included within an HTML document via the <object> tag. The SWF file is then downloaded by the browser and then the browser delegates control to a plug-in to execute the Flash file, as shown in Figure 6.2.

- A **browser plug-in** is a software add-on that extends the functionality and capabilities of the browser by allowing it to view and process different types of web content. It should be noted that a browser plug-in is different than a **browser extension**— these also extend the functionality of a browser but are not used to process downloaded content. For instance, FireBug in the Firefox browser provides a wide range of tools that help the developer understand what's in a page; it doesn't really alter how the browser displays a page.

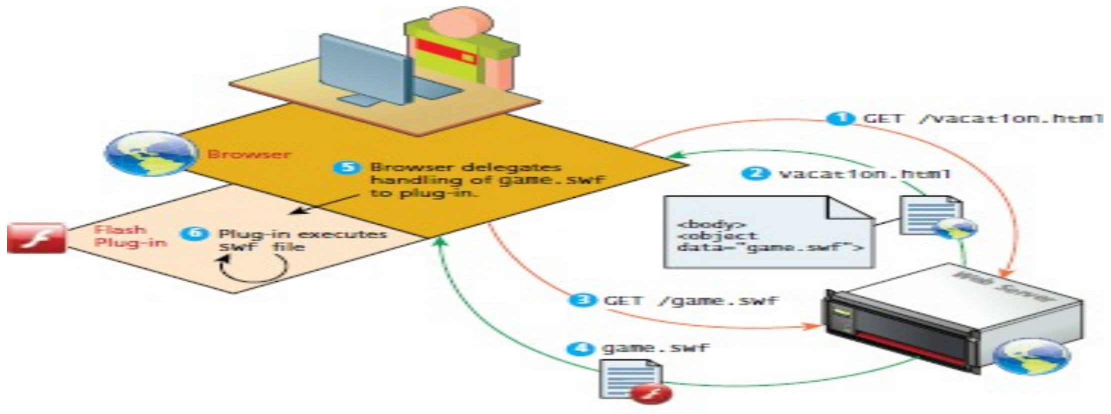


FIGURE 6.2 Adobe Flash

- The second (and oldest) of these alternatives to JavaScript is **Java applets**. An**applet** is a term that refers to a small application that performs a relatively smalltask. Java applets are written using the Java programming language and are separateobjects that are included within an HTML document via the <applet> tag,downloaded, and then passed on to a Java plug-in. This plug-in then passes on theexecution of the applet outside the browser to the Java Runtime Environment (JRE)that is installed on the client's machine.
- Figure 6.3 illustrates how Java applets workin the web environment. Both Flash plug-ins and Java applets are losing support by major players for a number of reasons. First, Java applets require the JVM be installed and up to date, which some players are not allowing for security reasons (Apple's iOS poweringiPhones and iPads supports neither Flash nor Java applets). Second, Flashand Java applets also require frequent updates, which can annoy the user andpresent security risks. With the

universal adoption of JavaScript and HTML5,JavaScript remains the most dynamic and important client-side scripting languagefor the modern web developer.

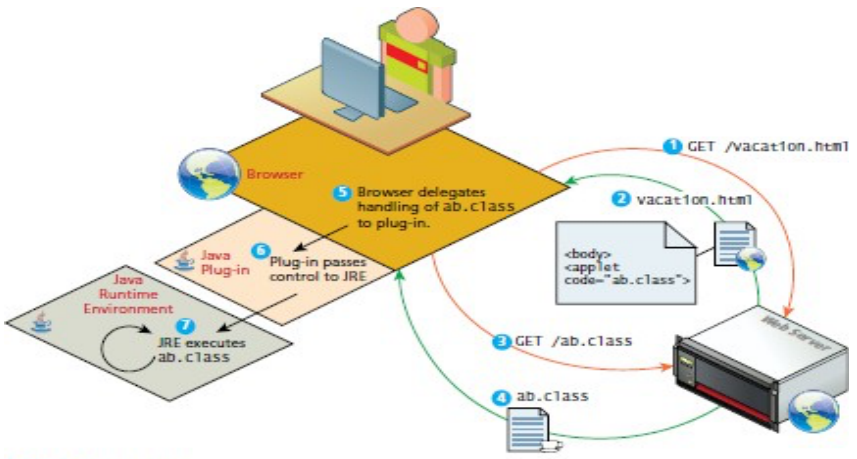


FIGURE 6.3 Java applets

JavaScript's History and Uses

- JavaScript was introduced by Netscape in their Navigator browser back in 1996. Itoriginally was called LiveScript, but was renamed partly because one of its original purposes was to provide a measure of control within the browser over Java applets.JavaScript is in fact an implementation of a standardized scripting language calledECMAScript.
- Internet Explorer (IE) at first did not support JavaScript, but instead had itsown browser-based scripting language (VBScript). While IE now does supportJavaScript, Microsoft sometimes refers to it as JScript, primarily for trademark reasons(Oracle currently owns the trademark for JavaScript).
- The current version for JavaScript at the time of writing is 1.8.5. At that time, JavaScript was only slightly useful, and quite often, very annoying to many users. At that time, JavaScript had only a few common uses: graphic roll-overs (that is, swapping one image for another when the user hovered the mouse over an image), pop-up alert messages, scrolling text in the status bar, opening new browser windows, and prevalidating user data in online forms. It wasn't until the middle of the 2000s with the emergence of so-called AJAX sites that JavaScript became a much more important part of web development.
- AJAX is both an acronym as well as a general term. As an acronym it means Asynchronous JavaScript and XML, which was accurate for some time; but since XML is no longer always the data format for data transport in AJAX sites, the acronym meaning is becoming less and less accurate.

- As a general term, AJAX refers to a style of website development that makes use of JavaScript to create more responsive user experiences. The most important way that this responsiveness is created is via asynchronous data requests via JavaScript and the XMLHttpRequest object. This addition to JavaScript was introduced by Microsoft as an ActiveX control (the IE version of plug-ins) in 1999.
- Figure 6.4 illustrates the processing flow for a page that requires updates based on user input using the normal synchronous non-AJAX page request-response loop. As you can see in Figure 6.4, such interaction requires multiple requests to the server, which not only slows the user experience, it puts the server under extra load.

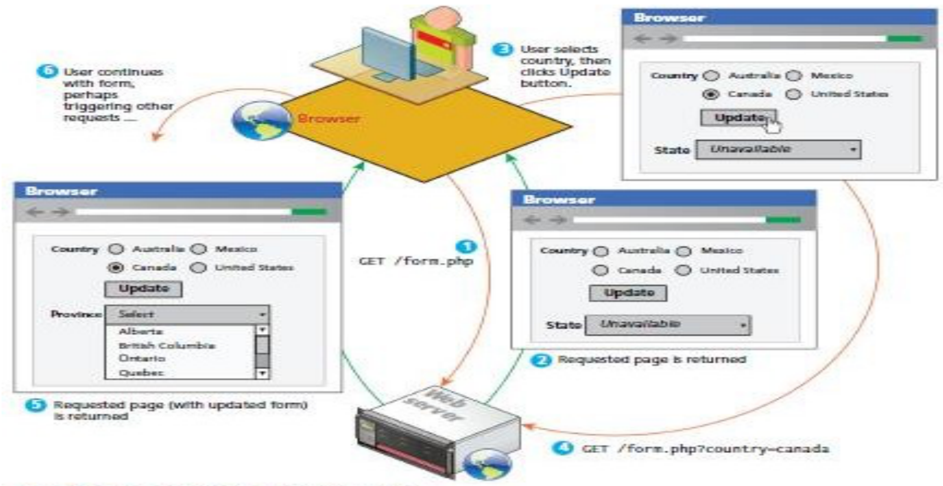

6 User continues with form, perhaps triggering other requests ...
Browser
3 User selects country, then clicks Update button.
Browser
Country Australia Mexico Canada United States
Update
State Unavailable
Browser
Country Australia Mexico Canada United States
Update
Province Select Alberta British Columbia Ontario Quebec
1 GET /form.php
Browser
Country Australia Mexico Canada United States
Update
State Unavailable
2 Requested page is returned
5 Requested page (with updated form) is returned
Web server
4 GET /form.php?country=canada

FIGURE 6.4 Normal HTTP request-response loop

- Figure 6.4, each request is invoking a server-side script. With ever-increasing access to processing power and bandwidth, sometimes it can be really hard to tell just how much impact these requests to the server have, but it's important to remember that more trips to the server do add up, and on a large scale this can result in performance issues.

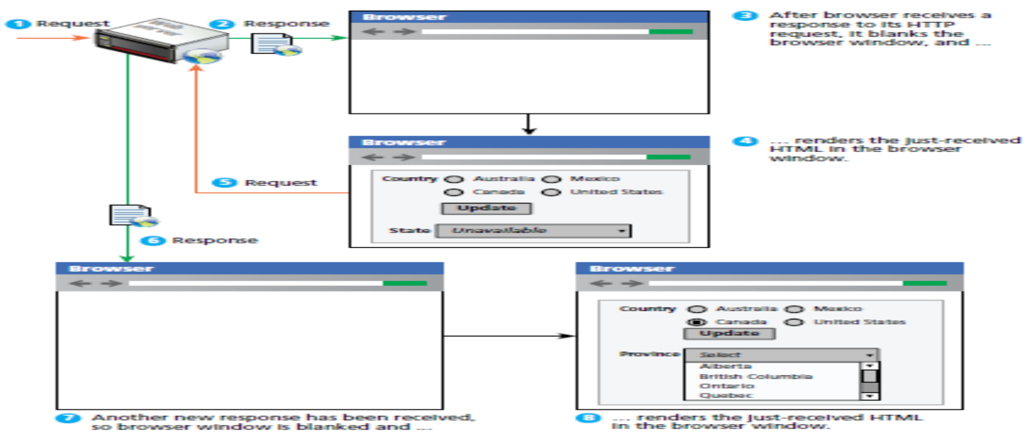

1 Request
2 Response
Browser
3 After browser receives a response to its HTTP request, it blanks the browser window, and ...
Browser
4 ... renders the just-received HTML in the browser window.
Country Australia Mexico Canada United States
Update
State Unavailable
5 Request
6 Response
Browser
Browser
Country Australia Mexico Canada United States
Update
Province Select Alberta British Columbia Ontario Quebec
7 Another new response has been received, so browser window is blanked and ...
8 ... renders the just-received HTML in the browser window.

FIGURE 6.5 Normal HTTP request-response loop, take two

- But as can be seen in Figure 6.5, when these multiple requests are being made across the Internet to a busy server, then the time costs of the normal HTTP requestresponse loop will be more visually noticeable to the user.

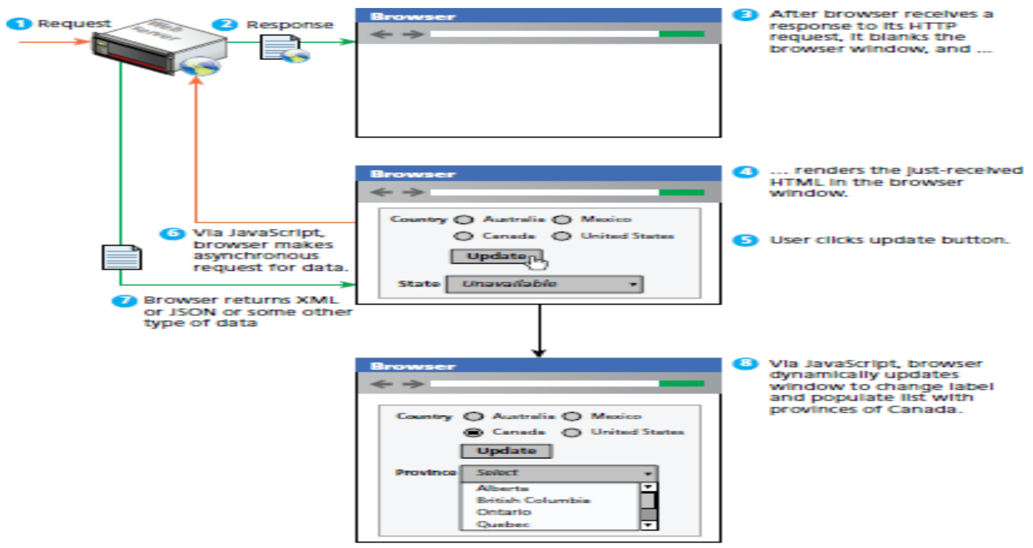

1 Request
Web Server
2 Response
Browser
3 After browser receives a response to its HTTP request, it blanks the browser window, and ...
4 ... renders the just-received HTML in the browser window.
5 User clicks update button.
6 Via JavaScript, browser makes asynchronous request for data.
7 Browser returns XML or JSON or some other type of data
8 Via JavaScript, browser dynamically updates window to change label and populate list with provinces of Canada.
Country
Australia
Mexico
Canada
United States
Update
State
Unavailable
Province
Select
Alberta
British Columbia
Ontario
Quebec

FIGURE 6.6 Asynchronous data requests

- AJAX provides web authors with a way to avoid the visual and temporal deficiencies of normal HTTP interactions. With AJAX web pages, it is possible to update sections of a page by making special requests of the server in the background, creating the illusion of continuity. Figure 6.6 illustrates how the interaction shown in Figure 6.4 would differ in an AJAX-enhanced web page.
- This type of AJAX development can be difficult but thankfully, the other key development in the history of JavaScript has made AJAX programming significantly less tricky. This development has been the creation of **JavaScript frameworks**, such as jQuery, Prototype, ASP.NET AJAX, and MooTools. These JavaScript frameworks reduce the amount of JavaScript code required to perform typical AJAX tasks. Some of these extend the JavaScript language; others provide functions and objects to simplify the creation of complex user interfaces. jQuery, in particular, has an extremely large user base, used on over half of the top 100,000 websites.

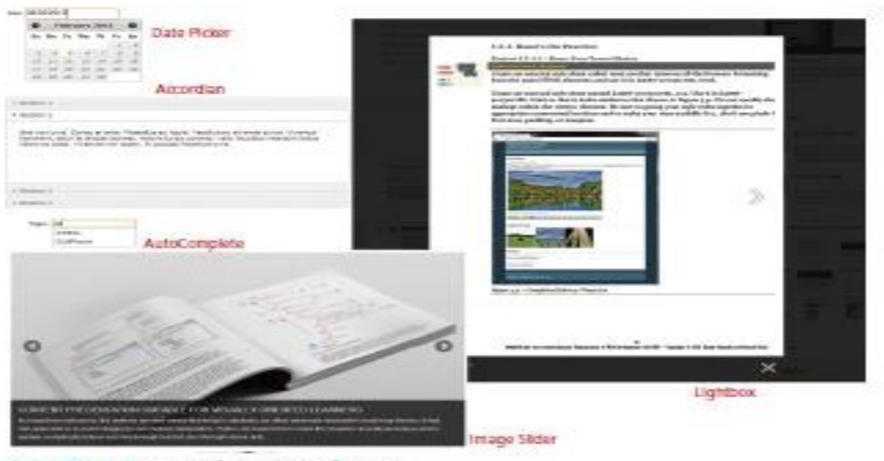


FIGURE 6.7 Example jQuery plug-ins

- Figure 6.7 illustrates some sample jQuery plug-ins, which are a way for developers to extend the functionality of jQuery.
- There are thousands of jQuery plug-ins available, which handle everything from additional user interface functionality to data handling. More recently, sophisticated MVC JavaScript frameworks such as AngularJS, Backbone, and Knockout have gained a lot of interest from developers wanting to move more data processing and handling from server-side scripts to HTML pages using a software engineering best practice, namely the separation of the model (data representation) from the view (presentation of data) design pattern.

JavaScript Design Principles

- Although frameworks and developer tools can help, there is some truth to this reputation. It should be said, however, that this reputation is based not so much on the language itself but in how developers have tended to use it.
- JavaScript has often been used through inline HTML hooks—that is, embedded into the HTML of a web page. Although this technique has been used for years, it has the distinct disadvantage of blending HTML and JavaScript together, which decreases code readability, and increases the difficulty of web development.

Layers

- When designing software to solve a problem, it is often helpful to abstract the solution a little bit to help build a cognitive model in your mind that you can then implement. Perhaps the most common way of articulating such a cognitive model is via the term **layer**. In object-oriented programming, a software layer is a way of conceptually grouping programming classes that have similar functionality and dependencies. Common software design layer names include:

- **Presentation layer.** Classes focused on the user interface.
- **Business layer.** Classes that model real-world entities, such as customers, products, and sales.
- **Data layer.** Classes that handle the interaction with the data sources.

Presentation Layer

This type of programming focuses on the display of information. JavaScript can alter the HTML of a page, which results in a change, visible to the user. These presentation layer applications include common things like creating, hiding, and showing divs, using tabs to show multiple views, or having arrows to page through result sets. This layer is most closely related to the user experience and the most visible to the end user.

Validation Layer

JavaScript can be also used to validate logical aspects of the user's experience. This could include, for example, validating a form to make sure the email entered is valid before sending it along. It is often used in conjunction with the presentation layer to create a coherent user experience, where a message to the presentation layer highlights bad fields. Both layers exist on the client machine, although the intention is to prevalidate forms before making transmissions back to the server.

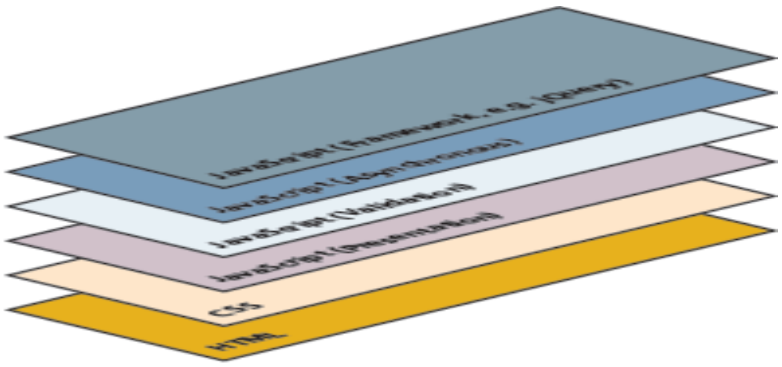

JavaScript (framework, e.g. jQuery)
JavaScript (Asynchronous)
JavaScript (Validation)
JavaScript (Presentation)
CSS
HTML

FIGURE 6.8 JavaScript layers

Asynchronous Layers

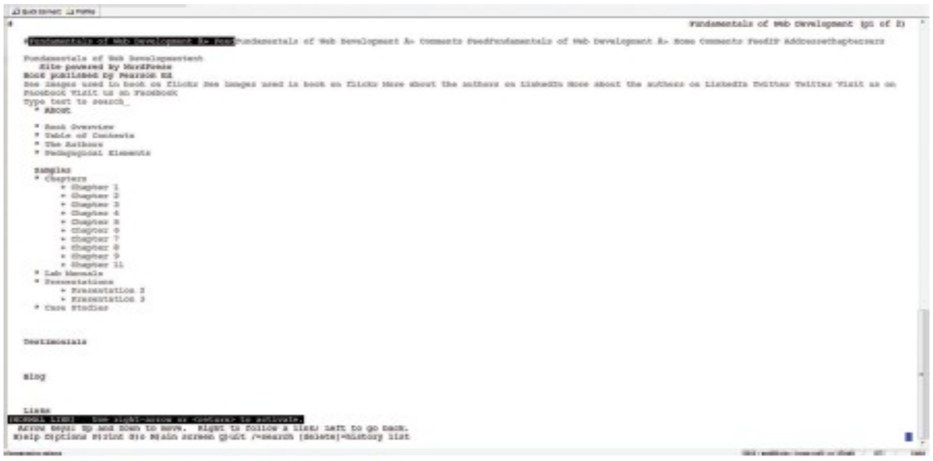
- JavaScript operates in a synchronous manner where a request sent to the server requires a response before the next lines of code can be executed.
- During the wait between request and response the browser sits in a loading state and only updates upon receiving the response. In contrast, an asynchronous layer can route requests to the server in the background.
- In this model, as certain events are triggered, the JavaScript sends the HTTP requests to the server, but while waiting for the response, the rest of the application functions normally, and the browser isn't in a loading state.

- When the response arrives JavaScript will (perhaps) update a portion of the page. Asynchronous layers are considered advanced versions of the presentation and validation layers above. Typically developers work on a single file or application, weaving aspects of logic and presentation together. Although this is a common practice, separating the presentation and logic in your code is a powerful technique worth keeping in mind as you code. Having separate presentation and logic functions/classes will help you achieve more reusable code, which also happens to be easier to understand and maintain as illustrated in Figure 6.20.

Users without JavaScript

- Too often website designers believe (erroneously) that users without JavaScript are somehow relics of a forgotten age, using decades-old computers in a bomb shelter somewhere philosophically opposed to updating their OS and browsers and therefore not worth worrying about.
- Nothing could be more of a straw man argument. Users have a myriad of reasons for not using JavaScript, and that includes some of the most important clients, like search engines. A client may not have JavaScript because they are a web crawler, have a browser plug-in, are using a text browser, or are visually impaired.
- **Web crawler**. A web crawler is a client running on behalf of a search engine to download your site, so that it can eventually be featured in their search results. These automated software agents do not interpret JavaScript, since it is costly, and the crawler cannot see the enhanced look anyway.
  - **Browser plug-in**. A browser plug-in is a piece of software that works within the browser, that might interfere with JavaScript. There are many uses of JavaScript that are not desirable to the end user. Many malicious sites use JavaScript to compromise a user's computer, and many ad networks deploy advertisements using JavaScript. This motivates some users to install plug-ins that stop JavaScript execution. An ad-blocking plug-in, for example, may filter JavaScript scripts that include the word *ad*, so a script named **advanced.js** would be blocked inadvertently.
  - **Text-based client**. Some clients are using a text-based browser. Text-based browsers are widely deployed on web servers, which are often accessed using a command-line interface. A website administrator might want to see what an HTTP

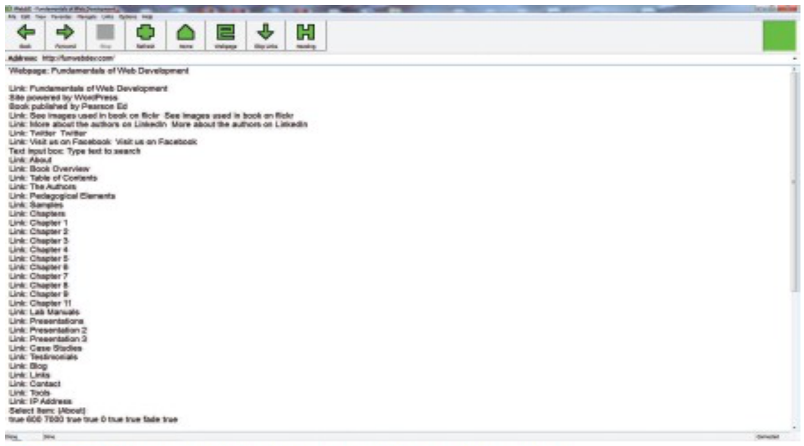
GET request to another server is returning for testing or support purposes. Such software includes lynx as shown in Figure 6.9.



```
                                                                  Fundamentals of Web Development (p1 of 2)
#Fundamentals of Web Development » FeedFundamentals of Web Development » Comments FeedFundamentals of Web Development » Some Comments FeedIP AddressChapterzers

Fundamentals of Web Development
   Site powered by WordPress
Book published by Pearson Ed
See images used in book on flickr See images used in book on flickr More about the authors on LinkedIn More about the authors on LinkedIn Twitter Twitter Visit us on
Facebook Visit us on Facebook
Type text to search_
  * About

  * Book Overview
  * Table of Contents
  * The Authors
  * Pedagogical Elements

  Samples
  * Chapters
       + Chapter 1
       + Chapter 2
       + Chapter 3
       + Chapter 4
       + Chapter 5
       + Chapter 6
       + Chapter 7
       + Chapter 8
       + Chapter 9
       + Chapter 11
  * Lab Manuals
  * Presentations
       + Presentation 2
       + Presentation 3
  * Case Studies


Testimonials


Blog


Links
(NORMAL LINK)   Use right-arrow or <return> to activate.
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

FIGURE 6.9 Surfing the web with Lynx

➢ **Visually disabled client**. A visually disabled client will use special web browsing software to read the contents of a web page out loud to them. These specialized browsers do not interpret JavaScript, and some JavaScript on sites is not accessible to these users. Designing for these users requires some extra considerations, with lack of JavaScript being only one of them. Open-source browsers like WebIE would display the same site as shown in Figure 6.10.



```
Address: http://funwebdev.com/
Webpage: Fundamentals of Web Development

Link: Fundamentals of Web Development
Site powered by WordPress
Book published by Pearson Ed
Link: See images used in book on flickr  See images used in book on flickr
Link: More about the authors on LinkedIn  More about the authors on LinkedIn
Link: Twitter  Twitter
Link: Visit us on Facebook  Visit us on Facebook
Text input box: Type text to search
Link: About
Link: Book Overview
Link: Table of Contents
Link: The Authors
Link: Pedagogical Elements
Link: Samples
Link: Chapters
Link: Chapter 1
Link: Chapter 2
Link: Chapter 3
Link: Chapter 4
Link: Chapter 5
Link: Chapter 6
Link: Chapter 7
Link: Chapter 8
Link: Chapter 9
Link: Chapter 11
Link: Lab Manuals
Link: Presentations
Link: Presentation 2
Link: Presentation 3
Link: Case Studies
Link: Testimonials
Link: Blog
Link: Links
Link: Contact
Link: Tools
Link: IP Address
Select Item: (About)
true 600 7600 true true 0 true true false true
```

FIGURE 6.10 Screenshot of WebIE, browser for the visually impaired

The <NoScript> Tag

➢ Now that we know there are many sets of users that may have JavaScript disabled, we may want to make use of a simple mechanism to show them special HTML content that will not be seen by those with JavaScript.

➢ That mechanism is the HTML tag <noscript>. Any text between the opening and closing tags will only be displayed to users without the ability to load JavaScript. It is often used to prompt users to enable JavaScript, but can also be used to show additional text to search engines.

- Increasingly, websites that focus on JavaScript or Flash only risk missing out on an important element to help get them noticed: search engine optimization (SEO). Moreover, older or mobile browsers may not have a complete JavaScript implementation. Requiring JavaScript (or Flash) for the basic operation of your site will cause problems eventually and should be avoided.
- In this spirit, we should create websites with all the basic functionality enabled using regular HTML. For those (majority) of users with JavaScript enabled we can then enhance the basic layout using JavaScript to: embellish the look of certain elements, animate certain user interactions, prevalidate forms, and generally replace static HTML elements with more visually and logically enhanced elements from JavaScript. Some examples of this principle would be to replace submit buttons with animated images, or adding dropdown menus to an otherwise static menu structure.
- This approach of adding functional replacements for those without JavaScript is also referred to as **fail-safe design**, which is a phrase with a meaning beyond web development. It means that when a plan (such as displaying a fancy JavaScript popup calendar widget) fails (because for instance JavaScript is not enabled), then the system's design will still work (for instance, by allowing the user to simply type in a date inside a text box).

Graceful Degradation and Progressive Enhancement

- The principle of fail-safe design can still apply even to browsers that have enabled JavaScript. Over the years, browser support for different JavaScript objects has varied. Something that works in the current version of Chrome might not work in IE version 8; something that works in a desktop browser might not work in a mobile browser.
- In such cases, what strategy should we take as web application developers? The principle of **graceful degradation** is one possible strategy. With this strategy you develop your site for the abilities of current browsers.
- For those users who are not using current browsers, you might provide an alternate site or pages for those using older browsers that lack the JavaScript (or CSS or HTML5) used on the main site. The idea here is that the site is "degraded" (i.e., loses

capability) “gracefully” (i.e., without pop-up JavaScript error codes or without condescending messages telling users to upgrade their browsers).

- Figure 6.11 illustrates the idea of graceful degradation. The alternate strategy is **progressive enhancement**, which takes the opposite approach to the problem. In this case, the developer creates the site using CSS, JavaScript, and HTML features that are supported by all browsers of a certain age or newer. (Eventually, one does have to stop supporting ancient browsers; many developers have, for instance, stopped supporting IE 6.)

Where Does JavaScript Go?

- JavaScript can be linked to an HTML page in a number of ways. Just as CSS styles can be **inline**, **embedded**, or **external**, JavaScript can be included in a number of ways. Just as with CSS these can be combined, but external is the preferred method for cleanliness and ease of maintenance.
- Running JavaScript scripts in your browser requires downloading the JavaScript code to the browser and then running it. Pages with lots of scripts could potentially run slowly, resulting in a degraded experience while users wait for the page to load.
- Different browsers manage the downloading and loading of scripts in different ways, which are important things to realize when you decide how to link your scripts.

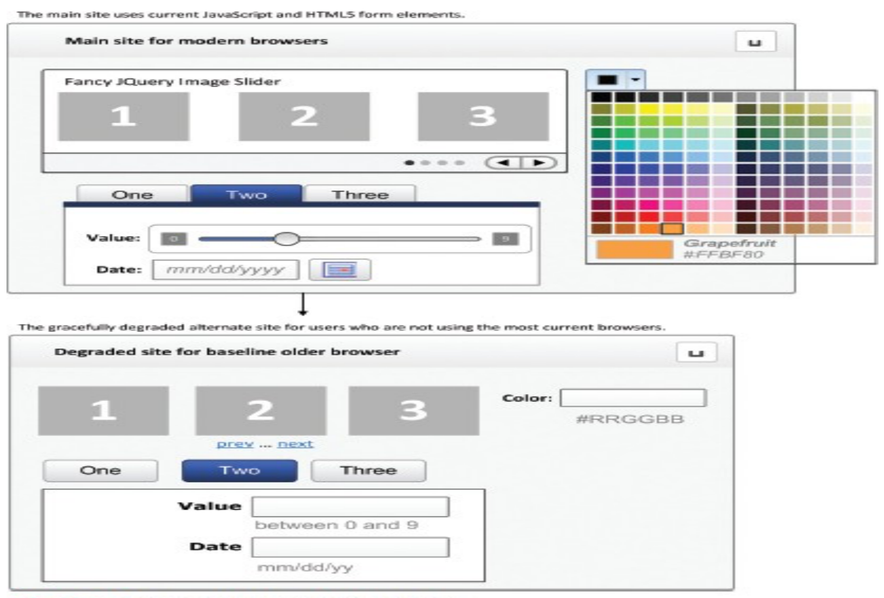

The main site uses current JavaScript and HTML5 form elements.
Main site for modern browsers
Fancy JQuery Image Slider
1
2
3
One
Two
Three
Value:
Date:
mm/dd/yyyy
Grapefruit
#FFBF80
The gracefully degraded alternate site for users who are not using the most current browsers.
Degraded site for baseline older browser
1
2
3
prev ... next
Color:
#RRGGBB
One
Two
Three
Value
between 0 and 9
Date
mm/dd/yy

FIGURE 6.11 Example of graceful degradation

Inline JavaScript

- Inline JavaScript refers to the practice of including JavaScript code directly within certain HTML attributes, such as that shown in Listing 6.1.

➢ The same is true with JavaScript. Infact, inline JavaScript is much worse than inline CSS. Inline JavaScript is a realmaintenance nightmare, requiring maintainers to scan through almost every line ofHTML looking for your inline JavaScript.

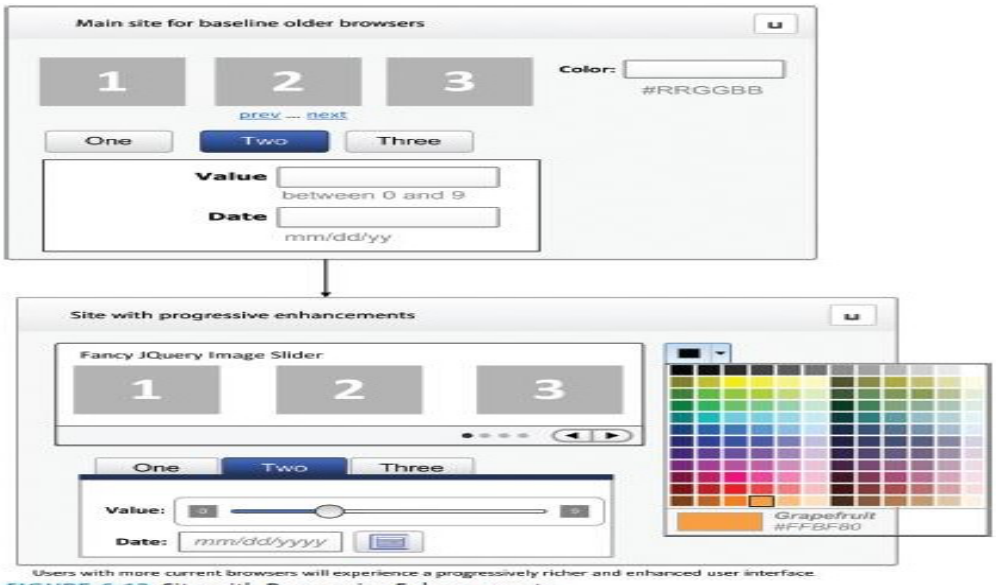

Main site for baseline older browsers
1
2
3
prev ... next
One
Two
Three
Value
between 0 and 9
Date
mm/dd/yy
Color:
#RRGGBB
Site with progressive enhancements
Fancy JQuery Image Slider
1
2
3
One
Two
Three
Value:
Date:
mm/dd/yyyy
Grapefruit
#FFBF80
Users with more current browsers will experience a progressively richer and enhanced user interface.

FIGURE 6.12 Site with Progressive Enhancements

```
<a href="JavaScript:OpenWindow();"more info</a>
<input type="button" onclick="alert('Are you sure?');" />
```

LISTING 6.1 Inline JavaScript example

Embedded JavaScript

➢ Embedded JavaScript refers to the practice of placing JavaScript code within a <script> element as shown in Listing 6.2. Like its equivalent in CSS, embedded JavaScript is okay for quick testing and for learning scenarios, but is frowned upon for normal realworld pages. Like with inline JavaScript, embedded scripts can be difficult to maintain

```
<script type="text/javascript">
/* A JavaScript Comment */
alert ("Hello World!");
</script>
```

LISTING 6.2 Embedded JavaScript example

External JavaScript

➢ JavaScript supports this separation by allowing links to an external file that contains the JavaScript, as shown in Listing 6.3.

- This is the recommended way of including JavaScript scripts in your HTML pages. By convention, JavaScript external files have the extension .js. Modern websites often have links to several, maybe even dozens, of external JavaScript files (also called **libraries**).

```
<head>
  <script type="text/JavaScript" src="greeting.js">
  </script>
</head>
```

LISTING 6.3 External JavaScript example

- These external files typically contain function definitions, data definitions, and other blocks of JavaScript code.
- In Listing 6.3, the link to the external JavaScript file is placed within the <head> element, just as was the case with links to external CSS files. While this is convention, it is in fact possible to place these links anywhere within the <body> element. We certainly recommend placing them either in the <head> element or the very bottom of the <body> element.
- The argument for placing external scripts at the bottom of the <body> has to do with performance. A JavaScript file has to be loaded completely before the browser can begin any other downloads (including images). For sites with multiple external JavaScript files, this can cause a noticeable delay in initial page rendering.

Advanced Inclusion of JavaScript

- Imagine for a moment a user with a browser that has JavaScript disabled. When downloading a page, if the JavaScript scripts are embedded in the page, they must download those scripts in their entirety, despite being unable to process them. A subtler version of that scenario is a user with JavaScript enabled, who has a slow

Syntax

- Since it’s a lightweight scripting language, JavaScript has some features (such as dynamic typing) that are especially helpful to the novice programmer. However, a novice programmer faces challenges when he or she tries to use JavaScript in the same way as a full object-oriented language such as Java, as JavaScript’s object features (such as prototypes and inline functions) are quite unlike those of more familiar languages.

- We will briefly cover the fundamental syntax for the most common programming constructs including **variables**, **assignment**, **conditionals**, **loops**, and **arrays** before moving on to advanced topics such as events and classes.

```
var x;      ←—— a variable x is defined
var y = 0;  ←—— y is defined and initialized to 0
y = 4;      ←—— y is assigned the value of 4
```

- JavaScript's reputation for being quirky not only stems from its strange way of implementing object-oriented principles, but also from some odd syntactic *gotchas* that every JavaScript developer will eventually encounter, some of which include:
  - Everything is type sensitive, including function, class, and variable names.
  - The scope of variables in blocks is not supported. This means variables declared inside a loop may be accessible outside of the loop, counter to what one would expect.
  - There is a === operator, which tests not only for equality but type equivalence.
  - Null and undefined are two distinctly different states for a variable.
  - Semicolons are not required, but are permitted (and encouraged).
  - There is no integer type, only number, which means floating-point rounding errors are prevalent even with values intended to be integers.

Variables

- **Variables** in JavaScript are **dynamically typed**, meaning a variable can be an integer, and then later a string, then later an object, if so desired. This simplifies variable declarations, so that we do not require the familiar type fields like *int*, *char*, and *String*. Instead, to declare a variable x, we use the var keyword, the name, and a semicolon as shown in Figure 6.13.

```
var x;      ←—— a variable x is defined
var y = 0;  ←—— y is defined and initialized to 0
y = 4;      ←—— y is assigned the value of 4
```

- If we specify no value, then (being typeless) the default value is undefined. **Assignment** can happen at declaration-time by appending the value to the declaration, or at run time with a simple right-to-left assignment as illustrated in Figure 6.13.

- This syntax should be familiar to those who have programmed in languages like C and Java. In addition, the conditional assignment operator, shown in Figure 6.14, can also be used to assign based on condition, although its use is sometimes discouraged.

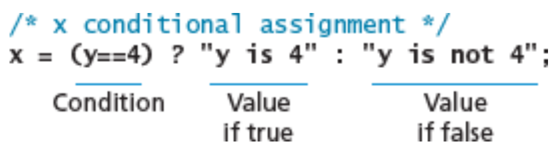
Comparison Operators

- The core of any programming language is the ability to distill things down to Boolean statements where something is either true or false. JavaScript is no exception and comes equipped with a number of operators to compare two values, listed in Table 6.1

Operator	Description	Matches (x=9)
`==`	Equals	(x==9) is true (x=="9") is true
`===`	Exactly equals, including type	(x==="9") is false (x===9) is true
`< , >`	Less than, greater than	(x<5) is false
`<= , >=`	Less than or equal, greater than or equal	(x<=9) is true
`!=`	Not equal	(4!=x) is true
`!==`	Not equal in either value or type	(x!=="9") is true (x!==9) is false

TABLE 6.1 Comparison Operators

- These operators will be familiar to those of you who have programmed in PHP, C, or Java. These comparison operators are used in conditional, loop, and assignment statements.



```
/* x conditional assignment */
x = (y==4) ? "y is 4" : "y is not 4";
```

Logical Operators

Comparison operators are useful, but without being able to combine several together, their usefulness would be severely limited. Therefore, like most languages JavaScript includes Boolean operators, which allow us to build complicated expressions. The Boolean operators and, or, and not and their truth tables are listed in Table 6.2. Syntactically they are represented with && (and), || (or), and ! (not).

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND Truth Table

A	B	A || B
T	T	T
T	F	T
F	T	T
F	F	F

OR Truth Table

A	! A
T	F
F	T

NOT Truth Table

TABLE 6.2 AND, OR, and NOT Truth Tables

Conditionals

JavaScript's syntax is almost identical to that of PHP, Java, or C when it comes to conditional structures such as if and if else statements. In this syntax the condition to test is contained within ( ) brackets with the body contained in { } blocks. Optional else if statements can follow, with an else ending the branch. Listing 6.4uses a conditional to set a greeting variable, depending on the hour of the day.

```
var hourOfDay;   // var to hold hour of day, set it later...
var greeting;   // var to hold the greeting message.
if (hourOfDay > 4 && hourOfDay < 12){
    // if statement with condition
   greeting =  "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 20){
   // optional else if
   greeting =  "Good Afternoon";
}
else{ // optional else branch
   greeting = "Good Evening";
}
```

LISTING 6.4 Conditional statement setting a variable based on the hour of the day

Loops

Like conditionals, loops use the ( ) and { } blocks to define the condition and the body of the loop.

While Loops

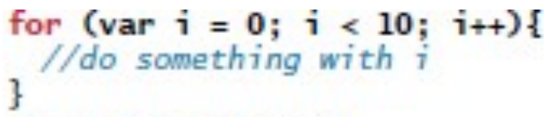
The most basic loop is the while loop, which loops until the condition is not met. Loops normally initialize a loop control variable before the loop, use it in the condition, and modify it within the loop. One must be sure that the variables that make up the condition are updated inside the loop (or elsewhere) to avoid an infinite loop!

```
var          i=0;
while(i < 10){
```

```
//do something with i
i++;
}
```

For Loops

A **for loop** combines the common components of a loop: initialization, condition, and post-loop operation into one statement. This statement begins with the for keyword and has the components placed between ( ) brackets, semicolon (;) separated as shown in Figure 6.15.



```
for (var i = 0; i < 10; i++){
  //do something with i
}
```

FIGURE 6.15 For loop

Functions

- **Functions** are the building block for modular code in JavaScript, and are even used to build **pseudo-classes**,They are defined by using the reserved word function and then the function name and (optional) parameters.
- Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type. Therefore a function to raise x to the yth power might be defined as:

```
function power(x,y){
var pow=1;
for (var i=0;i<y;i++){
pow = pow*x;
}
return pow;
}
```

  And called as power(2,10);
- With new programmers there is often confusion between defining a function and calling the function. Remember that when actually using the keyword function, we are defining what the function does.

Alert

- The alert() function makes the browser show a pop-up to the user, with whatever is passed being the message displayed. The following JavaScript code displays a simple hello world message in a pop-up:

  alert ("Good Morning");

- The pop-up may appear different to each user depending on their browser configuration. What is universal is that the pop-up obscures the underlying web page, and no actions can be done until the pop-up is dismissed Alerts are not used in production code, but are a useful tool for debugging andillustration purposes.

Errors Using Try and Catch

- When the browser's JavaScript engine encounters an error, it will *throw* an **exception**. These exceptions interrupt the regular, sequential execution of the program and can stop the JavaScript engine altogether. However, you can optionally catch these errors preventing disruption of the program using the **try–catch block** as shown in Listing 6.5.

```
try {
  nonexistantfunction("hello");
}
catch(err) {
  alert("An exception was caught:" + err);
}
```

LISTING 6.5 Try-catch statement

Throwing Your Own Exceptions

- Although try-catch can be used exclusively to catch built-in JavaScript errors, it can also be used by your programs, to throw your own messages.
- The throw keyword stops normal sequential execution, just like the built-in exceptions as shown in Listing 6.6.

```
try {
   var x = -1;
   if (x<0)
      throw "smallerthan0Error";
}
catch(err){
   alert (err + "was thrown");
}
```

LISTING 6.6 Throwing a user-defined exception

- The general consensus in software development is that try-catch and throw statements should be used for *abnormal* or *exceptional* cases in your program. They should not be used as a normal way of controlling flow, although no formal mechanismexists to enforce that idea.

- We will generally avoid try-catch statements inour code unless illustrative of some particular point. Listing 6.6 demonstrates the throwing of a user-defined exception as a string.
- In reality any object can be thrown, although in practice a string usually suffices. It should be noted that throwing an exception disrupts the sequential executionof a program. That is, when the exception is thrown all subsequent code is notexecuted until the catch statement is reached. This reinforces why try-catch is forexceptional cases.

JavaScript Objects

- JavaScript is not a full-fledged object-oriented programming language. It does not have classes per se, and it does not support many of the patterns you'd expect from an object-oriented language like inheritance and polymorphism in a straightforward way.
- The language does, however, support objects. User-defined objects are declared in a slightly odd way to developers familiar with languages like C++ or Java, so the syntax to build pseudo-classes can be challenging. Nonetheless the advantages of encapsulating data and methods into objects outweigh the syntactic hurdle you will have to overcome.
- Objects can have **constructors**, **properties**, and **methods** associated with them, and are used very much like objects in other object-oriented languages. There are objects that are included in the JavaScript language; you can also define your own kind of objects.

Constructors

Normally to create a new object we use the new keyword, the class name, and ( ) brackets with *n* optional parameters inside, comma delimited as follows: For some classes, shortcut constructors are defined, which can be confusing ifwe are not aware of them. For example, a String object can be defined with theshortcut

6.5.2 Properties

Each object might have properties that can be accessed, depending on its definition. When a property exists, it can be accessed using **dot notation** where a dot between the instance name and the property references that property **Methods** Objects can also have methods, which are functions associated with an instance of an object. These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method.

```
someObject.doSomething();
```

Methods may produce different output depending on the object they are associated with because they can utilize the internal properties of the object.

Objects Included in JavaScript

A number of useful objects are included with JavaScript. These include Array, Boolean, Date, Math, String, and others. In addition to these, JavaScript can also access Document Object Model (DOM) objects that correspond to the content of a page's HTML. These DOM objects let JavaScript code access and modify HTML and CSS properties of a page dynamically.

Arrays

- Arrays are one of the most used data structures, and they have been included in JavaScript as well. In practice, this class is defined to behave more like a linked list in that it can be resized dynamically, but the implementation is browser specific, meaning the efficiency of insert and delete operations is unknown.
- Arrays will be the first objects we will examine. Objects can be created using the new syntax and calling the object constructor. The following code creates a new, empty array named greetings:

  ```
  var greetings = new Array();
  ```

  To initialize the array with values, the variable declaration would look like the following:

  ```
  var greetings = new Array("Good Morning", "Good Afternoon");
  ```

  or, using the square bracket notation:

  ```
  var greetings = ["Good Morning", "Good Afternoon"];
  ```

While you should be careful to employ consistency in your own array declarations, it's important to familiarize yourself with notation that may be used by others. Teams should agree on some standards in this area.

Accessing and Traversing an Array

- To access an element in the array you use the familiar square bracket notation from Java and C-style languages, with the index you wish to access inside the brackets. alert (greetings[0]);

- One of the most common actions on an array is to traverse through the items sequentially. The following for loop quickly loops through an array, accessing the ith element each time using the Array object's length property to determine the maximum valid index. It will alert

  "Good Morning" and "Good Afternoon" to

  the user.

  for (var i = 0; i < greetings.length; i++){

  alert(greetings[i]);

  }

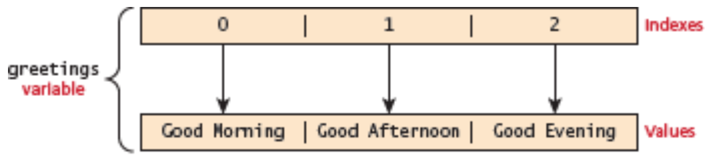


FIGURE 6.16 JavaScript array with indexes and values illustrated

Modifying an Array

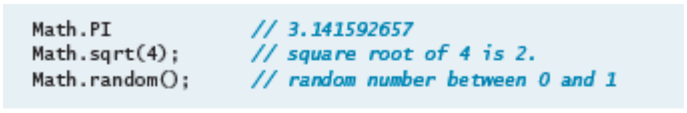
To add an item to an existing array, you can use the push method.

greetings.push("Good Evening");

Figure 6.16 illustrates an array with indexes and the corresponding values. The pop method can be used to remove an item from the back of an array. Additional methods that modify arrays include concat(), slice(), join(), reverse(), shift(), and sort(). A full accounting of all these methods is beyond the scope of a single chapter, but as you begin to use arrays you should explore them.

Math

The **Math class** allows one to access common mathematic functions and common values quickly in one place. This static class contains methods such as max(), min(), pow(), sqrt(), and exp(), and trigonometric functions such as sin(), cos(), and arctan(). In addition, many mathematical constants are defined such as PI, E (Euler's number), SQRT2, and some others as shown in Listing 6.7.



```
Math.PI            // 3.141592657
Math.sqrt(4);      // square root of 4 is 2.
Math.random();     // random number between 0 and 1
```

LISTING 6.7 Some constants and functions in the Math object

String

The **String class** has already been used without us even knowing it. That is because it is core to communicating with the user. Since it is so common, shortcuts have been defined for creating and concatenating strings. While one can use the new syntax to create a String object, it can also be defined using quotes as follows:

```
var greet = new String("Good"); // long form constructor
var greet = "Good"; // shortcut constructor
```

A common need is to get the length of a string. This is achieved through thelength property (just as in arrays).

```
alert (greet.length); // will display "4"
```

Another common way to use strings is to concatenate them together. Since
this is so common, the + operator has been overridden to allow for concatenation
in place.

```
var str = greet.concat("Morning"); // Long form concatenation
var str = greet + "Morning"; // + operator concatenation
```

Many other useful methods exist within the String class, such as accessing a
single character using charAt(), or searching for one using indexOf(). Strings allow splitting a string into an array, searching and matching with split(), search(), and match() methods.

Date

While not critical to JavaScript, the Date class is yet another helpful included object you should be aware of. It allows you to quickly calculate the current date or create date objects for particular dates. To display today's date as a string, we would simply create a new object and use the toString() method.

```
var d = new Date();
// This outputs Today is Mon Nov 12 2012 15:40:19 GMT-0700
alert ("Today is "+ d.toString());
```

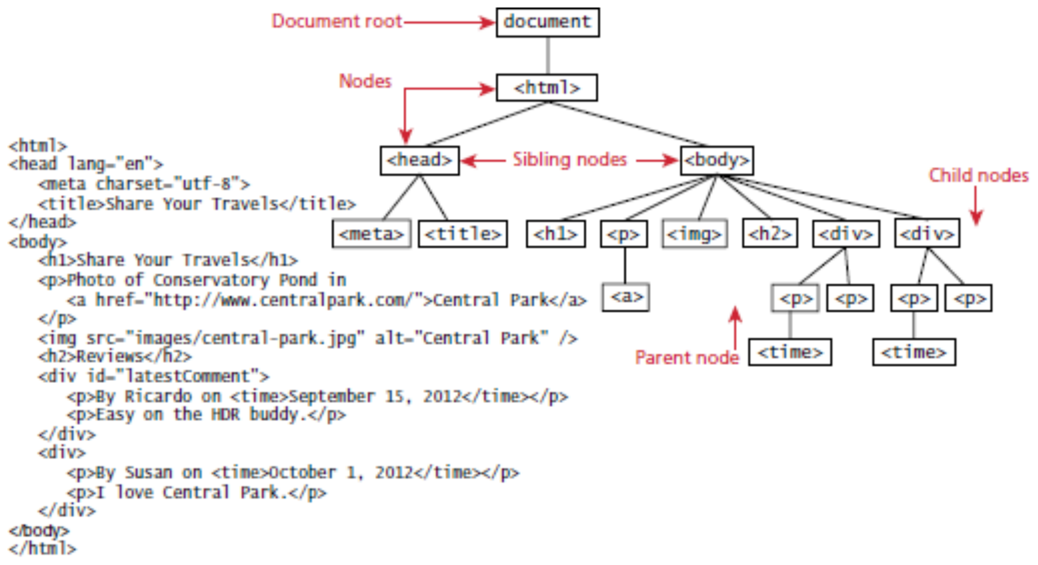
Window Object

The window object in JavaScript corresponds to the browser itself. Through it, you can access the current page's URL, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows. In fact, the alert() function mentioned earlier is actually a method of the window object.

The Document Object Model (DOM)

JavaScript is almost always used to interact with the HTML document in which it is contained. As such, there needs to be some way of programmatically accessing the elements and attributes within the HTML. This is accomplished through a programming interface (API) called the **Document Object Model(DOM)**.

We already know all about the DOM, but by another name. The tree structure from Chapter 2 (shown again in Figure 6.17) is formally called the **DOM Tree** with the root, or topmost object called the **Document Root**.

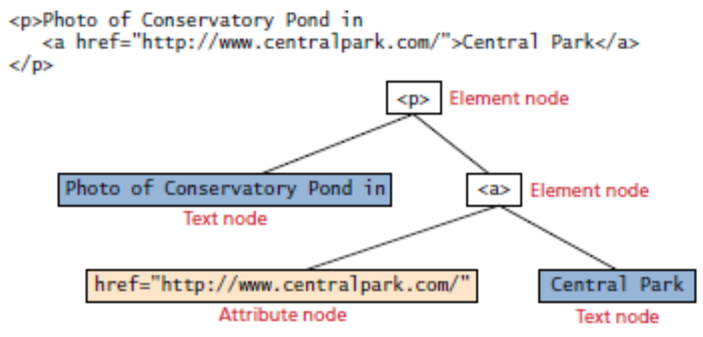


```
<html>
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>September 15, 2012</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```

FIGURE 6.17 DOM tree

Nodes

In the DOM, each element within the HTML document is called a **node**. If the DOM is a tree, then each node is an individual branch. There are element nodes, text nodes, and attribute nodes, as shown in Figure 6.18. All nodes in the DOM share a common set of properties and methods. Thus, most of the tasks that we typically perform in JavaScript involve finding a node, and then accessing or modifying it via those properties and methods. The most important of these are shown in Table 6.3.



<p>Photo of Conservatory Pond in
  <a href="http://www.centralpark.com/">Central Park</a>
</p>
<p>
Element node
Photo of Conservatory Pond in
Text node
<a>
Element node
href="http://www.centralpark.com/"
Attribute node
Central Park
Text node

FIGURE 6.18 DOM nodes

Property	Description
attributes	Collection of node attributes
childNodes	A NodeList of child nodes for this node
firstChild	First child node of this node
lastChild	Last child of this node
nextSibling	Next sibling node for this node
nodeName	Name of the node
nodeType	Type of the node
nodeValue	Value of the node
parentNode	Parent node for this node
previousSibling	Previous sibling node for this node.

TABLE 6.3 Some Essential Node Object Properties

Document Object

The DOM document object is the root JavaScript object representing the entire HTML document. It contains some properties and methods that we will use extensively in our development and is globally accessible as document. The attributes of this object include some information about the page including doctype and inputEncoding. Accessing the properties is done through the dot notation as illustrated on the next page.

Method	Description
createAttribute()	Creates an attribute node
createElement()	Creates an element node
createTextNode()	Creates a text node
getElementById(id)	Returns the element node whose id attribute matches the passed id parameter
getElementsByTagName(name)	Returns a NodeList of elements whose tag name matches the passed name parameter

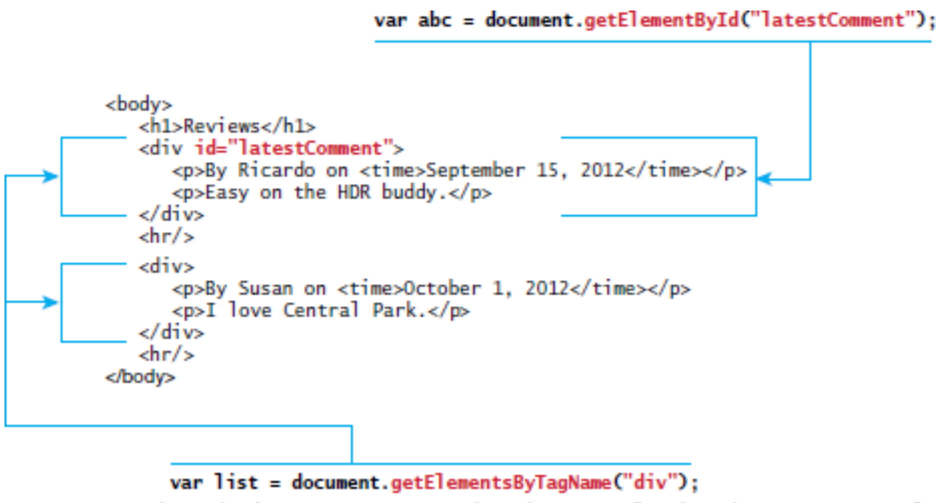
TABLE 6.4 Some Essential Document Object Methods

var a = document.doctype.name;

// specify the page encoding, for example ISO-8859-1

var b = document.inputEncoding;

In addition to these moderately useful properties, there are some essential methods (see Table 6.4) you will use all the time. They include getElementByTagName() and the indispensable getElementById(). While the former method returns an array of DOM nodes (called a NodeList) matching the tag, the latter returns a single DOM element (covered below), that matches the id passed as a parameter as illustrated in Figure 6.19.



```
var abc = document.getElementById("latestComment");

<body>
   <h1>Reviews</h1>
   <div id="latestComment">
      <p>By Ricardo on <time>September 15, 2012</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>October 1, 2012</time></p>
      <p>I love Central Park.</p>
   </div>
   <hr/>
</body>

var list = document.getElementsByTagName("div");
```

FIGURE 6.19 Relationship between HTML tags and getElementByID() and getElementsByTagName()

Selectors are generally poorly supported in pure JavaScript across the multitude of browsers and platforms available. The method getElementById() is universally implemented and thus used extensively. The newer querySelector() and querySelectorAll() methods allow us to query for DOM elements much the same way we specify CSS styles, but are only implemented in the newest browsers.3 For this reason jQuery selectors (in Chapter 15) will be introduced to show more powerful selector mechanisms. Until then we will rely on getElementByID().

Element Node Object

- The represents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element. As you may already have figured out, an element can itself contain more elements. Since IDs must be unique in an HTML document, getElementByID() returns a single node, rather than a set of results which is the case with other selector functions.
- The returned Element Node object has the node properties shown in Table 6.3. It also has a variety of additional properties, the most important of which are shown in Table 6.5.
-

Property	Description
className	The current value for the class attribute of this HTML element.
id	The current value for the id of this element.
innerHTML	Represents all the things inside of the tags. This can be read or written to and is the primary way in which we update particular <div> elements using JavaScript.
style	The style attribute of an element. We can read and modify this property.
tagName	The tag name for the element.

TABLE 6.5 Some Essential Element Node Properties

- While these properties are available for all HTML elements, there are some HTML elements that have additional properties that can be accessed. Table 6.6 lists some common additional properties and the HTML tags that have these properties.

Property	Description	Tags
href	The href attribute used in a tag to specify a URL to link to.	a
name	The name property is a bookmark to identify this tag. Unlike id, which is available to all tags, name is limited to certain form-related tags.	a, input, textarea, form
src	Links to an external URL that should be loaded into the page (as opposed to href, which is a link to follow when clicked)	img, input, iframe, script
value	The value is related to the value attribute of input tags. Often the value of an input field is user defined, and we use value to get that user input.	input, textarea, submit

TABLE 6.6 Some Specific HTML DOM Element Properties for Certain Tag Types

Modifying a DOM Element

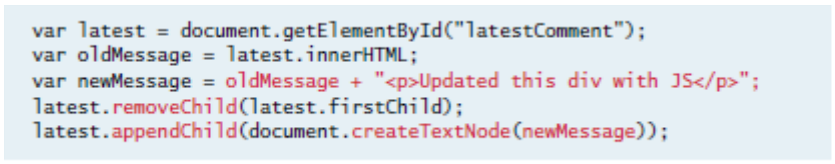
- In many introductory JavaScript textbooks the document.write() method is used to create output to the HTML page from JavaScript. While this is certainly valid, it always creates JavaScript at the bottom of the existing HTML page, and in practiceis good for little more than debugging.
- The modern JavaScript programmer will wantto write to the HTML page, but in a particular location, not always at the bottom.Using the DOM document and HTML DOM element objects, we can doexactly that using the innerHTML property as shown in Listing 6.8 (using the HTMLshown in Figure 6.19)

```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
latest.innerHTML = oldMessage + "<p>Updated this div with JS</p>";
```

LISTING 6.8 Changing the HTML using innerHTML

A More Verbose Technique

- ➢ Although the innerHTML technique works well (and is very fast), there is a more verbose technique available to us that builds output using the DOM.
- ➢ This more explicit technique has the advantage of ensuring that only valid markup is created,while the innerHTML could output badly formed HTML. DOM functions create-TextNode(), removeChild(), and appendChild() allow us to modify an element ina more rigorous way as shown in Listing 6.9.



```
var latest = document.getElementById("latestComment");
var oldMessage = latest.innerHTML;
var newMessage = oldMessage + "<p>Updated this div with JS</p>";
latest.removeChild(latest.firstChild);
latest.appendChild(document.createTextNode(newMessage));
```

LISTING 6.9 Changing the HTML using createTextNode( ) and appendChild( )

Changing an Element’s Style

- ➢ We can also modify the style associated with a particular block. We can add or remove any style using the style or className property of the Element node, which is something that you might want to do to dynamically change the appearance of an element. Its usage is shown below to change a node’s background color and add a three-pixel border.
- ➢ Armed with knowledge of CSS attributes you can easily change any style attribute. Note that the style property is itself an object, specifically a CSSStyleDeclaration type, which includes all the CSS attributes as properties and computes the current style from inline, external, and embedded styles. Although you can directly access CSS style elements we suggest you use classes whenever possible.
- ➢ The className property is normally a better choice, because it allows the styles to be created outside the code, and thus be better accessible to designers. Using this model we would change the background color by having two styles defined, and changing them in JavaScript code.

Additional Properties

- ➢ In addition to the global properties present in all tags, there are additional methods available when dealing with certain tags. Table 6.6 lists a few common ones.
- ➢ To get the password out of the following input field and alert the user

  <input type='password' name='pw' id='pw' />

  We would use the following JavaScript code:

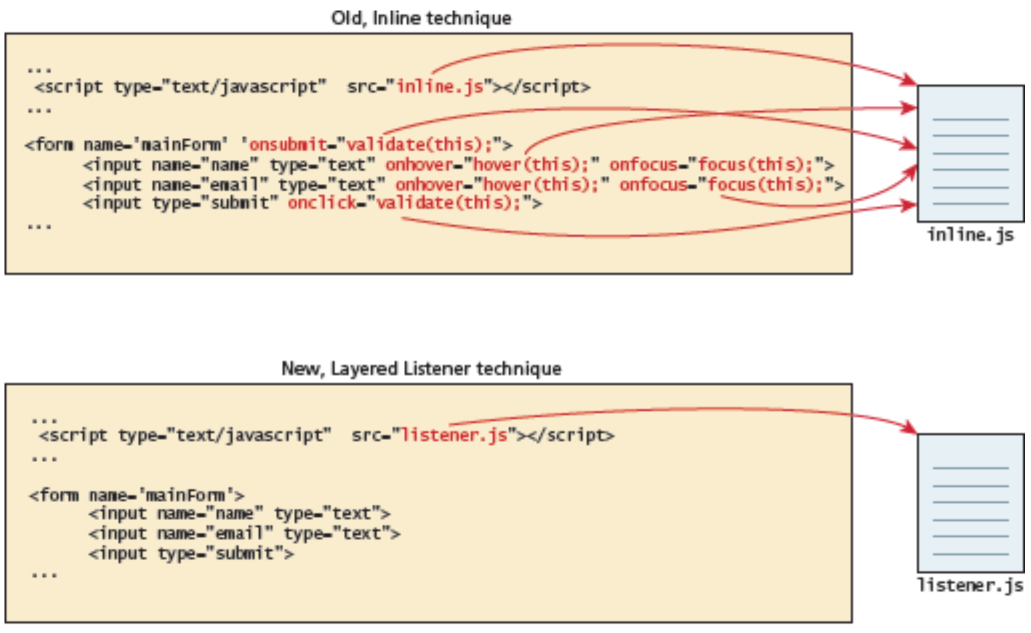
  var pass = document.getelementById("pw");

alert (pass.value);

- It should be obvious how getting the src or href properties out of appropriate tags could also be done. We leave it as an exercise to the reader.

JavaScript Events

- At the core of all JavaScript programming is the concept of an **event**. A JavaScript event is an action that can be detected by JavaScript. Many of them are initiated by user actions but some are generated by the browser itself. We say then that an event is *triggered* and then it can be *caught* by JavaScript functions, which then do something in response.
- In the original JavaScript world, events could be specified right in the HTML markup with *hooks* to the JavaScript code (and still can).4 This mechanism was popular throughout the 1990s and 2000s because it worked. As more powerful frameworks were developed, and website design and best practices were refined, this original mechanism was supplanted by the **listener** approach.
- A visual comparison of the old and new technique is shown in Figure 6.20. Note how the old method weaves the JavaScript right inside the HTML, while the listener technique has removed JavaScript from the markup, resulting in cleaner,easier to maintain HTML code.



```
...
<script type="text/javascript"  src="inline.js"></script>
...
<form name='mainForm' 'onsubmit="validate(this);">
      <input name="name" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input name="email" type="text" onhover="hover(this);" onfocus="focus(this);">
      <input type="submit" onclick="validate(this);">
...
```

```
...
<script type="text/javascript"  src="listener.js"></script>
...
<form name='mainForm'>
      <input name="name" type="text">
      <input name="email" type="text">
      <input type="submit">
...
```

FIGURE 6.20 Inline hooks versus the Layered Listener technique

Inline Event Handler Approach

- JavaScript events allow the programmer to react to user interactions. In early web development, it made sense to weave code and HTML together and to this day, inline

JavaScript calls are intuitive. For example, if you wanted an alert to pop-up ,when clicking a <div> you might program:

<div id="example1" onclick="alert('hello')">Click for pop-up</div>

- In this example the HTML attribute onclick is used to attach a handler to that event. When the user clicks the <div>, the event is triggered and the alert is executed.
- The problem with this type of programming is that the HTML markup and the corresponding JavaScript logic are woven together.
- This reduces the ability of designers to work separately from programmers, and generally complicates maintenance of applications. The better way to program this is to remove the JavaScript from the HTML.

Listener Approach

- The approach shown in Listing 6.10 is widely supported by all browsers. Thefirst line in the listing creates a temporary variable for the HTML element that willtrigger the event.

```
var greetingBox = document.getElementById('example1');
greetingBox.onclick = alert('Good Morning');
```

LISTING 6.10 The "old" style of registering a listener.

- The next line attaches the <div> element's onclick event to theevent handler, which invokes the JavaScript alert() method (and thus annoys theuser with a pop-up hello message).
- The main advantage of this approach is that thiscode can be written anywhere, including an external file that helps *uncouple* theHTML from the JavaScript. However, the one limitation with this approach (andthe inline approach) is that only one handler can respond to any given elementevent.
- The use of addEventListener() shown in Listing 6.11 was introduced in DOM Version 2, and as such is unfortunately not supported by IE 8 or earlier.

```
var greetingBox = document.getElementById('example1');
greetingBox.addEventListener('click', alert('Good Morning'));
greetingBox.addEventListener('mouseOut', alert('Goodbye'));

// IE 8
greetingBox.attachEvent('click', alert('Good Morning'));
```

LISTING 6.11 The "new" DOM2 approach to registering listeners.

- This approach has all the other advantages of the approach shown in Listing 6.10, and has the additional advantage that multiple handlers can be assigned to a single object's event

- The examples in Listing 6.10 and Listing 6.11 simply used the built-in JavaScript alert() function. What if we wanted to do something more elaborate when an event is triggered? In such a case, the behavior would have to be encapsulated within a function, as shown in Listing 6.12.

```
function displayTheDate() {
   var d = new Date();
   alert ("You clicked this on "+ d.toString());
}
var element = document.getElementById('example1');
element.onclick = displayTheDate;

// or using the other approach
element.addEventListener('click',displayTheDate);
```

LISTING 6.12 Listening to an event with a function

- An alternative to that shown in Listing 6.12 is to use an anonymous function (that is, one without a name), as shown in Listing 6.13. This approach is especially common when the event handling function will only ever be used as a listener

```
var element = document.getElementById('example1');
element.onclick = function() {
   var d = new Date();
   alert ("You clicked this on " + d.toString());
};
```

LISTING 6.13 Listening to an event with an anonymous function

Event Object

- No matter which type of event we encounter, they are all **DOM event objects** and the event handlers associated with them can access and manipulate them. Typically we see the events passed to the function handler as a parameter named *e*.

```
function someHandler(e) {
// e is the event that triggered this handler.
}
```

- These objects have many properties and methods. Many of these properties are not used, but several key properties and methods of the event object are worth knowing.
  - **Bubbles**. The bubbles property is a Boolean value. If an event's bubble property is set to true then there must be an event handler in place to handle the event or it will bubble up to its parent and trigger an event handler there. If the parent has no handler it continues to bubble up until it hits the document root, and then it goes away, unhandled.
  - **Cancelable**. The Cancelable property is also a Boolean value that indicates whether or not the event can be cancelled. If an event is cancelable, then the

default action associated with it can be canceled. A common example is a user clicking on a link. The default action is to follow the link and load the new page.

- **preventDefault**. A cancelable default action for an event can be stopped using the preventDefault()method as shown in Listing 6.14. This is a common practice when you want to send data asynchronously when a form is submitted, for example, since the default event of a form submit click is to post to a new URL (which causes the browser to refresh the entire page).

```
function submitButtonClicked(e) {
  if (e.cancelable){
    e. preventDefault();
  }
}
```

LISTING 6.14 A sample event handler function that prevents the default event

Event Types

- The most obvious event is the click event, but JavaScript and the DOM support several others. In actuality there are several classes of event, with several types of event within each class specified by the W3C. The classes are mouse events, keyboard events, form events, and frame events.

Mouse Events

- Mouse events are defined to capture a range of interactions driven by the mouse. These can be further categorized as mouse click and mouse move events.
- Table 6.7 lists the possible events one can listen for from the mouse. Interestingly, many mouse events can be sent at a time.
- The user could be moving the mouse off one <div> and onto another in the same moment, triggeringonmouseon and onmouseout events as well as the onmousemove event. The Cancelable and Bubbles properties can be used to handle these complexities.

Keyboard Events

- Keyboard events are often overlooked by novice web developers, but are important tools for power users.
- Table 6.8 lists the possible keyboard events These events are most useful within input fields. We could for example validatean email address, or send an asynchronous request for a dropdown list of suggestionswith each key press.

```
<input type="text" id="keyExample">
```

➢ The input box above, for example, could be listened to and each key pressed echoed back to the user as an alert as shown in Listing 6.15.

Event	Description
onclick	The mouse was clicked on an element
ondblclick	The mouse was double clicked on an element
onmousedown	The mouse was pressed down over an element
onmouseup	The mouse was released over an element
onmouseover	The mouse was moved (not clicked) over an element
onmouseout	The mouse was moved off of an element
onmousemove	The mouse was moved while over an element

TABLE 6.7 Mouse Events in JavaScript

```
document.getElementById("keyExample").onkeydown = function
myFunction(e){
   var keyPressed=e.keyCode;      //get the raw key code
   var character=String.fromCharCode(keyPressed); //convert to string
   alert("Key " + character + " was pressed");
}
```

LISTING 6.15 Listener that hears and alerts keypresses

Event	Description
onkeydown	The user is pressing a key (this happens first)
onkeypress	The user presses a key (this happens after onkeydown)
onkeyup	The user releases a key that was down (this happens last)

TABLE 6.8 Keyboard Events in JavaScript

Form Events

➢ Forms are the main means by which user input is collected and transmitted to the server. Table 6.9 lists the different form events. The events triggered by forms allow us to do some timely processing in response to user input.

➢ The most common JavaScript listener for forms is the onsubmit event. In the code below we listen for that event on a form with id loginForm. If the password field (with id pw) is blank, we prevent submitting to the server using preventDefault() and alert the user.

Event	Description
onblur	A form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
onchange	Some `<input>`, `<textarea>`, or `<select>` field had their value change. This could mean the user typed something, or selected a new choice.
onfocus	Complementing the `onblur` event, this is triggered when an element gets focus (the user clicks in the field or tabs to it).
onreset	HTML forms have the ability to be reset. This event is triggered when that happens.
onselect	When the users selects some text. This is often used to try and prevent copy/paste.
onsubmit	When the form is submitted this event is triggered. We can do some prevalidation when the user submits the form in JavaScript before sending the data on to the server.

TABLE 6.9 Form Events in JavaScript

- Otherwise we do nothing, which allows thedefault event to happen (submitting the form) as shown in Listing 6.16. Section 6.8 will examine form event handling in more detail.

```
document.getElementById("loginForm").onsubmit = function(e){
   var pass = document.getElementById("pw").value;
   if(pass==""){
      alert ("enter a password");
      e.preventDefault();
   }
}
```

LISTING 6.16 Catching the onsubmit event and validating a password to not be blank

Frame Events

- Frame events (see Table 6.10) are the events related to the browser frame that contains your web page. The most important event is the onload event, which tells us an object is loaded and therefore ready to work with.
- In fact, every nontrivial event listener you write requires that the HTML be fully loaded. However, a problem can occur if the JavaScript tries to reference a particular <div> in the HTML page that has not yet been loaded. If the code attempts to set up a listener on this not-yet-loaded <div>, then an error will be triggered.

Event	Description
`onabort`	An object was stopped from loading
`onerror`	An object or image did not properly load
`onload`	When a document or object has been loaded
`onresize`	The document view was resized
`onscroll`	The document view was scrolled
`onunload`	The document has unloaded

TABLE 6.10 Frame Events in JavaScript

- For this reason it is common practice to use the window.onload event to trigger the execution of the rest of the page's scripts.

```
window.onload= function(){
//all JavaScript initialization here}
```

Forms

To illustrate some form-related JavaScript concepts, consider the simple HTMLform depicted in Listing 6.17.

```
<form action='login.php' method='post' id='loginForm'>
  <input type='text' name='username' id='username'/>
  <input type='password' name='password' id='password'/>
  <input type='submit'></input>
</form>
```

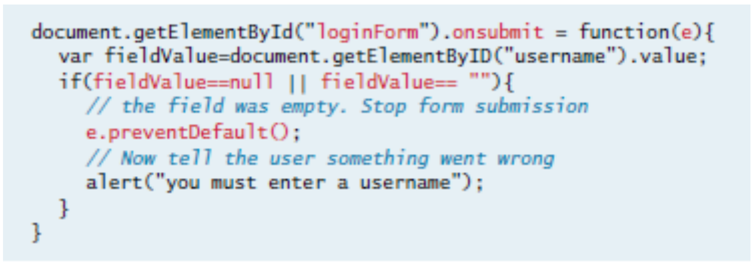
LISTING 6.17 A basic HTML form for a login example

Validating Forms

- Form validation is one of the most common applications of JavaScript. Writing code to prevalidate forms on the client side will reduce the number of incorrect submissions, thereby reducing server load. Although validation must still happen on the server side (in case JavaScript was circumvented), JavaScript prevalidation is a best practice.
- There are a number of common validation activities including email validation, number validation, and data validation. In practice regular expressions are used ,and allow for more complete and concise scripts to validate particular fields. However, the novice programmer may not be familiar or comfortable using regex, and will often resort to copying a regex from the Internet, without understanding how it works, and therefore, will be unable to determine if it is correct

Empty Field Validation

- A common application of a client-side validation is to make sure the user entered something into a field.
- There's certainly no point sending a request to log in if the username was left blank, so why not prevent the request from working? The way to check for an empty field in JavaScript is to compare a value to both null and the empty string ("") to ensure it is not empty, as shown in Listing 6.18 related JavaScript concepts, consider the simple HTMLform depicted in Listing 6.17.

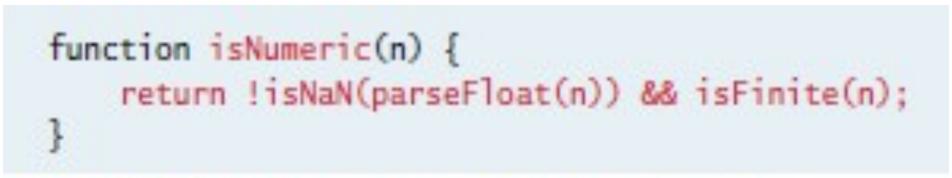


```
document.getElementById("loginForm").onsubmit = function(e){
  var fieldValue=document.getElementByID("username").value;
  if(fieldValue==null || fieldValue== ""){
    // the field was empty. Stop form submission
    e.preventDefault();
    // Now tell the user something went wrong
    alert("you must enter a username");
  }
}
```

LISTING 6.18 A simple validation script to check for empty fields

Number Validation

- Number validation can take many forms. You might be asking users for their agefor example, and then allow them to type it rather than select it. Unfortunately, nosimple functions exist for number validation like one might expect from a fullfledgedlibrary. Using parseInt(), isNAN(), and isFinite().
- Part of the problem is that JavaScript is dynamically typed, so "2" !== 2, but"2"==2. jQuery and a number of programmers have worked extensively on this issue and have come up with the function isNumeric()
- shown in Listing 6.19. Note: Thisfunction will not parse "European" style numbers with commas (i.e., 12.00 vs. 12,00).



```
function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}
```

LISTING 6.19 A function to test for a numeric value

Submitting Forms

- Submitting a form using JavaScript requires having a node variable for the form element. Once the variable, say, formExample is acquired, one can simply call the submit() method:

```
var formExample = document.getElementById("loginForm");
formExample.submit();
```

- This is often done in conjunction with calling preventDefault() on the onsubmit event. This can be used to submit a form when the user did not click the submit button, or to submit forms with no submit buttons at all (say we want to use an image instead). Also, this can allow JavaScript to do some processing before submitting a form, perhaps updating some values before transmitting.
- It is possible to submit a form multiple times by clicking buttons quickly, which means your server-side scripts should be designed to handle that eventuality. Clicking a submit button twice on a form should not result in a double order, double email, or double account creation, so keep that in mind as you design your applications.

What Is Server-Side Development?

- The basic hosting of your files is achieved through a web server whose responsibilities are described below. Server-side development is much more than web hosting: it involves the use of a programming technology like PHP or ASP.NET to create scripts that dynamically generate content.
- It is important to remember that when developing server-side scripts, you are writing software, just like a C or Java programmer would do, with the major distinction that your software runs on a web server and uses the HTTP request response loop for most interactions with the clients.
- This distinction is significant, since it invalidates many classic software development patterns, and requires different thinking for many seemingly simple software principles like data storage and memory management.

Comparing Client and Server Scripts

- The fundamental difference between client and server scripts is that in a client-side script the code is executed on the client browser, whereas in a server-side script, it is executed on the web server.
- As you saw clientside JavaScript code is downloaded to the client and is executed there. The server sends the JavaScript (that the user could look at), but you have no guarantee that the script will even execute.
- In contrast, server-side source code remains hidden from the client as it is processed on the server. The clients never get to see the code, just the HTML output from the script. Figure 8.1 illustrates how client and server scripts differ.

- The location of the script also impacts what resources it can access. Server scripts cannot manipulate the HTML or DOM of a page in the client browser as is possible with client scripts.
- Conversely, a server script can access resources on the web server whereas the client cannot. Understanding where the scripts reside and what they can access is essential to writing quality web applications.

Server-Side Script Resources

- A server-side script can access any resources made available to it by the server.
- The most commonly used resource is **data storage**, often in the form of a connection to a database management system.
- A **database management system (DBMS)** is a software system for storing, retrieving, and organizing large amounts of data.

A Web Server's Responsibilities

- As you learned in Chapter 1, in the client-server model the server is responsible for answering all client requests. No matter how static or simple the website is, there must be a web server somewhere configured to answer requests for that domain.
- Once a web server is configured and the IP address associated through a DNS server (see Chapter 1), it can then start listening for and answering HTTP requests.
- In the very simplest case the server is hosting static HTML files, and in response to a request sends the content of the file back to the requester.
- A web server has many responsibilities beyond responding to requests for HTML files. These include handling HTTP connections, responding to requests for static and dynamic resources, managing permissions and access for certain resources, encrypting and compressing data, managing multiple domains and URLs, managing database connections, cookies, and state, and uploading and managing files.
- As you will be using the LAMP software stack, which refers to the Linux operating system, the Apache web server, the MySQL DBMS, and the PHP scripting language.
- Outside of the chapters on security and deployment, this book will not examine the Linux operating system in any detail.
- However, since the Apache web server is an essential part of the web development pipeline, one should have some insight into how it works and how it interacts with PHP.

JQUERY

Introduction,

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.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

jQuery is a small, light-weight and fast JavaScript library. It is cross-platform and supports different types of browsers. It is also referred as ?write less do more? because it takes a lot of common tasks that requires many lines of JavaScript code to accomplish, and binds them into methods that can be called with a single line of code whenever needed. It is also very useful to simplify a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

- jQuery is a small, fast and lightweight JavaScript library.
- jQuery is platform-independent.
- jQuery means "write less do more".
- jQuery simplifies AJAX call and DOM manipulation.

Callbacks and Promises,
jQuery Callback Functions

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **$(*selector*).hide(*s* Examples**

The example below has a callback parameter that is a function that will be executed after the hide effect is completed:

Example with Callback

```
$("button").click(function(){
  $("p").hide("slow", function(){
    alert("The paragraph is now hidden");
  });
});
```

Example without Callback

```
$("button").click(function(){
  $("p").hide(1000);
  alert("The paragraph is now hidden");
});
```

jQuery Promises: Core Concepts

1. Basic Structure

jQuery introduced promises in v1.5 (2011) to handle asynchronous operations. The system consists of:

- **Deferred Object**: The controller that creates promises

  ```
  javascript
  Copy
  let deferred = $.Deferred();
  ```

- **Promise**: The read-only version of Deferred

  ```
  javascript
  Copy
  let promise = deferred.promise();
  ```

2. Three States

1. **Pending**: Initial state (not fulfilled/rejected)
2. **Resolved**: Operation completed successfully
3. **Rejected**: Operation failed

3. Core Methods

Method	Description
`.done()`	Handles successful resolution
`.fail()`	Handles rejection
`.always()`	Executes regardless of outcome
`.then()`	Chainable handler for both outcomes
`.when()`	Handle multiple promises

4. Common Patterns

Basic AJAX Example

```
javascript
Copy
$.ajax({
  url: 'api/data',
  method: 'GET'
})
.done(function(response) {
  // Handle success
  console.log('Data received:', response);
})
.fail(function(jqXHR, textStatus) {
  // Handle error
  console.error('Error:', textStatus);
});
```

Promise Chaining

```
javascript
Copy
function getUser(id) {
  return $.ajax(`/users/${id}`);
}

getUser(123)
  .then(function(user) {
    return $.ajax(`/posts?user=${user.id}`);
  })
  .then(function(posts) {
    console.log('User posts:', posts);
  });
```

Handling Multiple Promises

```
javascript
Copy
$.when(
  $.ajax('/api/users'),
  $.ajax('/api/products')
).then(function(usersResp, productsResp) {
  const [users, products] = arguments;
  console.log('Combined data:', users[0], products[0]);
});
```

5. Key Differences from ES6 Promises

1. **Error Handling**
   ```
   javascript
   Copy
   // jQuery
   promise.fail(errorHandler);

   // ES6
   promise.catch(errorHandler);
   ```

2. **Chaining Behavior**
   - jQuery prior to v3: Each `.then()` returns new promise
   - ES6: Native chaining with proper value propagation

3. **State Mutation**
   - jQuery: Deferred objects can be resolved/rejected multiple times
   - ES6: Promises settle permanently after first resolution/rejection

6. Best Practices

1. **Always return values** in `.then()` handlers for proper chaining
2. **Use `.always()`** for cleanup operations
3. **Wrap non-promise values** when needed:
   ```
   javascript
   Copy
   $.when($.Deferred().resolve(42))
    .done(function(value) {
      console.log(value); // 42
    });
   ```
4. **Convert to ES6 Promise** when necessary:
   ```
   javascript
   Copy
   const es6Promise = Promise.resolve(jQueryPromise);
   ```

7. Version Considerations

- **jQuery 1.x-2.x**: Promise implementation has some quirks
- **jQuery 3.x+**: Better alignment with ES6 standards
  - Uncaught errors now propagate correctly
  - `.then()` returns new promise like ES6

8. When to Use

- Legacy jQuery projects
- Simple asynchronous operations
- When ES6 support isn't available

9. Limitations

- Not fully Promise/A+ compliant
- No native `async/await` support

- Less intuitive error propagation compared to ES6

For modern development, consider using native JavaScript Promises or `async/await` syntax. However, understanding jQuery promises remains valuable for maintaining legacy codebases.

1. Components of Back-End Development
    - Server-Side Programming
    - Databases
    - APIs
    - Authentication
    - Security
    - Caching
    - Deployment/CI-CD

Check for any missed components. Maybe cloud services, but that's part of deployment. Microservices? Maybe under architecture. But keep it to main components.

Ensure each section is comprehensive but not overly technical. Balance depth with readability. Provide code examples for jQuery and perhaps a system diagram for back-end components.

Finally, conclude by linking the two areas, explaining how front-end (like jQuery) interacts with back-end through APIs, AJAX, etc. Emphasize the importance of both in web development.

Here's a comprehensive explanation covering both jQuery and back-end development components:

jQuery Deep Dive

Core Architecture

jQuery (v3.6.0+ in 2025) remains relevant for legacy systems despite modern frameworks. Key components:

1. **Selector Engine**
   Uses Sizzle.js under the hood:
   ```
   javascript
   Copy
   // Complex selector example
   $('div.container > input[type="text"]:visible')
   ```

2. **DOM Manipulation**
   Chainable methods for efficient operations:
   ```
   javascript
   Copy
   $('#list')
    .find('li')
    .addClass('item')
    .css('color', '#333')
    .fadeIn(300);
   ```

3. **Event System**
   Three-layer architecture:
   - Native DOM events
   - jQuery synthetic events
   - Custom event namespaces

   ```
   javascript
   Copy
   $('.btn').on('click.custom', handler);
   ```

4. **AJAX Layer**
   Unified interface for XHR:
   ```
   javascript
   Copy
   $.ajax({
     url: '/api',
     method: 'PUT',
     contentType: 'application/json',
     data: JSON.stringify(payload)
   });
   ```

5. **Effects Queue**
   Built-in animation scheduler:
   ```
   javascript
   Copy
   $('.box')
    .animate({width: '200px'}, 500)
    .delay(300)
    .fadeOut();
   ```

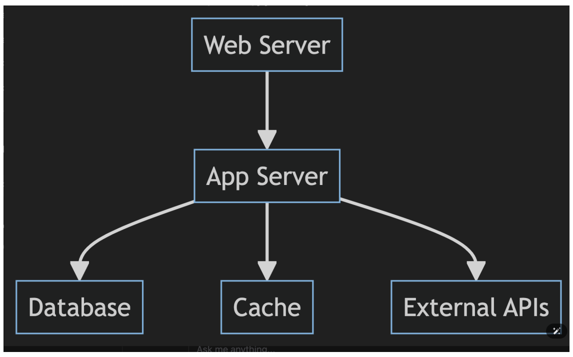
Modern Alternatives Comparison

Feature	jQuery	Modern Equivalent
DOM Manipulation	`.html()`	React/Vue templating
AJAX	`$.ajax()`	`fetch()` + Async/Await
Animation	`.animate()`	CSS Transitions/Web Animations
Event Handling	`.on()`	Native `addEventListener`

Back-End Development Components

Core Architecture Layers

1. Server-Side Runtime



2. Essential Components

1. Application Layer

- REST/GraphQL endpoints
- Session management
- Business logic
- Example Node.js structure:

```
javascript
Copy
app.post('/users',
  authMiddleware,
  validation(schema),
  async (req, res) => {
    // Business logic
    const user = await User.create(req.body);
    res.json(user);
  }
);
```

2. Data Layer

Database Type	Use Case	2025 Trends
SQL	Transactions, Reports	CockroachDB, Planetscale
NoSQL	Real-time apps	MongoDB Atlas, FaunaDB
VectorDB	AI applications	Pinecone, Weaviate

3. Service Integration

- Payment gateways (Stripe, Braintree)
- Auth providers (Auth0, Firebase Auth)
- Cloud services (AWS Lambda, Cloudflare Workers)

4. Security Matrix

```
json
Copy
{
  "authentication": "JWT/OAuth2.1",
  "authorization": "RBAC/ABAC",
  "encryption": "AES-256 + TLS 1.3",
  "protection": "CSP, CSRF tokens",
  "monitoring": "SIEM + WAF"
}
```

5. Performance Stack

- Caching: Redis/Memcached
- Queueing: RabbitMQ/Kafka
- Search: Elasticsearch/OpenSearch
- Real-time: WebSockets/Server-Sent Events

Modern Development Workflow

1. **Infrastructure as Code**
   Terraform/CloudFormation templates
2. **CI/CD Pipeline**
3. **Observability**
   - Metrics (Prometheus)
   - Logging (ELK Stack)
   - Tracing (Jaeger)

jQuery-Backend Interaction

Typical data flow:

1. jQuery makes AJAX request
2. Backend validates + processes
3. Response in JSON/XML

```
javascript
Copy
// Frontend
$.ajax({
  url: '/api/orders',
  method: 'POST',
  data: orderData
}).done(response => {
  updateUI(response);
});

// Backend (Node.js example)
router.post('/orders', (req, res) => {
  const validation = validateOrder(req.body);
  if (!validation.valid) {
    return res.status(400).json({ error: validation.errors });
  }
  database.saveOrder(req.body);
  res.json({ success: true });
});
```

Key Considerations in 2025:

- Web Components interoperability
- Progressive Enhancement strategies
- Legacy browser support costs
- Serverless backends with jQuery frontends

For modern greenfield projects, consider using:

- Frontend: React/Vue with TypeScript
- Backend: Node.js (NestJS) / Go / Rust
- Full-stack: Next.js/Nuxt.js frameworks

Model-View-Controller (MVC) Framework: Architecture & Implementation

1. Core Components

1.1 Model (Data Layer)

- **Responsibilities**:
  - Business logic implementation
  - Data validation rules
  - Database interactions
  - State management

```
csharp
Copy
// ASP.NET Core Example
public class Product
{
    [Key]
    public int Id { get; set; }

    [Required]
    [StringLength(100)]
    public string Name { get; set; }

    [Range(0, 10000)]
    public decimal Price { get; set; }
}
```

1.2 View (Presentation Layer)

- **Modern Features**:
  - Component-based templates
  - Reactive data binding
  - SSR/SSG capabilities
  - Accessibility compliance

```
razor
Copy
<!-- ASP.NET Razor View -->
@model IEnumerable<Product>
<div class="product-grid">
  @foreach (var item in Model)
  {
    <div class="product-card">
      <h3>@item.Name</h3>
      <p>@item.Price.ToString("C")</p>
    </div>
  }
</div>
```

1.3 Controller (Logic Layer)

- **Key Functions**:
  - Route handling
  - Request processing
  - Response generation
  - Middleware integration

```
java
Copy
```

```
// Spring Boot Example
@RestController
@RequestMapping("/api/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping
    public ResponseEntity<List<Product>> getAllProducts() {
        return ResponseEntity.ok(productService.getAll());
    }
}
```

2. Workflow Patterns

1. **Request Lifecycle**:

```
arduino
Copy
Client → Routing → Controller → Model ↔ Database
                       ↓
                 View ← ViewModel
```

2. **Data Flow**:

```
css
Copy
HTTP Request → Controller Action → Model Processing →
View Rendering → HTML/JSON Response
```

3. Modern Implementation Practices

3.1 Cross-Layer Communication

Pattern	Use Case	Implementation
Observer Pattern	Real-time updates	SignalR/WebSocket
Mediator Pattern	Complex interactions	CQRS Implementation
Dependency Injection	Loose coupling	.NET Core DI/Spring Beans

3.2 Security Considerations

- **Input Validation**

  ```
  python
  Copy
  # Django Example
  class ProductForm(forms.ModelForm):
      price = forms.DecimalField(
          validators=[MinValueValidator(0),
                      MaxValueValidator(100000)]
      )
  ```

- **Anti-Patterns**:
  - Business logic in views
  - Direct database access from controllers
  - Fat controllers with >500 LOC

3.3 Performance Optimization

1. **Caching Strategies**:
    - View-level caching
    - Query result caching
    - CDN integration
2. **Database Optimization**:
    - Eager vs lazy loading
    - Index optimization
    - Connection pooling

4. Testing Methodology

Testing Tools Matrix:

Layer	Tools (2025)
Unit	xUnit, Jest, pytest
Integration	Testcontainers, WireMock
E2E	Cypress, Playwright

5. Modern Framework Landscape

Popular Choices:

1. **.NET Core MVC**
    - Blazor integration
    - Minimal API support
    - Cloud-native features
2. **Spring Boot MVC**
    - Reactive stack support
    - GraalVM compatibility
    - Kubernetes-native
3. **Ruby on Rails 8**
    - Hotwire integration
    - Jbuilder-free API mode
    - Multi-database support

6. Evolution Trends (2025)

1. **AI-Assisted**
    - Automatic CRUD generation
    - Smart route optimization

  - Code vulnerability detection

2. **Serverless Integration**
   yaml
   Copy
   ```
   # serverless.yml
   functions:
     productsAPI:
       handler: com.example.ProductsController
       events:
         - httpApi:
             path: /products
             method: get
   ```

3. **Multi-Runtime Support**
   - WebAssembly modules
   - Edge computing deployment
   - Hybrid SSR/CSR rendering

7. Migration Strategies

Legacy System Modernization:

1. Strangler Fig Pattern
2. Modular Monolith → Microfrontends
3. Progressive Web Components Integration

Common Challenges:

- Session state management
- Vertical slice ownership
- Distributed tracing

For new projects, consider starting with vertical slice architecture (VSA) within MVC frameworks to balance structure and flexibility.