

JavaScript

Hướng dẫn học qua ví dụ

PHIÊN BẢN LẦN 2

TỦ SÁCH
BẢN QUYỀN
FPT Polytechnic

Steve Suehring



File thực hành trực tuyến

JavaScript

Step by Step



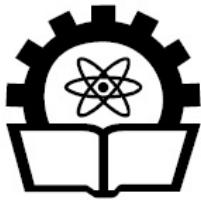
NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

Microsoft®

JavaScript

Hướng dẫn học qua ví dụ

JavaScript Step by Step



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

FPT Fpt University
TRƯỜNG ĐẠI HỌC FPT

Dịch thuật

Trường Đại Học FPT





Tựa: JavaScript - Hướng dẫn học qua ví dụ, phiên bản lần 2.

Tác giả: Steve Suehring

Dịch thuật: Trường Đại học FPT

ISBN – 13 (Bản dịch): 978-604-67-0103-3

Phiên bản gốc lần 2. Bản quyền bản gốc © 2010 thuộc về Steve Suehring.

Phiên bản dịch lần 1. Bản quyền bản dịch tiếng Việt © 2013-2016 thuộc về Đại học FPT.



Toàn bộ bản quyền liên quan tới xuất bản phẩm này đã được đăng ký bảo hộ. Không phần nào trong xuất bản phẩm này được phép sao chép hay phát hành dưới bất kỳ hình thức hay phương tiện nào, hoặc được lưu giữ trong cơ sở dữ liệu hay hệ thống truy cập, mà không có sự cho phép trước bằng văn bản của Trường Đại học FPT.

Original Title: JavaScript Step by Step, Second Edition

Author: Steve Suehring

Publisher: Microsoft Press.

ISBN – 13 (Original edition): 978-0-7356-4552-3

© 2013 FPT University

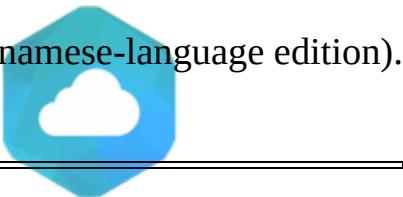
Authorized Vietnamese translation of the English edition of JavaScript Step by Step, Second Edition ISBN 978-0-7356-4552-3 © 2010 Steve Suehring

This Translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. This translation published under license. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Vietnamese-language edition copyright © 2013-2016 by FPT University.
All rights reserved.

ISBN: 978-604-67-0103-3 (Vietnamese-language edition).



Liên hệ hợp tác về nội dung bản dịch tiếng Việt và phân phối:

Phòng Bản quyền và Xuất bản, Trường Đại học FPT

Tầng 1, Tòa nhà FPT Polytechnic, Đường Hàm Nghi, KĐT Mỹ Đình I, Từ Liêm, Hà Nội.

Điện thoại: (04) 7 305 9886 - (04) 7 308 0898.

Email: caodang@fpt.edu.vn.

Dành tặng Chris
-Steve Suehring



Mục lục

Phần I JavaScript là gì, dùng ở đâu, tại sao dùng và sử dụng như thế nào?

[1 Hiểu hơn về JavaScript](#)

[2 Lập trình với JavaScript](#)

[3 Cú pháp và câu lệnh trong JavaScript](#)

[4 Làm việc với biến và kiểu dữ liệu](#)

[5 Sử dụng toán tử và biểu thức](#)



Phần II Áp dụng JavaScript

[6 Điều khiển luồng với câu lệnh điều kiện và vòng lặp](#)

[7 Làm việc với hàm](#)

[8 Các đối tượng trong JavaScript](#)

[9 Mô hình đối tượng trình duyệt](#)

Phần III Tích hợp JavaScript vào thiết kế trang

[10 Mô hình đối tượng tài liệu](#)

[11 Các sự kiện trong JavaScript và làm việc với trình duyệt](#)

[12 Tạo và sử dụng cookie](#)

[13 Làm việc với hình ảnh trong JavaScript](#)

[14 Sử dụng JavaScript với Web Form](#)

[15 JavaScript và CSS](#)

[16 Xử lý lỗi trong JavaScript](#)

Phần IV AJAX và tích hợp phía server

[17 JavaScript và XML](#)

[18 Các ứng dụng JavaScript](#)

[19 Sơ lược về AJAX](#)

[20 Tìm hiểu thêm về AJAX](#)



Phần V jQuery

[21 Giới thiệu về các thư viện và Framework của JavaScript](#)

[22 Giới thiệu về jQuery](#)

[23 Các hiệu ứng và plug-in cho jQuery](#)

Phụ lục

Lời cảm ơn

Mỗi khi viết một cuốn sách, tôi thường vấp phải khó khăn khi viết lời cảm ơn dành cho những người đã giúp tôi hoàn thành tác phẩm của mình. Tôi thường nghĩ mình nên cảm ơn tất cả mọi người và những người thân đáng kính của họ phòng trường hợp tôi sẽ chẳng viết thêm cuốn sách nào nữa. Nhưng đến bây giờ, tôi đã có trong tay vài cuốn sách, một số trong đó còn được tái bản đến lần thứ hai, vì vậy nhu cầu cảm ơn tất cả mọi người có vẻ bớt cấp thiết hơn. Lý do không phải vì những người tôi cần cảm ơn ít đi hay tôi đã tự mình làm tất cả - lý do thật sự khác xa thế. Nhưng lần nào cũng vậy, tôi luôn quên gửi lời cảm ơn một ai đó, và mặc dù họ chẳng lấy thế làm buồn lòng, tôi vẫn cảm thấy áy náy.

Và vì vậy, khi ngồi viết những dòng cảm ơn này, tôi vẫn muốn gửi lời cảm ơn của mình tới một số người. Như mọi lần, chẳng có một thứ bậc nào cụ thể và danh sách này có thể không đầy đủ. Nhưng trước hết, rõ ràng là tôi phải gửi lời cảm ơn tới Rebecca và Jakob cùng toàn thể gia đình tôi, những người đã giúp đỡ tôi trong những ngày tôi phải vật lộn từ 16 đến 20 tiếng đồng hồ bên các trang bản thảo, cố gắng hoàn thành nó nhanh nhất có thể. Tôi cũng xin cảm ơn Russell Jones, biên tập viên của nhà xuất bản O'Reilly vì sự động viên và góp ý của anh trong suốt quá trình hoàn thiện bản thảo, cũng như cảm ơn Neil Salkind của Studio B. Xin chân thành cảm ơn Chris Tuescher, John Hein, Jeremy Guthrie và Jim Leu, Andy BerBerkvam, Dan Noah, Justin Hoerter và Mark Little. Tất cả họ đều nói với tôi rằng mỗi khi tôi gửi lời cảm ơn họ trong cuốn sách của mình, họ đều bị thôi thúc phải mua ngay một cuốn về cho mình (nếu ai cũng có ý nghĩ như vậy, tôi nghĩ có lẽ mình nên chạy đi kiểm ngay một cuốn danh bạ điện thoại). Mặc dù tôi phải đi tìm người để nói lời cảm ơn, nhưng có những người tôi không thể không nói lời cảm ơn, đó là Jason, Kelly và Jeff cũng như đội làm web và tất cả các đồng nghiệp của tôi.

Tôi xin gửi lời cảm ơn tới anh trai Bob của tôi vì đã giúp tôi chọn nhạc để viết. Tôi cũng xin gửi lời cảm ơn tới Jim Oliva và John Eckendorf. Tôi không chỉ nghe dài khi viết vào một sáng thứ bảy, những câu chuyện của họ trên đài quá thật khiến các buổi sáng thứ bảy bù đầu vào bản thảo của tôi đỡ căng thẳng hơn rất nhiều. Xin cảm ơn Tim và Rob ở Partners, Pat Dunn và Dave Marie. Xin cảm ơn Jeff Currier vì anh đã lắp cửa cho phòng làm việc của tôi.

Tôi cũng xin gửi lời cảm ơn tới những độc giả đã gửi phản hồi cho phiên bản đầu tiên của cuốn sách. Điều này đã giúp tôi rất nhiều trong việc định hình những điểm cần chú trọng trong ấn bản thứ hai.

Sau khi đọc những lời cảm ơn mình viết ra, tôi nhận ra lẽ ra mình nên dùng tên gọi riêng (first name) của mọi người để gửi lời tri ân. Việc này, một cách hợp lý, có thể khiến người nhận không nghĩ đó chính là mình. “Vâng, khi tôi gửi lời cảm ơn John, có nghĩa là tôi chỉ định nói đến anh chứ không phải anh John nào khác”. Tôi đoán là có thể mình cũng hứa cảm ơn một ai khác nữa nhưng tôi không thể nhớ ra, dù vậy, tôi vẫn muốn gửi lời cảm ơn của mình tới họ.



Lời giới thiệu

JavaScript - Hướng dẫn học qua ví dụ

Kể từ thời điểm tác giả viết phiên bản gốc tiếng Anh đầu tiên của cuốn sách này (năm 2007) đã có rất nhiều thay đổi xảy ra. Những đặc tả cơ bản cho JavaScript đã được điều chỉnh rất nhiều; Windows Internet Explorer 8 và 9 lần lượt được Microsoft tung ra; các framework của JavaScript đã đạt tới độ chín và hiện đang được sử dụng rộng khắp, ngoài Internet Explorer và Firefox, các trình duyệt như Safari, Chrome và các trình duyệt khác dành cho thiết bị di động ngày càng trở nên phổ biến.

JavaScript - Hướng dẫn học qua ví dụ là bản dịch của cuốn sách gốc JavaScript Step by Step phiên bản lần thứ hai, được phát triển trên nền tảng của phiên bản thứ nhất. Kiến trúc nền tảng của ngôn ngữ JavaScript gần như không đổi, nhưng ứng dụng của nó thì trở nên rộng khắp và đặc biệt phát triển mạnh chỉ trong vòng ba năm trở lại đây. Vì thế, bố cục và bìa của cuốn sách cũng gần như được giữ nguyên, chỉ có hai điểm khác biệt cần chú ý: phiên bản này chú trọng nhiều hơn vào việc xử lý sự kiện trong JavaScript, và có riêng một phần nói về các thư viện JavaScript. Cụ thể, cuốn sách tập trung vào jQuery, một thư viện giúp đơn giản hóa việc lập trình với JavaScript, đặc biệt là trong những dự án lớn.

Xuyên suốt cuốn sách, bạn sẽ thấy có nhiều phần nội dung bổ sung thêm về các tính năng mới của JavaScript. Tương tự, các ví dụ được dùng trong cuốn sách cũng được xem xét kỹ lưỡng trên nhiều trình duyệt khác nhau để phản ánh thực tế về bối cảnh chung về web hiện nay. Phản hồi của độc giả về phiên bản thứ nhất được thể hiện qua phần nội dung và là lý do chính để tác giả bổ sung thêm phần jQuery đồng thời chú trọng hơn đến phần xử lý sự kiện.

Lời giới thiệu cho phiên bản đầu của cuốn sách này vẫn thích đáng và có thể áp dụng cho phiên bản lần hai, vì vậy bạn sẽ thấy nó trong cuốn sách này.

JavaScript là một ngôn ngữ lập trình web thuận túy, bất kể mục đích của bạn là bổ sung tính tương tác cho một trang web sẵn có hay tạo một ứng dụng hoàn toàn mới. Các trang web sẽ không được như hiện nay nếu thiếu JavaScript.

JavaScript là một ngôn ngữ dựa trên các chuẩn với đặc tả chính xác; tuy nhiên bất kỳ lập trình viên web nào cũng biết rằng gần như mỗi trình duyệt web lại có một kiểu thông dịch riêng đối với các đặc tả. May mắn là, hầu hết các trình duyệt web đều thống nhất các tính năng hỗ trợ và thông dịch các hàm cơ bản của JavaScript.

Cuốn sách này cung cấp một cái nhìn sơ lược về JavaScript, bao gồm một số hàm cơ bản cũng như các tính năng và khái niệm mới nhất, như AJAX (Asynchronous JavaScript and XML). Người dùng ngày nay duyệt web trên nhiều nền tảng và bằng nhiều trình duyệt khác nhau. Tác giả đã chú trọng vào thực tế này khi phát triển nội dung cuốn sách, vì vậy bạn sẽ thấy các ảnh màn hình thể hiện nhiều trình duyệt khác nhau và việc lập trình JavaScript dựa trên các chuẩn được chú trọng hơn là lập trình mang tính cá nhân.

Phần đầu của cuốn sách sẽ giúp bạn tìm hiểu về JavaScript và bắt đầu học cách phát triển các ứng dụng JavaScript. Bạn không cần bất kỳ một công cụ cụ thể nào khi lập trình JavaScript, vì vậy bạn sẽ học cách tạo các file JavaScript trong Microsoft Visual Studio, trong Eclipse và thậm chí trong cả Notepad (hay bất kỳ chương trình soạn thảo văn bản nào). Tiếp theo, cuốn sách sẽ trình bày phần ngôn ngữ cốt lõi và các hàm cơ bản trong JavaScript; sau đó, bạn sẽ tìm hiểu mối quan hệ giữa JavaScript và trình duyệt web. Cuối cùng, bạn sẽ được thấy những ví dụ minh họa về AJAX và tìm hiểu cách xây dựng các trang tìm kiếm động.

Phần cuối của cuốn sách sẽ tập trung vào các framework và thư viện JavaScript, cụ thể là jQuery và jQuery UI.

Đối tượng nào nên đọc cuốn sách này?

Đối tượng độc giả của cuốn sách này là những lập trình viên bắt đầu học lập trình JavaScript – những người muốn tìm hiểu các điểm cơ bản trong lập trình JavaScript hiện đại, cú pháp của ngôn ngữ này, nguyên lý hoạt động của nó trong các trình duyệt, những vấn đề thường gặp khi sử dụng các trình duyệt khác nhau và cách tận dụng AJAX và các thư viện bên thứ ba như jQuery để tăng tính tương tác cho trang web.

Đặc điểm và Ước của cuốn sách

Cuốn sách này sẽ từng bước hướng dẫn bạn lập trình với ngôn ngữ JavaScript.

Để thu được lợi ích tối đa và hiểu rõ ngôn ngữ lập trình JavaScript, bạn nên bắt đầu từ đầu cuốn sách và theo sát từng ví dụ cũng như các bài tập.

Nếu bạn đã quen với JavaScript, bạn thường có ý định bỏ qua chương đầu tiên. Tuy nhiên, Chương 1 “Hiểu hơn về JavaScript” sẽ trình bày chi tiết về lịch sử của JavaScript cũng như tiền đề cơ bản của cuốn sách này và cả hai điều này đều rất hữu ích trong việc định hình nội dung phần còn lại của cuốn sách. Chương 2 “Lập trình với JavaScript” sẽ chỉ cho bạn cách bắt đầu lập trình với JavaScript. Nếu có nhiều kinh nghiệm lập trình web, có thể bạn đã có sẵn một chương trình phát triển web, vì vậy có thể bạn muốn bỏ qua Chương 2. Tuy vậy, bạn nên làm quen với những chương trình phát triển web được giới thiệu thông qua các ví dụ lập trình JavaScript trong Chương 2.

Cuốn sách cũng có mục lục chi tiết, giúp bạn nhanh chóng tìm được các phần, mục trong các chương. Trong mỗi chương đều có danh sách chi tiết những vấn đề được thảo luận. Ngoài ra, bạn có thể tải về và sử dụng mã nguồn của các ví dụ được dùng xuyên suốt cuốn sách này.

| Quy ước | Ý nghĩa |
|--------------------------------|---|
| Ví dụ | Các bài tập hướng dẫn thực hành từng bước được giới thiệu dưới dạng một danh sách chi tiết với các bước bắt đầu từ 1. |
| Thông tin bổ sung | Phần này sẽ cung cấp cho bạn các nguồn thông tin khác về một đề tài cụ thể. |
| Mách nhỏ/ Chú ý/ Quan trọng | Những phần mách nhỏ và chú ý sẽ bổ sung thêm thông tin có thể hữu ích trong một chủ đề cụ thể nào đó. |
| Mã xuất hiện trong nội dung | Mã xuất hiện trong đoạn văn bản – sẽ được in nghiêng. |
| Khối mã | Khối mã được trình bày bằng font chữ khác để làm nổi bật đoạn mã. |

Tài nguyên đi kèm có gì?

Phần Tài nguyên đi kèm cung cấp tất cả những đoạn mã nguồn quan trọng của các ví dụ và các bài tập trong cuốn sách. Tài nguyên tải về bao gồm các dự án và các file được sắp xếp theo từng chương – mỗi chương có một thư mục riêng. Mỗi thư mục có những bài tập thực hành từng bước cho chương đó.

Do JavaScript thường phụ thuộc vào trang web liên quan, nên đoạn mã nguồn của các bài tập thực hành thường bị chia nhỏ trong các thư mục. Điều này cho phép bạn thực hiện sao chép và dán phần lớn mã HTML lặp và tập trung vào việc nhập mã JavaScript vào ví dụ.

Thư mục của mỗi chương đều có thư mục con CompletedCode, chứa ví dụ hoàn chỉnh. Bạn có thể mở các file trong thư mục CompletedCode để xem các ví dụ như đã in trong các chương.

Tải Tài nguyên đi kèm

Hầu hết các chương của cuốn sách này đều có phần bài tập cho phép bạn thực hành ngay những nội dung mới học. Bạn có thể tải về tất cả các dự án và các file trong mục Tài nguyên kèm sách tại trang <http://www.poly.edu.vn/tai-nguyen/JavaScript-Huong-dan-Hoc-qua-vi-du.html>.

Ngoài ra, bạn có thể tải Tài nguyên đi kèm về từ trang web của Nhà phát hành ấn bản gốc O'Reilly Media tại: <http://oreilly.com/catalog/9780735645523/>

Những yêu cầu tối thiểu về hệ thống

Mã JavaScrip chạy trên nhiều nền tảng (platform) khác nhau, trong đó có Microsoft Windows, Linux và Mac.



- **Bộ vi xử lý:** Tối thiểu là Pentium 133 MHz. (Bất kỳ máy tính nào có khả năng chạy các trình duyệt web có hỗ trợ JavaScript).
- **Bộ nhớ:** RAM 64 MB hoặc có dung lượng đủ để vận hành máy tính có khả năng chạy trình duyệt web có hỗ trợ JavaScript.
- **Ổ cứng:** Không gian ổ còn trống tối thiểu 2 MB.
- **Hệ điều hành:** Windows 98 hoặc các phiên bản sau này, đa số các phiên bản Linux và các phiên bản Mac OS X.
- **Màn hình hiển thị:** Với độ phân giải 640x480 hoặc cao hơn và độ sâu màu tối thiểu là 16-bit.
- **Phần mềm:** Bất kỳ trình duyệt web nào chạy được JavaScript. Bạn nên sử dụng trình duyệt Internet Explorer 6 hoặc phiên bản mới hơn, Mozilla Firefox 2.0 hoặc mới hơn, Opera 9 và Konqueror 3.5.2 hoặc mới hơn.

Hỗ trợ

Chúng tôi đã cố gắng hết sức để đảm bảo độ chính xác của cuốn sách và Tài

nguyên đi kèm. Nếu bạn gặp vấn đề nào đó, xin vui lòng tìm đúng nguồn hỗ trợ theo như chỉ dẫn sau đây.

Hỗ trợ về Sách và Tài nguyên đi kèm

Nếu bạn có câu hỏi và thắc mắc về nội dung của cuốn sách và Tài nguyên đi kèm, trước hết hãy truy cập trang Microsoft Press Knowledge Base, trang này sẽ cung cấp thông tin hỗ trợ cho những lỗi đã được phát hiện hoặc hiệu chỉnh của cuốn sách theo địa chỉ sau: www.microsoft.com/mspress/support/search.asp.

Nếu bạn không tìm được câu trả lời trên Knowledge Base, hãy gửi góp ý hoặc câu hỏi tới Microsoft Learning

Technical Support theo địa chỉ: msinput@microsoft.com hoặc gửi tới phòng Bản quyền và Xuất bản thuộc Đại học FPT tại địa chỉ: caodang@fpt.edu.vn.



Phần I

JavaScript là gì, dùng ở đâu, tại sao dùng và sử dụng như thế nào?

[Chương 1: Hiểu hơn về JavaScript](#)

[Chương 2: Lập trình với JavaScript](#)

[Chương 3: Cú pháp và câu lệnh trong JavaScript](#)

[Chương 4: Làm việc với biến và kiểu dữ liệu](#)

[Chương 5: Sử dụng toán tử và biểu thức](#)



Chương 1

Hiểu hơn về JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Có cái nhìn hoàn chỉnh về lịch sử phát triển của JavaScript.
- Nhận biết các thành phần của một chương trình JavaScript.
- Sử dụng giả giao thức javascript.
- Biết được các vị trí có thể đặt JavaScript trong một.
- Hiểu JavaScript có thể và không thể làm gì.
- Hiểu một số điểm thay đổi trong chuẩn JavaScript mới nhất (tính đến thời điểm phiên bản gốc tiếng Anh của cuốn sách này được xuất bản - năm 2010).

LƯỢC SỬ VỀ JavaScript

JavaScript không phải là Java. Vì thế, với cách hiểu như vậy, bạn có thể tiến tới tìm hiểu thứ gì đó to lớn và quan trọng hơn như tạo những menu thả xuống thật ấn tượng chẳng hạn. Tuy nhiên, nói một cách nghiêm túc, JavaScript là hệ thống dựng trên một chuẩn có tên gọi là ECMAScript. Bạn sẽ tìm hiểu thêm về ECMAScript trong phần sau của chương này.

JavaScript có nguồn gốc từ đâu? Có rất nhiều khả năng bạn không biết đến lịch sử đồ sộ và được truyền tụng thành giai thoại của JavaScript và cũng có nhiều khả năng bạn chẳng mấy quan tâm đến nó. Nếu đúng là như vậy, có thể bạn sẽ muốn nhảy sang chương tiếp theo và bắt đầu viết mã JavaScript ngay tắp lự. Tất nhiên, làm vậy là sai lầm bạn sẽ bỏ lỡ tất cả những thông tin tuyệt vời được nhắc đến trong chương này. Và hiểu chút ít về lịch sử của JavaScript là điều quan trọng để hiểu ngôn ngữ này hiện đang được triển khai như thế nào trong các môi trường khác nhau.



Ban đầu, JavaScript được phát triển bởi Brenda Eich, một lập trình viên của Netscape, vào khoảng thời gian 1995-1996. Thời điểm đó, ngôn ngữ này được gọi là LiveScript. Đó là cái tên hay cho một ngôn ngữ mới và câu chuyện có lẽ đã dừng lại tại đó nếu các chuyên gia marketing của công ty không tự ý quyết định theo ý họ và đặt lại tên ngôn ngữ này thành JavaScript. Với việc cố gắng ăn theo ngôn ngữ Java vốn đang rất nổi, sự nhầm lẫn được xảy ra ngay sau đó. Cái tên JavaScript bản thân nó đã tự tạo ra mối liên hệ giữa nó với ngôn ngữ Java. Điều này gây ảnh hưởng xấu đến JavaScript, bởi Java, mặc dù nổi tiếng vì được sử dụng nhiều lại không được ưa chuộng trong môi trường web client. Thời đó, Java được các lập trình viên tối thiểu sử dụng chỉ để biểu diễn dữ liệu hoặc thêm vào những cải tiến chẳng phục vụ cho mục đích nào (ví dụ: tạo những đoạn văn bản cuộn phiền phức). Người dùng phải trải nghiệm nhiều khó chịu khi duyệt các trang web có Java bởi chúng đòi hỏi họ phải cài thêm plug-in vào trình duyệt web thì mới chạy được. Việc này làm chậm quá trình duyệt web khiến khách ghé thăm cảm thấy bức bối khi gặp phải các vấn đề rắc rối liên quan đến việc truy cập trang web. Chỉ đến những năm gần đây, JavaScript mới bắt đầu thoát khỏi mối liên hệ tiêu cực này.

JavaScript không phải là ngôn ngữ được biên dịch, điểm này khiến nó có vẻ là

một ngôn ngữ thiếu sức mạnh. Tuy nhiên, các lập trình viên mới sử dụng JavaScript nhanh chóng nhận ra khả năng và sự hữu dụng của nó trong việc giả lập và tạo ra tính năng tương tác trên World Wide Web. Nhưng cho đến lúc nhận ra điều này, rất nhiều trang web đã được tạo mà chỉ dùng hình ảnh và mã HTML đơn giản, khiến các trang web thiếu sự hấp dẫn trực quan và khả năng tương tác trực tiếp với nội dung của trang.

Ban đầu, JavaScript sơ khai chú trọng vào việc kiểm tra tính hợp lệ của form ở phía máy client và xử lý hình ảnh trên trang web để phản hồi và tương tác, công việc tuy thô sơ nhưng hữu dụng với khách ghé thăm. Khi khách ghé thăm trang web điền vào form, JavaScript ngay lập tức kiểm tra tính hợp lệ của nội dung, thay vì thực hiện một hành trình đi – về – về tới server để làm việc này. Đặc biệt, trong giai đoạn trước khi băng thông rộng chiếm vị trí thống lĩnh, việc chặn các hành trình khứ hồi tới server là cách tuyệt vời giúp ứng dụng trống nhanh hơn và phản hồi tốt hơn – và đến ngày nay, nó vẫn vậy.

Đầu tiên là Internet Explorer 3.0

Với sự ra đời của Microsoft Internet Explorer 3.0 năm 1996, Microsoft đã bắt đầu hỗ trợ ngôn ngữ JavaScript, được biết đến trong IE như JScript, IE cũng hỗ trợ một ngôn ngữ kịch bản khác có tên là Microsoft Visual Basic, Scripting Edition hay VBScript. Mặc dù JavaScript và JScript tương tự nhau ở nhiều điểm, nhưng thực tế chúng được triển khai không hoàn toàn như nhau trên IE và Netscape. Vì vậy, các lập trình viên phải dùng nhiều cách khác nhau để xác định trình duyệt của người dùng sau đó cho thi hành đoạn mã tương ứng với ngôn ngữ được cài đặt trên mỗi trình duyệt. Công việc này được gọi là “Browser Detection” (dò trình duyệt) và nó sẽ được trình bày trong Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt”. Hiện nay, Browser Detection vẫn được sử dụng mặc dù nó không được mong muốn trong việc phát triển hầu hết các ứng dụng do phải phát sinh thêm mã đối phó với sự khác biệt giữa các ngôn ngữ kịch bản khác nhau.

Rồi đến ECMAScript

Giữa năm 1997, Microsoft và Netscape hợp tác với Liên minh Sản xuất Máy tính châu Âu ECMA tung ra phiên bản đầu tiên của đặc tả ngôn ngữ gọi là ECMAScript, tên gọi chính thức của nó là ECMA-262. Từ đó trở đi, tất cả các trình duyệt của Microsoft đều thực thi các phiên bản của chuẩn ECMAScript.

Các trình duyệt được ưa dùng khác như Firefox, Safari và Opera hiện cũng thực thi chuẩn này.

ECMA-262 phiên bản 3 được tung ra năm 1999. Tin tốt lành là các trình duyệt như Microsoft Internet Explorer 4.0 và Netscape 4.5 đều hỗ trợ chuẩn phiên bản 3 và kể từ đó trở đi tất cả các trình duyệt phổ biến đều hỗ trợ các phiên bản JavaScript được chuẩn hóa theo chuẩn ECMA-262 phiên bản 3. Điểm đáng tiếc là mỗi trình duyệt lại áp dụng chuẩn này theo những cách khác nhau, vì vậy vẫn đề không tương thích vẫn gây khó khăn cho người lập trình JavaScript.

Phiên bản mới nhất của ECMAScript, chuẩn hóa theo chuẩn ECMA-262, được tung ra vào cuối năm 2009 và được biết đến với cái tên ECMA-262 phiên bản 5. Phiên bản 4 bị bỏ qua vì nhiều lý do và để tránh gây nhầm lẫn với những đề xuất cho chuẩn này. ECMA-262 phiên bản 5 đã được hỗ trợ rộng rãi (khi cuốn sách này đang được viết) và có thể sẽ được hỗ trợ trong phiên bản của các trình duyệt phổ biến như Internet Explorer, Firefox, Opera và Safari khi cuốn sách này đến tay bạn.

Điều quan trọng cần chú ý là nếu bạn đang tích hợp JavaScript vào các ứng dụng web, bạn cần tính đến những điểm khác biệt giữa các phiên bản ECMA-262 cũng như các trình thông dịch JavaScript trên các trình duyệt khác nhau. Nghĩa là bạn sẽ phải viết mã với một mục đích cho nhiều môi trường khác nhau, rồi kiểm thử, kiểm thử và lại kiểm thử trên tất cả các trình duyệt cùng các nền tảng có thể gặp. Với Internet ngày nay, người dùng ít khi chấp nhận những ứng dụng được thiết kế để chỉ có thể chạy trên một loại trình duyệt.

Quan trọng Bạn phải chạy thử trang web của mình trên nhiều trình duyệt kể cả những ứng dụng web mà bạn nghĩ là sẽ chỉ được sử dụng với Internet Explorer. Ngay cả khi chắc chắn người dùng sẽ chỉ dùng ứng dụng với Internet Explorer hay bạn sẽ chỉ chấp nhận hỗ trợ khi cần thiết với IE, bạn vẫn nên tiến hành chạy thử ứng dụng đó trên các trình duyệt khác. Điều này quan trọng không chỉ vì tính bảo mật mà còn vì nó cho thấy bạn là một lập trình viên chu đáo, hiểu rõ công nghệ Internet hiện đại.

Rất nhiều tiêu chuẩn...

Có thể nói các chuẩn JavaScript được định nghĩa khá lỏng lẻo. Điều này dẫn đến mỗi trình duyệt có thể hỗ trợ JavaScript theo mỗi cách hơi khác nhau, khiến công việc của mọi lập trình viên thêm rắc rối. Việc phải cố gắng đáp ứng đầy đủ tất cả những khác biệt này còn mệt mỏi hơn là viết mã trong những ngôn ngữ được trình bày, cài đặt, triển khai thành những môi trường độc lập và duy nhất với mỗi phiên bản cụ thể ví dụ như Microsoft Visual Basic hay Perl chẳng hạn. Nhiệm vụ của bạn (và cũng là của tôi) là theo dõi những khác biệt này và chú ý khi cần, rồi cố gắng tìm ra nhiều điểm chung hết sức có thể.

Mô hình đối tượng tài liệu - DOM

Một chuẩn đang phát triển khác có liên quan tới công việc lập trình trong JavaScript là chuẩn *Mô hình đối tượng tài liệu (Document Object Model - DOM)* do Hiệp hội World Wide Web (WWW Consortium - W3C) phát triển. W3C định nghĩa DOM là “một giao diện kết nối độc lập với nền tảng và ngôn ngữ, cho phép các chương trình có thể sử dụng để truy cập động cập nhật nội dung, cấu trúc và style của văn bản”. Điều này có nghĩa là bạn có thể làm việc với tiêu chuẩn mà các trình duyệt web đều tuân thủ, nó giúp bạn xây dựng trang web động. DOM tạo ra một cấu trúc dạng cây cho các văn bản HTML, XML và cho phép các mã script truy xuất đến các đối tượng này. JavaScript cũng sử dụng DOM trong nhiều chức năng quan trọng.

Tương tự như JavaScript, DOM được thông dịch khác nhau tùy theo từng trình duyệt, điều này khiến việc lập trình JavaScript thêm phần thú vị. Internet Explorer 4.0 và các phiên bản trước đó của Netscape có hỗ trợ DOM cấp độ sơ khai, hay còn gọi là DOM cấp 0. Nếu sử dụng DOM cấp 0, người dùng chắc chắn sẽ được các trình duyệt này cũng như tất cả các trình duyệt phiên bản mới hơn hỗ trợ.

Microsoft Internet Explorer 5.0 và 5.5 có một số hỗ trợ cho DOM cấp 1, còn Windows Internet Explorer 6.0 và các phiên bản mới hơn hỗ trợ một phần của DOM cấp 2. Các phiên bản mới nhất của Firefox, Safari và Opera đều hỗ trợ DOM cấp 2. Safari cung cấp một phiên bản của bộ thông dịch webkit. Bộ thông dịch này là nền tảng cho các thiết bị như iPhone và iPad cũng như các thiết bị sử dụng hệ điều hành Android.

Khi tìm hiểu về các chuẩn cho JavaScript cũng như các chuẩn có liên quan đến DOM, có một điều mà người dùng cần ghi nhớ là phải đặc biệt chú ý đến đoạn mã mà mình viết ra và cú pháp được sử dụng để thực thi chương trình. Nếu

không, JavaScript có thể không chạy và khiến trình duyệt không thể thông dịch và hiển thị trang web theo đúng ý bạn. Chương 10 “Mô hình đối tượng tài liệu” sẽ trình bày chi tiết hơn về DOM.

Mách nhỏ W3C có một ứng dụng cho phép người dùng kiểm tra xem trình duyệt hỗ trợ DOM ở những cấp nào. Có thể tìm thấy ứng dụng này tại địa chỉ sau: <http://www.w3.org/2003/02/06-dom-support.html>.

Chương trình JavaScript gồm những gì?

Chương trình JavaScript là một tập hợp *các câu lệnh*, các câu lệnh này được tạo thành từ *các từ khóa*, *toán tử* và *chuỗi định danh* được sắp xếp theo một trật tự mà trình thông dịch Javascript có trong hầu hết các trình duyệt web có thể hiểu được. Câu lệnh có thể dài, nhưng đa phần đều không quá phức tạp kể cả với những lập trình viên sử dụng ngôn ngữ lập trình khác. Chúng có thể có dạng như sau:

```
var smallNumber = 4
```

Trong câu trên, từ khóa từ khóa *var* được đặt trước các ký hiệu khác như: *định danh* (*smallNumber*), *toán tử* (=) và *số nguyên* (4). (Bạn sẽ tìm hiểu thêm về những khái niệm này trong phần còn lại của cuốn sách). Mục đích của câu lệnh trên là thiết lập giá trị của biến *smallNumber* bằng 4.

Giống như các ngôn ngữ lập trình khác, các câu lệnh được sắp xếp theo trật tự nhất định tạo thành một chương trình thực hiện một hoặc nhiều chức năng. Nhưng Javascript có cách định nghĩa hàm theo cách riêng của nó, vấn đề này sẽ được trình bày kỹ hơn trong Chương 7 “Làm việc với hàm”. Javascript cũng định nghĩa sẵn nhiều hàm mà bạn có thể sử dụng luôn trong các chương trình của mình.

Dùng giả giao thức *javascript* và một hàm

1. Mở một trình duyệt bất kỳ như Internet Explorer hoặc Firefox hay Google Chrome.
2. Trên thanh địa chỉ, nhập dòng mã sau và nhấn phím Enter:

```
javascript:alert("Hello World");
```

Sau khi nhấn phím Enter, bạn sẽ nhìn thấy một hộp thoại tương tự như hình sau (Chú ý: Một số phiên bản Firefox không hỗ trợ tính năng này):



Chúc mừng! Vậy là bạn vừa viết thành công dòng mã JavaScript đầu tiên. Trong dòng lệnh đơn giản này có hai thành phần quan trọng thường được dùng trong lập trình JavaScript, đó là giả giao thức (pseudo-protocol) javascript có sẵn trong trình duyệt và quan trọng hơn là hàm alert. Những thành phần này sẽ được trình bày cụ thể hơn ở các chương sau. Còn ngay tại thời điểm này bạn đã được tiếp xúc một số thứ mà bạn sẽ dùng trong tương lai!

JavaScript cũng là *ngôn ngữ hướng sự kiện*, có nghĩa là nó có thể phản hồi những sự kiện nhất định hoặc “những hành động xảy ra”, như nhấn chuột, hoặc thay đổi ký tự trong một trường nhập dữ liệu. Gắn JavaScript với sự kiện là một phần trọng yếu trong lập trình JavaScript. Ở Chương 11, các bạn sẽ biết thêm về cách thức xử lý sự kiện bằng JavaScript.

Vị trí của JavaScript trong trang web

Nếu bạn mới bắt đầu tìm hiểu về HTML, điều đầu tiên bạn cần biết là HTML mô tả các thành phần trong một trang web bằng cách sử dụng các cặp thẻ nằm trong dấu ngoặc (<). Thẻ đóng sẽ bắt đầu với ký tự gạch chéo (/). Các thẻ có thể lồng vào nhau. Mã JavaScript nằm trong cặp thẻ <script> bên trong cặp thẻ <head> </head> và/hoặc cặp thẻ <body> </body> của một trang web, như ở ví dụ

dưới đây:
dưới đây:

```
<html>
<head>
<title>Tiêu đề trang web</title>
<script type="text/javascript">
// đoạn mã JavaScript được đặt ở đây
</script>
<body>
<script type="text/javascript">
// đoạn mã JavaScript được đặt ở đây
</script>
</body>
</html>
```

Trình duyệt sẽ tự động thực thi các đoạn mã JavaScript trong thẻ `<body>`, điều này sẽ rất hữu ích cho bạn khi bạn cần ghi nội dung trực tiếp lên các trang web bằng cách sử dụng hàm JavaScript như sau (các lời gọi hàm được in đậm):

```
<head>
<title>Tiêu đề trang web</title>
<script type="text/javascript">
// đoạn mã JavaScript được đặt ở đây
</script>
</head>
<body>
<script type="text/javascript">
document.write("Hello");
document.write("world");
</script>
</body>
</html>
```

Do cách trình duyệt gọi đến các mã JavaScript, khi đặt mã JavaScript vào trang HTML, tốt nhất là bạn nên đặt thẻ `<script>` ở phần cuối của phần `<body>` chứ không phải ở phần `<head>`. Điều này giúp đảm bảo nội dung của trang web được tải đầy đủ phòng trường hợp trình duyệt ngừng tải trang web trong khi các đoạn mã JavaScript vẫn được tải.

Khi sử dụng JavaScript trong một trang ngôn ngữ đánh dấu siêu văn bản mở rộng (XHTML), ký hiệu (<) và (&) sẽ được thông dịch thành mã XML, dẫn đến một số trực trặc cho JavaScript. Để vượt qua rào cản này, hãy nhớ sử dụng cú

pháp dưới đây trong trang XHTML.

```
<script type="text/javascript">  
![CDATA[  
// đoạn mã JavaScript được đặt ở đây  
]]>  
</script>
```

Những trình duyệt không tương thích với XHTML không thể thông dịch các thành phần CDATA một cách chính xác. Có một cách để giải quyết vấn đề trên đó là đặt phần CDATA trong chú thích JavaScript – chú thích là một hoặc nhiều dòng văn bản bắt đầu bằng hai dấu gạch chéo (//) như bên dưới:

```
<script type="text/javascript">  
//![CDATA[  
// đoạn mã JavaScript được đặt ở đây  
//]]>  
</script>
```

Chúng ta có thể dễ dàng nhận ra đoạn mã trên chưa thực sự hoàn hảo hay nói cách khác là rất xấu. Vậy nên trong [Chương 2](#) “Lập trình với JavaScript”, chúng ta sẽ đề cập chi tiết phương pháp xử lý vấn đề này.

Document Type (kiểu tài liệu)

Nếu đã từng lập trình web, chắc chắn bạn không hề xa lạ với việc khai báo Document Type (kiểu tài liệu), hay thường được gọi là khai báo DOCTYPE. Một trong những nhiệm vụ quan trọng nhất khi thiết kế web là đảm bảo khai báo DOCTYPE chính xác và đúng cú pháp ở phần đầu của trang. Việc khai báo DOCTYPE hay thường được viết tắt là DTD giúp các trình duyệt (hoặc các chương trình phân tích cú pháp) nhận biết những quy tắc phải theo khi phân tích các phần tử trong trang web.

Dưới đây là ví dụ về cách khai báo DOCTYPE cho HTML 4.01:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"  
      "http://www.w3.org/TR/html4/strict.dtd">
```

Nếu bạn sử dụng Microsoft Visual Studio 2005 hay phiên bản mới hơn để tạo trang web, mỗi trang sẽ được tự động chèn các đoạn mã khai báo

DOCTYPE theo chuẩn XHTML 1.0 như sau:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

HTML 5 sử dụng kiểu khai báo DOCTYPE đơn giản hơn nhiều.

```
<!DOCTYPE html>
```

Khi khai báo DOCTYPE sai, trình duyệt sẽ hiểu trang web đang theo chế độ Quirks. Chế độ Quirks trong nhiều trường hợp sẽ làm cho trang web chạy sai so với chủ đích của người tạo, đặc biệt là khi xem trang trên nhiều trình duyệt khác nhau.

Do vậy, khi khai báo DOCTYPE, chúng ta phải đảm bảo là các file HTML, CSS và JavaScript tuân thủ các chuẩn web để trang web có thể hiển thị đúng và phục vụ số đông người dùng bất kể họ sử dụng trình duyệt nào. Chúng ta sẽ bàn đến HTML và việc kiểm tra tính hợp lệ cho CSS kỹ hơn ở Chương 15 “JavaScript và CSS”. W3C cung cấp công cụ kiểm tra trực tuyến tại địa chỉ <http://validator.w3.org/> nhờ đó các lập trình viên có thể tự kiểm tra tính hợp lệ của bất kỳ website công cộng nào.

Mách nhỏ Sử dụng công cụ kiểm tra tính hợp lệ của mã đánh dấu - Markup Validator thường xuyên cho đến khi bạn quen với việc viết mã theo chuẩn và luôn kiểm tra tính hợp lệ trước khi mở cửa trang web của bạn.

Điều JavaScript có thể làm

JavaScript là ngôn ngữ bổ trợ, tức là chúng ta rất ít gặp một ứng dụng được viết hoàn toàn bằng JavaScript mà không có sự hỗ trợ của các ngôn ngữ khác như HTML và trình duyệt web. Một số sản phẩm của Adobe có hỗ trợ JavaScript nhưng JavaScript chủ yếu được sử dụng khi lập trình web.

JavaScript được thể hiện bằng chữ J trong cụm từ viết tắt AJAX (JavaScript và XML không đồng bộ), kỹ thuật được yêu thích của hiện tượng Web 2.0. Ngoài ra, JavaScript còn là một ngôn ngữ được sử dụng thường xuyên để tạo nên tính

tương tác mà người dùng web ngày nay đòi hỏi.

JavaScript có thể thực hiện nhiều tác vụ ở phía client của ứng dụng. Ví dụ, nó có thể bổ sung tính tương tác cần thiết cho một trang web bằng cách tạo các menu thả xuống, xử lý tài liệu trên trang, bổ sung các phần tử động vào trang và giúp nhập liệu trên form.

Trước khi tìm hiểu những điều JavaScript có thể làm (trọng tâm của cuốn sách này) bạn cần hiểu những điều JavaScript không thể làm nhưng lưu ý cả hai khía cạnh này trong cuốn sách đều không toàn diện.

Điều JavaScript không thể làm

Nhiều công việc mà JavaScript không thể thực hiện là do việc sử dụng JavaScript bị giới hạn trong môi trường trình duyệt web. Phần này sẽ giúp bạn tìm hiểu những tác vụ mà JavaScript không thể và không nên thực hiện.

Không thể bắt buộc JavaScript chạy trên máy client

JavaScript phụ thuộc vào một chương trình chủ để hoạt động (chứ không chạy ngay trên môi trường hệ điều hành được). Chương trình chủ này thường là các trình duyệt web trên máy client hay còn được gọi là user agent (phần mềm đại diện người dùng). Vì JavaScript là ngôn ngữ phía client, nó chạy trọn vẹn trong client nên chỉ có thể thực hiện những gì mà client cho phép thực hiện.

Hiện nay, một số người vẫn sử dụng các trình duyệt cũ không hỗ trợ JavaScript. Một số trường hợp khác lại không thể sử dụng các tính năng thú vị của JavaScript do sử dụng các chương trình hỗ trợ người khuyết tật, chương trình đọc trực tiếp văn bản hay các phần mềm add-on dạng hỗ trợ duyệt web. Và thậm chí có một số người chọn tắt JavaScript vì khó chịu với những quảng cáo pop-up hoặc vì lý do bảo mật.

Bất kể đó là lý do gì, bạn cũng cần thực hiện thêm một số việc để đảm bảo trang web bạn đang thiết kế có thể sẵn dùng với cả những người không được hỗ trợ JavaScript. Bạn có thể phản đối “Nhưng những tính năng đã có thật sự là rất tuyệt, tuyệt vời, dễ chịu, cần thiết”. Cho dù tính năng đó của trang web tuyệt vời

cỡ nào, bạn sẽ vẫn được hưởng lợi nhiều hơn khi khả năng tương tác của nó tốt thêm và dẫn tới số lượng khách thăm trang web tăng lên. Trong phần “Các mẹo sử dụng JavaScript” ngay sau đây, tôi sẽ đưa ra một số chỉ dẫn mà bạn có thể làm theo để sử dụng JavaScript hợp lý và hiệu quả trên trang web của mình.

Xem xét vấn đề này theo một hướng khác cũng có thể hữu ích. Khi bạn xây dựng ứng dụng web để chạy trên Microsoft Internet Information Services (IIS) 6.0, bạn có thể giả định rằng ứng dụng này thường hoạt động khi chạy trên một server IIS 6.0 ở bất kỳ đâu. Tương tự như vậy, khi bạn xây dựng ứng dụng cho Apache 2, bạn có thể chắc chắn là nó sẽ hoạt động trên các máy cài Apache 2 khác. Tuy nhiên, giả định này không đúng với JavaScript. Khi bạn viết một ứng dụng chạy tốt trên máy tính của mình, bạn không thể chắc nó sẽ hoạt động trên máy tính của người khác. Bạn không thể kiểm soát việc ứng dụng của bạn sẽ hoạt động thế nào khi nó được gửi tới máy client.

JavaScript không thể đảm bảo việc bảo mật dữ liệu



Vì JavaScript được vận hành hoàn toàn từ máy client, nên người lập trình phải học cách chấp nhận việc mất quyền kiểm soát chương trình, và đôi khi có thể gây ra những tác động nghiêm trọng. Khi chương trình nằm trên máy client, người dùng có thể thoải mái tác động lên dữ liệu. Cũng như với các ngôn ngữ lập trình khác, bạn không nên tin những dữ liệu được gửi trả từ phía client. Ngay cả khi đã sử dụng hàm JavaScript để kiểm tra tính hợp lệ của form, bạn vẫn phải kiểm tra lại dữ liệu đầu vào khi dữ liệu này được gửi lên server. Nếu phía client đã tắt JavaScript nó có thể gửi dữ liệu rác qua form trên trang web. Nếu bạn tin tưởng rằng hàm JavaScript phía client đã kiểm tra dữ liệu để đảm bảo tính hợp lệ, có thể bạn sẽ bắt gặp những dữ liệu không hợp lệ được gửi lại server, điều này gây ra những hệ quả nguy hiểm và khôn lường.

Quan trọng Nên nhớ JavaScript có thể bị tắt trên máy tính của người dùng. Do đó bạn không nên tin vào các kỹ thuật JavaScript của mình sẽ thành công, ví dụ như sử dụng JavaScript để vô hiệu hóa chuột phải hay ngăn người dùng xem mã nguồn của trang web. Ngoài ra, bạn cũng không nên dùng những kỹ thuật này như là các phương thức bảo mật trong ứng dụng.

JavaScript không thể chạy liên domain

Người lập trình với JavaScript cũng phải nắm được *Chính sách cùng nguồn gốc* (*Same Origin Policy*). Theo quy định của chính sách này, những đoạn mã chạy trên một domain không thể truy cập thuộc tính hay tác động đến đoạn mã của một domain khác. Chẳng hạn, JavaScript có thể được sử dụng để mở một cửa sổ trình duyệt mới, nhưng nội dung của cửa sổ đó bị hạn chế bởi đoạn mã gọi. Và tất nhiên, khi một trang trên trang web của tôi (braingia.org) có chứa mã JavaScript, nó cũng không có khả năng truy cập bất kỳ mã JavaScript nào đang được thực thi trên một domain khác chẳng hạn microsoft.com. Đây là đặc điểm cốt yếu của Chính sách cùng nguồn gốc: JavaScript phải được thực thi hoặc bắt nguồn từ cùng một vị trí.

Chính sách trên ngược lại với ngữ cảnh sử dụng frame và các đối tượng *XMLHttpRequest* của AJAX vốn cho phép gửi nhiều yêu cầu JavaScript tới các web server khác nhau. Windows Internet Explorer 8 của Microsoft có hỗ trợ đối tượng *XDomainRequest*, việc này cho phép truy cập phần nào dữ liệu của các domain khác. Các giải pháp thay thế chính sách này và một số cách sử dụng JavaScript trên nhiều domain sẽ được trình bày trong Chương 19 “Sơ lược về AJAX”. Còn bây giờ, bạn chỉ cần ý thức rằng JavaScript chỉ có thể thực hiện một số tác vụ nhất định trên cửa sổ trình duyệt web của chính bạn.

JavaScript không thực hiện ở phía server

Khi lập trình bằng các ngôn ngữ phía server như Visual Basic .NET hay PHP (PHP là từ viết tắt đê quy của *PHP: Hypertext Preprocessor*), bạn có thể chắc chắn rằng phía server sẽ được cài đặt những hàm nhất định, như truy xuất cơ sở dữ liệu hoặc trao quyền truy cập những mô-đun cần thiết cho ứng dụng web. JavaScript không thể truy cập đến bất kỳ biến nào phía server. Chẳng hạn, JavaScript không thể truy cập dữ liệu trên máy server. Mã JavaScript chỉ giới hạn trong nền tảng mà nó được chạy, nền tảng này thường là trình duyệt.

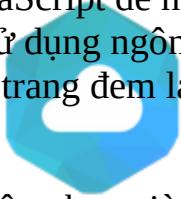
Bạn cần thêm một thay đổi trong suy nghĩ nếu đã quen lập trình phía server, đó là với JavaScript, bạn phải kiểm thử mã trên các môi trường máy client khác nhau để biết khả năng hỗ trợ của chúng. Khi bạn lập trình phía server, nếu server không cài đặt một hàm nào đó, bạn sẽ biết ngay bởi vì đoạn mã phía server sẽ lỗi

khi bạn cho chạy thử. Ngoại trừ trường hợp quản trị viên hệ thống hay tay máy, việc thực thi mã phía server không mấy khi thay đổi đột ngột, vì vậy bạn sẽ dễ dàng nhận ra mình có thể và không thể viết mã gì. Tuy vậy bạn không thể đoán hết được đoạn mã JavaScript sẽ chạy như thế nào bên phía người dùng bởi vì máy người dùng hoàn toàn nằm ngoài tầm kiểm soát của bạn.

Các mẹo sử dụng JavaScript

Có một số yếu tố có thể làm thiết kế web trở nên tốt hơn, nhưng vấn đề là ai sẽ phân xử cái gì tốt và cái gì không? Có thể có người ghé thăm trang web và cho đó là một mớ hỗn lõn màu mè như thể được tạo ra bằng cách cho mọi thứ vào một cái bao rồi xóc tung lên, nhưng cũng có thể sẽ có người thích thiết kế và cách phối màu của nó.

Vì bạn đang cầm trên tay cuốn sách này, nên tôi sẽ giả định luôn rằng bạn đang cần bổ sung kiến thức sử dụng JavaScript để nâng cấp trang web của mình. Cũng xin giả định luôn rằng bạn muốn sử dụng ngôn ngữ lập trình này để hỗ trợ khách thăm trang web cũng như làm cho trang đem lại cảm nhận tốt hơn và hoạt động tốt hơn.



Thiết kế trang web không và sẽ không bao giờ là một quá trình có mục tiêu rõ ràng. Mục tiêu của một trang web có thể là cung cấp thông tin, mục tiêu này sẽ dẫn tới phương thức thiết kế tương ứng, trong khi mục tiêu của một trang web khác có thể lại là kết nối với ứng dụng, do đó đòi hỏi phải có thiết kế và chức năng chuyên biệt. Điều này có nghĩa là nhiều trang web nổi tiếng và có vẻ được thiết kế tốt sẽ có một số điểm chung nhất định. Phần này sẽ cố gắng trình bày những điểm đó, dù vậy đề nghị bạn ghi nhớ rằng nội dung trình bày không phải là toàn bộ mọi thứ và mỗi mục cũng chỉ phản ánh quan điểm của cá nhân tác giả.

Một trang web được thiết kế tốt sẽ làm được những việc như sau:

- Chú trọng vào chức năng hơn là hình thức Khi ghé thăm một trang web, người dùng thường muốn thu thập thông tin hoặc thực hiện một công việc nào đó. Trang của bạn càng khó duyệt, khả năng người dùng chuyển sang trang khác càng lớn.
Ảnh động và những thứ nhấp nháy xuất hiện rồi biến mất, nhưng điều đọng lại là trang web có thông tin cơ bản được trình bày một cách chuyên nghiệp

và có thể truy cập dễ dàng. Sử dụng phần mềm hoạt ảnh sinh động hay công nghệ web mới nhất khiến tôi nghĩ đến thời của thẻ HTML `<bLink>`. Thẻ này khiến đoạn văn bản trong nó ẩn rồi lại hiện trên màn hình. Gần như tất cả các lập trình viên web đều ghét thẻ `<bLink>` và hiệu ứng của nó trên trang web. Nhưng cũng nên ghi nhớ rằng tính năng hấp dẫn hay hiệu ứng đặc biệt hôm nay của một trang web trong tương lai có thể sử dụng thẻ `<bLink>`. Các trang web thành công đều tuân thủ những nguyên tắc cơ bản và chỉ sử dụng những tính năng nổi bật theo yêu cầu của nội dung.

Hãy sử dụng các phần tử như site map (sơ đồ trang web), thẻ alt và những công cụ điều hướng đơn giản và đừng đặt ra yêu cầu phải sử dụng những phần mềm hay plug-in đặc biệt để xem nội dung chính của trang. Tôi rất thường gặp phải tình huống vào một trang web và bị chặn lại vì cần cài đặt plug-in hoặc phiên bản mới nhất của một chương trình nào đó để xem nó. Mặc dù site map, thẻ alt và công cụ điều hướng đơn giản có vẻ kỳ quặc nhưng chúng là những thành phần không thể thiếu để hỗ trợ truy cập web cho người khuyết tật. Chức năng tự đọc văn bản hay những công nghệ cho phép chuyển nội dung của trang web sang dạng âm thanh để hỗ trợ người khuyết tật thường dùng sử dụng những tính năng nói trên. Tuy nhiên các ứng dụng đó lại hay gặp rắc rối với những đoạn mã JavaScript phức tạp.

- Tuân theo chuẩn Các chuẩn web tồn tại là để được tuân thủ, vì vậy bỏ qua chuẩn sẽ không có lợi cho bạn. Sử dụng lệnh khai báo DOCTYPE chính xác và viết mã HTML đúng cú pháp sẽ giúp đảm bảo trang web của bạn hiển thị đúng. Bạn cũng nên sử dụng công cụ Markup Validator của W3C để kiểm tra tính hợp lệ của mã nguồn. Nếu trang của bạn bị vỡ, không hợp chuẩn, cần sửa lại ngay!
- Hiển thị thông tin chính xác trên nhiều trình duyệt Ngay cả khi Internet Explorer là trình duyệt được sử dụng nhiều nhất, lập trình viên cũng không nên bỏ qua các trình duyệt khác. Làm vậy có nghĩa là khả năng truy cập trang đã bị lập trình viên bỏ qua, và những người sử dụng các chức năng tự đọc văn bản hoặc các add-on khác sẽ không thể truy cập trang web. Những người không sử dụng hệ điều hành Microsoft Windows cũng có thể gặp rủi ro khi vào các trang web này.

Mặc dù Internet Explorer là trình duyệt được nhiều người sử dụng nhất, nhưng vẫn có khả năng rất lớn là có ít nhất ba đến bốn trong số mười khách ghé thăm trang web sử dụng trình duyệt khác. Tất nhiên, người xem càng có chuyên môn kỹ thuật bao nhiêu, bạn càng cần điều chỉnh để thích nghi với trình duyệt khác ngoài trình duyệt Internet Explorer bấy nhiêu. Vì vậy, nếu trang web của bạn dành cho dân kỹ thuật, có thể bạn sẽ cần lập trình sao cho

nó có thể chạy trên Firefox, Safari hoặc thậm chí Lynx (trình duyệt web trên nền text của Linux)

Bất kể trang web của bạn có nội dung gì, bạn sẽ không bao giờ muốn làm khách ghé thăm phải quay lưng bỏ đi vì lý do trình duyệt. Hãy tưởng tượng một người bán hàng quái đản cứ mỉa mai khách hàng tiềm năng thì lại từ chối ba người vì họ đi giày không vừa mắt. Cửa hàng đó chắc chắn sẽ chẳng kinh doanh được lâu – hay ít ra là nó sẽ không thành công như đáng lẽ ra phải thế.

Nếu trang web của bạn tuân thủ các chuẩn web, nhiều khả năng là bạn đã thực hiện hầu hết những gì cần làm để hỗ trợ việc chạy trang web trên nhiều trình duyệt. Tránh sử dụng các plug-in cá nhân cũng là một cách để đảm bảo trang web được thông dịch chính xác. Bạn chỉ cần nhìn vào chiếc iPad của Apple là sẽ thấy ngay ví dụ về một thiết bị thông dụng nhưng bị giới hạn chức năng sử dụng vì không hỗ trợ Flash hay Java. Vì lý do này, việc tạo các trang đáp ứng chuẩn và tránh sử dụng những plug-in cá nhân đảm bảo cho trang web của bạn sẵn dùng với số

- Sử dụng công nghệ phù hợp vào thời điểm phù hợp Nói về plug-in, một trang web được thiết kế tốt thường không lạm dụng hay sử dụng sai công nghệ. Trên một trang video, việc chạy các video là hoàn toàn phù hợp. Tương tự như vậy, trên một trang nhạc, việc chơi nhạc nền cũng là điều phù hợp. Tuy nhiên, những tính năng này có thể không phù hợp khi được sử dụng trên một số trang web khác. Nếu bạn cảm thấy trang của mình cần lồng nhạc nền, hãy quay trở lại bảng kế hoạch và kiểm tra lại lý do đầu tiên bạn muốn khởi tạo trang! Tôi vẫn rùng mình sợ hãi khi nghĩ đến lần ghé thăm trang web của một luật sư. Khi tôi vừa vào trang, tiếng leng keng nổi lên dù tôi không có bất kỳ hành động can thiệp nào. Không ai để bạn bè mình sử dụng nhạc nền trên website trừ khi những người bạn đó là ban nhạc Rush (một ban nhạc Rock nổi tiếng ở Canada) và bạn đang xây dựng trang web cho họ.

JavaScript phù hợp với trình duyệt nào?

Thế giới web ngày nay vẫn đang thay đổi không ngừng. Một trong những trào lưu được ưa chuộng hơn cả trong năm vừa qua là *viết mã hiệu quả*. Khái niệm này là một phần của trào lưu lớn hơn được biết đến với tên gọi *phân tách hành vi*. *Phân tách hành vi* đòi hỏi tách cấu trúc, style (kiểu trình bày) và hành vi của mỗi trang web ra khỏi nhau. Trong mô hình này, HTML hay XHTML sẽ cung cấp cấu trúc, CSS cung cấp style và JavaScript cung cấp hành vi. Viết mã

JavaScript theo kiểu hiệu quả tức là không can thiệp các thành phần khác là cấu trúc và style. Nếu trình duyệt không hỗ trợ JavaScript, trang web vẫn hoạt động bởi người dùng có thể sử dụng trang theo cách khác.

Khi được áp dụng đúng cách, viết mã hiệu quả cho phép giả định nếu JavaScript không được hỗ trợ phía client nhưng trang web khi bị lỗi JavaScript vẫn có thể hoạt động ở mức độ chấp nhận được (cung cấp được nội dung). Điều này có nghĩa là trang web vẫn hoạt động mà không cần JavaScript và có các phương thức phù hợp để làm cho JavaScript sẵn dùng khi được yêu cầu. Chương 11 sẽ trình bày một trong những phương thức như vậy.

Tôi ủng hộ kiểu viết mã hiệu quả vì điều đó có nghĩa là trang web tuân thủ các chuẩn và đảm bảo bốn yếu tố mà tôi đã chia sẻ trong phần trước. Đáng tiếc là điều này không phải lúc nào cũng đúng. Bạn có thể tách riêng HTML, CSS và JavaScript nhưng cuối cùng vẫn phải sử dụng các thẻ riêng. Tuy nhiên, khi lập trình để đảm bảo tính tiện lợi cho trang của mình, bạn thường chú ý hơn đến các chi tiết và quan tâm hơn đến kết quả cuối cùng phù hợp với chuẩn.

Xuyên suốt cuốn sách này, tôi cố gắng trình bày những điểm cơ bản về JavaScript và cách sử dụng JavaScript sao cho hiệu quả và tiện lợi nhất.

Một lưu ý về JScript, JavaScript và cuốn sách này

Cuốn sách này bàn về JavaScript theo chuẩn ECMA ở tất cả các phiên bản, từ đầu tới phiên bản 5. Ở một số phần, tôi sẽ chú trọng vào thông tin có liên quan đến JScript và JScript.NET. Bạn có thể tham khảo thêm về JScript tại địa chỉ:

JScript (Windows Script Technologies) <http://msdn.microsoft.com/en-us/library/hbxc2t98.aspx>

Những trình duyệt nên được hỗ trợ?

Tính tương thích ngược từ lâu đã là vấn đề đau đầu đối với người lập trình web. Việc lựa chọn phiên bản trình duyệt nào để hỗ trợ trở thành sự thỏa hiệp giữa việc sử dụng tính năng mới nhất có trong trình duyệt mới nhất và tính năng tương thích mà các trình duyệt cũ hơn đòi hỏi. Không có quy tắc ngắn gọn và cứng nhắc nào về trình duyệt mà bạn nên hỗ trợ, vì vậy câu trả lời là: còn tùy.

Quyết định phụ thuộc vào những gì bạn muốn làm với trang của mình và việc

bạn đánh giá cái nào cao hơn: Những khách sử dụng phần mềm và phần cứng đời cũ ghé thăm trang web hay tính năng bổ sung có trên phiên bản mới của trình duyệt. Một số trình duyệt quá cổ lỗ không thể hỗ trợ bởi chúng không thể thông dịch chính xác CSS cũng như JavaScript. Chìa khóa của việc hỗ trợ đa trình duyệt là chạy thử trang web trên các trình duyệt đó.

Nếu có tài khoản MSDN của Microsoft, bạn có thể tải về những sản phẩm cũ của hãng, bao gồm cả các phiên bản đầu của Internet Explorer và bạn sẽ thấy trang web hoạt động như thế nào khi chạy trên Internet Explorer 4.0. Tài nguyên bổ sung là Application Compatibility Virtual PC Images, bạn có thể tải về miễn phí từ trang Microsoft. Những tài nguyên này cho phép bạn sử dụng các phiên bản Microsoft Windows giới hạn thời gian và chứa các trình duyệt Internet Explorer 6.0 và Windows Internet Explorer 7. Bạn đọc có thể tìm hiểu chi tiết tại: <http://www.microsoft.com/downloads/details.aspx?FamilyId=21EABB90-958F-4B64-B5F1-73D0A413C8EF&displaylang=en>.

Nhiều thiết kế web và hàm JavaScript không đòi hỏi khách thăm phải sử dụng phiên bản mới của trình duyệt. Tuy nhiên như đã giải thích, việc kiểm tra rằng trang của mình có thể hiển thị đúng trên nhiều trình duyệt khác nhau là một ý tưởng tốt. Xem <http://browsers.evolt.org/> để tìm các liên kết đến phần lưu trữ những phiên bản cũ của các trình duyệt. Ngay cả khi không thể tiến hành kiểm thử trên nhiều trình duyệt, bạn vẫn có thể thiết kế trang web sao cho lỗi nếu có thì cũng ở mức độ chấp nhận được. Trang web cần hiển thị thích đáng bất kể người dùng sử dụng trình duyệt nào.

Có gì mới trong ECMAScript phiên bản 5?

Khi chuẩn ECMAScript-262 phiên bản 5 được phát hành năm 2009, nhiều cải tiến đã được đưa ra. Hầu hết những thay đổi này không tương thích ngược với phiên bản cũ hơn của các trình duyệt. Tuy nhiên, khi các phiên bản mới của trình duyệt ra đời, những thay đổi để thích ứng với ECMAScript 262 phiên bản 5 bắt đầu xuất hiện và trở nên sẵn dùng với các ứng dụng web đa trình duyệt.

Các phương thức mới để làm việc với mảng

ECMA-262 phiên bản 5 giới thiệu một số phương thức mới để dùng với mảng. Chúng bao gồm phương thức *foreach* để duyệt qua các phần tử của mảng cũng như phương thức *map*, *filter* và các phương thức khác xác định chỉ số và làm giảm kích thước mảng. Chương 8 “Các đối tượng trong JavaScript” sẽ trình bày chi tiết hơn về mảng và các phương thức mới này.

Những biện pháp kiểm soát mới đối với thuộc tính của đối tượng

Các thuộc tính (property) của đối tượng (sẽ giới thiệu trong Chương 8) có thể truy cập khá linh hoạt. ECMA-262 phiên bản 5 giới thiệu các hàm *get* và *set*, các hàm được gọi khi một thuộc tính được truy xuất hoặc được gán cho một giá trị tương ứng. Bạn có thể kiểm soát việc truy cập các thuộc tính này thông qua các đặc tính (attribute) *writable* (nếu đặc tính này có giá trị *false*, bạn không thể thay đổi giá trị của thuộc tính), *configurable* (nếu đặc tính này có giá trị *false*, bạn không thể xóa thuộc tính hoặc thay đổi các đặc tính *writable*, *configurable*, *enumerable*) và *enumerable* (nếu đặc tính này có giá trị *true*, bạn có thể sử dụng vòng lặp để duyệt các giá trị của thuộc tính). Ví dụ, một thuộc tính có một giá trị và được cấu hình giá trị *enumerable* bằng *false*, nghĩa là không thể dùng vòng lặp để duyệt giá trị của nó.

Hãy xem đoạn mã sau:

```
{  
  value: "testvalue"  
  enumerable: false  
}
```

Giá trị trên sẽ không được trả lại nếu chạy qua vòng lặp. Đừng nản lòng nếu bạn chưa hiểu cú pháp này. Các chương sau sẽ trình bày chi tiết về việc tạo các đối tượng và sử dụng vòng lặp.

Đối tượng mới JSON

JavaScript Object Notation (JSON) là một phương thức trao đổi dữ liệu thường dùng giữa ứng dụng phía client như trình duyệt chạy JavaScript và server. JSON đỡ công kẽm hơn XML trong việc trao đổi dữ liệu, đặc biệt trong các ứng dụng JavaScript và do đó thường được sử dụng như một định dạng dữ liệu thay thế dành cho các ứng dụng AJAX.

ECMA-262 phiên bản 5 bổ sung đối tượng JSON thuần vào đặc tả ngôn ngữ. Cuốn sách này sẽ trình bày về JSON trong phần IV “AJAX và tích hợp phía server”.

Những thay đổi với đối tượng Date

ECMA-262 phiên bản 5 bổ sung những phương thức mới cho đối tượng *Date* để phân tách và tạo dữ liệu ngày tháng theo định dạng ISO. Phương thức *toISOString()* tạo ra dữ liệu theo định dạng ISO-8601, như dưới đây:

2011-03-12T18:51:50.000Z



Chế độ strict mới

ECMA-262 phiên bản 5 giới thiệu chế độ mới để thực thi mã được gọi là *chế độ strict*. Trong chế độ strict, bộ JavaScript sử dụng một bộ các công cụ kiểm tra cú pháp nghiêm ngặt hơn giúp bắt các lỗi trong đoạn mã như lỗi mã hay biến không được khai báo.

Hỗ trợ của trình duyệt

ECMA-262 phiên bản 5 có rất nhiều thay đổi và cải tiến, ảnh hưởng đến việc lập trình với JavaScript. Đáng tiếc là, những thay đổi này không được hỗ trợ trong các trình duyệt cũ và nói ngắn gọn sẽ không được Internet Explorer hỗ trợ hoàn toàn. Bạn đọc có thể tìm thấy bảng so sánh tính tương thích của nhiều trình duyệt đối với một số tính năng có trong phiên bản ECMA 262 số 5 tại <http://kangax.github.com/es5-compat-table/>.

Để làm việc trong điều kiện thiếu sự hỗ trợ cho ECMA-262 phiên bản 5 trên các trình duyệt cũ, bạn cần viết những đoạn mã có cân nhắc đến chúng hoặc sử dụng một thư viện JavaScript như jQuery. Chương 21 “Giới thiệu về các thư viện và

Framework của JavaScript” trình bày về các thư viện trong JavaScript.

Bài tập

1. Đúng hay sai: JavaScript được định nghĩa bởi một tổ chức chuyên phát triển các chuẩn và được hỗ trợ trên tất cả các trình duyệt web.
2. Đúng hay sai: Khi JavaScript bị vô hiệu hóa trên máy tính của người dùng, bạn nên chặn quyền truy cập bởi không có lý do chính đáng nào lý giải cho việc vô hiệu hóa JavaScript.
3. Tạo ra một khối mã định nghĩa JavaScript, khối mã này có thể xuất hiện trên trang HTML trong khái `<head>` hoặc `<body>`.
4. Đúng hay sai: Cần khai báo phiên bản JavaScript đang được sử dụng trong khái định nghĩa DOCTYPE.
5. Đúng hay sai: Mã JavaScript có thể xuất hiện trong cả khái `<head>` và khái `<body>` của trang HTML.



Chương 2

Lập trình với JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Nắm được những tùy chọn có sẵn khi lập trình với JavaScript.
- Cấu hình máy tính của bạn để có được môi trường để lập trình JavaScript.
- Tạo mới và triển khai một ứng dụng JavaScript bằng Microsoft Visual Studio 2010.
- Tạo mới và triển khai một ứng dụng JavaScript bằng Eclipse.
- Tạo mới một ứng dụng JavaScript bằng Notepad (hoặc các trình soạn thảo khác).
- Nắm được các tùy chọn khi gỡ lỗi JavaScript.



Những tùy chọn khi lập trình với JavaScript

Vì JavaScript không phải là ngôn ngữ lập trình cần biên dịch, nên bạn không cần công cụ hay môi trường lập trình đặc biệt để viết và triển khai các ứng dụng JavaScript. Tương tự, bạn cũng không cần phần mềm server đặc biệt nào để chạy các ứng dụng này. Do đó, tùy chọn khi phát triển các chương trình JavaScript gần như vô hạn.

Bạn có thể sử dụng bất kỳ trình soạn thảo văn bản nào để viết mã JavaScript hay bất kỳ chương trình viết mã HTML (ngôn ngữ đánh dấu siêu văn bản) và CSS nào hoặc trong các môi trường phát triển tích hợp (IDE) đầy sức mạnh như Visual Studio. Bạn thậm chí có thể dùng cả ba cách trên. Ban đầu, có thể bạn phát triển web bằng Visual Studio nhưng sau đó bạn nhận thấy việc sử dụng những trình soạn thảo đơn giản như Notepad để xử lý JavaScript sẽ thuận tiện hơn. Quan trọng nhất là hãy sử dụng công cụ mà bạn cảm thấy phù hợp nhất với

mình.

Cuốn sách này tập trung chủ yếu hướng dẫn người dùng cách lập trình JavaScript với Visual Studio, tuy nhiên nhiều lúc bạn nên sử dụng các trình soạn thảo văn bản đơn giản như Notepad hay Vim (bạn có thể tải Vim về từ địa chỉ: <http://www.vim.org>). Bạn cũng có thể nhập trực tiếp các đoạn mã JavaScript vào thanh địa chỉ của trình duyệt bằng cách sử dụng giả giao thức *javascript*: như đã trình bày trong Chương 1 “Hiểu hơn về JavaScript”.

Khi đã lập trình JavaScript trong một thời gian, bạn sẽ nhận thấy có một số công việc được lặp đi lặp lại ở tất cả các trang web. Trong trường hợp này, chúng ta có thể đơn giản cắt dán các đoạn mã vào trang web được tạo. Tuy nhiên, cách tốt hơn cả là tạo một file bên ngoài chứa tất cả các hàm được sử dụng chung cho các trang web. Chương 10 “Mô hình đối tượng tài liệu” trình bày đầy đủ hơn về các hàm mặc dù vậy bạn sẽ thấy hàm được sử dụng xuyên suốt mười chương đầu.

Thiết lập môi trường



Phần này của cuốn sách trình bày về việc lập trình JavaScript bằng các công cụ khác nhau. Bạn cần biết cách sử dụng công cụ mà bạn cảm thấy tiện lợi nhất, vì vậy bạn đừng coi đây là danh sách công cụ mang tính toàn diện hay phiến diện.

Một công cụ hữu ích để lập trình JavaScript là Visual Studio 2010. Khi sử dụng công cụ này, một web server (chương trình máy chủ web) đơn giản – ASP.NET Development Server – đã được cài đặt sẵn, việc này sẽ giúp quá trình triển khai và kiểm thử ứng dụng trở nên dễ dàng. Điều này không có nghĩa là bạn phải mua cả bộ Visual Studio 2010 chỉ để phục vụ việc lập trình JavaScript. Nếu sử dụng Eclipse, công cụ thứ hai được đề cập trong chương này, bạn vẫn có thể kiểm thử các đoạn mã mình viết ra. Tương tự, bạn cũng có thể kiểm thử các đoạn mã JavaScript kể cả khi không dùng một IDE nào cả.

Microsoft Visual Web Developer 2010 Express cũng là một lựa chọn tốt để lập trình web. Công cụ này cung cấp cho người dùng giao diện của Visual Studio cùng một số công cụ và add-on (tiện ích bổ sung) trong gói sản phẩm miễn phí. Bạn có thể tải về tại địa chỉ: <http://www.microsoft.com/express/Web/>.

Lập trình với JavaScript không đòi hỏi bạn phải có web server. Trường hợp ngoại lệ là khi bạn lập trình web với JavaScript và XML không đồng bộ (AJAX).

AJAX không thể sử dụng giao thức *file://*, thêm vào đó Chính sách cùng nguồn gốc đã bàn đến ở Chương 1 chỉ cho AJAX hoạt động khi có web server. Tóm lại, nếu bạn có ý định lập trình với AJAX trong tương lai, bạn cần có một web server để làm việc.

Với AJAX, việc lập trình trở nên đơn giản hơn. Bạn có thể dùng bất kỳ web server nào vì mục đích chính của bạn là làm việc với HTML, JavaScript và có thể một chút CSS cho thêm phần thú vị. Apache server, (tải về tại <http://httpd.apache.org>), có thể chạy trên rất nhiều nền tảng khác nhau, gồm cả Microsoft Windows và hiện đang dần trở thành web server phổ biến nhất trên Internet.

Việc cấu hình Apache hay bất kỳ web server nào nằm ngoài phạm vi của cuốn sách này và xin lưu ý một lần nữa, bạn không nhất thiết phải có web server. Trên trang web của Apache có một số hướng dẫn rõ ràng về cách cài đặt Apache trên Windows và nếu bạn đang dùng bất kỳ phiên bản nào của Linux, việc cài đặt còn đơn giản hơn nhiều, thậm chí trong nhiều trường hợp Apache đã được cài sẵn. Nhiều ví dụ trong cuốn sách này sẽ chạy bất kể bạn dùng web server hay chỉ chạy cục bộ. Tuy nhiên, bạn nhất thiết phải có web server khi chạy các ví dụ sử dụng AJAX.



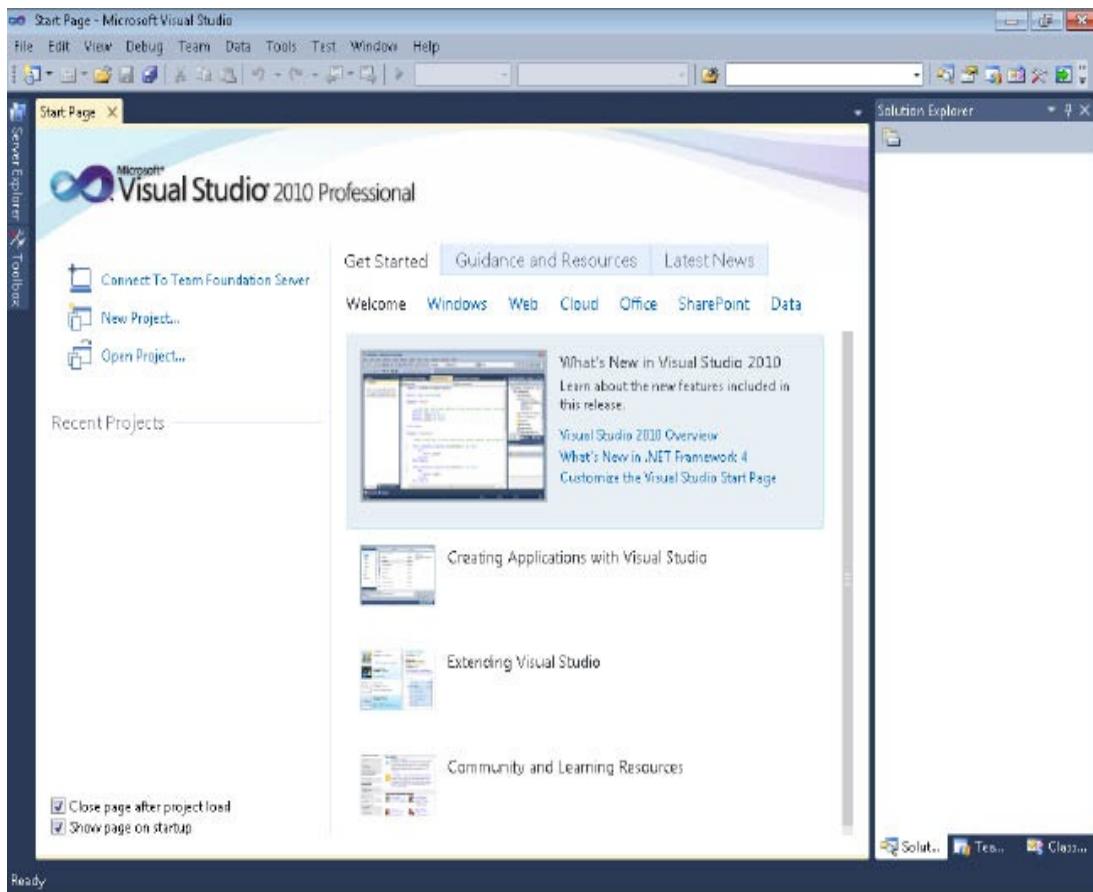
Lập trình JavaScript với Visual Studio 2010

Visual Studio 2010 giúp lập trình viên nhanh chóng triển khai các ứng dụng web được cải tiến tính năng bằng JavaScript. Lần đầu chạy Visual Studio 2010, người dùng có thể lựa chọn các kiểu môi trường phát triển Visual Studio khác nhau. Điều này sẽ làm cho giao diện hiển thị khác nhau, mỗi kiểu môi trường phát triển phù hợp với một mục đích hoặc một ngôn ngữ phát triển cụ thể. Hình 2-1 là ví dụ về hộp thoại thông báo trong Visual Studio.



Hình 2-1 Chọn kiểu môi trường lập trình.

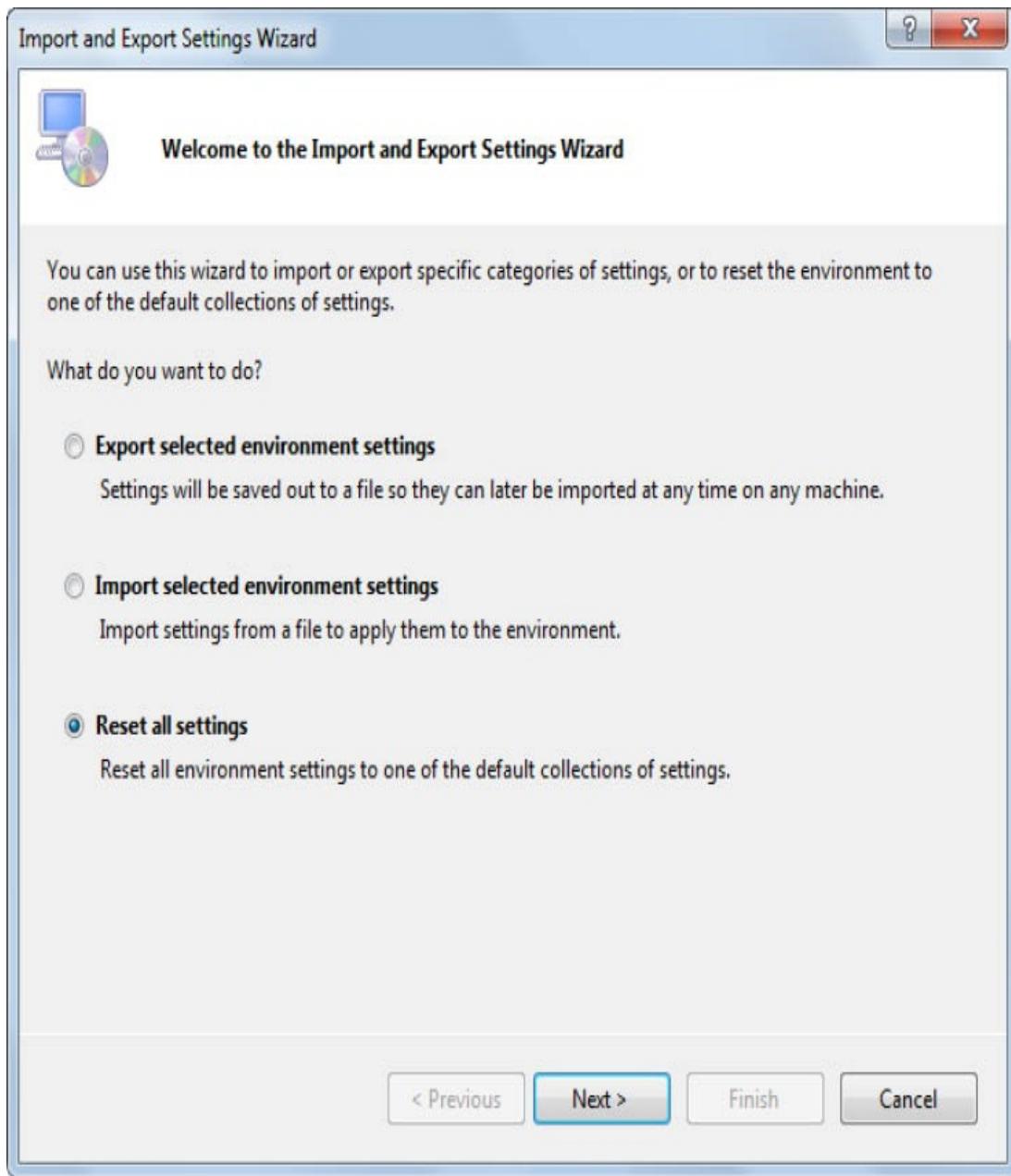
Nếu bạn chọn General Development Settings, Visual Studio của bạn sẽ trông giống như Hình 2-2 dưới đây:



Hình 2-2 General Development Settings cung cấp môi trường chung nhất cho các dự án lập trình.

Bạn có thể thay đổi thiết lập để Visual Studio sử dụng Web Development Settings bằng cách chọn Import and Export Settings từ menu Tools, khi đó Import And Export Settings Wizard sẽ hiển thị. Chọn Select Reset All Settings như trong Hình 2-3 và nhấn Next.

Chú ý Bạn không cần thay đổi thiết lập của môi trường lập trình. Tuy nhiên, trong cuốn sách này, chúng tôi giả định bạn đã chọn Web Development Settings trong Visual Studio 2010.



Hình 2-3 Chuẩn bị thay đổi thiết lập sang Web Development Settings trong Visual Studio 2010.

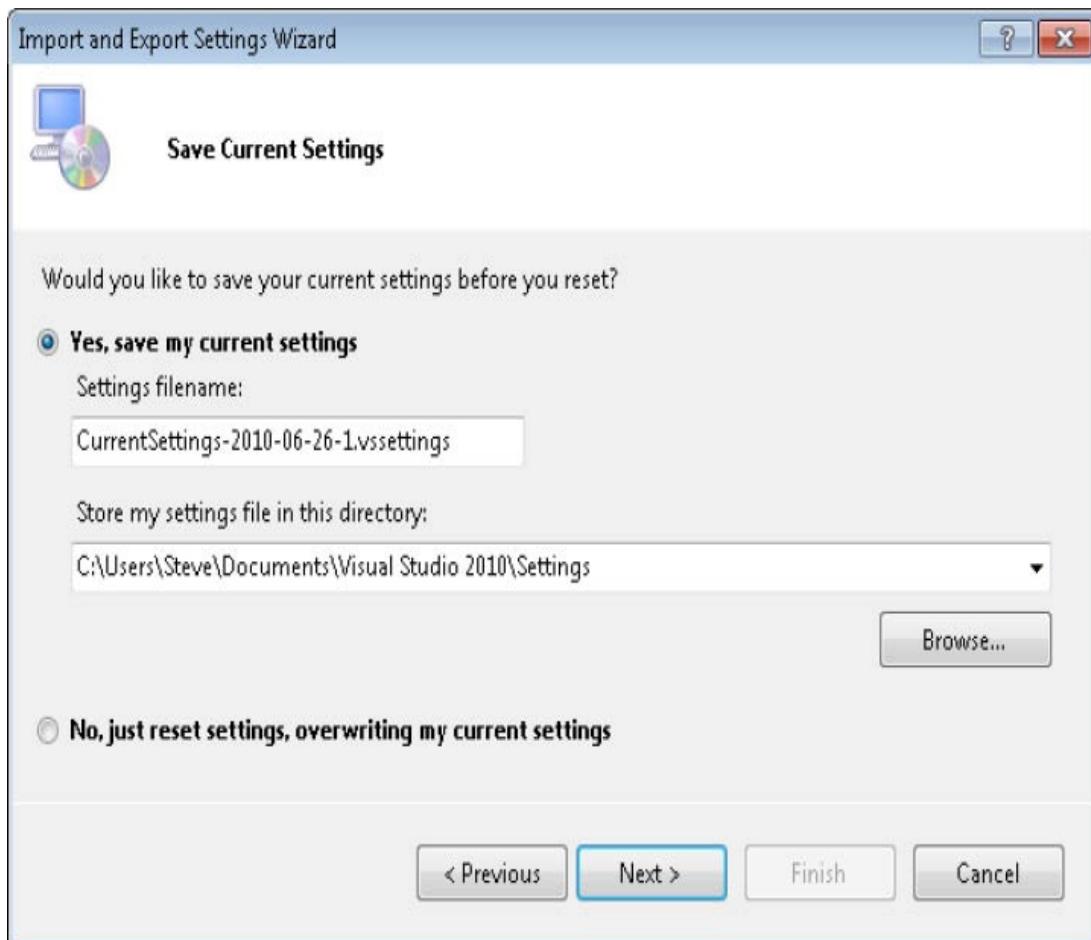
Tiếp đó, cửa sổ Save Current Settings sẽ xuất hiện như Hình 2-4. Nếu có thiết lập nào muốn lưu, chọn Yes, Save My Current Settings. Ngược lại, chọn No, Just Reset Settings, Overwriting My Current Settings. Chọn Next để tiếp tục.

Cửa sổ Choose A Default Collection Of Settings sẽ xuất hiện như Hình 2-5, chọn Select Web Development settings và chọn Finish.

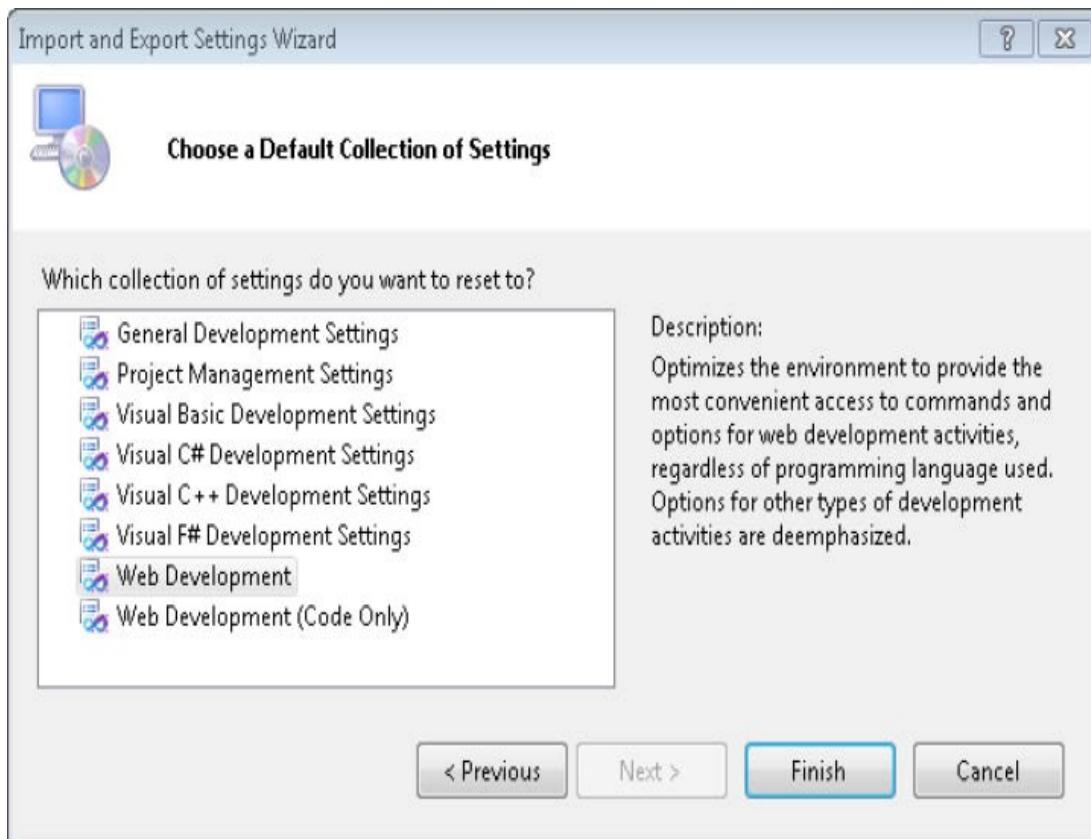
Hộp thoại Reset Complete xuất hiện. Nhấn vào Close để thiết lập lại môi trường

của bạn thành Web Development settings

Với các thiết lập Web Development hiện tại, người dùng có thể truy cập nhanh vào các tác vụ lập trình web với ASP.NET, JavaScript, HTML và CSS – những ngôn ngữ cốt lõi của web. Để tìm hiểu về các thiết lập trong Visual Studio 2010, xem thêm tại <http://msdn.microsoft.com/en-us/library/zbhkx167.aspx>.



Hình 2-4 Bằng cách lưu thiết lập hiện tại, người dùng có thể giữ cấu hình tùy chỉnh mà họ vừa chỉ định cho.



Hình 2-5 Chọn Web Development settings trong Visual Studio 2010.

Viết trang web đầu tiên có chứa mã JavaScript với Visual Studio 2010

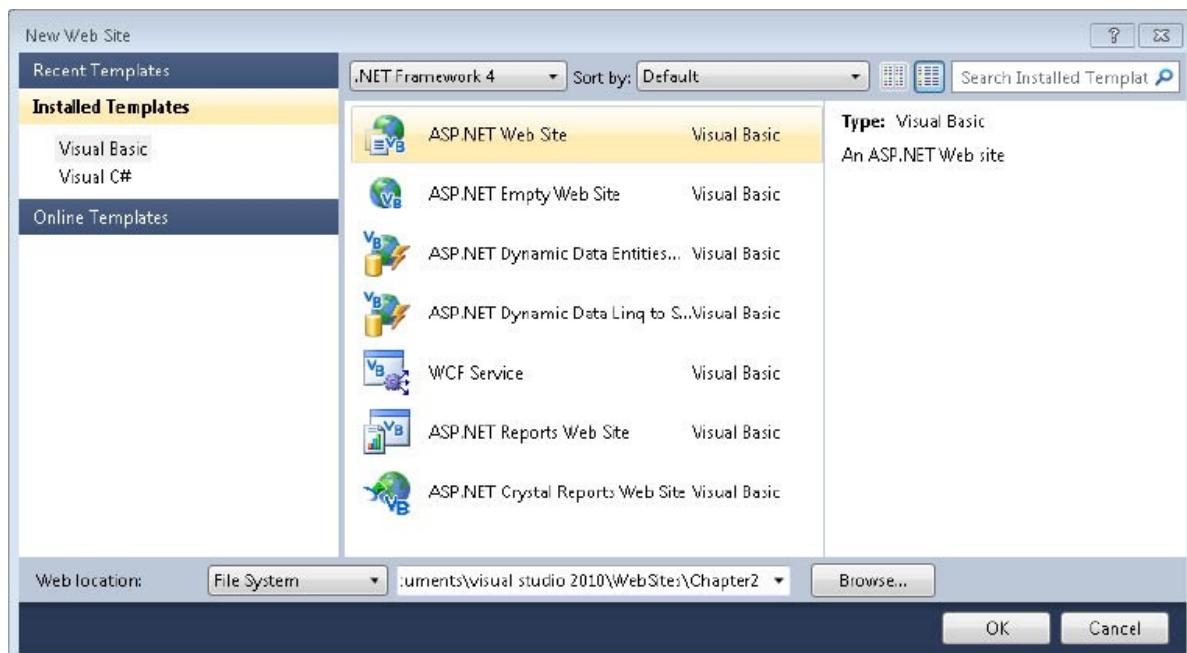
Ngay lúc này, bạn có thể tự tay tạo một dự án web và viết một đoạn mã JavaScript ngắn. Nếu không sử dụng Visual Studio, hãy bỏ qua phần này để đến phần “Lập trình JavaScript với Eclipse” hoặc “Lập trình JavaScript không dùng IDE”.

Chú ý Bạn có thể tải về các đoạn mã ví dụ của toàn bộ cuốn sách. Xem lại phần giới thiệu để biết cách tải về tài nguyên đi kèm.

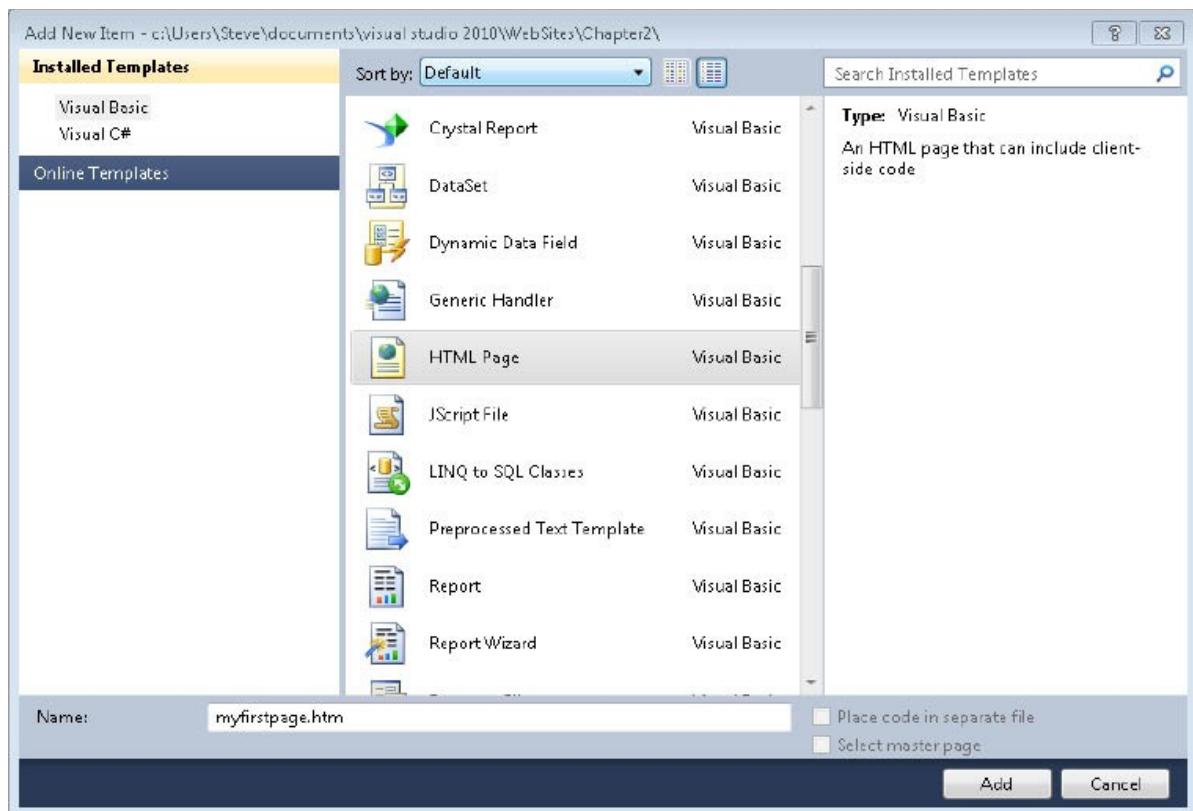
Tạo dự án web có chứa mã JavaScript bằng Visual Studio 2010

- Trong Visual Studio sử dụng Web Development Settings, chọn New Web Site từ menu File. Hộp thoại New Web Site mở ra.

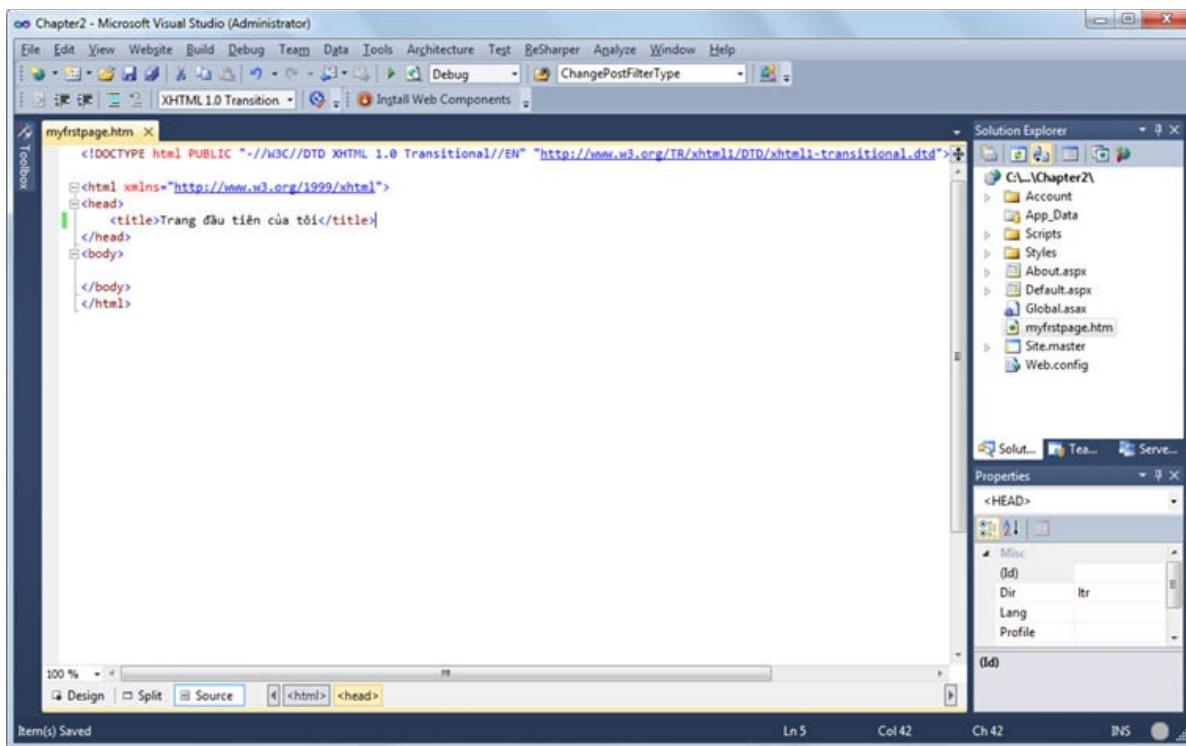
2. Chọn ASP.NET Web Site (việc chọn ngôn ngữ Visual Basic hay Visual C# không quan trọng) như hình dưới. Đổi tên thành **Chapter2**, với đường dẫn phù hợp với cấu hình đã thiết lập. Khi các thông tin được điền đúng và đủ, chọn “OK”. Visual Studio sẽ tạo một dự án web mới.



3. Visual Studio 2010 tự động tạo và mở một file Default.aspx trong chế độ soạn thảo. Hãy đóng file này và tạo file mới bằng cách nhấn chuột phải trong vùng Solution Explorer (Solution Explorer là một khung nằm ở phía trên bên phải của Visual Studio) và chọn Add New Item. (Hoặc bạn cũng có thể chọn New File từ menu File). Hộp thoại Add New Item mở ra như hình dưới đây. Chọn HTML Page, đổi tên thành **myfirstpage.htm**, sau đó nhấn Add. Visual studio tự động mở file này và chèn DOCTYPE cũng như các đoạn mã mặc định vào trang HTML.



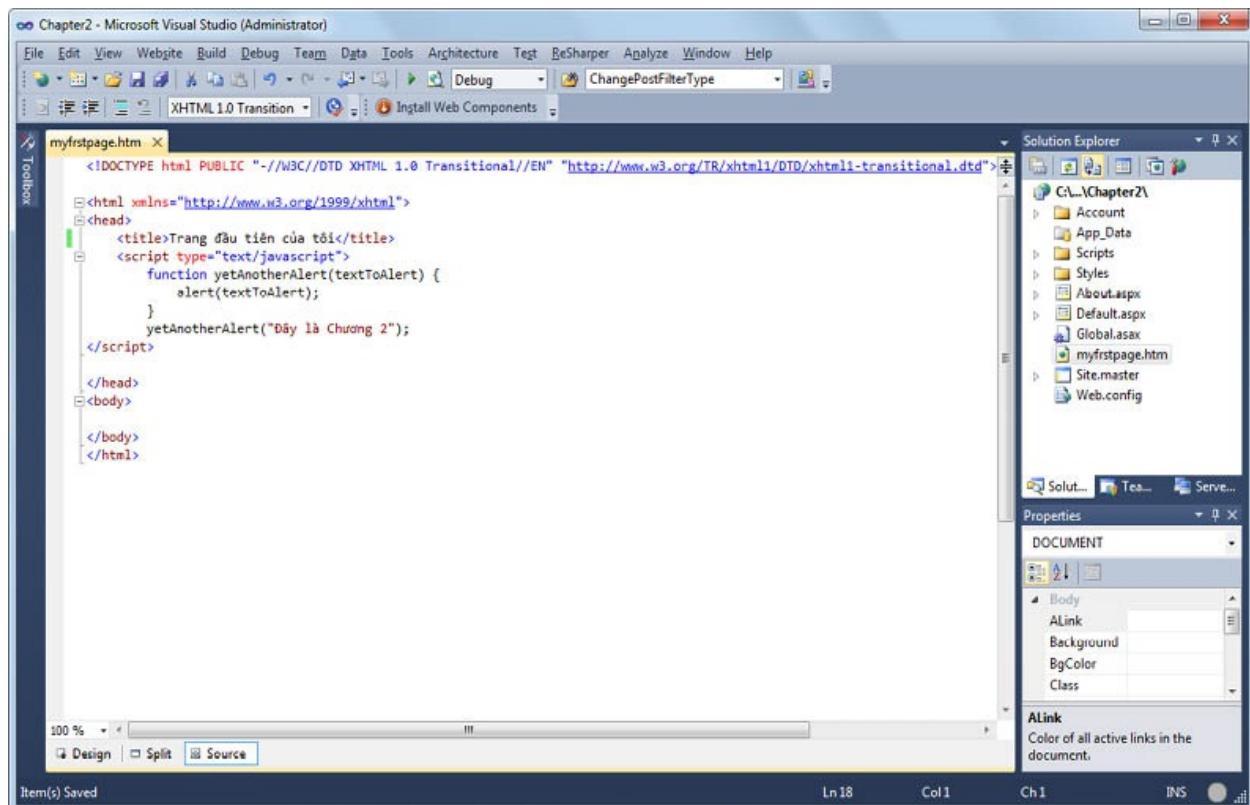
4. Trong trang myfirstpage.htm, đặt con trỏ chuột giữa cặp thẻ `<title>` `</title>` và đổi tựa đề thành **Trang đầu tiên của tôi**. Màn hình của bạn sẽ giống như hình dưới:



5. Giữa thẻ `<head>`, sau thẻ đóng `</title>`, thêm vào đoạn mã sau:

```
<script type="text/javascript">
    function yetAnotherAlert(textToAlert) {
        alert(textToAlert);
    }
    yetAnotherAlert("Đây là Chương 2");
</script>
```

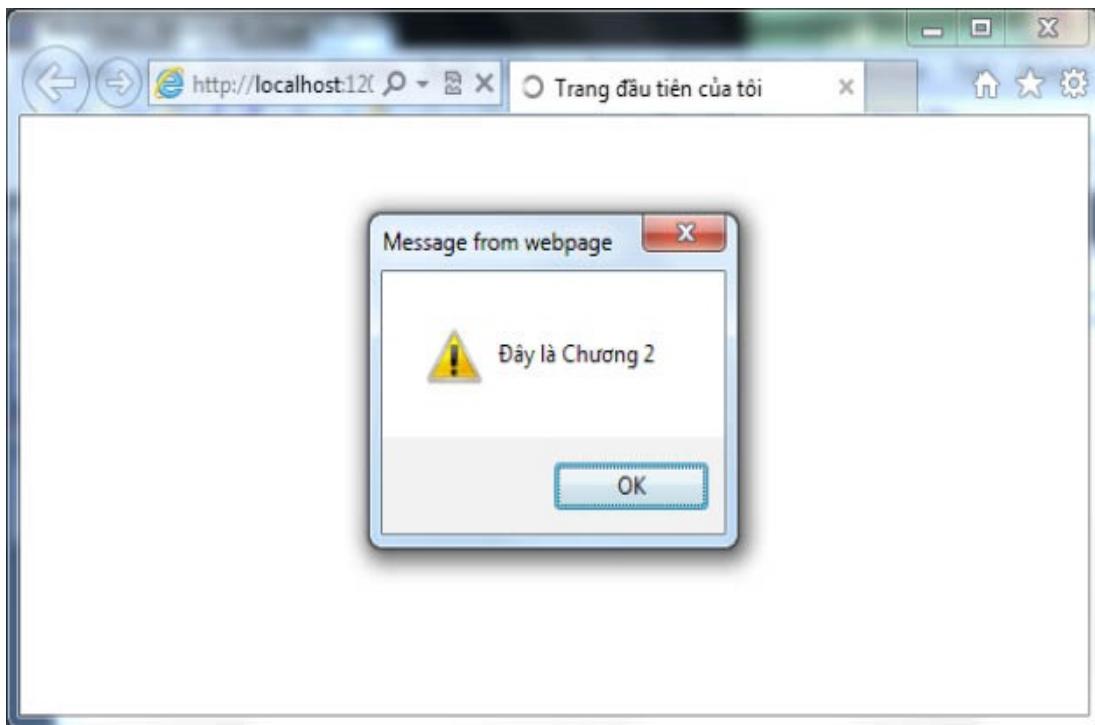
6. Chọn Save All từ menu File. Đoạn mã hoàn chỉnh và trang hiển thị giống như sau:



Để xem trang web vừa tạo, chọn Start Without Debugging từ menu Debug. Thao tác này khởi động ASP.NET Development Server - server phát triển ASP.NET (nếu chưa chạy) (nếu chương trình chưa chạy) và mở trang web trên trình duyệt mặc định. Trang web được mở ra với một hộp thoại thông báo như Hình 2-6. Nhấn OK và đóng trình duyệt.

Đoạn mã được thực thi như sau. Đầu tiên, thẻ script được gọi đến và khai báo rằng ngôn ngữ sử dụng là JavaScript:

```
<script type="text/javascript">
```



Hình 2-6 Chạy JavaScript trên ASP.NET Development Server.

Chú ý Bạn có thể khai báo JavaScript bằng cách khác, tuy nhiên đây là cách được hỗ trợ rộng rãi nhất.

Tiếp đó, đoạn mã khai báo hàm *yetAnotherAlert* nhận vào đối số *textToAlert* như sau:

```
function yetAnotherAlert(textToAlert) {
```

Hàm này có nhiệm vụ mở hộp thoại thông báo trên cửa sổ trình duyệt với bất kỳ đoạn văn bản nào được cung cấp làm đối số của hàm:

```
    alert(textToAlert);
```

Hàm kết thúc bởi dấu ngoặc nhọn ()). Dòng tiếp theo của đoạn mã gọi đến hàm vừa khai báo với đối số là một chuỗi đặt trong cặp ngoặc kép như sau:

```
yetAnotherAlert("Đây là Chương 2");
```

Với đoạn mã trên, bạn đã sẵn sàng lập trình với JavaScript bằng Visual Studio 2010. Nhưng trước khi ăn mừng, bạn cần học cách sử dụng các file JavaScript bên ngoài - external file (là những file nằm ngoài file HTML chứa mã

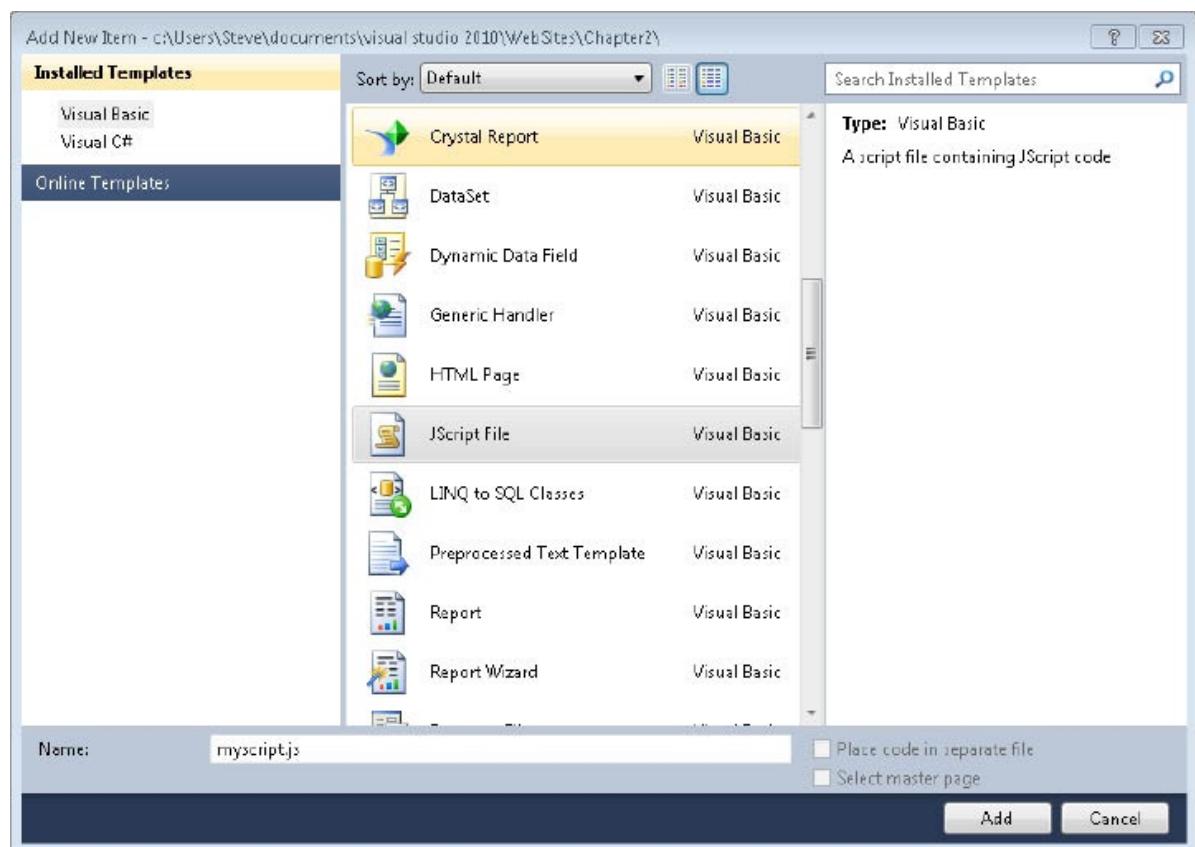
JavaScript).

Sử dụng file JavaScript ngoài với Visual Studio 2010

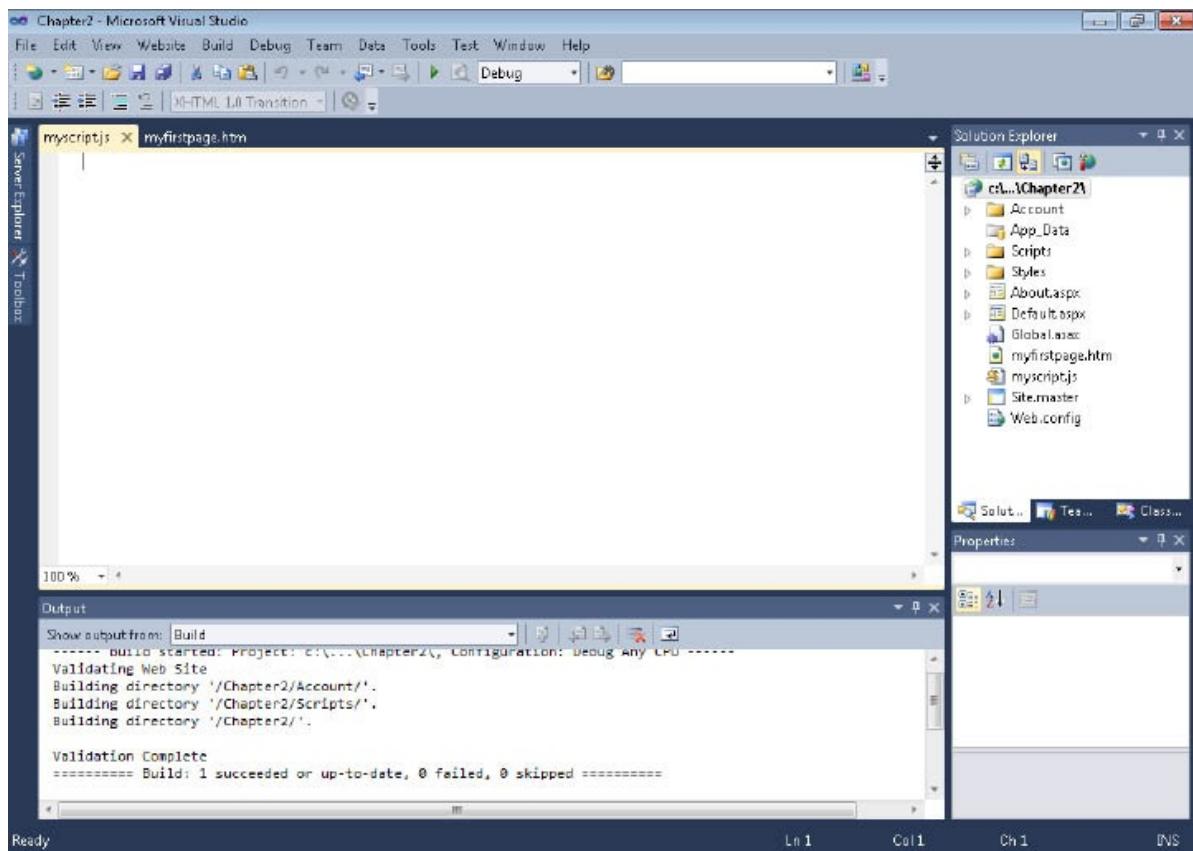
Bạn không cần phải đặt toàn bộ đoạn mã JavaScript vào các file HTML. Thay vào đó, bạn có thể tận dụng thuộc tính *src* của thẻ `<script>`. Thuộc tính của các thẻ giúp định nghĩa hoặc cung cấp thêm thông tin chi tiết cho một phần tử. Ví dụ, phần tử `<form>` có thể chứa một thuộc tính định nghĩa sự kiện nào sẽ xảy ra khi form được gửi đi. Với thuộc tính *src* của thẻ `<script>`, bạn có thể xác định vị trí của file JavaScript ngoài. Trình duyệt web sau đó sẽ đọc mã JavaScript chứa trong file này khi tải trang web. Với file JavaScript ngoài, bạn có thể giữ các đoạn mã JavaScript dùng chung ở cùng một nơi, thay vì phải cập nhật ở từng trang riêng biệt – điều này sẽ giúp tiết kiệm rất nhiều công sức. Lúc này, bạn đã tạo ra một trang web (tạo bởi Visual Studio) chỉ thực hiện một nhiệm vụ duy nhất là hiện ra một hộp thoại thông báo nhờ mã JavaScript nằm trong thẻ `<head>`. Trang web này có chứa mã JavaScript trong phần thẻ `<head>`. Trong phần tiếp theo, bạn sẽ học cách đặt mã JavaScript vào một file ngoài và tham chiếu đến đoạn mã trong file HTML.

Tạo file JavaScript ngoài với Visual Studio 2010

1. Nếu không mở được file myfirstpage.htm hãy chọn Open Project từ menu File của Visual Studio, chọn dự án mà bạn lưu file myfirstpage.htm và mở file này. Khi đó Visual Studio sẽ hiển thị như ở bước 6 của ví dụ trước.
2. Tạo mới một file để chứa mã JavaScript bằng cách chọn New File từ menu File. Hộp thoại Add New Item xuất hiện. Trong danh sách template (file mẫu), chọn Jscript File và đổi mục Name thành **myscript.js**, như hình dưới rồi nhấn vào nút Add. Lưu ý rằng danh sách template có thể khác nhau tùy thuộc bộ cài Visual Studio được sử dụng. Bạn có thể tìm thấy file myscript.js trong thư mục mã nguồn mẫu của Chương 2.



3. Một file JavaScript (Jscript) trống mở ra và được thêm vào dự án web. Bạn sẽ thấy một tab cho file myscript.js và một tab cho file myfirstpage.htm như hình dưới đây. Nếu file myfirstpage.htm chưa mở, hãy nhấn đúp để mở trong cửa sổ Solution Explorer.



Chú ý Phần đuôi mở rộng thông thường của file JavaScript và JScript là .js, nhưng bạn không bắt buộc phải dùng chúng. Sở dĩ chúng ta chọn kiểu file JScript trong bước 2 là vì kiểu file này tự động cập nhật đúng phần đuôi mở rộng. Bạn có thể dễ dàng chọn TextDocument từ hộp thoại Add New Item và đặt tên cho nó với phần mở rộng .js.

4. Nhấn vào tab myfirstpage.htm, bôi đen đoạn mã JavaScript. Tuy nhiên, giữ nguyên đoạn mã bên trong thẻ `<script>` và `</script>`. (Hiện tại, bạn chưa cần sử dụng những thẻ trên, chúng ta sẽ bàn đến chúng ở phần sau). Bạn cũng có thể tìm thấy file myfirstpage.htm trong thư mục mã nguồn mẫu của Tài nguyên đi kèm.
5. Sao chép đoạn mã được bôi đen bằng cách chọn Copy từ menu Edit.
6. Nhấn vào tab myscript.js, trỏ chuột đến dòng đầu tiên và chọn Paste từ menu Edit. Đoạn mã được dán tại vị trí trỏ chuột. Thay đổi đoạn văn bản trong lời gọi hàm yetAnotherAlert thành “Đây là ví dụ thứ hai” như dưới đây:

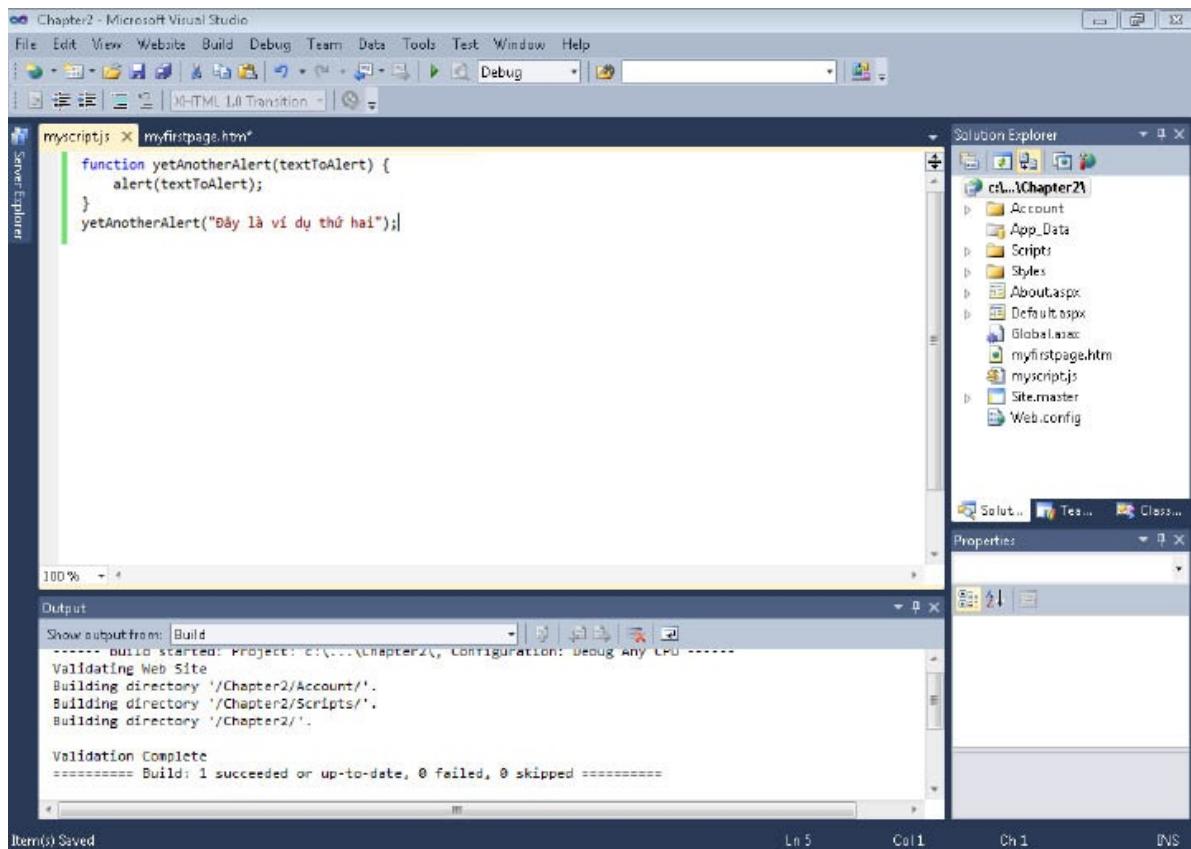
```
function yetAnotherAlert(textToAlert) {
```

```

        alert(textToAlert);
    }
yetAnotherAlert("Đây là ví dụ thứ hai");

```

7. Lưu file myscript.js bằng cách chọn Save từ menu File. File này có dạng giống như sau:



8. Với mã JavaScript được lưu trong file myscript.js, bạn có thể xóa đoạn mã JavaScript trong file myfirstpage.htm, chỉ để lại cặp thẻ script có dạng như sau:

```

<script type="text/javascript">
</script>

```

9. Bây giờ, thêm thuộc tính *src* vào thẻ mở *<script>*:

```

<script type="text/javascript" data-src="myscript.js">

```

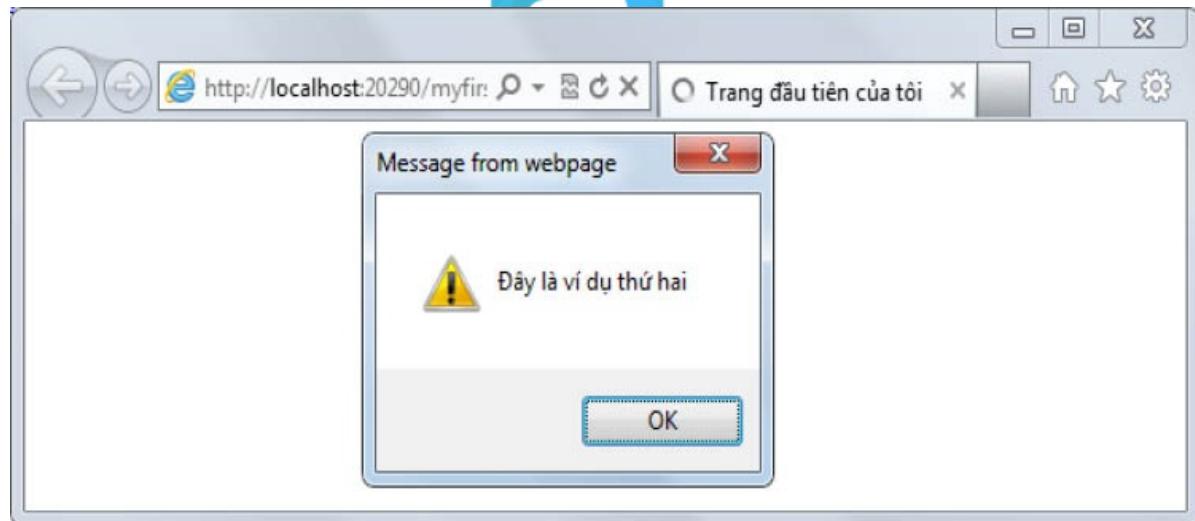
10. Nếu muốn, bạn có thể chuột lại đoạn mã bằng cách xóa dấu chuyển dòng và đưa thẻ đóng *</script>* lên cùng dòng với thẻ mở.

```
<script type="text/javascript" data-src="myscript.js"></sc
```

Toàn bộ nội dung của file myfirstpage.html sẽ giống như dưới đây:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" >  
<head>  
<title>Trang đầu tiên của tôi</title>  
<script type="text/javascript" data-src="myscript.js"></script>  
</head>  
<body>  
</body>  
</html>
```

- |1. Lưu file myfirstpage.html.
- |2. Để xem trang bằng trình duyệt, chọn Start Without Debugging từ menu Debug. Trang web được duyệt qua web server và mở trên trình duyệt. Kết quả là hộp thoại thông báo “Đây là ví dụ thứ hai” như hình sau:



- |3. Nhấn OK để đóng hộp thoại thông báo. Bây giờ, xem lại các đoạn mã để thấy sự khác biệt. Trên trình duyệt web, chọn Source từ menu View. Lưu ý lúc này thẻ <script> chưa một tham chiếu đến file JavaScript ngoài.

Như vậy, bạn đã biết cách lập trình JavaScript với Visual Studio 2010. Tới đây, bạn có thể bỏ qua một số phần để đến phần giới thiệu về cách gỡ lỗi hoặc đọc

tiếp để tìm hiểu cách lập trình JavaScript với các công cụ khác.

Lập trình JavaScript bằng Eclipse

Một IDE phổ biến khác trong giới lập trình web là Eclipse. Lập trình viên có thể cài đặt các framework khác nhau để hỗ trợ cho các mục đích lập trình khác nhau. Ví dụ, họ có thể cài đặt Web Tools Platform hay bộ công cụ lập trình PHP (PHP Hypertext Preprocessor) để tạo môi trường tiện ích tối ưu hóa cho công việc của họ. Việc thảo luận về các dự án Eclipse tiềm năng không thuộc phạm vi của cuốn sách này, tuy nhiên chúng ta sẽ thảo luận về việc lập trình JavaScript trên nền Eclipse cơ bản.

Để lập trình JavaScript với Eclipse, trước tiên bạn phải tải về bộ cài Eclipse và đôi khi nền tảng chạy các ứng dụng Java (JRE). Thông tin chi tiết và cách thức tải về có trên trang web Eclipse (<http://www.eclipse.org>). Trong phần này, chúng tôi giả định bạn chưa từng dùng Eclipse và đây là lần đầu tiên bạn làm quen với Eclipse. Tuy nhiên, phần này không cung cấp hướng dẫn cài đặt Eclipse. Bạn nên tìm đọc các tài liệu hướng dẫn có sẵn trên website của Eclipse để nắm được những thông tin mới cập nhật.



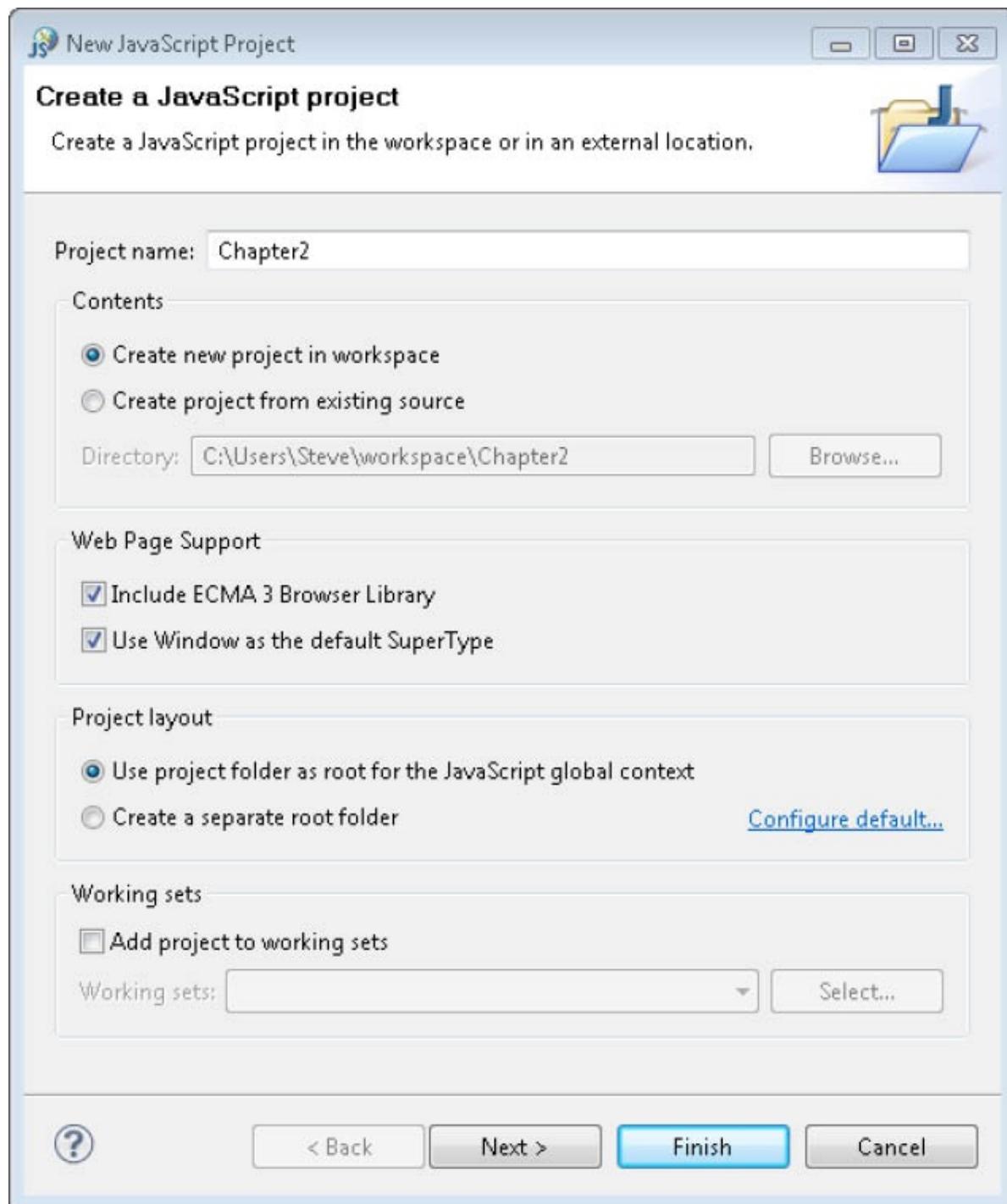
Viết dự án web đầu tiên chứa mã JavaScript bằng Eclipse

Bây giờ đã đến lúc tạo một trang web có chứa mã JavaScript bằng Eclipse. Nếu bạn không dùng Eclipse, hãy bỏ qua phần này. Ở phần cuối của chương, chúng ta sẽ thảo luận cách lập trình không cần đến IDE và một số lời khuyên nhỏ khi gỡ lỗi JavaScript.

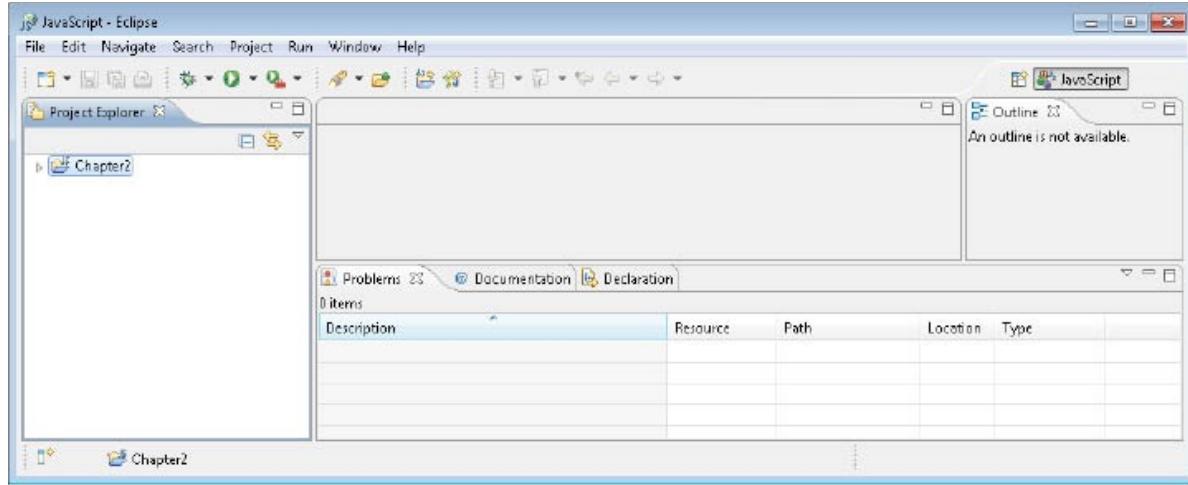
Chú ý Phần này sẽ hướng dẫn sử dụng IDE Eclipse cho các lập trình viên JavaScript. Giao diện Eclipse của bạn có thể sẽ khác một chút với các hình minh họa trong cuốn sách. Khi mở Eclipse lần đầu, bạn sẽ được yêu cầu chọn vùng làm việc (workspace). Hãy chọn vùng làm việc mặc định.

Tạo dự án web với JavaScript trong Eclipse

1. Tạo dự án mới bằng cách chọn New > JavaScript từ menu File. Hộp thoại JavaScript xuất hiện. Nhập **Chapter2** vào mục Project name (tên dự án) rồi nhấn Finish.

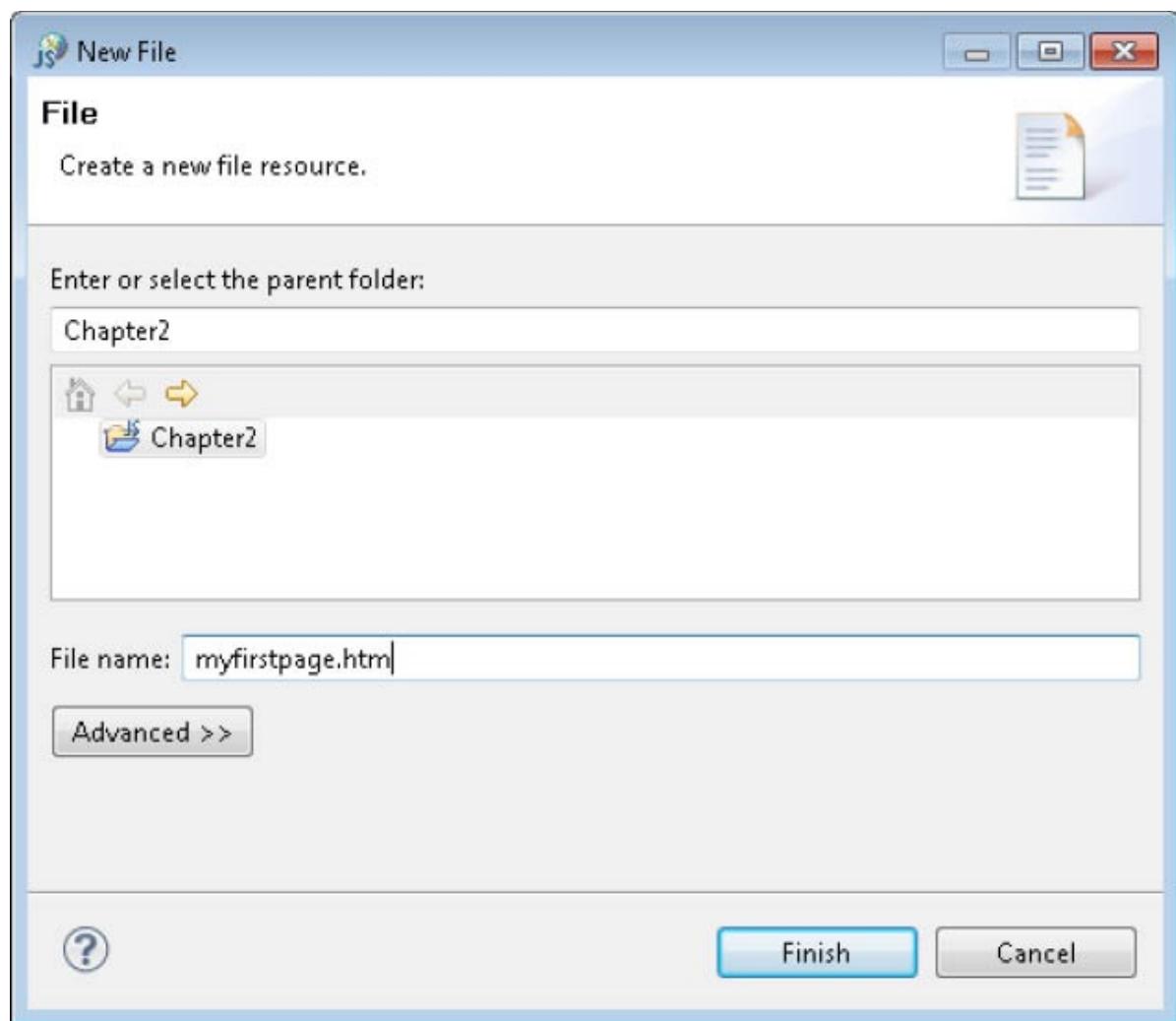


2. Thư mục rỗng Chapter2 được mở ra trong phần Project Explorer như bên dưới:

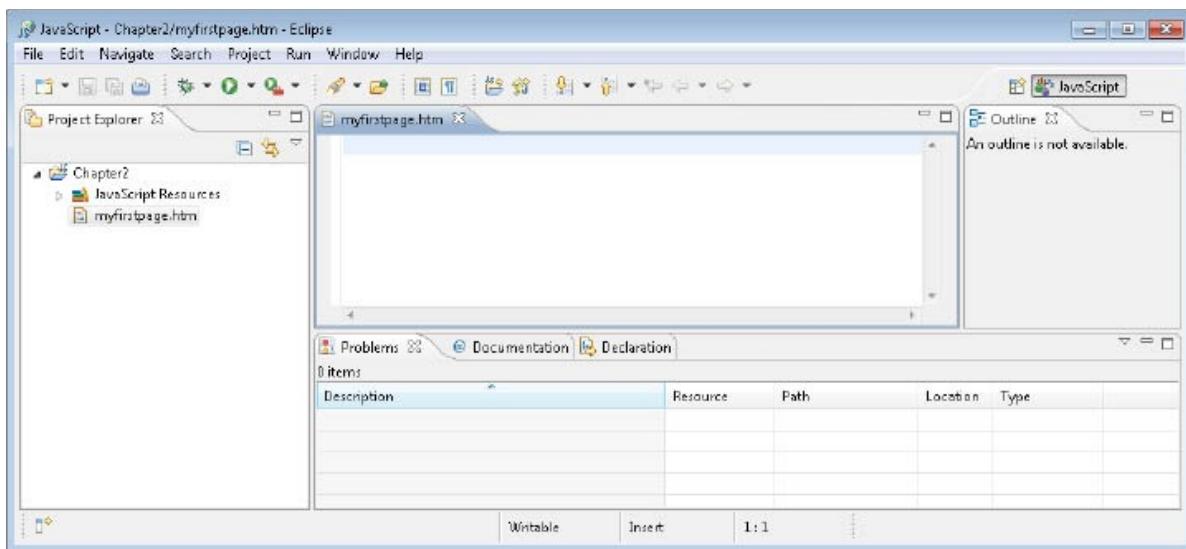


3. Nhấn chuột phải vào thư mục Chapter2, chọn New > File, hộp thoại New File mở ra. Nhập **myfirstpage.htm** vào trường File Name và chọn Finish. Bạn có thể tìm thấy file **eclipse_myfirstpage.htm** trong thư mục mã nguồn mẫu của Chương 2. Nếu muốn sử dụng file này, hãy đặt lại tên file thành **myfirstpage.htm**.





4. Sau khi chọn Finish, Eclipse mở trang vừa tạo bằng trình duyệt riêng. Tuy vậy, chúng ta cần sửa chữ không phải chạy trang web. Nhấn phải chuột vào file myfirstpage.htm trong Project Explorer và chọn Open. Trang này được mở trực tiếp trong trình soạn thảo của Eclipse như sau:



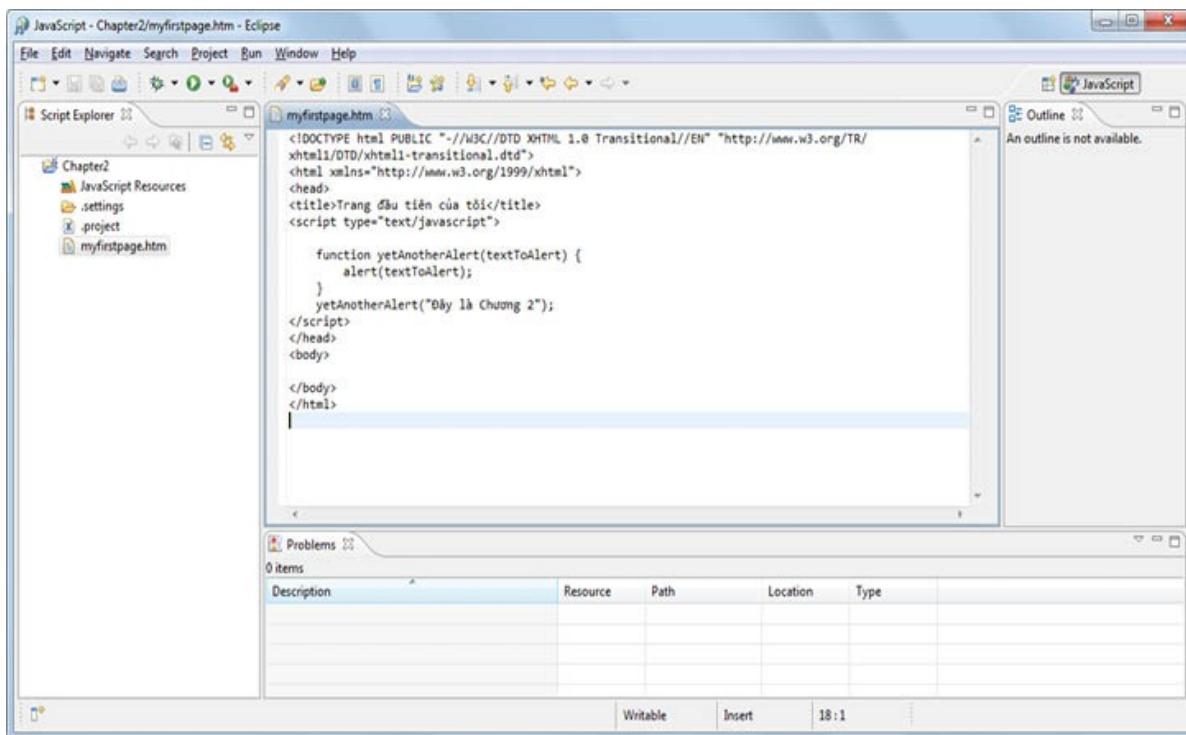
5. Bây giờ là lúc viết mã trong cửa sổ soạn thảo. Hãy nhập vào đoạn mã dưới đây:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional,
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Trang đầu tiên của tôi</title>
<script type="text/javascript">
function yetAnotherAlert(textToAlert) {
alert(textToAlert);
}
yetAnotherAlert("Đây là Chương 2");
</script>
</head>
<body>
</body>
</html>
```

Chú ý Với mục đích của ví dụ này, nếu không muốn mất thời gian, bạn có thể bỏ qua phần khai báo DOCTYPE và bắt đầu với thẻ `<html>`.

Ngược lại, nếu bạn lập trình web thật sự thì việc khai báo DOCTYPE là cần thiết. Xem Chương 1 để tìm hiểu tầm quan trọng của khai báo DOCTYPE.

6. Chọn Save từ menu File. Đoạn mã và trang hoàn chỉnh sẽ như hình sau:



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Trang đầu tiên của tôi</title>
<script type="text/javascript">

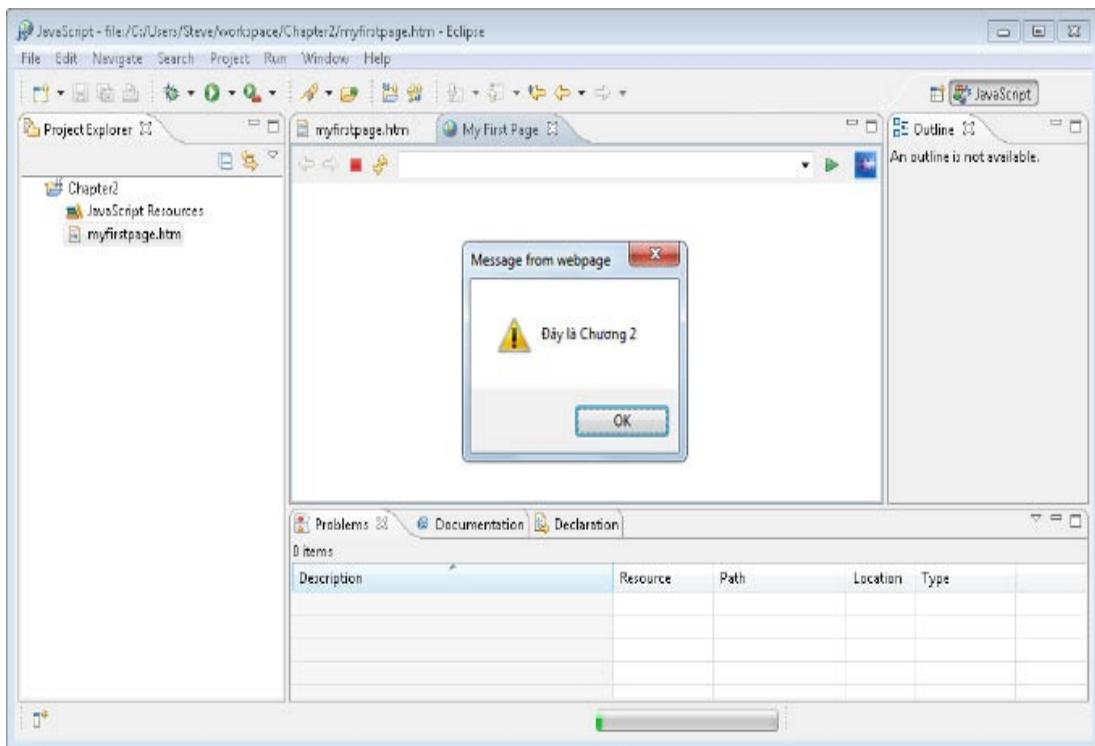
    function yetAnotherAlert(textToAlert) {
        alert(textToAlert);
    }
    yetAnotherAlert("Đây là Chương 2");
</script>
</head>
<body>

</body>
</html>
```

Để xem trang vừa tạo, nhấn chuột phải vào phần Project Explorer, chọn Open With rồi chọn Web Browser. Trang web sẽ hiển thị trên trình duyệt của Eclipse với một thông báo như Hình 2-7.

Ngoài ra, bạn cũng có thể xem trang vừa tạo bằng các trình duyệt khác chẳng hạn như trình duyệt mặc định. Để mở bằng trình duyệt mặc định, duyệt đến file vừa tạo (ví dụ C:\Users\Steveworkspace\Chapter2\folder) và nhấn đúp vào file.

Chú ý Nếu sử dụng Windows Internet Explorer, bạn có thể nhận được cảnh báo về việc truy cập các nội dung bị chặn, tuỳ thuộc vào mức độ bảo mật được cấu hình. Truy cập địa chỉ <http://windows.microsoft.com/en-US/windows7/Internet-Explorer-Information-bar-frequently-asked-questions> để tìm hiểu thêm về tính năng này và làm thế nào để tắt nó.



Hình 2-7 Hiển thị file trong Eclipse.

Trong ví dụ này, bạn đã tạo được một trang web cơ bản có nhúng mã JavaScript. Phần JavaScript của trang web chỉ chứa một vài phần tử. Đầu tiên, thẻ script được gọi đến và khai báo rằng ngôn ngữ sử dụng là JavaScript như dưới đây:

```
<script type="text/javascript">
```

Chú ý Bạn có thể khai báo JavaScript bằng cách khác, tuy nhiên đây là cách được hỗ trợ rộng rãi nhất.

Tiếp đó, đoạn mã khai báo hàm *yetAnotherAlert* có một tham số *textToAlert* như sau:

```
function yetAnotherAlert(textToAlert) {
```

Nội dung của hàm có nhiệm vụ duy nhất là hiển thị một hộp thoại thông báo trong cửa sổ trình duyệt. Thông báo được hiển thị có thể là bất kỳ đoạn văn bản nào được truyền vào như tham số của hàm. Tất cả được thực hiện bởi lệnh dưới đây:

```
    alert(textToAlert);
```

Hàm kết thúc bởi dấu ngoặc nhọn.

}

Dòng tiếp theo của đoạn mã gọi đến hàm vừa khai báo với đối số là một chuỗi đặt trong ngoặc kép như sau:

```
yetAnotherAlert("Đây là Chương 2");
```

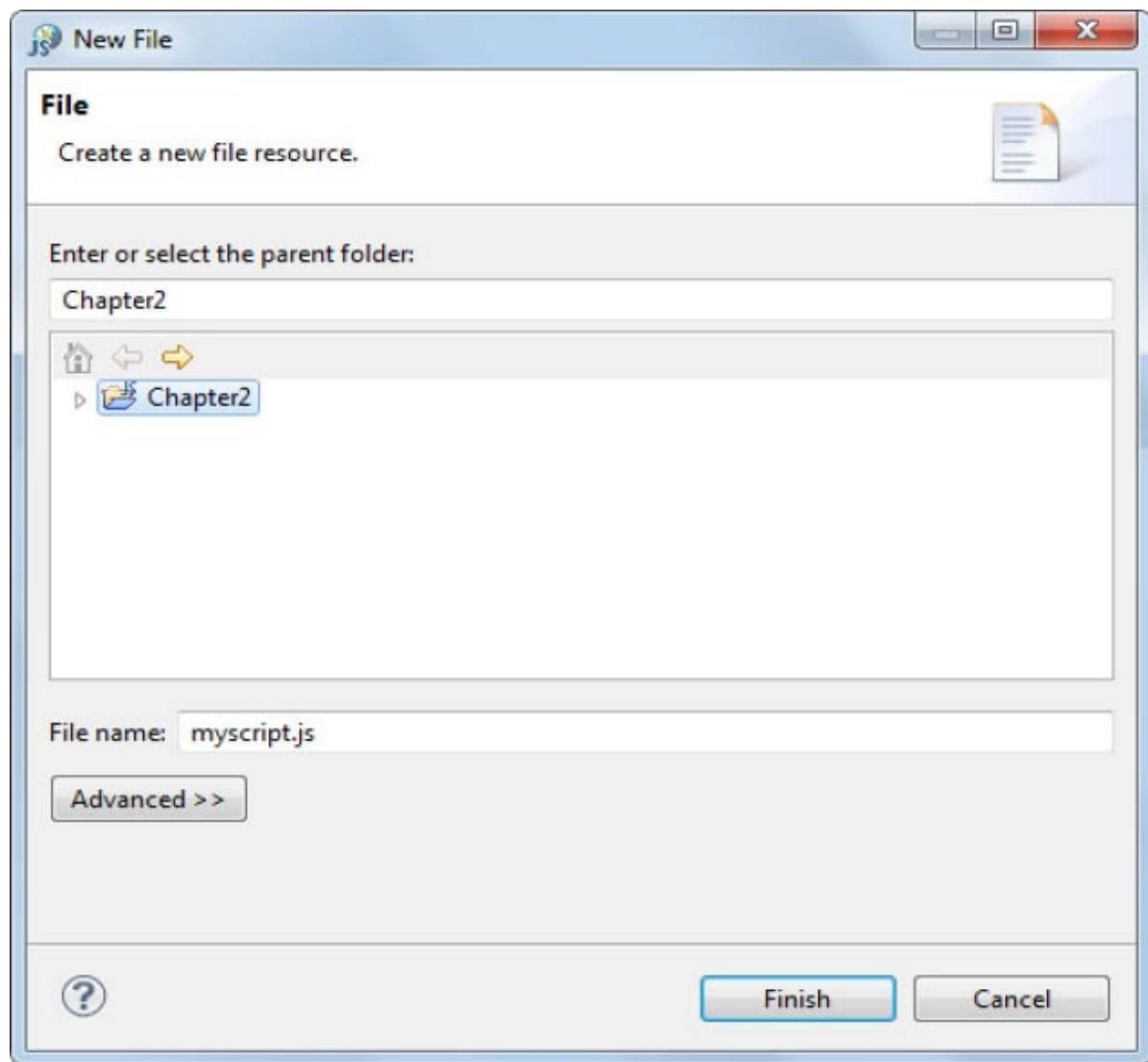
Qua ví dụ ngắn trên, bạn đã thấy cách lập trình JavaScript với Eclipse. Phần tiếp theo sẽ hướng dẫn cách đặt mã JavaScript vào một file bên ngoài, đây là phương thức được sử dụng phổ biến khi lập trình JavaScript.

Sử dụng file JavaScript ngoài với Eclipse

Lúc này, bạn đã có một trang web (được tạo với Eclipse) hiển thị một hộp thoại thông báo khi được duyệt. Đoạn mã JavaScript thực hiện việc này nằm trong cặp thẻ <head> của trang. Trong phần tiếp theo, bạn sẽ được hướng dẫn cách lưu mã JavaScript vào một file bên ngoài và cách gọi đến file này trong mã HTML.

Tạo file JavaScript ngoài với Eclipse

- Đầu tiên, cần chắc chắn đã mở file myfirstpage.htm trong Eclipse (Bạn có thể tìm thấy file này trong phần Tài nguyên đi kèm cuốn sách) bằng cách chọn dự án chứa file myfirstpage.htm và mở nó bằng cách nhấn chuột phải, chọn Open With, chọn Text Editor.
- Tạo một file mới để chứa mã JavaScript bằng cách chọn New > File từ menu File. Hộp thoại New File mở ra. Nhập **myscript.js** vào trường File name như bên dưới rồi chọn Finish.



3. Eclipse sẽ thêm một file JavaScript rỗng vào dự án. Nếu file này chưa tự động mở, nhấp chuột phải vào file myscript.js trong cửa sổ Project Explorer, chọn Open With > Text Editor. Lúc này bạn sẽ thấy hai tab cho file myscript.js và file myfirstpage.htm.

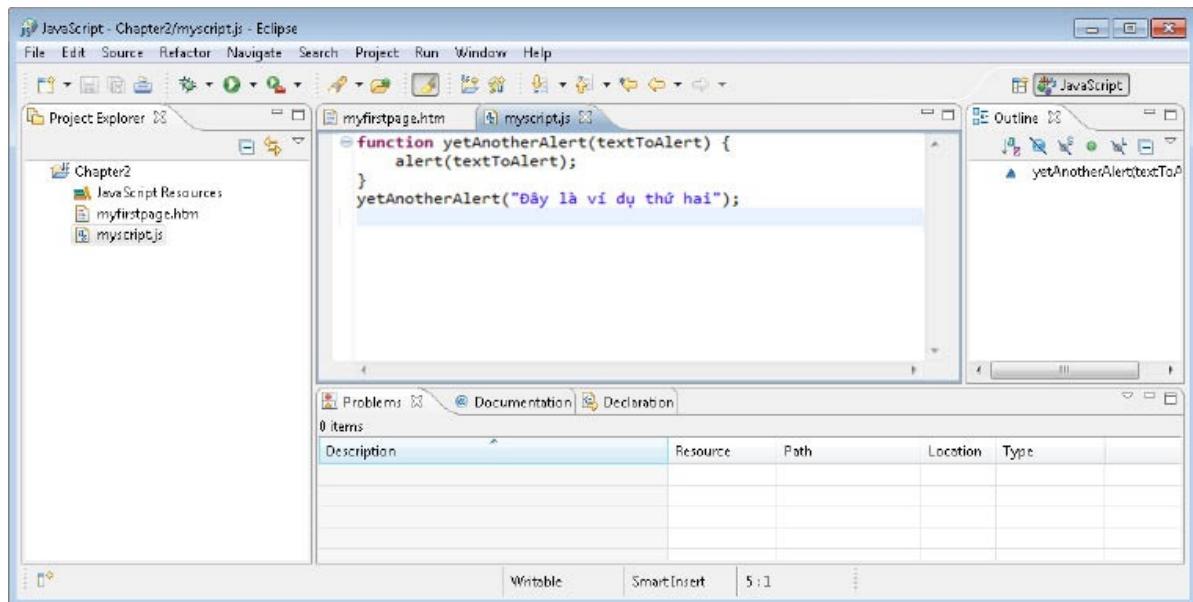
Chú ý Bạn không cần sử dụng đuôi .js cho phần mở rộng cho file JavaScript, tuy nhiên đuôi này sẽ giúp bạn dễ dàng nhận diện các file về sau.

4. Nhấp chuột vào tab myfirstpage.htm, bôi đen đoạn mã JavaScript đã tạo lúc trước, ngoại trừ cặp thẻ `<script> </script>` (Chúng ta sẽ để cập đến cặp thẻ này sau).

5. Sao chép đoạn mã đã chọn bằng cách chọn Copy từ menu Edit.
6. Nhấn chuột vào tab myscript.js, dán đoạn mã đã sao chép bằng cách chọn Paste từ menu Edit. Đổi đoạn văn bản trong lời gọi hàm thành “Đây là ví dụ thứ hai”. Đoạn mã sẽ giống như sau:

```
function yetAnotherAlert(textToAlert) {  
    alert(textToAlert);  
}  
yetAnotherAlert("Đây là ví dụ thứ hai");
```

7. Lưu file myscript.js bằng cách chọn Save từ menu File. File này sẽ giống như hình minh họa dưới:



8. Khi mã JavaScript đã được lưu trong file myscript.js, bạn có thể xóa đoạn mã JavaScript trong file myfirstpage.htm, chỉ để lại cặp thẻ script như dưới đây:

```
<script type="text/javascript">  
</script>
```

9. Bây giờ, thêm thuộc tính *src* vào thẻ mở *<script>*:

```
<script type="text/javascript" data-src="myscript.js">
```

10. Nếu muốn, bạn có thể chuột đoạn mã bằng cách xóa dấu chuyển dòng để đưa đoạn mã về cùng một dòng như sau:

```
<script type="text/javascript" data-src="myscript.js"></sc
```

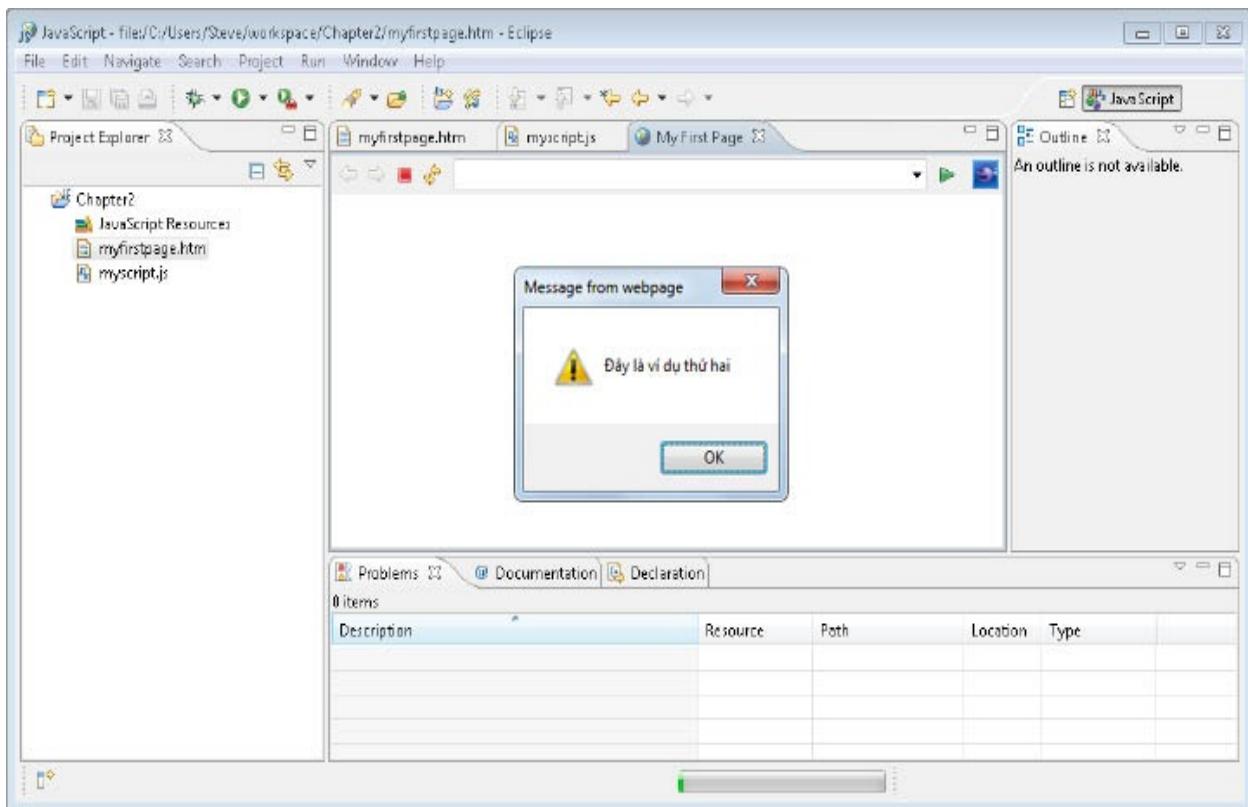
Lúc này, toàn bộ nội dung của myfirstpage.html sẽ giống như dưới đây:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional,
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Trang đầu tiên của tôi</title>
<script type="text/javascript" data-src="myscript.js"></sc
</head>
<body>
</body>
</html>
```

1. Lưu file myfirstpage.htm.
2. Xem trang web trên trình duyệt bằng cách nhấn chuột phải vào file myfirstpage.htm trong cửa sổ Project Explorer, trả đến Open With, chọn Web Browser. Trang web sẽ được mở trực tiếp trên trình duyệt của Eclipse.

Kết quả trả về sẽ là một hộp thoại thông báo với dòng chữ “Đây là ví dụ thứ hai” như sau:





Bài học cơ bản về lập trình JavaScript với Eclipse đã hoàn tất. Mặc dù vậy, còn có rất nhiều điều bổ ích về lập trình trong Eclipse, bạn nên truy cập vào trang web của Eclipse để tìm hiểu thêm.

Lập trình JavaScript không dùng IDE

Trong nhiều trường hợp, việc sử dụng các chương trình soạn thảo văn bản như Notepad hay Vim sẽ là cách tiếp cận đơn giản và dễ dàng hơn việc sử dụng các IDE khi lập trình JavaScript. Tuy nhiên, cần tránh việc sử dụng các hệ thống xử lý văn bản phức tạp như Microsoft Office Word vì nó có thể tự động thêm vào file mã nguồn những thành phần không thể hiện trên màn hình khiến cho hoạt động của trang web bị hỏng.

Viết dự án web đầu tiên có chứa mã

JavaScript bằng Notepad

Phần này sẽ giới thiệu một ví dụ về lập trình với JavaScript trên Notepad.

Tạo trang web chứa mã JavaScript bằng Notepad

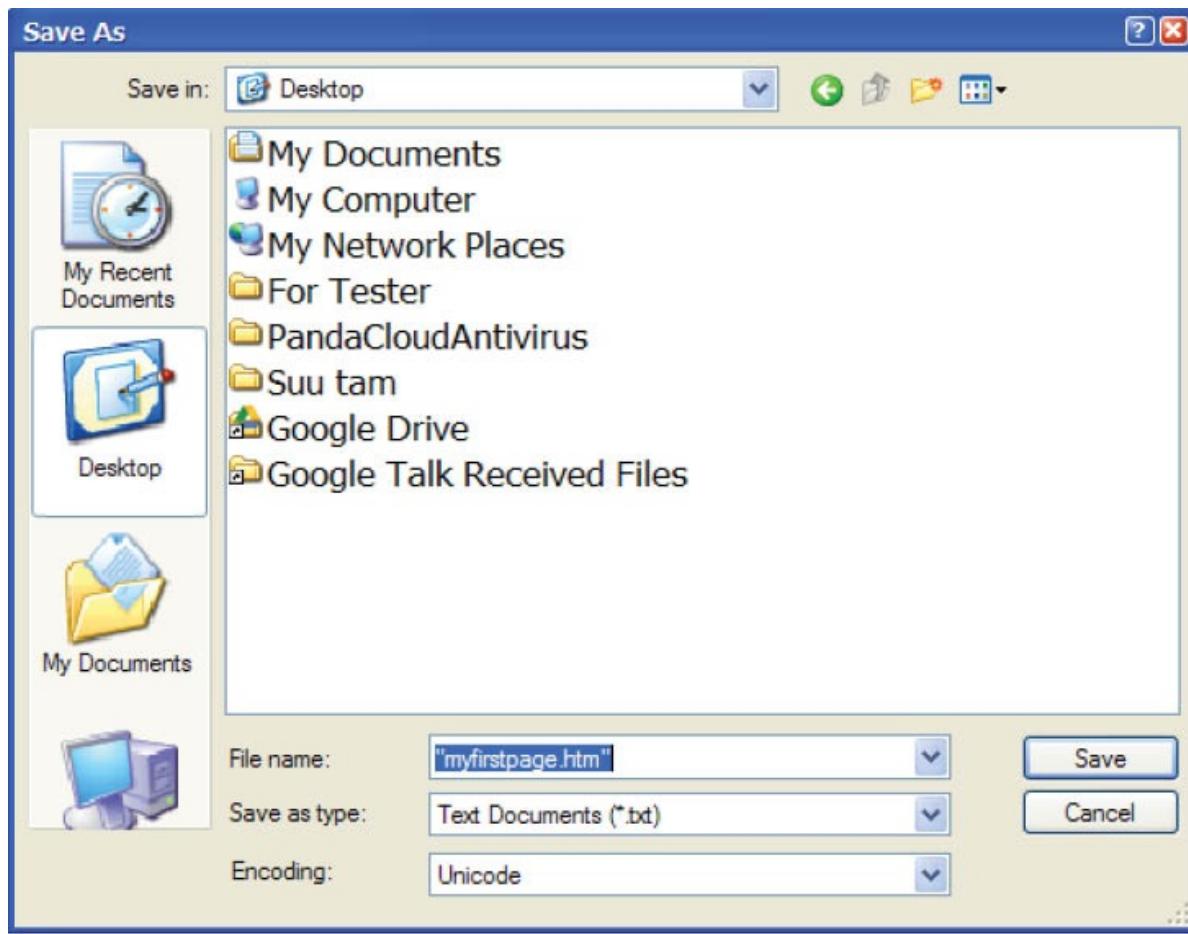
- Trong Microsoft Windows 7 (hay Microsoft Windows XP, Microsoft Windows Vista), mở Notepad bằng cách chọn nút Start, chọn All programs, Accessories, Notepad. Nhập vào đoạn mã sau:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title>Trang đầu tiên của tôi</title>  
<script type="text/javascript">  
function yetAnotherAlert(textToAlert) {  
alert(textToAlert);  
}  
yetAnotherAlert("Đây là Chương 2");  
</script>  
</head>  
<body>  
</body>  
</html>
```



Chú ý Trong trường hợp chỉ xét với mục đích của ví dụ trên, bạn có thể bỏ qua khai báo DOCTYPE nếu không muốn nhập nó. Tuy nhiên với những mục đích khác ngoài quyển sách này, có thể bạn sẽ cần khai báo DOCTYPE. Xem lại Chương 1 về vấn đề này.

- Chọn Save từ menu File. Hộp thoại Save As được mở ra. Notepad sẽ gắn mặc định đuôi .txt cho file vừa tạo trừ khi bạn sử dụng dấu ngoặc kép. Do vậy, hãy nhớ đặt tên file trong dấu ngoặc kép, ví dụ: “myfirstpage.htm”. Nếu không có dấu ngoặc kép, Notepad sẽ chèn đuôi .txt vào file của bạn, lúc đó tên file sẽ trở thành “myfirstpage.htm.txt”. Hình minh họa dưới đây là ví dụ cho việc đặt tên file trong dấu ngoặc kép. Bạn đừng quên vị trí lưu file này.

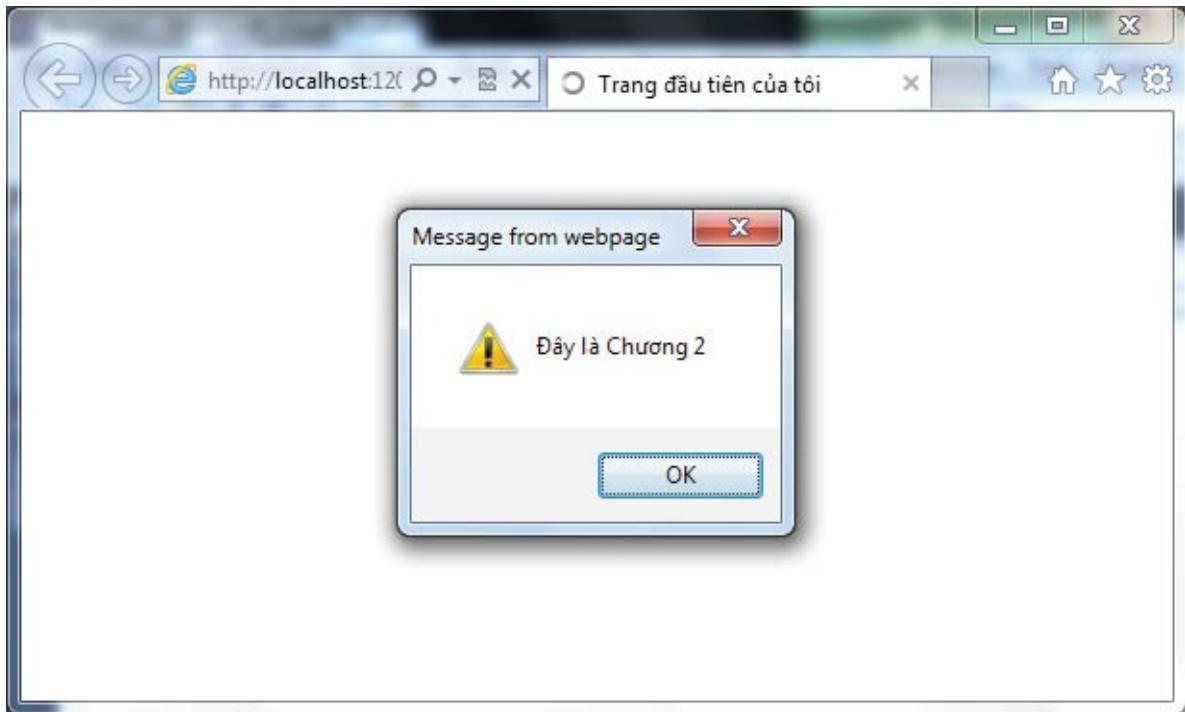


Chú ý Để không bị lỗi hiển thị tiếng Việt trong Notepad, bạn hãy vào menu Format>Font chọn kiểu font, ví dụ Times New Roman. Khi đó bạn có thể gõ các ký tự tiếng Việt mà không bị lỗi font. Sau khi hoàn thành, lưu lại file bằng cách chọn Save As, trong hộp thoại Save As chọn mục Encoding là Unicode.

- Để xem trang html vừa tạo, dùng trình duyệt ưa thích để tìm đến vị trí lưu file và chạy file này. (Nếu như bạn lưu file như hình minh họa trên, hãy tìm đến màn hình Desktop). Hình minh họa tiếp theo cho thấy cách Firefox hiển thị file html.

Chú ý Nếu sử dụng Windows Internet Explorer, bạn có thể nhận được cảnh báo về việc truy cập các nội dung bị chặn tùy thuộc mức độ bảo mật được cấu hình. Hãy truy cập địa chỉ <http://windows.microsoft.com/en-US/windows7/Internet-Explorer-Information-har-frequently-asked-questions> để tìm hiểu thêm về tính

năng này và làm thế nào để tắt nó đi.



Trong ví dụ này, bạn đã tạo được một trang web cơ bản có nhúng thêm JavaScript. Phần JavaScript của trang web chỉ chứa một vài phần tử. Trước tiên, thẻ script được gọi và khai báo ngôn ngữ sử dụng là JavaScript:

```
<script type="text/javascript">
```

Chú ý Bạn có thể khai báo JavaScript bằng cách khác, tuy nhiên đây là cách được hỗ trợ rộng rãi nhất.

Tiếp đó, đoạn mã khai báo hàm *yetAnotherAlert* nhận vào đối số *textToAlert* như sau

```
function yetAnotherAlert(textToAlert) {
```

Nội dung của hàm có nhiệm vụ duy nhất là hiển thị một hộp thoại thông báo trong cửa sổ trình duyệt. Thông báo được hiển thị có thể là bất kỳ đoạn văn bản nào được truyền vào như tham số của hàm. Tất cả được thực hiện bởi lệnh dưới đây:

```
alert(textToAlert);
```

Hàm kết thúc bởi dấu ngoặc đóng ()). Dòng tiếp theo của đoạn mã gọi đến hàm mà bạn vừa khai báo với đối số là một chuỗi đặt trong ngoặc kép như sau:

```
yetAnotherAlert("Đây là Chương 2");
```

Như vậy, qua ví dụ sơ lược trên, bạn đã nắm được cách lập trình JavaScript mà không dùng IDE. Phần tiếp theo sẽ giới thiệu cách thức phổ biến nhất khi sử dụng JavaScript đó là dùng các file chứa mã JavaScript nằm ngoài trang web.

Tạo file JavaScript ngoài không dùng IDE

Lúc này, bạn đã có một trang web (được tạo bằng Notepad) hiển thị một lời thông báo khi được duyệt. Đoạn mã JavaScript thực hiện việc này nằm trong cặp thẻ <head> của trang. Trong phần tiếp theo, bạn sẽ được hướng dẫn cách lưu mã JavaScript vào một external file và cách gọi đến file này trong mã HTML.

Tạo file JavaScript ngoài với Notepad

1. Đầu tiên, mở file myfirstpage.htm. Nếu đang sử dụng Notepad, bạn có thể nhấn chuột phải vào file, chọn Open With, chọn Notepad.
2. Bôi đen đoạn mã JavaScript, ngoại trừ đoạn mã bên trong cặp thẻ <script> </script>. Sao chép đoạn mã đã chọn bằng cách chọn Copy từ menu Edit.
3. Tạo file mới để chứa mã JavaScript bằng cách chọn New từ menu File. File mới tạo được mở ra trong cửa sổ Notepad. Dán đoạn mã JavaScript vào file bằng cách chọn Paste từ menu Edit. Đổi đoạn văn bản là tham số của hàm gọi thành “Đây là ví dụ thứ hai”. Đoạn mã có dạng như sau:

```
function yetAnotherAlert(textToAlert) {  
    alert(textToAlert);  
}  
yetAnotherAlert("Đây là ví dụ thứ hai");
```

4. Lưu file bằng cách chọn Save từ menu File. Nhập **myscript.js** vào trường File name, hãy nhớ đặt tên file trong dấu ngoặc kép, đuôi file phải là .js,

không phải là .txt.

Chú ý Bạn không cần sử dụng đuôi mở rộng của file JavaScript, tuy nhiên nó sẽ giúp bạn dễ dàng nhận diện các file về sau.

- Với mã JavaScript được lưu trong file myscript.js, bạn có thể xóa đoạn mã JavaScript trong file myfirstpage.htm, chỉ để lại thẻ script như dưới:

```
<script type="text/javascript">  
</script>
```

Mách nhỏ Hãy nhớ chọn xem All files thay cho Text Documents khi cần mở các file không có đuôi .txt như đuôi .js hay .htm vừa tạo. Chọn All Files từ menu thả xuống Files Of Type trong hộp thoại Open.

- Thêm thuộc tính *src* vào thẻ mở *<script>*

```
<script type="text/javascript" data-src="myscript.js">
```

Nếu muốn, bạn có thể chuột đoạn mã của mình bằng cách xóa dấu chuyển dòng và đưa thẻ đóng *</script>* lên cùng dòng với thẻ mở.

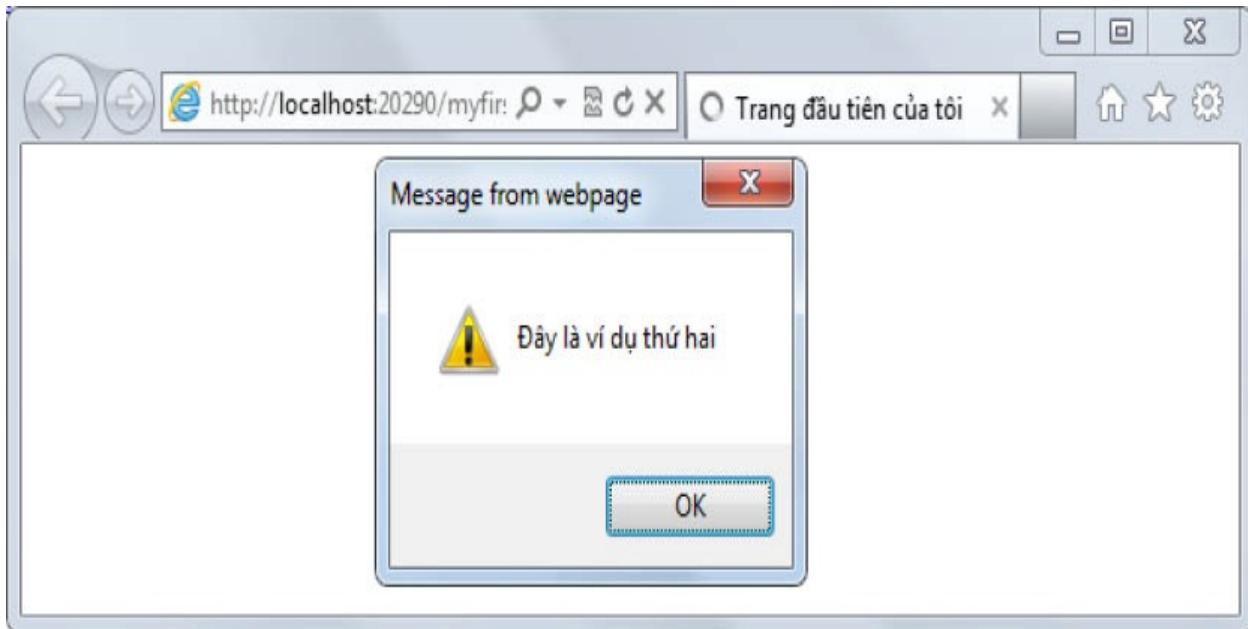
```
<script type="text/javascript" data-src="myscript.js"></sc
```

Lúc này, toàn bộ nội dung của myfirstpage.html sẽ giống như dưới đây:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title>Trang đầu tiên của tôi</title>  
<script type="text/javascript" data-src="myscript.js"></sc  
</head>  
<body>  
</body>  
</html>
```

- Lưu file myfirstpage.htm.
- Dùng trình duyệt mở trang web. Kết quả trả về sẽ là hộp thoại thông báo

“Đây là ví dụ thứ hai”.

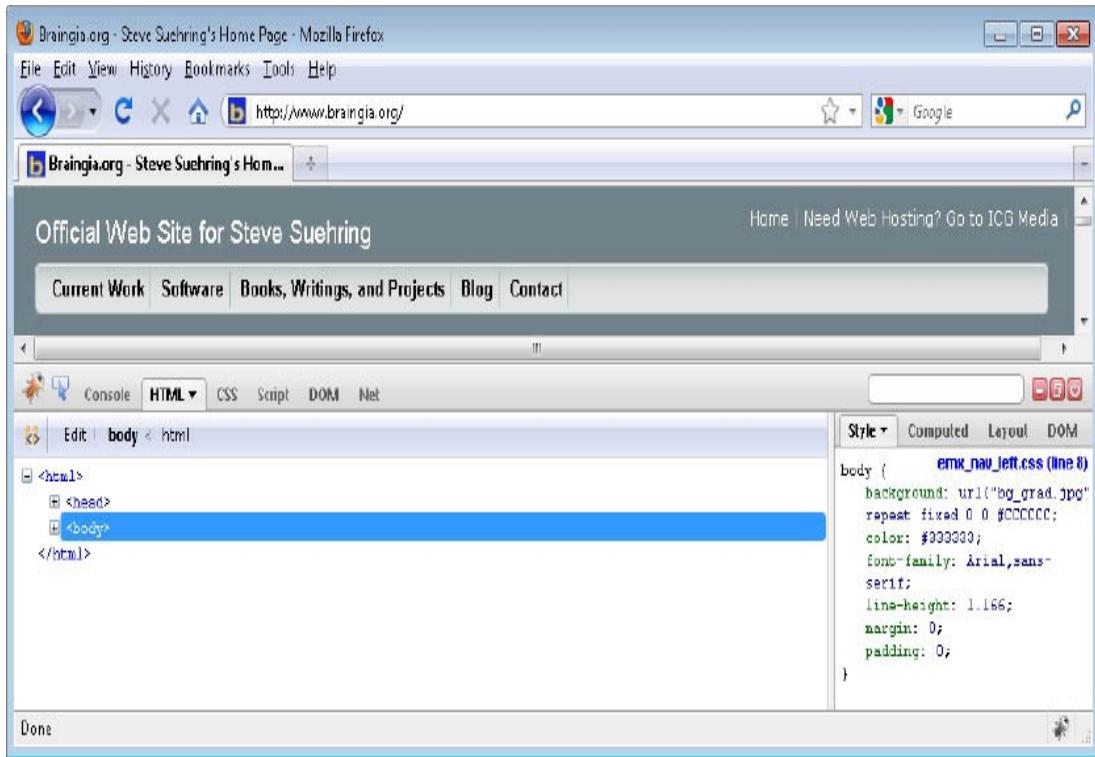


Như vậy, bài học cơ bản về lập trình JavaScript không sử dụng IDE đã kết thúc. Trên đây chỉ là ví dụ với Notepad, một số trình biên soạn khác cũng rất thích hợp cho việc lập trình cơ bản, trong đó có Vim và Textpad của Helio Software Solutions. So với Notepad, hai công cụ này mạnh hơn nhiều.

Gỡ lỗi trong JavaScript

Vìệc gỡ lỗi trong JavaScript có thể gặp nhiều khó khăn, đặc biệt khi chúng ta gỡ lỗi cho các chương trình phức tạp. Một số công cụ, như là Venkman (<http://www.mozilla.org/projects/venkman/>), có thể giúp chúng ta gỡ lỗi JavaScript, tuy nhiên công cụ chính để gỡ lỗi JavaScript lại là trình duyệt. Các trình duyệt phổ biến hiện nay đều cho phép gỡ lỗi JavaScript. Trong các chương trình gỡ lỗi, bạn hãy chú ý đến Firebug, một plug-in nổi tiếng của trình duyệt Firefox. Bạn có thể tải về Firebug tại địa chỉ <http://www.getfirebug.com/>.

Có thể nói Firebug gần như không thể thiếu trong việc lập trình web, đặc biệt là với JavaScript và AJAX. Phần mềm này cho phép bạn kiểm tra mọi thành phần của một trang web, lấy được kết quả của các lời gọi AJAX, xem được CSS, tất cả đều trong thời gian thực khiến cho việc gỡ lỗi trở nên dễ dàng hơn. Hình 2-8 là một ví dụ về dùng Firebug trong gỡ lỗi.



Hình 2-8 Firebug - công cụ không thể thiếu của các lập trình viên web.

Lời khuyên của tác giả là hãy sử dụng Firebug cho việc lập trình và gỡ lỗi JavaScript. Khi gỡ lỗi JavaScript, hàm `alert()` là một hàm rất hữu ích. Chỉ cần một vài lời gọi `alert()` hợp lý, bạn tìm ra giá trị trong các biến và hiểu đoạn mã hiện đang làm gì. Hàm `alert()` sẽ làm xuất hiện hộp thoại thông báo trên trình duyệt, nên nếu bạn để hàm `alert()` trong một vòng lặp và sơ suất để vòng lặp đó chạy vô hạn, bạn sẽ phải thoát khỏi trình duyệt bằng cách bất thường, chẳng hạn sử dụng Task Manager.

Bài tập

1. Hãy tạo trang web có tên `mysecondpage.htm`. Trong phần `<body>` của trang web, tạo đoạn mã làm hiển thị hộp thoại thông báo có nội dung là tên của bạn. Chạy thử đoạn mã trên ít nhất hai trình duyệt khác nhau.
2. Sửa trang web tạo trong Bài tập 1 và tạo một hàm trong phần `<head>` của trang đó, sau đó chuyển đoạn mã làm hiển thị hộp thoại thông báo vào trong hàm mới đó. Gọi hàm từ đoạn mã trong phần `<body>`.

- Chuyển hàm vừa tạo ở Bài tập 2 vào file JavaScript ngoài và tham chiếu hoặc gọi tới file này từ trang web của bạn.



Chương 3

Cú pháp và câu lệnh trong JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Nắm được các nguyên tắc cơ bản khi sử dụng ngôn ngữ lập trình JavaScript.
- Đặt mã JavaScript trong một trang web.
- Nhận biết câu lệnh JavaScript.
- Nhận biết các từ khóa trong JavaScript.

Một vài công việc lặt vặt

Phần còn lại của cuốn sách sẽ đề cập cụ thể hơn về những đặc trưng của JavaScript và làm thế nào để thực hiện các nhiệm vụ cụ thể. Tuy nhiên, bạn phải học đi trước khi học chạy. Vì thế, trước khi tìm hiểu sâu về JavaScript, bạn cần nắm được một số *cấu trúc từ vựng* của nó – đó là các quy tắc hay còn được gọi là *quy tắc cú pháp* của ngôn ngữ này.

Phân biệt chữ hoa chữ thường

JavaScript phân biệt chữ hoa và chữ thường. Cần chú ý điều này khi đặt tên cho biến và sử dụng từ khóa của ngôn ngữ. Một biến được đặt tên là *remote* sẽ khác biến có tên là *Remote* hay *REMOTE*. Tương tự, từ khóa cho vòng lặp là *while*, nhưng nếu viết thành *WHILE* hoặc *Whi1e* thì sẽ gây ra lỗi.

Các từ khóa được viết bằng chữ thường, còn biến có thể kết hợp tùy ý cả chữ hoa và chữ thường miễn là bạn có thể tự kiểm soát được. Ví dụ: Các tên biến được liệt kê dưới đây đều là tên biến hợp lệ trong JavaScript.

```
buttonOne  
txt1  
a  
C
```

Mách nhỏ Bạn sẽ thấy JavaScript thường sử dụng chữ thường trong mã, ngoại trừ một vài trường hợp cần thiết, ví dụ như hàm `isNaN()`, dùng để xác định xem giá trị đầu vào có phải là số hay không (`Nan` trong tên hàm có nghĩa là Not a Number - không phải số). Bạn sẽ học về hàm này trong Chương 4 “Làm việc với biến và kiểu dữ liệu”

Chương 4 sẽ cung cấp thêm kiến thức về biến và quy ước đặt tên, còn bây giờ, bạn hãy chú ý đến chữ hoa, chữ thường khi viết tên biến trong JavaScript.

Ký tự trắng

Trong hầu hết trường hợp, JavaScript bỏ qua ký tự trắng nằm giữa các câu lệnh. Bạn có thể thêm ký tự trắng, lùi đầu dòng hoặc viết mã theo bất cứ quy ước mã nào để đoạn mã JavaScript rõ ràng và dễ đọc hơn. Tuy nhiên, có một vài ngoại lệ trong quy tắc này. Một số từ khóa, như `return` có thể bị thông dịch sai bởi trình thông dịch JavaScript khi được đặt cùng dòng với từ khóa `return` khác. Phần sau của chương sẽ trình bày ví dụ về vấn đề này.

Làm cho chương trình dễ đọc là một lý do để sử dụng ký tự trắng. Ví dụ như đoạn mã dưới đây dùng rất ít ký tự trắng và khoảng lùi đầu dòng:

```
function cubeme(incomingNum) { // Hàm tính lập phương
if (incomingNum == 1) {
return ****Bạn đang làm gì đấy?****;
} else {
return Math.pow(incomingNum, 3);
}
}
var theNum = 2;
var finalNum = cubeme(theNum);
if (isNaN(finalNum)) {
alert("Bạn nên nhớ rằng 1 lũy thừa bao nhiêu vẫn là 1.");
} else {
alert("Lập phương của " + theNum + " là " + finalNum);
}
```

Bây giờ chúng ta xem xét lại đoạn mã trên có sử dụng thêm những khoảng lùi đầu dòng. (Bạn có thể tìm thấy đoạn mã này trong file example1.txt, thư mục mã nguồn mẫu Chương 3 của phần Tài nguyên đi kèm.)

```
function cubeme(incomingNum) { // Hàm tính lập phương
    if (incomingNum == 1) {
        return ****Bạn đang làm gì đấy?****;
    } else {
        return Math.pow(incomingNum, 3);
    }
}

var theNum = 2;
var finalNum = cubeme(theNum);

if (isNaN(finalNum)) {
    alert("Bạn nên nhớ rằng 1 lũy thừa bao nhiêu vẫn là :");
} else {
    alert("Lập phương của " + theNum + " là " + finalNum)
}
```

Đoạn mã thứ hai thực thi chức năng giống như đoạn mã đầu nhưng dễ đọc và dễ theo dõi hơn! Thực tế là có thể bạn chỉ mất một chút thời gian để viết mã, nhưng bạn sẽ phải làm việc với nó trong nhiều năm liền. Như trường hợp của tôi khi đọc lại đoạn mã sau một năm viết, tôi đã rất vui mừng vì trước đó mình đã viết mã một cách dễ đọc và dễ theo dõi.

Chú thích

Với việc viết mã và duy trì chúng trong một thời gian dài, chú thích là người bạn đồng hành tốt. Những đoạn mã trông hết sức rõ ràng nhưng lần tới đọc lại chúng sẽ không được như thế nữa, đặc biệt khi bạn viết từ quá lâu. Chú thích có thể được đặt trong đoạn mã JavaScript theo hai cách: chú thích nhiều dòng và chú thích một dòng.

Bạn sẽ quen với chú thích nhiều dòng trong JavaScript nếu như bạn từng sử dụng ngôn ngữ lập trình C. Chú thích *nhiều dòng* bắt đầu bằng ký tự / và kết thúc bằng ký tự / như dưới đây:

```
/* Đây là chú thích nhiều dòng trong JavaScript
```

Giống như chú thích trong ngôn ngữ C, đoạn chú thích này có

Chú thích *một dòng* bắt đầu với hai dấu gạch chéo (//) và không cần ký tự kết thúc vì nó chỉ ở trong một dòng. Dưới đây là ví dụ:

```
// Đây là chú thích một dòng.
```

Bạn có thể sử dụng nhiều chú thích một dòng. Với những khối chú thích ngắn, bạn có thể sử dụng nhiều chú thích một dòng thay vì dạng chú thích nhiều dòng như đã trình bày ở trên. Ví dụ như, bạn hãy xem khối chú thích dưới đây:

```
// Đây là một khối chú thích khác.  
// Khối chú thích này có nhiều dòng.  
// Trước mỗi dòng có hai dấu gạch chéo.
```

Mách nhỏ Với những chú thích ngắn, chỉ dài một hoặc hai dòng, bạn sẽ thao tác nhanh hơn nếu sử dụng cách chú thích với hai dấu gạch chéo. Với những chú thích dài hơn, ví dụ những chú thích ở đầu chương trình hay đầu đoạn mã, cách chú thích nhiều dòng là lựa chọn tốt hơn vì nó giúp chúng ta có thể dễ dàng thêm hoặc xóa thông tin.



Dấu chấm phẩy

Trong JavaScript, dấu chấm phẩy được sử dụng để phân tách các biểu thức. Về mặt kỹ thuật, dấu chấm phẩy là không bắt buộc trong hầu hết các câu lệnh và biểu thức. Tuy nhiên, những vấn đề khó lường bạn gặp phải khi không sử dụng dấu chấm phẩy có thể gây ra những lỗi không đúng do đó làm tổn thời gian tìm và gỡ lỗi. Trong một số trường hợp, trình thông dịch JavaScript có thể tự động chèn thêm dấu chấm phẩy khi bạn không muốn. Ví dụ như câu lệnh dưới đây:

```
return  
(varName);
```

Thông thường, bạn nên viết là:

```
return(varName);
```

Nhưng JavaScript tự ý thêm một dấu chấm phẩy vào câu lệnh *return*, làm cho đoạn mã trở thành như dưới đây trong trình thông dịch JavaScript:

```
return;  
(varName);
```

Đoạn mã này không hoạt động, trình thông dịch đã hiểu nhầm ý của bạn. Nếu bạn dùng đoạn mã này trong một hàm, nó sẽ trả về giá trị *undefined* (không xác định) cho hàm gọi, và đó không phải là điều bạn mong muốn. Đây là một ví dụ cho thấy chúng ta không nên tùy tiện sử dụng ký tự trắng - bạn cũng không thể sử dụng dấu ngắt dòng (nội dung này sẽ được trình bày cụ thể ở phần kế tiếp) để tách biệt từ khóa *return* và giá trị trả về.

Việc lập trình JavaScript sẽ trở nên dễ dàng hơn nếu bạn sử dụng dấu chấm phẩy như là một quy tắc thay vì phải nhớ xem không cần dùng nó ở đâu.

Tuy nhiên, bạn nhất thiết không được dùng dấu chấm phẩy trong trường hợp sau: sử dụng vòng lặp và các điều kiện:

```
if (a == 4)  
{  
    // đoạn mã được đặt ở đây  
}
```

Trong trường hợp này, bạn sẽ không sử dụng dấu chấm phẩy ở cuối câu lệnh *if* vì câu lệnh hoặc các khối câu lệnh bên trong cặp dấu ngoặc đi sau điều kiện là một phần của câu lệnh điều kiện, trong trường hợp này là câu lệnh *if*. Giả sử bạn đặt dấu chấm phẩy cuối câu lệnh *if*, phần đầu của câu lệnh *if* sẽ bị tách khỏi toàn bộ phần còn lại. Ví dụ, đoạn mã ở dưới đây là sai. (Đoạn mã trong cặp dấu ngoặc nhọn được thực thi bất kể giá trị *a* có bằng 4 hay không).

```
if (a == 4);  
{  
    // đoạn mã được đặt ở đây  
}
```

Mách nhỏ Không sử dụng dấu chấm phẩy khi viết vòng lặp hoặc khai báo hàm.

Dấu ngắt dòng

Liên quan chặt chẽ tới ký tự trắng và thậm chí dấu chấm phẩy trong JavaScript là

dấu ngắt dòng, đôi khi còn được gọi là *ký tự trả về đầu dòng*. Trong chuẩn ECMA-262, dấu ngắt dòng hay còn được gọi là “dấu kết thúc dòng” được dùng để phân biệt một dòng mã với dòng mã tiếp theo. Cũng như dấu chấm phẩy, vị trí đặt dấu ngắt dòng cũng cần phải được lưu ý. Có thể thấy từ ví dụ của phần trước, đặt sai vị trí dấu ngắt dòng có thể dẫn đến các lỗi khó lường.

Không có gì ngạc nhiên khi lợi ích thường gặp nhất của dấu ngắt dòng là để phân biệt các dòng mã khác nhau cho dễ đọc. Bạn cũng có thể làm cho các dòng mã dài dễ hiểu hơn bằng cách xuống dòng. Tuy nhiên, khi làm như vậy, bạn phải cẩn thận với những lỗi như kết quả trả về từ câu lệnh *return* đã nêu ở trên, trong đó dấu ngắt dòng thêm vào có thể gây ra những hiệu ứng không mong muốn, làm thay đổi ý nghĩa của đoạn mã.

Đặt đoạn mã JavaScript đúng vị trí

Bạn có thể đặt mã JavaScript ở nhiều vị trí trong trang HTML: trong phần *<head> </head>*, hoặc giữa cặp thẻ *<body> </body>*. Vị trí phổ biến của đoạn mã JavaScript là ở giữa cặp thẻ *<head> </head>* ở gần đầu trang. Tuy nhiên, cách đặt đoạn mã *<script>* trong phần *<body>* đang trở nên phổ biến hơn. Bạn phải khai báo trước loại script (mã kịch bản) mà bạn sử dụng. Vì đây là cuốn sách về JavaScript nên chúng ta sẽ khai báo như sau trong thẻ *<script>*:

```
<script type="text/javascript">
```

Một vấn đề quan trọng cần lưu ý khi bạn sử dụng JavaScript với các trang được khai báo dạng XHTML là trong các trang này, thẻ *<script>* cần phải được khai báo trong phần CDATA. Nếu không, XHTML sẽ phân tích thẻ *<script>* như một thẻ XML và đoạn mã trong đó có thể không chạy như bạn mong đợi. Do đó, để sử dụng JavaScript trong XHTML cần khai báo như sau:

```
<script type="text/javascript">
<![CDATA[
    // Đoạn mã JavaScript được đặt ở đây
]]>
</script>
```

Các phiên bản trình duyệt cũ có thể không phân tích chính xác CDATA. Vấn đề này có thể được giải quyết bằng cách đặt dòng mở đầu và kết thúc của CDATA bên trong dấu chú thích của JavaScript như ví dụ dưới đây:

```
<script type="text/javascript">  
//<![CDATA[  
    // Đoạn mã JavaScript được đặt ở đây  
]]>  
</script>
```

Khi đặt đoạn mã JavaScript trong một file ngoài (như bạn đã đọc ở Chương 2 “Lập trình với JavaScript”), bạn không cần dùng phần CDATA nữa. Bạn sẽ thấy rằng ngoại trừ những đoạn JavaScript ngắn nhất, sẽ tốt hơn nếu bạn định nghĩa JavaScript trong các file ngoài - thường là với đuôi mở rộng .js - và sau đó liên kết các file trong trang. Chương 2 đã mô tả cụ thể vấn đề này, nhưng phần này sẽ nói đến cách thức liên kết đến một file sử dụng thuộc tính *src* trong thẻ *<script>*:

```
<script type="text/javascript" data-src="myscript.js">
```

Đặt đoạn JavaScript vào file ngoài có nhiều lợi ích, ví dụ như:

- **Phân tách đoạn mã ra khỏi các thẻ đánh dấu:** Lưu mã JavaScript vào file ngoài khiến việc bảo trì đoạn mã HTML dễ hơn và giúp giữ cấu trúc của HTML mà không phải sử dụng phần CDATA cho XHTML.
- **Dễ bảo trì:** Với việc đặt mã JavaScript vào một file ngoài, bạn có thể thay đổi file đó mà không làm ảnh hưởng đến các file khác của website.
- **Lưu trữ tạm thời (Caching):** Nếu sử dụng file JavaScript ngoài, trình duyệt web có thể lưu trữ tạm thời file đó và nhờ thế làm tăng tốc độ tải trang web cho người dùng.

Câu lệnh trong JavaScript

Cũng như các chương trình được viết bằng ngôn ngữ khác, chương trình được viết bằng JavaScript cũng bao gồm các câu lệnh được sắp xếp để trình thông dịch JavaScript có thể thực hiện một hay nhiều tác vụ. Và cũng như các ngôn ngữ khác, câu lệnh trong JavaScript có thể đơn giản hoặc phức tạp. Phần này sẽ mô tả ngắn gọn mẫu câu lệnh JavaScript, với giả định rằng bạn đã xem qua một số ví dụ ở các chương trước cũng như ở các chương sau của cuốn sách.

Câu lệnh JavaScript có những phần tử gì?

Như đã trình bày trong Chương 1 “Hiểu hơn về JavaScript”, câu lệnh JavaScript, hay biểu thức, là một tập hợp các từ khóa, toán tử hoặc chuỗi định danh được đặt với nhau để tạo thành một chương trình mà trình thông dịch JavaScript có thể hiểu được. Câu lệnh thường kết thúc với dấu chấm phẩy, ngoại trừ trong một vài trường hợp đặc biệt như lệnh khai báo cấu trúc rẽ nhánh và vòng lặp như *if*, *while*, và *for* được trình bày cụ thể hơn trong Chương 5 “Sử dụng toán tử và biểu thức”.

Dưới đây là một vài ví dụ về câu lệnh cơ bản trong JavaScript:

```
var x = 4;  
var y = x * 4;  
alert("Xin chào");
```

Hai loại câu lệnh JavaScript

Câu lệnh JavaScript có hai loại cơ bản: đơn và phức hợp. Bạn không cần mất nhiều thời gian với khái niệm câu lệnh. Tuy vậy, bạn nên hiểu sự khác nhau giữa câu lệnh đơn và câu lệnh phức hợp.

Câu lệnh đơn là một câu lệnh như dưới đây:

```
var x = 4;
```

Câu lệnh phức hợp kết hợp nhiều cấp độ logic với nhau. Một câu lệnh điều kiện *if/then/else* như dưới đây là ví dụ điển hình cho câu lệnh phức hợp:

```
if (something == 1) {  
    // đoạn mã đặt ở đây  
} else {  
    // đoạn mã khác được đặt ở đây  
}
```

Từ khóa trong JavaScript

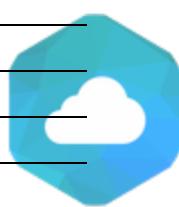
Một số từ trong JavaScript là *từ khóa*, nghĩa là bạn không thể sử dụng nó làm biến, chuỗi định danh hoặc hằng số trong chương trình của bạn. Nếu bạn làm vậy, đoạn mã sẽ gặp những kết quả không mong đợi như bị lỗi. Ví dụ, bạn đã thấy từ khóa `var` trong một vài ví dụ trước đây. Sử dụng từ khóa `var` ngoài mục đích để khai báo biến có thể gây ra lỗi hoặc các hiệu ứng không mong muốn khác tùy vào trình duyệt. Ví dụ như câu lệnh dưới đây:

```
// Không được viết như thế này!  
var var = 4;
```

Đoạn mã ví dụ này không dẫn đến lỗi trực tiếp trên trình duyệt, nhưng nó sẽ không hoạt động như bạn mong muốn.

Danh sách dưới đây bao gồm những từ được dùng làm từ khóa theo chuẩn ECMA-262:

| | | | | |
|----------|----------|------------|--------|-------|
| break | delete | if | this | while |
| case | do | in | throw | with |
| catch | else | instanceof | try | |
| continue | finally | new | typeof | |
| debugger | for | return | var | |
| default | function | switch | void | |



Một số từ khóa khác (được liệt kê ở danh sách dưới đây) được dành cho các mục đích sử dụng trong tương lai và do đó bạn không nên sử dụng chúng trong chương trình của mình.

| | | | |
|-------|--------|---------|-------|
| class | enum | extends | super |
| const | export | import | |

Danh sách các từ khóa dưới đây được dành để sử dụng trong chế độ strict:

| | | | | |
|------------|---------|-----------|--------|-------|
| implements | let | private | public | yield |
| interface | package | protected | static | |

Sơ lược về hàm

Bạn đã được xem nhiều ví dụ về hàm ở các chương trước. JavaScript có nhiều

hàm có sẵn hoặc hàm do bản thân ngôn ngữ này định nghĩa. Chúng ta đã đề cập đến hàm `alert()` ngoài ra còn nhiều hàm khác nữa. Tùy theo phiên bản ngôn ngữ mà bạn đang sử dụng, sẽ có các hàm có sẵn khác nhau. Nhiều hàm chỉ có ở các phiên bản gần đây của JavaScript – những phiên bản chỉ được hỗ trợ bởi một số trình duyệt. Tìm ra các hàm (và các đối tượng) có sẵn trên trình duyệt là một công việc quan trọng để xác định xem trình duyệt của người dùng có thể chạy mã JavaScript mà bạn tạo trên trang web của mình không. Nội dung này sẽ được đề cập ở Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt”.

Mách nhỏ Một nguồn thông tin rất hữu ích về sự tương thích trình duyệt là trang web QuirksMode: (<http://www.quirksmode.org/compatibility.html>).

Tương tự các ngôn ngữ lập trình khác, JavaScript cũng chấp nhận các hàm người dùng định nghĩa. Ví dụ trước đây trong chương này đã định nghĩa một hàm có tên `cubeme()` để tính lập phương của một số. Đoạn mã đó cũng cho thấy cách dùng JavaScript ở cả thẻ `<head>` và `<body>` của trang web.

Đặt hàm người dùng định nghĩa trong JavaScript

1. Dùng Visual Studio, Eclipse, hoặc một trình soạn thảo khác, thay đổi file `example1.htm` trong thư mục mã nguồn mẫu của Chương 3.
2. Trong trang web, thay thế dòng chú thích TODO bằng đoạn mã in đậm dưới đây:

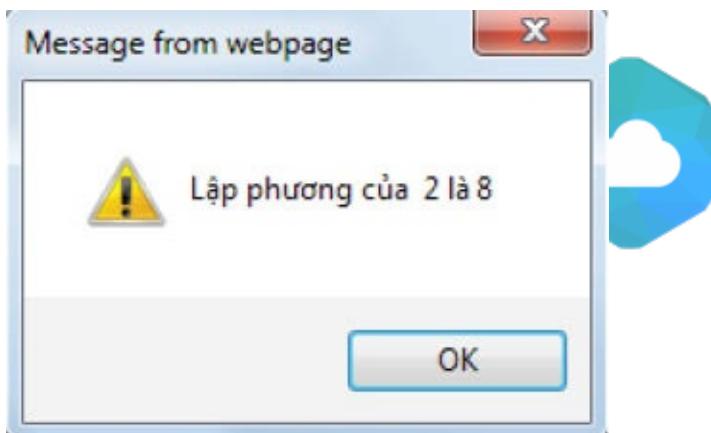
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script type="text/javascript">
function cubeme(incomingNum) {
if (incomingNum == 1) {
return "Bạn đang làm gì đấy?";
} else {
return Math.pow(incomingNum, 3);
}
}
</script>
```

```

<title>Ví dụ Chương 3</title>
</head>
<body>
<script type="text/javascript">
var theNum = 2;
var finalNum = cubeme(theNum);
if (isNaN(finalNum)) {
    alert("Bạn nên nhớ rằng 1 lũy thừa bao nhiêu vẫn là 1.");
} else {
    alert("Lập phương của " + theNum + " là " + finalNum);
}
</script>
</body>
</html>

```

3. Lưu file này lại, sau đó chạy đoạn mã hoặc sử dụng trình duyệt để mở trang web. Bạn sẽ thấy một thông báo như sau:



Đoạn mã trong ví dụ này kết hợp đoạn mã từ ví dụ trước đó để tạo thành một trang HTML hoàn chỉnh, bao gồm cả khai báo DOCTYPE. Đoạn mã này khai báo một hàm là *cubeme()* trong thẻ *<head>* của tài liệu, như sau:

```

function cubeme(incomingNum) {
    if (incomingNum == 1) {
        return "Bạn đang làm gì đấy?";
    } else {
        return Math.pow(incomingNum, 3);
    }
}

```

Hàm này chấp nhận một đối số là *incomingNum*. Câu lệnh điều kiện *if/then*

là trọng tâm của hàm. Khi *incomingNum* bằng 1, hàm sẽ trả về chuỗi ký tự “Bạn đang làm gì đấy?”. Và ngược lại, khi *incomingNum* khác 1, phương thức *Math.pow* được gọi với *incomingNum* và số nguyên 3 là đối số. Việc gọi hàm *Math.pow* đã tăng giá trị của *incomingNum* lên lũy thừa bậc 3 và giá trị này được trả về hàm gọi. Cách thức gọi hàm sẽ được đề cập tiếp ở Chương 4.

Toàn bộ đoạn mã trước đều được đặt trong thẻ *<head>* của tài liệu, do đó nó có thể được các đoạn mã khác gọi đến; đây chính là việc mà chúng ta sắp thực hiện. Trình duyệt khi đó sẽ hiển thị phần *<body>* của tài liệu, bao gồm một vài đoạn mã JavaScript khác. Và đoạn mã tiếp theo này sẽ gán cho biến *theNum* giá trị bằng 2.

```
var theNum = 2;
```

Sau đó, đoạn mã sẽ gọi hàm được định nghĩa trước đó là *cubeme()*, sử dụng biến *theNum* làm đối số. Cần chú ý rằng biến *finalNum* sẽ nhận giá trị trả về từ hàm *cubeme()*, như dưới đây:

```
var finalNum = cubeme(theNum);
```

Đoạn mã JavaScript cuối cùng trong trang là tập hợp các điều kiện *if/then*. Đoạn mã này kiểm tra để xác định xem giá trị trả về được lưu trong biến *finalNum* có là số hay không, bằng cách sử dụng hàm *isNaN()*. Nếu giá trị trả về không phải là số, hộp thoại thông báo sẽ được hiển thị với nội dung cho thấy 1 đã được dùng làm đối số (Tất nhiên có nhiều lý do khiến giá trị trả về không phải là số, tuy nhiên chúng ta hãy tạm thời chấp nhận ví dụ đã nêu). Nếu như giá trị trả về là số, số đó được hiển thị, và bạn có thể thấy ở hộp thoại của hàm *alert()* hiển thị giá trị như ở bước 3 bên trên.

Chế độ strict của JavaScript

ECMA-262 phiên bản 5 giới thiệu một biến thể nghiêm ngặt, thường được gọi là *chế độ strict*, trong đó có cải tiến chức năng kiểm tra lỗi và bảo mật. Ví dụ, để chặn nguy cơ gõ nhầm tên biến, việc khai báo biến bắt buộc phải sử dụng từ khóa *var*.Thêm vào đó, những thay đổi hàm *eval()* và các khía cạnh khác cũng giúp chúng ta nâng cao chất lượng mã JavaScript.

Chế độ strict được kích hoạt bằng cú pháp rất giống với cú pháp dùng trong Perl:

```
"use strict";
```

Chế độ strict có phạm vi cục bộ tại khối mà nó được sử dụng vì thế bạn có thể kích hoạt toàn cục bằng cách đặt dòng `use strict` ở đầu đoạn mã JavaScript hoặc bạn có thể kích hoạt nó trong một hàm, bằng cách đặt dòng `use strict` trong chính hàm đó ví dụ như:

```
function doSomething() {  
    "use strict";  
    //Đoạn mã của hàm được đặt ở đây.  
}
```

Một cải tiến khác trong chế độ strict có thể bắt được lỗi đánh máy là tính năng ngăn chặn các biến chưa được khai báo. Tất cả các biến trong chế độ strict cần được khởi tạo trước khi sử dụng. Ví dụ như câu lệnh dưới đây:

```
"use strict";  
x = 4; // Tạo ra lỗi cú pháp
```

Đoạn mã này sẽ gây lỗi vì biến `x` vẫn chưa được khai báo trước với từ khóa `var`, như dưới đây:



```
"use strict";  
var x = 4; // Cú pháp này là đúng.
```

Một cải tiến đáng chú ý về bảo mật mà chế độ strict mang lại là thay đổi cách thức vận dụng hàm `eval()`. Trong chế độ strict, `eval()` không thể khởi tạo một biến hoặc hàm mới để dùng bên ngoài câu lệnh `eval()`. Ví dụ:

```
"use strict";  
eval("var testVar = 2;");  
alert(testVar); // Tạo ra lỗi cú pháp.
```

Trong đoạn mã ví dụ, sẽ có một lỗi cú pháp vì chế độ strict được kích hoạt làm cho biến `testVar` không truy xuất được bên ngoài câu lệnh `eval()`.

Chế độ strict cũng ngăn chặn việc trùng tên biến trong một đối tượng hoặc một lời gọi hàm:

```
"use strict";  
var myObject ={  
    testVar: 1,  
    testVar: 2
```

};

Đoạn mã trên sẽ gây ra lỗi ở chế độ strict vì biến *testVar* được thiết lập hai lần trong đoạn mã định nghĩa đối tượng.

Cũng như các tính năng khác của ECMA-262 phiên bản 5, chế độ strict có thể không khả dụng ở tất cả các trình duyệt và nhiều khả năng không chạy được trên các trình duyệt cũ. Bạn có thể tìm hiểu thêm thông tin về ECMA-262 phiên bản 5 và ví dụ đầy đủ tại đây: <http://besen.sourceforge.net/>.

Bài tập

1. Câu lệnh JavaScript nào dưới đây hợp lệ? (Chọn tất cả các đáp án có thể)

- a. if (var == 4) { // Viết mã ở đây }
- b. var testVar = 10;
- c. if (a == b) { // Viết mã ở đây }
- d. testVar = 10;
- e. var case = "Yes";



2. Đúng hay sai: Bạn bắt buộc phải dùng dấu chấm phẩy để kết thúc toàn bộ các câu lệnh JavaScript?

3. Kiểm tra đoạn JavaScript dưới đây. Kết quả sẽ như thế nào? (Giả định rằng khai báo JavaScript đã có và đoạn mã này được đặt hợp lệ trong phần *<head>* của trang).

```
var orderTotal = 0;
function collectOrder(numOrdered) {
    if (numOrdered > 0) {
        alert("Bạn đã đặt " + orderTotal);
        orderTotal = numOrdered * 5;
    }
    return orderTotal;
}
```

Chương 4

Làm việc với biến và kiểu dữ liệu

Sau khi đọc xong chương này, bạn có thể:

- Hiểu về các kiểu dữ liệu cơ sở trong JavaScript.
- Sử dụng các hàm liên quan với các kiểu dữ liệu.
- Tạo các biến.
- Định nghĩa các đối tượng và mảng.
- Hiểu về phạm vi của biến.
- Gỡ lỗi JavaScript bằng Firebug.

Kiểu dữ liệu trong JavaScript



Kiểu dữ liệu của một ngôn ngữ xác định những thành phần cơ bản có thể được sử dụng trong ngôn ngữ đó. Bạn có thể đã quen thuộc với các kiểu dữ liệu như kiểu chuỗi hoặc số nguyên trong các ngôn ngữ khác. JavaScript có từ ba tới sáu kiểu dữ liệu tùy thuộc vào định nghĩa về kiểu dữ liệu. Bạn sẽ thường xuyên làm việc với tất cả các kiểu dữ liệu này tuy nhiên một số kiểu có thể được dùng nhiều hơn các kiểu khác.

Sáu kiểu dữ liệu trong JavaScript được thảo luận trong chương này bao gồm:

- Kiểu số
- Kiểu chuỗi
- Kiểu logic
- Kiểu null
- Kiểu undefined (Không xác định)
- Kiểu đối tượng

Ba kiểu dữ liệu đầu tiên – kiểu số, chuỗi và logic – có lẽ đã trở nên quen thuộc với tất cả các lập trình viên. Ba kiểu dữ liệu sau – kiểu null, undefined và đối tượng – cần được giải thích thêm. Ngay sau đây, chúng ta sẽ xem xét từng kiểu dữ liệu và sẽ tìm hiểu kỹ hơn về kiểu đối tượng trong Chương 8 “Các đối tượng trong JavaScript”.

Bên cạnh đó, JavaScript còn có một vài kiểu dữ liệu tham chiếu khác như kiểu *Array*, *Date* và *RegExp*. Kiểu *Date* (*ngày tháng*) và *RegExp* (*bíểu thức chính quy*) được thảo luận trong chương này, còn kiểu *Array* (*mảng*) sẽ được bàn tới trong Chương 8.

Làm việc với kiểu dữ liệu số

Kiểu dữ liệu số trong JavaScript đơn giản là những con số. Tuy nhiên, những lập trình viên đã quen với kiểu dữ liệu trong các ngôn ngữ như C sẽ thấy ngạc nhiên vì số nguyên và số thực dấu chấm động trong JavaScript không được tách riêng. Cả hai đều là kiểu số hợp lệ trong JavaScript. Ví dụ:

4
51.50
-14
0xd



Ví dụ cuối cùng, *0xd* là một số nguyên hệ thập lục phân (hệ cơ số 16). Cả số hệ thập lục phân và hệ bát phân (hệ cơ số 8) đều hợp lệ trong JavaScript và bạn sẽ không ngạc nhiên khi biết rằng JavaScript chấp nhận việc thực hiện phép toán trên các số thuộc những định dạng này. Hãy thử làm bài tập sau.

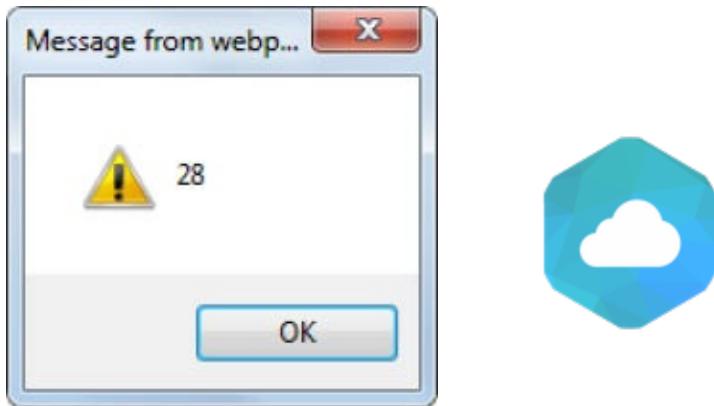
Thực hiện tính toán trên hệ cơ số 16 với JavaScript

1. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo khác, chỉnh sửa file example.htm trong thư mục mã nguồn mẫu của Chương 4 trong phần Tài nguyên đi kèm.
2. Trong trang này, thay thế chú thích TODO bằng đoạn mã in đậm sau (bạn có thể tìm thấy đoạn mã này trong file example1.txt của phần Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
<head>
<title> Số dạng thập lục phân </title>
<script type="text/javascript">
var h = 0xe;
var i = 0x2;
var j = h * i;
alert(j);
</script>
</head>
<body>
</body>
</html>
```

3. Dùng trình duyệt mở trang web. Bạn sẽ thấy một hộp thoại như sau:



Đoạn mã trên khai báo hai biến (bạn sẽ tìm hiểu về khai báo biến trong phần sau của chương này) và thiết lập cho chúng giá trị thập lục phân tương ứng là 0xe (14 trong hệ thập phân) và 0x2 (2 trong hệ thập phân).

```
var h = 0xe;
var i = 0x2;
```

Sau đó, khai báo một biến mới và gán cho nó giá trị bằng tích của hai biến trước như sau:

```
var j = h * i;
```

Biến kết quả sau đó được truyền vào hàm *alert()* và hiển thị ra hộp thoại ở bước 3. Thật thú vị khi thực hiện nhân hai giá trị thập lục phân mà kết quả nhận được trên hộp thoại lại là một giá trị thập phân.

Các hàm số học

JavaScript tích hợp sẵn các hàm (và đối tượng) để làm việc với các giá trị số. Tiêu chuẩn (ECMA) của Hiệp hội Sản xuất Máy tính châu Âu đã định nghĩa rất nhiều hàm. Một trong số những hàm phổ biến là `isNaN()`.

NaN là viết tắt của cụm từ *Not a Number*, có nghĩa là “không phải số hợp lệ”. Bạn sử dụng hàm `isNaN()` để xác định một số có hợp lệ theo đặc tả ECMA-262 hay không. Ví dụ, kết quả của một phép chia cho 0 không phải là số hợp lệ trong JavaScript. Chuỗi giá trị “Đây không phải là một số” đương nhiên không phải là số. Mặc dù mỗi người lại có cách hiểu khác nhau để phân biệt đâu là số và ngược lại. Theo hàm `isNaN()`, chuỗi ký tự “bốn” không phải là số, tuy nhiên chuỗi “4” thì lại là số. Hàm `isNaN()` luôn cố gắng chứng minh giá trị của một biến *không phải là số*. Dưới đây là một số ví dụ bạn có thể thử để xem một giá trị như thế nào là số không hợp lệ

Thực hành với hàm `isNaN()`

1. Mở trình duyệt IE.
2. Trên thanh địa chỉ, nhập dòng lệnh sau (có trong file isnan.txt của phần Tài nguyên đi kèm):

```
javascript:alert(isNaN("4"));
```



Bạn nhận được một hộp thoại với thông báo False như hình sau:



Hàm `isNaN()` trả về `false` cho biểu thức trên vì giá trị nguyên 4 là một số. Nhớ rằng đoạn mã trên trả lời cho câu hỏi “Có đúng 4 không phải là số?”. Vì 4 đúng

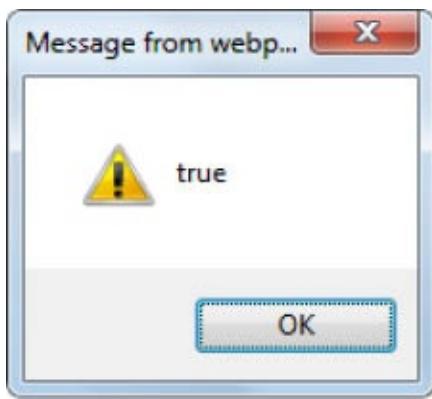
là số, do đó kết quả trả về là false (sai).

Bây giờ, hãy xem xét ví dụ sau:

1. Mở trình IE.
2. Trên thanh địa chỉ, nhập:

```
javascript:alert(isNaN("bốn"));
```

Bạn nhận được một hộp thoại với thông báo true (đúng) như sau:



Trong trường hợp này, vì chuỗi ký tự *bốn* không phải là số, nên hàm trả về *true*: chuỗi *bốn* không phải là số. Việc cẩn ý sử dụng dấu nháy kép trong mỗi ví dụ ("4" và "bốn") nhằm chỉ ra rằng với hàm này, dấu nháy kép không gây ra lỗi. Vì JavaScript đủ thông minh để nhận ra "4" là số, nó sẽ tự chuyển đổi kiểu cho bạn. Mặc dù vậy, sự chuyển đổi này đôi khi gây bất tiện, chẳng hạn như khi bạn làm việc với biến hoặc giá trị thuộc một kiểu nhất định.

Hàm *isNaN()* thường được dùng để kiểm tra xem dữ liệu đầu vào - có thể là từ form - thuộc dạng số hay ký tự.

Các hằng số số học

Nhiều hằng số số học có sẵn trong JavaScript, một số được mô tả ở Bảng 4-1. Các hằng số này luôn sẵn dùng khi cần.

Bảng 4-1 Các hằng số số học chọn lọc.

|Hằng số|Miêu tả|

| - |

|*Infinity*|Thể hiện giá trị dương vô cực.|

|*Number.MAX_VALUE*|Số lớn nhất có thể được biểu diễn trong JavaScript.|

|*Number.MIN_VALUE*|Số nhỏ nhất có thể được biểu diễn trong JavaScript.|

|*Number.NEGATIVE_INFINITY*|Giá trị âm vô cực.|

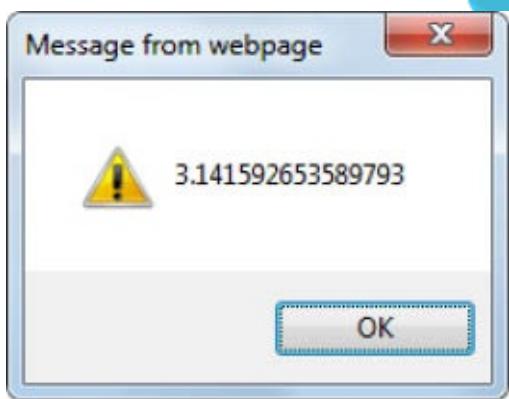
|*Number.POSITIVE_INFINITY*|Giá trị dương vô cực.|

Đối tượng Math

Math là đối tượng đặc biệt có sẵn để làm việc với các số trong JavaScript. Một số thuộc tính hữu ích của *Math* khi lập trình JavaScript bao gồm thuộc tính trả về giá trị của số pi, căn bậc 2 của một số, số giả ngẫu nhiên (pseudo-random) và giá trị tuyệt đối. Một số là thuộc tính giá trị, nghĩa là nó trả về giá trị, số khác hoạt động như các hàm và trả về giá trị thông qua đối số truyền vào. Hãy xem một ví dụ về thuộc tính giá trị *Math.PI*:

```
javascript:alert(Math.PI);
```

Kết quả được thể hiện như Hình 4-1.



Hình 4-1 Xem giá trị của _thuộc tính Math.PI

Ký hiệu dấu chấm

Ký hiệu dấu chấm đơn hay *dot* được dùng để truy xuất các thành phần của một đối tượng. Dấu chấm (.) phân tách các thành phần. Ví dụ, để truy xuất thuộc tính “chiều dài của biến *room*”, bạn có thể viết là `room.length_. Toán tử (.) được sử dụng tương tự nhau trong nhiều ngôn ngữ lập trình.

Đối tượng *Math* còn rất nhiều thuộc tính khác có thể hữu ích cho chương trình của bạn. Một vài trong số đó hoạt động như hàm hoặc phương thức trên đối tượng và được liệt kê trong Bảng 4-2. Bạn có thể xem đầy đủ các thuộc tính của đối tượng *Math* trong đặc tả ECMA-262 tại <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.

BẢNG 4-2 Các thuộc tính của đối tượng *Math*.

| Hằng số | Miêu tả |
|----------------------|----------------------------------|
| <i>Math.random()</i> | Trả về một số ngẫu nhiên. |
| <i>Math.abs(x)</i> | Trả về giá trị tuyệt đối của x. |
| <i>Math.pow(x,y)</i> | Trả về giá trị x^y . |
| <i>Math.round(x)</i> | Làm tròn x tới giá trị gần nhất. |

Làm việc với chuỗi

Chuỗi là một kiểu dữ liệu cơ bản khác trong JavaScript. Chuỗi bao gồm một hoặc nhiều ký tự nằm trong dấu nháy. Các ví dụ sau là chuỗi:

- "*Hello world*"
- "B"
- "*Đây là 'một chuỗi khác'*"



Cần giải thích thêm về ví dụ cuối trong danh sách trên. Chuỗi được bao quanh bởi dấu nháy đơn hoặc kép. Chuỗi nằm trong dấu nháy đơn có thể chứa các dấu nháy kép. Tương tự, một chuỗi nằm trong dấu nháy kép, như bạn thấy trong ví dụ trên, có thể chứa các dấu nháy đơn. Vì vậy, nếu chuỗi được bao quanh bởi một loại dấu nháy, bạn có thể sử dụng dấu nháy khác bên trong nó. Sau đây là một số ví dụ:

- 'Bò kêu "moo".'
- 'Đồng hồ thông báo "12h trưa".'
- ""Ai cũng có khoảng thời gian tuyệt vời' là khẩu hiệu chính thức."

Thoát ký tự nháy

Nếu bạn sử dụng cùng một loại dấu nháy (nằm ở trong chuỗi, và bao quanh

chuỗi) và muốn dấu nháy nằm ở trong chuỗi hiển thị như một ký tự bình thường chứ không phải là một phần của cú pháp thông báo cho JavaScript biết là hết chuỗi bạn có thể dùng ký tự số chéo ngược (backslash) đặt ngay trước dấu nháy. Ví dụ:

- 'Tôi đang sử dụng 'dấu nháy đơn' cả trong và ngoài ví dụ này.'
- "Đây là ví dụ "tuyệt vời" về việc sử dụng "dấu nháy kép" trong một chuỗi"

Các ký tự thoát khác

JavaScript cho phép kết hợp ký tự số chéo ngược với một số ký tự thông thường tạo thành chuỗi ký tự thoát để biểu diễn một số ký tự đặc biệt. Bảng 4-3 trình bày những chuỗi ký tự thoát này.

BẢNG 4-3 Chuỗi ký tự thoát trong JavaScript.

| Chuỗi ký tự thoát | Giá trị hiển thị |
|-------------------|-------------------|
| b | Dấu xóa |
| t | Dấu tab |
| n | Xuống dòng |
| v | Dấu tab dọc |
| f | Sang trang |
| r | Trở về đầu dòng |
| \ | Dấu số chéo ngược |



Sau đây là ví dụ về cách dùng một số chuỗi ký tự thoát (Xin thứ lỗi trước vì tôi tiếp tục sử dụng hàm `alert()`). Tôi sẽ sớm dùng thêm các cách phức tạp hơn để hiển thị kết quả trả về).

Sử dụng các chuỗi ký tự thoát

1. Mở trình duyệt IE.
2. Trên thanh địa chỉ, gõ dòng lệnh sau (bạn cũng có thể thấy dòng lệnh này trong file `escapesequences.txt` của phần Tài nguyên đi kèm):

```
javascript:alert("xin chào tnxin chàoontạm biệt");
```

Hộp thoại sau xuất hiện (nếu hộp thoại không xuất hiện, hãy đóng và mở lại

trình duyệt). Chú ý rằng trên một số trình duyệt, chẳng hạn Chrome, cách sử dụng ký tự tab như trong ví dụ trên là không chính quy.



Đây là cách dùng chuỗi ký tự thoát trong thực tế. Trong đoạn mã, hộp thoại hiển thị hai từ “xin chào” bao quanh là hai dấu tab, biểu diễn bởi chuỗi ký tự thoát *t*, sau đó là một ký tự xuống dòng biểu diễn bởi *n*, cuối cùng là từ “tạm biệt”.

Các phương thức và thuộc tính của chuỗi

JavaScript định nghĩa một số thuộc tính và phương thức để làm việc với chuỗi. Các thuộc tính và phương thức này được truy cập bằng dấu chấm (“.”) đã được trình bày ở phần trước của chương này và cũng đã quen thuộc với nhiều lập trình viên.

Chú ý Như đã thống nhất, cuốn sách này chỉ liệt kê một số thành phần trong JavaScript, một số thuộc tính và phương thức chính của kiểu chuỗi trong đặc tả ECMA-262. Tham khảo đặc tả này để có thông tin đầy đủ hơn về các phương thức và thuộc tính của chuỗi.

Thuộc tính *length* trong đối tượng *string* đại diện cho chiều dài của chuỗi, không tính dấu nháy kép kèm theo. Thuộc tính này có thể được gọi trực tiếp trên chuỗi như trong ví dụ sau:

```
alert("Đây là một chuỗi.".length);
```

Tuy nhiên, cách dùng phổ biến hơn là gọi thuộc tính *length* trên một biến như sau:

```
var x = "Đây là một chuỗi.";
alert(x.length);
```

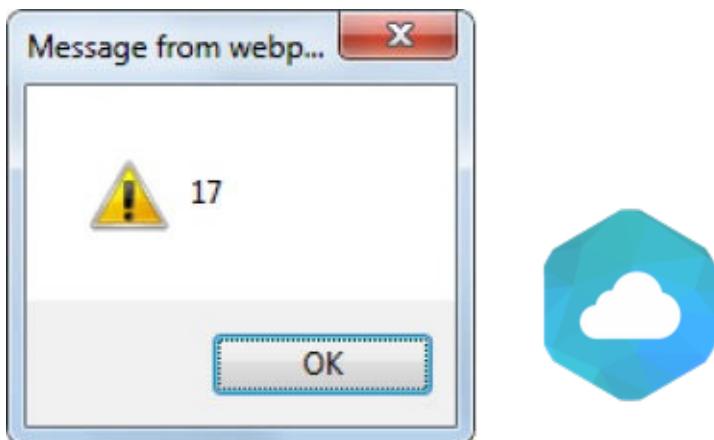
Cả hai ví dụ đều cho ra cùng một kết quả, bạn có thể xem ở ví dụ tiếp theo dưới đây.

Lấy chiều dài của một chuỗi

1. Mở trình duyệt IE.
2. Trên thanh địa chỉ, gõ dòng lệnh sau (bạn có thể thấy dòng lệnh này trong file stringlength.txt của phần Tài nguyên đi kèm):

```
javascript:alert("Đây là một chuỗi.".length);
```

Kết quả là một hộp thoại hiển thị 17 như sau:



1. Bây giờ, thử nhập đoạn mã sau trên thanh địa chỉ:

```
javascript:var x = "Đây là một chuỗi."; alert(x.length);
```

Kết quả là hộp thoại hiển thị kết quả 17 như hình trên.

Phương thức *substring* trả về chuỗi con bao gồm các ký tự từ đối số đầu tiên tới ký tự kề trước đối số thứ hai như trong ví dụ sau:

```
alert(x.substring(0, 3));
```

Ví dụ tiếp theo trả về các ký tự từ vị trí đầu tiên đến vị trí thứ 5 của chuỗi x. Ví dụ:

```
var x = "Steve Suehring";
alert(x.substring(0, 5));
```

Kết quả là hộp thoại hiển thị chuỗi "Steve".3.

```
javascript:var x = "Đây là một chuỗi."; alert(x.length);
```

Kết quả là hộp thoại hiển thị kết quả 17 như hình trên.

Phương thức *substring* trả về chuỗi con bao gồm các ký tự từ đối số đầu tiên tới ký tự kề trước đối số thứ hai như trong ví dụ sau:

```
alert(x.substring(0,3));
```

Ví dụ tiếp theo trả về các ký tự từ vị trí đầu tiên đến vị trí thứ 5 của chuỗi x. Ví dụ:

```
var x = "Steve Suehring";
alert(x.substring(0,5));
```

Kết quả là hộp thoại hiển thị chuỗi "Steve".

Cách xác định chỉ số

Cách xác định chỉ số trong phương thức *substring* tương đối khác, hoặc ít nhất theo tôi là như vậy. Ký tự đầu tiên đại diện bởi số nguyên 0. Điều này hoàn toàn bình thường, vì 0 được sử dụng là chỉ số đầu tiên trong nhiều ngôn ngữ lập trình. Tuy nhiên, chỉ số cuối trong phương thức *substring* lại lớn hơn chỉ số của ký tự cuối cùng trong chuỗi con 1 đơn vị.

Thông thường, bạn sẽ nghĩ với giá trị chỉ số là 0 và 5 (như trong ví dụ trên), kết quả nhận được sẽ là 6 ký tự đầu tiên, từ 0 tới 5, đó là chuỗi "Steve ", kết thúc bằng một ký tự trắng. Tuy nhiên, rất tiếc không phải vậy! `longer` lớn hơn chuỗi con mà bạn mong muốn 1 đơn vị và chuỗi con này không bao gồm ký tự đó.

Ngoài *substring*, một số phương thức của chuỗi được sử dụng phổ biến là *slice*, *substr*, *concat*, *toUpperCase*, *toLowerCase* và các phương thức so khớp như *match*, *search* và *replace*. Sau đây, tôi sẽ giới thiệu cụ thể từng loại.

Các phương thức thay đổi chuỗi bao gồm *slice*, *substring*, *substr* và *concat*. Phương thức *slice* và *substring* trả về chuỗi giá trị dựa trên một chuỗi khác. Chúng chấp nhận hai đối số: vị trí đầu và vị trí cuối tùy chọn. Dưới

đây là một số ví dụ:

```
var myString = "Biến là một chuỗi.";  
alert(myString.substring(3)); //Trả về "n là một chuỗi."  
alert(myString.substring(3,9)); //Trả về "n là m"  
alert(myString.slice(3)); //Trả về "n là một chuỗi."  
alert(myString.slice(3,9)); //Trả về "n là m"
```

Phương thức *substr* cũng chấp nhận hai đối số: đối số đầu là vị trí đầu để trả về, đối số thứ hai là số ký tự trả về chứ không phải vị trí kết thúc (khác với *substring/slice*). Xem xét ví dụ với *substr*:

```
var myString = "Biến là một chuỗi.";  
alert(myString.substr(3)); //Trả về "n là một chuỗi."(Giống :  
alert(myString.substr(3,9)); //Trả về "n là m" (Khác subs
```

Phương thức *concat* nối hai chuỗi với nhau:

```
var firstString = "Hello ";  
var finalString = firstString.concat("World");  
alert(finalString); //Trả về "Hello World"
```

Phương thức *toUpperCase* và *toLowerCase* cùng với hai phương thức cùng loại là *toLocaleUpperCase* và *toLocaleLowerCase* chuyển tất cả các ký tự trong chuỗi thành chữ hoa hoặc chữ thường:

```
var myString = "biến là một Chuỗi";  
alert(myString.toUpperCase()); // "BIẾN LÀ MỘT CHUỖI"  
alert(myString.toLowerCase()); // "biến là một chuỗi"
```

Chú ý Các phương thức *toLocale* thực hiện chuyển đổi tùy theo từng vùng xác định.

Như tôi đã nói ở phần trước, có rất nhiều phương thức và thuộc tính của chuỗi. Bạn có thể tìm hiểu danh sách phương thức và thuộc tính đầy đủ tại địa chỉ <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.

Kiểu logic

Trong JavaScript, kiểu logic (Boolean) là một kiểu dữ liệu bị động. *Bị động*

nghĩa là kiểu logic không được dùng như kiểu số và chuỗi; bạn vẫn có thể khai báo và sử dụng biến logic nhưng thông thường nó chỉ được sử dụng để xác định giá trị cho các biểu thức logic. Kiểu logic chỉ có hai giá trị *true* hoặc *false*, được dùng để đánh giá biểu thức logic trong các câu lệnh điều kiện ví dụ như *if/then/else*.

Xem xét đoạn mã sau:

```
If (myNumber > 18) {  
// Viết mã ở đây  
}
```

Biểu thức logic được sử dụng làm điều kiện trong câu lệnh *if* để xác định khi nào đoạn mã trong dấu ngoặc nhọn {} sẽ được thực thi. Nếu biến *myNumber* có giá trị lớn hơn 18, biểu thức logic sẽ có giá trị *true* (đúng), ngược lại có giá trị *false* (sai).

Kiểu Null

Null là một kiểu dữ liệu đặc biệt trong JavaScript (cũng như trong hầu hết các ngôn ngữ lập trình). Khi một biến là *null*, nó không là gì và cũng không chứa gì cả. Tuy vậy, *null* khác với giá trị *rỗng*. Ví dụ, khai báo và thiết lập một biến là chuỗi rỗng sẽ có dạng như sau:

```
var myVariable = '';
```

Biến *myVariable* là rỗng, nhưng không phải là *null*.

Undefined

Undefined (không xác định) là trạng thái, đôi khi được dùng như là giá trị, biểu thị biến chưa chứa giá trị nào. Trạng thái này khác so với *null*, mặc dù cả *null* và *undefined* có thể được đánh giá như nhau. Bạn sẽ học cách phân biệt giữa giá trị *null* và *undefined* trong Chương 5 “Sử dụng toán tử và biểu thức”.

Kiểu đối tượng

Tương tự như hàm, nội dung chi tiết về đối tượng được trình bày trong một

chương riêng (Chương 8). Tuy nhiên, chương này, sẽ giới thiệu ngắn gọn về đối tượng. JavaScript là ngôn ngữ dựa trên đối tượng, chứ không phải là ngôn ngữ hướng đối tượng. Tuy nhiên, JavaScript cũng có các chức năng tương tự như một ngôn ngữ hướng đối tượng và trong hầu hết các cách sử dụng JavaScript cơ bản, bạn sẽ không nhận thấy có sự khác biệt.

Đối tượng trong JavaScript là tập hợp các thuộc tính, mỗi thuộc tính chứa một giá trị cơ bản. Các thuộc tính - được hiểu như là *chìa khóa* - cho phép truy xuất tới các giá trị. Mỗi thuộc tính có thể chứa một giá trị, một đối tượng hoặc thậm chí một phương thức. Bạn có thể khai báo các đối tượng bằng JavaScript, hoặc sử dụng những đối tượng có sẵn.

Đối tượng được tạo bằng cặp dấu ngoặc nhọn, đoạn mã dưới đây tạo một đối tượng rỗng có tên là *myObject*:

```
var myObject = {};
```

Còn đây là ví dụ về một đối tượng có nhiều thuộc tính:

```
var dvdCatalog = {  
    "identifier": "1",  
    "name": "Coho Vineyard"  
};
```



Ví dụ trên tạo đối tượng *dvdCatalog* có hai thuộc tính: *identifier* và *name*. Giá trị của mỗi thuộc tính tương ứng là *1* và *"Coho Vineyard"*. Bạn có thể truy xuất thuộc tính *name* của đối tượng *dvdCatalog* như sau:

```
alert(dvdCatalog.name);
```

Sau đây là ví dụ hoàn chỉnh về đối tượng (Bạn có thể tìm thấy đoạn mã này trong file object.txt).

```
// Tạo một đối tượng mới sử dụng dấu ngoặc nhọn  
var star = {}  
// Tạo bốn đối tượng được đặt tên theo tên của bốn ngôi sao  
star["Polaris"] = new Object  
star["Deneb"] = new Object;  
star["Vega"] = new Object;  
star["Altair"] = new Object;
```

Các phần tiếp theo của cuốn sách này sẽ trình bày thêm về thuộc tính của đối tượng cũng như cách truy cập các thuộc tính này. Các vấn đề cụ thể về đối tượng

sẽ được trình bày kỹ trong Chương 8.

Kiểu mảng

Trong ví dụ trước, bạn đã được giới thiệu cách tạo ra một đối tượng với tên riêng. Bạn cũng có thể sử dụng những đối tượng không có tên được truy xuất thông qua chỉ số. Đó là các mảng theo kiểu cũ, vốn quen thuộc trong rất nhiều ngôn ngữ lập trình. Bạn cũng vừa thấy nhiều đối tượng có thể được khai báo cùng lúc và đặt tên theo các ngôi sao. Đoạn mã sau tạo một mảng với phần tử là các đối tượng trên. Ví dụ này có trong file stararray.txt của phần Tài nguyên đính kèm.

```
var star = new Array();
star[0] = "Polaris";
star[1] = "Deneb";
star[2] = "Vega";
star[3] = "Altair";
```

Bạn cũng có thể sử dụng cách khởi tạo mảng tường minh như dưới đây:

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
```

Mảng có thể chứa giá trị lồng nhau, như trong ví dụ sau của các ngôi sao được gộp theo chòm sao mà nó xuất hiện:

```
var star = [["Polaris", "Ursa Minor"], ["Deneb", "Cygnus"],
["Vega", "Lyra"], ["Altair", "Aquila"]];
```

Cuối cùng, mặc dù không phổ biến, bạn có thể gọi hàm khởi tạo *Array()* với đối số là các phần tử mảng như sau:

```
var star = new Array("Polaris", "Deneb", "Vega", "Altair");
```

Chú ý Việc gọi hàm khởi tạo *Array()* với đối số là một số nguyên sẽ thiết lập độ dài của mảng chứ không phải tạo ra một mảng có một phần tử với giá trị là đối số truyền vào.

Đặc tả ECMA-262 mới phiên bản 5 có thêm một vài phương thức mới cho việc duyệt và làm việc với mảng. Kiến thức về mảng bao gồm các phương thức duyệt và làm việc với các phần tử sẽ được trình bày kĩ hơn trong Chương 8.

Khai báo và sử dụng biến

Dù sử dụng ngôn ngữ nào lập trình viên cũng cần làm quen với biến. Biến chứa những dữ liệu có thể thay đổi trong suốt vòng đời của chương trình. Bạn đã thấy những ví dụ về khai báo biến trong các chương trước. Ở phần này, bạn sẽ chính thức tìm hiểu cách sử dụng biến trong JavaScript.

Khai báo biến

Trong JavaScript, biến được khai báo với từ khóa `var`. Bạn có thể tìm thấy các ví dụ về khai báo biến trong file `variablenaming.txt` của phần Tài nguyên đi kèm. Các cách khai báo biến như sau đều hợp lệ:

```
var x;  
  
var myVar;  
  
var counter1;
```



Tên của biến có thể chứa ký tự in hoa, in thường cũng như số, tuy nhiên chúng không được bắt đầu bằng một chữ số. Tên biến không được chứa ký tự trắng hoặc các ký tự đặc biệt, ngoại trừ dấu gạch dưới `()`. (*Mặc dù trên thực tế, tôi ít thấy ký tự trong biến JavaScript*). Các tên biến sau là không hợp lệ:

```
var 1stCounter;  
  
var new variable;  
  
var new.variable;  
  
var var;
```

Trong các ví dụ trên, ba tên biến đầu không hợp lệ vì chúng sử dụng các ký tự không hợp lệ (hoặc là sử dụng ký tự hợp lệ ở sai vị trí, như trong ví dụ đầu tiên), còn tên biến cuối cùng, `var`, là không hợp lệ vì nó trùng với từ khóa. Để biết thêm thông tin về từ khóa trong JavaScript, đọc lại Chương 2, “Lập trình với JavaScript”.

Bạn có thể khai báo nhiều biến trên một dòng lệnh như sau:

```
var x, y, zeta;
```

Biến cũng có thể được khai báo và khởi tạo giá trị trên cùng một dòng:

```
var x = 1, y = "hello", zeta = 14;
```

Kiểu biến

Các biến trong JavaScript không có kiểu rõ ràng. Bạn không cần khai báo trước một biến là số nguyên, số thực hay là chuỗi. Bạn có thể thay đổi kiểu dữ liệu của biến đơn giản bằng cách gán một giá trị khác kiểu cho nó. Hãy xem ví dụ sau, biến `x` ban đầu chứa giá trị nguyên, nhưng sau đó được gán lại, nó thay đổi để chứa giá trị kiểu chuỗi:

```
var x = 4;
```

```
var x = "Bây giờ nó là một chuỗi.";
```



Phạm vi của biến

Phạm vi của biến đề cập tới các vị trí mà trong đó giá trị của biến có thể được truy xuất. Biến có phạm vi toàn cục có thể được truy xuất ở bất kỳ đâu trong một chương trình. Trong ngữ cảnh một trang web – hoặc một tài liệu, bạn có thể truy xuất và sử dụng biến toàn cục ở bất cứ đâu.

Biến được khai báo trong một hàm thì chỉ được sử dụng bên trong hàm đó. Điều này có nghĩa là giá trị của những biến này không thể được truy xuất bên ngoài hàm. Tham số của hàm có phạm vi ở bên trong hàm đó.

Sau đây là một vài ví dụ thực tế về phạm vi của biến, bạn có thể tìm thấy chúng trong file `scope1.txt` của phần Tài nguyên đi kèm:

```
<script type="text/javascript">  
  
var aNewVariable = "Biến toàn cục";  
  
function doSomething(incomingBits) {
```

```

        alert(aNewVariable);

        alert(incomingBits);

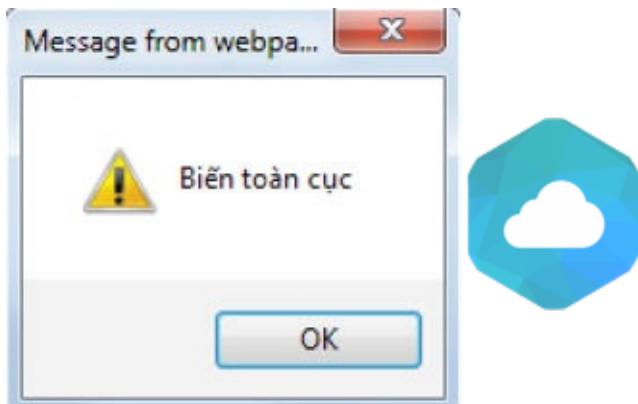
    }

doSomething("Biến cục bộ");

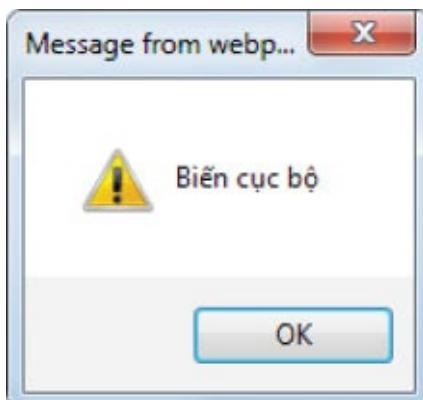
</script>

```

Đoạn mã trên định nghĩa hai biến: biến toàn cục *aNewVariable* và biến cục bộ *incomingBits* chỉ có phạm vi bên trong hàm *doSomething()*. Cả hai biến đều được truyền vào hàm *alert()*. Khi hàm *doSomething()* được gọi, nội dung của cả hai biến được gửi thành công và hiển thị lên màn hình như trong Hình 4-2 và 4-3.



Hình 4-2 Biến *_aNewVariable_* là biến toàn cục.



Hình 4-3 Biến *incomingBits* có phạm vi bên trong hàm *doSomething*.

Sau đây là một ví dụ phức tạp hơn.

Kiểm tra phạm vi của biến

1. Sử dụng Visual Studio, Eclipse hoặc một trình soạn thảo khác sửa file scoping.htm trong thư mục mã nguồn mẫu Chương 4.
2. Trong trang này, thay thế chú thích TODO bằng đoạn mã in đậm sau (đoạn mã này có thể được tìm thấy trong file scoping.txt của phần Tài nguyên đi kèm):

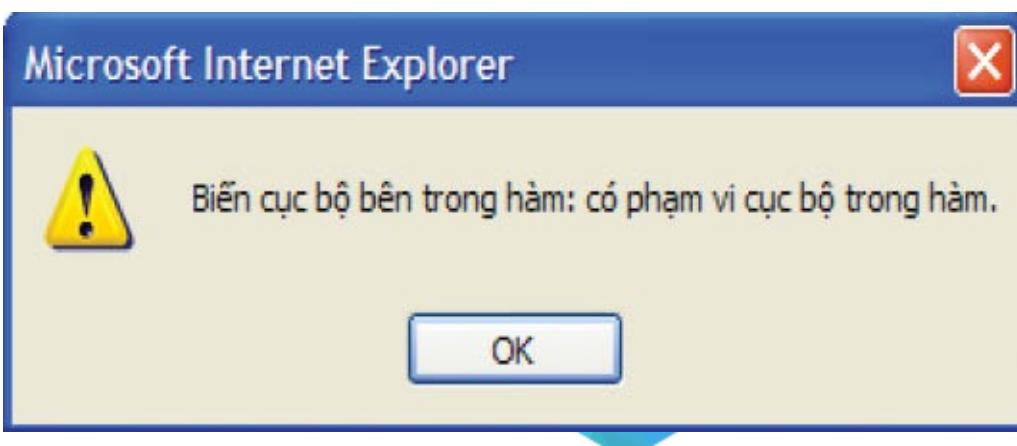
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Ví dụ về phạm vi</title>  
    <script type="text/javascript">  
        var aNewVariable = "có phạm vi toàn cục.";  
        function doSomething(incomingBits) {  
            alert("Biến toàn cục bên trong hàm: " + aN  
                alert("Biến cục bộ bên trong hàm: " + inc  
            }  
        </script>  
    </head>  
    <body>  
        <script type="text/javascript">  
            doSomething(" có phạm vi cục bộ trong hàm.");  
            alert("Biến toàn cục bên ngoài hàm: " + aNewVariable);  
            alert("Biến cục bộ bên ngoài hàm: " + incomingBits);  
        </script>  
    </body>  
</html>
```



Kết quả là ba hộp thoại trên màn hình. Hộp thoại thứ nhất:



Hộp thoại thứ hai:



Hộp thoại thứ ba:



Bây giờ, bạn hãy đọc kỹ lại đoạn mã. Bạn thấy có bao nhiêu lời gọi hàm `alert()`? Gợi ý: Hai lời gọi trong phần `<head>` và hai lời gọi khác trong phần `<body>`. Tổng cộng có bốn lần gọi hàm `alert()`. Vậy tại sao chỉ có ba hộp thoại hiện lên màn hình?

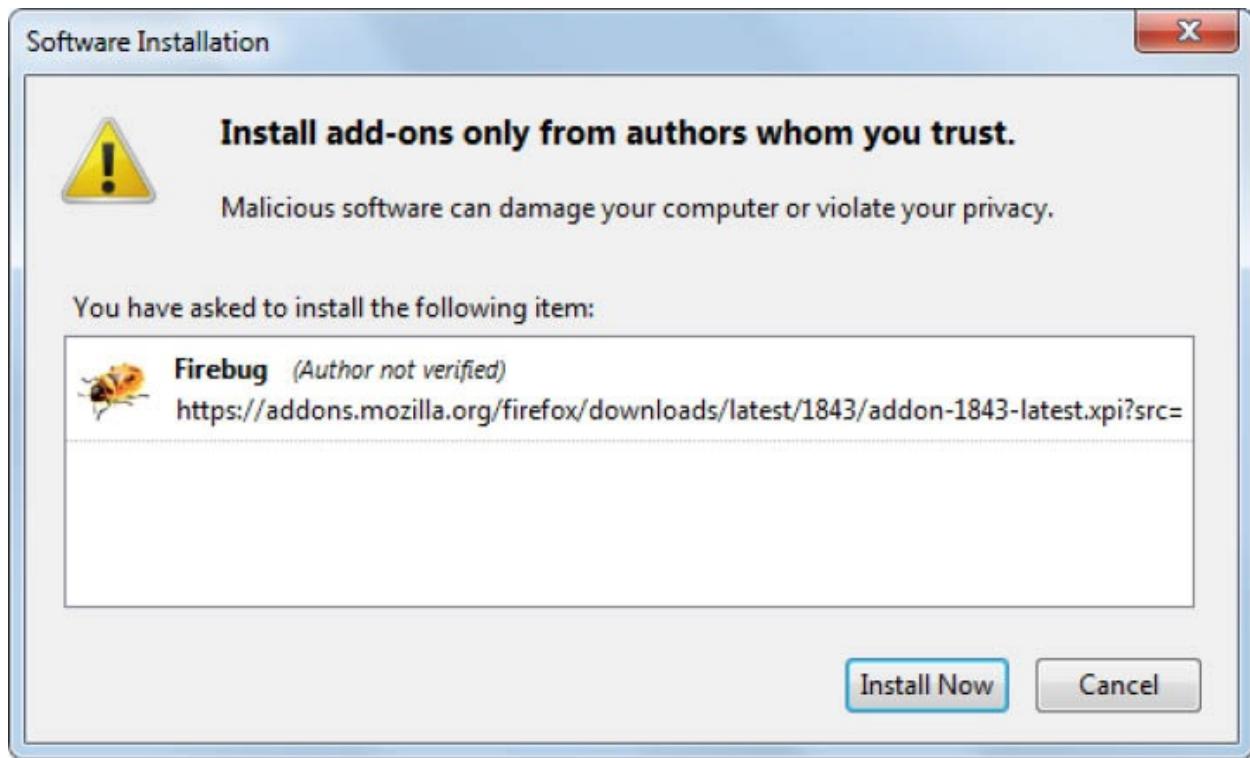
Bởi vì đây là phần nói về phạm vi của biến (và tôi đã vừa gợi ý), bạn có thể tự suy luận ra câu trả lời. Ví dụ này cũng minh họa cho cách xử lý các vấn đề trong JavaScript khi kết quả không như những gì chúng ta muốn.

Bước tiếp theo yêu cầu sử dụng add-on Firebug của trình duyệt Mozilla Firefox. Tôi tin rằng đến lúc này bạn đã cài đặt trình duyệt Firefox trên máy tính của mình (xem Chương 1, “Hiểu hơn về JavaScript” để biết vì sao nên cài đặt Firebug). Nếu máy tính của bạn chưa có Firefox, hãy tải về từ địa chỉ: <http://www.mozilla.com/firefox/>.

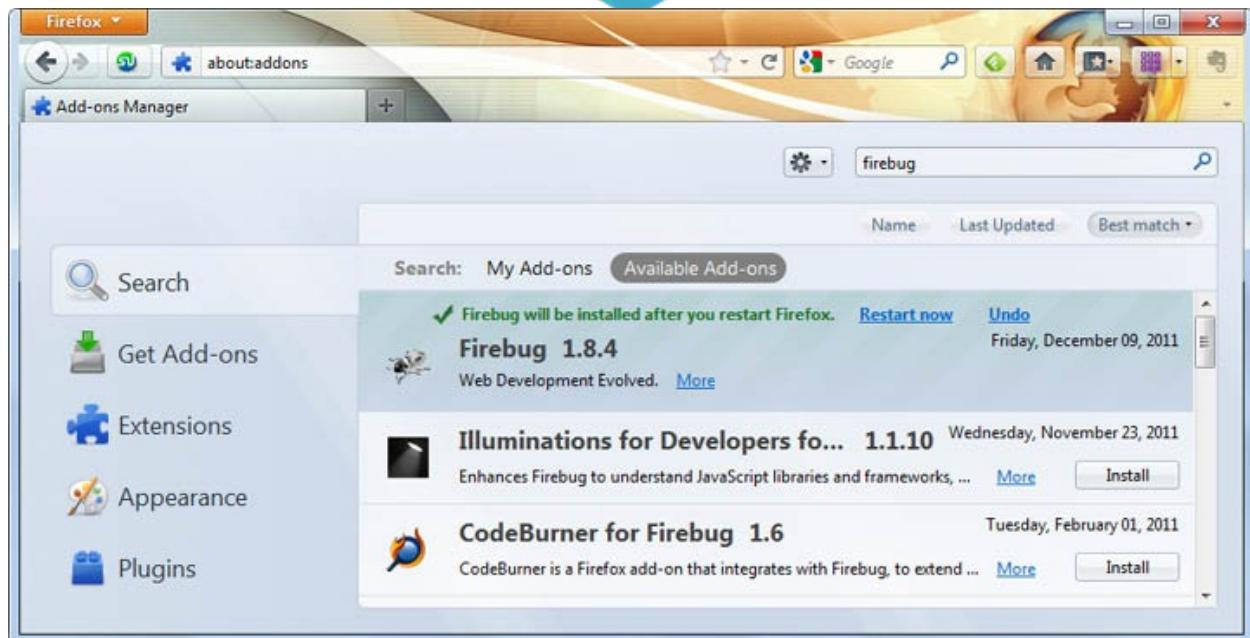
Cài đặt Firebug

Bước đầu tiên hướng dẫn bạn cách cài đặt Firebug trong Firefox. Mặc dù IE có trình gỡ lỗi riêng là Microsoft Script Debugger, Firebug vẫn là công cụ mạnh mẽ và linh hoạt hơn.

1. Sau khi cài Firefox, bạn có thể tiếp tục cài Firebug. Bắt đầu bằng việc vào trang <http://www.getfirebug.com/>. Khi trang web tải xong, nhấn vào liên kết cài đặt. (Nếu đây là lần đầu tiên cài đặt Firebug, bạn có thể nhận được cảnh báo rằng Firefox chặn việc cài đặt phần mềm. Thông báo này không xuất hiện trong các phiên bản sau của Firefox).
2. Sau khi nhấn vào liên kết cài đặt, một cửa sổ cài đặt phần mềm được mở ra như sau. Nhấn vào Install Now.

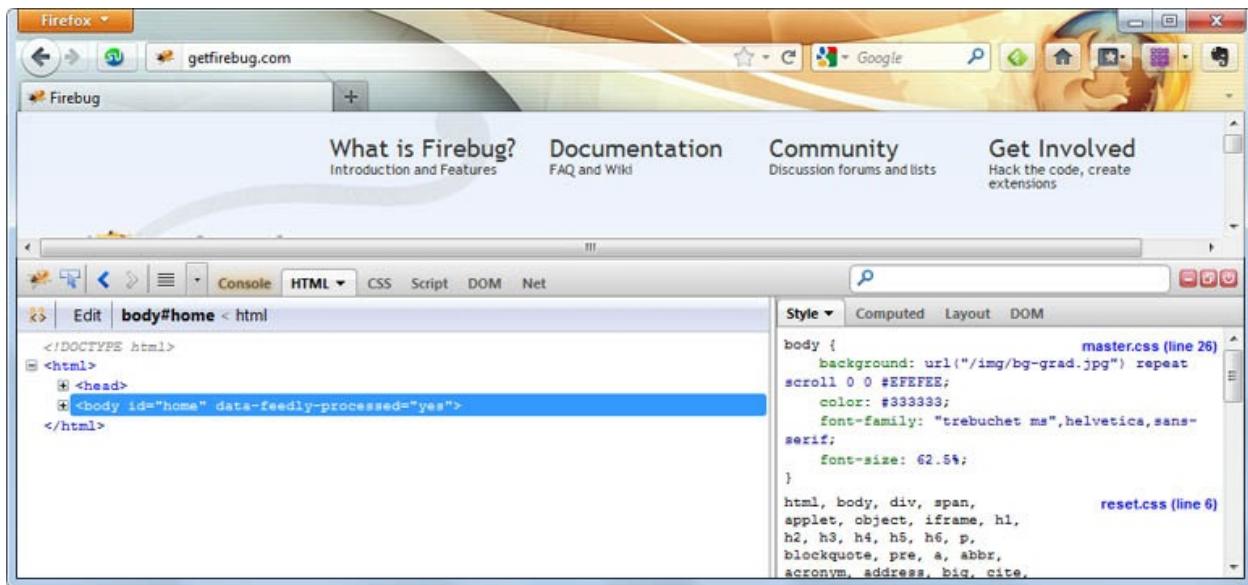


3. Cửa sổ Add-Ons mở ra (như trong hình dưới) và Firebug được tải về. Quá trình cài đặt hoàn tất khi bạn khởi động lại Firefox, vì vậy hãy nhấn Restart Firefox sau khi tải xong Firebug.



4. Sau khi đóng và mở lại Firefox, các add-on mới cài đặt sẽ được hiển thị. Đóng hộp thoại Add-Ons lại. Chúc mừng! Firebug đã được cài đặt xong. Bạn có thể

thấy biểu tượng nhỏ ở góc phải bên dưới cửa sổ trình duyệt. Nhấn vào biểu tượng này để mở giao diện điều khiển Firebug:

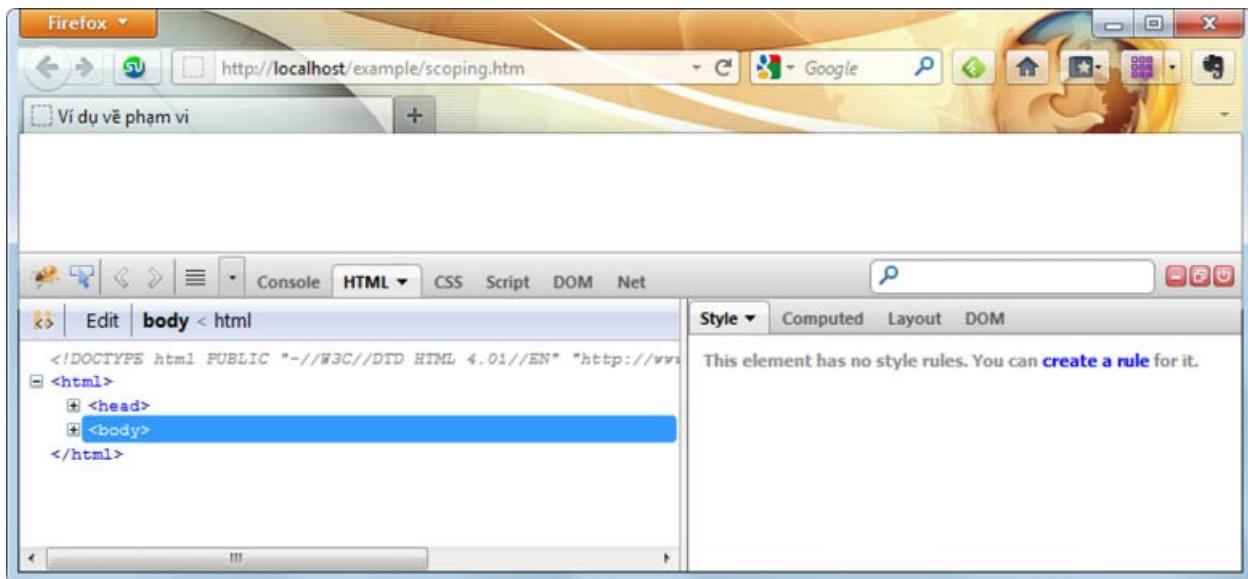


5. Firebug đang bị tắt nhưng chúng ta sẽ học cách khởi động và sử dụng nó ngay sau đây. Hãy thoải mái thử nghiệm bằng cách bật Firebug trong trang này hoặc tắt cả các trang khác.

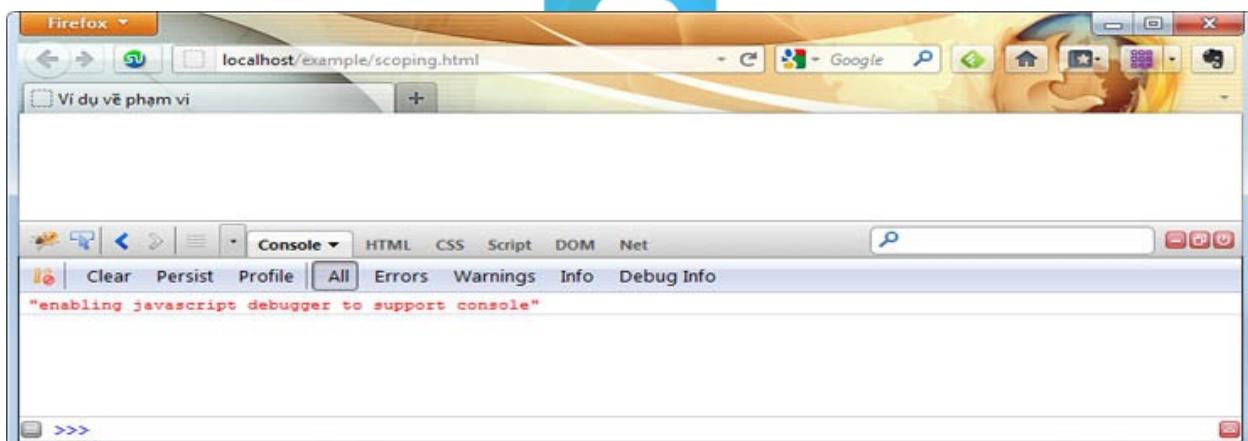
Sau khi đã cài đặt Firebug, bạn có thể gõ lỗi cho vấn đề trong ví dụ về phạm vi của biến ở trên.

Xử lý lỗi với Firebug

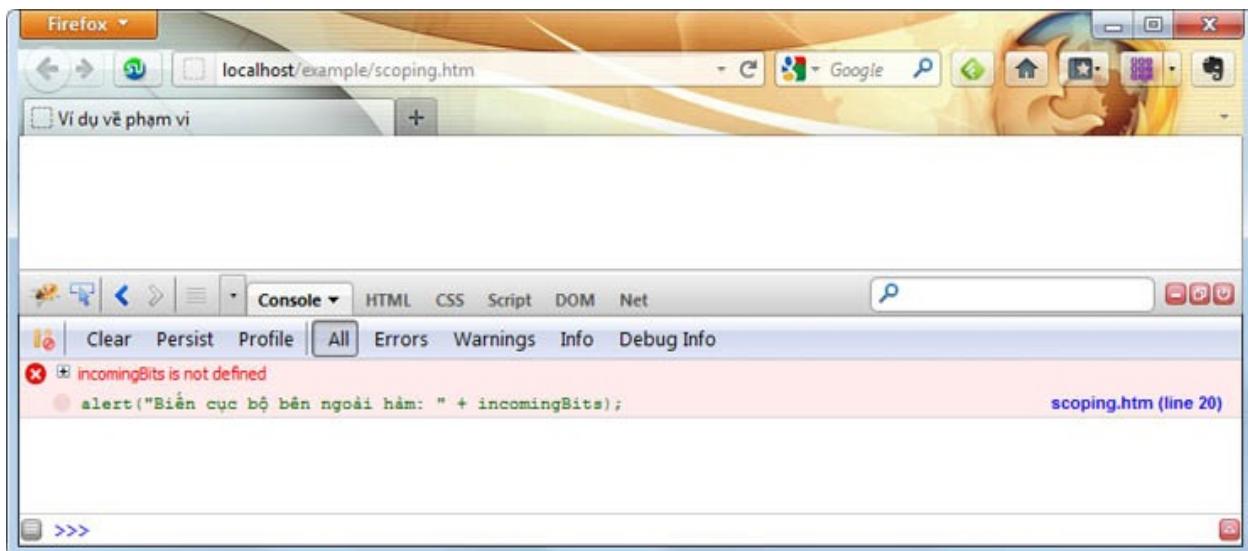
1. Mở Firefox và chọn ví dụ scoping.htm đã được tạo ở phần đầu chương. Mã JavaScript thực thi và vẫn chỉ hiển thị ba hộp thoại thông báo như trước. Đóng tất cả các hộp thoại thông báo. Bạn sẽ thấy một trang trắng được tải lên Firefox.
2. Nhấn vào biểu tượng ở góc dưới cùng bên phải của cửa sổ trình duyệt Firefox để mở Firebug.



3. Nhấn vào tab Script để mở khung Script, chú ý là nó đã bị tắt. Bạn cần kích hoạt khung Console để nhận được tất cả các thông báo lỗi. Khung Script được kích hoạt khi khung Console đã được kích hoạt, vì vậy hãy kích hoạt khung Console. Nhấn vào tab Console, tiếp tục nhấn vào biểu tượng mũi tên/tam giác bên cạnh từ Console và nhấn Enabled. Trình gỡ lỗi JavaScript được kích hoạt.



4. Khi cả khung Console và Script đều được kích hoạt, nhấn vào nút Reload trên thanh công cụ chính của Firefox hoặc chọn Reload từ menu View. Trang được tải về và mã JavaScript sẽ thực thi lần nữa. Cả ba hộp thoại lại hiển thị, nhưng lưu ý là lúc này Firebug đã tìm ra một lỗi, biểu thị bằng chữ X màu đỏ và dấu hiệu Error ở góc phải của thanh trạng thái như minh họa trong hình bên dưới:



5. Nếu khung Console không mở, nhấp vào tab Console trong Firebug để tìm lỗi, lỗi ở đây là do biến *incommingBits* chưa được khai báo. Cửa sổ này cũng chỉ ra dòng lệnh bị lỗi. Tuy nhiên hãy chú ý rằng do cách thức phân tích tài liệu nên số dòng trong mã nguồn của bạn có thể không chính xác. Dù vậy, bạn có thể thấy rằng biến *incommingBits* chưa được khai báo trong khối *<body>* bởi vì phạm vi của nó bị giới hạn trong hàm *doSomething()*.

Phần này trình bày cách sử dụng Firebug đồng thời nói về sự khác nhau giữa biến toàn cục và biến cục bộ. Firebug là phần không thể thiếu với trình gỡ lỗi JavaScript. Vì vậy, hãy dành thời gian tìm hiểu xem Firebug, JavaScript, CSS và HTML tương tác với nhau ra sao.

Cách gỡ lỗi trong ví dụ trên là khai báo lại biến *incommingBits* để khởi tạo nó bên ngoài lời gọi hàm. (Dòng lệnh thêm vào được viết như dưới đây. Bạn cũng có thể xem trong file scoping-fixed.htm trong thư mục Chương 4 của phần Tài nguyên đi kèm). Vì biến này được khai báo bên trong câu lệnh khai báo hàm nên nó không thể tồn tại bên ngoài phạm vi của hàm.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Ví dụ về phạm vi</title>  
    <script type="text/javascript">  
        var aNewVariable = "có phạm vi toàn cục.";  
        function doSomething(incomingBits) {  
            alert("Biến toàn cục bên trong hàm:  
                " + aNewVariable);  
        }  
    </script>  
</head>  
<body>  
    <h1>Ví dụ về phạm vi</h1>  
    <p>Hàm doSomething() nhận một tham số là incomingBits.</p>  
    <p>Trong hàm, chúng ta truy cập biến aNewVariable mà không cần khai báo.</p>  
    <p>Kết quả hiển thị:</p>  
    <pre>Biến toàn cục bên trong hàm:  
        có phạm vi toàn cục.  
</pre>  
</body>
```

```
        alert("Biến cục bộ bên trong hàm: " +  
    }  
    </script>  
</head>  
<body>  
<script type="text/javascript">  
    var incomingBits = "phải được định nghĩa nếu  
    doSomething("có phạm vi cục bộ trong hàm.");  
    alert("Biến toàn cục bên ngoài hàm " + aNewVariable);  
    alert("Biến cục bộ bên ngoài hàm: " + incomingBits);  
</script>  
</body>  
</html>
```

Bạn có thể tìm hiểu thêm về các hàm trong Chương 7 “Làm việc với hàm”.

Đối tượng *Date*



Đối tượng *Date* bao gồm rất nhiều phương thức hữu ích để làm việc với dữ liệu ngày tháng. Trong cuốn sách này, tôi chỉ đưa ra một số ví dụ mà có thể bạn sẽ gặp thường xuyên trong quá trình làm việc.

Một trong những nhược điểm của đối tượng *Date* trong JavaScript là các phương thức của *Date* được thực thi rất khác nhau tùy theo trình duyệt và hệ điều hành. Ví dụ, hãy xem xét đoạn mã trả về ngày hiện tại theo múi giờ địa phương và được định dạng tự động bởi phương thức *toLocaleDateString()*:

```
var myDate = new Date();  
alert(myDate.toLocaleDateString());
```

Khi chạy trên trình duyệt IE8 với hệ điều hành Windows 7, kết quả trả về thời gian như hiển thị trong Hình 4-4.



Hình 4-4 Phương thức `toLocaleString()` của đối tượng `Date` trong IE8.

Hình 4-5 hiển thị kết quả khi chạy đoạn mã tương tự trên trình duyệt Firefox 3.6, hệ điều hành Linux.



Hình 4-5 Phương thức `toLocaleString()` của đối tượng `Date` hiển thị kết quả khác khi chạy trên trình duyệt Firefox, hệ điều hành Linux.

Cách hiển thị khác nhau của hai hộp thoại này có vẻ như không đáng bận tâm, nhưng nếu bạn muốn sử dụng các ngày trong tuần (Monday trong ví dụ trên) trong đoạn mã của mình thì mọi chuyện sẽ hoàn toàn khác. Đừng nghĩ rằng vấn đề này chỉ là do hệ điều hành khác nhau. Sự khác biệt của đối tượng `Date` và các phương thức của nó cũng tồn tại ở các trình duyệt khác nhau trên cùng hệ điều hành Microsoft Windows.

Phương thức `getYear()` của đối tượng `Date` cũng là một ví dụ cho thấy sự khác nhau trong thực thi JavaScript, lần này là trên hai máy chạy cùng phiên bản Windows 7. Xem xét đoạn mã sau:

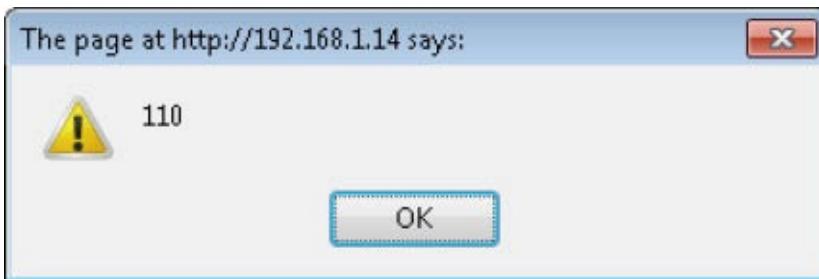
```
var myDate = new Date();
alert(myDate.getYear());
```

Khi gọi bằng IE8, đoạn mã này xuất ra kết quả như hiển thị trong Hình 4-6.



Hình 4-6 Kết quả trả về của phương thức `getYear()` trong IE8 hiển thị năm đầy đủ.

Khi chạy trên Firefox 3.6 cũng trên một máy chạy Windows, đoạn mã trên xuất ra kết quả hiển thị như Hình 4-7.



Hình 4-7 Kết quả trả về phương thức `getYear()` trên Firefox hiển thị năm tính từ mốc 1900.

Phiên bản Firefox trả về 110 (2010-1900), là năm bắt đầu từ mốc 1900. Kết quả nào đúng hơn? Phiên bản Firefox trả về kết quả đúng theo đặc tả ECMA-262 về phương thức `getYear()`: “Trả về năm hiện tại - 1900”.

Rất may là chúng ta có một cách đơn giản để giải quyết sự khác biệt này. Đặc tả ECMA-262 cung cấp phương thức `getFullYear()`, trả về năm đầy đủ (2010) như trong ví dụ đã đưa.

Cách duy nhất để giải quyết triệt để vấn đề kết quả khác nhau thực thi ứng dụng JavaScript là tiến hành kiểm thử trên các trình duyệt và các nền tảng khác nhau. Cách này sẽ kéo dài thời gian phát triển ứng dụng, tuy nhiên việc tìm và sửa lỗi trong quá trình phát triển chắc chắn có chi phí thấp hơn nhiều so với việc sửa lỗi sau khi người sử dụng phát hiện ra chúng.

Ngày nào?

Chú ý đến thời gian trong Hình 4-4 và 4-5: Saturday, June 16, 2001. Bạn có thể tự hỏi liệu có phải cuốn sách này được viết từ năm 2001. Không phải vậy. Chỉ đơn giản là việc dùng một năm đã qua sẽ giúp minh họa cho vấn đề

về dữ liệu thời gian trong JavaScript. Các hàm JavaScript trả về thời gian được thiết lập cho máy tính đang chạy đoạn mã JavaScript đó.

Để chứng minh điều này, tôi đã thay đổi thời gian trong máy tôi thành 16 tháng 6 năm 2001. Khi bạn sử dụng bất cứ phương thức nào của đối tượng *Date*, hãy nhớ rằng nó luôn phản ánh thời gian trên máy của người dùng.

Đối tượng *Date* có thể xử lý từ 0 tới 7 đối số. Nếu khi khởi tạo, đối tượng *Date* được truyền vào một đối số dạng chuỗi, chuỗi này chứa dữ liệu thời gian. Khi đối số truyền vào có dạng số, đối số này được giả định là ngày theo mili giây tính từ mốc 1/1/1970 và khi cả bảy đối số được truyền vào, chúng sẽ có dạng như sau:

```
new Date(năm, tháng, ngày, giờ, phút, giây, mili giây)
```

Chú ý Chỉ bắt buộc truyền đối số năm và tháng, còn lại là không bắt buộc.



Hãy nhớ những điều sau khi sử dụng đối tượng *Date*:

- Năm nên gồm 4 chữ số trừ trường hợp bạn ấn định một năm nằm trong khoảng từ năm 1900 tới năm 2000, khi đó bạn chỉ cần truyền số gồm hai chữ số từ 0 tới 99, chúng sẽ được cộng thêm 1900. Vì vậy, 2008 tương ứng với năm 2008, còn 98 thì là 1998.
- Tháng là số nguyên từ 0 tới 11, với 0 là tháng một và 11 là tháng mười hai.
- Ngày là một số nguyên từ 1 tới 31.
- Giờ là số nguyên từ 0 tới 23, 23 là 11PM.
- Phút và giây đều là số nguyên từ 0 tới 59.
- Mili giây là số nguyên từ 0 tới 999.

Ví dụ tiếp theo trình bày cách hiển thị ngày giờ trên trang web - một thao tác rất phổ biến trong lập trình.

Hiển thị ngày và giờ trên trang web

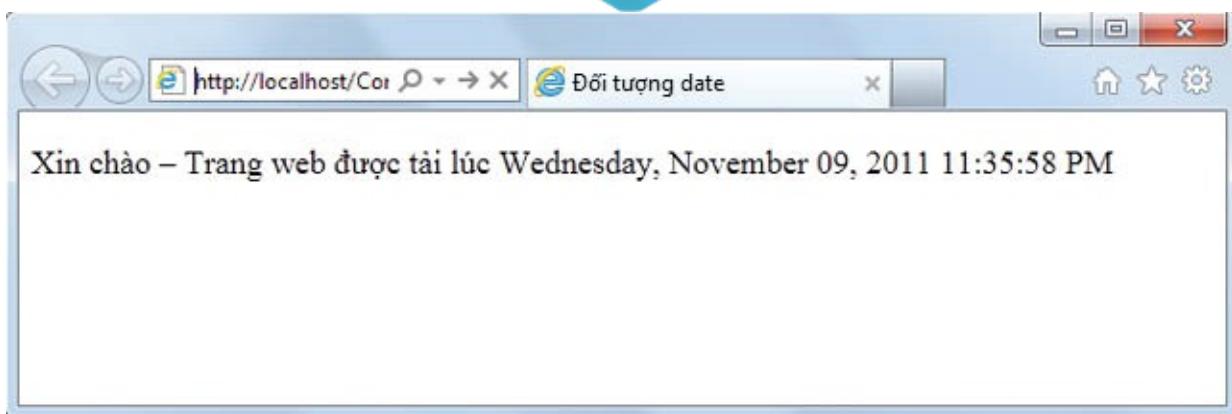
1. Sử dụng Visual Studio, Eclipse hoặc một trình soạn thảo khác chỉnh sửa file

writingthedate.htm trong thư mục mã nguồn mẫu của Chương 4.

- Trong trang này, thêm đoạn mã in đậm dưới đây:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title>Đổi tượng date</title>  
</head>  
<body>  
<p id="dateField">&lt;p>  
<script type="text/javascript">  
var myDate = new Date();  
var dateString = myDate.toLocaleDateString() + " " + myDate.toLocaleTimeString();  
var dateLoc = document.getElementById("dateField");  
dateLoc.innerHTML = "Xin chào - Trang web được tải lúc" + dateString;  
</script>  
</body>  
</html>
```

- Đóng và dùng trình duyệt mở lại trang web, bạn sẽ thấy một trang như hình dưới (ngày hiển thị có thể sẽ khác):



Mã JavaScript của trang web:

```
var myDate = new Date();  
var dateString = myDate.toLocaleDateString() + " " + myDate.toLocaleTimeString();  
var dateLoc = document.getElementById("dateField");  
dateLoc.innerHTML = "Xin chào - Trang web được tải lúc" + dateString;
```

Mã JavaScript dùng trên đổi tượng *Date* thường khá đơn giản. Sử dụng hai

phương thức `toLocaleDateString()` và `toLocaleTimeString()` để trả về ngày và giờ địa phương. Hai phương thức này được nối với nhau bằng một dấu cách và gán vào biến `dateString` như sau:

```
var dateString = myDate.toLocaleDateString() + " " + myDate.
```

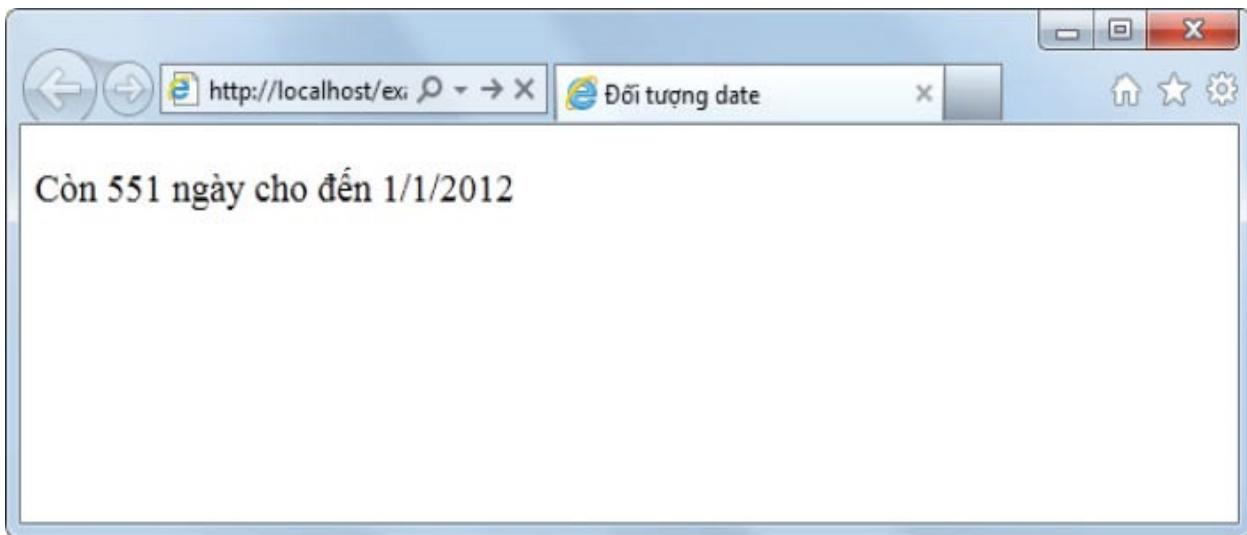
Câu lệnh tiếp theo ghi nội dung của biến `dateString` ra trang web. Chương 10 “Mô hình đối tượng tài liệu” sẽ trình bày chi tiết hơn về vấn đề này.

Đếm ngược đến một ngày cụ thể trong tương lai

1. Sử dụng Visual Studio, Eclipse hoặc một trình soạn thảo khác chỉnh sửa file `countdown.htm` trong thư mục mã nguồn mẫu của Chương 4.
2. Thêm đoạn mã in đậm dưới đây:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title> Đối tượng date</title>  
</head>  
<body>  
<p id="dateField"> </p>  
<script type="text/javascript">  
var today = new Date();  
var then = new Date();  
// Thiết lập ngày là 1/1/2012  
then.setFullYear(2012, 0, 1);  
var diff = then.getTime() - today.getTime();  
diff = Math.floor(diff / (1000 * 60 * 60 * 24));  
var dateLoc = document.getElementById("dateField");  
dateLoc.innerHTML = "Còn " + diff +  
" ngày cho đến ngày 1/1/2012";  
</script>  
</body>  
</html>
```

3. Lưu và dùng trình duyệt mở lại trang web. Tùy theo thời gian được thiết lập trên máy tính của bạn, số ngày được hiển thị sẽ khác, nhưng nhìn chung, trang sẽ hiển thị như sau:



Mách nhỏ Hãy cẩn thận khi sử dụng ngày tháng trong JavaScript vì bất kỳ mục đích nào khác ngoài hiển thị chúng. Vì ngày tháng phụ thuộc vào thời gian được thiết lập trên máy của người dùng nên hãy tránh sử dụng nó khi cần tính thời gian chính xác, chẳng hạn trong những hệ thống đặt hàng.

Ví dụ bạn vừa sử dụng thêm một số hàm của đối tượng *Math* và *Date*, là *floor()* và *getTime()*. Mặc dù cuốn sách này bao quát khá nhiều vấn đề về JavaScript, nó vẫn không phải là tài liệu tham khảo đầy đủ về ngôn ngữ này. Hãy tham khảo chuẩn ECMA-262 tại <http://www.ecma-international.org/publications/standards/Ecma-262.htm> để biết thêm thông tin.

Phần tiếp theo sẽ hướng dẫn bạn cách tính toán (hoặc ước tính) thời gian một trang web cần để tải lên trình duyệt của người dùng.

Chú ý Tính toán này có thể không chính xác vì nó không bao gồm thời gian cần thiết để tải những thành phần không phải văn bản như hình ảnh hoặc các thành phần đa phương tiện khác.

Tính toán thời gian tải trang web

1. Sử dụng Visual Studio, Eclipse hoặc một trình soạn thảo khác chỉnh sửa file render.htm trong thư mục mã mẫu của Chương 4, phần Tài nguyên đi kèm.
2. Thêm đoạn mã in đậm dưới đây:

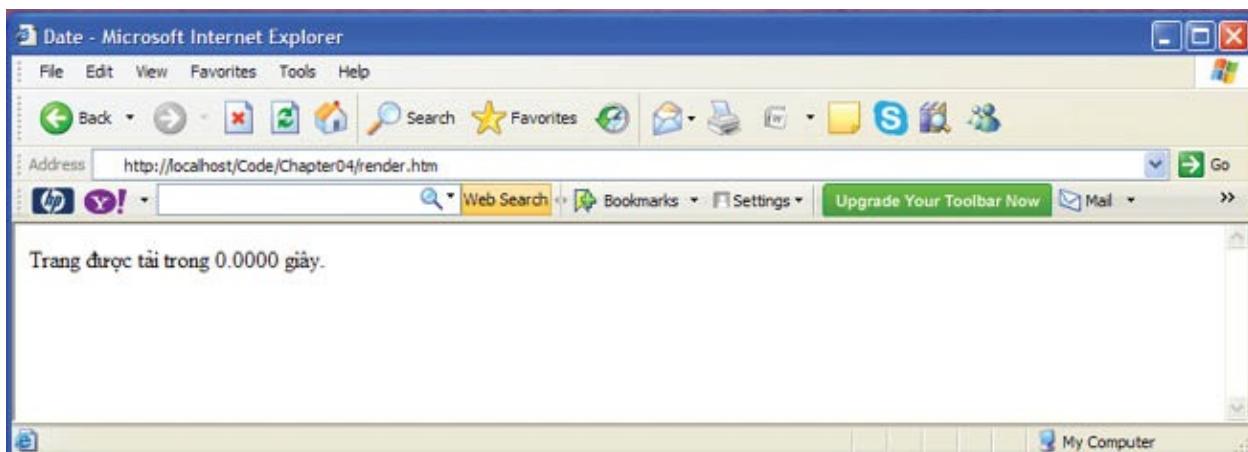
```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional,
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
    <title>Date</title>
    <script type="text/javascript">
        var started = new Date();
        var now = started.getTime();
    </script>
</head>
<body>
    <p id="dateField"> </p>
    <script type="text/javascript">
        var bottom = new Date();
        var diff = (bottom.getTime() - now)/1000;
        var finaltime = diff.toPrecision(5);
        var dateLoc = document.getElementById("dateField");
        dateLoc.innerHTML = "Trang được tải trong " + finaltime +
    </script>
</body>
</html>

```



3. Lưu và dùng trình duyệt mở lại trang. Tùy thuộc vào tốc độ của máy tính, tốc độ của web server và kết nối mạng mà bạn có thể tải về một trang web chỉ sau 0 giây như dưới đây:



1. Nếu kết quả của bạn cũng như trên, bạn có thể tạo độ trễ cho trang đó để thử lại. (Tuy nhiên, trên thực tế sẽ không ai làm như vậy vì không ai muốn làm chậm trang web của mình! Tất nhiên, việc tạo độ trễ rất tiện lợi khi tiến hành

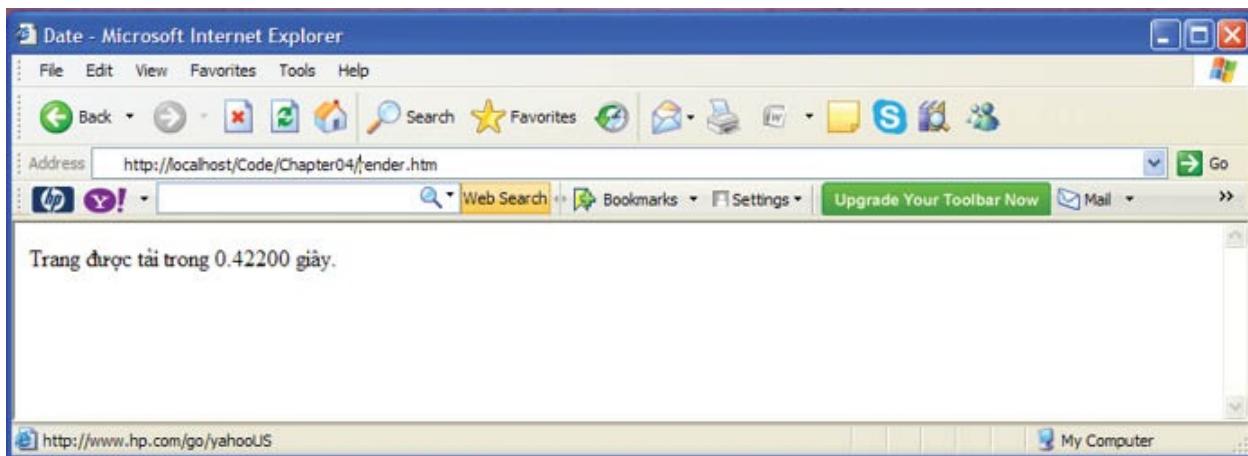
kiểm thử trang web). Sử dụng vòng lặp *for* là cách đơn giản và nhanh chóng nhất để làm điều này:

```
for (var i = 0; i < 1000000; i++) {  
//khi vòng lặp chạy sẽ tạo độ trễ  
})
```

Giá trị 1000000 là tùy ý. Bạn có thể chọn một giá trị lớn hoặc bé hơn để tạo độ trễ mong muốn. Đoạn mã cuối cùng sẽ như sau:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-tran  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <title>Date</title>  
    <script type="text/javascript">  
      var started = new Date();  
      var now = started.getTime();  
      for (var i = 0; i < 1000000; i++) {  
        //khi vòng lặp chạy sẽ tạo độ trễ  
      }  
    </script>  
  </head>  
  <body>  
    <p id="dateField"> </p>  
    <script type="text/javascript">  
      var bottom = new Date();  
      var diff = (bottom.getTime()-now)/1000;  
      var finaltime = diff.toPrecision(5);  
      var dateLoc = document.getElementById("dateField");  
      dateLoc.innerHTML = "Trang được tải  
    </script>  
  </body>  
</html>
```

1. Lưu và mở lại bằng trình duyệt. Bạn sẽ thấy giá trị hiển thị là số dương.



Trên thực tế, để tính toán chính xác thời gian tải về một trang, hãy đặt một biến ở đầu trang và một biến ở gần cuối trang.

Chú ý là phương thức `now()` của đối tượng `Date` cũng có thể được dùng để thay thế cho `getTime()`.

Bạn vừa học một vài phương thức trong khoảng hơn 40 phương thức của đối tượng `Date`. Nhiều phương thức trong số đó sử dụng phiên bản giờ UTC (Coordinated Universal Time), nghĩa là chúng ưu tiên thiết lập và khởi tạo ngày và giờ UTC hơn là giờ địa phương. Bảng 4-4 liệt kê các phương thức của đối tượng `Date`. Ngoại trừ `getTime()` và `getTimezoneOffset()`, tất cả các phương thức sử dụng giờ UTC được đặt tên theo định dạng như `getUTCDate()` hoặc `getUTCDay()`.

BẢNG 4-4 Các phương thức `get` của đối tượng `Date`

| Phương thức | Mô tả |
|--------------------------------|---|
| <code>getDate()</code> | Trả về ngày trong tháng |
| <code>getDay()</code> | Trả về ngày trong tuần |
| <code>getFullYear()</code> | Trả về năm có 4 chữ số, được khuyến khích dùng trong hầu hết các trường hợp |
| <code>getHours()</code> | Trả về giờ của đối tượng <code>date</code> |
| <code>getMilliseconds()</code> | Trả về mili giây của đối tượng <code>date</code> |
| <code>getMinutes()</code> | Trả về phút của đối tượng <code>date</code> |
| <code>getSeconds()</code> | Trả về giây của đối tượng <code>date</code> |

|`getTime()`|Trả về mili giây kể từ mốc 1/1/1970|

|`getTimezoneOffset()`|Trả về chênh lệch (theo phút) giữa giờ UTC và giờ địa

phương

Hầu hết các phương thức `get...()` đều có phương thức `set...()` tương ứng, như trong Bảng 4-5. Cũng như `get`, hầu hết các phương thức `set...()` sử dụng giờ UTC, trừ `setTime()`.

BẢNG 4-5 Các phương thức `set` của đối tượng `Date`.

| Phương thức | Mô tả |
|---------------------------------|---|
| <code> setDate()</code> | Thiết lập ngày trong tháng của một đối tượng date. |
| <code> setFullYear()</code> | Thiết lập năm (định dạng 4 ký tự) cho đối tượng date. Chấp nhận cả các số nguyên tháng và ngày trong tháng. |
| <code> setHours()</code> | Thiết lập giờ cho một đối tượng date. |
| <code> setMilliseconds()</code> | Thiết lập mili giây cho đối tượng date. |
| <code> setMinutes()</code> | Thiết lập phút cho đối tượng date. |
| <code> setMonth()</code> | Thiết lập tháng là một số nguyên cho đối tượng date. |
| <code> setSeconds()</code> | Thiết lập giây cho đối tượng date. |
| <code> setTime()</code> | Thiết lập thời gian tính bằng mili giây kể từ 1/1/1970. |

Có nhiều phương thức chuyển đổi đối tượng `Date` sang dạng chuỗi ở nhiều định dạng khác nhau. Bạn cũng đã thực hành một số phương thức như `toLocaleDateString()`. Các phương thức tương tự khác bao gồm `toLocaleString()`, `toGMTString()`, `toLocaleTimeString()`, `toString()`, `toISOString()`, `toDateString()`, `toUTCString()`, và `toTimeString()`. Hãy thoải mái thực hành với chúng, chỉ lưu ý là `toISOString()` là một phương thức mới trong đặc tả ECMA-262 phiên bản 5 và một vài trình duyệt có thể chưa hỗ trợ (hầu hết các phiên bản IE đều chưa hỗ trợ). Ví dụ đơn giản sau giúp bạn bắt đầu quá trình thử nghiệm các hàm này. Hãy nhập chúng vào thanh địa chỉ trên trình duyệt:

```
javascript:var myDate = new Date(); alert(myDate.toLocaleDateString());  
javascript:var myDate = new Date(); alert(myDate.toLocaleString());  
javascript:var myDate = new Date(); alert(myDate.toGMTString());  
javascript:var myDate = new Date(); alert(myDate.toLocaleTimeString());  
javascript:var myDate = new Date(); alert(myDate.toString());  
javascript:var myDate = new Date(); alert(myDate.toISOString());  
javascript:var myDate = new Date(); alert(myDate.toDateString());  
javascript:var myDate = new Date(); alert(myDate.toUTCString());  
javascript:var myDate = new Date(); alert(myDate.toTimeString());
```

Bạn có thể viết đoạn mã này mà không cần tạo biến *myDate* như sau:

```
javascript: alert(new Date().toUTCString());
```

Sử dụng đối tượng *RegExp*

Biểu thức chính quy là các cú pháp bạn sử dụng để so khớp và thao tác trên các chuỗi. Nếu bạn từng làm việc với giao diện dòng lệnh trong Microsoft Windows hoặc Linux/Unix, bạn có thể đã quen với cách tìm kiếm các file bằng cách so tên với một dấu hoa thị, hoặc sao (*), như sau:

```
dir *.*
```

hoặc:

```
dir *.txt
```

Nếu như bạn từng làm việc với ký tự đại diện như *hay ?* thì bạn sẽ thấy quen thuộc với *biểu thức chính quy*. Trên thực tế, *dấu* và *?* cũng là các kí tự được dùng trong biểu thức chính quy.



JavaScript cung cấp công cụ rất mạnh để làm việc với chuỗi ký tự và các chữ số thông qua việc sử dụng đối tượng *RegExp* và các ký tự đại diện trong biểu thức chính quy. Phiên bản trình phân tích biểu thức chính quy theo chuẩn ECMA-262 chủ yếu vay mượn từ trình phân tích biểu thức chính quy của Perl 5. Ví dụ sau là biểu thức chính quy để so khớp từ JavaScript:

```
var myRegex = /JavaScript/;
```

Biểu thức chính quy ở trên so khớp chuỗi JavaScript ở bất kỳ chỗ nào mà nó xuất hiện. Ví dụ, nó sẽ khớp trong câu “Đây là sách về JavaScript” và chuỗi “DaylasachveJavaScript” nhưng sẽ không khớp với “Đây là sách về javascript”, vì biểu thức chính quy có phân biệt chữ hoa và chữ thường. (Bạn sẽ học cách thay đổi điều này ở cuối chương).

Cú pháp của biểu thức chính quy

Các biểu thức chính quy trong JavaScript thường được rút gọn và có cú pháp

khó hiểu. Tuy nhiên, bạn không phải lo ngại về chuyện này vì đó thực chất là thẻ mạnh của biểu thức chính quy. Ví dụ, biểu thức chính quy sau tìm các chữ số và sau đó định dạng lại thành khối địa chỉ IP (Internet Protocol) ở dạng 192.168.0/24 bằng cách gom nhóm. Đây không phải là một ví dụ phức tạp về biểu thức chính quy. (Nó là một phần của đoạn mã Perl được dùng để phân tích danh sách mạng trong tường lửa của Trung tâm Thông tin Mạng châu Á Thái Bình Dương- APNIC).

```
s/([0-9]+)\.([0-9]+)(/[0-9]+)/$1.$2\.0$3/;
```

Biểu thức chính quy tương tự có thể được viết trong JavaScript bằng cách sử dụng hàm *replace* của đối tượng *string*, như sau:

```
var theIP = "192.168.0/28";
alert(theIP.replace(/([0-9]+)\.([0-9]+)(/[0-9]+)/, "$1.$2\.0$3");
```

Cú pháp của biểu thức chính quy bao gồm rất nhiều ký tự có ý nghĩa đặc biệt, bao gồm cả các ký tự đánh dấu so khớp để bắt đầu hoặc kết thúc một chuỗi, ký tự đại diện, ký tự nhóm và một số ký tự khác. Bảng 4-6 trình bày các ký tự này.

BẢNG 4-6 Những ký tự đặc biệt phổ biến trong biểu thức chính quy của JavaScript.

| Ký tự | Miêu tả |
|-------|---|
| ^ | So khớp với bất kỳ chuỗi nào bắt đầu bằng chuỗi đó. Ví dụ ^n: Bất kỳ chuỗi nào bắt đầu bằng ký tự n. |
| \\$ | So khớp với bất kỳ chuỗi nào kết thúc bằng chuỗi đó. Ví dụ n\$: Bất kỳ chuỗi nào kết thúc bằng ký tự n. |
| . | Khớp với bất kỳ ký tự nào. |
| * | Khớp với ký tự trước nó 0 hoặc nhiều lần. Đây là một ký tự đại diện. |
| + | Khớp với ký tự trước nó 1 hoặc nhiều lần. |
| ? | Khớp với ký tự trước nó 0 hoặc một lần. |
| () | Nhóm các ký tự khớp vào trong cặp dấu ngoặc đơn để sử dụng sau. |
| {n} | Khớp với ký tự trước nó ít nhất n lần. |
| {n,m} | Khớp với ký tự trước nó ít nhất n lần nhưng không quá m lần. |
| [] | Định nghĩa một tập ký tự để tiến hành so khớp với bất kỳ ký tự nào nằm trong tập này. Ký tự này có thể sử dụng dài như 0-9 để so khớp các chữ số hoặc a-z để so khớp các ký tự. |
| [^] | Việc sử dụng dấu ^ trong một tập ký tự sẽ phủ định tập ký tự đó, nghĩa là các ký tự bên trong tập không thể xuất hiện trong chuỗi so khớp. |

Được sử dụng như một ký tự thoát, nghĩa là những gì theo sau dấu gạch chéo ngược được coi là

một ký tự thông thường thay vì có ý nghĩa đặc biệt. Dấu cũng có thể được sử dụng để khai báo các bộ ký tự đặc biệt, được liệt kê trong Bảng 4-7.

Ngoài ký tự đặc biệt, còn nhiều chuỗi ký tự được dùng để so khớp với các nhóm ký tự hoặc các ký tự không phải chữ số. Một số chuỗi ký tự như vậy được trình bày trong Bảng 4-7.

BẢNG 4-7 Những ký tự đặc biệt phổ biến trong biểu thức chính quy của JavaScript

| Ký tự | Chuỗi so khớp |
|-------|--|
| b | Tìm và so khớp chuỗi nằm sau ký tự b với những ký tự ở vị trí bắt đầu và kết thúc của một từ. Ví dụ: bW3 sẽ khớp với chuỗi W3 trong từ W3School. |
| B | Tìm và so khớp chuỗi nằm sau ký tự B với những ký tự không nằm ở vị trí bắt đầu và kết thúc của một từ. Ví dụ: chuỗi School trong từ W3School sẽ so khớp với biểu thức chính quy BSchool |
| c | Ký tự Control khi sử dụng kết hợp với ký tự khác. Ví dụ: cA là ký tự thoát cho Control-A. |
| d | Chữ số |
| D | Không phải là chữ số. |
| n | Xuống dòng. |
| r | Về đầu dòng. |
| s | Ký tự khoảng trắng (dấu cách hoặc tab). |
| S | Một ký tự không phải là khoảng trắng. |
| t | Dấu tab |
| w | Ký tự chữ hoặc số |
| W | Ký tự không phải là chữ hay số. |



Ngoài các ký tự trong Bảng 4-7, bạn có thể sử dụng ký tự *i* để quy định biểu thức chính quy không phân biệt chữ hoa chữ thường và *g* để xác định nó sẽ tiếp tục so khớp sau khi tìm thấy kết quả trùng khớp đầu tiên.

Đối tượng *RegExp* có các phương thức riêng, bao gồm *exec* và *test*, phương thức *test* kiểm tra một biểu thức chính quy trên một chuỗi và trả về kết quả *true* (đúng) hay *false* (sai) dựa trên kết quả so khớp biểu thức chính quy với chuỗi đó. Tuy nhiên, cách phổ biến khi làm việc với các biểu thức chính quy là sử dụng các hàm quen thuộc của kiểu dữ liệu chuỗi như *match*, *search*, *replace*.

Phương thức *exec()* của đối tượng *RegExp* được sử dụng để phân tích biểu thức chính quy trên một chuỗi và trả về kết quả. Ví dụ sau phân tích một URL

đơn giản và trích xuất ra tên miền:

```
var myString = "http://www.braingia.org";
var myRegex = /http:\//\w+.(.*);
var results = myRegex.exec(myString);
alert(results[1]);
```

Kết quả của đoạn mã này là hộp thoại hiển thị tên miền, như trong Hình 4-8.



Hình 4-8 Phân tích URL sử dụng biểu thức chính quy.

Hãy phân tích tuần tự từng dòng mã. Đầu tiên bạn có một khai báo chuỗi:

```
var myString = "http://www.braingia.org";
```

Sau đó là khai báo một biểu thức chính quy và gọi hàm `exec()`, kết quả phân tích chứa trong biến `results`.

```
var myRegex = /http:\//\w+.(.*);
var results = myRegex.exec(myString);
```

Biểu thức chính quy trên chứa một số thành phần quan trọng. Nó bắt đầu bằng chuỗi `http:`, thêm hai dấu gạch chéo, nhưng vì dấu gạch chéo (`/`) là ký tự đặc biệt trong biểu thức chính quy, bạn phải thoát chúng bằng cách sử dụng dấu gạch chéo ngược (`\`), tạo thành `http:_/_`.

Phần tiếp theo của biểu thức chính quy, `\w`, dùng để tìm kiếm bất kỳ ký tự chữ hoặc số nào. Địa chỉ web thông thường chứa `www`, vì vậy đừng nhầm lẫn rằng `w` đi tìm 3 ký tự `w`. Host (nơi lưu trữ website) trong ví dụ này có thể có tên là `web`, `host1`, `myhost`, hoặc `www`, như chuỗi trong đoạn mã bạn đang xem. Vì `\w` so khớp với một ký tự và host thường có tối 3 ký tự (`www`), nên biểu thức chính quy có thêm một ký tự đặc biệt `+` để chỉ ra rằng biểu thức chính quy phải tìm ký tự chữ số ít nhất một lần. Bây giờ đoạn mã là `http:\//\w+` sẽ so khớp với địa chỉ

tên miền `http://www`

Bây giờ cần tính đến dấu chấm giữa tên host (`www`) và tên miền (`braingia.org`). Bạn sẽ hoàn thiện bằng cách thêm một dấu chấm (.), nhưng vì dấu chấm cũng là một ký tự đặc biệt, bạn cần dùng ký tự thoát \. Vậy là chúng ta có `http : \ \ / w + \ .`, có thể so khớp với tất cả các thành phần của một địa chỉ tên miền thông thường.

Cuối cùng, bạn cần giữ lại tên miền để sử dụng sau này, vì vậy hãy đặt tên miền vào trong dấu ngoặc đơn. Vì bạn không cần quan tâm tới tên miền là gì hoặc cái gì xuất hiện sau nó, hãy sử dụng thêm hai ký tự đặc biệt: dấu chấm để so khớp bất kỳ ký tự nào và dấu hoa thị để so khớp một hoặc tất cả các ký tự trước nó. Bạn đã có biểu thức chính quy cuối cùng, được sử dụng trong hàm `exec()`. Kết quả được lưu vào biến `results`.

Nếu tìm được chuỗi so khớp, kết quả trả về từ hàm `exec()` sẽ là một mảng chứa toàn bộ chuỗi này và chỉ số cho từng phần so khớp của biểu thức. Chỉ số thứ hai (1) được truyền vào hộp thoại thông báo, như hiển thị trong Hình 4-8.

```
alert(results[1]);
```



Ví dụ này đã khá đầy đủ, song biểu thức chính quy này còn có thể được cải tiến hơn bằng cách thêm những ký tự khác để đánh dấu so khớp và để đại diện cho những ký tự sau tên miền cũng như những ký tự không phải chữ hay số trong phần tên host. Tuy nhiên, để ví dụ đơn giản hơn chúng ta nên dùng phương thức so khớp ít nghiêm ngặt hơn.

Kiểu đối tượng chuỗi chứa ba phương thức để so khớp và làm việc với các chuỗi, và chúng sử dụng biểu thức chính quy để làm điều này. Tất cả các phương thức `match`, `replace` và `search` đều sử dụng mẫu biểu thức chính quy để so khớp. Vì bạn vừa được học về biểu thức chính quy, nên đây là thời điểm thích hợp để giới thiệu những phương thức này.

Phương thức `match` trả về một mảng với thông tin tương tự như hàm `exec` của kiểu `RegExp`. Đây là một ví dụ:

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\ .com/;
var checkMatch = emailAddr.match(myRegex);
alert(checkMatch[0]); //Kết quả trả về.com
```

Bạn có thể cải tiến đoạn mã trên sử dụng trong một biểu thức điều kiện để xác định địa chỉ email có chứa chuỗi `.com` hay không:

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\.com/;
var checkMatch = emailAddr.match(myRegex);
if (checkMatch !== null) {
    alert(checkMatch[0]); //Kết quả trả về .com
}
```

Phương thức `search` làm việc tương tự như hàm `match` nhưng chỉ trả về chỉ số (vị trí) của kết quả so khớp đầu tiên như sau:

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\.com/;
var searchResult = emailAddr.search(myRegex);
alert(searchResult); //Kết quả trả về 17
```

Nếu chuỗi không khớp với biểu thức chính quy, hàm `search` trả về `-1`.

Phương thức `replace` thay thế một chuỗi bằng chuỗi khác nếu một so khớp được tìm thấy. Giả sử như trong ví dụ về địa chỉ email, bạn muốn thay đổi địa chỉ `.com` thành `.net`, bạn có thể sử dụng hàm `replace` như sau:

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\.com$/;
var replaceWith = ".net";
var result = emailAddr.replace(myRegex, replaceWith);
alert(result); //Kết quả trả về suehring@braingia.net
```

Nếu không tìm thấy chuỗi con so khớp, chuỗi gốc sẽ được truyền vào biến `result`; ngược lại, chuỗi mới sau khi thay thế sẽ được trả về.

Chú ý Bạn có thể sử dụng nhiều ký tự đặc biệt để thực hiện thay thế. Vui lòng xem đặc tả ECMA-262 để có thêm thông tin về các ký tự này.

Các chương sau trình bày nhiều ví dụ về các phương thức liên quan tới biểu thức chính quy của kiểu chuỗi. Bạn có thể sử dụng chương này để tham khảo cách dùng các ký tự đặc biệt trong biểu thức chính quy.

Tham chiếu và dọn dẹp bộ nhớ

Một số biến hoặc giá trị của chúng là kiểu dữ liệu cơ sở, trong khi một số khác là các kiểu dữ liệu tham chiếu. Ban đầu bạn có thể không để tâm tới điều này.

Nhưng mọi chuyện sẽ thay đổi ngay khi bạn bắt gặp một hành vi lạ với biến vừa được sao chép.

Có thể giải thích như sau: các đối tượng, mảng và hàm là *kiểu tham chiếu*, còn kiểu số, logic, *null* và *undefined* là *kiểu tham trị*. Theo đặc tả ECMA-262, còn có các kiểu dữ liệu cơ bản khác là kiểu số và kiểu chuỗi, nhưng kiểu chuỗi không nằm trong phạm vi thảo luận này.

Khi một số được sao chép, bạn mong đợi số sao chép và số gốc có cùng giá trị. Tuy nhiên, nếu bạn thay đổi biến gốc, biến sao chép sẽ không bị ảnh hưởng. Đây là một ví dụ:

```
// Gán cho biến myNum giá trị là 20.  
var myNum = 20;  
// Tạo một biến khác, anotherNum, và sao chép nội dung của b  
// Bây giờ, cả anotherNum và myNum đều là 20  
var anotherNum = myNum;  
// Thay đổi giá trị của myNum thành 1000.  
myNum = 1000;  
// Hiển thị nội dung của cả 2 biến  
// Chú ý là nội dung của biến anotherNum chưa được thay đổi  
alert(myNum);  
alert(anotherNum);
```

Hộp thoại thông báo sẽ hiển thị tương ứng là **1000** và **20**. Sau khi biến *anotherNum* sao chép giá trị từ biến *myNum*, nó sẽ giữ mãi giá trị này mà không bị ảnh hưởng bởi những thay đổi của biến *myNum* sau đó. Lý do là vì kiểu số là kiểu dữ liệu cơ bản trong JavaScript.

Xem xét một ví dụ đối nghịch trên biến kiểu tham chiếu như sau:

```
// Tạo một mảng với 3 phần tử số tên là myNumbers  
var myNumbers = [20, 21, 22];  
// Tạo một bản sao chép mới của myNumbers là một biến tên co  
var copyNumbers = myNumbers;  
// Thay đổi giá trị đầu tiên của myNumbers thành 1000.  
myNumbers[0] = 1000;
```

```
// hộp thoại thông báo
alert(myNumbers);
alert(copyNumbers);
```

Lần này, vì mảng là kiểu tham chiếu, cả hai hộp thoại cảnh báo hiển thị `1000,21,22`, mặc dù chỉ có biến `myNumbers` bị thay đổi trực tiếp trong đoạn mã. Bài học rút ra là phải ghi nhớ rằng các biến đối tượng, mảng và hàm là thuộc loại tham chiếu, do đó bất kỳ sự thay đổi nào của biến gốc cũng làm thay đổi các biến sao chép.

Một vấn đề khác cũng liên quan đến sự khác nhau giữa kiểu dữ liệu cơ bản và kiểu dữ liệu tham chiếu là việc dọn dẹp bộ nhớ. *Dọn dẹp bộ nhớ (garbage collection)* ám chỉ việc hủy các biến không sử dụng của trình thông dịch JavaScript để tiết kiệm bộ nhớ. Khi một biến không còn được sử dụng trong chương trình, trình thông dịch giải phóng bộ nhớ để tái sử dụng. Điều tương tự cũng diễn ra trong Java Virtual Machine hoặc .NET Common Language Runtime.

Việc tự động giải phóng bộ nhớ trong JavaScript khác với cách mà các ngôn ngữ khác như C++ sử dụng. Trong các ngôn ngữ đó, lập trình viên phải thực thi việc dọn dẹp bộ nhớ thủ công. Đây là tất cả những gì bạn cần biết về dọn dẹp bộ nhớ.

Tìm hiểu các loại chuyển đổi kiểu dữ liệu

Trước khi khép lại phần thảo luận về biến và kiểu dữ liệu, bạn cũng nên biết một chút về *chuyển đổi kiểu dữ liệu*. JavaScript thường thực thi ngầm việc chuyển đổi kiểu cho bạn, tuy nhiên trong nhiều trường hợp, bạn nên dùng cách ép kiểu hay chuyển đổi tường minh.

Chuyển đổi kiểu số

Bạn đã thấy việc chuyển đổi hai định dạng số, từ hệ thập lục phân về hệ thập phân, trong ví dụ ở phần “Kiểu dữ liệu trong JavaScript” ở phần đầu của chương. Bạn cũng có thể chuyển đổi số thành chuỗi theo cách tương tự. JavaScript chuyển đổi ngầm một số sang chuỗi khi số đó được sử dụng trong

ngữ cảnh một chuỗi.

Để chuyển đổi tường minh một số thành chuỗi, ép kiểu nó thành một chuỗi, như trong ví dụ sau:

```
// Chuyển đổi myNumString sang một chuỗi với giá trị 100
var myNumString = String(100);
```

Chuyển đổi kiểu chuỗi

Tương tự như cách chuyển số thành chuỗi, bạn cũng có thể làm điều ngược lại bằng cách ép kiểu chuỗi thành số. (Xem thêm ví dụ này trong file stringconversion.txt của phần Tài nguyên đi kèm.)

```
var myNumString = "100";
var myNum = Number(myNumString);
```

Mách nhỏ JavaScript tự chuyển đổi chuỗi thành số khi chuỗi này được sử dụng trong ngữ cảnh của số. Tuy nhiên, trên thực tế, việc chuyển đổi ngầm này nhiều khi không hiệu quả và tốt nhất là bạn nên chuyển đổi tường minh khi cần. Tất nhiên điều này khiến chương trình của bạn dài ra, nhưng như thế vẫn tốt hơn là phụ thuộc vào trình thông dịch JavaScript.

Chuyển đổi kiểu logic

Kiểu logic được tự động chuyển sang kiểu số khi sử dụng trong ngữ cảnh một số. Giá trị của *true* là *1* và *false* là *0*. Khi được sử dụng trong ngữ cảnh một chuỗi, *true* trở thành "*true*", và *false* trở thành "*false*". Dùng hàm *Boolean()* nếu bạn cần chuyển đổi chính xác số hoặc chuỗi sang giá trị logic.

Bài tập

1. Khai báo ba biến - một số và hai chuỗi. Số có giá trị 120 và chuỗi là “5150” và “Hai trăm ba mươi”.
2. Tạo một mảng mới với ba phần tử số và hai phần tử chuỗi.
3. Sử dụng hàm `alert()` để hiển thị chuỗi sau, nhớ sử dụng ký tự thoát: Phản

ứng của Steve thật “Dễ thương!” .

4. Sử dụng Firebug để kiểm tra ba website ưa thích của bạn. Xem kỹ những lỗi JavaScript mà Firebug báo cáo. Sử dụng IE để mở các website này và thử gỡ lỗi bằng cách sử dụng các công cụ dành cho IE và những công cụ liên quan khác.



Chương 5

Sử dụng toán tử và biểu thức

Sau khi đọc xong chương này, bạn có thể:

- Hiểu được các toán tử có trong JavaScript.
- Sử dụng các toán tử JavaScript để thực hiện tính toán, kiểm tra đăng ký, kiểm tra các quan hệ và các phép gán.
- Sử dụng toán tử void để mở cửa sổ mới bằng một liên kết.

Tìm hiểu về các toán tử

Chuẩn ECMA-262 phân loại toán tử thành nhiều dạng khác nhau. Bao gồm:



- Toán tử cộng
- Toán tử nhân
- Toán tử thao tác bit
- Toán tử bằng
- Toán tử quan hệ
- Toán tử một ngôi
- Toán tử gán

Các toán tử có thể được sử dụng cho giá trị chữ, biến cũng như các đối tượng khác trong JavaScript.

Toán tử cộng

Thuật ngữ *toán tử cộng* bao hàm cả toán tử cộng và trừ. Thuật ngữ này nghe

có vẻ như đã bị đặt nhầm nhưng như tất cả chúng ta đều biết, phép trừ chỉ là phép cộng với số âm. Ký hiệu tương ứng cho toán tử cộng và trừ là + và - Sau đây là một vài ví dụ về cách thức dùng hai toán tử này.

Chú ý Bạn có thể tìm các ví dụ này trong file additiveops.txt của phần Tài nguyên đi kèm.

```
4 + 5; // Kết quả là 9.
```

```
x + y; // Cộng x và y với nhau.
```

```
5 - 1; // Kết quả là 4.
```

Toán tử cộng được thực thi theo nhiều cách tùy theo loại giá trị được cộng vào. Khi cộng hai chuỗi với nhau, toán tử cộng ghép đổi số bên trái và bên phải với nhau. Khi cộng khác loại, bạn có thể nhận được kết quả kỳ lạ vì JavaScript cần phải chuyển đổi một trong các loại đấy trước khi thực hiện toán tử cộng (hoặc các toán tử khác). Ví dụ, bạn sẽ không nhận được kết quả mong muốn khi bạn nghĩ bạn có một biến số nhưng trình thông dịch JavaScript lại hiểu đó là chuỗi. Dưới đây là một vài ví dụ cụ thể:

```
var aNum = 947;
var aStr= "Rush";
var anotherNum = 53;
var aStrNum = "43";
var result1 = aNum + aStr; // Kết quả sẽ là chuỗi "947Rush";
var result2 = aNum + anotherNum; // result2 sẽ là số 1000;
var result3 = aNum + aStrNum; // result3 sẽ là số 94743;
```

Như đã thảo luận ở Chương 4 “Làm việc với biến và kiểu dữ liệu”, trong nhiều trường hợp, bạn có thể thay đổi trực tiếp hoặc chuyển đổi kiểu dữ liệu này sang kiểu dữ liệu khác trong JavaScript. Hãy xem lại biến *result3* trong ví dụ trước. Có lẽ bạn muốn *result3* lưu giá trị kết quả tính toán của $947+43$. Nhưng vì giá trị thứ hai, *aStrNum*, là một chuỗi nên biểu thức ghép hai giá trị lại thay vì dùng phép cộng toán học cho các con số. Tuy nhiên, nếu ta dùng hàm *ToString()* chuyển đổi *aStrNum* sang kiểu dữ liệu số, câu lệnh sẽ thực thi biểu thức toán học, trong trường hợp này là phép cộng. Đây là đoạn mã đã được sửa lại theo hướng bạn muốn:

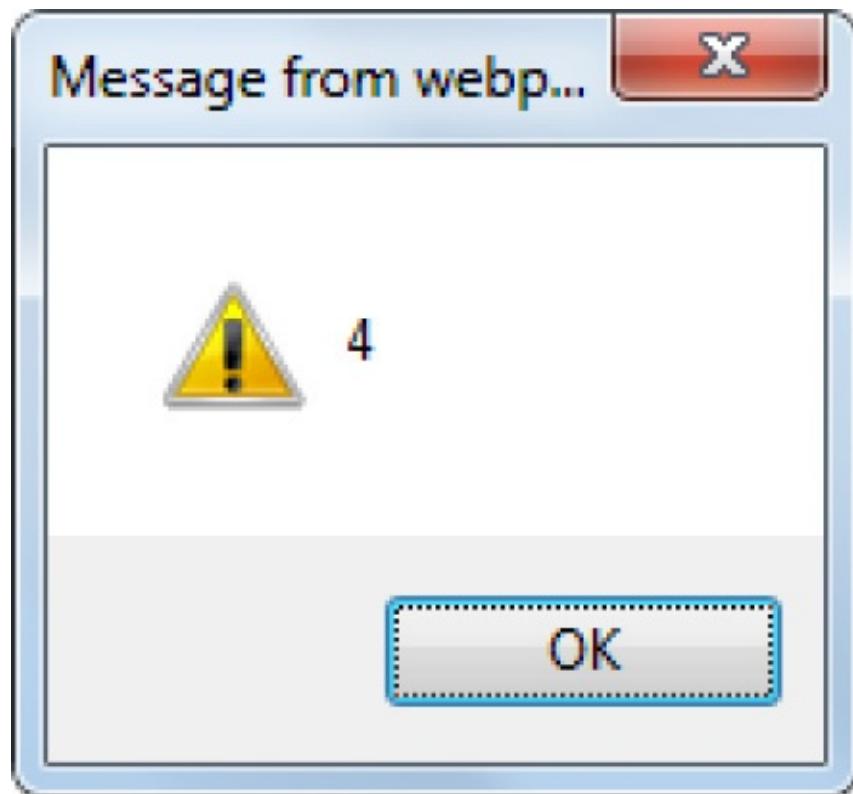
```
var aNum = 947;
```

```
var aStrNum = ToNumber("43");
var result3 = aNum + aStrNum; // result3 sẽ là 990;
```

Toán tử nhân

Cũng như toán tử cộng, toán tử nhân thực hiện phép tính nhân và chia. Toán tử nhân (*) nhân hai số với nhau, còn toán tử chia (/) thì chia các số cho nhau. Đây là một ví dụ về toán tử nhân và kết quả đầu ra của nó:

```
javascript:alert(2 * 2);
```



Toán tử nhân còn bao gồm toán tử mô-đun, được thể hiện bằng ký tự phần trăm (%). Toán tử mô-đun cho ra kết quả là phần dư của phép chia hai số với nhau. Ví dụ, phần dư của 4 chia cho 3 là 1, được thể hiện ở đoạn mã tiếp theo:

Chú ý Bạn có thể tìm các ví dụ này trong file multiplicativeops.txt của phần Tài nguyên đi kèm.

```
javascript:alert(4 % 3);
```

Kết quả được hiển thị ở đây:



Toán tử thao tác bit

Toán tử thao tác bit bao gồm **AND**, **OR**, **XOR**, **NOT**, **Shift Left**, **Shift Right With Sign**, và **Shift Right With Zero Fill**. Mỗi toán tử được thể hiện bởi một hoặc một vài ký tự như trong Bảng 5-1.

BẢNG 5-1 Toán tử thao tác bit.

| Toán tử | Nghĩa |
|---------|-------|
| & | AND |
| | OR |
| ^ | XOR |
| ~ | NOT |

| | |
|-----|----------------------------|
| << | Shift Left |
| >> | Shift Right With Sign |
| >>> | Shift Right With Zero Fill |

Cuốn sách này không đề cập chi tiết về toán tử thao tác bit, tuy nhiên toán tử này vẫn sẽ được đề cập kỹ hơn ở các chương sau. Bạn có thể tham khảo thêm thông tin về toán tử thao tác bit trong tài liệu của chuẩn ECMA-262.

Toán tử bằng

Chúng ta dùng toán tử bằng để kiểm tra xem hai biểu thức giống hay khác nhau. Các toán tử này luôn trả về kiểu dữ liệu Boolean: *true* (*đúng*) hoặc *false* (*sai*).

BẢNG 5-2 Toán tử bằng.

| Toán tử | Nghĩa |
|-------------------|---|
| <code>==</code> | Bằng. |
| <code>!=</code> | Không bằng. |
| <code>====</code> | Bằng, sử dụng phương pháp chặt chẽ hơn. |
| <code>!==</code> | Không bằng, sử dụng phương pháp chặt chẽ hơn. |

Bảng 5-2 này cho thấy chúng ta có thể kiểm tra điều kiện bằng hoặc không bằng theo hai cách khác nhau. Những cách thức này khác nhau ở mức độ nghiêm ngặt - mức độ để xác định hai giá trị có thực sự bằng nhau hay không. Nghiêm ngặt hơn toán tử bằng (`==`) là toán tử bằng (`====`). Toán tử này không chỉ yêu cầu giá trị của biểu thức bằng nhau, mà kể cả kiểu dữ liệu cũng phải giống hệt nhau. Kiểm tra nghiêm ngặt sẽ xác định chuỗi với giá trị là "42" không bằng với số có giá trị 42, trong khi kiểm tra ít nghiêm ngặt hơn sẽ cho kết quả là hai giá trị này bằng nhau. Ví dụ dưới đây sẽ giúp các bạn hiểu rõ hơn.

Kiểm tra toán so sánh tử bằng

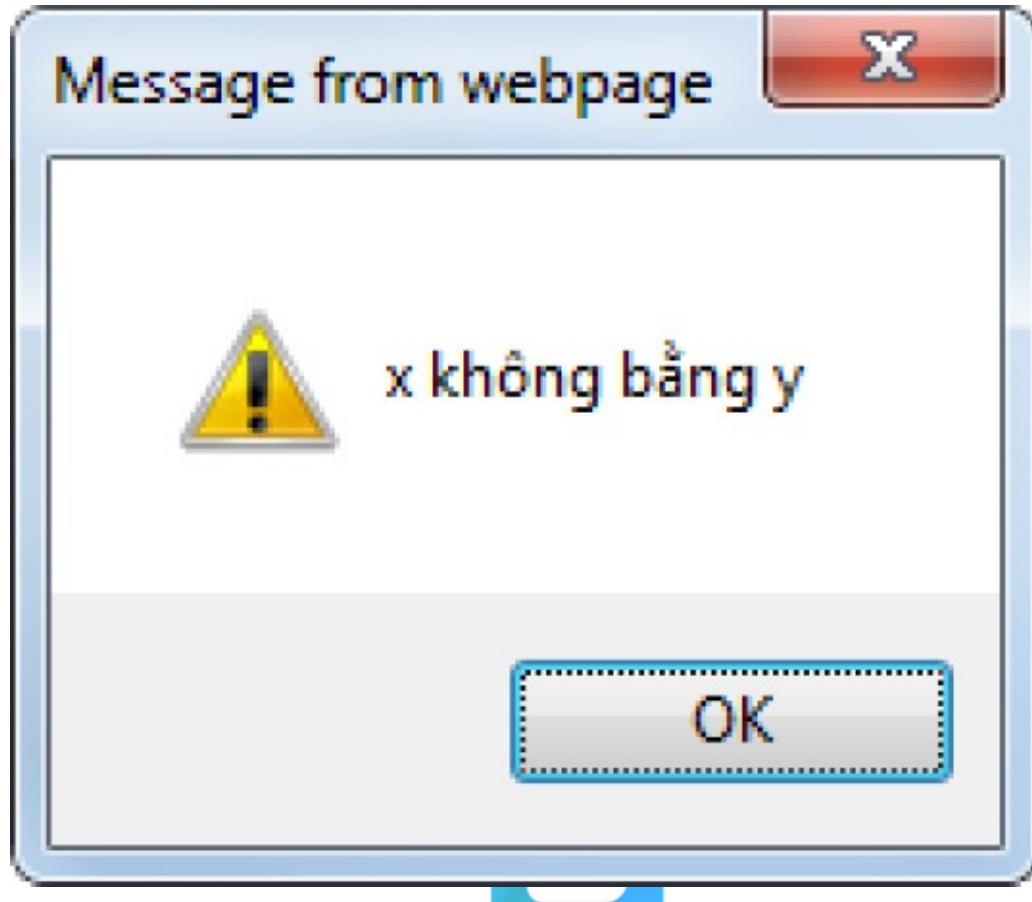
1. Dùng Visual Studio, Eclipse hoặc một trình soạn thảo khác thay đổi file equality.htm trong thư mục mã mẫu Chương 5, phần Tài nguyên đi kèm.

2. Trên trang web, thay thế dòng chú thích TODO bằng đoạn mã in đậm dưới đây. (Có thể tìm thấy đoạn mã này trong file equality.txt).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
    <head>  
        <title>Toán tử bằng</title>  
        <script type="text/javascript">  
            var x = 42;  
            var y = "42";  
            if (x == y) {  
                alert("x bằng y qua phép kiểm thử")  
            } else {  
                alert("x không bằng y")  
            }  
        </script>  
    </head>  
    <body>  
    </body>  
</html>
```



3. Dùng trình duyệt mở trang vừa tạo. Đoạn mã này khá rõ ràng. Đoạn mã khai báo hai biến `x` và `y`. Biến `x` được lưu giá trị là số `42`, và `y` được lưu giá trị là chuỗi `"42"` (chú ý đến dấu nháy kép). Kiểm tra ở cấp đơn giản, sử dụng `==`. Loại kiểm tra này chỉ so sánh giá trị và bỏ qua việc xem xét sự giống và khác nhau trong kiểu dữ liệu của biến. Khối lệnh `if` gọi hàm `alert()` thích hợp dựa trên kết quả. Bạn có thể sẽ thấy một thông báo như sau:

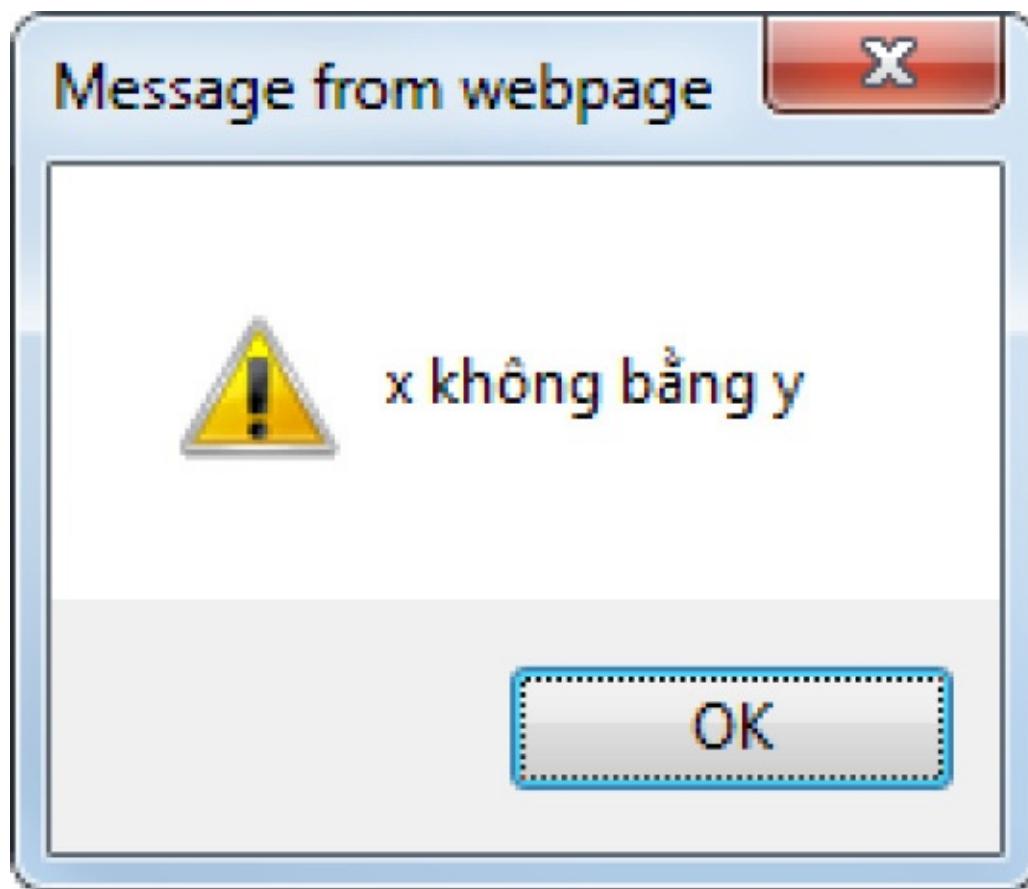


4. Thay đổi cách thức kiểm tra toán tử bằng bằng phương thức kiểm tra nghiêm ngặt. Để làm được điều này, đầu tiên phải thay đổi toán tử bằng ở trên và sử dụng toán tử bằng ở chế độ nghiêm ngặt (đó là `==`), sau đó thay đổi hàm `alert` để đọc *nghiêm ngặt* thay vì *đơn giản*. Dưới đây là đoạn mã đầy đủ (dòng thay đổi được in đậm và nằm trong file equality2.txt của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Toán tử bằng</title>  
    <script type="text/javascript">  
        var x = 42;  
        var y = "42";  
        if (x == y) {  
            alert("x bằng y qua kiểm thử nghiêm  
} else {  
            alert("x không bằng y");
```

```
    }
  </script>
</head>
<body>
</body>
</html>
```

5. Một lần nữa dùng trình duyệt mở trang. Bây giờ, bạn sẽ sử dụng cách thức nghiêm ngặt hơn để kiểm tra tương đương, ===. Kiểm tra nghiêm ngặt hơn giống kiểm tra đơn giản ở cách thức kiểm tra dữ liệu, nhưng khác nhau là nó kiểm tra cả kiểu dữ liệu của biến. Hàm `alertView()` thích hợp được gọi dựa trên kết quả kiểm tra. Và lần này hộp thoại thông báo sẽ có dạng như sau:



Toán tử quan hệ

Toán tử quan hệ so sánh các biểu thức với nhau, hoặc kiểm tra giá trị cho trước

có nằm trong một danh sách hay không hoặc thuộc một kiểu dữ liệu nào đó hay không. Bảng 5-3 liệt kê các toán tử quan hệ trong JavaScript.

BẢNG 5-3 Toán tử quan hệ

| Toán tử | Nghĩa |
|------------|--|
| > | Lớn hơn |
| < | Nhỏ hơn |
| >= | Lớn hơn hoặc bằng |
| <= | Nhỏ hơn hoặc bằng |
| in | Nằm trong một biểu thức hoặc đối tượng |
| instanceof | Là một thực thể (instance) của một đối tượng |

Có thể các bạn đã quen với bốn toán tử quan hệ đầu tiên của Bảng 5-3, tuy nhiên tôi vẫn xin giới thiệu một vài ví dụ về các toán tử này. Xem xét đoạn mã dưới đây (bạn có thể tìm thấy đoạn mã này trong file relational.txt của phần Tài nguyên đi kèm):

```
if (3 > 4) {  
    // viết mã ở đây  
}
```



Số nguyên 3 luôn nhỏ hơn số nguyên 4, vì thế đoạn mã này không bao giờ trả về giá trị *true* (đúng), và đoạn mã trong khối *if* cũng không được thực thi. Tương tự, đoạn mã này kiểm tra xem biến *x* có nhỏ hơn biến *y* hay không:

```
if (x < y) {  
    // viết mã ở đây  
}
```

Toán tử *in*

Toán tử *in* thường được dùng để đánh giá xem một thuộc tính có nằm trong đối tượng hay không. Cần chú ý rằng toán tử *in* tìm kiếm sự tồn tại của một thuộc tính chứ không phải giá trị của thuộc tính đó. Do đó, đoạn mã dưới đây (có thể tìm trong file inop.txt của phần Tài nguyên đi kèm) sẽ thực thi được vì thuộc tính có tên *star* nằm trong đối tượng *myObj*.

```
var myObj = {  
    star: "Algol",
```

```

        constellation: "Perseus"
    };
    if ("star" in myObj) {
        alert("Có một thuộc tính có tên star trong đó)
    }

```

Toán tử *in* thường được dùng để duyệt đối tượng. Sẽ có một ví dụ trình bày cụ thể cách dùng của toán tử này ở Chương 8 “Đối tượng trong JavaScript”.

Toán tử *instanceof*

Toán tử *instanceof* kiểm tra xem một biểu thức cho sẵn, thường là biến, có phải là thực thể (instance) của đối tượng được đưa vào trong biểu thức hay không. Điều này nghe có vẻ phức tạp. Thay vì tìm cách giải thích nó, tôi sẽ đi thẳng vào ví dụ sau đây để bạn thấy dễ hiểu hơn:

```

var myDate = new Date();
if (myDate instanceof Date) {
    // viết mã ở đây
}

```



Bởi vì biến *myDate* là thực thể của đối tượng tích hợp sẵn là *Date*, toán tử *instanceof* trả về giá trị *true*. Toán tử *instanceof* có thể được dùng cho các đối tượng do người dùng định nghĩa cũng như đến đối tượng tích hợp sẵn như minh họa ở ví dụ trên.

Toán tử một ngôi

Toán tử một ngôi chỉ có một toán hạng hoặc chỉ làm việc với một biểu thức đơn trong JavaScript. Bảng 5-4 liệt kê các toán tử một ngôi trong JavaScript.

BẢNG 5-4 Toán tử một ngôi.

| Toán tử | Nghĩa |
|---------|---|
| delete | Loại bỏ thuộc tính. |
| void | Trả về <i>undefined</i> . |
| typeof | Trả về một chuỗi đại diện cho kiểu dữ liệu. |
| ++ | Tăng một số. |

| | |
|----|---|
| -- | Giảm một số. |
| + | Chuyển kiểu của toán hạng sang kiểu số. |
| - | Phủ định toán hạng. |
| ~ | Phủ định (Thao tác bit). |
| ! | Phủ định (logic). |

Bởi vì cách dùng toán tử một ngôi không rõ ràng nên trong chương này tôi sẽ giải thích chi tiết hơn.

Tăng và giảm

Toán tử `++` và `--` được dùng để tăng hoặc giảm một số như minh họa ở đoạn mã dưới đây (Bạn có thể tìm thấy đoạn mã này trong file incrementing.txt của phần Tài nguyên đi kèm):

```
var aNum = 4;
aNum++;
++aNum;
```

Vị trí đặt toán tử so với toán hạng sử dụng nó sẽ xác định giá trị trả về từ đoạn mã. Khi được thêm vào sau biến, như trong dòng mã thứ hai của ví dụ trên, toán tử trả về giá trị *trước khi* tăng (hoặc giảm trong trường hợp tương ứng). Khi được đặt ở trước toán hạng, như ở dòng mã cuối của ví dụ trên, toán tử trả về giá trị sau khi tăng (hoặc giảm).

Đây là hai ví dụ thể hiện sự khác biệt giữa việc đặt toán tử ở trước hay sau trong đoạn mã. Trong ví dụ đầu tiên toán tử được đặt ở sau:

```
var aNum = 4;
var y = aNum++; // y bây giờ có giá trị là 4, còn aNum có gi...
```

Trong ví dụ thứ hai, toán tử được đặt ở trước:

```
var aNum = 4;
var y = ++aNum; // Cả y và aNum đều có giá trị là 5
```

Trên thực tế, bạn sẽ dùng toán tử tăng chèn sau nhiều hơn toán tử tăng chèn trước hoặc toán tử giảm chèn trước vì nó là bộ đếm tiện lợi trong cấu trúc vòng lặp. Bạn sẽ tìm hiểu về vòng lặp trong JavaScript ở Chương 6 “Điều khiển luồng với lệnh điều kiện và vòng lặp”.

Chuyển sang kiểu số với ký hiệu cộng

Ký hiệu cộng (+) được dùng để chuyển một giá trị sang kiểu số. Trên thực tế, tôi thấy toán tử này không đáng tin cậy – hoặc ít nhất là không đủ độ tin cậy để đưa vào sản phẩm thương mại. Khi tôi cần chuyển một kiểu bất kỳ sang kiểu số, tôi dùng hàm tường minh `ToNumber()`. Tuy nhiên bạn có thể sử dụng ký tự cộng như là toán tử một ngôi để chuyển kiểu giá trị, như ví dụ dưới đây (đoạn mã ví dụ này cũng có trong file converting.txt của phần Tài nguyên đi kèm).

```
var x = +"43";
```

Đoạn mã này có kết quả là chuỗi "43" được chuyển thành kiểu số bằng mã JavaScript và số 43 được lưu trong biến x.

Tạo số âm với dấu trừ

Không ngạc nhiên khi bạn dùng dấu trừ (-) trước một số, số đó sẽ được chuyển thành số âm, như trong ví dụ dưới đây (ví dụ này có trong file creating.txt của phần Tài nguyên đi kèm).

```
var y = "754";
var negat = -y;
alert(negat);
```



Phủ định theo thao tác bit và logic

Ký tự tương đương (~) là toán tử phủ định theo thao tác **bit** và dấu chấm than (!) là toán tử phủ định theo **logic**. Hai toán tử này phủ định lẫn nhau. Trong trường hợp thao tác **bit**, giá trị bổ sung bit của toán tử ~ được dùng, giá trị 0 được chuyển thành giá trị -1 và giá trị -1 chuyển thành giá trị 0. Toán tử phủ định **logic**, loại phủ định thường dùng trong lập trình JavaScript, là để phủ định biểu thức. Nếu biểu thức đúng, toán tử phủ định logic biến giá trị biểu thức thành sai.

*Để tham khảo thêm về toán tử thao tác bit, xem trang:
http://en.wikipedia.org/wiki/bitwise_operation.*

Sử dụng toán tử delete

Toán tử *delete* (xóa) nhận vào thuộc tính của một đối tượng hoặc chỉ mục (index) của một mảng và loại bỏ nó hoặc chuyển giá trị thành undefined. Đây là ví dụ đơn giản cho thao tác dùng mảng:

```
var myArray = ("The RCMP", "The Police", "State Patrol");
delete myArray[0]; // Mảng myArray bây giờ chỉ chứa "The Pol
```

Đoạn mã trên tạo một mảng có tên *myArray* và ngay tức thì xóa giá trị ở chỉ mục đầu tiên. Toán tử *delete* cũng được thực hiện cho đối tượng như trong ví dụ dưới đây:

Sử dụng toán tử *delete* với đối tượng

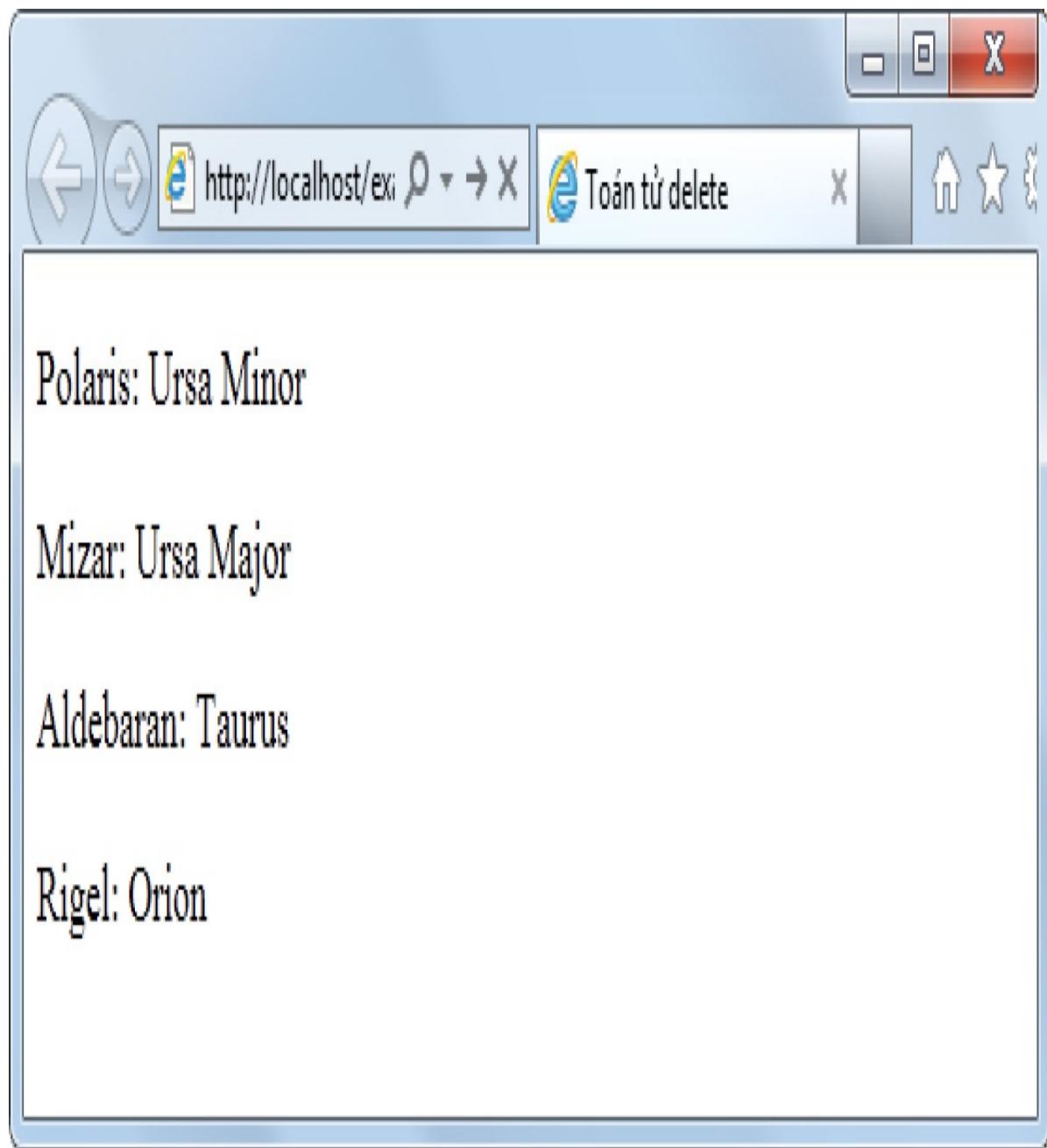
1. Dùng Visual Studio, Eclipse hoặc trình soạn thảo khác thay đổi file deleteop1.htm trong thư mục mã mẫu Chương 5 của phần Tài nguyên đi kèm.
2. Tạo nội dung cho một trang cơ bản và dùng toán tử *delete* ở những bước sau. Trong trang web, thay thế dòng chú thích TODO bằng đoạn mã in đậm dưới đây. (Có thể tìm thấy đoạn mã trong file deleteop1.txt của phần Tài nguyên đi kèm).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Toán tử delete</title>
    <script type="text/javascript">
        var star = {};
        star["Polaris"] = new Object();
        star["Mizar"] = new Object();
        star["Aldebaran"] = new Object();
        star["Rigel"] = new Object();
        star["Polaris"].constellation = "Ursa Minor";
        star["Mizar"].constellation = "Ursa Major";
        star["Aldebaran"].constellation = "Taurus";
        star["Rigel"].constellation = "Orion";
    </script>
</head>
<body id="mainbody">
```

```
<script type="text/javascript">
    for (starName in star) {**
        var para = document.createElement("p");
        para.id = starName;
        para.appendChild(document.createTextNode(starName));
        document.getElementsByTagName("body")[0].appendChild(para);
    }
</script>
</body>
</html>
```

Trong phần `<head>`, bạn đã tạo một đối tượng ngôi sao và một vài đối tượng ngôi sao khác, được đặt tên là `star["starname"]`. Sau đó, bạn cho đối tượng một thuộc tính *constellation* (chòm sao) có giá trị là tên chòm sao. Sau đó, trong phần `<body>` của đoạn mã, vòng lặp `for` sẽ thực thi để duyệt từng ngôi sao trong đối tượng ngôi sao. Đoạn mã này sử dụng mô hình đối tượng tài liệu (DOM) sẽ được giới thiệu ở Chương 10 "Mô hình đối tượng tài liệu". Còn ở chương này, chúng ta không quan tâm nhiều đến ý nghĩa của đoạn mã trong vòng lặp `for`.

3. Lưu file và dùng trình duyệt mở trang. Kết quả như sau:



4. Thêm toán tử *delete* ở phía trên vòng lặp *for* trong đoạn mã để loại bỏ *constellation* khỏi Polaris. Đoạn mã (có trong file deleteop2.txt của phần Tài nguyên đi kèm) sẽ có dạng như dưới đây:

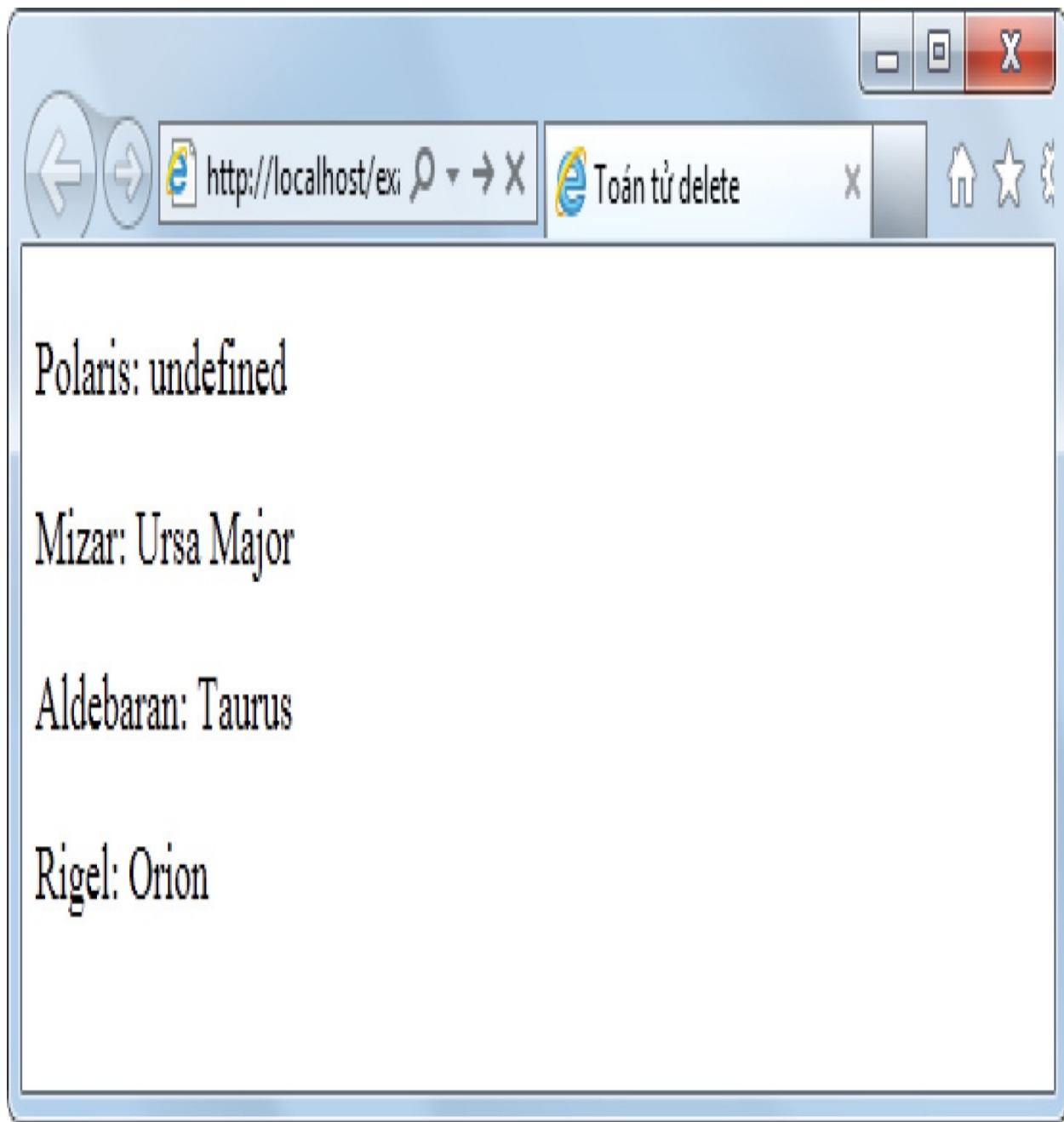
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title>Toán tử delete</title>
```

```
<script type="text/javascript">
var star = {};
star["Polaris"] = new Object();
star["Mizar"] = new Object();
star["Aldebaran"] = new Object();
star["Rigel"] = new Object();
star["Polaris"].constellation = "Ursa Minor";
star["Mizar"].constellation = "Ursa Major";
star["Aldebaran"].constellation = "Taurus";
star["Rigel"].constellation = "Orion";
</script>
</head>
<body id="mainbody">
<script type="text/javascript">
delete(star["Polaris"].constellation);
for (starName in star) {
    var para = document.createElement("p");
    para.id = starName;
    para.appendChild(document.createTextNode(": " + star[starName].constellation));
    document.getElementsByTagName("body").appendChild(para);
}
</script>
</body>
</html>
```



Lưu ý toán tử `delete` được thêm vào thẻ `<script>` trong phần body của văn bản (được in đậm).

5. Lưu file và dùng trình duyệt mở trang. Kết quả trả về sẽ như dưới đây:



Việc sử dụng toán tử *delete* sẽ khiến thuộc tính *constellation* của Polaris trở thành *undefined*. Bạn có thể xóa toàn bộ đối tượng trong Polaris bằng cách:

```
delete(star["Polaris"]);
```

Trả về kiểu biến với toán tử *typeof*

Toán tử *typeof* trả về kiểu biến của một toán hạng cho trước. Sử dụng *typeof*,

bạn có thể xác định biến được tạo và biến đó đang được dùng dưới dạng chuỗi, số, boolean hoặc biến đó là một loại đối tượng hay hàm nào đó. Xem đoạn mã dưới đây:

```
var star= {};
if (typeof(star) == "object") {
    alert("star là một đối tượng");
}
```

Toán tử *typeof* trả về "number" nếu đang đánh giá một số, trả về "string" nếu đang đánh giá một chuỗi ký tự, và (có thể thấy từ ví dụ) trả về "object" nếu đang đánh giá một đối tượng. Khi bạn dùng các thuộc tính, JavaScript có thể giả định rằng bạn muốn biết kiểu biến của thuộc tính đó, hơn là kiểu của đối tượng, vì thế JavaScript sẽ trả về kiểu dữ liệu của thuộc tính. Đây là ví dụ mượn một đoạn mã nhỏ từ ví dụ ở phần trước của chương.

Sử dụng toán tử *typeof*

1. Sử dụng Visual Studio, Eclipse hoặc trình soạn thảo khác thay đổi file *typeof.htm* trong thư mục mã mẫu Chương 5 của phần Tài nguyên đi kèm.
2. Trên trang web, thêm đoạn mã  được in đậm dưới đây (có trong file *typeof.txt* của phần Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Ví dụ về toán tử Typeof</title>
    <script type="text/javascript">
        var star = {};
        star["Polaris"] = new Object();
        star["Polaris"].constellation = "Ursa Minor";
        alert(typeof star["Polaris"].constellation);
    </script>
</head>
<body>
</body>
</html>
```

3. Trong đoạn mã trong thẻ *<script>* tạo một đối tượng mới cho ngôi sao

Polaris và thiết lập thuộc tính *constellation* bằng "*Ursa Minor*". Sau đó, gọi một hộp thoại thông báo sử dụng toán tử *typeof* để thông báo kiểu của thuộc tính *star["Polaris"].constellation* là một chuỗi.

4. Lưu file và dùng trình duyệt mở trang. Bạn sẽ nhận được một hộp thoại thông báo như sau:



Sử dụng toán tử *typeof*, bạn còn có thể thấy được sự khác biệt giữa *null* và *undefined*.

Toán tử void

Nếu như bạn từng xem một đoạn mã nguồn trong JavaScript, bạn có thể gặp toán tử *void*. Toán tử *void* trả về giá trị *undefined* sau khi kiểm tra đối số của nó. Điều này có nghĩa là toán tử *void* cho phép lập trình viên web gọi một hàm mà không hiển thị kết quả trong trình duyệt. Dù toán tử này và giao thức mã giả *javascript*: tồn tại trong JavaScript, hãy tránh dùng chúng khi lập trình. Cuốn sách này sẽ đề cập đến toán tử *void* và giao thức mã giả *javascript*: để bạn hiểu

được vai trò của toán tử này khi gỡ lỗi cho mã JavaScript.

Cách thức dùng phổ biến của toán tử *void* là để gửi form hoặc để mở cửa sổ mới. Ví dụ dưới đây minh họa cách dùng của toán tử *void*:

```
void(window.open());
```

Cách dùng phổ biến hơn là đặt đoạn mã *javascript:void* vào đường link đến trang web để mở cửa sổ mới, như dưới đây:

```
<a href="javascript:void(window.open())">Mở cửa sổ mới bằng ..
```

Chú ý Toán tử *void*, hoặc cụ thể hơn là cách dùng của giao thức mã giả *javascript:* trong thuộc tính *href*, nhìn chung không được khuyến khích dùng.

Toán tử gán

Trong chương này bạn đã biết đến toán tử gán, và bạn sẽ gặp chúng xuyên suốt cuốn sách. Toán tử gán chủ yếu (thường được sử dụng nhiều nhất) là dấu bằng (=). Loại toán tử này được biết đến như là *toán tử gán đơn giản*. JavaScript có nhiều toán tử gán khác, bao gồm những toán tử được liệt kê ở Bảng 5-5.

BẢNG 5-5 Các toán tử gán phức hợp.

| Toán tử | Nghĩa |
|----------------------------|---|
| <code>*=</code> | Nhân giá trị của toán hạng trái với toán hạng phải. |
| <code>/=</code> | Chia giá trị của toán hạng trái cho toán hạng phải. |
| <code>%=</code> | Lấy số dư của phép chia môđun của toán hạng trái và phải. |
| <code>+=</code> | Cộng toán hạng phải vào toán hạng trái. |
| <code>-=</code> | Trừ toán hạng trái một giá trị bằng toán hạng phải. |
| <code><<=</code> | Thao tác đẩy bit sang bên trái. |
| <code>>>=</code> | Thao tác đẩy bit sang bên phải. |
| <code>>>>=</code> | Thao tác đẩy bit của số dương sang bên phải. |
| <code>&=</code> | Thao tác bit <i>AND</i> . |
| <code>^=</code> | Thao tác bit <i>XOR</i> . |
| <code>=</code> | Thao tác bit <i>OR</i> . |

Các toán tử gán phức hợp cung cấp cách viết tắt giúp tiết kiệm một số thao tác gõ phím và tiết kiệm vài byte dữ liệu. Ví dụ, bạn có thể cộng hoặc trừ một số sử dụng `+=` và `-=`, tương ứng, như ở ví dụ dưới đây:

```
var myNum = 10;  
alert(myNum);  
myNum += 30;  
alert(myNum);
```

Hộp thoại thông báo đầu tiên, sau khi biến được định nghĩa và gán bằng 10, là:



Hộp thoại thông báo tiếp theo, sau khi sử dụng toán tử gán cộng phức hợp, là:



Tầm quan trọng của việc thu gọn mã

Tiết kiệm kích cỡ là một đề tài quan trọng đối với mỗi lập trình viên JavaScript. Tiết kiệm kích cỡ liên quan đến việc lập trình với các phương thức tắt sao cho kết quả lập trình JavaScript (hoặc các ngôn ngữ khác trong cùng một vấn đề) sử dụng ít bộ nhớ và băng thông hơn. Khi bạn tận dụng được tính năng để tiết kiệm mã, ví dụ các câu lệnh gán phức hợp, chương trình sẽ vận hành tốt hơn.

Kích cỡ mã nhỏ hơn nghĩa là người dùng sẽ tải xuống đoạn mã nhẹ hơn. Việc xác định số lượng byte bạn có thể tiết kiệm hoặc được phép dùng, là rất khó. Nhiều lập trình viên có thể phản bác rằng hiệu quả là không đáng kể - và với những đoạn mã nhỏ, ý kiến này có thể đúng, đặc biệt khi số lượng người dùng có băng thông rộng hoặc kết nối nhanh hơn hiện ngày càng tăng. Nhưng hiệu quả của các phương thức tắt thông minh này có thể rõ với những đoạn mã lớn hơn, đặc biệt khi những đoạn mã được tải từ kết nối quay số hoặc các loại kết nối chậm khác.

Một phương pháp phổ biến để tiết kiệm kích cỡ mã trong quá trình tải là thu

gọn JavaScript. Việc thu gọn mã loại bỏ toàn bộ các phần tử không cần thiết trong file JavaScript của một trang web đang hoạt động. Các phần tử không cần thiết bao gồm chú thích, phím cách và ký tự xuống dòng. Kết quả của việc thu gọn mã không quá rõ rệt trừ khi bạn chèn lại các phím cách cũng như các ký tự xuống dòng.

Toán tử phẩy

Toán tử phẩy phân tách các biểu thức với nhau và cho phép thực thi lần lượt các biểu thức đó. Thông thường, dấu phẩy được dùng để phân tách các lệnh khai báo biến, cho phép nhiều biến có thể được khai báo trên cùng một dòng:

```
var num1, num2, num3;  
Ngoài ra, bạn cũng có thể khởi tạo giá trị cho các biến:  
var num1=3, num2=7, num3=10;
```



Bài tập

1. Sử dụng toán tử cộng (+) để hiển thị ba hộp thoại thông báo `alert()` trên màn hình (bạn có thể dùng ba chương trình riêng biệt). Hộp thoại thông báo đầu tiên công hai số. Hộp thoại thông báo thứ hai cộng một số và một chuỗi. Hộp thoại thông báo số ba cộng hai chuỗi. Tất cả đều được biểu diễn bằng các biến.
2. Sử dụng toán tử tăng (++) chèn sau để tăng một số lưu trong biến.
3. Hiển thị giá trị của biến trước, trong và sau khi tăng. Sử dụng toán tử tăng chèn trước để tăng một số và hiển thị kết quả của nó trước, trong và sau khi tăng bằng hộp thoại thông báo.
4. Sử dụng toán tử `typeof` để kiểm tra kiểu biến đã tạo ở Bài tập 1.
5. Đúng hay sai: Các toán tử một ngôi thường không xuất hiện trong JavaScript.

- Đúng hay sai: Bạn nên tiết kiệm kích cỡ mã (sử dụng các phương thức tắt của JavaScript bất cứ khi nào có thể), thay vì viết xuống dòng và thụt đầu dòng, khiến việc tải trang trở nên chậm hơn.



Phần 2

Áp dụng JavaScript

[Chương 6: Điều khiển luồng với câu lệnh điều kiện và vòng lặp](#)

[Chương 7: Làm việc với hàm](#)

[Chương 8: Các đối tượng trong JavaScript](#)

[Chương 9: Mô hình đối tượng trình duyệt](#)

Chương 6

Điều khiển luồng với câu lệnh điều kiện và vòng lặp

Sau khi đọc xong chương này, bạn có thể:

- Nắm được các câu lệnh điều kiện trong JavaScript.
- Sử dụng câu lệnh điều kiện *if else* để điều khiển việc thực thi các đoạn mã.
- Sử dụng câu lệnh *switch*.
- Nắm được các loại cấu trúc điều khiển vòng lặp trong JavaScript.
- Sử dụng vòng lặp *while* và *do...while* để chạy lại các đoạn mã.
- Sử dụng các loại vòng lặp *for* để duyệt trong một khoảng giá trị.

If (và cách dùng)

Câu lệnh *if* đánh giá một biểu thức và dựa trên kết quả để quyết định thực thi đoạn mã nào trong chương trình. Câu lệnh *if* phức hợp quyết định đoạn mã được thực thi dựa trên nhiều điều kiện. Nếu từng đặt vé máy bay qua mạng, bạn sẽ hình dung ra quy trình ra quyết định. Ví dụ, bạn muốn sắp xếp đi nghỉ vào cuối tuần. Vì thế, khi quyết định đặt mua vé, bạn tính toán: “Nếu giá vé thấp hơn \\$350, tôi sẽ đặt chỗ, nếu không, tôi sẽ chọn địa điểm khác”. Tương tự, giả sử tôi muốn đi đổ rác. Tôi nên mang rác ra vỉa hè ngay tối nay hay đợi đến sáng hôm sau? Nếu dự báo thời tiết là đêm đó trời sẽ có gió, rác có thể bị thổi tung sang bãi cỏ của nhà hàng xóm, nhưng nếu đợi đến sáng, tôi có thể lỡ xe rác.

Mặc dù không giúp bạn trong những quyết định thực tế này, nhưng JavaScript sẽ hỗ trợ đáng kể bằng việc ra những quyết định như điều khiển cách thức thực hiện của chương trình dựa trên việc liệu biến có chứa một giá trị cụ thể nào đó hay không hoặc trường nhập dữ liệu nào đó được nhập đúng hay sai. Phần này sẽ đề cập đến cú pháp của câu lệnh *if* trong JavaScript.

Cú pháp của câu lệnh *if*

Cú pháp của câu lệnh *if* rất quen thuộc với những người từng sử dụng các ngôn ngữ lập trình khác như Perl hoặc PHP. Cấu trúc cơ bản của câu lệnh *if* là:

```
if (điều kiện) {  
    // Viết mã ở đây  
}
```

Chú ý Câu lệnh if còn được gọi là câu lệnh điều kiện *if*. Tôi sẽ dùng cả hai thuật ngữ này để các bạn làm quen với cả hai. Bạn đừng nhầm lẫn giữa câu lệnh điều kiện *if* (toute bộ câu lệnh if) với điều kiện *if* là biểu thức *logic* mà câu lệnh *if* đánh giá.

Câu lệnh *if* kiểm tra xem tính hợp lệ và chính xác của điều kiện để quyết định thực thi đoạn mã trong câu lệnh điều kiện (trong cặp ngoặc nhọn). Điều kiện là một biểu thức logic, khi được đánh giá là *true* (đúng) thì câu lệnh *if* thực thi đoạn mã trong thân lệnh. (Bạn cũng có thể phủ định biểu thức trong điều kiện để đoạn mã trong thân lệnh thực thi nếu biểu thức đánh giá là *false* - sai). Hãy nhớ lại cách dùng kiểu logic và toán tử một ngôi ở Chương 5 “Sử dụng toán tử và biểu thức”. Dưới đây là ví dụ:

```
if (! điều kiện) {  
    // viết mã ở đây  
}
```

Trong trường hợp này, câu điều kiện bắt đầu với toán tử phủ định, tức là điều kiện này cần được đánh giá là *false* (sai) để đoạn mã bên trong thân lệnh được thực thi.

Ví dụ về chi phí vé máy bay trên thực tế của ví dụ ở phần đầu chương này có thể tạo thành đoạn mã giả như sau:

```
if (flightCost < 350) { //chi phí chuyến bay nhỏ hơn 350 đô-  
    bookFlight(); //đặt vé máy bay  
}
```

Nếu giá vé thấp hơn 350 đô-la đoạn mã trong thân lệnh được thực thi. Ví dụ về quyết định đổ rác sẽ như sau:

```
if (forecast != "windy") { //dự báo trời không có gió  
    takeGarbageOut(); //mang rác ra ngoài  
}
```

Phần sau của chương này sẽ trình bày cách dùng câu lệnh *else* để thực thi các đoạn mã khi điều kiện là *false* (sai).

Bạn có thể dùng câu lệnh *if* với các toán tử được nhắc đến ở Chương 5, đặc biệt là các toán tử quan hệ để kiểm tra xem một giá trị lớn hơn hay nhỏ hơn một giá trị khác và toán tử bằng để kiểm tra xem hai giá trị có bằng nhau hay không.

Xem xét ví dụ dưới đây:

```
var x = 4;  
var y = 3;  
//Kiểm tra bằng nhau  
if (x == y) {  
    // viết mã ở đây  
}
```

Vì giá trị trong biến *x* (4) không bằng giá trị trong biến *y* (3), do đó đoạn mã trong câu điều kiện *if* (trong dấu ngoặc đơn) không được thực thi. Dưới đây là một ví dụ với toán tử quan hệ:

```
var x = 4;  
var y = 3;
```

```
// kiểm tra quan hệ  
if (x > y) {  
    // viết mã ở đây  
}
```

Trong trường hợp này, giá trị trong biến *x* (4) lớn hơn giá trị trong biến *y* (3), đoạn mã trong dấu ngoặc đơn được thực thi.

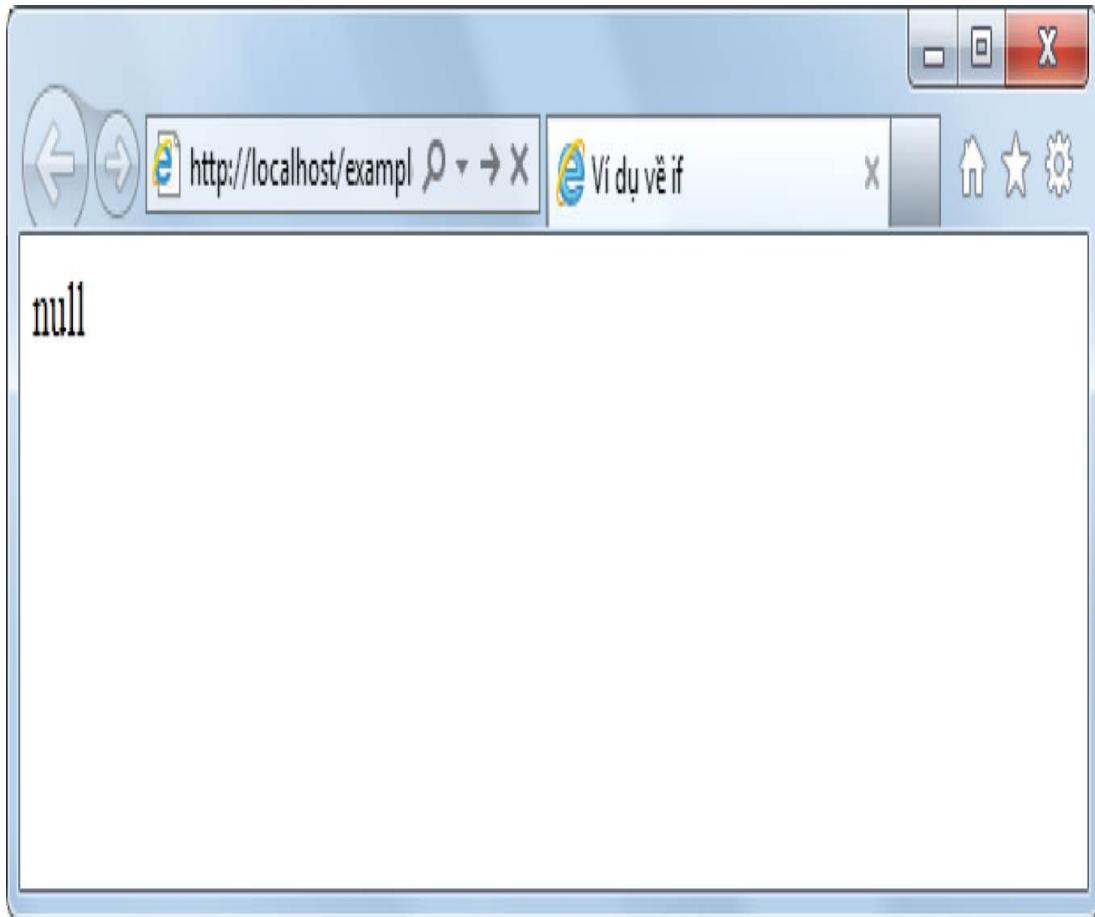
Phần tiếp theo giới thiệu một ví dụ mà bạn có thể tự thực hiện. Ví dụ này dùng hàm *prompt()* để lấy giá trị do người dùng nhập vào thông qua một giao diện đơn giản.

Hàm *prompt()* trong trình duyệt Internet Explorer

Kể từ Windows Internet Explorer 7, hàm *prompt()* không được kích hoạt mặc định nữa. Nếu muốn dùng hàm *prompt()* với Internet Explorer, bạn sẽ thấy một cảnh báo về bảo mật như ở Hình 6-1, hoặc có thể là một trang với từ *null*, như cảnh báo ở Hình 6-2.



HÌNH 6-1 Đây là cảnh báo về bảo mật tạo ra bởi hàm *prompt()* trong Internet Explorer.



HÌNH 6-2 Khi sử dụng giả giao thức javascript: với hàm `prompt()`, đôi khi bạn có thể nhận được một trang với từ `null`.

Bạn có thể tắt tính năng này bằng cách nhấp chuột vào thanh thông tin (ở Hình 6-1) và chọn cho phép script, hoặc bằng cách thay đổi thiết lập về bảo mật. Bạn có thể thay đổi các thiết lập về bảo mật trong Internet Explorer bằng cách chọn Internet Options từ menu Tools, nhấp chuột vào tab Security, nhấp chuột vào Customer Level và kích hoạt lựa chọn Allow Web Sites To Prompt For Information Using Scripted Windows trong phần Scripting.

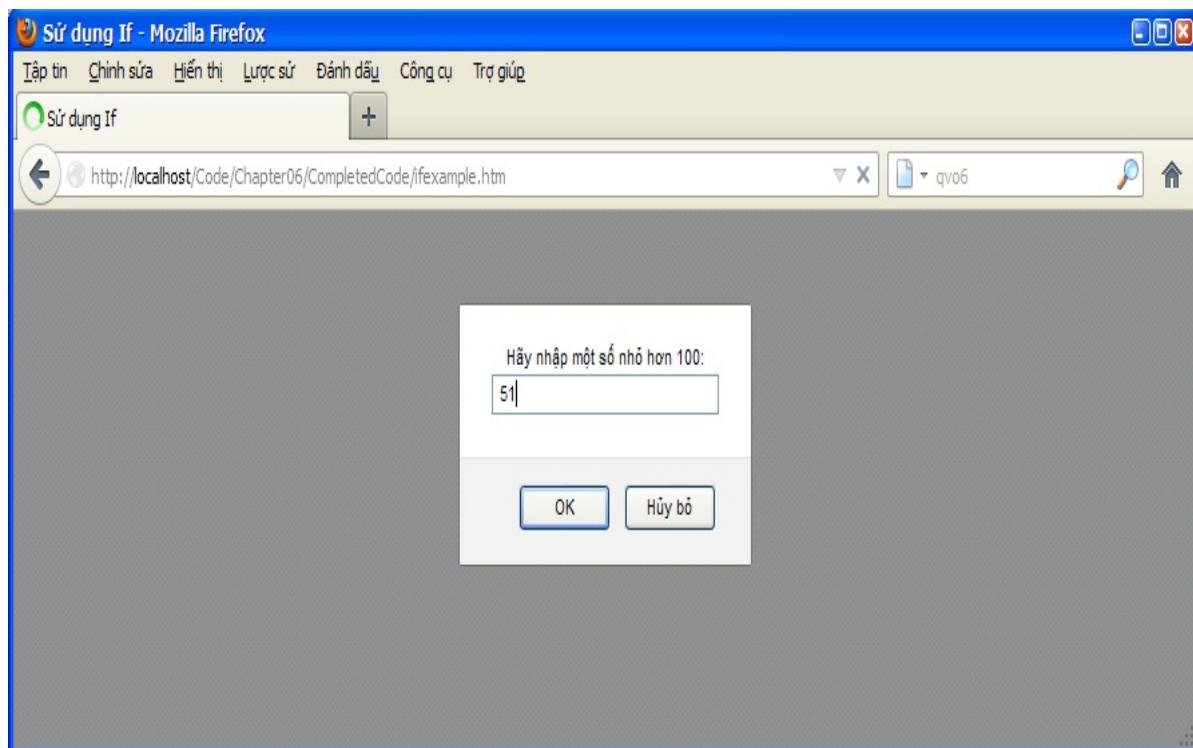
Tuy nhiên, bạn không thể giả định người truy cập sẽ thiết lập như thế với trình duyệt Internet Explorer của họ. Do đó, hàm `prompt()` không còn hữu dụng như trước khi Internet Explorer 7 ra mắt. Nhiều lập trình viên kết luận rằng hàm `prompt()` gây nhiều phiền toái (và tôi đồng ý rằng đôi khi nó làm nảy sinh nhiều vấn đề), nhưng nó cũng có nhiều ưu điểm và việc tắt nó đi chỉ góp phần rất nhỏ cho việc bảo mật. Hàm `prompt` cũng rất hữu ích cho mục đích kiểm thử, ví dụ như trong bài tập dưới đây.

Sử dụng *if* để quyết định luồng chạy của chương trình

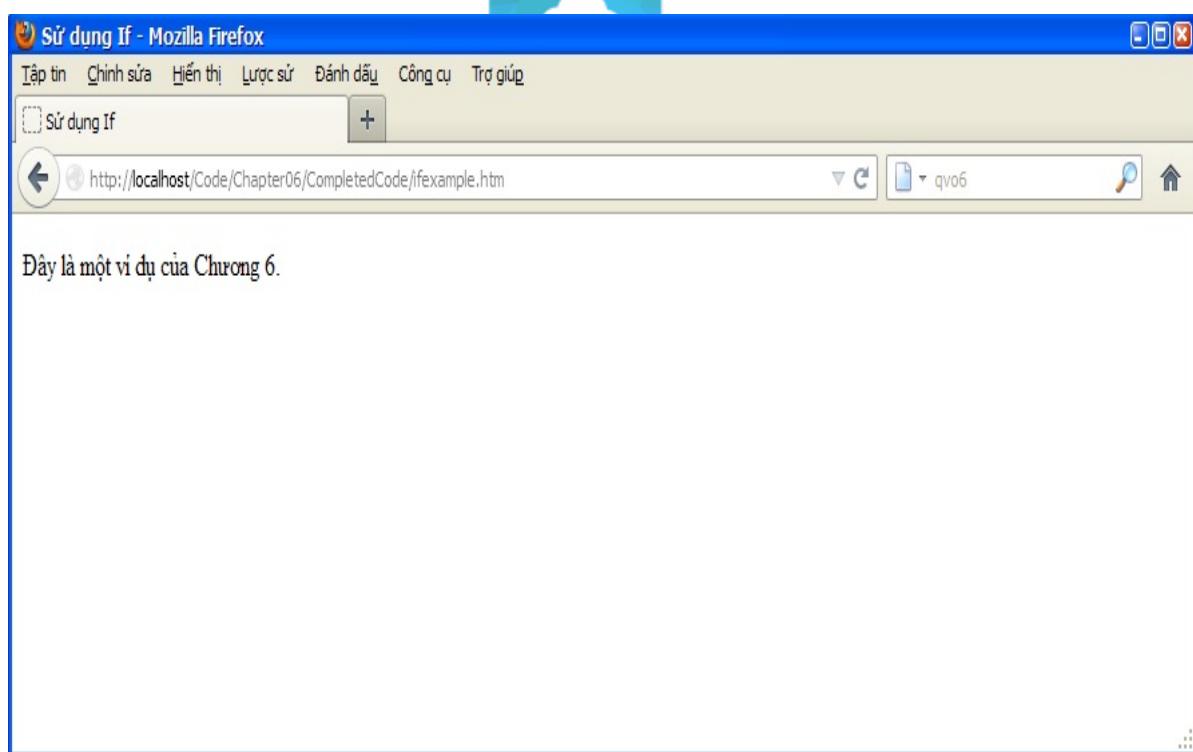
1. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo khác thay đổi file ifexample.htm trong thư mục mã nguồn mẫu của Chương 6, phần Tài nguyên đi kèm.
2. Trong trang này, thay dòng chú thích TODO bằng đoạn mã in đậm dưới đây (Đoạn mã này cũng nằm trong file ifexample.txt của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Sử dụng If</title>
</head>
<body>
<script type="text/javascript">
var inputNum = prompt("Hãy nhập một số nhỏ hơn 100:");
if (inputNum > 99) {
    alert("Số, " + inputNum + ", không nhỏ hơn 100.");
}
</script>
<p>Đây là một ví dụ của Chương 6.</p>
</body>
</html>
```

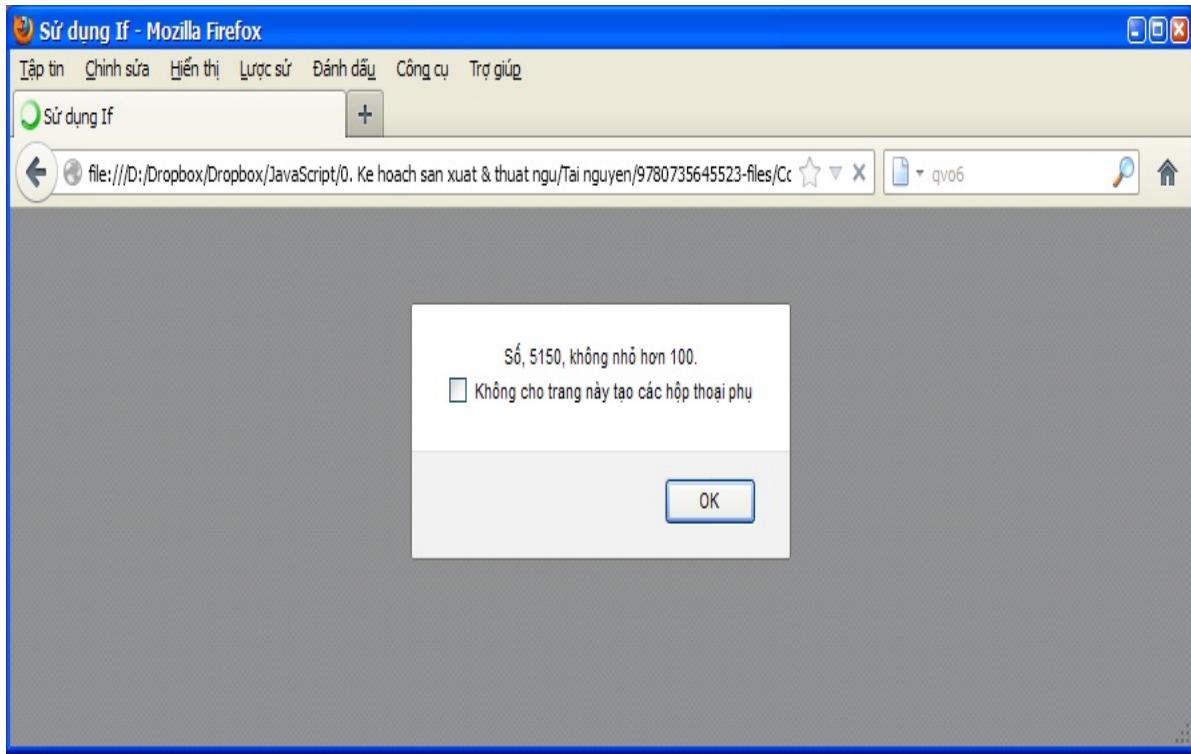
3. Lưu và dùng trình duyệt mở lại trang này. Nếu bạn xem trang này bằng Internet Explorer và nhận được một cảnh báo bảo mật, bạn cần thay đổi các thiết lập về bảo mật như hướng dẫn ở trên. Bạn cũng có thể dùng FireFox hoặc các trình duyệt khác.
4. Khi bạn xem trang, hộp thoại yêu cầu nhập một số nhỏ hơn 100 xuất hiện. Thông thường, Internet Explorer tự điền vào ô trống trong hộp thoại từ “undefined”. Nhập một số và nhấn OK như ví dụ sau:



5. Nhấn OK. Bạn sẽ thấy một trang như dưới đây:



6. Tải lại trang và lần này khi được hỏi nhập một số lớn hơn 100. Bạn sẽ thấy thông báo sau:



Ngoại trừ HTML và các thẻ script mở mà bạn đã được xem ở những ví dụ trước, đoạn mã hoạt động như dưới đây:

Dòng đầu tiên trong thẻ `<script>` ở phần thân thiết lập biến `inputNum` và sau đó gán cho biến này bằng với kết quả từ hàm `prompt()`:

```
var inputNum = prompt("Hãy nhập một số nhỏ hơn 100:");
```

Những dòng mã tiếp theo sử dụng câu lệnh `if` để đánh giá giá trị trong biến `inputNum`. Nếu giá trị lớn hơn 99, một hộp thoại thông báo xuất hiện:

```
if (inputNum > 99) {  
    alert("Số, " + inputNum + ", không nhỏ hơn 100.");  
}
```

Ví dụ này còn rất sơ sài và những ví dụ sau sẽ hướng dẫn bạn cách cải tiến đoạn mã từ những gì bạn đã học và sử dụng các kỹ thuật mới sẽ được học ở phần sau của chương này.

Điều kiện phức hợp

Có nhiều lúc bạn cần kiểm tra nhiều hơn một điều kiện trong cùng câu lệnh `if`.

Xem lại ví dụ trước. Giả sử bạn muốn khách truy cập nhập vào một số trong khoảng từ 51 đến 99. Bạn cần kết hợp những kiểm tra trên trong cùng câu lệnh *if*:

```
if ((inputNum < 51) || (inputNum > 99)) {  
    alert("Số,"+inputNum+", không nằm trong khoảng từ 50  
}
```

Chú ý Bạn cũng có thể viết câu lệnh *if* trên mà không cần thêm các dấu ngoặc đơn cho mỗi điều kiện cần đánh giá. Tuy nhiên, tôi thấy rằng việc thêm các dấu ngoặc đơn sẽ giúp đoạn mã dễ đọc hơn.

Bạn có thể thấy đoạn mã hoàn chỉnh của ví dụ trước với câu lệnh *if* phức hợp được in đậm, trong Ví dụ 6-1. (Đoạn mã này có trong file listing6-1.htm của phần Tài nguyên đi kèm).

VÍ DỤ 6-1 Câu lệnh *if* phức hợp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Sử dụng If</title>  
</head>  
<body>  
    <script type="text/javascript">  
        var inputNum = prompt("Hãy nhập một số trong khoảng từ 50  
        if ((inputNum < 51) || (inputNum > 99)) {  
            alert("Số,"+inputNum+", không nằm trong khoảng từ  
        }  
    </script>  
    Đây là một ví dụ của Chương 6.  
</body>  
</html>
```

Câu lệnh trong Ví dụ 6-1 sử dụng toán tử logic OR và được đọc là “Nếu *inputNum* lớn hơn 99 hoặc *inputNum* nhỏ hơn 51, thực hiện thao tác này”.

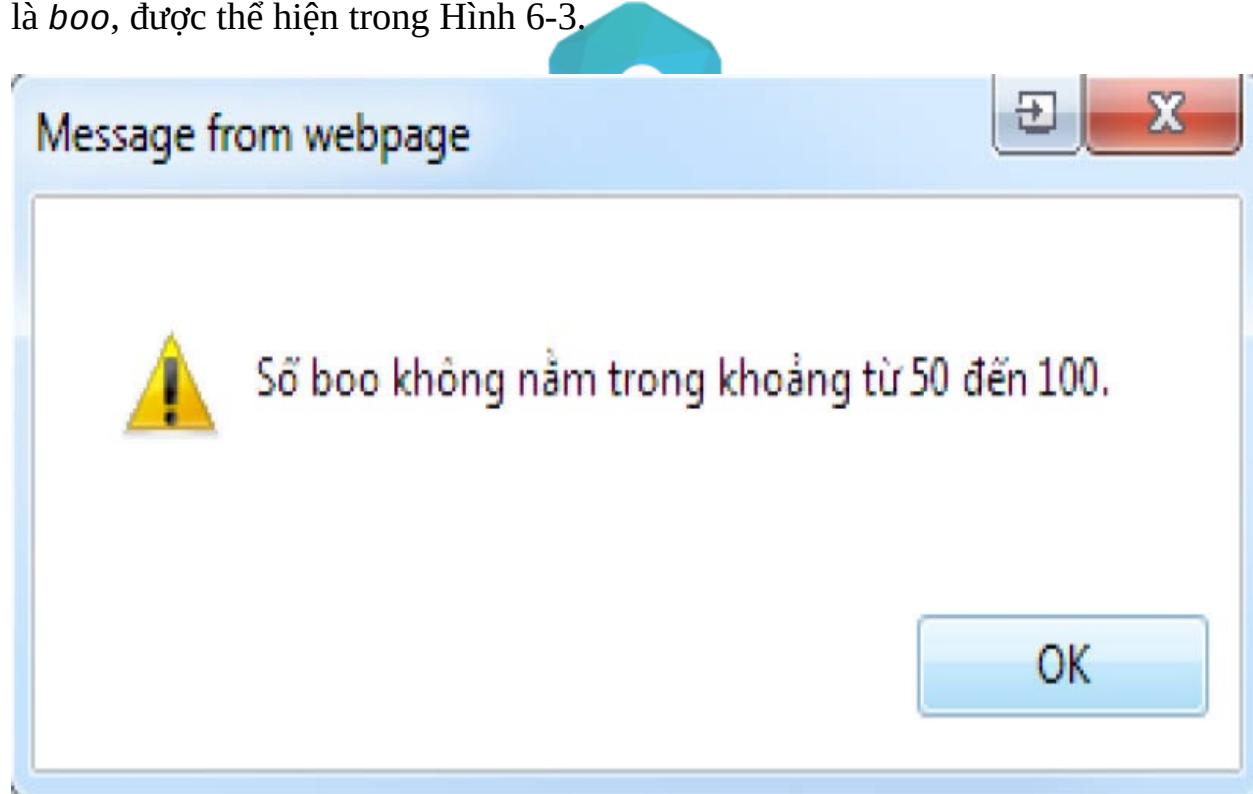
Xem lại ví dụ mà chúng ta sử dụng nhiều trong chương này. Nếu nhập một số lớn hơn 99 hoặc nhỏ hơn 51, bạn sẽ nhận được một hộp thoại thông báo. Nhưng

điều gì sẽ xảy ra nếu đầu vào không phải là một số? Điều gì xảy ra nếu bạn nhập vào từ *boo*? Bạn sẽ không nhận được hộp thoại thông báo bởi vì điều kiện hiện tại chỉ kiểm tra biến lớn hơn hoặc nhỏ hơn các số được thiết lập.

Do đó, đoạn mã cần kiểm tra giá trị trong biến có là số hay không. Bạn có thể thực hiện nhiệm vụ này với hàm *isNaN()* và lồng các điều kiện vào nhau như dưới đây:

```
if (isNaN(inputNum) || ((inputNum > 99) || (inputNum < 51)))
    alert("Số " + inputNum + " không nằm trong khoảng 50 đến 100")
```

Đầu tiên, câu lệnh điều kiện sẽ kiểm tra xem giá trị trong biến *inputNum* có là số hay không. Nếu điều kiện kiểm tra đầu tiên này đạt giá trị true (người dùng đã không nhập số), chương trình không cần thực hiện các thao tác sau và ngừng kiểm tra các điều kiện còn lại. Nếu người dùng nhập số, hàm *isNaN* trả về *false*, câu lệnh *if* sẽ tiếp tục thực hiện việc kiểm tra giá trị số nằm trong cặp ngoặc đơn và tạo thành một bộ điều kiện. Kết quả khi chạy với giá trị nhập vào là *boo*, được thể hiện trong Hình 6-3.



HÌNH 6-3 Chạy ví dụ với hàm *isNaN()* trong câu điều kiện ghép.

Đoạn mã hoàn chỉnh có trong Ví dụ 6-2 (xem trong file listing6-2.htm của phần

Tài nguyên đi kèm). Điều kiện phức hợp được in đậm.

VÍ DỤ 6-2 Câu lệnh *if* phức hợp.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Sử dụng If</title>  
</head>  
<body>  
    <script type="text/javascript">  
        var inputNum = prompt("Hãy nhập một số trong khoảng từ 50  
        if (isNaN(inputNum) || ((inputNum > 99) || (inputNum < 51))  
            alert("Số"+inputNum+" không nằm trong khoảng từ 50  
        }  
    </script>  
    Đây là một ví dụ.  
</body>  
</html>
```



Sử dụng câu lệnh *if* và *else*

Vẫn đề tiếp theo với đoạn mã ví dụ là hộp thoại thông báo trong Hình 6-3 luôn chỉ ra là bạn đã nhập số. Điều này hiển nhiên không phải lúc nào cũng đúng – tôi đã nhập từ *boo*. Bạn cần thực hiện việc kiểm tra nhiều điều kiện riêng biệt. Bạn làm việc đó như thế nào? Câu trả lời là lệnh *else if* và *else*.

Else if

Hầu hết các ngôn ngữ lập trình hiện đại đều có cấu trúc câu lệnh điều kiện *if/else if/else* nhưng cách sử dụng sẽ khác nhau, đặc biệt là cách viết chính tả hoặc cấu trúc câu lệnh *else if*. Một số ngôn ngữ định nghĩa nó là *elsif* (viết liền thành một từ và sai chính tả). Các ngôn ngữ khác định nghĩa nó là *elseif* (viết liền thành một từ và đúng chính tả). Việc nhớ những cách viết này là một thách thức, chưa nói tới cách dùng khác nhau cho các loại dấu ngoặc trong các ngôn ngữ để định nghĩa đoạn mã được thực

thi. Trong lập trình JavaScript, bạn sẽ dùng *else if* – viết thành hai từ riêng biệt và đúng chính tả.

Sử dụng *else if* và *else*, bạn có thể tạo ra nhiều cấp độ của điều kiện, mỗi điều kiện sẽ được kiểm tra lần lượt. Đoạn mã trong điều kiện đúng đầu tiên sẽ được thực thi. Nếu không có điều kiện nào đúng, đoạn mã trong điều kiện *else* (nếu có) sẽ được thực thi. Ví dụ 6-3 (file listing6-3.htm của Tài nguyên đi kèm) cho thấy đoạn mã kiểm tra đầu tiên xem biến *inputNum* có chứa số hay không. Nếu như giá trị là số, câu lệnh *else if* sẽ thực hiện việc kiểm tra để chắc chắn giá trị đầu vào nằm trong khoảng thích hợp. Đoạn mã gọi hàm *alert()* tương ứng dựa trên điều kiện đúng. Nếu bạn nhập số, điều kiện *else* sẽ chạy và hiển thị hộp thoại thông báo về số đã nhập.

VÍ DỤ 6-3 Sử dụng điều kiện *else if* và *else*.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Sử dụng If</title>
</head>
<body>
<script type="text/javascript">
var inputNum = prompt("Hãy nhập một số trong khoảng từ 50
if (isNaN(inputNum)) {
    alert(inputNum + " không phải là một số.");
}
else if ((inputNum > 99) || (inputNum < 51)) {
    alert("****Số ****"+inputNum+" không nằm trong kho
}
else {
    alert("Bạn đã nhập số: " + inputNum);
}
</script>
Đây là một ví dụ của Chương 6.
</body>
</html>
```

Tương tự như cách bạn dùng *else if* và *else* để kiểm tra nhiều điều kiện khác nhau, bạn có thể (đôi khi còn phải) sử dụng nhiều cấp độ của câu điều kiện. Ví

dụ, bạn có thể kiểm tra cho một điều kiện nào đó và khi thành công, thực thi các điều kiện bổ sung. Dưới đây là ví dụ sử dụng hàm *match()* và một biểu thức chính quy. Để có thêm thông tin về biểu thức chính quy, đọc lại Chương 4 “Làm việc với biến và kiểu dữ liệu”.

Sử dụng các câu lệnh điều kiện nhiều cấp độ và biểu thức chính quy

1. Mở một trình soạn thảo. Nếu bạn đã làm theo những bước trước ở chương này, hãy mở file mà bạn đã thay đổi, ifexample.htm (Tài nguyên đi kèm).

Trong file sẽ có đoạn mã dưới đây. (Nếu bạn không thực hiện theo ví dụ trước, chỉ cần tạo một file trống, dán vào đoạn mã dưới đây và đến bước tiếp theo).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Sử dụng If</title>  
</head>  
<body>  
    <script type="text/javascript">  
        var inputNum = prompt("Hãy nhập một số trong khoảng  
        if (inputNum > 99) {  
            alert("Số "+inputNum+" không nằm  
        }  
    </script>  
    Đây là một ví dụ của Chương 6.  
    </body>  
</html>
```

1. Lưu file với một tên khác.
2. Nhập đoạn mã được in đậm dưới đây vào file vừa mới lưu ở bước trước: Lưu ý những phần thay đổi so với ví dụ trước được in đậm:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Sử dụng câu lệnh If lồng nhau </title>  
</head>
```

```

<body>
<script type="text/javascript">
var inputNum = prompt("Hãy nhập một số trong khoảng từ 50
if (isNaN(inputNum)) {
    if (inputNum.match(/một|hai|ba|bốn|năm|sáu|bảy|tám/))
        alert("Dù nó là một số, nó không thực sự là số.");
    } else {
        alert(inputNum + " không phải là một số.");
    }
}
else if ((inputNum > 99) || (inputNum < 51)) {
    alert("Số "+inputNum+" không nằm trong khoảng từ 50 đến 100.");
}
</script>
Đây là một ví dụ của Chương 6.
</body>
</html>

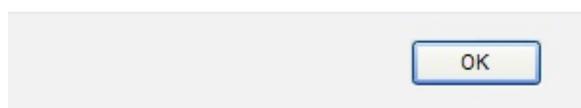
```

3. Kiểm tra tất cả các điều kiện. Bắt đầu bằng việc vào trang từ trình duyệt web. Bạn được thông báo phải nhập vào một số. Trong lần kiểm thử đầu tiên, nhập một từ như dưới đây:



4. Nhấn OK. Điều kiện *if* đầu khớp, sau đó điều kiện *if* lồng bên trong bắt đầu được kiểm tra đầu vào. Đầu vào trùng với chuỗi "bốn", kết quả được thể hiện ở hộp thoại dưới đây:

Dù nó là một số, nó không thực sự là một số đối với tôi.
 Không cho trang này tạo các hộp thoại phụ



5. Nhấn OK để đóng hộp thoại. Tải lại trang. Bây giờ nhập từ pizza:

Hãy nhập một số trong khoảng từ 50 đến 100:

pizza

OK

Hủy bỏ

6. Nhấn OK. Giống với lần chạy trước, điều kiện đầu tiên (`isNaN()`) đúng. Tuy nhiên, vì câu lệnh `if` ở trong không khớp với từ pizza, nên mệnh đề `else` của câu lệnh sẽ được thực thi. Kết quả được thể hiện ở hộp thoại dưới đây:

pizza không phải là một số.

Không cho trang này tạo các hộp thoại phụ

OK

7. Nhấn OK để đóng hộp thoại và mở lại trang. Lần này nhập số 4 vào trường dữ liệu như dưới đây:



Hãy nhập một số trong khoảng từ 50 đến 100:

4

OK

Hủy bỏ

8. Nhấn OK. Bây giờ điều kiện `if` đầu tiên không đúng nữa vì 4 là một số. Do đó, xét tiếp điều kiện `else if`. Vì số 4 nhỏ hơn 51, nên điều kiện `else if` đúng và hiển thị hộp thoại thông báo như sau:

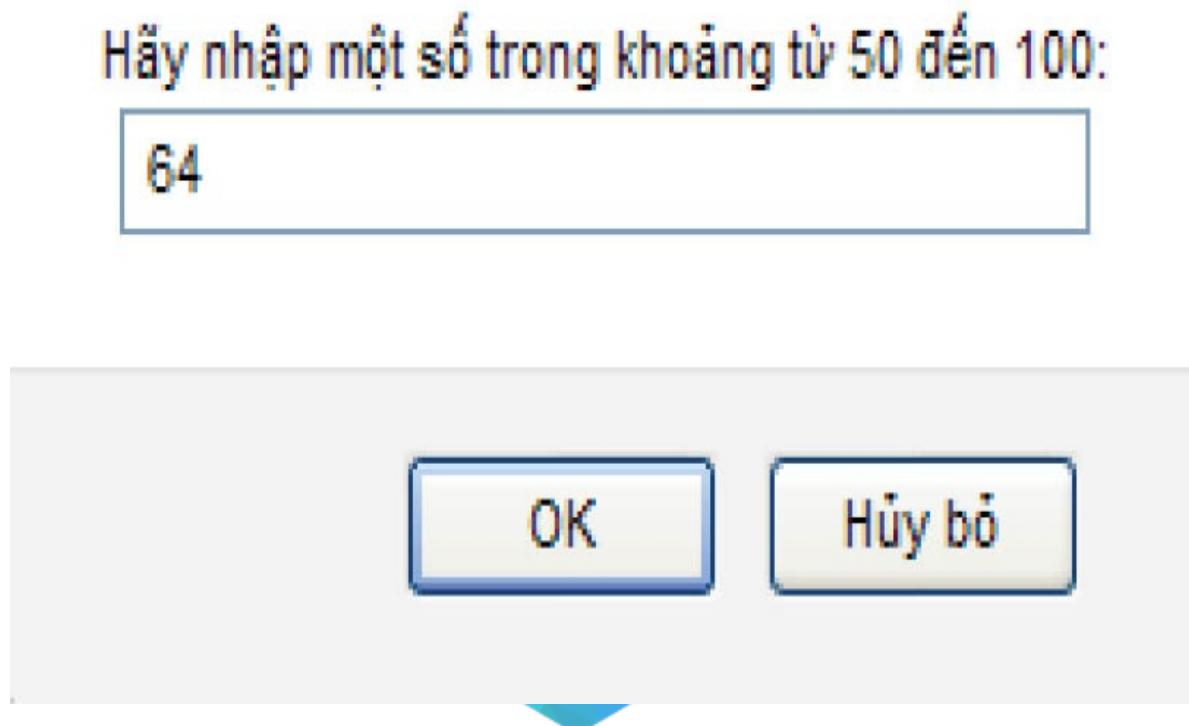
Số 4 không nằm trong khoảng từ 50 đến 100.

Không cho trang này tạo các hộp thoại phụ

OK

9. Một kinh nghiệm hay là bạn nên kiểm thử cả những số lớn hơn 99. Bạn hoàn

toàn có thể làm như thế. Khi sẵn sàng tiếp tục, nhấn OK để đóng hộp thoại và mở lại trang. Lần này, nhập một số như dưới đây:



10. Khi nhấn OK, bạn không nhận được hộp thoại thông báo nào, vì số 64 nằm trong khoảng từ 50 đến 100, vì thế không điều kiện kiểm thử nào đúng.

Tôi đã giải thích đoạn mã bạn đang xem ở bước này và các bước trước nhưng vẫn chưa nói về biểu thức chính quy nằm trong câu lệnh *if* lồng bên trong. Câu lệnh đó là:

```
if (inputNum.match(/một|hai|ba|bốn|năm|sáu|bảy|tám|chín|mười.
```

Biểu thức chính quy được dùng với hàm (hoặc thuộc tính) *match()* của biến *inputNum*. Hàm *match()* nhận một biểu thức chính quy làm đối số. Trong trường hợp này, đối số là:

```
/một|hai|ba|bốn|năm|sáu|bảy|tám|chín|mười/
```

Biểu thức này được chia thành hai phần: một phần là dấu gạch chéo (/) và sau đó là chuỗi "một|hai|ba|bốn|năm|sáu|bảy|tám|chín|mười".

mỗi chuỗi thể hiện một logic *OR*, nghĩa là biểu thức chính quy này sẽ khớp với bất kỳ chuỗi nào trong dãy đó nhưng không nhiều hơn một.

Điểm thú vị là biểu thức chính quy này tuy đơn giản nhưng vẫn có lỗi. Để biểu thức chính quy này hoàn chỉnh hơn, bạn cần đánh dấu vị trí của chuỗi phù hợp. Và trong đoạn mã được viết, biểu thức sẽ khớp với chuỗi *sáu mươi* tương tự như với chuỗi *sáu*.

Mục đích của tôi không phải là chỉ ra một biểu thức chính quy hoàn hảo, mà là để bạn làm quen với một ví dụ hoàn chỉnh để khi cần làm việc với nó, bạn sẽ không bỏ chạy!

Làm việc với điều kiện ba ngôi

Một định dạng khác của cấu trúc điều kiện là *điều kiện ba ngôi* (*ternary conditional*). Loại điều kiện này sử dụng dấu hỏi (?) để tạo cấu trúc *if/else* rút gọn. Cấu trúc cơ bản của điều kiện ba ngôi khá đơn giản.

```
(name == "steve") ? "Xin chào Steve" : "Xin chào người chưa quen"
```

Câu lệnh có thể được diễn giải như sau: Nếu tên bằng steve, “Xin chào Steve”, ngược lại “Xin chào người chưa quen”.

Bạn có thể sử dụng biểu thức ba ngôi trong một câu lệnh, như ví dụ dưới đây (xem trong file ternary.txt của phần Tài nguyên đi kèm):

```
var greeting = (name == "steve") ? "Xin chào Steve" : "Xin chào người chưa quen";
alert(greeting);
```

Đoạn mã thiết lập biến *greeting* với kết quả của phép kiểm tra ba ngôi. Nếu giá trị từ biến name là “Steve”, biến *greeting* sẽ nhận được chuỗi giá trị “Xin chào Steve”; nếu không, biến *greeting* nhận được chuỗi giá trị “Xin chào người chưa quen”: Đây là đoạn mã viết theo hình thức *if/else* truyền thống:

```
(name == "steve") {
    var greeting = "Xin chào Steve"
}
else {
    var greeting = ***"Xin chào người chưa quen***";
```

```
}
```

```
alert(greeting);
```

Cấu trúc ba ngôi đôi khi gây nhầm lẫn nếu bạn chưa từng sử dụng nó. Bạn vẫn có thể viết theo cú pháp *if/else* truyền thống nếu nhờ thế, chương trình của bạn dễ đọc hơn - đặc biệt nếu người đọc mã không biết kiểu cấu trúc ba ngôi này.

Kiểm tra với *switch*

Câu lệnh *switch* là cách dễ dàng và hiệu quả để kiểm tra một biến với nhiều giá trị và thực thi đoạn mã dựa trên trường hợp trùng khớp. Mặc dù bạn cũng có thể thực hiện nhiệm vụ này bằng cách sử dụng câu lệnh *if/else if*, nhưng làm như vậy sẽ khiến đoạn mã dài hơn. Trong trường hợp này, câu lệnh *switch* sẽ hữu dụng hơn.

Xem xét ví dụ về một trang web cần thực thi đoạn mã nào đó dựa trên ngôn ngữ mà người dùng chọn. Với bài tập này, giả định rằng người dùng đã chọn ngôn ngữ của họ qua một form. (Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt” sẽ giải thích cách xác định ngôn ngữ mặc định trong trình duyệt của người dùng).

Nếu trang này cần thực thi đoạn mã với nhiều ngôn ngữ, chúng ta có thể dùng một tập hợp các câu lệnh điều kiện *if/else if/else*. Giả sử ta có một biến là *languageChoice* (tùy chọn ngôn ngữ) với giá trị của ngôn ngữ được chọn, đoạn mã có dạng như sau:

```
if (languageChoice == "en") {
    // Ngôn ngữ là tiếng Anh, thực thi đoạn mã cho tiếng
}
else if (languageChoice == "de") {
    // Ngôn ngữ là tiếng Đức, thực thi đoạn mã cho tiếng
}
else if (languageChoice == "pt") {
    // Ngôn ngữ là tiếng Bồ Đào Nha, thực thi đoạn mã cho
    // tiếng Bồ Đào Nha.
}
else {
    // Ngôn ngữ không được chọn, sử dụng tiếng Thụy Điển
```

}

Đoạn mã này chạy tốt khi có một vài ngôn ngữ được chọn, nhưng bạn hãy thử tưởng tượng tình huống có nhiều hơn hai mươi ngôn ngữ được chọn. Khi đó, việc viết mã để thực thi cho mỗi điều kiện sẽ nhanh chóng bị rối. Đây là đoạn mã tương tự với *switch* (Bạn có thể thấy đoạn mã này trong file switches.txt của phần Tài nguyên đi kèm):

```
switch(languageChoice) {  
    case "en":  
        // Ngôn ngữ là tiếng Anh, thực thi đoạn mã c  
        break;  
    case "de":  
        // Ngôn ngữ là tiếng Đức, thực thi đoạn mã c  
        break;  
    case "pt":  
        // Ngôn ngữ là tiếng Bồ Đào Nha, thực thi đ  
        // cho tiếng Bồ Đào Nha.  
        break;  
    default:  
        // Địa điểm không được chọn, sử dụng tiếng T  
}  
// Chuyển đến đoạn mã sau câu lệnh switch.
```

Câu lệnh *switch* tìm kiếm từng trường hợp ngôn ngữ và thực thi đoạn mã cho trường hợp đó. Câu lệnh *break* cho biết khỏi lệnh *switch* sẽ ngắt sau khi thực thi trường hợp trùng khớp. Câu lệnh *break* khiến toàn bộ phần còn lại của câu lệnh *switch* bị bỏ qua và đoạn mã tiếp tục thực thi sau dấu ngoặc đóng của câu lệnh *switch*.

Ví dụ, nếu biến *languageChoice* có giá trị là *de* và thiếu câu lệnh *break*, đoạn mã cho tiếng Đức sẽ được thực thi, nhưng câu lệnh *switch* vẫn tiếp tục chạy đoạn mã còn lại để kiểm tra các ngôn ngữ khác cho đến khi gặp câu lệnh *break* hoặc thực thi đến cuối câu lệnh *switch*.

Bạn nên dùng câu lệnh *break* với mỗi *case* (trường hợp) trong khỏi lệnh *switch*. Tuy nhiên, tiện ích của câu lệnh *switch* được thể hiện rõ khi có nhiều trường hợp cần thực thi cùng một đoạn mã. Hãy xem một ví dụ, trong đó người truy cập chọn nước hoặc vùng mà họ lưu trú. Trên những trang như thế, người truy cập từ Mỹ, Úc và Vương quốc Anh sẽ muốn trang được hiển thị bằng tiếng Anh, dù người từ những nước đó viết (phát âm) nhiều từ khác nhau. Sau đây là

một ví dụ sử dụng câu lệnh *switch* cho tình huống này (xem trong file switches.txt của Tài nguyên đi kèm):

```
switch(countryChoice) {  
    case "US":  
    case "Australia":  
    case "Great Britain":  
        // Ngôn ngữ là tiếng Anh, thực thi đoạn mã cl  
        break;  
    case "Germany":  
        // Ngôn ngữ là tiếng Đức, thực thi đoạn mã cl  
        break;  
    case "Portugal":  
        // Ngôn ngữ là tiếng Bồ Đào Nha, thực thi đoạn  
        // cho tiếng Bồ Đào Nha.  
        break;  
    default:  
        // Địa điểm không được chọn, sử dụng tiếng Ti  
}  
// Chuyển đến đoạn mã sau câu lệnh switch.
```

Chú ý Tốt hơn hết là bạn nên để người dùng thuộc bất kỳ quốc gia nào cũng có thể chọn ngôn ngữ mà trang web hỗ trợ, ví dụ như tiếng Pháp. Trong ví dụ này, bạn có thể bỏ qua chức năng đó, nhưng hãy ghi nhớ khi bạn thiết kế trang web của mình.

Nếu người dùng chọn quốc gia là Australia, trường hợp (*case*) Australia sẽ trùng khớp, do đó đoạn mã cho tiếng Anh sẽ được thực thi. Tới câu lệnh *break*, JavaScript sẽ nhảy ra khỏi khối lệnh *switch* đang thực thi và chỉ thực thi dòng đầu tiên của đoạn mã sau câu lệnh *switch*.

Vòng lặp *while*

Câu lệnh *while* tạo một vòng lặp, trong đó đoạn mã được thực thi chừng nào điều kiện đúng. Phần này trình bày câu lệnh *while* và câu lệnh *do...while*.

Câu lệnh while

Vòng lặp while thực thi đoạn mã nằm trong dấu ngoặc chừng nào điều kiện còn đúng. Xem ví dụ dưới đây (đoạn mã này nằm trong file while.txt của Tài nguyên đi kèm):

```
var count = 0;  
while (count < 10) {  
    // Viết mã ở đây  
    // Nhiều dòng mã cũng được  
    // Đừng quên tăng biến đếm.  
    count++;  
}
```

Luôn ghi nhớ hai khía cạnh sau của vòng lặp *while*:

- Có thể đoạn mã nằm trong câu lệnh *while* không bao giờ được thực thi, dựa trên giá trị bắt đầu của biến hoặc điều kiện được kiểm tra.
- Điều kiện được kiểm tra bởi câu lệnh *while* phải được thay đổi trong vòng lặp.



Đảm bảo đoạn mã được thực thi ít nhất một lần

Trong đoạn mã ví dụ trước, biến *count* được khởi tạo với giá trị *0*. Câu lệnh *while* sẽ chạy như sau: Câu lệnh *while* sẽ đánh giá giá trị của biến *count* để kiểm tra xem nó có nhỏ hơn *10* hay không. Nhờ thế, đoạn mã trong ngoặc được thực thi. (Tuy nhiên, nếu giá trị của biến *count* không nhỏ hơn *10*, đoạn mã trong ngoặc của vòng lặp *while* không bao giờ được thực thi - dù chỉ một lần).

Trong JavaScript, vòng lặp *do...while* thực thi đoạn mã một lần, bất kể điều kiện ban đầu là gì. Chúng ta sẽ bàn về vòng lặp *do...while* ở phần sau của chương.

Thay đổi điều kiện

Như đã trình bày, câu lệnh *while* trong ví dụ kiểm tra xem biến có nhỏ hơn *10* hay không. Nếu biến *count* nhỏ hơn *10*, đoạn mã trong vòng lặp *while* sẽ được thực thi.

Có một dòng mã trong vòng lặp *while* sử dụng toán tử một ngôi *++* để tăng giá

trị của biến đếm *count*:

```
count++;
```

Khi đoạn mã trong câu lệnh *while* chạy xong, việc kiểm tra được thực hiện lại. Nếu không có đoạn mã làm tăng giá trị của biến *count*, *count* luôn nhỏ hơn 10 sẽ dẫn đến một vòng lặp vô hạn – đây không phải là điều chúng ta muốn.

Mách nhỏ Khi sử dụng một biến đếm, như trong ví dụ trên, việc tăng biến đó có thể được thực hiện bên trong cặp ngoặc nhọn của câu lệnh *while* hoặc trong chính điều kiện của câu lệnh *while*. Đây là một ví dụ: *while* (*i*++ < 10). Xem lại Chương 5 để hiểu rõ hơn về toán tử chèn sau.

Bài học rút ra là bạn phải chắc chắn sẽ tăng hoặc thay đổi bất cứ điều kiện gì mà bạn sử dụng trong câu lệnh *while*.

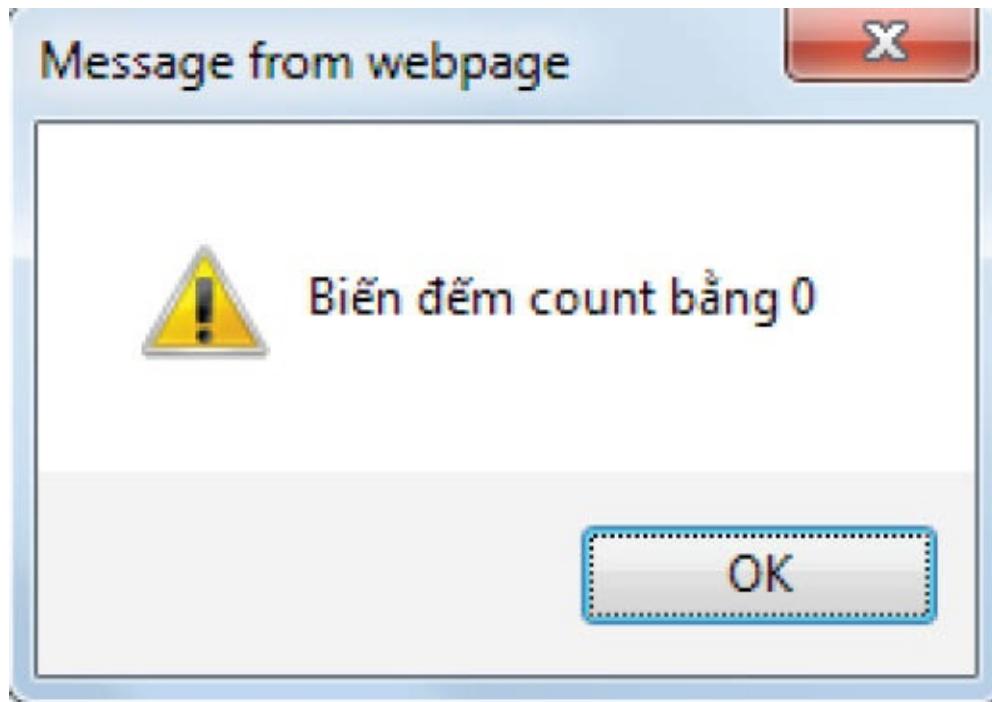
Câu lệnh do...while

Không như câu lệnh *while*, câu lệnh *do...while* thực thi đoạn mã nằm trong dấu ngoặc ít nhất một lần. Câu lệnh *while* có thể được diễn giải như sau: "Khi điều kiện đúng, chạy đoạn mã này". Mặt khác, câu lệnh *do...while* được diễn giải như sau: Thực thi đoạn mã này cho đến khi điều kiện còn đúng. Xem đoạn mã dưới đây (trong file dowhile.htm của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Sử dụng vòng lặp Do While</title>  
</head>  
<body>  
    <script type="text/javascript">  
        var count = 0;  
        do {  
            alert("Biến đếm count bằng " + count);  
            count++;  
        }  
        while (count < 3);  
    </script>
```

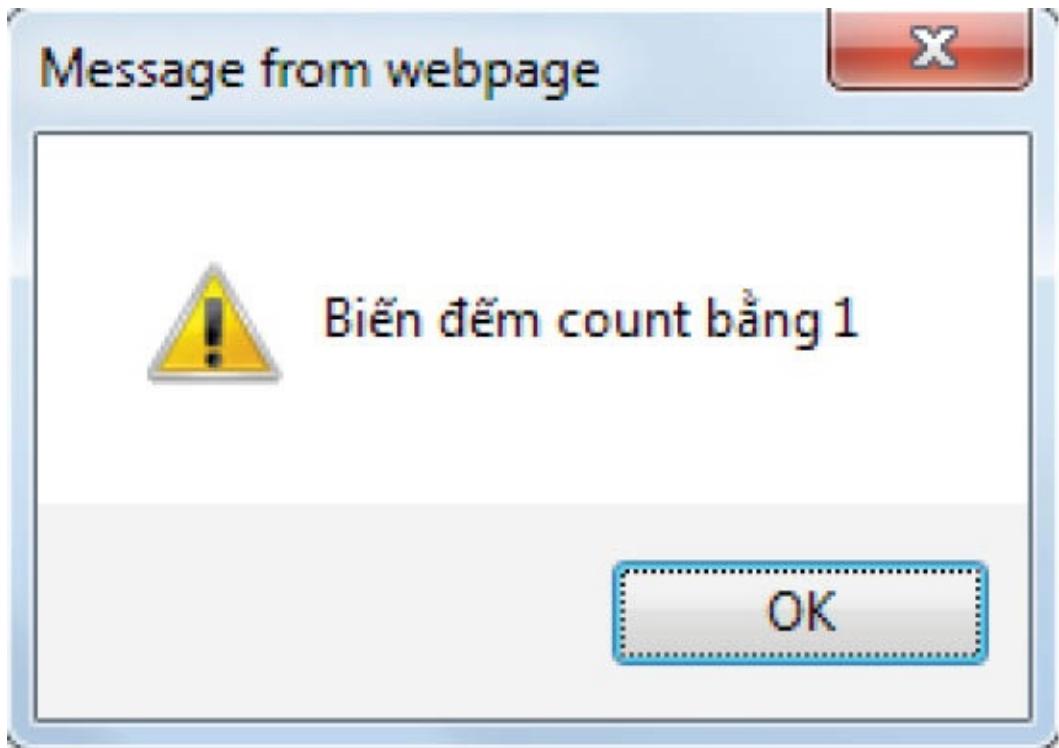
```
</body>  
</html>
```

Khi thực thi đoạn mã này, ba hộp thoại xuất hiện. Trong lần chạy đầu tiên, biến *count* có giá trị 0 bởi vì đó vẫn là giá trị ban đầu và hộp thoại sẽ hiển thị như Hình 6-4.



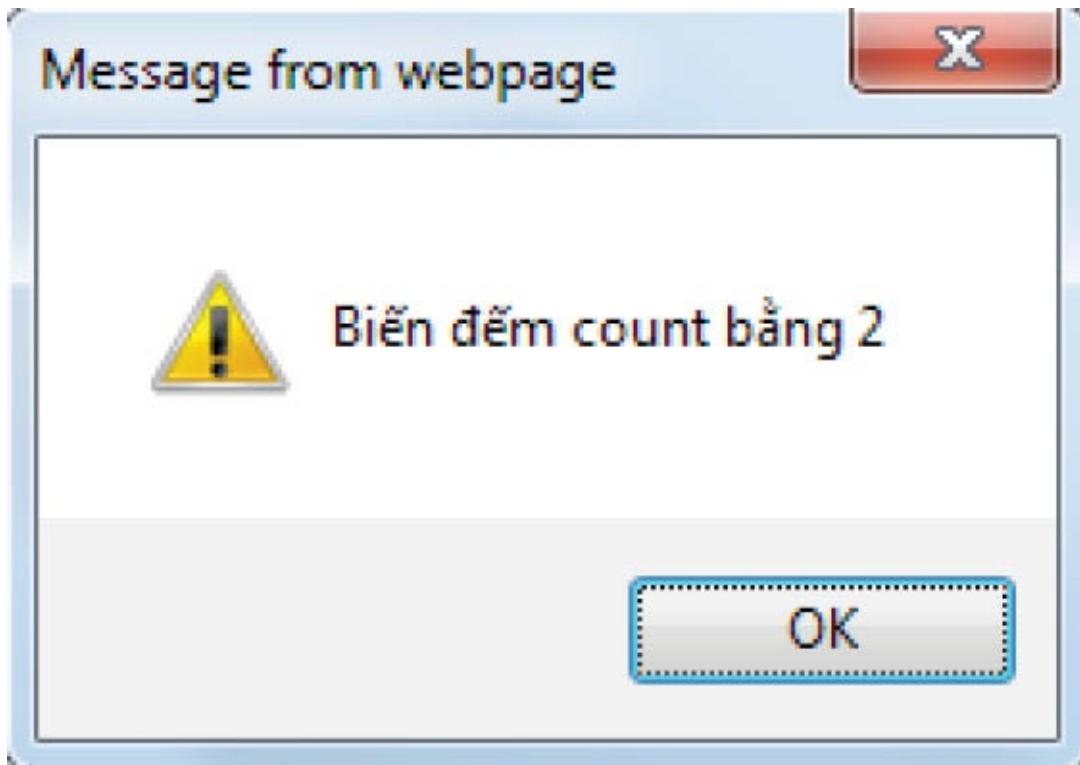
HÌNH 6-4 Biến count có giá trị 0 trong lần thực thi đầu tiên.

Sau khi chạy một lần, biến *count* được tăng lên. Khi kiểm tra tiếp câu lệnh *while*, biến *count* vẫn nhỏ hơn 3, vì thế đoạn mã tiếp tục được thực thi, kết quả hiển thị ở hộp thoại như Hình 6-5.



HÌNH 6-5 Khi chạy, đoạn mã tăng biến đếm và cho ra kết quả của lần thực thi tiếp theo.

Quá trình tương tự lặp lại. Biến *count* được tăng lên và điều kiện *while* tiếp tục được kiểm tra. Giá trị của biến *count* vẫn còn nhỏ hơn 3, vì thế đoạn mã trong ngoặc tiếp tục chạy, hiển thị hộp thoại khác như Hình 6-6.



HÌNH 6-6 Biến count sau lần chạy khác

Thực hành với câu lệnh `while` và `do...while` cho đến khi bạn quen và hiểu được sự khác biệt giữa hai câu lệnh này.

Sử dụng vòng lặp `for`

Vòng lặp `for` thường được dùng tương tự như vòng lặp `while` để thực thi một đoạn mã lặp lại một số lần nhất định. Trong JavaScript, vòng lặp `for` có hai dạng: vòng lặp `for...in` và vòng lặp `for each...in`. Phần này sẽ giải thích cả hai loại vòng lặp trên.

Vòng lặp `for`

Chúng ta sử dụng vòng lặp `for` để tạo một vòng lặp trong đó điều kiện được khởi tạo, đánh giá và thay đổi trong một cấu trúc ngắn gọn. Dưới đây là ví dụ:

```
for (var count = 0; count < 10; count++) {  
    // Thực thi đoạn mã dưới đây
```

}

Câu lệnh *for* có ba mệnh đề trong ngoặc đơn. Mệnh đề đầu tiên thiết lập biểu thức ban đầu như trong ví dụ trên và cả ví dụ dưới đây:

```
var count = 0;
```

Mệnh đề tiếp theo của câu lệnh *for* chỉ định biểu thức kiểm tra, thể hiện bởi đoạn mã sau:

```
count < 10;
```

Biểu thức cuối cùng thường dùng để tăng biến đếm sử dụng cho việc kiểm tra. Trong đoạn mã ví dụ, biểu thức này là mệnh đề cuối trong ngoặc đơn:

```
count++
```

Chú ý Biểu thức cuối cùng trong cấu trúc của vòng lặp *for* không có dấu chấm phẩy.

Dưới đây là ví dụ sử dụng vòng lặp *for* để duyệt mảng.

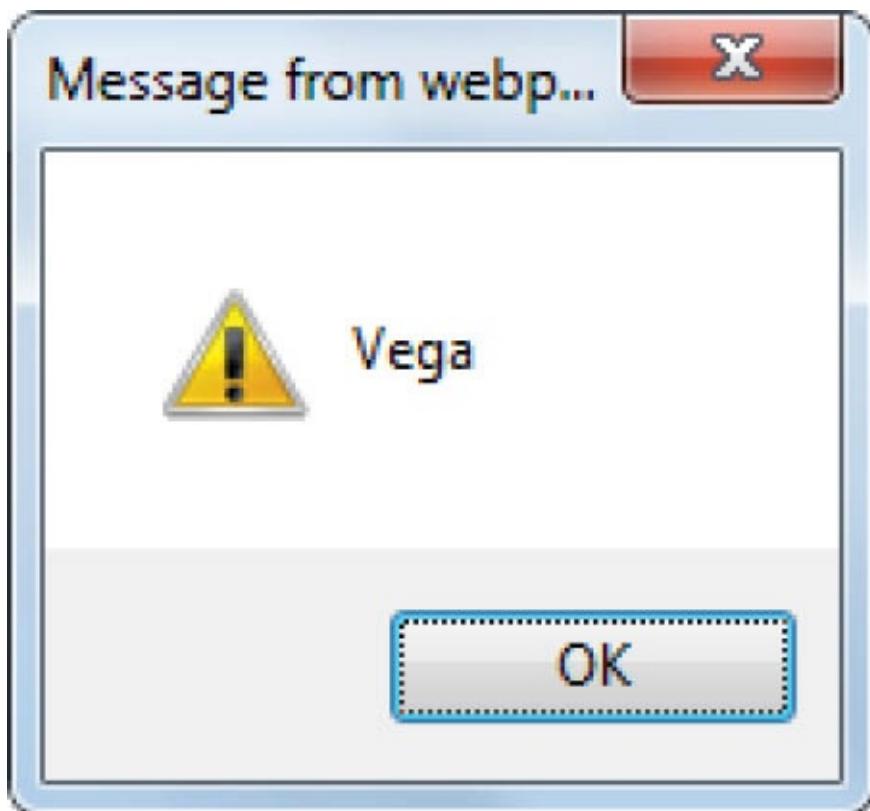
Sử dụng vòng lặp *for* với mảng

1. Sử dụng Visual Studio, Eclipse, hoặc một trình soạn thảo khác, thay đổi đoạn mã trong file forloop.htm của thư mục mã nguồn mẫu Chương 6, phần Tài nguyên đi kèm.
2. Trong trang này, thay dòng chú thích TODO bằng đoạn mã in đậm dưới đây (bạn có thể tìm thấy đoạn mã này trong file forloop.txt của thư mục Chương 6, Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Ví dụ về vòng lặp For</title>  
</head>  
<body>  
<script type="text/javascript">
```

```
var myArray = ["Vega", "Deneb", "Altair"];
var arrayLength = myArray.length;
for (var count = 0; count < arrayLength; count++ ) {
    alert(myArray[count]);
}
</script>
</body>
</html>
```

3. Lưu trang này và dùng trình duyệt mở lại. Bạn nhận được ba hộp thoại thông báo liên tiếp nhau:





Có thể thấy từ những hộp thoại này, đoạn mã chạy lặp lại với mỗi giá trị trong

mảng *myArray*. Tôi muốn phân tích rõ một số đoạn mã trong ví dụ này. Ở Chương 4, bạn đã được học về cách tạo mảng và biết rằng mảng trong JavaScript được đánh chỉ số bằng số nguyên bắt đầu từ 0. (Sau đây, bạn sẽ có điều kiện áp dụng những kiến thức này). Đây là dòng tương ứng trong ví dụ trước:

```
var myArray = ["Vega", "Deneb", "Altair"];
```

Đoạn mã tạo ra một biến là *arrayLength* và thiết lập giá trị độ dài của mảng. Cách lấy độ dài của mảng *myArray* minh họa cho cách dùng thuộc tính *length*. (Tôi sẽ giải thích chi tiết hơn về các đối tượng trong Chương 8 "Các đối tượng trong JavaScript"). Lấy độ dài trong một biến riêng (trong đoạn mã trên đây là *arrayLength*) thay vì sử dụng thuộc tính *length* trong vòng lặp for giúp tăng tốc độ thực thi chương trình.

```
var arrayLength = myArray.length;
```

Đầu tiên vòng lặp *for* tạo và đặt giá trị cho biến *count*, sau đó kiểm tra xem liệu biến *count* có nhỏ hơn độ dài của mảng *myArray* hay không (độ dài này được lưu trong biến *arrayLength*). Cuối cùng, nó tăng giá trị của biến *count*. Đoạn mã trong nội dung của vòng lặp for hiển thị một hộp thoại thông báo, sử dụng biến *count* để duyệt qua từng chỉ số của mảng *myArray*. Dưới đây là đoạn mã:

```
for (var count = 0; count < arrayLength; count++) {  
    alert(myArray[count]);  
}
```

Vòng lặp *for...in*

Vòng lặp *for...in* duyệt qua các thuộc tính của một đối tượng, trả về tên của chính các thuộc tính đó. Dưới đây là ví dụ:

```
for (var myProp in myObject) {  
    alert(myProp + " = " + myObject[myProp]);  
}
```

Trong đoạn mã này, biến *myProp* thiết lập một thuộc tính mới cho *myObject* mỗi khi thực thi vòng lặp. Dưới đây là ví dụ hoàn chỉnh hơn.

Sử dụng vòng lặp *for...in*

1. Dùng Visual Studio, Eclipse hoặc một trình soạn thảo khác thay đổi file forinloop.htm trong thư mục mã nguồn mẫu Chương 6 của Tài nguyên đi kèm.
2. Trong trang này, thay dòng chú thích TODO bằng đoạn mã in đậm dưới đây (đoạn mã này cũng nằm trong file forinloop.txt của phần Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Ví dụ về vòng lặp For...In</title>
</head>
<body>
<script type="text/javascript">
    var star = new Object();
    star.name = "Polaris";
    star.type = "Double/Cepheid";
    star.constellation = "Ursa Minor"; //constellation
    for (var starProp in star) {
        alert(starProp + " = " + star[starProp]);
    }
</script>
</body>
</html>
```

3. Lưu và dùng trình duyệt mở trang. Bạn nhận được ba hộp thoại thông báo sau:

Message from webpage



name = Polaris

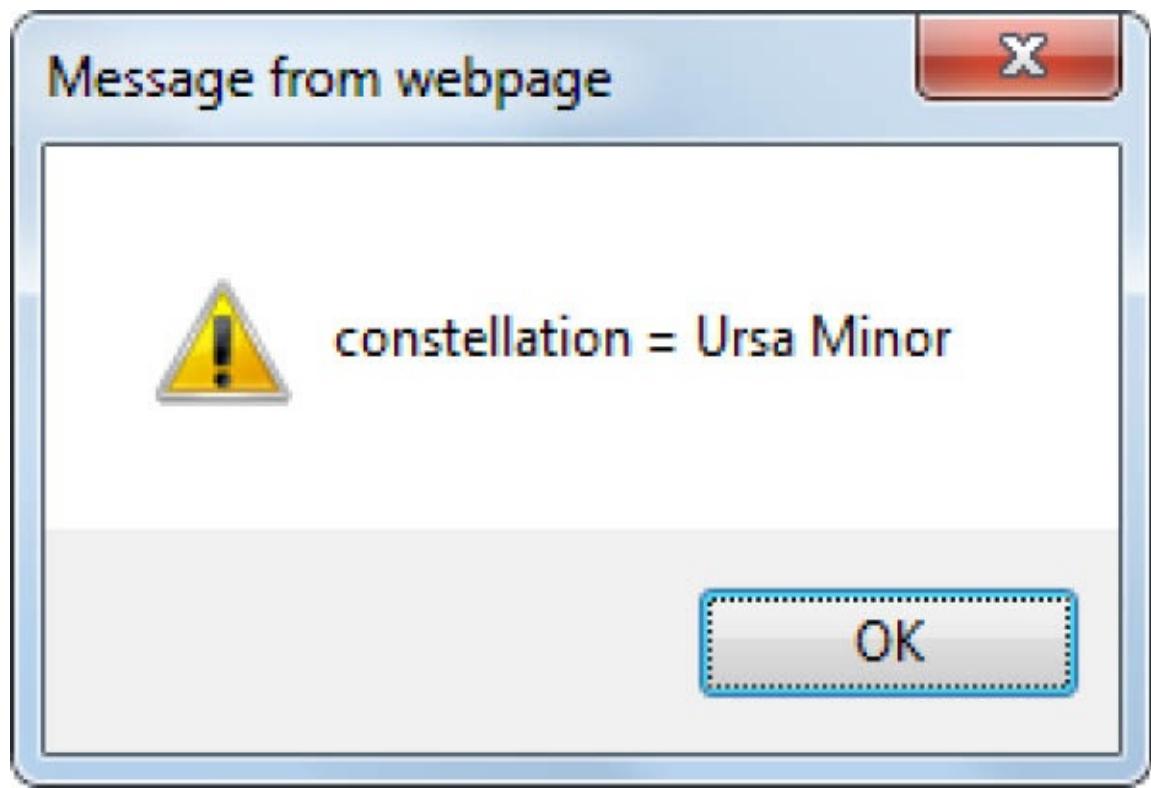
OK

Message from webpage



type = Double/Cepheid

OK



Trong đoạn mã ví dụ, biến `starProp` nhận tên của thuộc tính, trái lại khi sử dụng `starProp` dưới dạng chỉ số của đối tượng `star` sẽ cho ra giá trị của thuộc tính đó.

Mách nhỏ Đôi khi, bạn sẽ thấy vòng lặp `for...in` được dùng để duyệt một mảng như trong ví dụ ở phần trước. Tuy nhiên việc dùng `for...in` để duyệt mảng có thể dẫn tới nhiều kết quả lẩn lộn. Một trong những vấn đề dễ thấy nhất của phương thức này là vòng lặp `for...in` không trả về các thuộc tính theo thứ tự cố định nào. Đây có thể là vấn đề, đặc biệt khi bạn muốn dùng JavaScript để làm hiển thị chữ trên một trang web. Vì vậy, khi bạn muốn duyệt một mảng đơn giản bằng vòng lặp, hãy sử dụng vòng lặp `for` thay vì vòng lặp `for...in`.

Vòng lặp `for each...in`

Một cấu trúc mới trong JavaScript là vòng lặp `for each...in`. Vì cấu trúc này mới nên vẫn chưa được hỗ trợ trong tất cả các trình duyệt – đặc biệt nó không được hỗ trợ trong IE8 và các phiên bản cũ hơn. Tuy nhiên, nó được hỗ trợ trong

Firefox 2.0 và các phiên bản mới hơn.

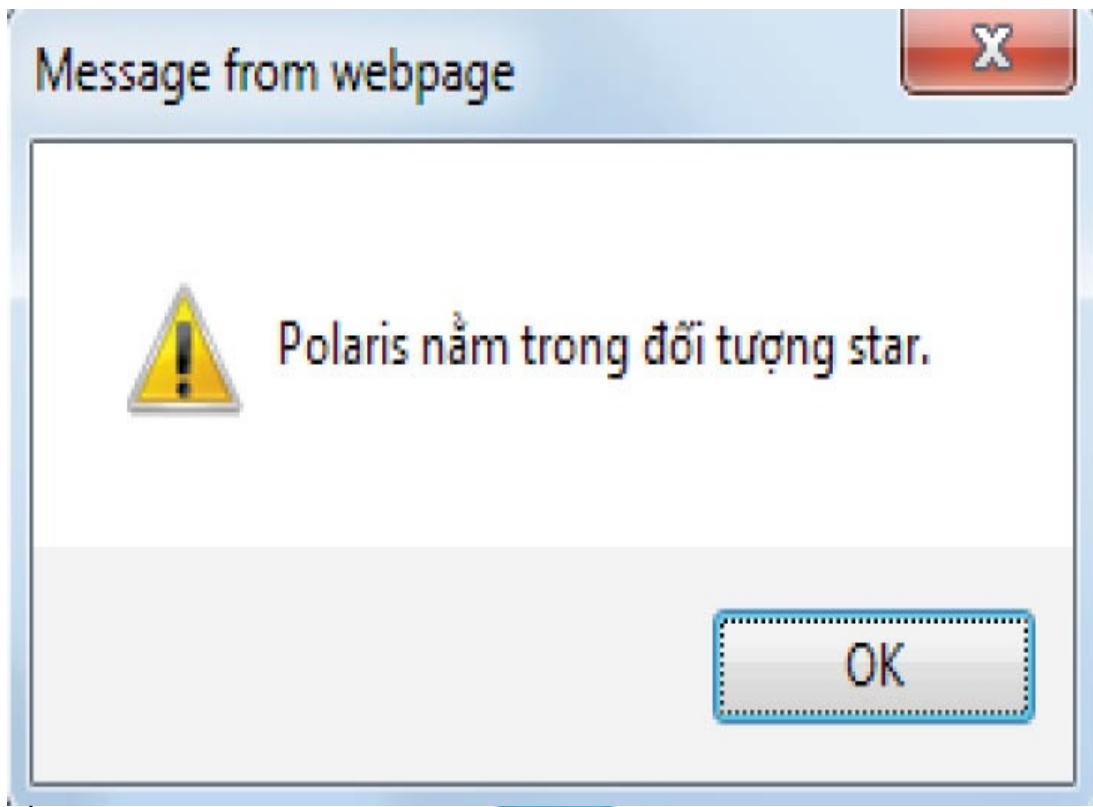
Cấu trúc `for...in` trả về tên của thuộc tính, còn vòng lặp `for each...in` trả về giá trị của thuộc tính. Về cơ bản, cú pháp của cả hai giống nhau, vòng lặp `for each...in` chỉ thêm từ `each`:

```
for each (var myValue in myObject) {  
    alert(myValue " nằm trong đối tượng.");  
}
```

Thay thế vòng lặp `for...in` từ ví dụ trước bằng vòng lặp `for each...in`, kết quả sẽ là đoạn mã in đậm dưới đây: (File đã chỉnh sửa là `foreach.htm` của phần Tài nguyên đi kèm).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Ví dụ về vòng lặp For Each In</title>  
</head>  
<body>  
<script type="text/javascript">  
    var star = new Object;  
    star.name = "Polaris";  
    star.type = "Double/Cepheid";  
    star.constellation = "Ursa Minor";  
    for each (var starValue in star) {  
        alert(starValue + " nằm trong đối tượng.");  
    }  
</script>  
</body>  
</html>
```

Khi bạn xem trang web bằng Internet Explorer, bạn sẽ nhận được hộp thoại báo lỗi (hoặc chỉ nhận được một màn hình trắng). Tuy nhiên, nếu xem trang này bằng FireFox 3.0, đoạn mã sẽ thực thi đúng. Hình 6-7 hiển thị một trong ba hộp thoại.



HÌNH 6-7 Duyệt đối tượng sử dụng vòng lặp *for each...in*.

Có thể, bạn chưa muốn dùng cấu trúc của vòng lặp *for each...in* bởi vì nó không được hỗ trợ trong các phiên bản cũ (cũng là phiên bản vẫn được dùng phổ biến) của Internet Explorer.

Kiểm tra tính hợp lệ của form với các lệnh điều kiện

Ở các phần trước, chúng ta dùng hàm *prompt()* để lấy thông tin từ người dùng. Hàm *prompt()* ít được dùng và nhanh chóng lỗi thời vì bị chặn bởi Internet Explorer 7. Phần này trình bày qua việc sử dụng các web form với JavaScript. Toàn bộ Chương 14 "Sử dụng JavaScript với Web Form" sẽ thảo luận đề tài này.

Việc sử dụng điều kiện *if else ... if else* để kiểm tra tính hợp lệ của dữ liệu đầu vào khá phổ biến, vì thế chúng ta hãy thử làm bài tập sau.

Kiểm tra tính hợp lệ của dữ liệu đầu vào với câu lệnh điều kiện

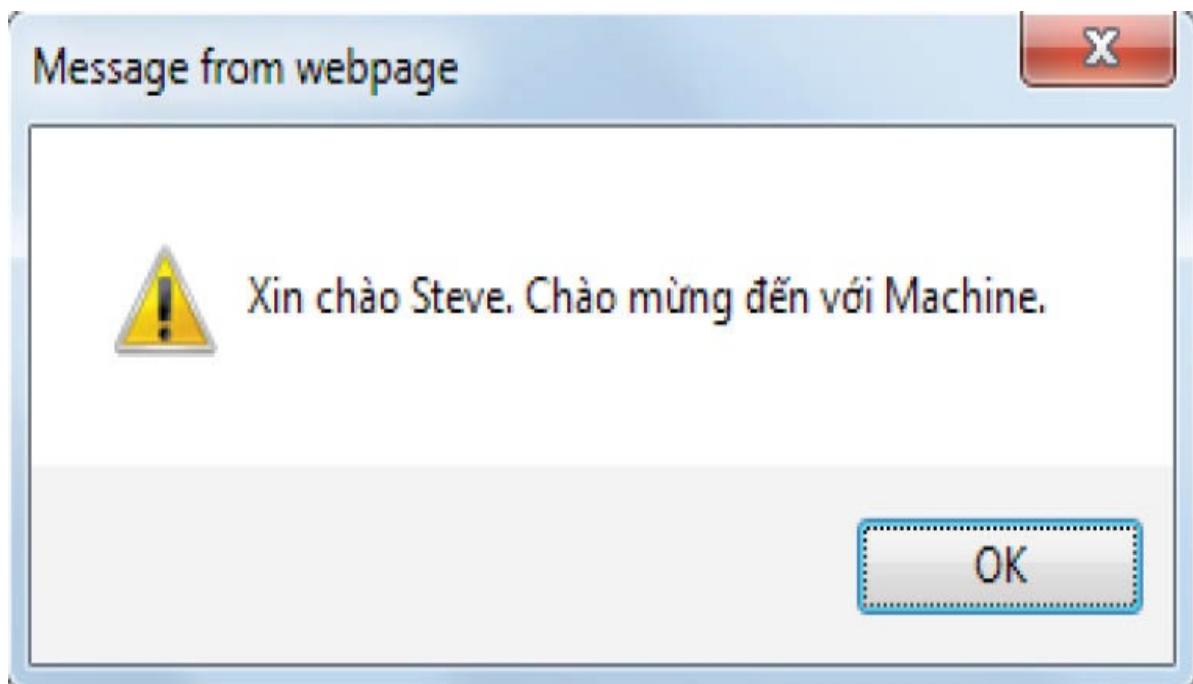
1. Sử dụng Visual Studio, Eclipse hoặc một trình soạn thảo văn bản khác tạo một trang web. Đặt tên cho file này là: form1example.htm.
2. Trong trang này, nhập đoạn mã sau và thêm đoạn mã in đậm dưới đây thay cho dòng chú thích TO DO (xem trong file form1example.txt của phần Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Form đơn giản</title>
    <script type="text/javascript">
        function alertName() {
            var name = document.forms[0].nametext.value;
            if (name == "steve") {
                alert("Xin chào Steve. Chào mừng bạn!");
            }
            else if (name == "nancy") {
                alert("Xin chào Tim.");
            }
            else {
                alert("Xin chào " + name);
            }
            return true;
        } //Kết thúc hàm**
    </script>
</head>
<body>
<form id="myform" action="#" onsubmit="return alertName();">
    <p>Username: <input id="nametext" name="username" type="text" />
    <p><input type="submit" /> </p>
</form>
</body>
</html>
```

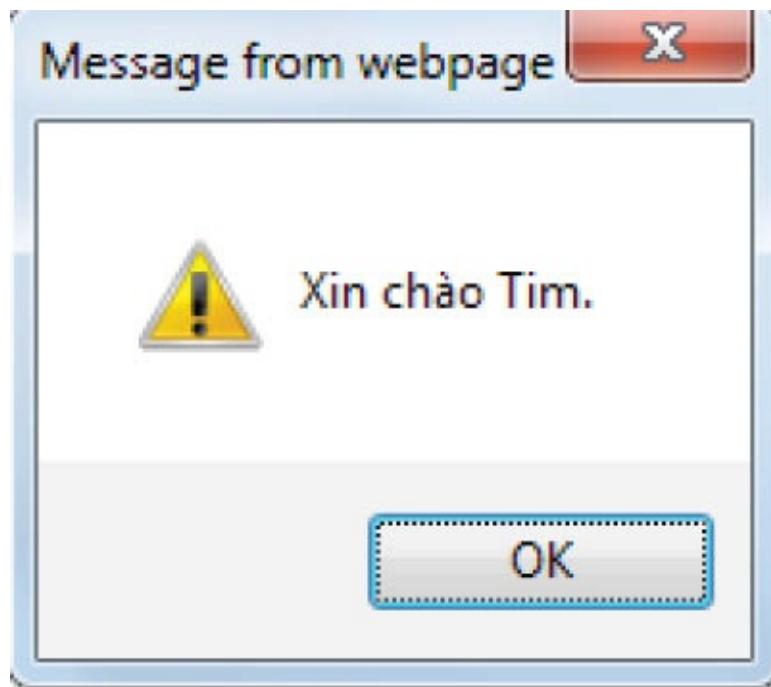
3. Lưu trang này và dùng trình duyệt web để mở lại. Bạn sẽ thấy một trang như sau:

A screenshot of a web browser window titled "Form cơ bản". The address bar shows the URL "http://localhost/ex...". The main content area contains a form with a label "Username:" followed by an empty input field, and a "Submit Query" button below it.

- Trong form, nhập tên **steve**, không dùng dấu ngoặc kép và chỉ sử dụng ký tự thường. Nhấn Submit Query (Gửi yêu cầu) và bạn sẽ nhận được một hộp thoại như sau:

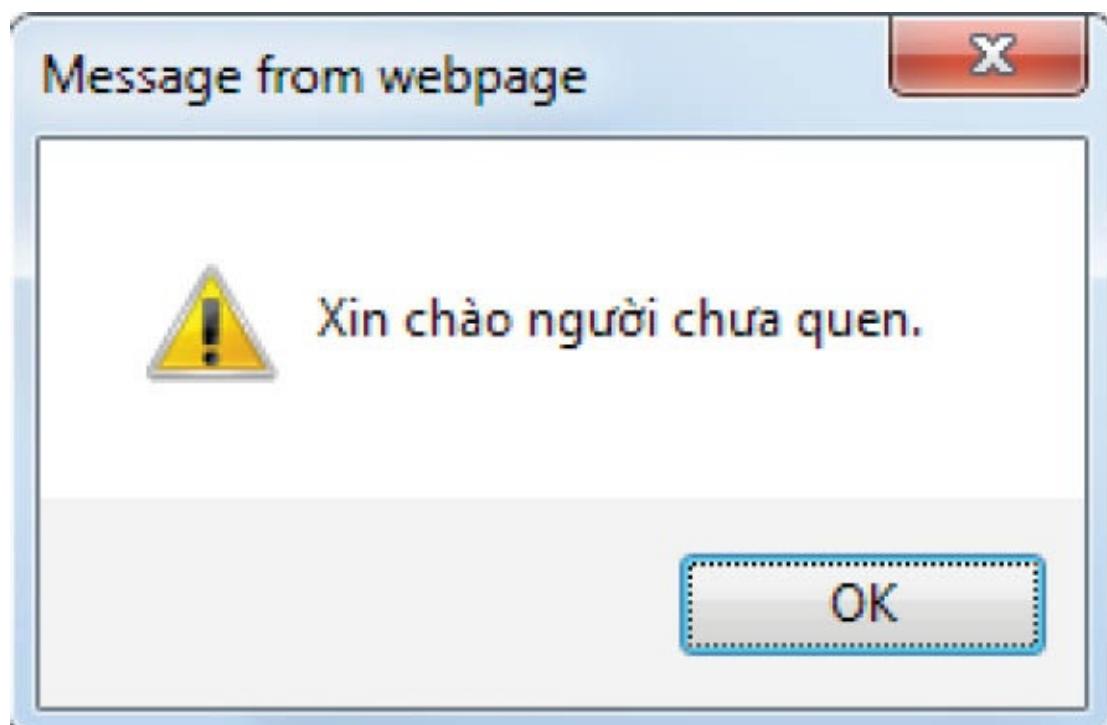


- Nhấn OK và nhập tên **nancy**, không có dấu ngoặc kép và bằng ký tự thường. Nhấn Submit Query, bạn sẽ nhận được một hộp thoại như sau:



Trong trường hợp bạn còn thắc mắc, nickname của Tim là Nancy, vì thế thông báo trong hộp thoại là có nghĩa. Điều này cũng đúng với đoạn mã trong ví dụ.

- Nhấn OK để đóng hộp thoại. Nhập một tên khác và nhấn vào button Submit Query. Bạn sẽ nhận được hộp thoại này:



7. Nhấn OK để đóng hộp thoại.

Bạn đã tạo một form cơ bản cho web, truy cập form đó bằng JavaScript, và dùng câu lệnh điều kiện để thực hiện thao tác dựa trên dữ liệu đầu vào của người dùng. Đừng lo lắng nếu cách dùng trong ví dụ này không dễ hiểu. Mục đích chính của ví dụ này là đưa ra một vài trường hợp áp dụng câu lệnh điều kiện bạn đã học ở chương này.

Chương 7 “Làm việc với hàm” sẽ giải thích cách sử dụng hàm trong JavaScript. Ví dụ đầu tiên trong Chương 14 sẽ cho thấy cách viết một đoạn mã JavaScript kiểm tra tính hợp lệ để bảo đảm các trường bắt buộc được điền đầy đủ.

Bài tập

1. Sử dụng hàm *prompt()* để lấy thông tin tên của một người. Sử dụng câu lệnh *switch* để thực thi hộp thoại hiển thị câu “Chào mừng <ten nhap vao>” nếu tên nhập vào là tên của bạn, “Đi đi” nếu tên nhập vào là Steve, và “Vui lòng trở lại sau, <ten nhap vao>” cho toàn bộ các trường hợp khác
2. Sử dụng hàm *prompt()* để lấy thông tin nhiệt độ hiện tại được nhập bởi người truy cập. Nếu nhiệt độ nhập vào trên 100, yêu cầu người dùng giảm nhiệt độ. Nếu nhiệt độ dưới 20, yêu cầu người dùng tăng nhiệt độ.
3. Sử dụng câu lệnh ba ngôi để hoàn thành nhiệm vụ ở Bài tập 2.
4. Sử dụng vòng lặp *for* để đếm từ 1 đến 100. Khi số là 99, hiển thị hộp thoại thông báo.
5. Sử dụng vòng lặp *while* để hoàn thành nhiệm vụ tương tự ở Bài tập 4.

Chương 7

Làm việc với hàm

Sau khi đọc xong chương này, bạn có thể:

- Hiểu được mục đích sử dụng hàm trong JavaScript.
- Tự định nghĩa hàm.
- Gọi hàm và nhận dữ liệu trả về.
- Nắm được cách sử dụng một số hàm có sẵn trong JavaScript.

Hàm là gì?



Hàm trong JavaScript là tập hợp các câu lệnh, hàm có thể có tên hoặc không (vô danh), và có thể được gọi từ bất kỳ đâu trong một chương trình JavaScript. Hàm có thể nhận vào các đối số, đây là những giá trị đầu vào được truyền vào hàm. Trong thân hàm, các đối số sẽ được thao tác và kết quả sẽ được trả về tại vị trí hàm được gọi thông qua giá trị *trả về*.

Hàm là lựa chọn hoàn hảo khi đoạn mã lặp lại nhiều lần trong chương trình. Thay vì viết lại đoạn mã đó, bạn dùng hàm (có thể hiểu là một chương trình con bên trong chương trình) để thay thế. Thậm chí khi bạn có những đoạn mã rất giống nhau – dù không giống hoàn toàn – trong suốt chương trình, bạn vẫn có thể khai quát hóa chúng thành một hàm.

Sử dụng hàm để kiểm tra các trường bắt buộc phải điền vào form là một ví dụ cho việc khai quát hóa những đoạn mã tương tự nhau. Bạn có thể viết mã JavaScript để kiểm tra cho từng trường trong form hoặc sử dụng hàm. Chương 14 “Sử dụng JavaScript với Web Form” đưa ra ví dụ xây dựng một hàm cụ thể và sau đó khai quát hóa nó.

Qua các ví dụ của những chương trước, bạn đã làm quen với hoạt động của hàm. Một hàm được định nghĩa bằng từ khóa *function*, theo sau là tên hàm và cặp

ngoặc đơn chứa danh sách các tham số. Chúng ta dùng cặp ngoặc nhọn để giới hạn các câu lệnh được thực thi trong phạm vi của hàm:

```
function tenHam() {  
    // Viết mã cho hàm ở đây  
}
```

Mách nhỏ Cần lưu ý rằng khi hàm được định nghĩa (bạn có thể thấy điều này trong định nghĩa hàm cơ bản ở phần trước), đoạn mã sẽ chưa được chạy cho đến khi hàm được gọi. Cách gọi hàm sẽ được trình bày chi tiết ở phần sau của chương này.

Đối số của hàm

Đối số truyền vào hàm được đặt trong cặp ngoặc đơn của định nghĩa hàm. Dưới đây là ví dụ về cách dùng đối số:

```
function myFunction(doiSo1, doiSo2, ..., doiSoN) {  
}
```



Ví dụ tiếp theo có hai đối số:

```
function myFunction(doiSo1, doiSo2) {  
    // Viết mã ở đây  
}
```

Thao tác gọi hàm đơn giản như sau:

```
myFunction(val1, val2);
```

Một trong những điểm khác biệt giữa JavaScript (theo chuẩn ECMA-262) và các ngôn ngữ khác là trong JavaScript bạn không cần chỉ ra số lượng đối số truyền vào hàm, và số lượng đối số được truyền vào không bắt buộc phải khớp với danh sách đối số đã định nghĩa của hàm. Khi được gọi, hàm nhận vào một đối tượng (hoạt động như một mảng) có tên là *arguments*. *Arguments* chứa các đối số đã truyền vào hàm, giúp bạn biết số lượng đối số được truyền vào. Sau đây là ví dụ minh họa cách thức hoạt động của đối số (bạn có thể tìm thấy đoạn mã này trong file functionbasics.txt của phần Tài nguyên đi kèm):

```
function myFunction() {  
    var firstArg = arguments[0];  
    var secondArg = arguments[1];  
}
```

Cách tốt hơn là bạn lấy ra độ dài của đối tượng *arguments* và chạy vòng lặp qua từng đối số như sau (đoạn mã này cũng có trong file functionbasics.txt):

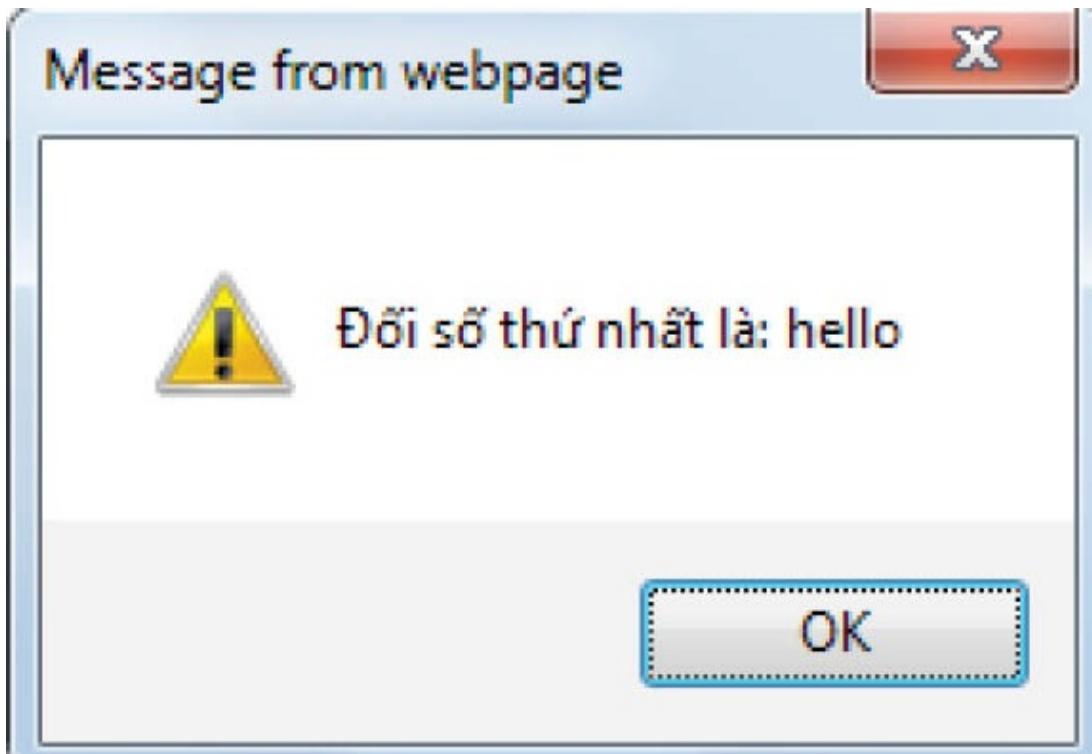
```
function myFunction() {  
    //truy xuất độ dài của đối tượng arguments  
    var argLength = arguments.length;  
    for (var i = 0; i < argLength; i++) {  
        // Xử lý từng đối số (i)  
    }  
}
```

Sau đây là ví dụ hoàn chỉnh hơn minh họa tác dụng của việc sử dụng đối tượng *arguments*

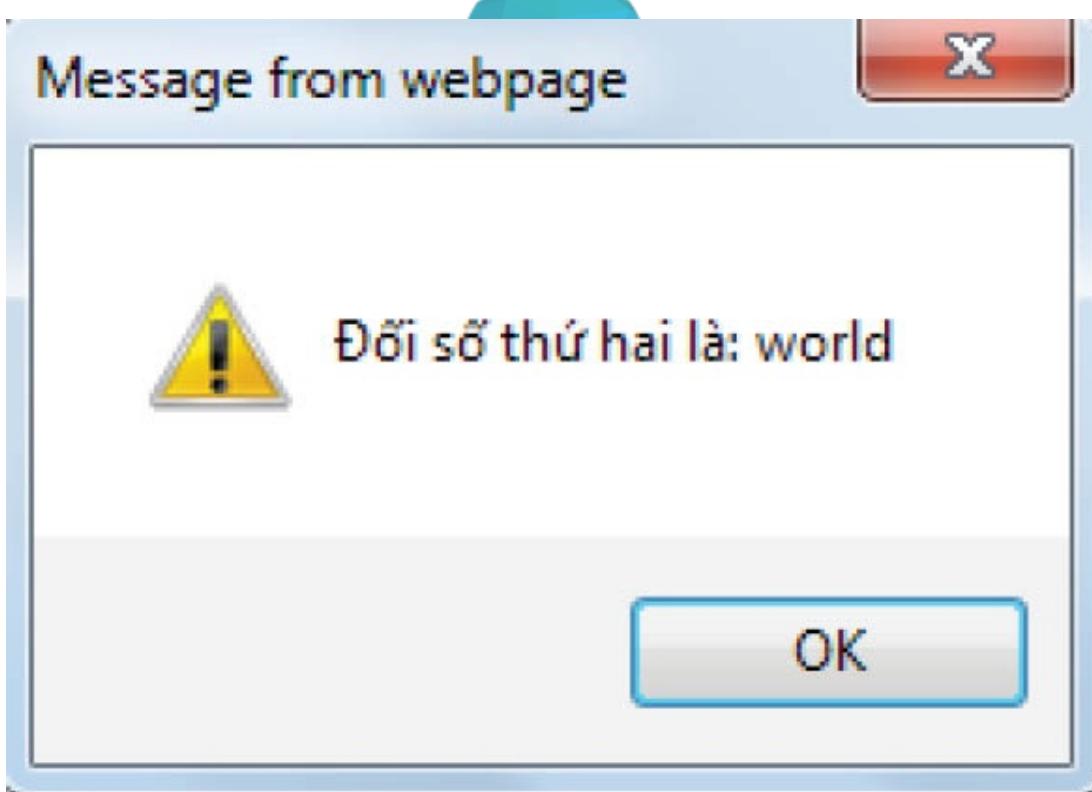
(đoạn mã có trong file functionexample.htm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Mảng đối số</title>  
</head>  
<body>  
<script type="text/javascript">  
function myFunction() {  
    var firstArg = arguments[0]; // Đối số thứ 1  
    var secondArg = arguments[1]; // Đối số thứ 2  
    alert("Đối số thứ nhất là: " + firstArg);  
    alert("Đối số thứ hai là: " + secondArg);  
}  
myFunction("hello", "world");  
</script>  
</body>  
</html>
```

Khi đoạn mã chạy, hai hộp thoại thông báo sẽ xuất hiện, như minh họa trong Hình 7-1 và 7-2.



HÌNH 7-1 Dùng đối tượng `arguments` trong hàm để truy xuất đối số thứ nhất.



HÌNH 7-2 Dùng đối tượng `arguments` trong hàm để truy xuất đối số thứ hai.

Sử dụng đối tượng *arguments* theo cách này, bạn có thể truy xuất bất cứ đối số

nào, chứ không chỉ hai đối số kể trên.

Ôn lại phạm vi của biến

Đối số của hàm thường là tên biến và chúng ta không nên đặt tên đối số giống tên biến của lời gọi hàm. Tôi chủ định dùng từ “*không nên*” thay cho “*không được*” vì bạn có thể sử dụng cùng một tên cho biến trong hàm và biến ở lời gọi hàm, nhưng làm như vậy có thể gây nhầm lẫn trong đoạn mã và phạm vi của biến.

Chương 4 “Làm việc với biến và kiểu dữ liệu” có một phần nói về phạm vi của biến, bao gồm một bài tập về phạm vi biến trong và ngoài hàm. Sau đây là đoạn mã từ ví dụ về phạm vi biến trong Chương 4 (và được lặp lại trong file scopingrevisit.txt của phần Tài nguyên đi kèm):

```
<head>
    <title>Ví dụ về phạm vi biến</title>
    <script type="text/javascript">
        var aNewVariable = "có phạm vi toàn cục.";
        function doSomething(incomingBits) {
            alert("Biến toàn cục bên trong hàm: " + aNewVariable);
            alert("Biến cục bộ bên trong hàm: " + incomingBits);
        }
    </script>
</head>
<body>
<script type="text/javascript">
    doSomething("có phạm vi cục bộ.");
    alert("Biến toàn cục bên ngoài hàm: " + aNewVariable);
    alert("Biến cục bộ bên ngoài hàm: " + incomingBits);
</script>
</body>
```

Ví dụ này cho thấy bạn có thể khai báo và xác định phạm vi biến toàn cục và cục bộ bên trong và ngoài một hàm. Tuy nhiên, ví dụ này giữ cho các biến tách biệt về logic ở điểm đoạn mã không dùng cùng tên cho biến và sau đó chỉ thay đổi giá trị biến. Tiếp theo là ví dụ trong đó việc dùng cùng tên biến gây ra sự nhầm lẫn. Đây là đoạn mã tôi viết nhiều năm trước và nó đủ rắc rối dù không có các vần đề về phạm vi, vì vậy tránh viết như sau:

```
function addNumbers() {  
    firstNum = 4;  
    secondNum = 8;  
    result = firstNum + secondNum;  
    return result;  
}  
result = 0;  
sum = addNumbers();
```

Có lẽ bạn đã nhìn ra vấn đề trong đoạn mã này. Đoạn mã thiếu từ khóa *var*. Dù đoạn mã có khởi tạo biến *result* với giá trị *0* bên ngoài hàm, biến này lại được thay đổi sau lời gọi hàm *addNumber()*. Hàm *addNumber()* thay đổi giá trị biến *result* thành *12*, kết quả của phép cộng *4* và *8* bên trong hàm.

Nếu bạn thêm dòng lệnh *alert* để hiển thị giá trị biến *result* ngay sau khi khởi tạo, hộp thoại sẽ cho ra kết quả *0*. Và nếu bạn thêm một lệnh *alert* khác để hiển thị giá trị biến *result* sau khi gọi hàm *addNumbers()*, thông báo sẽ hiển thị là *12*. Việc thêm các lệnh *alert* vào đúng vị trí sẽ được thực hiện trong bài tập ở phần sau.

Tóm lại, mọi thứ sẽ suôn sẻ hơn khi bạn dùng tên khác nhau cho các biến ở bên trong và ngoài hàm và luôn sử dụng từ khóa *var* để khởi tạo các biến. Tùy thuộc vào đoạn mã bên trong hàm, hàm có thể có hoặc không có giá trị trả về. Giá trị trả về được chuyển lại cho nơi gọi hàm như bạn sẽ thấy ở phần tiếp theo.

Giá trị trả về

Khi hàm kết thúc quá trình thực thi, nó có thể trả về giá trị cho nơi gọi hàm bằng cách sử dụng từ khóa *return*. Hãy xem xét Ví dụ 7-1 (ví dụ này có trong file listing7-1.txt của Tài nguyên đi kèm).

Ví dụ 7-1 Ví dụ về giá trị trả về đơn giản.

```
function multiplyNums(x) {  
    return x * 2;  
}  
var theNumber = 10;  
var result = multiplyNums(theNumber);  
alert(result);
```

Ví dụ 7-1 tạo một hàm có tên *multiplyNums* với giá trị đầu vào được gán cho biến *x*. Hàm có chức năng: trả về giá trị gấp đôi của đối số:

```
function multiplyNums(x) {  
    return x * 2;  
}
```

Sau đó, biến *theNumber* được khởi tạo:

```
var theNumber = 10;
```

Tiếp theo, một biến nữa có tên *result* được tạo. Biến này lưu kết quả của lời gọi hàm *multiplyNums*. Hàm *multiplyNums* sử dụng biến *theNumber* với vai trò đối số.

```
var result = multiplyNums(theNumber);
```

Khi chạy, đoạn mã hiển thị hộp thông báo như trong Hình 7-3.



Hình 7-3 Hộp thoại thông báo hiển thị giá trị trả về từ lời gọi hàm.

Bạn có thể trả về giá trị từ bất cứ vị trí nào trong hàm chứ không chỉ ở cuối hàm. Sử dụng lệnh *return* trong khối lệnh điều kiện hoặc sau vòng lặp là cách làm

phổ biến, ví dụ như sau (bạn có thể tìm thấy đoạn mã này trong file morereturnexamples.txt của Tài nguyên đi kèm):

```
function myFunction(x) {  
    if (x == 1) {  
        return true;  
    } else {  
        return false;  
}
```

Tuy vậy, bạn cần chú ý vị trí đặt lệnh *return*, vì khi gặp lệnh *return*, hàm sẽ trả lại giá trị và không thực thi những dòng mã sau đó. Ví dụ, đoạn mã như sau (bạn có thể tìm thấy đoạn mã này trong file morereturnexamples.txt) sẽ không thực thi như ý định của bạn:

```
function myFunction() {  
    var count = 0;  
    var firstNum = 48;  
    return;  
    var secondNum = 109;  
}
```



Đoạn mã này sẽ không bao giờ khởi tạo biến *secondNum*.

Bổ sung về thao tác gọi hàm

Bạn sẽ luôn gọi hàm với một vài đối số hoặc chỉ với cặp ngoặc trống như sau:

```
var result = orderFruit();
```

Nếu hàm đòi hỏi phải có đối số, lời gọi hàm sẽ như sau:

```
var result = orderFruit(type, quantity);
```

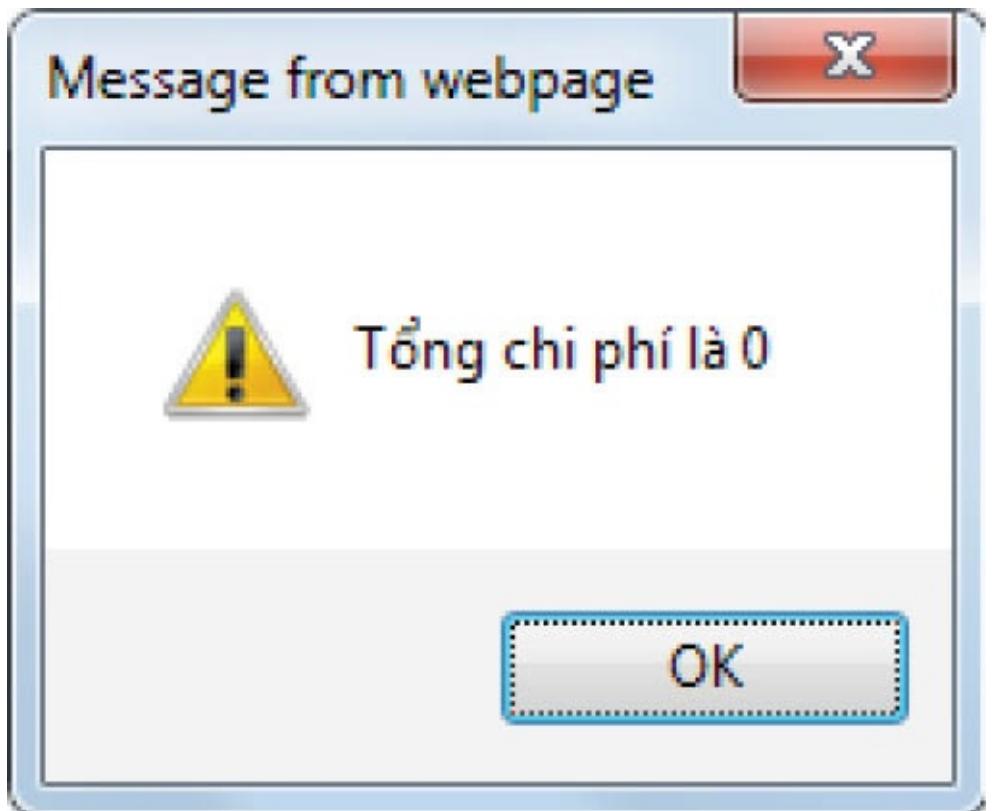
Bỏ qua cặp ngoặc đơn khi gọi hàm có thể gây ra những kết quả hoàn toàn khác so với ý định của bạn. Việc gọi hàm không có cặp ngoặc đơn sẽ trả lại tên của hàm, thay vì giá trị trả về của hàm. Hơn nữa, hàm đó sẽ không được thực thi.

Sau đây là một ví dụ. Ví dụ 7-2 minh họa một đoạn mã JavaScript cơ bản (bạn có tìm thấy trong file listing7-2.htm của phần Tài nguyên đi kèm).

Ví dụ 7-2 Gọi hàm.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Đặt hoa quả</title>
    <script type="text/javascript">
        function orderFruit() {
            var total = 0;
            // Gọi hàm khác để đặt đơn hàng
            return total;
        }
    </script>
</head>
<body>
    <script type="text/javascript">
        var result = orderFruit();
        Alert("Tổng chi phí là " + result);
    </script>
</body>
</html>
```

Khi chạy, đoạn mã này gọi đến hàm *orderFruit()*. Hàm *orderFruit()* gọi một hàm khác (không có trong ví dụ) để tạo đơn đặt hàng. Sau đó, tổng chi phí được tính và trả lại cho hàm *goi*. Đoạn mã trên sẽ hiển thị hộp thoại thông báo như trong Hình 7-4.

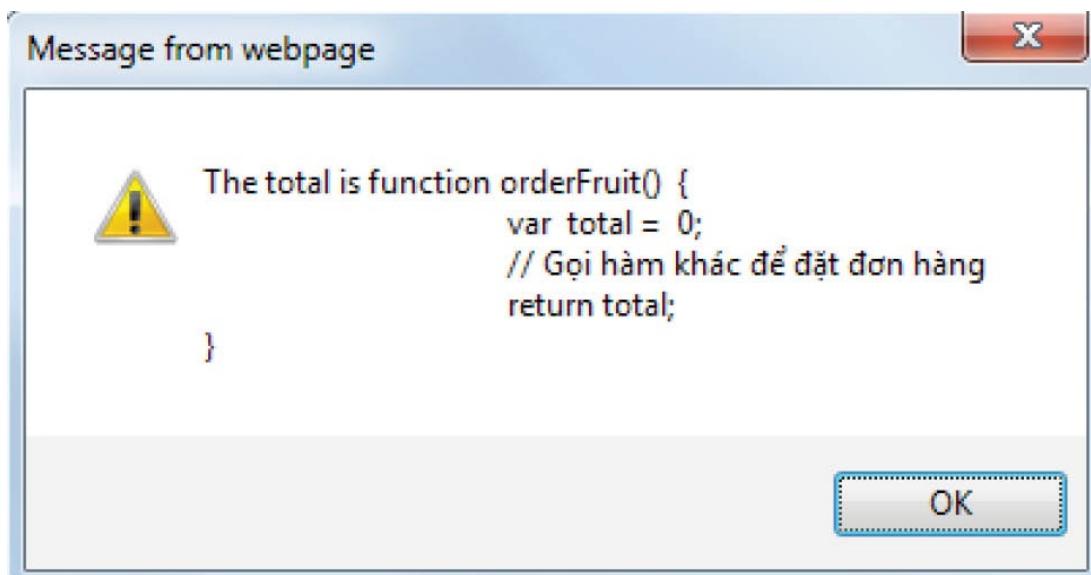


Hình 7-4 Gọi hàm `orderFruit()` với cặp ngoặc đơn trả lại kết quả như dự tính.

Chỉnh sửa một chút – cụ thể là xóa **cặp dấu ngoặc đơn** khỏi lời gọi hàm sẽ làm thay đổi toàn bộ kết quả:

```
var result = orderFruit;
```

Kết quả được thể hiện trong Hình 7-5.



Hình 7-5 Gọi hàm `orderFruit` không có cặp ngoặc đơn sẽ tạo kết quả không như mong muốn.

Bất kể hàm có trả về giá trị hoặc nhận đối số hay không, việc gọi hàm với cặp ngoặc đơn là thao tác quan trọng.

Hàm vô danh/không được đặt tên

Các hàm bạn đã thấy ở trên đều được định nghĩa theo chuẩn. Tuy nhiên, JavaScript không đòi hỏi hàm phải được định nghĩa theo cách đó. Ví dụ, với cách tạo hàm không tên hay còn gọi là hàm vô danh, hàm có thể được định nghĩa và gắn với biến như sau:

```
var divNums = function(firstNum, secondNum) { return firstNum
```

Bạn có thể dễ dàng kiểm tra hàm này với giả giao thức javascript:. Hãy nhập dòng mã sau vào thanh địa chỉ trên trình duyệt của bạn:

```
javascript:var divNums = function(firstNum, secondNum) {  
    return firstNum / secondNum;};  
alert(divNums(8,2));
```

Các hàm vô danh thường được dùng trong JavaScript hướng đối tượng và dùng làm hàm xử lý sự kiện. Bạn có thể xem ví dụ về cách sử dụng này trong Chương 8 “Các đối tượng trong JavaScript” và trong các chương sau.

Bao đóng - Closure

Trong JavaScript, các hàm con bên trong có quyền truy cập đến các biến của hàm cha bên ngoài. *Closure* chỉ sự tồn tại của các biến bên ngoài ngữ cảnh thực thi thông thường của hàm. Closure thường được vô ý tạo ra và có thể gây rò rỉ bộ nhớ trong Windows Internet Explorer nếu không được xử lý đúng cách. Tuy nhiên, closure lại là một trong những tính năng mạnh (và cao cấp) trong JavaScript.

Xem xét ví dụ sau:

```
function myFunction() {  
    var myNum = 10;  
    function showNum() {  
        alert(myNum);
```

```
        }
    return showNum();
}
var callFunc = myFunction();
myFunction();
```

Trong ví dụ này, hàm *showNum* có quyền truy cập tới biến *myNum* được tạo ở hàm bên ngoài, hàm *myFunction*. Biến *callFunc* được tạo trong ngữ cảnh toàn cục và chứa một tham chiếu tới hàm *showNum*. Khi biến *callFunc* được tạo, nó lập tức có quyền truy cập tới biến *myNum*.

Closure có thể được sử dụng để giả lập phương thức private bên trong đối tượng, và có thể áp dụng trong các hàm xử lý sự kiện. Closure là một trong những khái niệm mạnh và nâng cao trong JavaScript, bởi vậy đề tài này không thích hợp để thảo luận chi tiết trong một cuốn sách cơ bản. Bạn có thể tìm hiểu thêm về closure tại địa chỉ <http://msdn.microsoft.com/en-us/scriptjunkie/ff696765.aspx> và từ những nguồn thông tin khác trên Internet.

Phương thức



Phương thức được hiểu một cách đơn giản nhất chính là các hàm được định nghĩa bên trong một đối tượng. Bạn có thể truy cập phương thức của một đối tượng bằng toán tử dấu chấm ("."). Các đối tượng có sẵn trong JavaScript, như *Math*, *Date*, và *String*, đều có các phương thức như bạn đã thấy (hoặc sẽ thấy) trong cuốn sách này. Các hàm như hàm *alert()* thực chất là các phương thức của đối tượng *window* và có thể được viết là *window.alert()* thay vì chỉ là *alert()*. Chương 8 sẽ thảo luận chi tiết hơn về đối tượng và phương thức.

Chú ý Trong nhiều phần của cuốn sách, tác giả dùng tương đương hai thuật ngữ *phương thức* và *hàm*. Như vậy, bạn có thể hiểu rằng ranh giới giữa hai khái niệm này trong hầu hết trường hợp rất mờ nhạt. Khi một hàm được sử dụng theo cách thức hướng đối tượng, dùng thuật ngữ phương thức sẽ rõ ràng hơn. Khi không trực tiếp dùng trong cách thức hướng đối tượng – ví dụ, cách bạn dùng hàm *alert()* – thuật ngữ hàm là chấp nhận được.

Tự định nghĩa hàm và sử dụng các hàm có sẵn

Như bạn đã thấy xuyên suốt cuốn sách, JavaScript có rất nhiều hàm/phương thức tích hợp sẵn. Ngoài việc sử dụng các hàm có sẵn, bạn sẽ cần tự tạo hàm. Ngoại trừ những ứng dụng hết sức đơn giản, hầu hết ứng dụng bạn viết sẽ dùng những hàm tự tạo.

Tuy nhiên, trong một số trường hợp, bạn có thể định nghĩa hàm và sau đó phát hiện ra JavaScript đã có sẵn một hàm với chức năng tương tự. Nếu bạn biết hàm có sẵn trong JavaScript thực hiện cùng chức năng với hàm của bạn, dùng hàm có sẵn của JavaScript thường là phương án tối ưu hơn.

Sơ lược về các hàm hội thoại

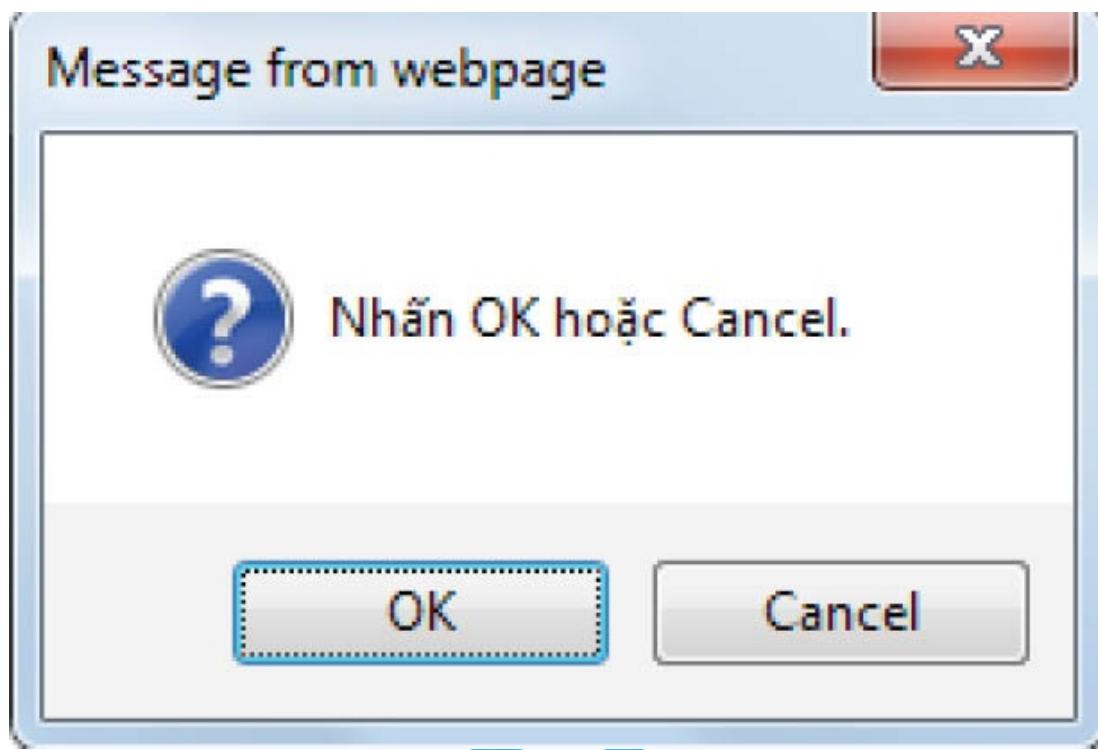
Cho tới giờ, bạn đã biết tất cả về hàm `alert()` trong JavaScript. Bạn cũng đã biết rằng hàm `alert()` thực chất là một phương thức của đối tượng `window`. Phần này giới thiệu cách dùng phổ biến của hàm `alert()` trong JavaScript và hai hàm liên quan khác của đối tượng `window`.

Thông tin bổ sung Đối tượng `window` rất quan trọng và sẽ được trình bày kỹ hơn trong Chương 9 “Mô hình đối tượng trình duyệt”. Chương 9 cũng thảo luận về nhiều phương thức khác của đối tượng `window`.

Mặc dù đối tượng `window` có nhiều phương thức, nhưng trong phần này, chúng ta sẽ xem xét ba phương thức (cũng có thể gọi là hàm) sau `alert()`, `confirm()` và `prompt()`. Vì bạn đã thấy quá nhiều hộp thoại `alert()` trong cuốn sách, chúng ta sẽ không đưa nó vào đây nữa. Chương 6 “Điều khiển luồng với câu lệnh điều kiện và vòng lặp” đã trình bày cách sử dụng hàm `prompt()` và việc Internet Explorer 7 mặc định chặn hàm này vì lý do bảo mật. Dù vậy, hàm `confirm()` vẫn có trong Internet Explorer.

Hàm `confirm()` hiển thị một hộp thoại trạng thái với hai button OK và Cancel, như minh họa trong Hình 7-6. (Hộp thoại trạng thái ngăn các hành động khác

trên trình duyệt cho đến khi người dùng đóng hộp thoại bằng cách nhấn OK hoặc Cancel).



Hình 7-6 Hàm `confirm()` hiển thị hộp thoại yêu cầu người dùng xác nhận hành động.

Khi bạn nhấn OK, hàm *confirm()* trả về *true*. Khi bạn nhấn Cancel, hàm *confirm()* trả về *false*.

Giống như *alert()* và *prompt()*, hàm *confirm()* tạo một hộp thoại trạng thái trên hầu hết các platform (nền tảng). Nếu được sử dụng quá nhiều hoặc được dùng không đúng lúc, các hàm này sẽ gây khó chịu cho người dùng. Nếu được dùng hợp lý để cung cấp phản hồi quan trọng và tiếp nhận thông tin thiết yếu, những hàm này khá hữu dụng.

Mách nhỏ Đừng dùng hàm *confirm()* thay cho form để tiếp nhận thông tin đầu vào trên trang web. Form tốt hơn nhiều về mặt điều hướng và thường được người dùng dễ dàng chấp nhận hơn.

Bài tập tiếp theo hướng dẫn bạn sử dụng hàm *confirm()* để nhận thông tin đầu vào và đưa ra quyết định dựa trên những thông tin đó.

Xác nhận thông tin với hàm *confirm()*

1. Dùng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác mở file confirm.htm trong thư mục mã nguồn mẫu Chương 7 của Tài nguyên đi kèm.
2. Trong trang này, thay thế dòng chú thích TO DO bằng đoạn mã in đậm sau (bạn có thể tìm thấy đoạn mã này trong file confirm.txt của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC " -//W3C//DTD HTML 4.01//EN "
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Xác nhận thông tin</title>
    <script type="text/javascript">
        function processConfirm(answer) {
            var result = "";
            if (answer) {
                result = "Tuyệt. Chúng ta sẽ chơi
            } else {
                result = "Có thể một lúc nữa.";**
            }
            return result;
        }
    </script>
</head>
<body>
<script type="text/javascript">
var confirmAnswer = confirm("Chúng ta chơi một ván chứ?");
var theAnswer = processConfirm(confirmAnswer);
**alert(theAnswer);**
</script>
</body>
</html>
```

3. Lưu và dùng trình duyệt mở trang. Bạn sẽ thấy một hộp thoại giống như sau:

Message from webpage

X



Chúng ta chơi một ván chứ?

OK

Cancel

- Nhấn OK. Bạn sẽ thấy một hộp thoại `alert()`:

Message from webpage

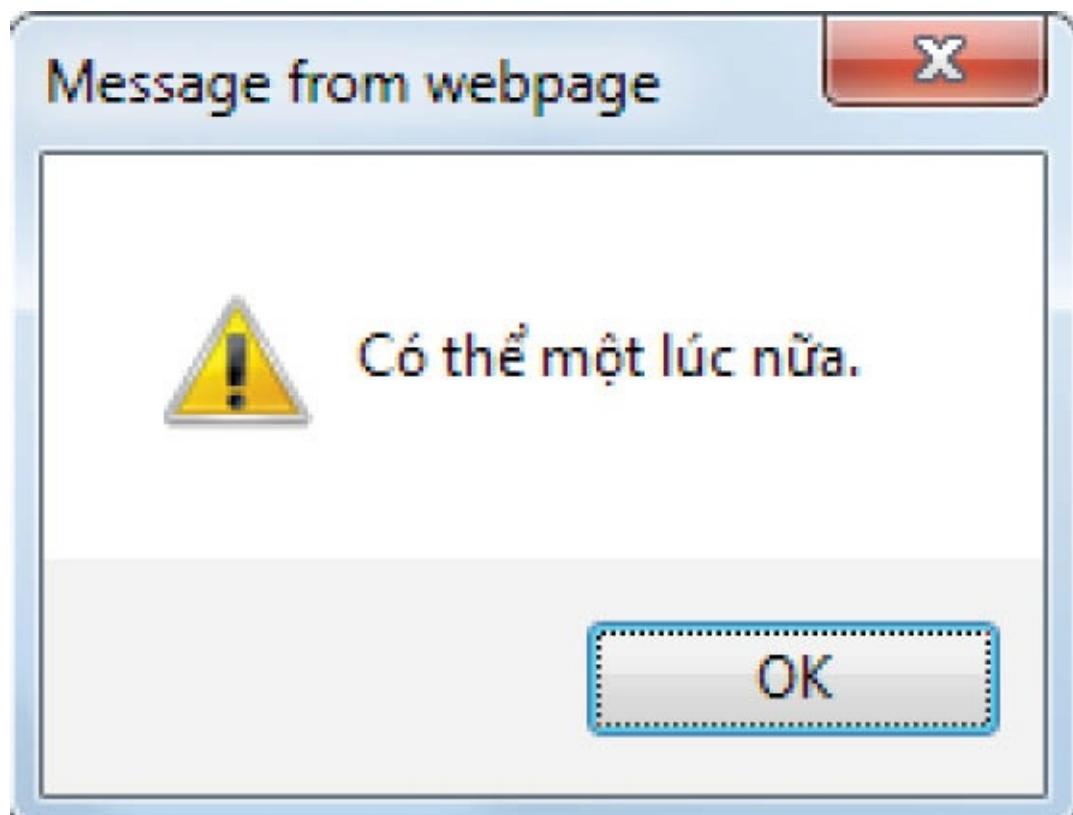
X



Tuyệt. Chúng ta sẽ chơi một ván cờ hay.

OK

- Nhấn OK và sau đó tải lại trang web.
- Hộp thoại `confirm()` hiện lên, hỏi bạn có muốn chơi một ván cờ không. Lần này, bạn chọn Cancel. Hộp thoại `alert()` khác sẽ hiện ra:



7. Nhấn OK để đóng hộp thoại.



Đoạn mã có hai phần chính cần chú ý, một trong thẻ `<head>` và một trong thẻ `<body>`. Hàm `processConfirm(answer)` được tạo trong phần `<head>` của trang web:

```
function processConfirm(answer) {  
    var result = "";  
    if (answer) {  
        result = "Tuyệt. Chúng ta sẽ chơi một ván cờ  
    } else {  
        result = "Có thể một lúc nữa."  
    }  
    return result;  
}
```

Hàm này đánh giá giá trị có trong đối số `answer`. Nếu giá trị của biến `answer` là `true`, đồng nghĩa với việc người dùng chọn OK, hàm sẽ tạo biến `result` và gán giá trị cho biến `result` là chuỗi “Tuyệt. Chúng ta sẽ chơi một ván cờ hay”. Nếu giá trị của biến `answer` là `false`, đồng nghĩa với việc người dùng chọn Cancel, hàm sẽ vẫn tạo biến `result` nhưng gán giá trị cho biến `result` là chuỗi “Có thể

một lúc nữa". Bất kể biến *answer* lưu giá trị gì, hàm *processConfirm* trả về biến *result* cho hàm gọi bằng câu lệnh *return* bên trong hàm. Bạn có thể viết hàm này ngắn gọn như sau:

```
function processConfirm(answer) {  
    if (answer) {  
        return "Tuyệt. Chúng ta sẽ chơi một ván cờ h  
    } else {  
        result = "Có thể một lúc nữa.";  
    }  
}
```

Và thậm chí ngắn gọn hơn nữa:

```
function processConfirm(answer) {  
    var result;  
    (answer) ? result = "Tuyệt. Chúng ta sẽ chơi một ván  
    return result;  
}
```

Chú ý Tôi sẽ chọn sử dụng hàm *processConfirm* cuối cùng. Tuy nhiên, nhiều lập trình viên không thoải mái lắm với kiểu điều kiện ba ngôi như trong ví dụ trên. Vì thế, để dễ đọc, chúng ta sẽ chọn cách làm tinh明 hơn trong mẫu thứ hai:

```
function processConfirm(answer) {  
    if (answer) {  
        return "Tuyệt. Chúng ta sẽ chơi một ván cờ h  
    } else {  
        result = "Có thể một lúc nữa.";  
    }  
}
```

Mã JavaScript bên trong phần *<body>* của đoạn mã tạo một hộp thoại xác nhận, gọi hàm *processConfirm()* và hiển thị kết quả.

```
var confirmAnswer = confirm("Chúng ta chơi một ván chử?");  
var theAnswer = processConfirm(confirmAnswer);  
alert(theAnswer);
```

Giống như hàm *alert()*, hàm *confirm()* nhận vào một đối số duy nhất, đó là

nội dung thông báo để hiển thị trong hộp thoại. Chỉ có một lưu ý là với hàm `confirm()`, bạn nên đặt nội dung thông báo theo dạng câu hỏi hoặc thông điệp cho phép người dùng lựa chọn. Nếu người dùng không đưa ra lựa chọn nào, hãy thay thế bằng hàm `alert()`. Cách viết ngắn gọn kết hợp cả ba dòng lệnh như sau:

```
alert(processConfirm(confirm("Chúng ta chơi một ván chứ?")))
```

Bài tập

1. Định nghĩa một hàm nhận vào một đối số dạng số, tăng giá trị đối số đó và trả lại giá trị cho lời gọi hàm. Gọi hàm từ trong phần `<body>` của trang web và hiển thị kết quả lên màn hình.
2. Định nghĩa hàm nhận vào hai tham số dạng số. Nếu giá trị của tham số thứ nhất lớn hơn tham số thứ hai, hiển thị hộp thoại thông báo cho người dùng. Nếu giá trị của tham số thứ nhất nhỏ hơn hoặc bằng tham số thứ hai, trả về tổng của hai tham số.
3. Thêm các hàm `alert()` vào những vị trí thích hợp trong đoạn mã sau để bạn có thể nhìn thấy giá trị của biến `result` trước và sau lời gọi hàm. Đoạn mã có dạng như sau:

```
function addNumbers() {  
    firstNum = 4;  
    secondNum = 8;  
    result = firstNum + secondNum;  
    return result;  
}  
result = 0;  
result = addNumbers();
```

4. Tạo một mảng với bảy phần tử dạng chuỗi có giá trị là tên các ngôi sao sau: Polaris, Aldebaran, Deneb, Vega, Altair, Dubhe và Regulus. Tạo một mảng khác với bảy phần tử dạng chuỗi có giá trị là tên các chòm sao tương ứng chứa các ngôi sao: Ursa Minor, Taurus, Cygnus, Lyra, Aquila, Ursa Major và Leo. Tiếp theo, tạo một hàm nhận vào một tham số dạng chuỗi duy nhất. Trong hàm này, duyệt qua mảng thứ nhất để tìm tên ngôi sao. Nếu tìm thấy,

trả lại giá trị tương ứng với chỉ số đó ở mảng thứ hai. Nói cách khác, trả lại tên chòm sao của ngôi sao đó. Trong phần `<body>` của trang web, dùng hộp thoại *prompt* để người dùng nhập vào tên của ngôi sao, sau đó gọi hàm với thông tin đầu vào. Đừng quên xử lý trong trường hợp không tìm thấy ngôi sao nào. Cho kết quả hiển thị trên màn hình.



Chương 8

Các đối tượng trong JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Hiểu về đối tượng trong JavaScript, bao gồm thuộc tính, phương thức và lớp.
- Tạo đối tượng.
- Định nghĩa thuộc tính và phương thức cho đối tượng.
- Hiểu về mảng trong JavaScript.
- Sử dụng một số phương thức của mảng.

Lập trình hướng đối tượng

Nếu bạn mới làm quen hoặc muốn bổ sung kiến thức về lập trình hướng đối tượng, hãy đọc tiếp phần này. Nếu bạn đã quen với lập trình hướng đối tượng, hãy bỏ qua đến phần “Tạo đối tượng”.

Phương thức lập trình mô tả phương pháp luận được sử dụng để giải quyết vấn đề. Có tới hơn 25 phương thức lập trình khác nhau đang tồn tại, một vài trong số đó khó có thể được tìm thấy trong chương trình thực tế nào. Có thể, bạn đã từng nghe nói đến một vài phương thức hoặc thậm chí đã áp dụng mà không hề hay biết. Có thể kể ra đây một số phương thức lập trình sau: lập trình hướng chức năng, lập trình hướng sự kiện, lập trình hướng thành phần và lập trình cấu trúc.

Những phương thức lập trình lần lượt xuất hiện rồi biến mất. Lập trình hướng đối tượng đã tồn tại nhiều năm, tuy nhiên phương thức này sẽ vẫn được áp dụng rộng rãi. Phần này chỉ cung cấp cho bạn cái nhìn toàn cảnh về chủ đề trên, tuy vậy bạn cần làm quen với các thuật ngữ và kỹ thuật lập trình hướng đối tượng để hiểu những phương pháp và thuật ngữ thường dùng của lập trình viên JavaScript.

Đối tượng

Đối tượng (Object) chính là vật thể. Trong thế giới thực, trái ngược với thế giới ảo và thế giới siêu thực của lập trình máy tính, quả bóng, cái bàn, xe ô tô đều là đối tượng. Đối tượng có những đặc điểm có thể mô tả, bạn có thể tương tác với nó và nó hoạt động theo một cách thức nhất định. Đối tượng trong phương thức lập trình hướng đối tượng là sự kết hợp đoạn mã và dữ liệu để biểu diễn các đặc điểm và phương thức theo cách thức tương tự.

Thuộc tính

Đối tượng có các *thuộc tính* (*property*) – xác định các đặc tính của nó. Ví dụ, trên thực tế, quả bóng có thuộc tính màu sắc – nó có thể đỏ, trắng hay nhiều màu. Nó cũng có thuộc tính về kích cỡ - nhỏ như quả bóng chày hoặc lớn như quả bóng rổ. Những thuộc tính này có thể biểu diễn như sau:

```
ball.color  
ball.size
```



Phương thức

Không chỉ có thuộc tính, đối tượng còn có cả phương thức. *Phương thức* (*method*) định nghĩa cách thức hoạt động của đối tượng. Một quả bóng có thể có phương thức lăn, phương thức này tính toán quả bóng sẽ lăn được bao xa. Về lý thuyết, không phải tất cả các đối tượng đều có phương thức hay thuộc tính, tuy vậy trên thực tế hầu hết các đối tượng đều có ít nhất một phương thức hay một thuộc tính.

Hãy nhớ lại Chương 7 “Làm việc với hàm”, phương thức chỉ là một hàm thuộc về đối tượng. Định nghĩa phương thức sử dụng hàm *literal* (function *literal*) cho phương thức *roll* có dạng như sau:

```
ball.roll= function(){//roll - lăn  
    var distance = this.size * this.forceApplied;  
    //distance - khoảng cách, forceApplied - lực tác dụng  
}
```

This là gì?

Trong ví dụ `ball.roll` có sử dụng từ khóa `this`, nhằm chỉ đến đối tượng chứa hàm hay thuộc tính hiện tại. Trong ngữ cảnh của đối tượng, từ khóa `this` là để gọi đến đối tượng. Từ khóa `this` có thể được sử dụng để thiết lập các thuộc tính của đối tượng trong lời gọi hàm.

Từ khóa `this` rất có ích cho các lập trình viên JavaScript khi kiểm tra tính hợp lệ của web form. Bạn sẽ tìm hiểu kỹ hơn về nội dung này trong Chương 14 “Sử dụng JavaScript với Web Form”.

Lớp

Trong lập trình hướng đối tượng, *lớp (class)* định nghĩa một tập các đối tượng có chung thuộc tính và phương thức. Lớp đơn giản hóa việc tạo nhiều đối tượng cùng kiểu. Tuy vậy, ECMA-262 không có khái niệm về lớp trong giao diện đối tượng. Để tối ưu hóa lợi ích của việc lập trình dựa trên lớp, bạn cần phải sử dụng các mẫu để tạo lớp giả.

Hãy xem lại ví dụ về các chòm sao mà tôi đã trình bày ở các chương trước. Ví dụ 8-1 (file Listing8-1.txt trong Tài nguyên đi kèm) cho thấy những gì bạn cần để có một trang web bao gồm thông tin về 14 ngôi sao chính.

VÍ DỤ 8-1 Lắp ghép đối tượng *star*.

```
var star= {};
star["Polaris"]= newObject;
star["Mizar"]= newObject;
star["Aldebaran"]= newObject;
star["Rigel1"]= newObject;
star["Castor"]= newObject;
star["Albireo"]= newObject;
star["Acrux"]= newObject;
star["Gemma"]= newObject;
star["Procyon"]= newObject;
star["Sirius"]= newObject;
star["Rigel Kentaurus"]= newObject;
star["Deneb"]= newObject;
```

```
star["Vega"] = newObject;
star["Altair"] = newObject;
star["Polaris"].constellation= "Ursa Minor";
star["Mizar"].constellation= "Ursa Major";
star["Aldebaran"].constellation= "Taurus";
star["Rigel"].constellation= "Orion";
star["Castor"].constellation= "Gemini";
star["Albireo"].constellation= "Cygnus";
star["Acrux"].constellation= "Crux";
star["Gemma"].constellation= "Corona Borealis";
star["Procyon"].constellation= "Canis Minor";
star["Sirius"].constellation= "Canis Major";
star["Rigil Kentaurus"].constellation= "Centaurus";
star["Deneb"].constellation= "Cygnus";
star["Vega"].constellation= "Lyra";
star["Altair"].constellation= "Aquila";
star["Polaris"].type= "Double/Cepheid";
star["Mizar"].type= "Spectroscopic Binary";
star["Aldebaran"].type= "Irregular Variable";
star["Rigel"].type= "Supergiant with Companion";
star["Castor"].type= "Multiple/Spectroscopic";
star["Albireo"].type= "Double";
star["Acrux"].type= "Double";
star["Gemma"].type= "Eclipsing Binary";
star["Procyon"].type= "Double";
star["Sirius"].type= "Double";
star["Rigil Kentaurus"].type= "Double";
star["Deneb"].type= "Supergiant";
star["Vega"].type= "White Dwarf";
star["Altair"].type= "White Dwarf";
star["Polaris"].spectralClass= "F7";
star["Mizar"].spectralClass= "A1 V";
star["Aldebaran"].spectralClass= "K5 III";
star["Rigel"].spectralClass= "B8 Ia";
star["Castor"].spectralClass= "A1 V";
star["Albireo"].spectralClass= "K3 II";
star["Acrux"].spectralClass= "B1 IV";
star["Gemma"].spectralClass= "A0 V";
star["Procyon"].spectralClass= "F5 IV";
star["Sirius"].spectralClass= "A1 V";
star["Rigil Kentaurus"].spectralClass= "G2 V";
star["Deneb"].spectralClass= "A2 Ia";
```

```

star["Vega"].spectralClass= "A0 V";
star["Altair"].spectralClass= "A7 V";
star["Polaris"].mag= 2.0;
star["Mizar"].mag= 2.3;
star["Aldebaran"].mag= 0.85;
star["Rigel"].mag= 0.12;
star["Castor"].mag= 1.58;
star["Albireo"].mag= 3.1;
star["Acrux"].mag= 0.8;
star["Gemma"].mag= 2.23;
star["Procyon"].mag= 0.38;
star["Sirius"].mag= -1.46;
star["Rigel Kentaurus"].mag= -0.01;
star["Deneb"].mag= 1.25;
star["Vega"].mag= 0.03;
star["Altair"].mag= 0.77;

```

Như bạn đã thấy, Danh sách 8-1 chứa rất nhiều đoạn mã lặp đi lặp lại. Mỗi ngôi sao được định nghĩa và khai báo bốn thuộc tính: chòm sao (constellation), loại sao (type), lớp quang phổ (spectral class) và độ lớn (*mag*).

Tiếp theo, hãy xem Ví dụ 8-2 (nằm trong file Listing8-2.txt của Tài nguyên đi kèm). Kết quả giống như đoạn mã trong Ví dụ 8-1 với sự trợ giúp của mẫu hàm khởi tạo để tạo lớp giả.

VÍ DỤ 8-2 Lắp ghép đối tượng *star* sử dụng lớp giả.

```

var star= {};
function Star(constell,type,specclass,magnitude) {
    this.constellation= constell;
    this.type= type;
    this.spectralClass= specclass;
    this.mag = magnitude;
}
star["Polaris"]= new Star("Ursa Minor", "Double/Cepheid", "F");
star["Mizar"]= new Star("Ursa Major", "Spectroscopic Binary");
star["Aldebaran"]= new Star("Taurus", "Irregular Variable");
star["Rigel"]= new Star("Orion", "Supergiant with Companion");
star["Castor"]= new Star("Gemini", "Multiple/Spectroscopic");
star["Albireo"]= new Star("Cygnus", "Double", "K3 II", 3.1);
star["Acrux"]= new Star("Crux", "Double", "B1 IV", 0.8);
star["Gemma"]= new Star("Corona Borealis", "Eclipsing Binari

```

```
star["Procyon"] = new Star("Canis Minor", "Double", "F5 IV", 0  
star["Sirius"] = new Star("Canis Major", "Double", "A1 V", -1.  
star["Rigel Kentaurus"] = new Star("Centaurus", "Double", "G2  
star["Deneb"] = new Star("Cygnus", "Supergiant", "A2 Ia", 1.25  
star["Vega"] = new Star("Lyra", "White Dwarf", "A0 V", 0.03);  
star["Altair"] = new Star("Aquila", "White Dwarf", "A7 V", 0.7
```

Hàm *Star* được in đậm trong Ví dụ 8-2 tạo một giao diện giúp khởi tạo đối tượng *Star* nhanh chóng.

Khi được gọi, hàm này trả về một đối tượng *Star* mới:

```
star["Polaris"] = new Star("Ursa Minor", "Double/Cepheid", "F7"
```

Mặc dù Ví dụ 8-1 và 8-2 giống nhau về chức năng, nhưng đoạn mã trong Ví dụ 8-2 ngắn gọn và dễ hiểu hơn. Hãy tưởng tượng một đối tượng có chín thay vì bốn thuộc tính như trình bày ở đây.

Việc tạo giao diện giống như lớp trong phần này sử dụng mẫu hàm khởi tạo. Mẫu hàm khởi tạo khá hữu dụng ngoại trừ việc chúng tạo ra nhiều thực thể của cùng một phương thức. Cách tối ưu và cao cấp hơn khi tạo nhiều đối tượng là sử dụng nguyên mẫu prototype. Xem thêm <http://msdn.microsoft.com/en-us/magazine/cc163419.aspx>. về cách tạo đối tượng sử dụng prototype.

Tạo đối tượng

Có hai cách để tạo đối tượng trong JavaScript:

- Sử dụng từ khóa:

```
var star = new Object;
```

- Hoặc sử dụng dấu ngoặc nhọn:

```
var star = {};
```

Cả hai cách đều cho những kết quả giống nhau. Vì vậy, việc sử dụng cách thức nào phụ thuộc vào sở thích cá nhân của bạn.

Thêm thuộc tính cho đối tượng

Sau khi tạo xong đối tượng, bạn cần gán thuộc tính và phương thức cho đối tượng. Nếu như chỉ có một đối tượng *star*, bạn có thể trực tiếp gán thuộc tính cho nó:

```
star.name= "Polaris";
star.constellation= "Ursa Minor";
```

Khi cần tạo nhiều đối tượng liên quan đến nhau, bạn có thể gán thuộc tính một cách hiệu quả bằng cách làm như ví dụ ở phần trước.

Hiển thị thuộc tính của đối tượng

Với vòng lặp *for...in*, bạn có thể duyệt từng thuộc tính của đối tượng.

Duyệt các thuộc tính của đối tượng

1. Sử dụng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác sửa file proploop.htm trong thư mục mã nguồn mẫu của Chương 8, phần Tài nguyên đi kèm.
2. Sử dụng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác sửa file proploop.htm trong thư mục mã nguồn mẫu của Chương 8, phần Tài nguyên đi kèm.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Thuộc tính</title>
<script type="text/javascript">
    var star= {};
    function Star(constell,type,specclass,magnitude) {
        this.constellation = constell;//chòm sao
        this.type = type; //kiểu
        this.spectralClass = specclass; //lớp quang
        this.mag = magnitude; //độ lớn
    }

```

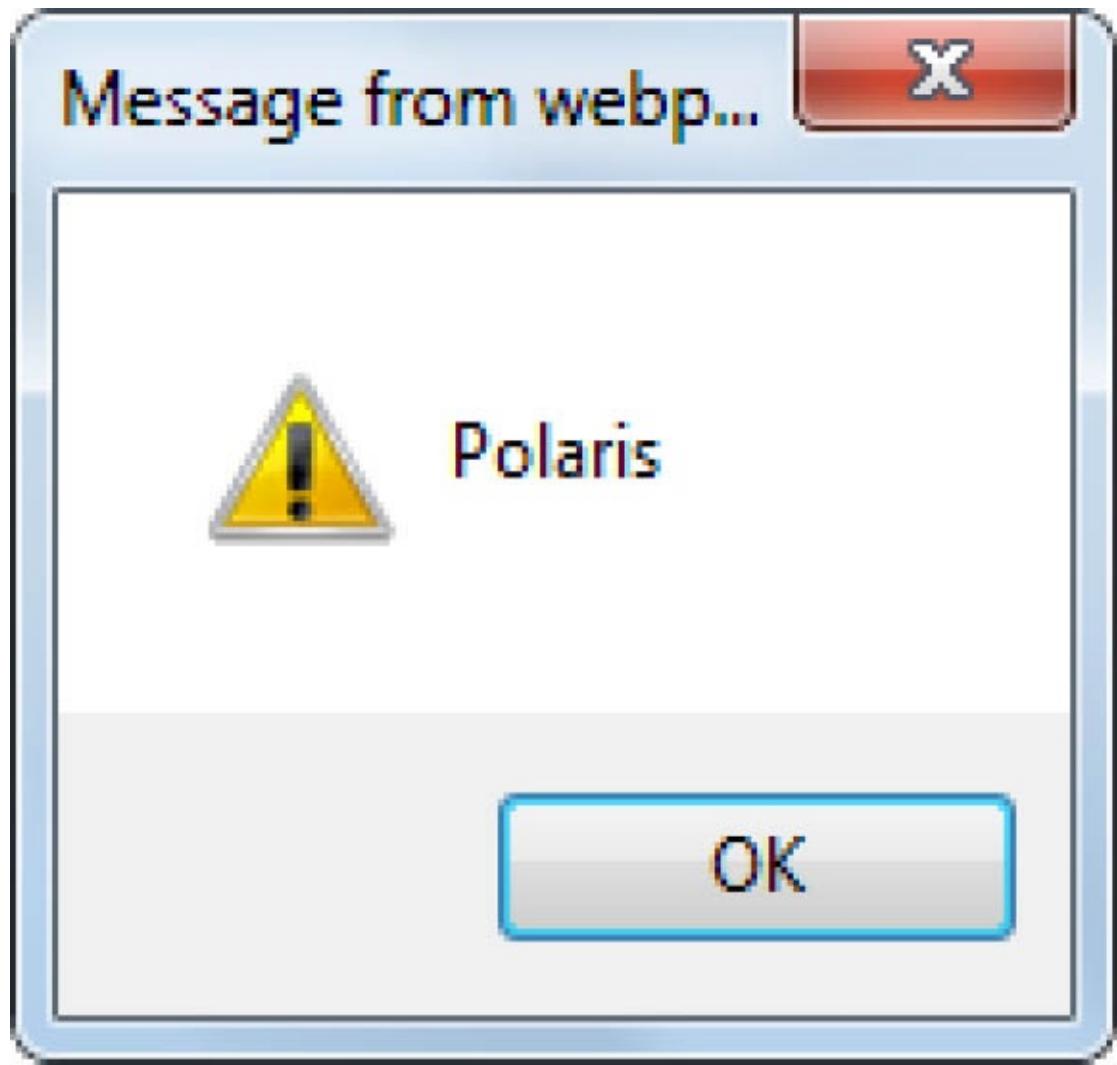
```

star["Polaris"]= new Star("Ursa Minor","Double Cepheid","F0 Ia");
star["Mizar"]= new Star("Ursa Major","Spectroscopic Binary");
star["Aldebaran"]= new Star("Taurus","Irregular Variable");
star["Rigel"]= new Star("Orion","Supergiant with Companion");
star["Castor"]= new Star("Gemini","Multiple/Spectroscopic");
star["Albireo"]= new Star("Cygnus","Double","K3 II",3.1);
star["Acrux"]= new Star("Crux","Double","B1 IV",0.8);
star["Gemma"]= new Star("Corona Borealis","Eclipsing Binary");
star["Procyon"]= new Star("Canis Minor","Double","F5 IV",0.12);
star["Sirius"]= new Star("Canis Major","Double","A1 V",-1.46);
star["Rigel Kentaurus"]= new Star("Centaurus","Double","G2 V",0.94);
star["Deneb"]= new Star("Cygnus","Supergiant","A2 Ia",1.25);
star["Vega"]= new Star("Lyra","White Dwarf","A0 V",0.03);
star["Altair"]= new Star("Aquila","White Dwarf","A7 V",0.77);
</script>
</head>
<body>
<script type="text/javascript">
for(var propt in star){
  alert(propt);
}
</script>
</body>
</html>

```



3. Dùng trình duyệt mở trang web. Hộp thoại thông báo cho 14 ngôi sao sẽ lần lượt hiện ra. (Bạn sẽ phải nhấn OK khá nhiều). Đây là ví dụ về hộp thoại mà bạn sẽ nhìn thấy:



Bài tập hướng dẫn từng bước này dựa trên ví dụ trước đó về cách sử dụng lớp giả để định nghĩa thuộc tính của đối tượng. Trong trường hợp này, đối tượng *star* được tạo bởi dòng mã:

```
var star= {};
```

Đối tượng gán từng thuộc tính riêng cho ngôi sao bằng cách sử dụng lời gọi tạo một đối tượng *Star* mới (sử dụng lớp giả):

```
star["Polaris"]= new Star("Ursa Minor","Double/Cepheid","F7"
```

Từng thuộc tính của đối tượng gốc *star*, trong trường hợp này là tên của từng ngôi sao, được liệt kê trong phần *<body>* bởi đoạn mã sử dụng vòng lặp *for...in*.

```
for(var prop in star){  
    alert(prop);  
}
```

Có thể bạn sẽ băn khoăn làm thế nào để biết thuộc tính thật của các ngôi sao, như chòm sao, độ lớn, loại và lớp quang phổ. Chương 14 sẽ hướng dẫn cách liệt kê từng thuộc tính trên.

Tìm kiếm thuộc tính

Đôi khi, bạn không muốn hoặc không cần duyệt qua từng thuộc tính. Đôi khi, bạn chỉ muốn biết một thuộc tính có tồn tại trong đối tượng không. Bạn có thể sử dụng toán tử `in` để kiểm tra thuộc tính như đoạn mã giả sau:

```
if(property in object){ // property và object là tên thuộc tính  
// viết mã ở đây  
}
```

Ví dụ 8-3 phức tạp hơn (ví dụ này có trong file Listing8-3.txt của Tài nguyên đi kèm) kiểm tra đối tượng `star` để tìm tên ngôi sao “Polaris” và nếu tìm thấy, thì thêm vào nó một thuộc tính mới.

Ví dụ 8-3 Tìm kiếm thuộc tính.



```
var star= {};  
function Star(constell,type,specclass,magnitude) {  
    this.constellation = constell;  
    this.type = type;  
    this.spectralClass = specclass;  
    this.mag = magnitude;  
}  
star["Polaris"]= new Star("Ursa Minor","Double/Cepheid","F0 V");  
star["Mizar"]= new Star("Ursa Major","Spectroscopic Binary", "A1 V", 1.8);  
star["Aldebaran"]= new Star("Taurus","Irregular Variable", "K1 Ia", 0.9);  
star["Rigel"]= new Star("Orion","Supergiant with Companion", "B1 Ia", -4.2);  
star["Castor"]= new Star("Gemini","Multiple/Spectroscopic", "A1 V", 0.6);  
star["Albireo"]= new Star("Cygnus", "Double", "K3 II", 3.1);  
star["Acrux"]= new Star("Crux", "Double", "B1 IV", 0.8);  
star["Gemma"]= new Star("Corona Borealis", "Eclipsing Binary", "A1 V", 0.6);  
star["Procyon"]= new Star("Canis Minor", "Double", "F5 IV", 0.3);  
star["Sirius"]= new Star("Canis Major", "Double", "A1 V", -1.4);  
star["Rigil Kentaurus"]= new Star("Centaurus", "Double", "G2 V", 0.8);
```

```
star["Deneb"] = new Star("Cygnus", "Supergiant", "A2 Ia", 1.25  
star["Vega"] = new Star("Lyra", "White Dwarf", "A0 V", 0.03);  
star["Altair"] = new Star("Aquila", "White Dwarf", "A7 V", 0.7  
if ("Polaris" in star) {  
    star["Polaris"].aka = "The North Star";  
    alert("Sao Polaris đã được tạo, sao này còn có tên  
    ")
```

Chú ý Có một vài cách khác để kiểm tra sự tồn tại của thuộc tính nhưng chúng không được đề cập đến ở đây, ví dụ như việc sử dụng toán tử `!==`.

Thêm phương thức vào đối tượng

Bạn cũng có thể thêm các phương thức cho đối tượng tự định nghĩa bằng cách tương tự như cách thêm thuộc tính. Giả sử bạn muốn mở rộng lớp `Star` ở ví dụ trước đó để có thêm phương thức `show()`, phương thức gọi ra hộp thoại `alert()`. Bạn có thể mở rộng phương thức này tùy ý. Ví dụ, hãy xem đoạn mã bên dưới (xem trong file adding-methods.txt của Tài nguyên đi kèm):

```
function Star(constell, type, specclass, magnitude) {  
    this.constellation = constell;  
    this.type = type;  
    this.spectralClass = specclass;  
    this.mag = magnitude;  
    this.show = function(){  
        alert("xin chào, đây là phương thức.");  
    }  
}
```

Để gọi đến phương thức, bạn dùng dòng mã như sau:
`star["Polaris"].show();`

Lập trình hướng đối tượng trong JavaScript không chỉ dừng lại ở đây. Những tính năng chuyên sâu của phương thức lập trình hướng đối tượng như tính kế thừa, siêu lớp và prototype (nguyên mẫu) đều có trong JavaScript, tuy vậy chúng không nằm trong phạm vi của cuốn sách. Tạp chí MSDN có đăng một bài báo về những khái niệm chuyên sâu, bạn có thể tìm đọc tại địa chỉ <http://msdn.microsoft.com/en-us/magazine/cc163419.aspx>.

Tìm hiểu thêm về mảng

Chương 4 đã giới thiệu về mảng và đưa ra một số ví dụ để định nghĩa mảng. Sử dụng mảng, bạn có thể nhóm một tập các giá trị vào một đối tượng và truy cập đến những giá trị này thông qua giá trị của chỉ số (index). Ví dụ, bạn có thể sử dụng hàm khởi tạo như sau (xem file morearrays.txt của Tài nguyên đi kèm):

```
var star= new Array();
star[0]= "Polaris";
star[1]= "Deneb";
star[2]= "Vega";
star[3]= "Altair";
```

Bạn cũng có thể sử dụng hàm tạo mảng ngầm định (biểu thị bởi cặp ngoặc vuông) để thực hiện tác vụ trên như sau:

```
var star= ["Polaris", "Deneb", "Vega", "Altair"];
```



Thuộc tính length

Thuộc tính *length* của một mảng trả về số lượng các phần tử trong mảng. Có sự khác biệt quan trọng giữa số lượng phần tử mà mảng đang chứa và số lượng đã được định nghĩa. Đây là một ví dụ đơn giản. Hãy xem xét cách định nghĩa mảng *star* mà chúng ta đã đề cập trước đó. Có bốn ngôi sao: Polaris, Deneb, Vega và Altair. Thuộc tính *length* trả về cùng kết quả:

```
var numStars= star.length; // star.length bằng 4.
```

Các phần tử được đếm bởi thuộc tính *Length* chưa được định nghĩa hay khởi tạo. Đây là ví dụ về việc tạo mảng chứa nhiều phần tử hơn nó được gán:

```
var myArray= new Array(5);
```

Phương thức của mảng

Phần này giới thiệu bạn làm quen với một số phương thức của đối tượng *array*. Bạn có thể tìm hiểu thêm thông tin về các phương thức này trong bản đặc tả

ECMA-262 tại <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.

Thêm và bớt phần tử

Bạn có thể thêm phần tử vào đầu hoặc cuối mảng bằng nhiều cách thức khác nhau.

Sử dụng concat() để thêm phần tử. Phương thức *concat()* giúp thêm phần tử vào cuối mảng. Để sử dụng, gọi hàm *concat()* với đối số chứa phần tử muốn thêm vào. Hàm này trả về mảng mới như sau (xem file morearray.txt của Tài nguyên đi kèm):

```
var myArray= new Array();
myArray[0]= "first";
myArray[1]= "second";
var newArray= myArray.concat("third");
// Mảng newArray bây giờ có các phần tử là[first,second,third]
```

Bạn có thể kết hợp hai mảng vào làm một như sau:

```
var myFirstArray= [51, 67];
var mySecondArray= [18, "hello", 125];
var newArray= myFirstArray.concat(mySecondArray)
// Mảng newArray bây giờ có các phần tử là[51, 67, 18, "hello", 125]
```

Thêm phần tử với concat()

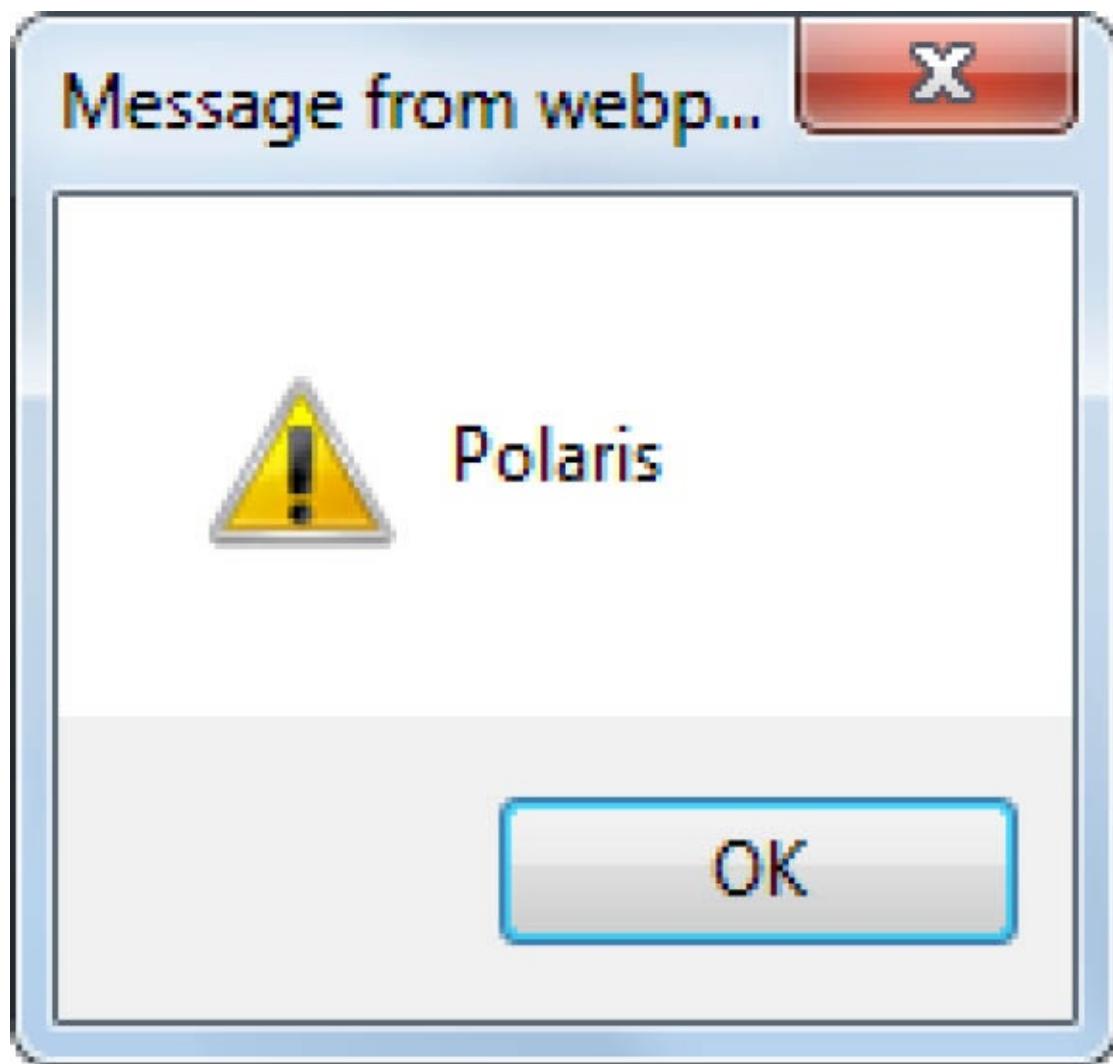
1. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo khác, sửa file concat.htm trong thư mục mã nguồn mẫu Chương 8 phần Tài nguyên đi kèm.
2. Trong trang này, thêm vào đoạn mã được in đậm (xem phần đầu của file concat.txt).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Phương thức Concat</title>
    <script type="text/javascript">
        var star = ["Polaris", "Deneb", "Vega", "Altair"];
    </script>
</head>
<body>
    <h1>Mảng Star</h1>
    <p>Mảng ban đầu: <code>star</code></p>
    <pre>[
        <b>"Polaris">
        <b>"Deneb">
        <b>"Vega">
        <b>"Altair">
    ]</pre>
    <hr>
    <h2>Sau khi sử dụng concat()</h2>
    <pre>[
        <b>"Polaris">
        <b>"Deneb">
        <b>"Vega">
        <b>"Altair">
        <b>"Sirius">
        <b>"Antares">
        <b>"Betelgeuse">
        <b>"Spica">
    ]</pre>
</body>

```

```
        for (var i= 0; i<star.length; i++) {  
            alert(star[i]);  
        }  
    </script>  
</head>  
<body>  
</body>  
</html>
```

3. Lưu trang web, sau đó dùng trình duyệt mở lại. Bạn nhận được hộp thoại thông báo `alert()` cho tên của bốn ngôi sao đã định nghĩa trong mảng ngôi sao.



4. Sửa đoạn mã để nối thêm một vài ngôi sao khác vào mảng (Bạn có thể thêm

trực tiếp vào mảng, nhưng như vậy là không trung thực). Sau đây là đoạn mã hoàn chỉnh (những thay đổi so với file concat.txt ở trên đã được in đậm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Phương thức Concat</title>  
    <script type="text/javascript">  
        var star = ["Polaris", "Deneb", "Vega", "Altair"];  
        var newstars=  ["Aldebaran", "Rigel"];  
        var morestars= star.concat(newstars);  
        var mStarLength= morestars.length;  
        for (var i= 0; i<mStarLength; i++) {  
            alert(morestars[i]);  
        }  
    </script>  
</head>  
<body>  
</body>  
</html>
```



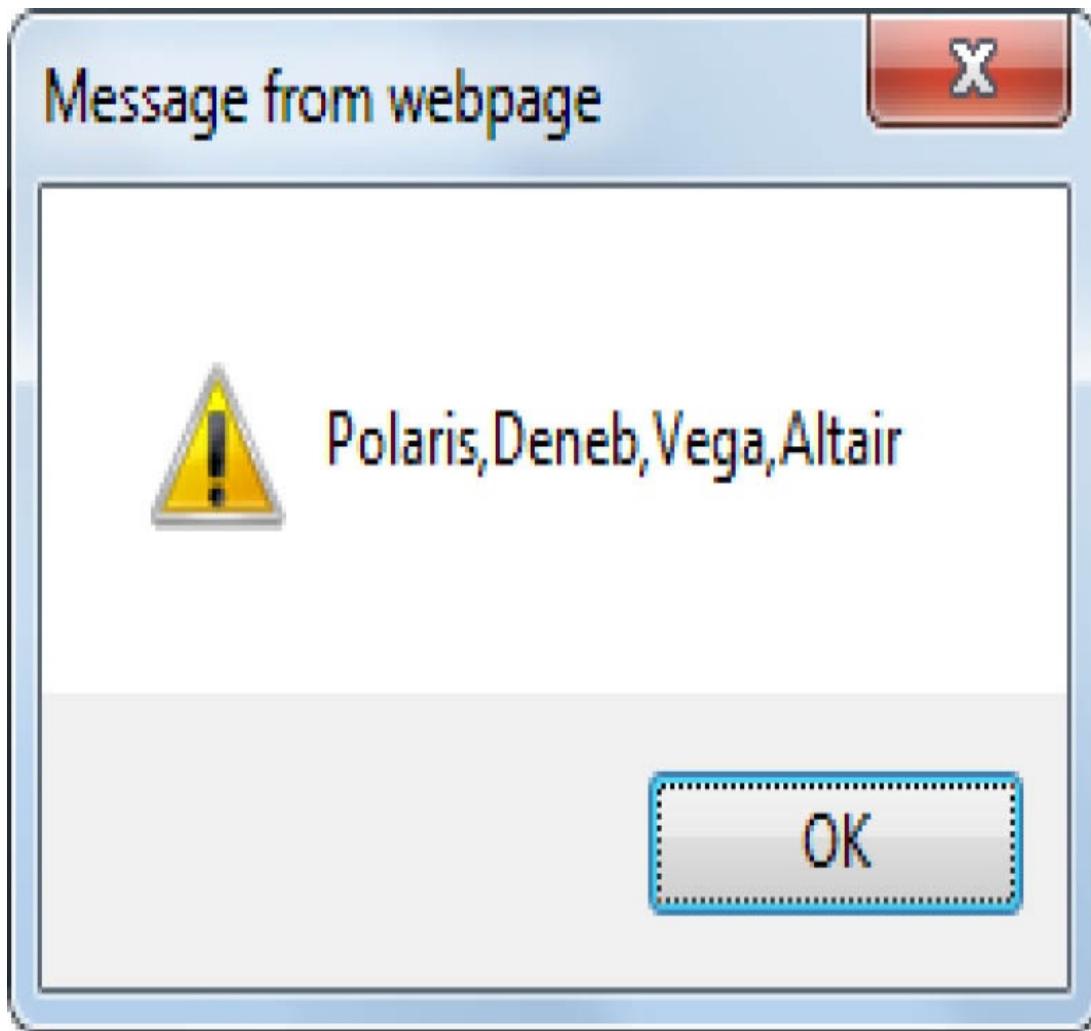
5. Lưu file và dùng trình duyệt mở lại. Lúc này, bạn nhận được sáu hộp thoại `alert()` cho mỗi ngôi sao, ví dụ dưới đây là hộp thoại cho sao Aldebaran:



Kết nối các phần tử trong mảng với phương thức *join()*. Phương thức *join()* chuyển đổi tất cả các phần tử của mảng thành một chuỗi. Phương thức này khác với *concat()* vì phương thức *concat()* chỉ thực hiện chèn thêm vào mảng nhưng không thực hiện chuyển đổi kiểu dữ liệu. Sau đây là đoạn mã:

```
var star= ["Polaris", "Deneb", "Vega", "Altair"];  
var starString= star.join();
```

Biến *starString* chứa Polaris, Deneb, Vega, Altair như Hình 8-1.

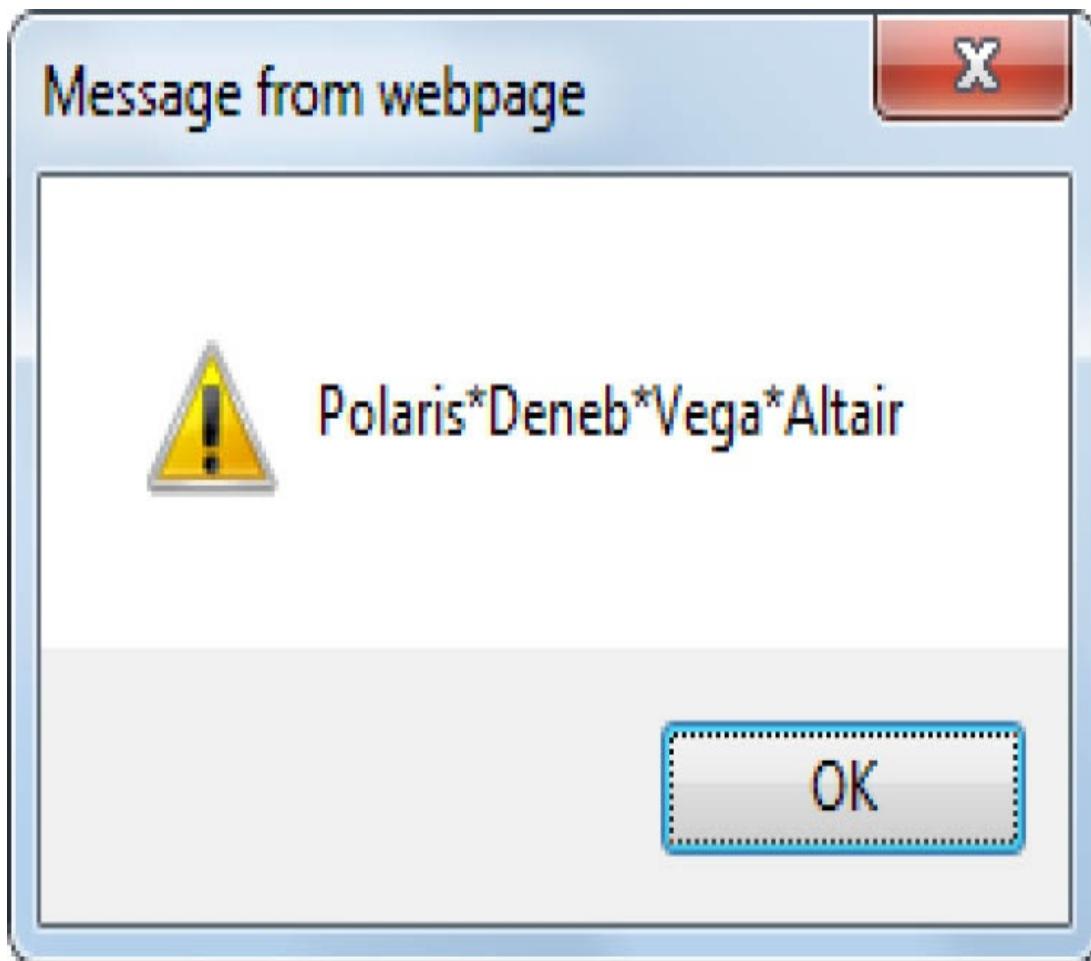


HÌNH 8-1 Sử dụng hàm join() để kết nối các phần tử trong một mảng.

Phương thức này cho phép chỉ định dấu phân tách giữa các phần tử khi thực hiện kết nối. Thay vì sử dụng dấu phẩy, bạn có thể muốn dùng dấu sao như sau:

```
var star= ["Polaris", "Deneb", "Vega", "Altair"];
var starString= star.join("*");
```

Khi đó, kết quả trả về là Polaris*Deneb*Vega*Altair như Hình 8-2.



HINH 8-2 Kết nối sử dụng dấu * làm dấu phân tách.

Mách nhỏ Phương thức *join()* là phương thức xem nhanh nội dung của mảng thay vì tạo cấu trúc vòng lặp *for*.

Sử dụng *push* và *pop* để thêm và bớt phần tử**. Trong khi phương thức *concat()* trả về mảng mới, thì *push()* và *pop()* thêm và bớt phần tử trực tiếp. Phương thức *push()* trả về độ dài của mảng mới còn *pop()* trả về phần tử bị xóa. Phương thức *push()* và *pop()* thực thi tại phần cuối của mảng, như mô tả dưới đây (bạn có thể xem ví dụ này trong file evenmorearrays.txt của Tài nguyên đi kèm):

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
star.push("Aldebaran");
```

Sau khi chạy, đoạn mã trên trả về đối tượng *star* có năm phần tử: Polaris,

Deneb, Vega, Altair và Aldebaran.

Phương thức `pop()` xóa đi phần tử cuối của mảng và trả về phần tử này.

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
var removedElement = star.pop();
```

Biến `removedElement` lúc này chứa chuỗi “Altair” – phần tử cuối của mảng. Độ dài của mảng vì thế cũng bị giảm đi 1.

Sử dụng `shift` và `unshift` để thêm bớt các phần tử**. Phương thức `push()` và `pop()` thực thi ở cuối của mảng. Phương thức `shift()` và `unshift()` hoạt động tương tự như `push()` và `pop()` nhưng thực thi ở đầu mảng. Trong đoạn mã dưới đây, phương thức `unshift()` thêm một phần tử vào đầu mảng:

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
star.unshift("Aldebaran");
```

Lúc này mảng `star` gồm:

```
["Aldebaran", "Polaris", "Deneb", "Vega", "Altair"]
```

Sử dụng phương thức `shift()` để xóa phần tử đầu tiên của mảng. Chú ý là giống như `pop()`, `shift()` trả về phần tử bị xóa:

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
var removedElement = star.shift();
```

Lúc này mảng `star` gồm:

```
["Deneb", "Vega", "Altair"]
```

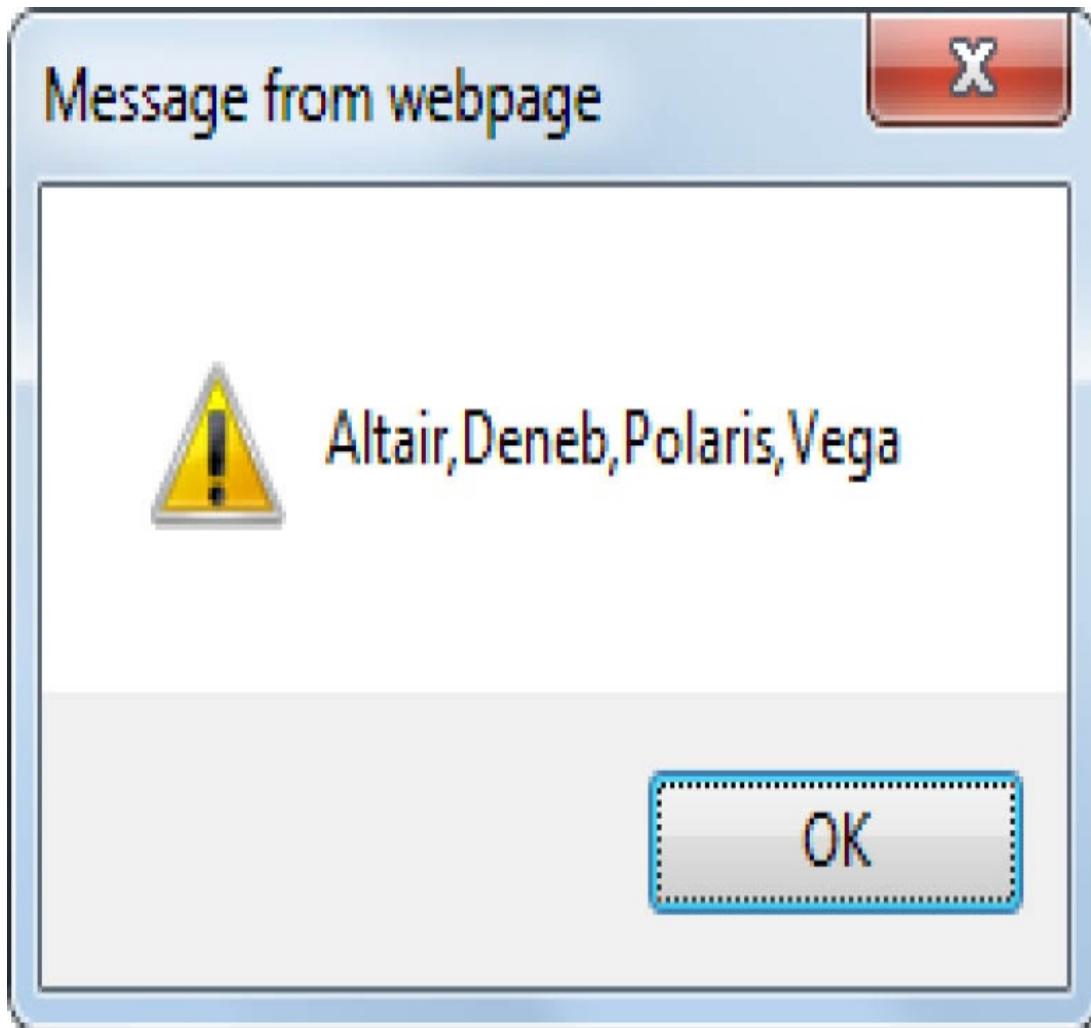
Sử dụng `slice` để trả về một phần của mảng**. Phương thức `slice()` rất hữu ích khi bạn cần trả về một phần của mảng, nhưng bạn nên cẩn thận vì hàm này sẽ thay đổi luôn mảng gốc. Ví dụ, đoạn mã sau trả về và gán vào biến `cutStars` giá trị “`Vega, Altair`” vì `Vega` và `Altair` lần lượt là phần tử thứ ba và bốn của mảng (nhớ rằng mảng bắt đầu từ 0).

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
var cutStars = star.slice(2, 3);
```

Sắp xếp phần tử với `sort`. Đôi khi bạn cần sắp xếp các phần tử trong mảng. Xem đoạn mã sau:

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
var sortedStars = star.sort();
```

Kết quả trả về như Hình 8-3. Như bạn thấy, các phần tử được sắp xếp theo thứ tự bảng chữ cái mặc dù khi viết mã chúng ta không sắp xếp chúng theo thứ tự. Chú ý: Cả mảng *star* gốc và biến *sortedStars* đều chứa danh sách đã được sắp xếp.

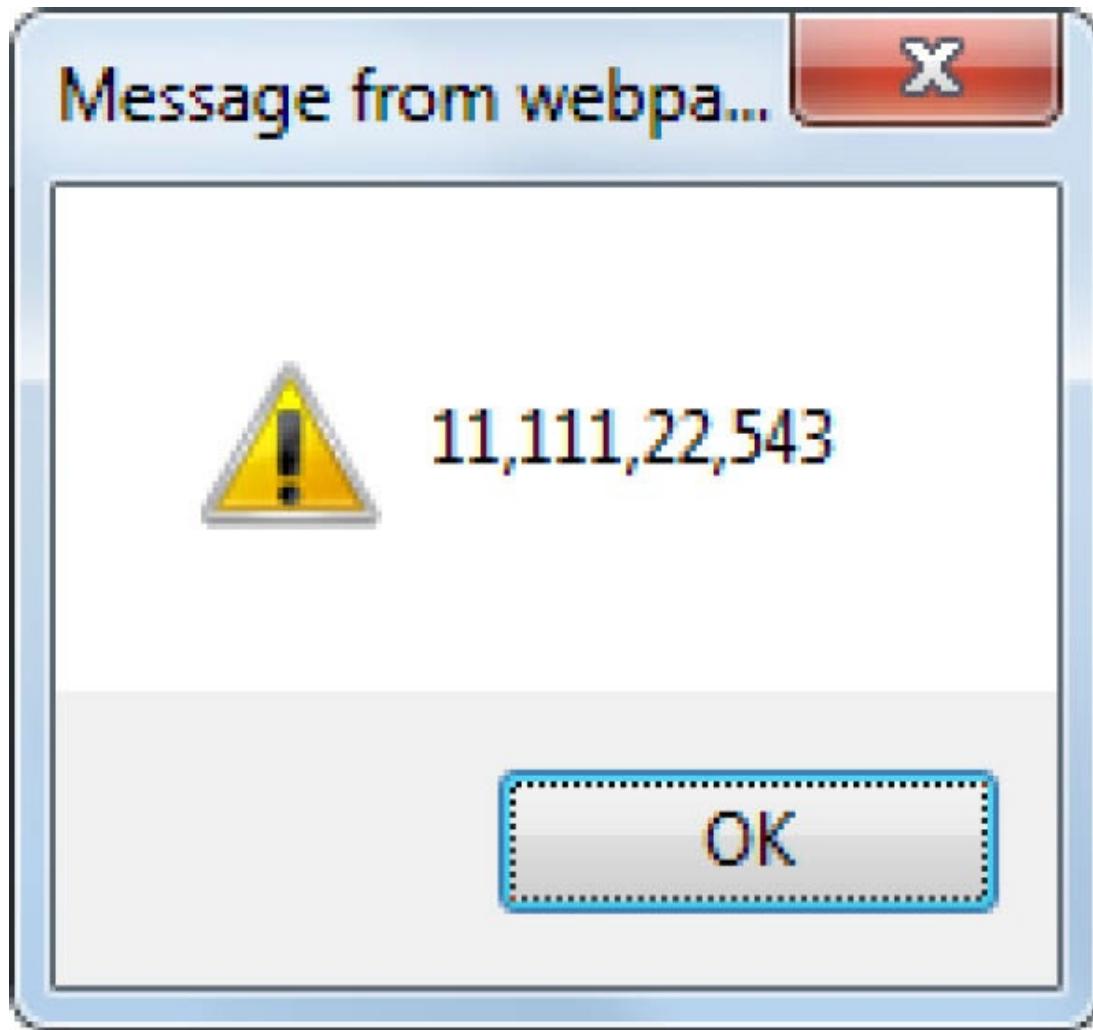


HINH 8-3 Kết quả của việc sắp xếp mảng sử dụng phương thức sort()

Hãy cẩn thận khi sử dụng phương thức *sort()* để sắp xếp các con số. Xem xét đoạn mã sau:

```
var nums = [11, 543, 22, 111];
var sortedNums = nums.sort();
```

Bạn mong muốn biến *sortedNums* trả về 11,22,111,543 nhưng thực tế phương thức *sort()* vẫn sắp xếp các giá trị theo thứ tự bảng chữ cái như trong Hình 8-4.



HÌNH 8-4 Việc sử dụng phương thức *sort()* để sắp xếp số sẽ hoạt động không như mong muốn.

Duyệt toàn bộ mảng. Trong JavaScript, có hai cách để duyệt qua các phần tử của mảng. Tại thời điểm viết cuốn sách này, phương thức chủ yếu có thể hoạt động trên nhiều trình duyệt là dùng vòng lặp *for()*. Bạn có thể ôn lại cú pháp của vòng lặp *for()* qua ví dụ sau:

```
var candies = ["chocolate", "licorice", "mints"];
for (var i= 0; i<candies.length; i++) {
    alert(candies[i]);
}
```

Được giới thiệu trong ECMA-262 phiên bản 5 và được hỗ trợ trên tất cả các loại trình duyệt ngoại trừ Windows Internet Explorer 8 và các phiên bản trước đó, phương thức *forEach()* cũng duyệt qua từng phần tử của mảng. Cấu trúc của *forEach()* tương tự như *for()*:

```
var candies= ["chocolate", "licorice", "mints"];
candies.forEach(function(candy){
alert(candy);
});
```

Chú ý Cần chú ý khi sử dụng phương thức *forEach()* (và một số phương thức mới khác) vì chúng có thể chưa được hỗ trợ rộng rãi.

Bạn cũng nên tìm hiểu thêm một số phương thức của đối tượng *array*. Một vài phương thức được liệt kê trong Bảng 8-1, nhưng bạn nên tham khảo chuẩn ECMA-262 tại địa chỉ <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf> để xem danh sách đầy đủ. Những phương thức mới xuất hiện trong ECMA phiên bản 5 đã được đánh dấu.

BẢNG 8-1 Lựa chọn các phương thức của đối tượng Array

| Phương thức | Mô tả | Mới có trong ECMA-262 phiên bản 5 |
|----------------------|--|---|
| <i>reverse()</i> | Đảo ngược thứ tự các phần tử. | Không |
| <i>map()</i> | Thực thi một hàm với từng phần tử trong mảng và trả về một mảng. | Có |
| <i>IndexOf()</i> | Trả về chỉ số của phần tử đầu tiên của mảng có giá trị bằng giá trị của đối số. | Có |
| <i>lastIndexOf()</i> | Trả về chỉ số cuối của đối số trong mảng. | Có |
| <i>every()</i> | Thực thi một hàm với từng phần tử trong mảng trong khi hàm vẫn trả về <i>true</i> . | Có |
| <i>filter()</i> | Thực thi một hàm với từng phần tử trong mảng và trả về một mảng chỉ chứa những phần tử mà hàm trả về giá trị <i>true</i> . | Có |
| <i>some()</i> | Thực thi một hàm với từng phần tử trong mảng trong khi hàm vẫn trả về <i>false</i> . | Có |
| | Chèn hay xóa phần tử trong mảng. Trả về mảng chứa | |

splice()

những phần tử bị xóa.

Không

Lợi thế của đối tượng có sẵn

Ngôn ngữ JavaScript tạo ra một số đối tượng hữu ích có sẵn để phục vụ cho những công việc chung trong chương trình JavaScript. Trong Chương 4, bạn đã làm quen với một số đối tượng như *Date*, *Number* và *Math*.

Đối tượng toàn cục

JavaScript có một đối tượng toàn cục chứa một số phương thức mà chúng ta đã bàn đến, ví dụ *isNaN()*. Ba phương thức thường sử dụng đối tượng toàn cục là *encodeURI()*, *encodeURIComponent()*, và *eval()* sẽ được đề cập sau đây.

Giúp URI an toàn

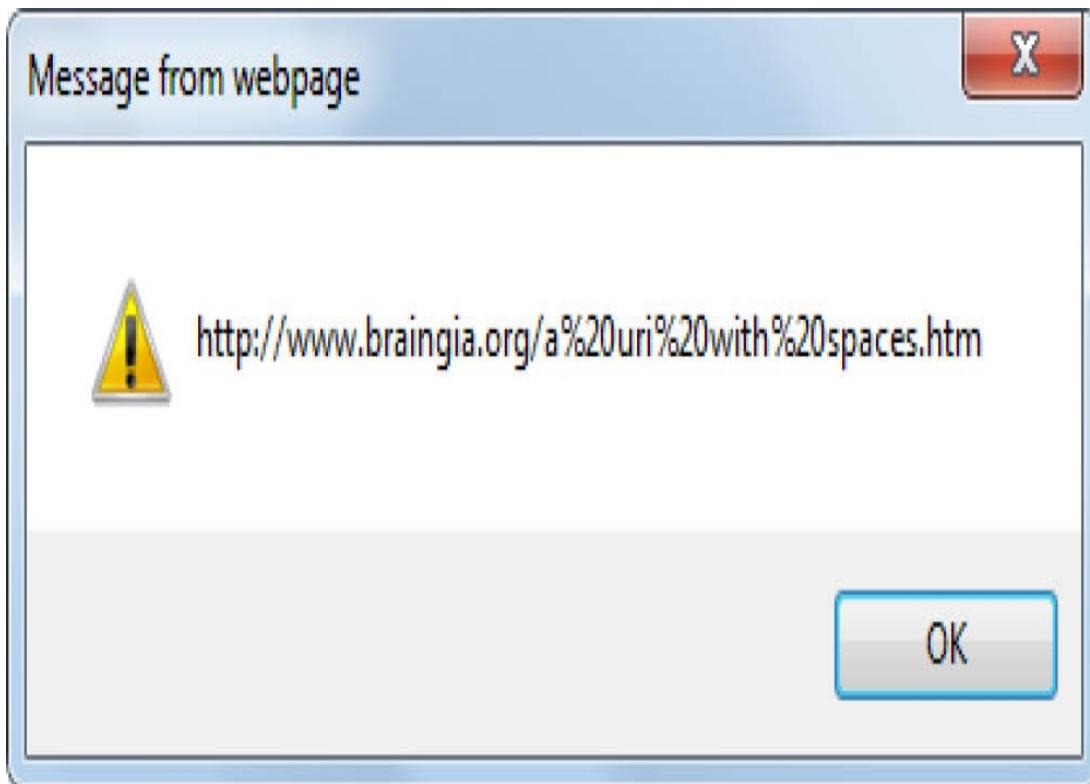
Phương thức *encodeURI()* lấy một URI (Uniform Resource Identifier) có chứa các ký tự không cho phép trong mẫu URI cho trước và mã hóa nó để có thể sử dụng theo chuẩn. Ví dụ, tài liệu RFC (Request for comments) 2396 định nghĩa các cú pháp chung cho URI. Phương thức *encodeURI()* có thể được dùng để sửa lại địa chỉ URI sau:

`http://www.braingia.org/a uri with spaces.htm`

Địa chỉ URI trên có chứa các khoảng trắng không hợp lệ cho HTTP URI, do đó URI cần phải được mã hóa:

`alert(encodeURI("http://www.braingia.org/a uri with spaces.htm"))`

Hình 8-5 hiển thị kết quả.



HÌNH 8-5 Sử dụng phương thức encodeURI() để mã hóa đúng địa chỉ URI trong JavaScript.

Trong khi phương thức `encodeURI` làm việc với toàn bộ URI, như trong Hình 8-5, phương thức `encodeURIComponent()` chỉ làm việc trên một phần của URI, như phần `/a uri with spaces.htm` trong ví dụ trên.

Cả 2 phương thức `encodeURI()` và `encodeURIComponent()` đều có các phương thức giải mã tương ứng là `decodeURI()` và `decodeURIComponent()`.

Sử dụng phương thức `eval()`

Phương thức `eval()` là một trong những phương thức mạnh nhưng cũng nguy hiểm trong JavaScript. Phương thức `eval()` nhận vào một đối số sẽ được thông dịch và thực thi bởi JavaScript, ví dụ:

```
eval("alert('helloworld')");
```

Phương thức `eval()` thực thi đoạn mã `alert`, giống như khi đoạn mã đó được thực thi trực tiếp. Thông thường, bạn sử dụng `eval()` khi gọi AJAX, tuy nhiên việc đó sẽ gây ra vấn đề bảo mật vì đoạn mã trả về từ lời gọi AJAX có thể thực thi như các đoạn mã thông thường trong khi nó có thể là mã độc.

Bài tập

- Viết đoạn mã để duyệt từng phần tử của một mảng có bốn phần tử như bên dưới và hiển thị chúng bằng hộp thoại `alert()`, mỗi phần tử một hộp thoại:

```
var star = ["Polaris", "Deneb", "Vega", "Altair"];
```

- Tạo một đối tượng chứa tên ba bài hát yêu thích của bạn. Đối tượng này cần có những thuộc tính như tên nghệ sĩ, độ dài, tựa đề cho từng bài hát.
- Sau đây là ví dụ về danh sách các ngôi sao và lớp tạo ra các đối tượng này:

```
function Star(constell, type, specclass, magnitude) {  
    this.constellation = constell;  
    this.type = type;  
    this.spectralClass = specclass;  
    this.mag = magnitude;  
}  
star["Polaris"] = new Star("Ursa Minor", "Double Cepheid", "F0 V");  
star["Mizar"] = new Star("Ursa Major", "Spectroscopic Binary", "A0 V");  
star["Aldebaran"] = new Star("Taurus", "Irregular Variable", "K5 II");  
star["Rigel"] = new Star("Orion", "Supergiant with Companion", "B8 Ia");  
star["Castor"] = new Star("Gemini", "Multiple/Spectroscopic", "A0 V");  
  
star["Albireo"] = new Star("Cygnus", "Double", "K3 II", 3.1);  
star["Acrux"] = new Star("Crux", "Double", "B1 IV", 0.8);  
star["Gemma"] = new Star("Corona Borealis", "Eclipsing Binary", "G5 V");  
star["Procyon"] = new Star("Canis Minor", "Double", "F5 IV", 0.1);  
star["Sirius"] = new Star("Canis Major", "Double", "A1 V", -1.46);  
star["Rigil Kentaurus"] = new Star("Centaurus", "Double", "G2 V", 0.05);  
star["Deneb"] = new Star("Cygnus", "Supergiant", "A2 Ia", 1.25);  
star["Vega"] = new Star("Lyra", "White Dwarf", "A0 V", 0.03);
```

```
star["Altair"] = new Star("Aquila", "White Dwarf", "A7 V", 0.1
```

Đoạn mã sử dụng vòng lặp *for* đơn giản để duyệt qua từng đối tượng *star* và hiển thị tên của chúng:

```
for (var propt in star) {  
    alert(propt);  
}
```

Nhiệm vụ của bạn là sửa đoạn mã này để chỉ hiển thị một hộp thoại thông báo chứa tất cả tên các ngôi sao thay vì hiển thị mỗi ngôi sao một hộp thoại.



Chương 9

Mô hình đối tượng trình duyệt

Sau khi đọc xong chương này, bạn có thể:

- Hiểu về các đối tượng con có sẵn trong đối tượng *window*.
- Sử dụng đối tượng *navigator* để xem các thuộc tính trong trình duyệt của người dùng.
- Lấy thông tin chiều cao và chiều rộng của màn hình khách truy cập.
- Sử dụng JavaScript để kiểm tra trình duyệt có đang bật Java không.
- Phân tích chuỗi truy vấn gửi từ trình duyệt.

Giới thiệu về trình duyệt



Trước chương này, bạn chủ yếu tiếp cận lý thuyết về JavaScript. Từ chương này trở đi, chúng ta sẽ bắt đầu xem JavaScript được áp dụng vào thực tế ra sao.

Có một thực tế hiển nhiên nhưng rất quan trọng mà chúng ta cần nhắc lại: *Trình duyệt là trung tâm trong lập trình JavaScript*. Những dự án như Rhino (<http://www.mozilla.org/rhino/>) muốn thay đổi điều này, nhưng việc hiểu môi trường mà trình duyệt cung cấp là yếu tố quyết định để viết những đoạn mã JavaScript có thể chạy tốt trên nhiều loại trình duyệt và nền tảng khác nhau. Phần này sẽ giới thiệu với bạn về Mô hình đối tượng trình duyệt (Browser Object Model - BOM).

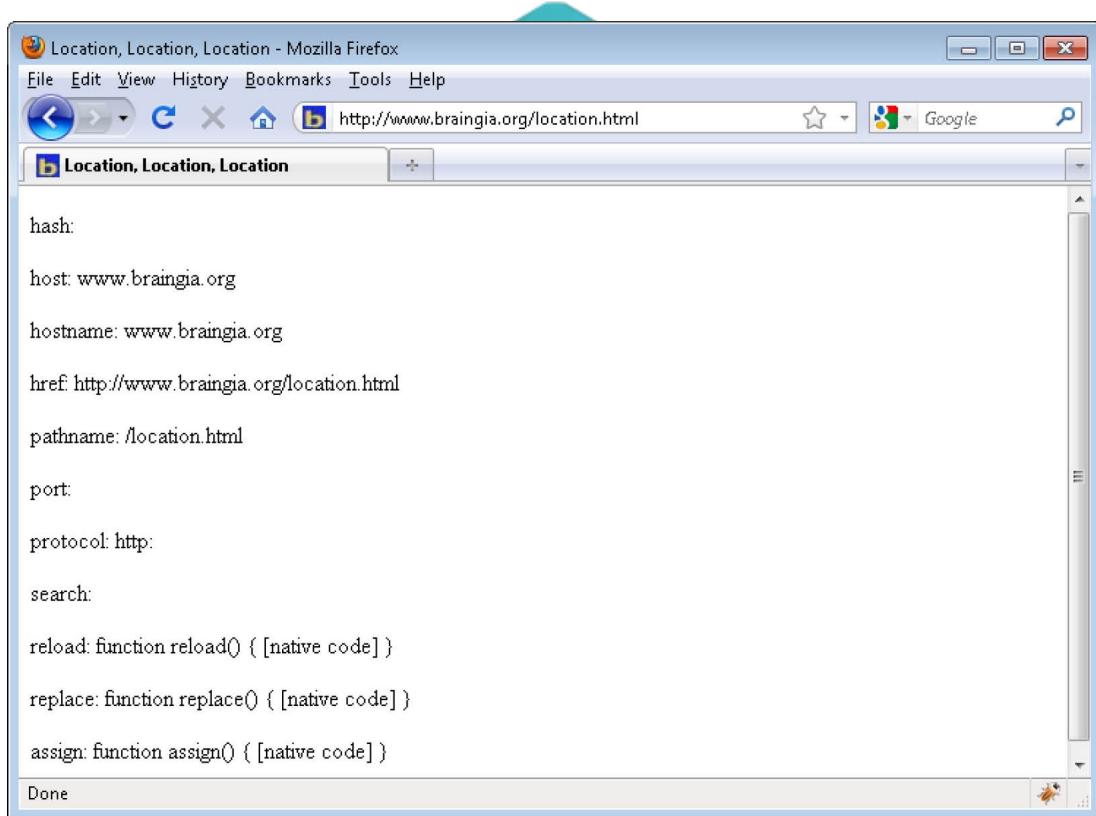
Hệ thống phân cấp của trình duyệt

Mô hình đối tượng trình duyệt tạo ra một hệ thống phân cấp đối tượng theo hình cây, rất nhiều trong số đó cung cấp các thuộc tính và phương thức cho lập trình viên JavaScript. Bản thân trình duyệt được biểu diễn bởi một đối tượng gọi là

window. Đối tượng *window* có một số đối tượng con sau:

- *document*
- *frames*
- *history*
- *location*
- *navigator*
- *screen*
- *self/window/parent*

Đối tượng con *document* của *window* khá đặc biệt vì nó có thêm một số đối tượng con khác, thậm chí cả các đối tượng cháu. Hình 9-1 minh họa đối tượng *window*, các đối tượng con và vị trí của chúng trong cây phân cấp.



HÌNH 9-1 Đối tượng *window* và đối tượng con của nó.

Chúng ta sẽ thảo luận về đối tượng *document* trong một chương riêng –

Chương 10 “Mô hình đối tượng tài liệu”. Chương này sẽ giới thiệu về các đối tượng con khác của đối tượng *window*.

Sự kiện

Sự kiện đã được giới thiệu sơ lược trong Chương 1 - “Hiểu hơn về JavaScript”. Bạn cần dùng nhiều đến sự kiện khi lập trình JavaScript và làm việc với các form trên web. Sự kiện xảy ra khi một hành động xảy ra. Hành động này có thể bắt nguồn từ người dùng khi họ nhấn vào một nút hay một đường liên kết hoặc di chuyển con trỏ chuột ra hoặc vào một vùng nào đó. Sự kiện cũng có thể bắt nguồn từ những sự kiện của ứng dụng, ví dụ khi trang web đang tải. Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt” sẽ trình bày chi tiết hơn về các sự kiện liên quan đến đối tượng *window*; Chương 14 “Sử dụng JavaScript với Web Form” cung cấp nhiều thông tin hơn về form web.

Bàn luận về đối tượng *window*



Đối tượng *window* là đối tượng toàn cục gắn với cửa sổ đang mở trong trình duyệt. Đối tượng *window* có một vài thuộc tính, phương thức và đối tượng con. Bạn đã sử dụng một số phương thức này như *alert()* và *prompt()*. Vì *window* là đối tượng toàn cục nên bạn không cần đặt từ *window* trước các thuộc tính và phương thức. Thay vào đó, bạn có thể trực tiếp gọi chúng như trong các ví dụ gọi thẳng đến phương thức *alert()*.

Các đối tượng con cháu trực tiếp của *window* không cần phải có tiền tố *window*, nhưng khi làm việc với các đối tượng không phải là đối tượng con cháu trực tiếp của *window*, bạn cần thêm tên đối tượng *window* trước nó. Ví dụ, đối tượng *document* là đối tượng con trực tiếp của đối tượng *window* do đó không cần thêm tiền tố *window*, nhưng những đối tượng con cháu của *document* thì cần như trong ví dụ dưới đây:

```
alert("something"); // không cần tiền tố window.  
document.forms[0] // cần tiền tố document. nhưng không cần w.
```

Đối tượng *window* cũng có các thuộc tính và phương thức. Trong số các thuộc tính đó có những thuộc tính tự thân, tức là những thuộc tính thuộc về *window*.

Bảng 9-1 liệt kê những thuộc tính thường dùng của đối tượng *window*. Bạn sẽ tìm hiểu các thuộc tính này thông qua các ví dụ xuyên suốt cuốn sách.

BẢNG 9-1 Những thuộc tính của đối tượng *window*.

| Thuộc tính | Mô tả |
|----------------------|--|
| <i>closed</i> | Có giá trị <i>true</i> khi cửa sổ bị đóng. |
| <i>defaultStatus</i> | Dùng để thiết lập dòng chữ mặc định trên thanh trạng thái của trình duyệt. |
| <i>name</i> | Tên của cửa sổ khi nó mới mở ra lần đầu. |
| <i>opener</i> | Tham chiếu đến cửa sổ tạo ra nó. |
| <i>parent</i> | Thường dùng để chỉ frame/window cha chứa/sinh ra frame/window hiện tại. |
| <i>status</i> | Thường dùng để thiết lập đoạn văn bản sẽ hiển thị trên thanh trạng thái khi người dùng di chuột qua một phần tử, ví dụ như đường liên kết. |
| <i>top</i> | Chỉ cửa sổ cha ở trên cùng. |

Bảng 9-2 và 9-3 mô tả một số phương thức của đối tượng *window*. Bạn sẽ biết cách sử dụng những phương thức này thông qua các ví dụ trong phần còn lại của cuốn sách.



BẢNG 9-2 Một số phương thức của đối tượng *window*.

| Phương thức | Mô tả |
|------------------------------|---|
| <i>addEventListener()</i> | Phương thức tạo hàm xử lý sự kiện trên các trình duyệt (trừ Windows Internet Explorer). Xem Chương 11 để biết thêm. |
| <i>attachEvent()</i> | Phiên bản của <i>addEventListener()</i> trong Internet Explorer. Xem Chương 11 để biết thêm. |
| <i>blur()</i> | Thay đổi tiêu điểm của bàn phím ra khỏi cửa sổ trình duyệt. |
| <i>focus()</i> | Chuyển tiêu điểm của bàn phím vào cửa sổ trình duyệt. |
| <i>close()</i> | Đóng cửa sổ trình duyệt. |
| <i>detachEvent()</i> | Phiên bản của <i>removeEventListener()</i> trong Internet Explorer. |
| <i>removeEventListener()</i> | Phương thức bỏ hàm xử lý sự kiện trên các trình duyệt (trừ Internet Explorer). |
| <i>open()</i> | Mở một cửa sổ. |
| <i>print()</i> | Gọi chức năng in từ trình duyệt: xử lý tương tự như khi người dùng nhấn Print trong trình duyệt. |

Một số phương thức để di chuyển và thay đổi kích cỡ cửa sổ của *window* được mô tả trong Bảng 9-3.

BẢNG 9-3 Một số phương thức di chuyển và thay đổi kích thước cửa sổ của đối tượng *window*.

| Phương thức | Mô tả |
|-------------------|---|
| <i>moveBy()</i> | Dùng để di chuyển cửa sổ đến vị trí tương đối. |
| <i>moveTo()</i> | Sử dụng để di chuyển đến một vị trí nhất định. |
| <i>resizeBy()</i> | Dùng để thay đổi kích cỡ cửa sổ một lượng bằng giá trị cho trước. |
| <i>resizeTo()</i> | Dùng để thay đổi kích cỡ cửa sổ đến kích thước nhất định. |

Các phương thức liên quan đến thời gian trong JavaScript sẽ được đề cập trong Chương 11. Sau đây là một vài phương thức liên quan đến thời gian của đối tượng *window*:

- *clearInterval()*
- *clearTimeout()*
- *setInterval()*
- *setTimeout()*



Phần còn lại của chương sẽ đi sâu hơn vào việc tìm hiểu các đối tượng con của *window*.

Lấy thông tin về màn hình

Đối tượng *screen* cung cấp phương thức để lấy thông tin về màn hình của người truy cập. Bạn cần những thông tin này để xác định xem nên hiển thị ảnh nào hoặc trang web có thể lớn đến đâu để điều chỉnh. Bất kể bạn có sử dụng đối tượng *screen* hay không, bạn vẫn cần tạo một thiết kế CSS cơ bản tốt để hỗ trợ mọi kích cỡ màn hình.

Chú ý Bạn thường thấy đối tượng con của *window* được coi là thuộc tính – ví dụ, thuộc tính *screen* thường được gọi hơn là đối tượng *screen*.

Những thuộc tính sẵn có của đối tượng *screen* là:

- *availHeight*
- *availWidth*
- *colorDepth*
- *height*
- *width*

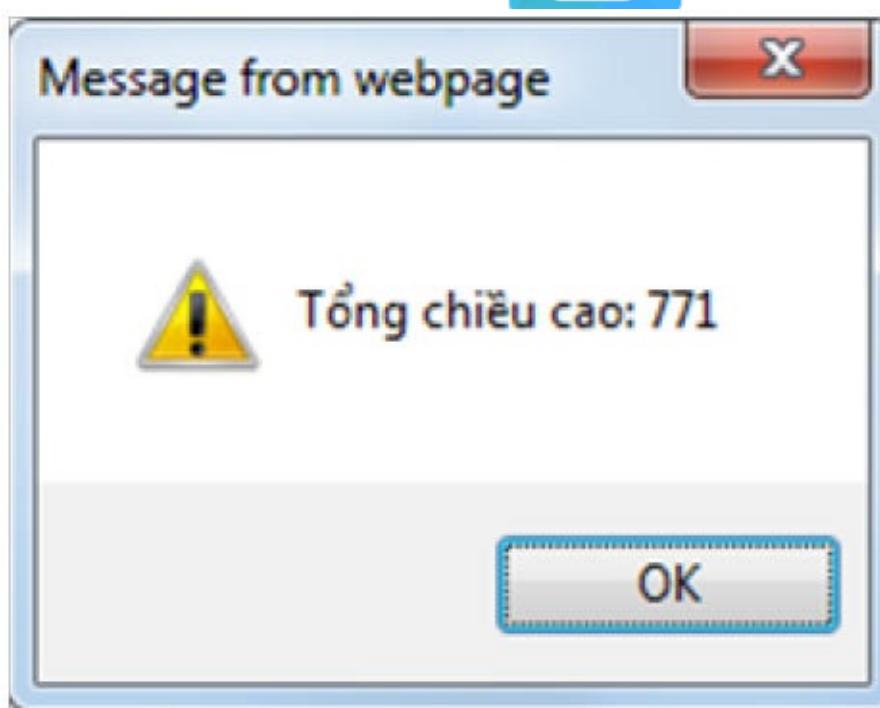
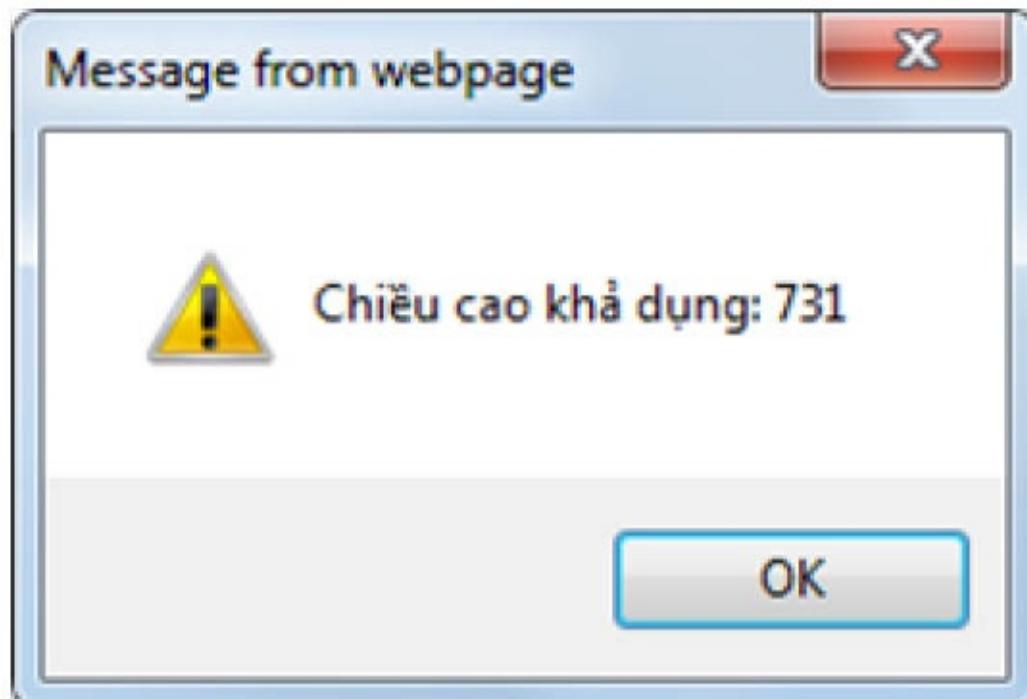
Bạn có thể sẽ băn khoăn về sự khác nhau giữa thuộc tính *availHeight*, *availWidth* và thuộc tính *height* và *width*. Thuộc tính *availHeight* và *availWidth* trả về chiều cao và chiều rộng còn lại của màn hình sau khi đã trừ đi diện tích được sử dụng bởi các tác vụ điều khiển khác, ví dụ như thanh tác vụ của Microsoft Windows. Đối tượng *height* và *width* trả về chiều cao và chiều rộng của toàn bộ màn hình. Ví dụ sau sẽ giúp bạn hiểu rõ hơn.

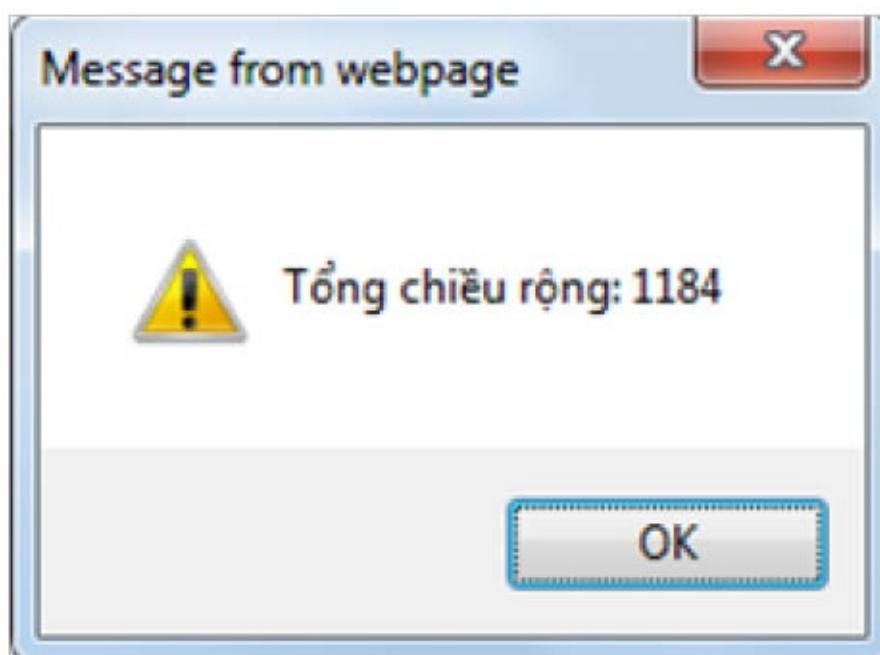
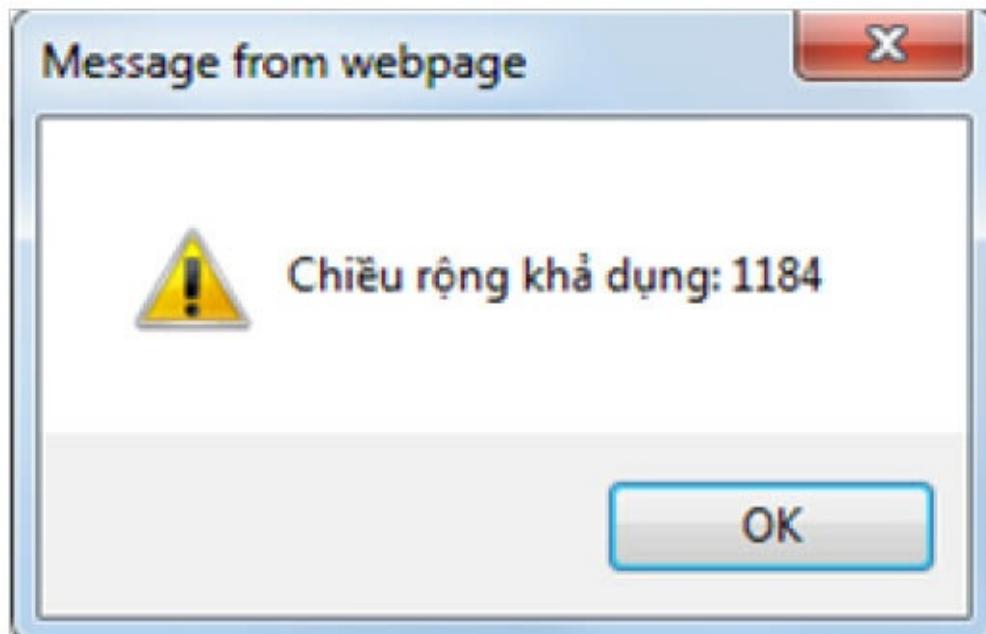
Xác định chiều cao và chiều rộng của màn hình người truy cập

1. Sử dụng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác sửa file screen.htm trong thư mục mã nguồn mẫu Chương 9 phần Tài nguyên đi kèm.
2. Thêm vào đoạn mã được in đậm dưới đây (file screen.txt, Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4. 01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Đối tượng Screen</title>
</head>
<body>
<script type="text/javascript">
    alert("Chiều cao khả dụng:"+ screen.availHeight);
    alert("Tổng chiều cao:"+ screen.height);
    alert("Chiều rộng khả dụng:"+ screen.availWidth);
    alert("Tổng chiều rộng:"+ screen.width);
</script>
</body>
</html>
```

3. Lưu file và dùng trình duyệt mở lại. Bạn nhận được bốn hộp thoại thông báo ứng với từng thuộc tính được gọi. Hình minh họa dưới đây phản ánh màn hình có độ phân giải 1024 x 768.





Như bạn đã thấy, chiều rộng và chiều cao của màn hình lần lượt là 1184 và 771 pixels. Chú ý rằng chiều rộng khả dụng chỉ còn lại 1184, chiều cao khả dụng cũng giảm từ 771 xuống 731 do thanh tác vụ.

Sử dụng đối tượng *navigator*

Đối tượng *navigator* cung cấp một số thuộc tính giúp nhận biết các thành phần khác nhau của trình duyệt và môi trường của người dùng. Một trong những thao tác phổ biến mà JavaScript có thể thực hiện là nhận biết người truy cập đang sử dụng trình duyệt nào. (Phần này không trình bày thao tác đó. Xem mục “Các vấn đề khi nhận biết trình duyệt” để biết thêm chi tiết).

Các vấn đề khi nhận biết trình duyệt

Từ rất lâu, các trang web đã sử dụng đối tượng *navigator* để nhận biết trình duyệt của người dùng. (Thời gian dài trong thời đại Internet có thể là một vài năm hoặc chỉ là vài tháng, tùy thuộc vào việc chúng ta đang bàn về công nghệ nào). Nhận biết trình duyệt là việc làm cần thiết để các đoạn mã viết riêng cho từng loại trình duyệt có thể chạy được. Trái với mục đích cơ bản của việc nhận biết trình duyệt, một số website thiết kế kém lại sử dụng kỹ thuật này để chặn khi người dùng duyệt nó trên một số trình duyệt nhất định.

Rất ít người biết rằng thông tin gửi qua trình duyệt có thể bị giả mạo. Tiện ích The User Agent Switcher của Firefox có thể thay đổi những thông tin này, do đó làm cho việc nhận dạng trình duyệt với đối tượng *navigator* trở nên vô tác dụng.



Mách nhỏ Chúng ta đã bàn đến vấn đề này trước đây và bây giờ chúng ta sẽ nói lại (có thể điều này sẽ còn được lặp lại): Đừng bao giờ tin tưởng các thông tin được gửi về từ trình duyệt người dùng. Luôn luôn xác thực chúng. Tin rằng trình duyệt trả về là Internet Explorer vì thông tin trả về nói thế là không đủ. Chương 11 sẽ trình bày một phương thức giúp nhận biết tốt hơn liệu trình duyệt có khả năng chạy mã JavaScript trên trang web của bạn hay không.

Khi sử dụng đối tượng *navigator* để nhận biết trình duyệt, bạn sẽ gặp thêm một vấn đề khác vì có rất nhiều loại trình duyệt khác nhau. Lập trình viên web có thể sẽ tiêu tốn khá nhiều thời gian để theo dõi xem trình duyệt nào hỗ trợ được tính năng nào và cố gắng làm tất cả các tính năng chạy được trên các trình duyệt.

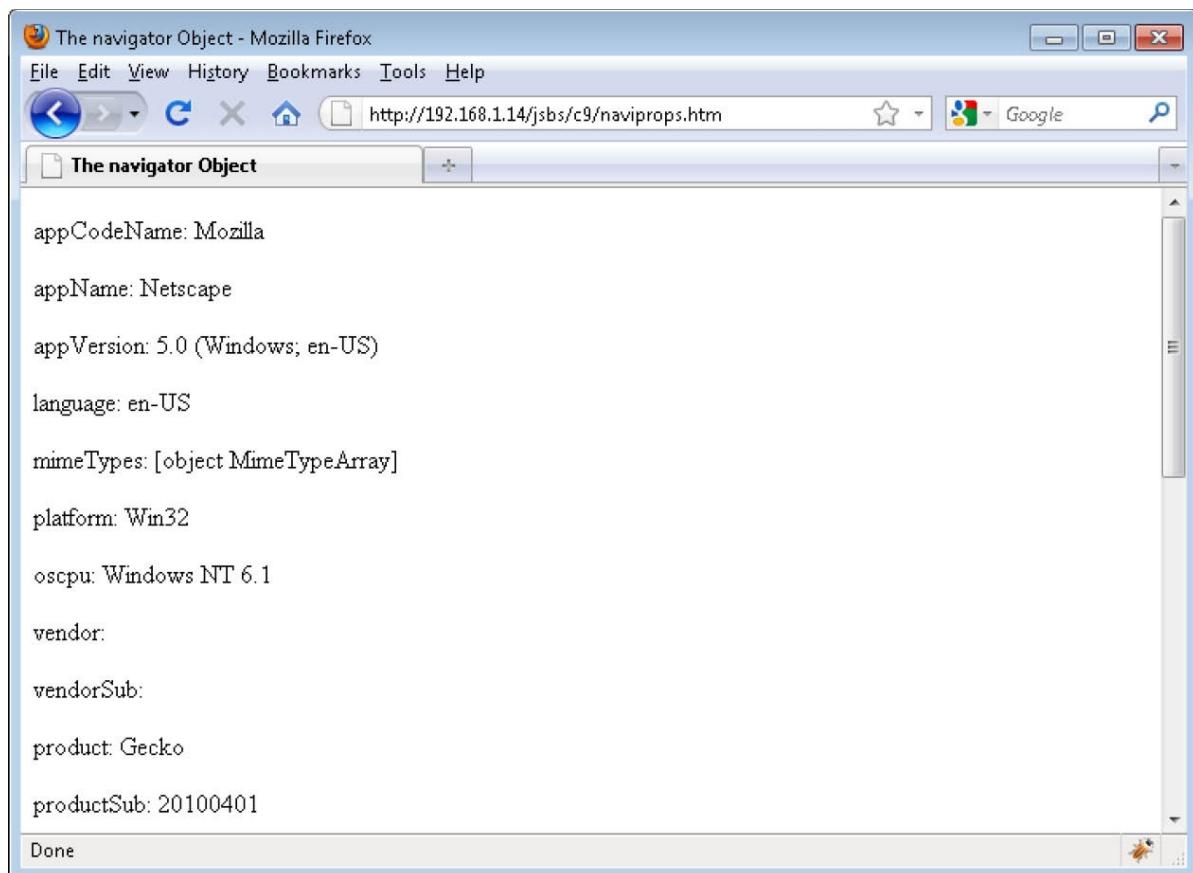
Trong bài tập này bạn sẽ tìm hiểu các thuộc tính của đối tượng *navigator* và giá trị của chúng.

Tìm hiểu đối tượng *navigator*

1. Sử dụng Visual Studio, Eclipse hoặc trình soạn thảo khác sửa file naviprops.htm trong thư mục mã nguồn mẫu Chương 9 của Tài nguyên đi kèm.
2. Thay thế dòng chú thích TODO bằng đoạn mã được in đậm dưới đây (đoạn mã này có trong file naviprops.txt của Tài nguyên đi kèm)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4. 01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Đối tượng Navigator</title>  
</head>  
<body>  
    <script type="text/javascript">  
        //lấy các phần tử bằng tên thẻ  
        var body = document.getElementsByTagName('br/>  
        for (var prop in navigator) {  
            // tạo mới phần tử  
            var text =  
                document.createTextNode(prop+ ":"-  
                elem.appendChild(text); //thêm nút  
                body.appendChild(elem);  
        }  
    </script>  
</body>  
</html>
```

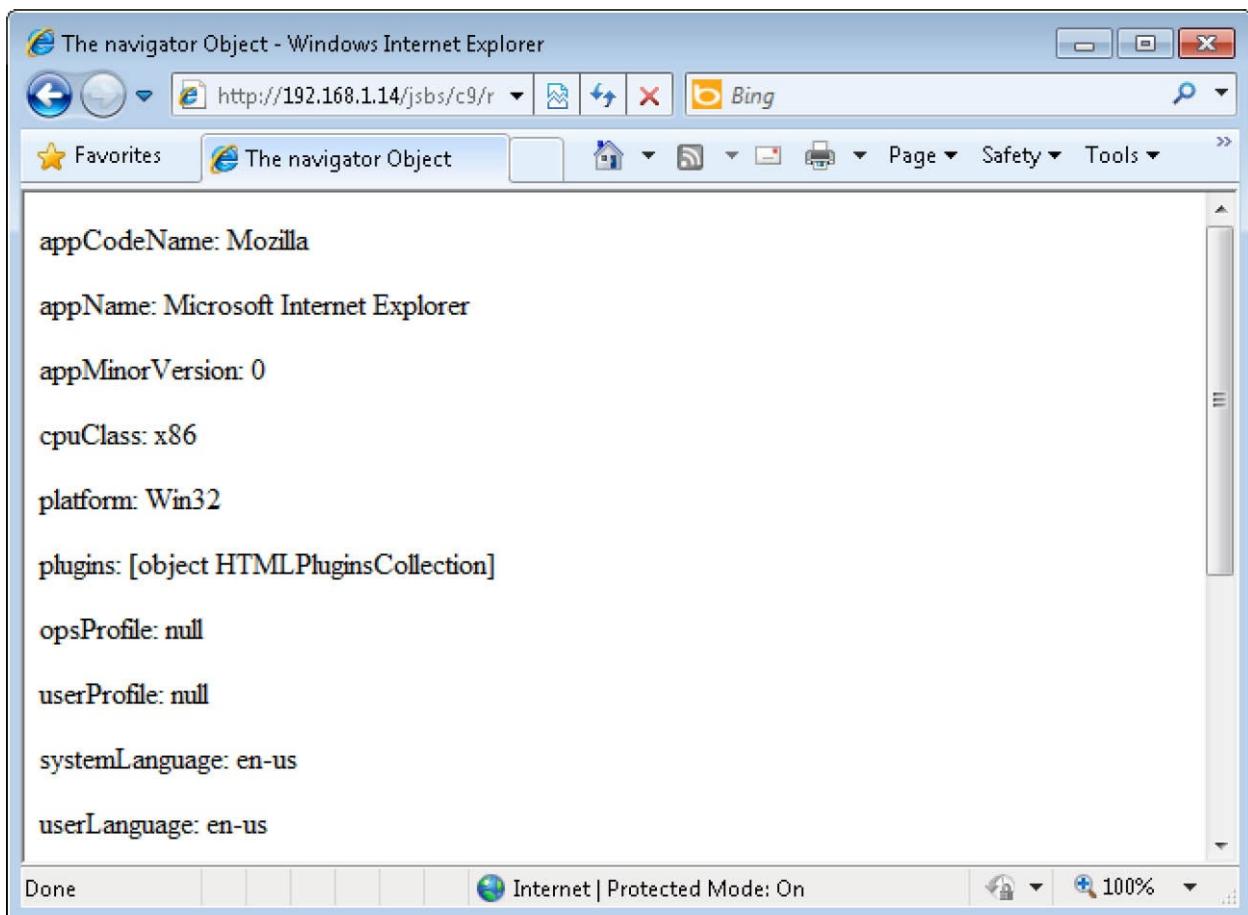
3. Lưu file và dùng trình duyệt mở lại. Nếu chọn Firefox, trang web của bạn sẽ giống như hình dưới:



The screenshot shows a Mozilla Firefox window with the title "The navigator Object - Mozilla Firefox". The address bar displays the URL "http://192.168.1.14/jsbs/c9/naviprops.htm". The main content area is titled "The navigator Object" and lists various properties of the Navigator object:

```
appCodeName: Mozilla
appName: Netscape
appVersion: 5.0 (Windows; en-US)
language: en-US
mimeTypes: [object MimeTypeArray]
platform: Win32
oscpu: Windows NT 6.1
vendor:
vendorSub:
product: Gecko
productSub: 20100401
```

4. Nếu chọn Internet Explorer, ta sẽ có trang tương tự, tuy vậy hãy chú ý đến điểm khác biệt giữa các thuộc tính.



Chúng ta không sử dụng hộp thoại `alert()` trong bài tập này, thay vào đó chúng ta sẽ dùng một số hàm mới. Những phần tử trong ví dụ này sẽ được giới thiệu trong Chương 10 và 11.

Đoạn mã trong bài tập này chứa một hàm, hàm này sử dụng mô hình đối tượng `tài liệu` để tạo ra các phần tử HTML trên trang web. Vòng lặp `for` được dùng để duyệt từng thuộc tính của đối tượng `navigator`:

```
function showProps() {
    var body = document.getElementsByTagName("body")[0];
    for (var prop in navigator) {
        var elem = document.createElement("p");
        var text = document.createTextNode(prop+ ":"+
            elem.appendChild(text);
        body.appendChild(elem);
    }
}
```

Nếu đoạn mã JavaSript của bạn không chạy được trên một phiên bản của trình

duyệt nào đó, bạn có thể xử lý bằng cách nhận biết trình duyệt thông qua đối tượng *navigator* nhưng cần cân nhắc vì phương pháp này không thật sự đáng tin cậy và nên hạn chế. Tuy vậy, đôi khi bạn lại cần sử dụng phương pháp này.

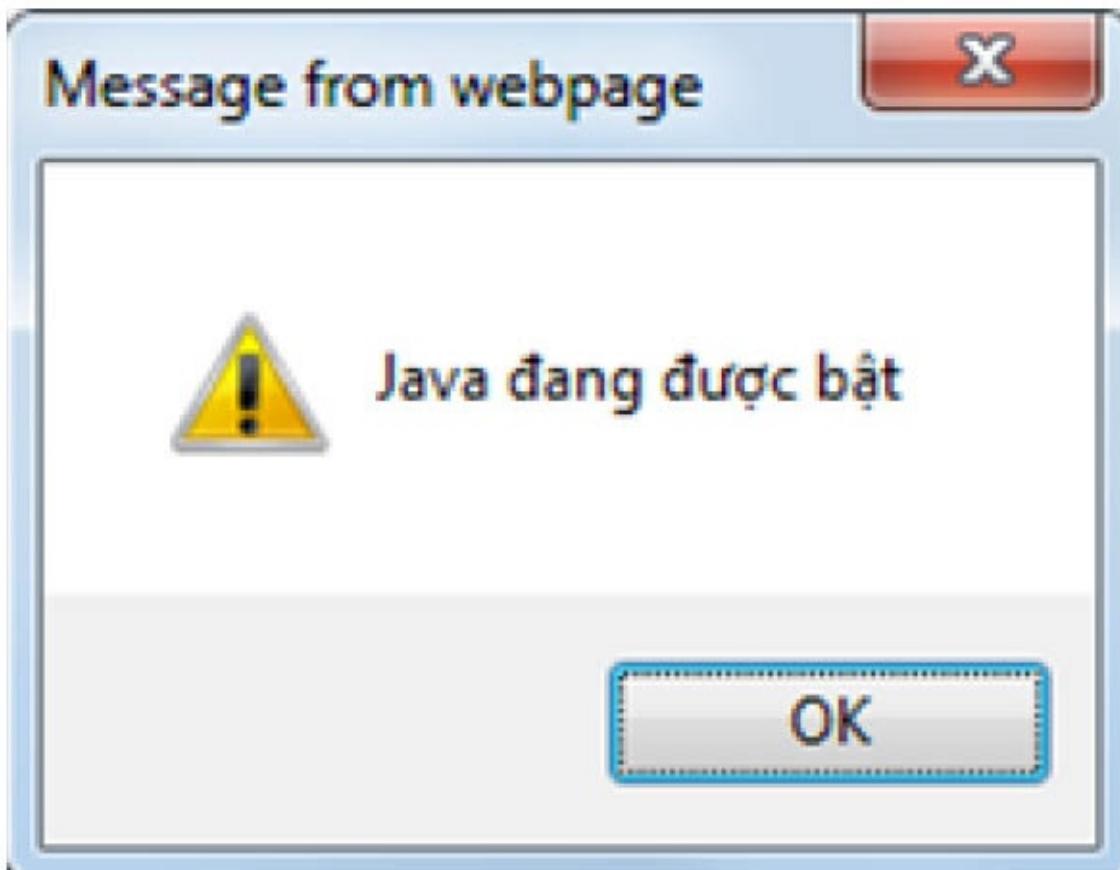
Nếu trang web của bạn sử dụng Java, bạn có thể sử dụng đối tượng *navigator* để kiểm tra Java đã được bật hay chưa. Cách làm như sau:

Sử dụng đối tượng *navigator* để nhận biết Java

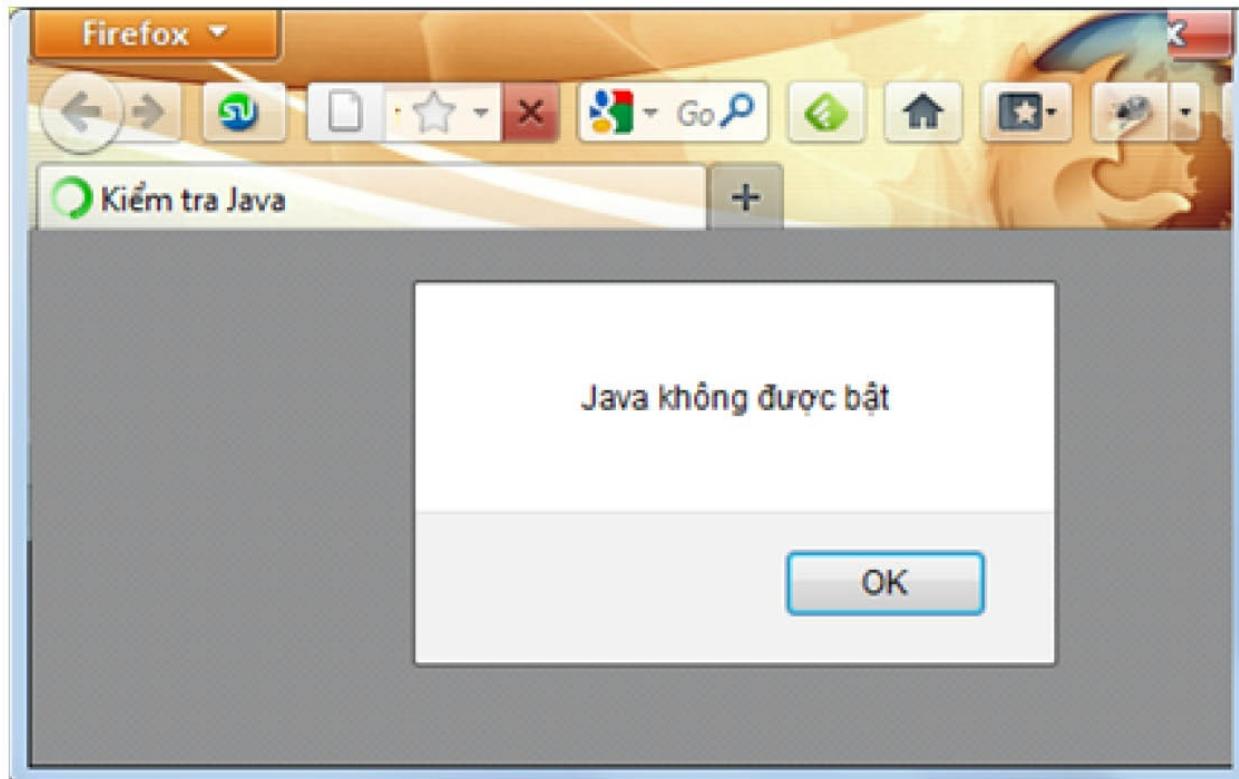
1. Sử dụng Visual Studio, Eclipse hoặc trình soạn thảo khác để mở file javatest.htm trong thư mục mã nguồn mẫu Chương 9.
2. Thay thế dòng chú thích TODO bằng đoạn mã được in đậm dưới đây (đoạn mã nằm trong file javatest.txt của Tài nguyên đi kèm).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4. 01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
  <head>  
    <title>Kiểm tra JavaScript có được bật hay không</title>  
    <script type="text/javascript">  
      if(navigator.javaEnabled()){  
        alert("Java đang được bật");  
      }else{  
        alert("Java không được bật");  
      }  
    </script>  
  </head>  
  <body>  
  </body>  
</html>
```

3. Lưu file này và xem trên Internet Explorer (nếu đã cài đặt nó). Mặc định trình duyệt Internet Explorer của bạn đã bật Java, bạn sẽ thấy hộp thoại thông báo như sau:



4. Đổi sang Firefox, tắt Java (trong phiên bản Firefox của hệ điều hành Windows, chọn Add-Ons từ menu Tools, chọn Plugin và chọn Disable For the Java Plugins). Khi bạn tắt Java, tải lại trang web, bạn sẽ thấy một hộp thoại như sau:

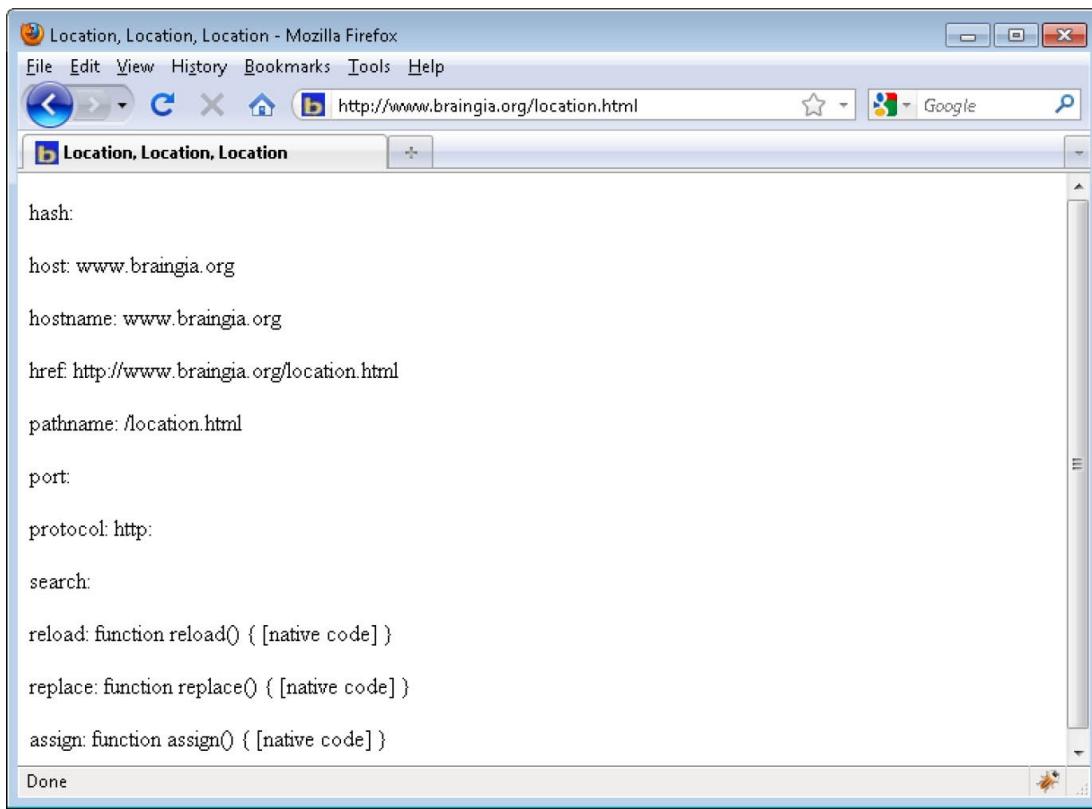


Đối tượng *location*

Đối tượng *location* cho phép bạn truy cập URI của trang web hiện tại, bao gồm bất kỳ thông tin nào về chuỗi truy vấn, giao thức đang sử dụng và các thành phần liên quan. Đây là ví dụ về URI:

http://www.braingia.org/location.html

Nếu trang web ở địa chỉ URI đó chạy đoạn mã JavaScript trong ví dụ sau, kết quả sẽ giống như Hình 9-2.



HÌNH 9-2 Đối tượng location được sử dụng để hiển thị các thuộc tính khác nhau.

Giao thức trong trường hợp này là `http:`, tên host là `www.braigia.org`, đường dẫn là `location.html`. Không có giá trị nào trong chuỗi truy vấn, vì thế giá trị tìm kiếm là rỗng. Cổng (port) được dùng là cổng chuẩn cho giao thức HTTP, `tcp/80`, vì thế port cũng mang giá trị rỗng.

Sau đây là ví dụ về việc sử dụng chuỗi truy vấn.

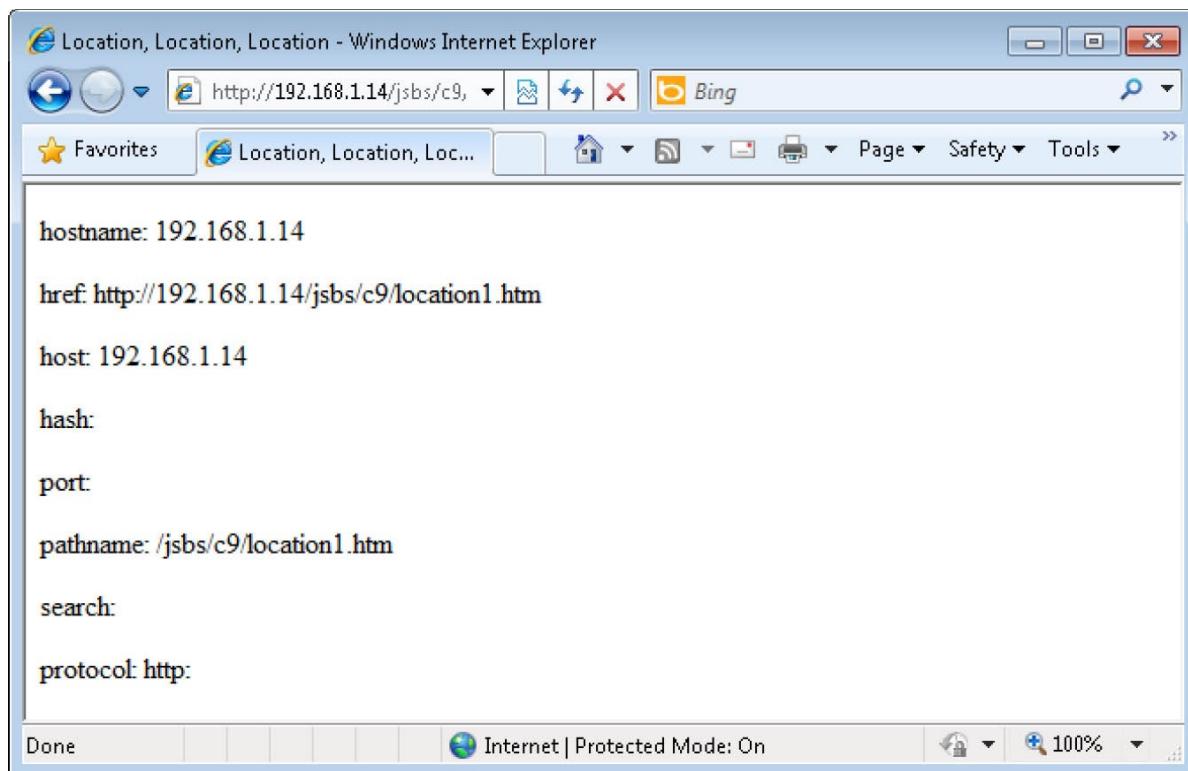
Tìm hiểu đối tượng *location*

1. Sử dụng VisualStudio, Eclipse hoặc một trình soạn thảo khác mở file `location1.htm` trong thư mục mã nguồn mẫu Chương 9 của Tài nguyên đi kèm.
2. Phần HTML và JavaScript đầu tiên tạo một trang web như trong Hình 9-2. (Thực ra, đây là đoạn mã lấy từ một ví dụ trước đó cho đối tượng `navigator` và được thay đổi cho đối tượng `location`). Chúng ta sử dụng đoạn mã đó cho bài tập này, bạn hãy thêm đoạn mã in đậm dưới đây vào trang `location1.htm`:

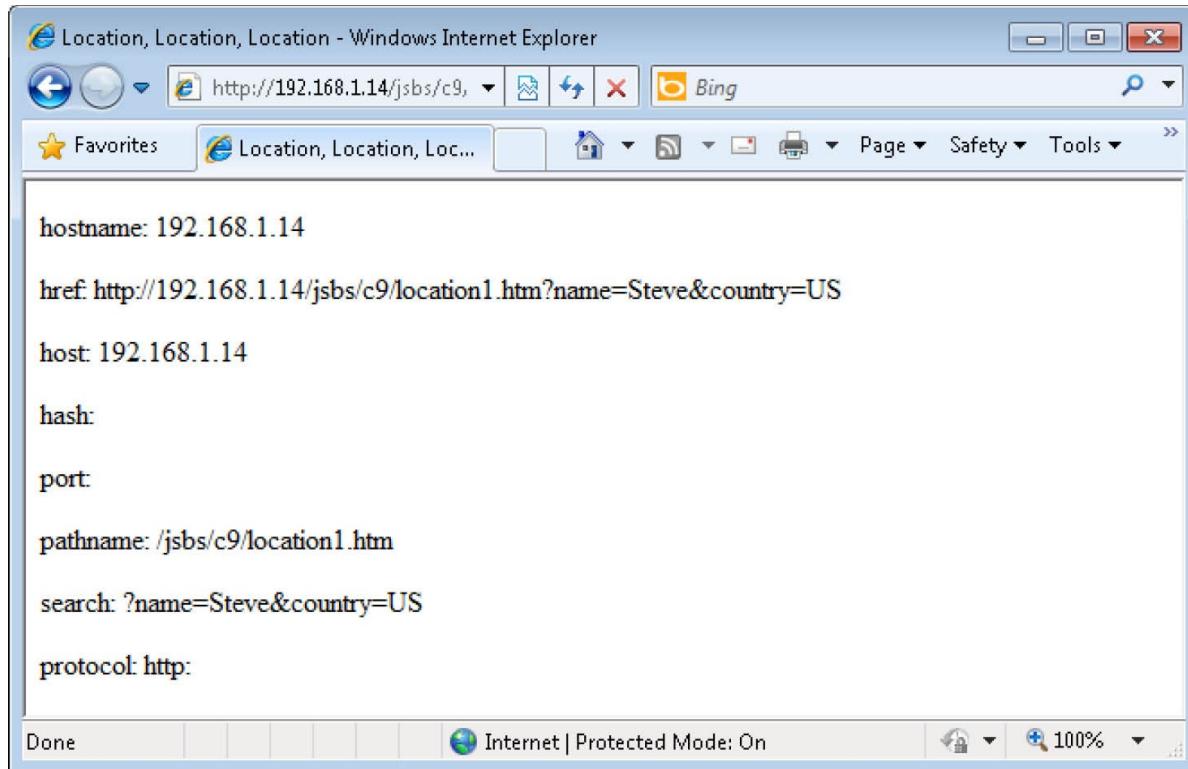
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
    <title>Location, Location, Location</title>  
</head>  
<body>  
    <script type="text/javascript">  
        //lấy các phần tử bằng tên thẻ  
        var body = document.getElementsByTagName('br/>        for (var prop in location) {  
            //tạo phần tử  
            var elem = document.createElement('br/>            //tạo nút văn bản  
            var text =  
                document.createTextNode(prop+ ":"-  
                elem.appendChild(text); //thêm phầ  
                body.appendChild(elem);  
        }  
    </script>  
</body>  
</html>
```



3. Dùng trình duyệt mở trang web. Kết quả trả về sẽ khác nhau tùy thuộc vào việc chúng ta cài đặt máy chủ web nào. Ví dụ này mô tả máy chủ web Apache chạy trên địa chỉ IP cục bộ: 192.168.1.14.



4. Sửa địa chỉ URI mà bạn dùng để gọi đến trang web bằng cách thêm vào một số cặp tham số/giá trị cho chuỗi truy vấn. Ví dụ, URI của môi trường mà tôi dùng là *http://192.168.1.14/jsbs/c9/location1.htm*. (Môi trường và vị trí bạn gọi đến file có thể khác ở đây). Sửa địa chỉ URL và thêm vào hai tham số, *name=Steve* và *country=US*. Bạn có thể thay đổi các tham số bằng tên và nước của mình (nếu bạn không phải là người Mỹ). Giá trị bạn chọn ở đây không quá quan trọng, vẫn đề là bạn sử dụng nhiều hơn một cặp tham số/giá trị. Đây là URI hoàn chỉnh *http://localhost:1627/Chapter9/location1.htm?name=Steve&country=US*.
5. Khi tải lại trang web với tham số vừa thêm vào, thuộc tính *search* lúc này có giá trị như hình sau:



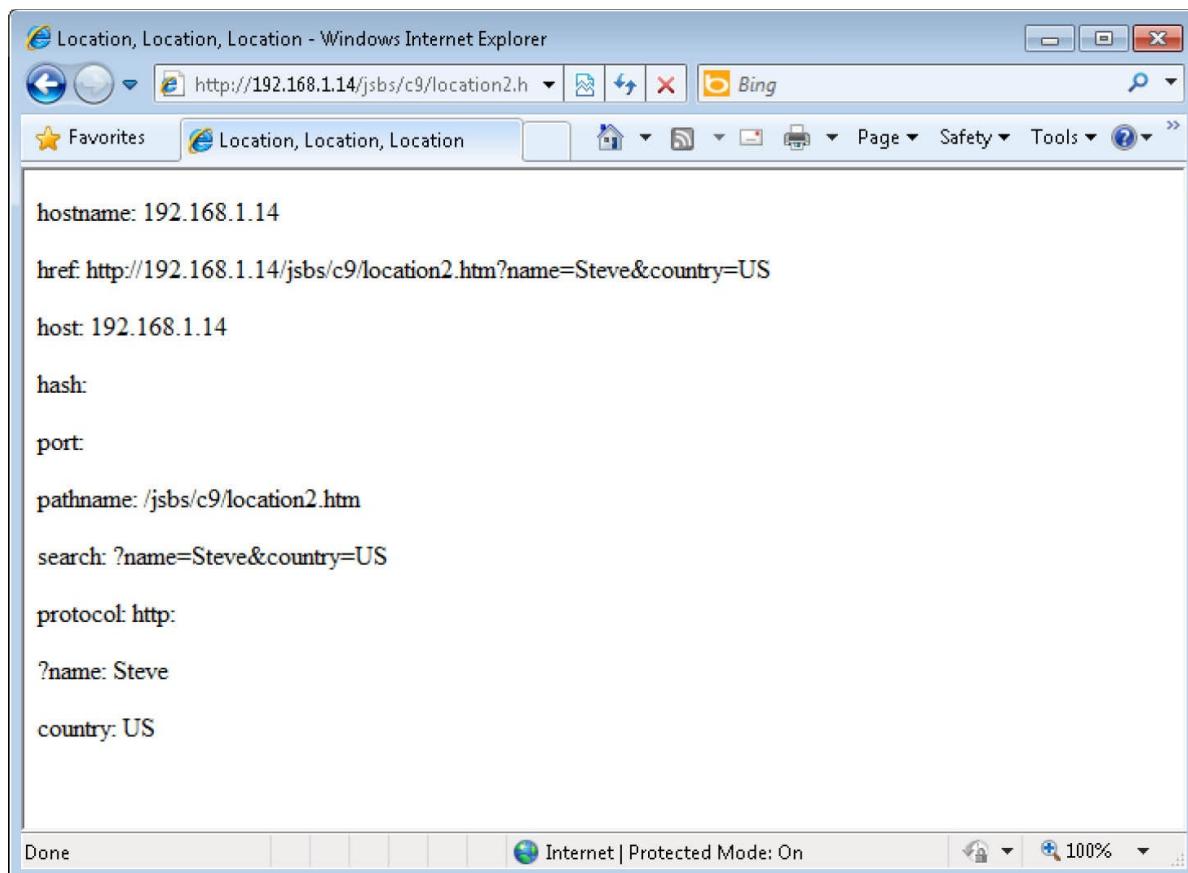
6. Mở file location1.htm và lưu lại với tên **location2.htm**.
7. Sửa file location2.htm để nó kiểm tra thuộc tính *search* như sau (phần thay đổi được in đậm, bạn có thể tìm thấy ví dụ này trong file location2.txt của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Location, Location, Location</title>
</head>
<body>
    <script type="text/javascript">
        //lấy các phần tử bằng tên thẻ
        var body = document.getElementsByTagName('
        for (var prop in location) {
            // tạo phần tử
            var elem = document.createElement(
                //tạo nút văn bản
                var text =
                    document.createTextNode(prop+ ":"-
```

```
        elem.appendChild(text); //thêm phẩn tử text vào elem
    }
    if(location.search){
        var querystring= location.search;
        var splits= querystring.split('&');
        for (var i= 0; i<splits.length; i++){
            var splitpair= splits[i].split('=');
            var elem= document.createElement('p');
            var text = document.createTextNode(splitpair[1]);
            elem.appendChild(text);
            body.appendChild(elem);
        }
    }
</script>
</body>
</html>
```

8. Chạy đoạn mã bằng cách chỉ đến *location2.htm*?

name=Steve&country=US trong trình duyệt. Lúc này, bạn nhận được một trang danh sách các thuộc tính đã gấp trước đó và thêm các giá trị khác được lấy về từ chuỗi truy vấn như sau:



9. Chú ý rằng tham số đầu tiên, `name`, có dấu hỏi chấm (?) trong chuỗi truy vấn và chúng ta không muốn điều này. Có một số cách để giải quyết vấn đề này. Cách đơn giản nhất là sử dụng phương thức `substring()`. Thay đổi biến `querystring` bằng dòng lệnh sau:

```
var querystring= location.search.substring(1);
```

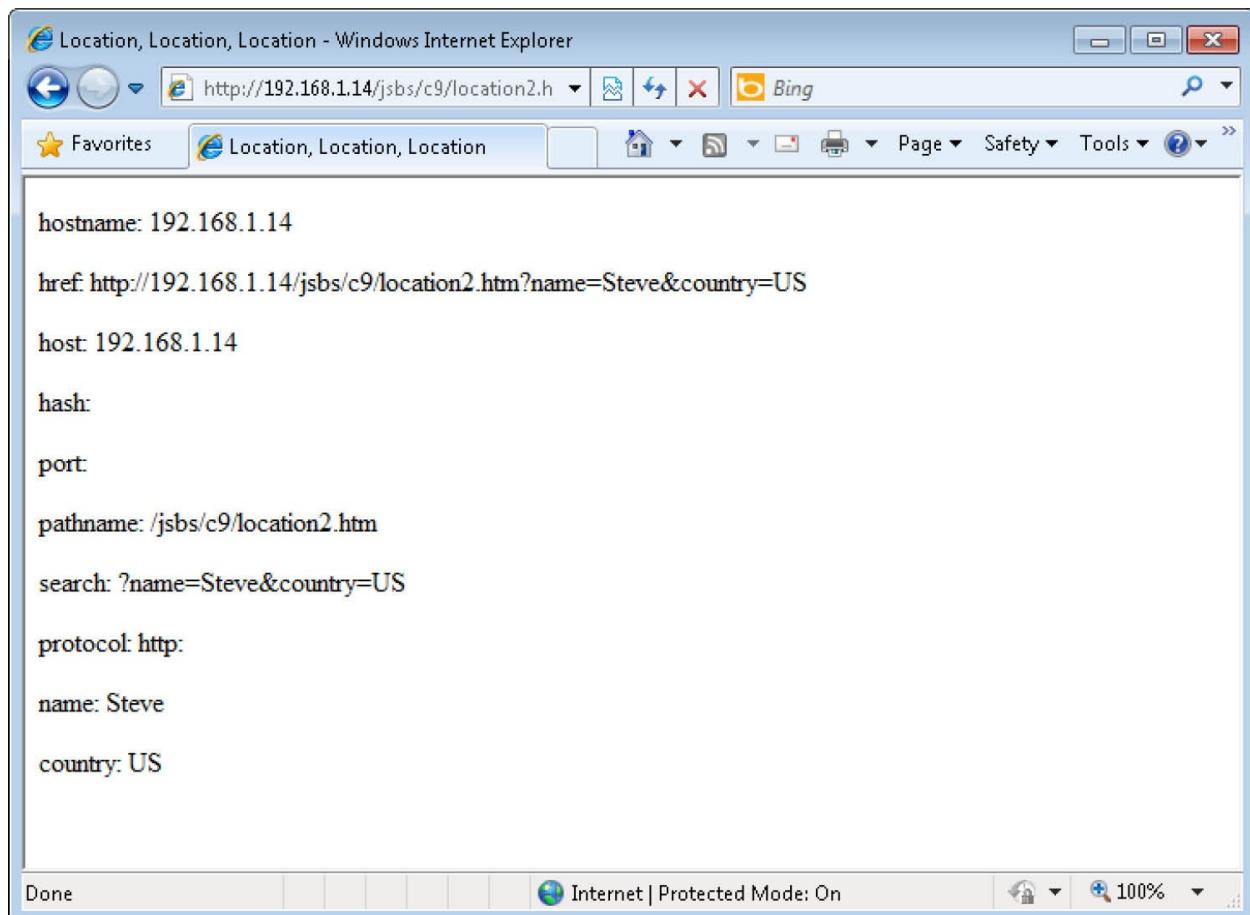
Phương thức `substring()` trả về một chuỗi bắt đầu từ vị trí đã chọn. Trong trường hợp này, ký tự đầu tiên của `location.search` (tại chỉ số 0) là dấu hỏi; do vậy sử dụng `substring()` tại chỉ số 1 sẽ giải quyết vấn đề. Đoạn mã cuối cùng (với thay đổi là phần in đậm) giống như dưới đây. (Bạn có thể tìm thấy đoạn mã này trong file location3.txt của Tài nguyên đi kèm).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Location, Location, Location</title>
</head>
```

```
<body>
    <script type="text/javascript">
        var body= document.getElementsByTagName('body');
        for (var prop in location) {
            // tạo phần tử
            var elem = document.createElement(prop);
            //tạo nút văn bản
            var text =
                document.createTextNode(prop);
            elem.appendChild(text); //thêm nó vào phần tử
            body.appendChild(elem);
        }
        if(location.search){
            var querystring= location.search;
            var splits= querystring.split('&');
            for (var i= 0; i<splits.length; i++) {
                var splitpair= splits[i].split('=');
                var elem= document.createElement(splitpair[0]);
                var text= document.createTextNode(splitpair[1]);
                elem.appendChild(text);
                body.appendChild(elem);
            }
        }
    </script>
</body>
</html>
```



10. Lưu đoạn mã này thành file **location3.htm** và mở lại. Nhìn vào kết quả, chúng ta sẽ thấy dấu hỏi chấm đã biến mất.



Vị trí này cũng có thể được thiết lập bằng cách sử dụng đối tượng *location* của JavaScript. Thông thường, chúng ta sử dụng phương thức *assign()* của đối tượng *location*. Ví dụ, để chuyển hướng tới một trang web khác, có thể dùng đoạn mã sau:

```
location.assign("http://www.braingia.org");
```

Gọi phương thức *assign()* cũng giống như việc thiết lập giá trị cho thuộc tính *href*:

```
location.href= "http://www.braingia.org";
```

Bạn cũng có thể thay đổi các thuộc tính khác của đối tượng *location* như *công*, chuỗi truy vấn hay đường dẫn. Ví dụ, để lập đường dẫn đến */blog/*, bạn làm như sau:

```
location.pathname= "blog";
```

Đổi chuỗi truy vấn sang *?name=Steve* làm như sau:

```
location.search= "?name=Steve";
```

Gọi phương thức `reload()` để tải lại trang web.

```
location.reload();
```

Khi bạn gọi `location.reload()`, trình duyệt sẽ tải lại trang web từ bộ lưu trữ tạm thời (cache) chứ không tạo một yêu cầu đến máy chủ. Tuy vậy, nếu truyền vào một giá trị Boolean `true` vào hàm, trình duyệt sẽ nạp lại trang từ máy chủ:

```
location.reload(true);
```

Chú ý Hãy thận trọng khi sử dụng phương thức `reload()`. Việc cố gắng tải lại trang trong mã, trái ngược với việc tải lại dựa vào hàm bẫy sự kiện (hay hàm lắng nghe sự kiện) rất dễ dẫn đến chu trình lặp liên tục.

Đối tượng *history*

Đối tượng *history* cung cấp phương thức để di chuyển đến các trang trước hay sau trang hiện tại trong lịch sử truy cập. (Tuy nhiên, vì lý do bảo mật, JavaScript không thể truy cập URI của những site mà trình duyệt đã ghé thăm. Cụ thể, bạn có thể sử dụng phương thức `back()`, `forward()` và `go()`. Phương thức `back()` trả về trang trước đó còn `forward()` trả về trang kế tiếp. Phương thức `go()` di chuyển đến giá trị chỉ số (chính là đối số của phương thức).

Chú ý Nếu ứng dụng không đi đến những trang khác hay vị trí khác trong thanh địa chỉ, bộ nhớ trình duyệt sẽ không lưu những trang này và không thể dùng với những phương thức bên trên.

Dưới đây là đoạn mã nguồn mẫu thực hiện thao tác quay trở lại trang web trước đó và tiến đến trang web sau đó. Những ví dụ trong các chương sau sẽ mô tả chi tiết làm thế nào để áp dụng những đoạn mã này trong thực tế.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>
```

```
<head>
    <title>Đối tượng history</title>
    <script type="text/javascript">
        function moveBack(){//Chuyển đến trang trước
            history.back();
        }
        function moveForward(){ //Chuyển đến trang sau
            history.forward();
        }
    </script>
</head>
<body>
<p><a href="#" onclick="moveBack()">Nhấn vào để trở về trang trước</a>
<p><a href="#" onclick="moveForward()">Nhấn vào để tới trang sau</a>
</body>
</html>
```

Chú ý Đoạn mã trên sử dụng hàm xử lý sự kiện tại chỗ (*onclick*), khuyến cáo không nên dùng hàm này cho kiểu viết mã JavaScript tiện lợi vì các hàm xử lý sự kiện sẽ chèn hành vi vào mã HTML. Cách sử dụng *onclick* ở đây chỉ có mục đích minh họa để tránh việc giới thiệu khái niệm hàm xử lý sự kiện sẽ được bàn đến trong Chương 11.

Khai báo hàm xử lý

HTML 5.0 giới thiệu hai phương thức mới của đối tượng *navigator*: *registerContentHandler()* và *registerProtocolHandler()*. Sử dụng những phương thức này, website có thể đăng ký URI để xử lý một số loại thông tin nhất định như là RSS feed. Tuy nhiên, vì các phương pháp này chưa được hỗ trợ rộng rãi, nên chúng ta sẽ không đề cập đến trong cuốn sách này.

Bài tập

1. Sử dụng phương thức *availHeight* và *availWidth* để xác định liệu màn

hình có chiều cao tối thiểu là 768 pixel và rộng tối thiểu là 1024 pixel không. Nếu không, hiển thị hộp thoại thông báo kích cỡ của màn hình.

2. Sửa bài tập đã được trình bày trong nội dung chương, sử dụng đối tượng *location* để hiển thị hộp thoại *alert()* dựa vào giá trị của chuỗi truy vấn. Cụ thể, hiển thị từ “Obrigado” nếu nước được nhắc đến là Brazil, hiển thị “Thank you” nếu là nước Anh (Great Britain). Kiểm tra những điều kiện đó.
3. Cài đặt add-on User Agent Switcher lên Firefox hay add-on tương tự lên Internet Explorer. Tiếp đó sử dụng đoạn mã trong bài tập “Tìm hiểu đối tượng *navigator*” trước đó và thử nghiệm với nhiều giá trị khác nhau. Bài tập này giúp bạn hiểu tại sao không nên chỉ sử dụng đối tượng *navigator* để xác định độ tương thích của trình duyệt. Chú ý: Bạn có thể tự chọn trình duyệt.



Phần 3

Tích hợp JavaScript vào thiết kế trang

[Chương 10: Mô hình đối tượng tài liệu](#)

[Chương 11: Các sự kiện trong JavaScript và làm việc với trình duyệt](#)

[Chương 12: Tạo và sử dụng cookie](#)

[Chương 13: Làm việc với hình ảnh trong JavaScript](#)

[Chương 14: Sử dụng JavaScript với Web Form](#)

[Chương 15: JavaScript và CSS](#)

[Chương 16: Xử lý lỗi trong JavaScript](#)



Chương 10

Mô hình đối tượng tài liệu

Sau khi đọc xong chương này, bạn có thể:

- Sử dụng Mô hình đối tượng tài liệu (DOM) để truy xuất các phần tử của một tài liệu.
- Tạo thêm các phần tử mới trong một tài liệu
- Thay đổi các phần tử trong một tài liệu.
- Xóa các phần tử trong một tài liệu.

Mô hình đối tượng tài liệu

DOM hay Mô hình đối tượng tài liệu là một chuẩn được định nghĩa bởi Tổ chức Web Toàn Cầu (World Wide Web Consortium - W3C). DOM cung cấp cách thức để truy cập và thay đổi nội dung của các tài liệu HTML. Phần lớn các trình duyệt web hiện nay đều triển khai DOM dưới nhiều hình thức và mức độ khác nhau.

Cũng như rất nhiều chuẩn khác, đặc biệt các chuẩn liên quan đến lập trình web, DOM được phát triển và cải tiến qua nhiều năm. Hiện có ba đặc tả DOM, được gọi là các *cấp độ* của DOM, đặc tả thứ tư hiện đang được phát triển.

Cần nhấn mạnh rằng DOM mạnh hơn tất cả những gì sẽ được trình bày trong chương này hay thậm chí trong cuốn sách này. Bạn có thể dùng DOM cho nhiều mục đích khác ngoài việc lập trình JavaScript. Cuốn sách này chỉ tập trung vào việc bạn sẽ dùng JavaScript như thế nào để truy cập và xử lý DOM.

Khi nhắc tới DOM, tôi sẽ tập trung vào mối quan hệ của nó với công việc hiện tại chứ không bàn rộng ra các khái niệm liên quan trong DOM. Chẳng hạn, cuốn sách này sẽ tập trung trình bày cách DOM sắp xếp các tài liệu HTML dưới dạng cây. Tất nhiên, DOM cũng xử lý các tài liệu XML theo cách tương tự, nhưng vì đây là cuốn sách về JavaScript nên trước tiên chúng ta sẽ chú tâm vào mối quan hệ giữa DOM và HTML.

Để tìm thêm thông tin về DOM, hãy đọc kỹ đặc tả DOM trên website của W3C tại địa chỉ: <http://www.w3.org/DOM/>.

Chú ý Ví dụ trong chương này sử dụng các hàm xử lý sự kiện tại chỗ (hay còn gọi là hàm xử lý sự kiện nội tuyến - hàm viết ngay trong thẻ html) như hàm `onload` được gắn vào thẻ `<body>` và hàm `onclick` được gắn vào rất nhiều thẻ HTML khác. Việc sử dụng các hàm tại chỗ không phải là một thói quen tốt và chỉ được dùng với mục đích minh họa. Chương 11 "Các sự kiện trong JavaScript và làm việc với trình duyệt" sẽ giới thiệu cách tốt hơn để gắn các sự kiện vào mã HTML.

DOM cấp độ 0: DOM Sơ khai

DOM cấp độ 0 được triển khai trước khi tất cả các đặc tả chính thức khác về DOM ra đời. Sau khi DOM cấp độ 1 được định nghĩa, các công nghệ liên quan đến việc viết mã kịch bản cho tài liệu trước đây được hệ thống hóa thành DOM cấp độ 0 (dù khi đó chưa có tổ chức về chuẩn nào chính thức công nhận điều này). Hiện nay, tất cả các trình duyệt phổ biến vẫn hỗ trợ các thành phần của DOM cấp độ 0 nhằm đảm bảo khả năng tương thích ngược. Tuy nhiên, bạn không bao giờ muốn những đoạn mã viết từ năm 1998 không chạy được trên trình duyệt đương thời.

DOM cấp độ 0 tập trung chủ yếu vào việc truy cập đến các phần tử trên form, ngoài ra nó cũng hỗ trợ truy cập vào các liên kết và hình ảnh. Chương 14 “Sử dụng JavaScript với Web Form” sẽ nói về form và cách truy cập chúng bằng DOM. Ở đây, thay vì dành thời gian cho các ví dụ về DOM cấp độ 0, tôi sẽ tập trung hơn vào DOM cấp độ 1 và 2 bởi bạn sẽ chủ yếu sử dụng chúng khi lập trình với JavaScript.

DOM cấp độ 1 và 2



Tổ chức W3C ra mắt đặc tả cấp độ 1 của DOM vào năm 1998. Cũng như DOM cấp độ 0, DOM cấp độ 1 được hỗ trợ dưới nhiều hình thức khác nhau bởi tất cả các trình duyệt chính. DOM cấp độ 2 được giới thiệu chính thức vào năm 2000. Việc hỗ trợ cho DOM cấp độ 2 tương đối khác nhau giữa các trình duyệt. Có thể nói, các phiên bản của các trình duyệt đều hỗ trợ DOM ở mức độ khác nhau.

Các phiên bản của IE trước phiên bản 9 đều công bố hỗ trợ DOM, nhưng theo cách khác với những trình duyệt khác. Kết quả là, bạn phải luôn ghi nhớ tính năng DOM mà bạn đang sử dụng trong mã JavaScript có thể không hoạt động trên IE hoặc dùng được trên IE nhưng lại không hoạt động trên các trình duyệt khác. IE phiên bản 9 trên Windows là một bước đi đúng hướng, nhưng bạn vẫn phải để ý đến vấn đề tương thích giữa các trình duyệt. Tôi sẽ cố gắng chỉ ra những khác biệt trong cách thực thi DOM của các trình duyệt và cách xử lý vấn đề.

DOM và cấu trúc cây

DOM trình bày tài liệu HTML theo cấu trúc cây ngược. Xem xét tài liệu HTML

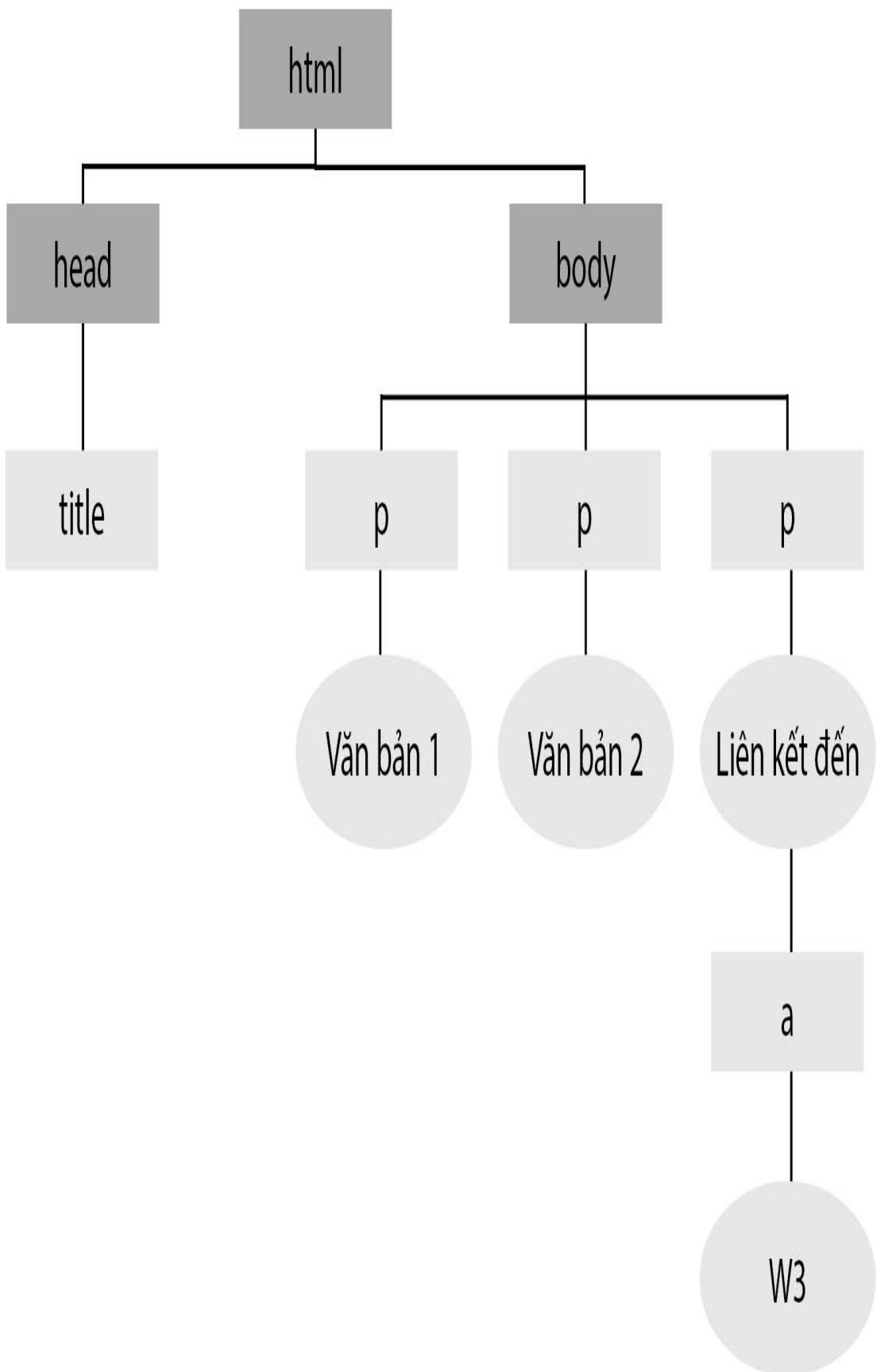
đơn giản trong Ví dụ 10-1.

VÍ DỤ 10-1 Tài liệu HTML đơn giản

```
<html>
<head>
<title>Hello world </title>
</head>
<body>
<p>Văn bản 1</p>
<P>Văn bản 2</P>
<p>Liên kết đến <a href="http://www.w3.org">w3</a></p>
</body>
</html>
```

Hình 10-1 trình bày tài liệu HTML trong Ví dụ 10-1 theo cấu trúc cây ngược của DOM.





Hình 10-1 Tài liệu đơn giản được trình bày dưới dạng cấu trúc cây.

Nhiều phần tử HTML có thể có các thuộc tính, chẳng hạn trong ví dụ 10.1, phần tử `<a>` có thuộc tính `href`. Trong phần sau của chương này, bạn sẽ được học cách truy xuất và thiết lập giá trị cho các thuộc tính này bằng DOM.

Khi làm việc với DOM, bạn cần phân biệt rõ ràng việc truy xuất phần tử, thiết lập giá trị cho phần tử và xóa các phần tử. Phương thức làm việc với các phần tử của DOM sẽ phản ánh sự khác biệt này.

Làm việc với các nút

Các phần tử trên cấu trúc cây đôi khi được gọi là các *nút* hay *các đối tượng nút*. Các nút bên dưới một nút khác được gọi là *nút con*. Chẳng hạn, trên Hình 10-1, nút `body` có ba nút con, đều là các phần tử `p` và một trong số các nút `p` có một nút con `a`. Nút `body` được gọi là nút cha của các nút `p`. Bất cứ nút nào nằm dưới một nút nhất định đều được gọi là con của nút đó. Ba nút `p` trong Hình 10-1 được gọi là anh em vì chúng cùng cấp bậc.

Tương tự như cách làm việc với các phần tử của DOM, bạn cũng dùng phương thức để làm việc với các nút. Chẳng hạn, bạn dùng `appendChild()` để thêm nút vào một nút cha có sẵn (xem ví dụ ở phần sau của chương này).

Truy xuất các phần tử

Truy xuất các phần tử của tài liệu là một trong những thao tác cơ bản khi sử dụng DOM trong lập trình với JavaScript. Trong phần này, chúng ta sẽ xem xét hai trong các phương thức cơ bản được dùng để truy xuất các phần tử: `getElementById()` và `getElementsByName()`.

Truy xuất phần tử bằng ID

Phương thức `getElementById()` là một trong những phương thức cơ bản của DOM, truy xuất các phần tử xác định của một tài liệu HTML và trả về tham chiếu đến phần tử đó. Phần tử phải có thuộc tính `id` để được truy xuất bằng phương thức này. Chẳng hạn, bạn có thể thêm thuộc tính `id` cho phần tử `a` trong

đoạn mã HTML ở Ví dụ 10-1 như phần in đậm dưới đây:

```
<html>
<head>
<title>Hello world </title>
</head>
<body>
<p>Văn bản 1</p>
<p>Văn bản 2</p>
<p>Liên kết đến <a id="w3link" href="http://www.w3.org">W3</a>
</body>
</html>
```

Giờ thì *a* đã có *id* và có thể được truy xuất nó bằng phương thức *getElementById()* như sau:

```
var a1 = document.getElementById("w3link");
```

Tham chiếu đến phần tử có ID là *w3link* sẽ được đặt trong biến *a1* của đoạn mã JavaScript.

Tất cả các phần tử HTML đều hỗ trợ thuộc tính *id*, nhờ đó JavaScript sẽ luôn hỗ trợ truy xuất các phần tử này. Trong ví dụ tiếp theo, tất cả các phần tử *p* đều có ID nên chúng ta có thể truy xuất chúng bằng phương thức *getElementById()*.

```
<html>
<head>
<title>Hello world</title>
<body>
<p id="sometext">Văn bản 1</p>
<p id="moretext">Văn bản 2</p>
<p id="linkp">Liên kết đến <a id="w3link" href="http://www.w3.org">W3</a>
</body>
</html>
```

Các phần tử *<p>* được truy xuất như sau:

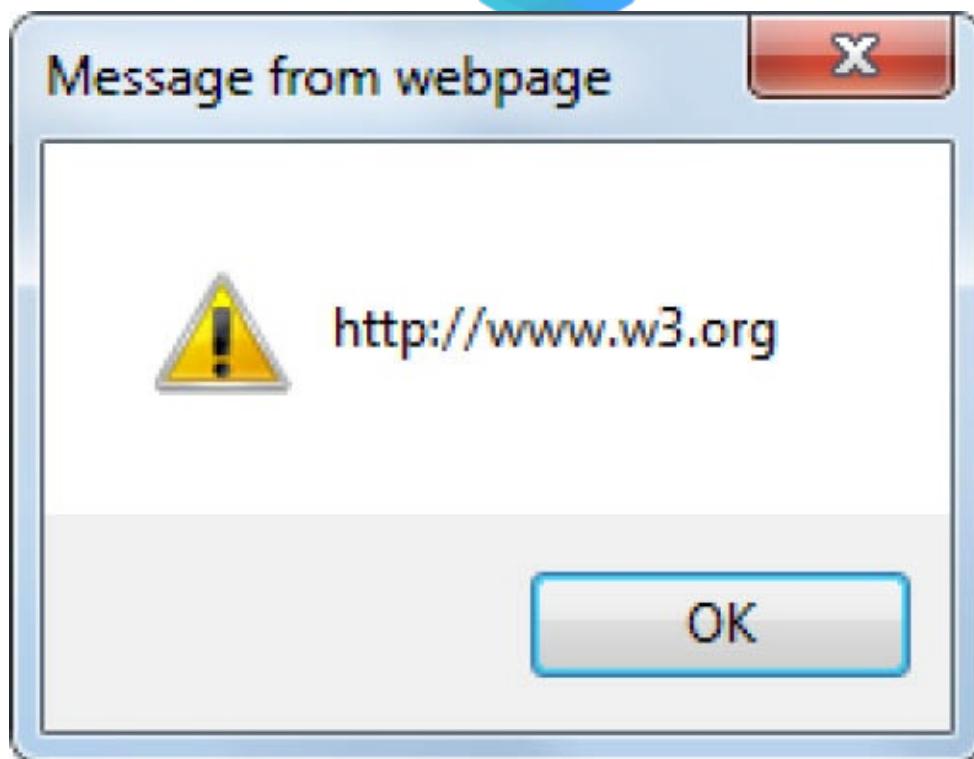
```
var p1 = document.getElementById("sometext");
var p2 = document.getElementById("moretext");
var plink = document.getElementById("linkp");
```

Bạn có thể làm gì với các phần tử này sau khi truy xuất? Với các phần tử như *a*, bạn có thể truy cập các thuộc tính của nó bằng cách lấy giá trị của thuộc tính

href, như sau: Đoạn mã này cũng có thể được tìm thấy trong file getelement.htm của phần Tài nguyên đi kèm.

```
<html>
<head>
    <title>Truy xuất bằng ID</title>
    <script type="text/javascript">
        function checkhref() {
            var a1 = document.getElementById("w3link");
            alert(a1.href);
        }
    </script>
</head>
<body onload="checkhref()">
<p id="sometext">Văn bản 1</p>
<p id="moretext">Văn bản 2</p>
<p id="linkp">Liên kết đến <a id="w3link" href="http://www.w3.org/"></a></p>
</body>
</html>
```

Trang html chứa đoạn mã này sẽ hiển thị một hộp thoại thông báo thuộc tính *href* của phần tử *a* như trong Hình 10-2.



Hình 10-2 Thuộc tính *href* được truy xuất nhờ phương thức *getElementById()*

Phần sau của chương này sẽ trình bày thêm về cách thay đổi các phần tử và các thuộc tính.

Chú ý về thuộc tính *innerHTML*

Một trong số cách thay đổi văn bản của phần tử là sử dụng thuộc tính *innerHTML*. Thuộc tính *innerHTML* cho phép truy cập văn bản của những phần tử như *p* một cách nhanh chóng và dễ dàng. Trên thực tế, dù nhiều người có thể không thích thuộc tính này song họ vẫn phải dùng nó vì nó thực sự hoạt động rất hiệu quả.

Vấn đề duy nhất với *innerHTML* là nó không được W3C định nghĩa một cách chính thức vì vậy nó không được hỗ trợ trên tất cả các trình duyệt như các đối tượng cụ thể khác trong DOM. Tuy nhiên, *innerHTML* sẽ sớm xuất hiện trong đặc tả HTML 5.0 và với việc thực thi khó lường trong các cấp DOM hiện tại, một đặc tả DOM chính thức cho thuộc tính này sẽ luôn cần thiết. Phần lớn các trình duyệt đều hỗ trợ *innerHTML* – theo cách khá thống nhất.

Hãy xem xét ví dụ sau (bạn cũng có thể thấy nó trong file innerhtml.htm của Tài nguyên đi kèm).

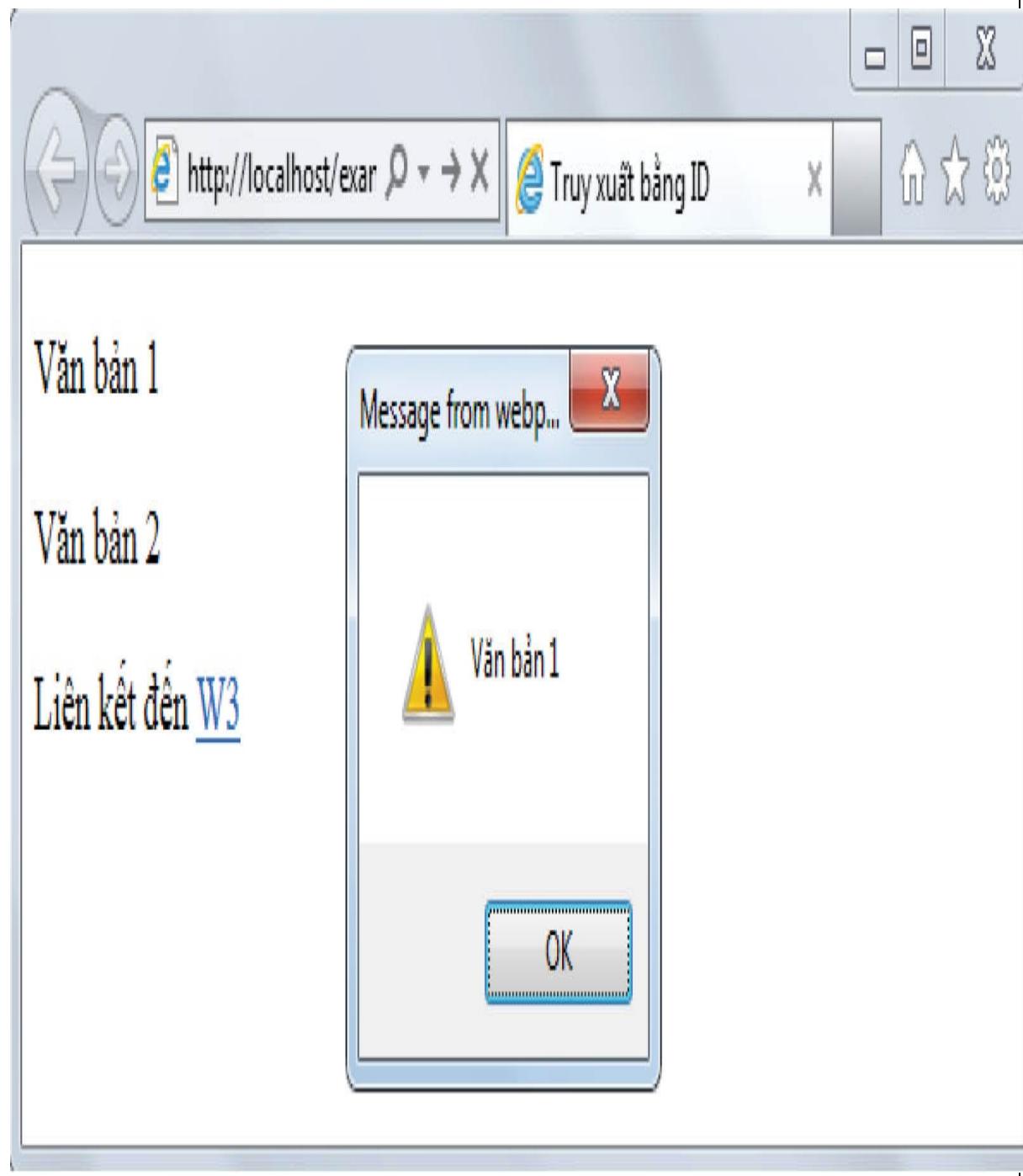


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Truy xuất bằng ID</title>
    <script type="text/javascript">
        function changetext() {
            var p1 = document.getElementById("sometext");
            alert(p1.innerHTML);
            p1.innerHTML = "Văn bản 1 đã thay đổi.";
        }
    </script>
</head>
<body onload="changetext()">
<p id="sometext">Văn bản 1</p>
<p id="moretext">Văn bản 2</p>
<p id="linkp">Liên kết đến <a id="w3link" href="http://www.
</body>
</html>
```

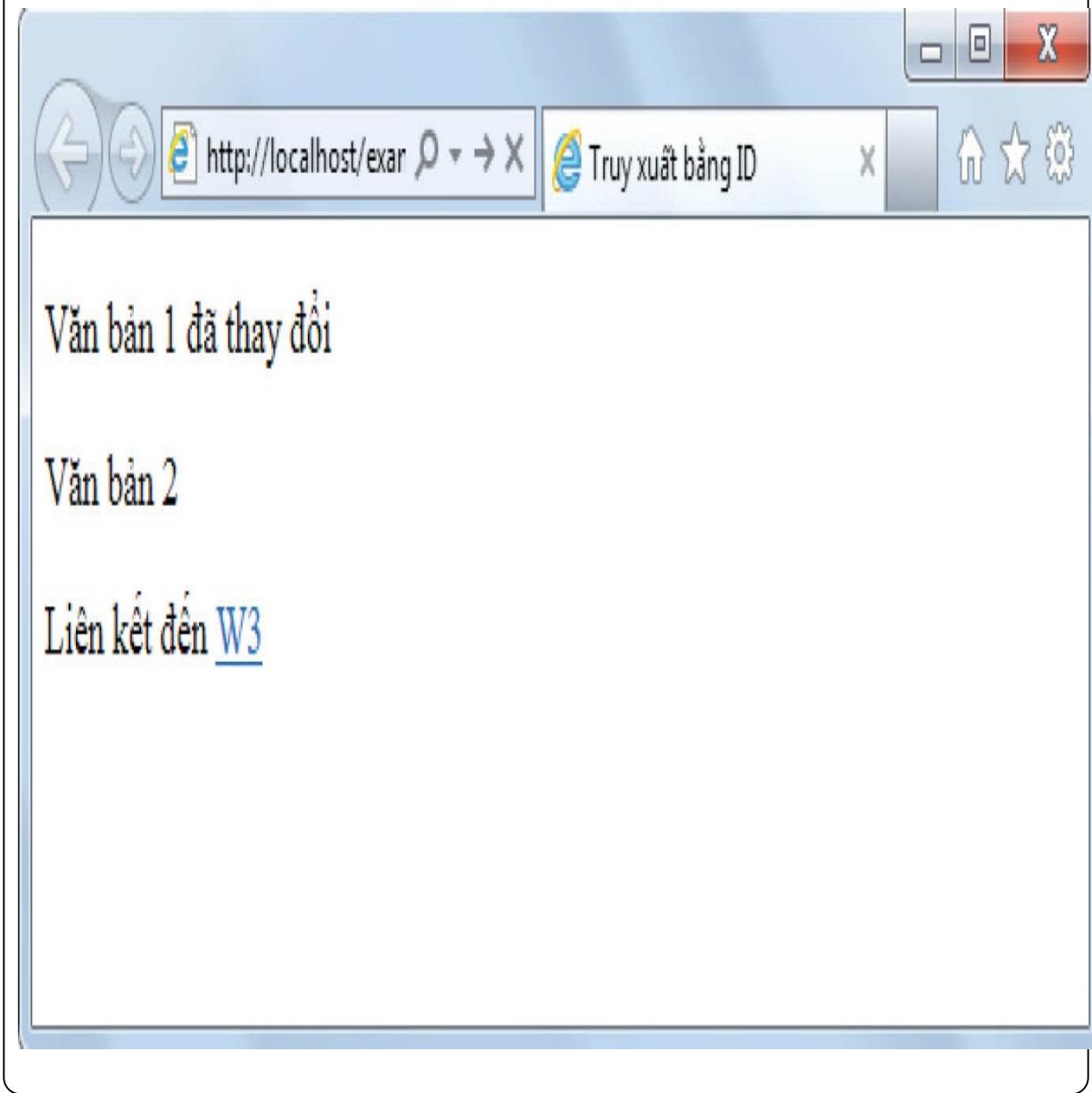
Hàm `changetext()` truy xuất phần tử với ID là `sometext`, sau đó gán tham chiếu đến phần tử này cho biến `p1`, sử dụng câu lệnh sau:

```
var p1 = document.getElementById("sometext");
```

Sau đó, thuộc tính `innerHTML` được gọi, kết quả là một hộp thoại `alert()`.



Chú ý hộp thoại `alert()` và dòng đầu tiên trên cửa sổ nền: Khi người dùng nhấn OK, hộp thoại `alert()` sẽ biến mất, dòng lệnh JavaScript tiếp theo được thực thi, sử dụng thuộc tính `innerHTML` để thay đổi văn bản của phần tử `p` đầu tiên thành “Văn bản 1 đã thay đổi”. Kết quả được hiển thị như sau:



Truy xuất bằng tên thẻ HTML

Phương thức `getElementById()` rất hữu dụng khi bạn chỉ cần truy xuất một hoặc vài phần tử. Tuy nhiên, nếu bạn muốn truy xuất nhiều phần tử cùng lúc thì

dùng phương thức `getElementsByName()` sẽ tốt hơn nhiều.

Phương thức `getElementsByName()` trả về một mảng hoặc danh sách tất cả các phần tử có tên thẻ trùng với tên thẻ trong tham số. Ví dụ, để truy xuất tất cả các hình ảnh (thẻ ``) trong một tài liệu, bạn viết đoạn mã sau:

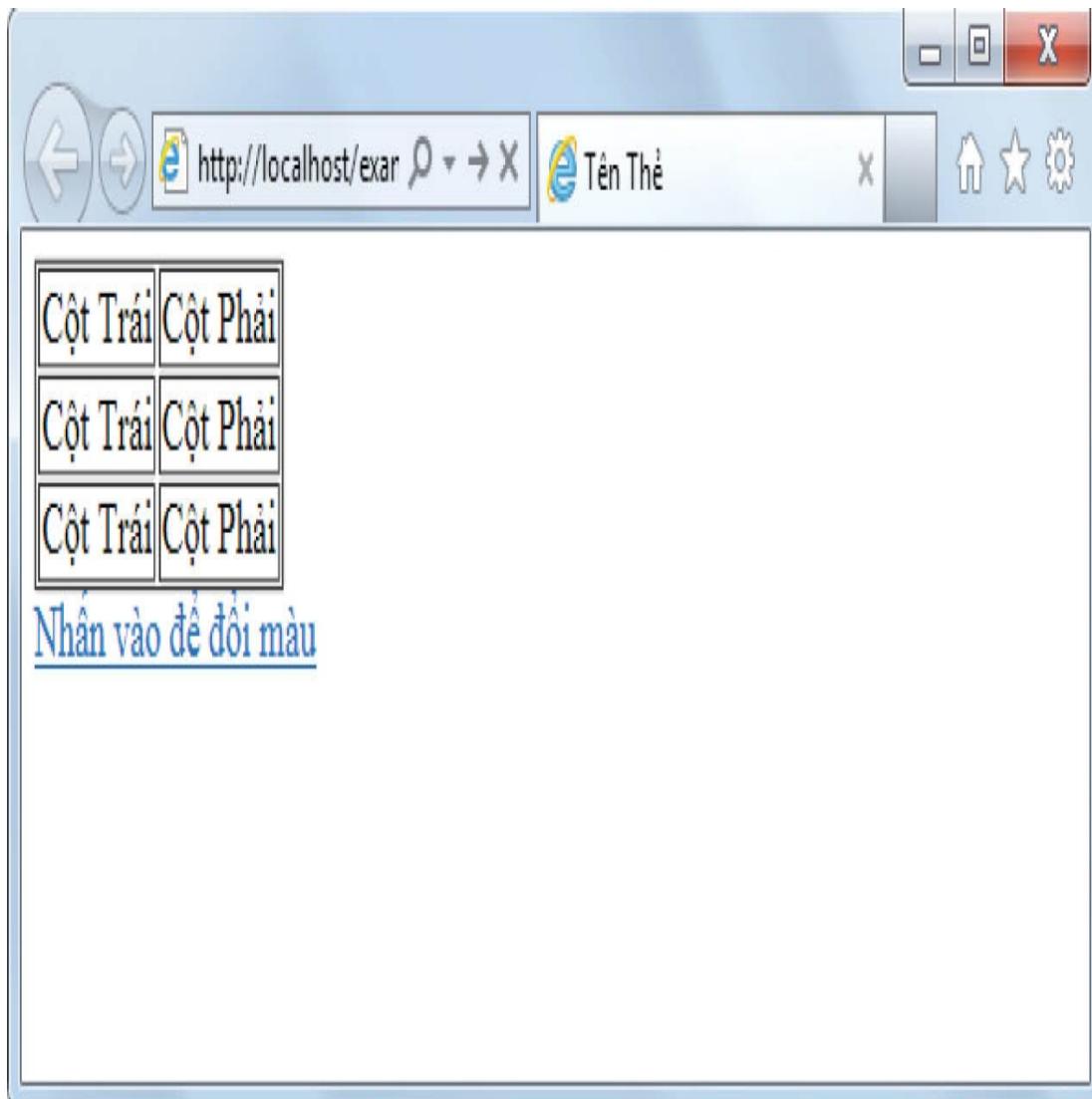
```
var images = document.getElementsByTagName("img");
```

Tiếp theo, bạn có thể xem xét tất cả các thuộc tính của phần tử `img` chứa trong biến `images` bằng cách duyệt qua tất cả phần tử.

Sau đây là ví dụ về việc chỉnh sửa một bảng. Đoạn mã này đổi màu nền của các phần tử `td` bên trong bảng khi người dùng nhấp vào liên kết Nhấn vào để đổi màu. Bạn có thể tìm thấy đoạn mã trong file `getbytagname.htm` của Tài nguyên đi kèm.

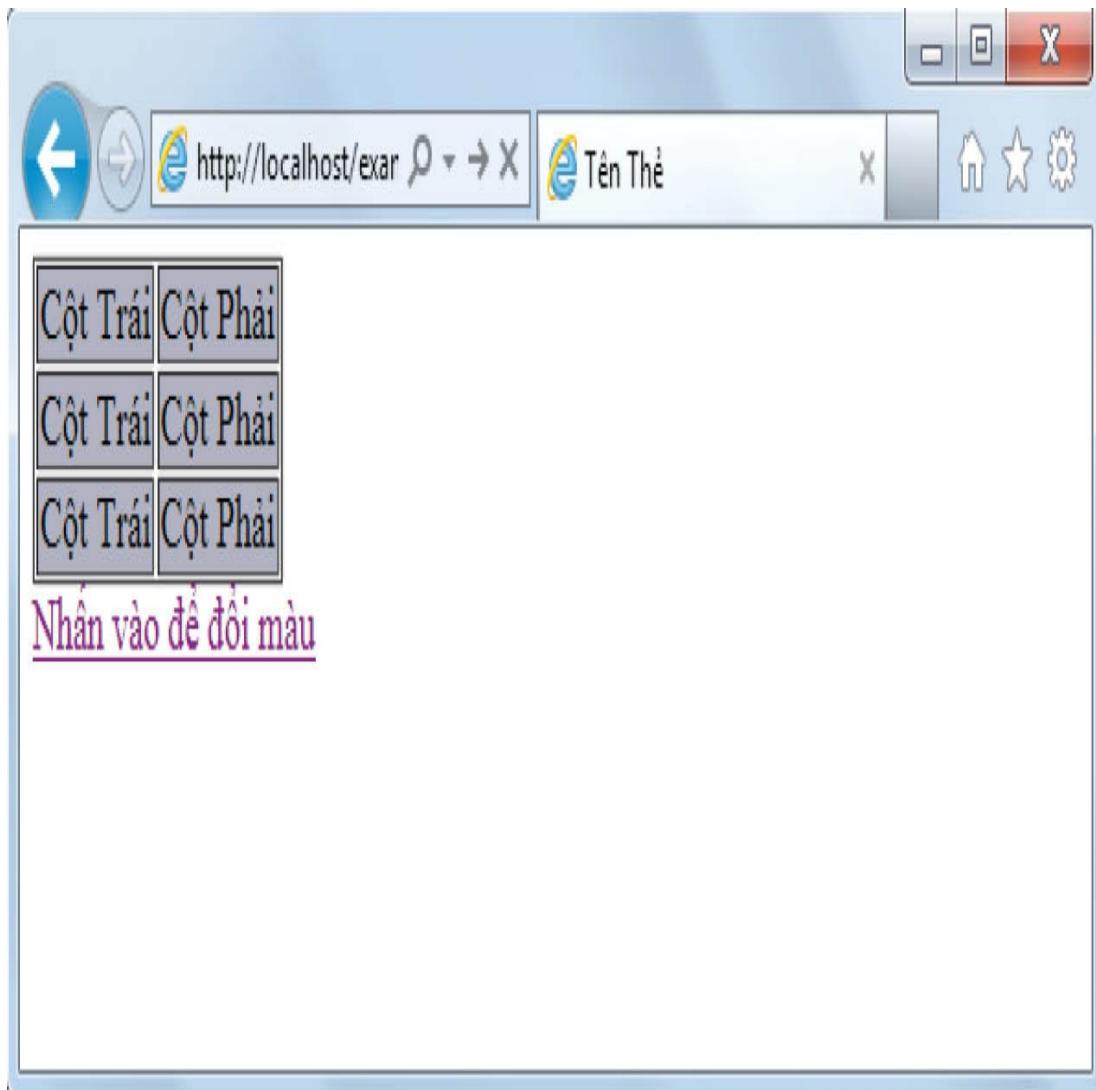
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Tên thẻ</title>
    <script type="text/javascript">
        function changecolors() { //thay đổi màu
            var a1 = document.getElementsByTagName("td")
            var a1Length = a1.length; // Truy xuất chiều
            for (var i = 0; i < a1Length; i++) {
                a1[i].style.background = "#aaabba";
            }
        }
    </script>
</head>
<body>
<table id="mytable" border="1">
<tr><td id="lefttd0">Cột Trái</td><td id="righttd0">Cột Phải</td>
<tr><td id="lefttd1">Cột Trái</td><td id="righttd1">Cột Phải</td>
<tr><td id="lefttd2">Cột Trái</td><td id="righttd2">Cột Phải</td>
</table>
<a href="#" onclick="return changecolors();">Nhấn vào để đổi</a>
</body>
</html>
```

Hình 10-3 là giao diện của trang web khi được mở trên trình duyệt.



Hình 10-3 Sử dụng phương thức `getElementsByTagName()` để định dạng các phần tử trong một bảng.

Khi người dùng nhấn vào liên kết, màu nền của các ô trong bảng sẽ thay đổi như trong Hình 10-4.



Hình 10-4 Sau khi người dùng nhấn vào liên kết, màu nền của các ô trong bảng thay đổi.

Xem xét đoạn mã, bạn sẽ thấy ở vị trí `<head>` của trang web có đoạn mã JavaScript tạo hàm tên là `changecolors()` (thay đổi màu).

```
function changecolors() {
```

Hàm này truy xuất tất cả các phần tử `td` bằng cách sử dụng phương thức `getElementsByName()`, rồi đặt chúng vào mảng `a1`:

```
var a1 = document.getElementsByTagName("td");
```

Tiếp đó, sử dụng vòng lặp `for` duyệt qua tất cả các phần tử của mảng. Đoạn mã trên dùng biến `a1Length` biến này chứa độ dài của mảng `a1` trong câu lệnh gán ngay bên trên vòng `for`.

Bên trong vòng *for*, từng dòng lệnh thay đổi màu nền của mỗi phần tử thành `#aaabba`, gam màu xanh lam. Thông thường, bạn nên sử dụng CSS để thay đổi định dạng cho các phần tử HTML thay vì thay đổi từng thuộc tính một như trong ví dụ trên. Song, trước khi bạn học về CSS và JavaScript trong Chương 15 “JavaScript và CSS”, cách vừa rồi là đủ.

```
for (var i = 0; i < a1Length; i++) {  
    a1[i].style.background = "#aaabba";  
}
```

Liên kết gọi đến hàm *changecolors()* nhờ sự kiện *onclick*:

```
<a href="#" onclick="return changecolors();">Nhấn vào để đổi
```

Chú ý Sự kiện *onclick*, *onload* và các sự kiện khác được mô tả đầy đủ trong Chương 11.

Sau ví dụ này, nhiều người sẽ đặt câu hỏi làm thế nào để thay đổi màu cho tất cả các dòng khác trong bảng. Bạn có thể làm việc đó một cách dễ dàng bằng cách sử dụng JavaScript và CSS theo hướng dẫn trong Chương 15.

Các tập hợp (collection) trong HTML

Có nhiều đối tượng chứa một nhóm các phần tử trong một tài liệu. Các đối tượng này bao gồm:

- ***document.anchors*** Nhóm chứa tất cả các phần tử có tên là `<a>`.
- ***document.forms*** Nhóm chứa tất cả phần tử có tên là `<form>`.
- ***document.images*** Nhóm chứa tất cả phần tử có tên là ``.
- ***document.links*** Nhóm chứa tất cả phần tử có tên là `<a>` và có chứa thuộc tính *href*.

Làm việc với các phần tử anh em

JavaScript chứa các phương thức và thuộc tính để xử lý mối quan hệ cha/con, anh/em trong một tài liệu HTML. Chẳng hạn, thuộc tính *childNodes* trả về

nhóm các nút con của một phần tử nhất định. Nhóm này tương tự mảng, mặc dù nó không thuộc kiểu *Array* trong JavaScript. Ví dụ, một phần tử *<div>* với ID là *mydiv* có rất nhiều con là phần tử *<a>*. Dòng lệnh sau truy xuất phần tử con đầu tiên của *mydiv* và đặt nó vào biến *childOne*:

```
var childOne = document.getElementById("mydiv").childNodes[0]
```

Mỗi nút cha có một hoặc nhiều nút con, nhưng mỗi nút con chỉ có một nút cha được truy xuất bằng thuộc tính *parentNode*. Bạn có thể duyệt qua nhóm các nút con bằng cách sử dụng thuộc tính *nextSibling* và *previousSibling*. Nếu không còn phần tử anh em nữa, các thuộc tính sẽ trả về giá trị *null*. Ví dụ, thuộc tính *previousSibling* sẽ trả về giá trị *null* khi sử dụng trên nút con đầu tiên và thuộc tính *nextSibling* cũng sẽ trả về *null* khi sử dụng trên nút con cuối cùng.

Các thuộc tính *firstChild* và *lastChild* lần lượt chứa nút con đầu tiên (*childNodes[0]*) và cuối cùng của một phần tử. Khi một phần tử không chứa bất kỳ nút con nào, các thuộc tính đều trả về *null*.

Làm việc với các thuộc tính

Các thuộc tính của phần tử đều có thể được truy xuất và gán giá trị thông qua JavaScript. Trong phần này, chúng ta sẽ tìm hiểu về cả hai thao tác trên.

Xem thuộc tính

Khi mới bắt đầu lập trình JavaScript, có thể bạn không biết một phần tử có những thuộc tính gì. Tuy nhiên, đừng lo lắng về điều này vì chúng ta có thể dùng một vòng lặp để gọi phương thức *getAttribute()* (lấy các thuộc tính). Hàm sau đây sẽ hiển thị tất cả các thuộc tính của một phần tử cho trước:

```
function showattribs(e) {
    var e = document.getElementById("braingialink");
    var elemList = ""; //khai báo và khởi tạo danh sách
    for (var element in e) {
        var attrib = e.getAttribute(element);
        elemList = elemList + element + ": " + attrib
    }
    alert(elemList)
}
```

```
        }
        alert(elemList);
    }
```

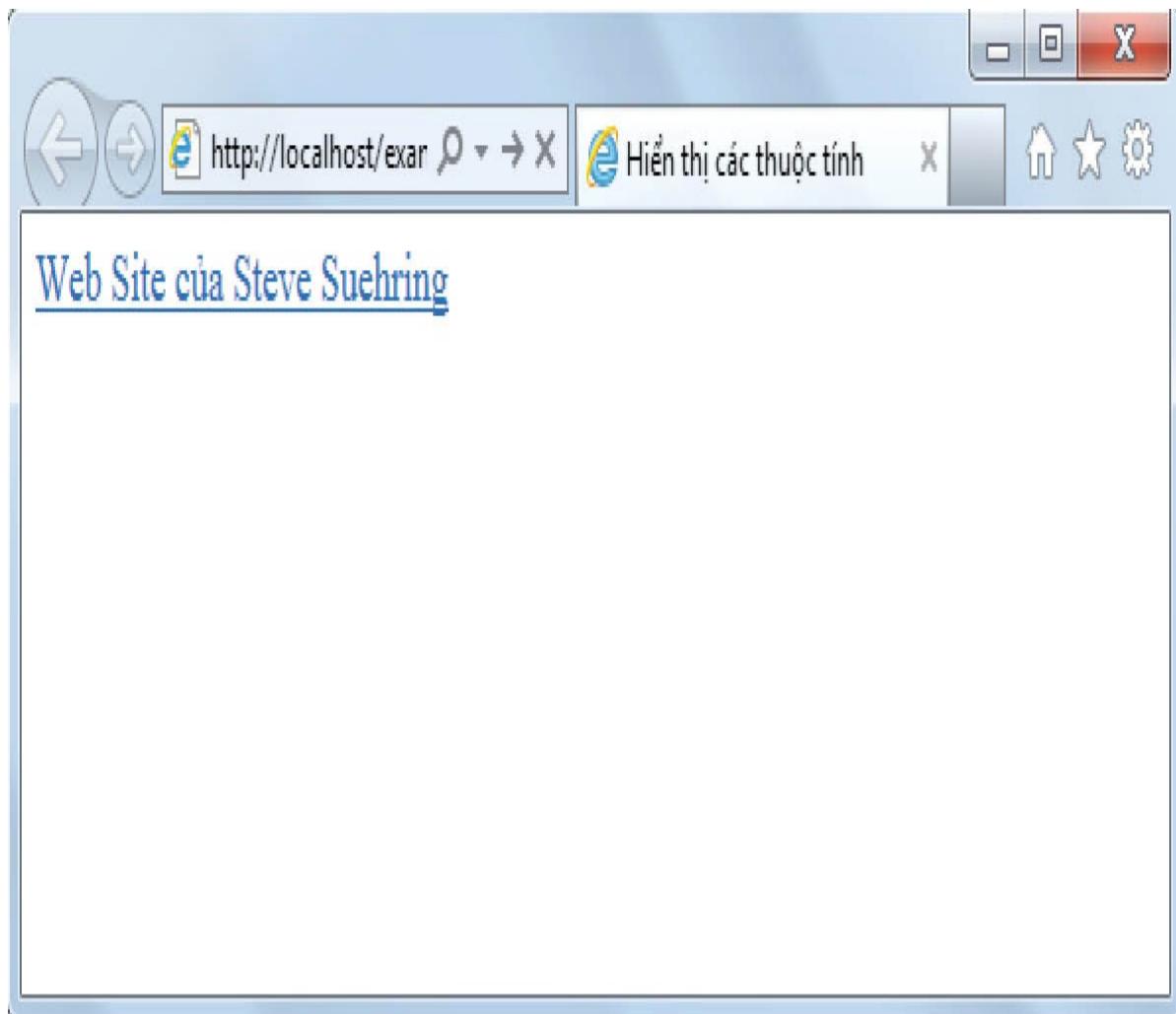
Chỉ cần vài câu lệnh JavaScript với phương thức `getElementById()` là bạn có thể gọi được hàm này như trong bài tập sau.

Truy xuất các thuộc tính của một phần tử

1. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo khác sửa file showattribs.htm trong thư mục mã nguồn mẫu của Chương 10.
2. Trong trang này, thay thế chú thích TODO bằng đoạn mã in đậm sau. (Đoạn mã này cũng có trong file showattribs.txt của Tài nguyên đi kèm.)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Hiển thị các thuộc tính</title>
    <script type="text/javascript">
        function showattribs(e) {
            var e = document.getElementById("braingia");
            var elemList = ""; //khai báo và khởi tạo
            // các phần tử
            for (var element in e) {
                var attrib = e.getAttribute(element);
                elemList = elemList + element + "=" + attrib + "\n";
            }
            alert(elemList);**
        }
    </script>
</head>
<body>
<a onclick="return showattribs();" href="http://www.braingialink.com/id="braingialink">Web Site của Steve Suehring</a>
<script type="text/javascript">
</script>
</body>
</html>
```

3. Lưu file và dùng trình duyệt mở lại. Bạn sẽ thấy một trang như sau:



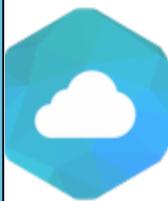
4. Nhấn vào liên kết. Hàm JavaScript sẽ được thực thi. Hàm này truy xuất tới các thuộc tính của phần tử `a`, duyệt qua và gắn chúng vào một biến. Cuối cùng, biến này được hiển thị trong hộp thoại `alert()` như trong hình dưới đây:

Message from webpage

X



```
nextSibling: null
onresizeend: null
onrowenter: null
aria-haspopup: null
childNodes: null
ondragleave: null
canHaveHTML: null
onbeforepaste: null
ondragover: null
onbeforecopy: null
aria-disabled: null
onpage: null
recordNumber: null
previousSibling: null
nodeName: null
onbeforeactivate: null
accessKey: null
currentStyle: null
scrollLeft: null
onbeforeeditfocus: null
oncontrolselect: null
aria-hidden: null
onblur: null
hideFocus:
clientHeight: null
style: null
onbeforedeactivate: null
dir: null
aria-expanded: null
onkeydown: null
nodeType: null
ondragstart: null
onscroll: null
onpropertychange: null
ondragenter: null
id: braingialink
aria-level: null
onrowsinserted: null
scopeName: null
lang: null
onmouseup: null
aria-busy: null
oncontextmenu: null
language: null
scrollTop: null
offsetWidth: null
onbeforeupdate: null
onreadystatechange: null
```



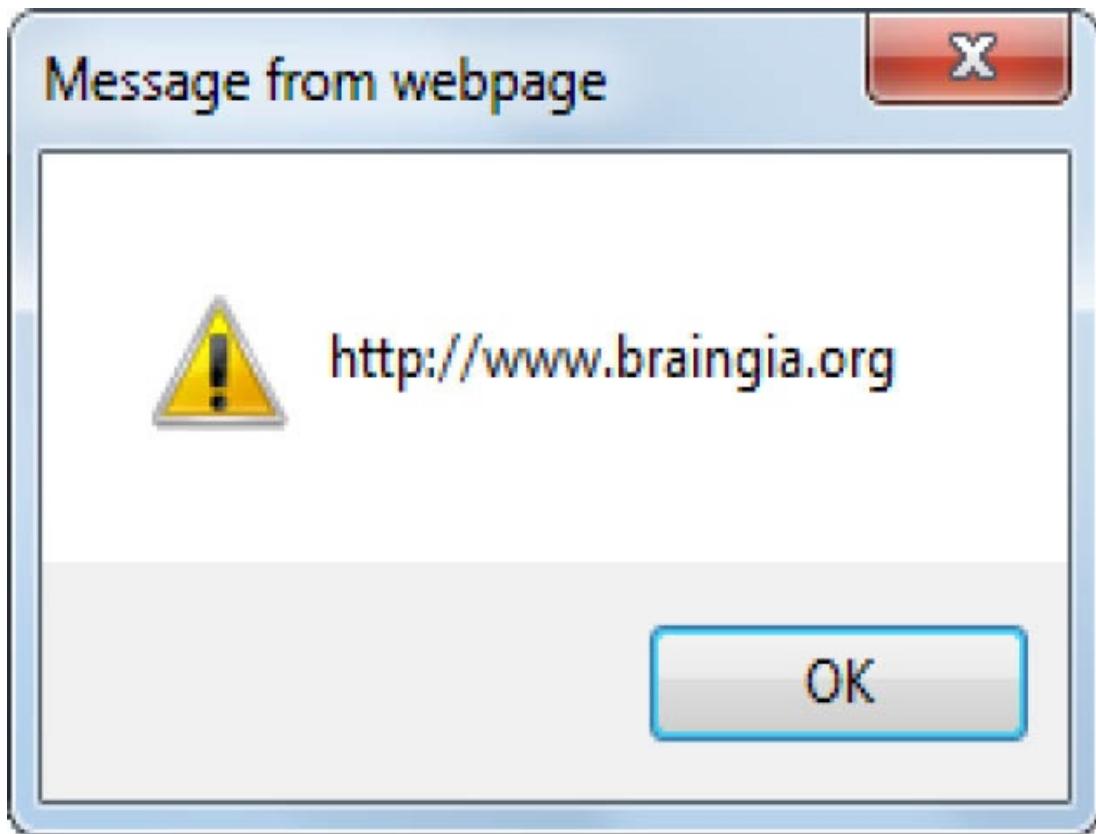
Thiết lập các thuộc tính

Bạn đã học cách sử dụng hàm `getAttribute()` để lấy giá trị trong một thuộc tính. Sau đây, bạn sẽ học cách sử dụng phương thức `setAttribute()` để thiết lập giá trị cho thuộc tính.

Phương thức `setAttribute()` nhận vào hai tham số: thuộc tính mà bạn muốn thay đổi và giá trị mà bạn muốn thiết lập cho thuộc tính đó. Sau đây là ví dụ về việc thay đổi giá trị thuộc tính `href`, bạn cũng có thể tìm thấy ví dụ này trong file setattrib.htm của Tài nguyên đi kèm.

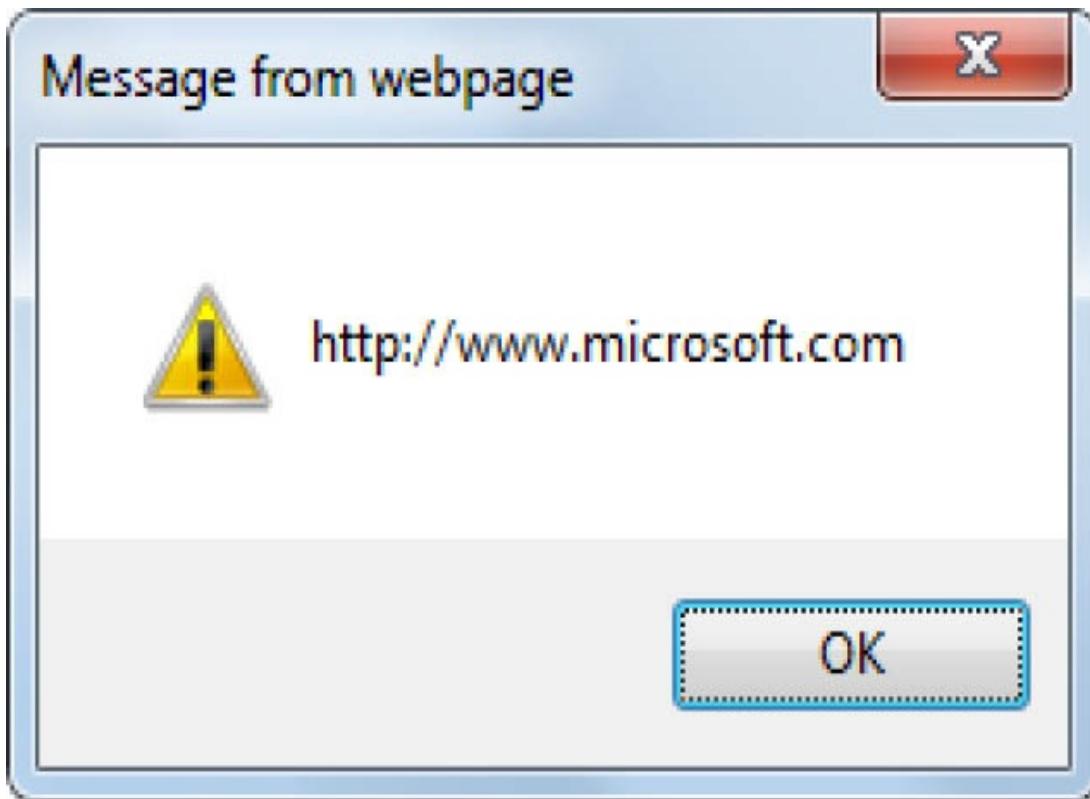
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Thiết lập các thuộc tính</title>
</head>
<body>
<a href="http://www.braingia.org" id="braingialink">Web Site
<script type="text/javascript">
    var a1 = document.getElementById("braingialink");
    alert(a1.getAttribute("href"));
    a1.setAttribute("href", "http://www.microsoft.com");
    alert(a1.getAttribute("href"));
</script>
</body>
</html>
```

Khi mở trang web này trong một trình duyệt, bạn sẽ thấy hộp thoại thông báo giá trị hiện tại của thuộc tính `href` như trong Hình 10-5.



Hình 10-5 Giá trị ban đầu của thuộc tính `href`

Khi hộp thoại đóng lại, phương thức `setAttribute()` sẽ được thực thi và thuộc tính `href` sẽ thay đổi như trong Hình 10-6.



Hình 10-6 Giá trị mới của thuộc tính `href`

Phương thức `setAttribute()` không được hỗ trợ một cách thống nhất trong các phiên bản IE trước phiên bản 8, vì vậy, có một cách chắc chắn hơn để thiết lập thuộc tính là dùng dấu chấm để truy cập vào thuộc tính của phần tử.

Ví dụ, thiết lập thuộc tính `href`:

```
a1.href = "http://www.braingia.org";
```

Tuy nhiên, nếu ứng dụng không hỗ trợ các phiên bản cũ của IE thì bạn nên dùng các phương thức `setAttribute()` và `getAttribute()`. Ngoài ra, bạn có thể dùng phương thức `removeAttribute()` để xóa hoàn toàn thuộc tính của phần tử. Tất nhiên, bạn vẫn cần chú ý đến việc hỗ trợ trên các phiên bản IE cũ.

Tạo các phần tử

Bên cạnh các phần tử tồn tại sẵn trên trang web, bạn có thể tạo thêm các phần tử mới bằng DOM. Phần này sẽ hướng dẫn một số cách tạo mới các phần tử.

Thêm văn bản

Phương thức `createElement()` của đối tượng `document` cho phép tạo hoặc thêm phần tử vào một tài liệu. Ví dụ:

```
var newelement = document.createElement("p");
```

Biến `newelement` giờ đây tham chiếu đến phần tử mới được tạo ra. Để hiển thị phần tử này, bạn cần chèn nó vào tài liệu – thường việc này sẽ thực hiện sau khi thêm văn bản vào nó. Phần tử mới được chèn vào tài liệu bằng phương thức `appendChild()` như sau:

```
document.body.appendChild(newelement);
```

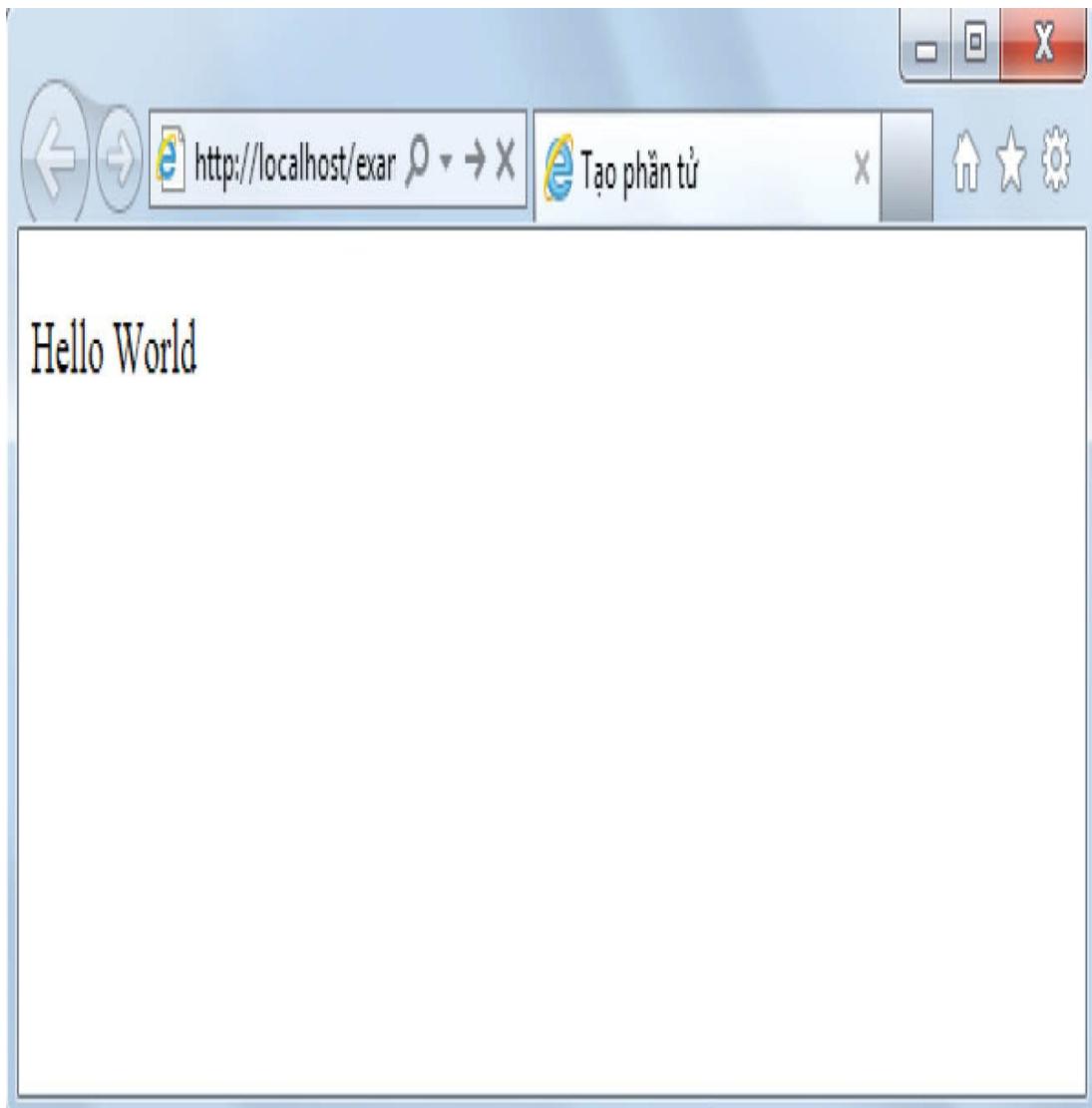
Nhưng phần tử `p` không có nội dung thì có ích gì? Bạn có thể dùng `appendChild()` kết hợp với phương thức `createTextNode()` như sau:

```
newelement.appendChild(document.createTextNode("Xin chào"));
```

Ba dòng mã trên có thể được dùng tại bất cứ đâu ngay sau khi phần `<body>` của tài liệu được khai báo. Dưới đây là cách dùng đoạn mã trong ngữ cảnh một trang web. Đoạn mã này có thể được tìm thấy trong file `create.htm` của Tài nguyên đi kèm.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Tạo phần tử</title>
</head>
<body>
    <script type="text/javascript">
        var newelement = document.createElement("p");
        document.body.appendChild(newelement);
        newelement.appendChild(document.createTextNode("Hello"));
    </script>
</body>
</html>
```

Khi dùng trình duyệt để mở, ta được kết quả như trong Hình 10-7.



Hình 10-7 Sử dụng phương thức `createElement`, `createTextNode` và `appendChild` để tạo phần tử mới.

Thêm phần tử và thiết lập ID

Ví dụ trước nói về cách thêm một phần tử vào tài liệu. Thông thường, bạn sẽ muốn thiết lập thuộc tính cho phần tử mới. Ví dụ tiếp theo mở rộng ví dụ trước bằng cách thêm thuộc tính *id* (bạn có thể tìm thấy ví dụ này trong file createid.htm của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>
```

```
<title>Tạo phần tử</title>
</head>
<body>
    <script type="text/javascript">
        var newelement = document.createElement("p");
        newelement.setAttribute("id", "newelement");
        document.body.appendChild(newelement);
        newelement.appendChild(document.createTextNode("Hello"));
    </script>
</body>
</html>
```

Xóa các phần tử

Bạn có thể xóa các phần tử trong tài liệu bằng phương thức *removeChild()*. Hãy nhớ lại đoạn mã để thêm phần tử mới ở phần trước. Chúng ta sẽ mở rộng bằng cách thêm vài phần tử *p* nữa:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <title>Tạo phần tử</title>
    </head>
    <body>
        <script type="text/javascript">
            for (var i = 0; i < 3; i++) {
                var element = document.createElement("p");
                element.setAttribute("id", "element" + i);
                document.body.appendChild(element);
                element.appendChild(document.createTextNode(
                    "Xin chào, tôi là phần tử " + i));
            }
        </script>
    </body>
</html>
```

Khi dùng trình duyệt mở trang, ta có một trang web như trong Hình 10-8.

**

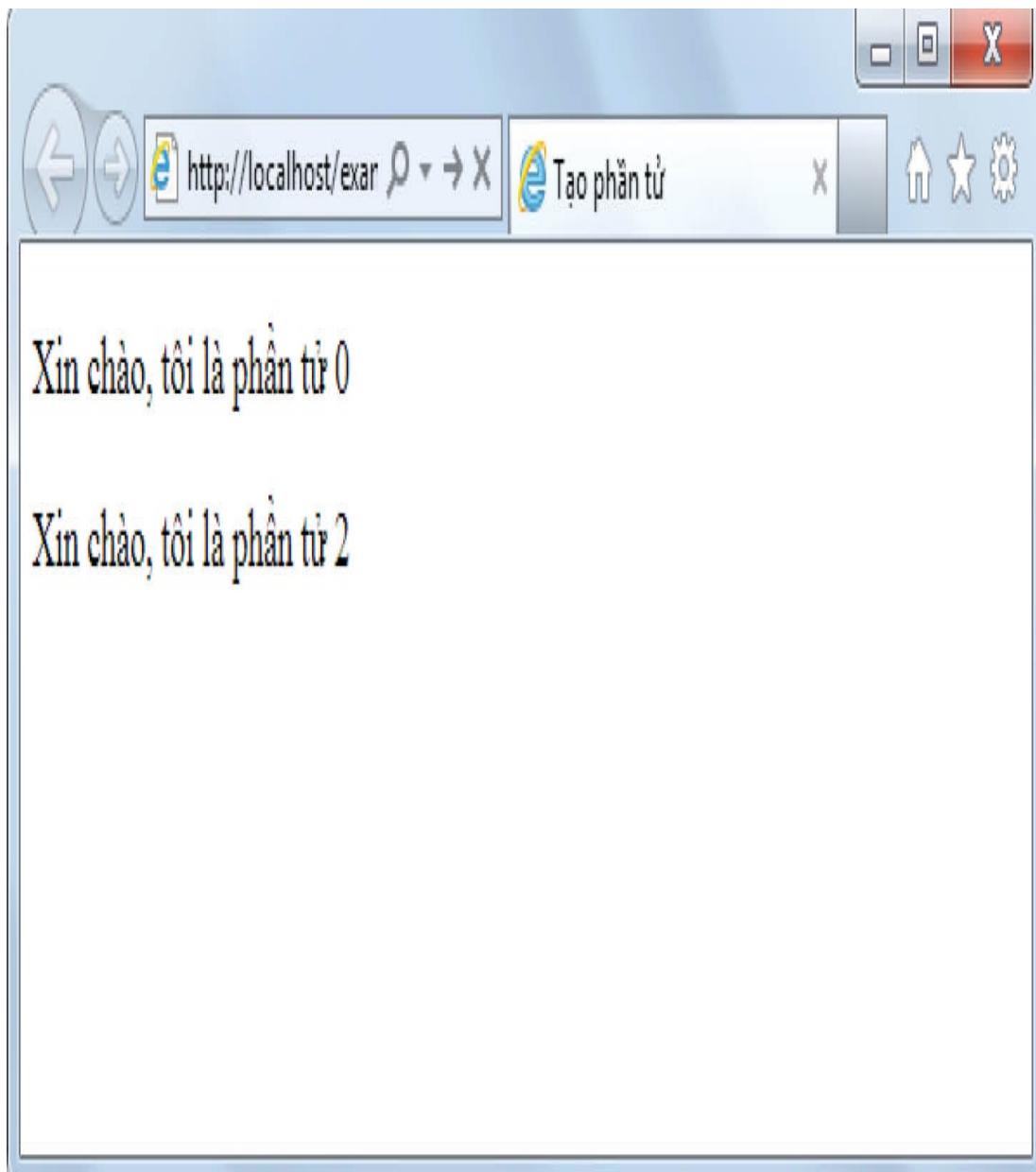
Bạn có thể thêm một số dòng lệnh để xóa phần tử vừa tạo: Lưu ý phương thức *removeChild()* có thể xóa bất cứ phần tử nào trong tài liệu chứ không chỉ những phần tử vừa tạo.

```
var removeel = document.getElementById("element1");
document.body.removeChild(removeel);
```

Trong ví dụ này, chúng ta thêm hai dòng lệnh bên trên vào ngay sau đoạn mã tạo phần tử. Trên thực tế, bạn có thể đặt lời gọi hàm *removeChild()* bất cứ đâu miễn là sau khi phần tử được tạo. Đoạn mã cuối cùng với các dòng lệnh mới được in đậm có dạng như sau. (Bạn có thể tìm thấy đoạn mã này trong file removeel.htm của Tài nguyên đi kèm).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Tạo phần tử</title>
</head>
<body>
    <script type="text/javascript">
        for (var i = 0; i < 3; i++) {
            var element = document.createElement("p");
            element.setAttribute("id", "element" + i);
            document.body.appendChild(element);
            element.appendChild(document.createTextNode(
                "Xin chào, tôi là phần tử " + i));
        }
        var removeel = document.getElementById("element1");
        document.body.removeChild(removeel);
    </script>
</body>
</html>
```

Hình 10-9 minh họa kết quả. Vòng lặp *for* vẫn tạo ra ba phần tử mới, nhưng đoạn mã in đậm đã xóa phần tử ở giữa sau khi nó vừa được tạo.



Hình 10-9 Sử dụng `removeChild()` để xóa một phần tử khỏi một tài liệu.

Bài tập

1. Tạo một tài liệu chứa một đoạn văn bản được tạo mới và thêm vào bằng cách sử dụng DOM. Tạo một liên kết ngay phía sau đoạn văn bản này và liên kết đến website mà bạn chọn. Hãy thiết lập thuộc tính *id* cho tất cả các phần tử.
2. Tạo tài liệu html với bất kỳ phần tử nào mà bạn muốn, hoặc sử dụng tài liệu

HTML có sẵn chứa các phần tử có thuộc tính *id*. Truy xuất hai trong số các phần tử đó, thay đổi và đưa chúng trở lại tài liệu. Những thay đổi mà bạn có thể tạo ra tùy thuộc vào kiểu phần tử mà bạn chọn. Ví dụ, nếu chọn phần tử *a*, bạn có thể thay đổi *href*; nếu chọn phần tử *p*, bạn có thể thay đổi văn bản.

3. Dùng DOM, tạo một tài liệu chứa bảng với ít nhất hai cột và hai dòng. Thêm văn bản vào các ô trong bảng.



Chương 11

Các sự kiện trong JavaScript và làm việc với trình duyệt

Sau khi đọc xong chương này, bạn có thể:

- Hiểu mô hình sự kiện trước đây.
- Hiểu mô hình sự kiện JavaScript W3C.
- Thêm hàm xử lý sự kiện trên trang web bằng JavaScript.
- Mở cửa sổ mới bằng JavaScript.
- Mở tab mới trên một trình duyệt web.
- Tạo bộ định thời bằng JavaScript.



Hiểu các sự kiện Window

Trong các chương trước, bạn đã được học cách dùng hàm xử lý sự kiện để hồi đáp hoạt động của người dùng hay các sự kiện trên tài liệu. Hãy nhớ lại, các sự kiện của đối tượng Window bao gồm `mouseover()`, `mouseout()`, `load()` và `click()`. Các sự kiện này được hỗ trợ khá tốt trên hầu hết các trình duyệt, nhưng nhiều sự kiện khác thì không. Phần này sẽ trình bày về sự kiện và cách sử dụng sự kiện trong lập trình JavaScript.

Các mô hình sự kiện

Thách thức đầu tiên khi tìm hiểu các sự kiện là hiểu về hai mô hình sự kiện khác nhau sau đây: mô hình được sử dụng trong các phiên bản trình duyệt Windows Internet Explorer trước phiên bản 9 và mô hình được tổ chức World Wide Web Consortium (W3C) định nghĩa. Mô hình cũ hơn - DOM cấp độ 0 – bao gồm các sự kiện đã được giới thiệu trong các chương trước (Chương 1 “Hiểu hơn về

JavaScript” đã giới thiệu sơ qua về DOM cấp 0). Mô hình sự kiện DOM cấp 0 tương thích với hầu hết các trình duyệt hỗ trợ JavaScript. Trong phần này, tôi sẽ giới thiệu ngắn gọn về DOM cấp 0 và sau đó giải thích sự khác biệt giữa hai mô hình sự kiện: mô hình W3C và mô hình Internet Explorer.

Sử dụng mô hình DOM cấp 0

Mô hình DOM cấp 0 là mô hình dễ sử dụng nhất (như bạn đã thấy từ các chương trước) và là mô hình thích hợp nhất cho việc xử lý sự kiện trong JavaScript. (Như đã đề cập, DOM cấp 0 được hỗ trợ trong tất cả các trình duyệt web phổ biến). Vậy tại sao chúng ta không sử dụng mô hình sự kiện DOM cấp 0 ở mọi nơi? Lý do thật đơn giản: Nó thiếu sự linh hoạt cần thiết để xử lý những sự kiện phức tạp. Ví dụ, DOM cấp 0 không thể xử lý nhiều sự kiện trên cùng một phần tử. Tuy nhiên, cũng không có gì sai khi dùng DOM cấp 0 cho các đoạn mã đơn giản như trong suốt cuốn sách này.

Mô hình sự kiện DOM cấp 0 gồm một số sự kiện mà các thẻ HTML tạo ra khi phản hồi lại những hành động hay sự thay đổi trạng thái của người dùng. Bảng 11-1 mô tả các sự kiện này.

BẢNG 11-1 Các sự kiện DOM cấp 0



| Tên sự kiện | Mô tả |
|----------------------------|--|
| <code>onblur()</code> | Phần tử bị mất focus (không còn được chọn bởi người dùng). |
| <code>onchange()</code> | Phần tử đã thay đổi (ví dụ, một người dùng gõ vào một ô nhập liệu) hoặc mất focus. |
| <code>onclick()</code> | Nhấn chuột vào một phần tử. |
| <code>ondbclick()</code> | Nhấn đúp chuột vào một phần tử. |
| <code>onfocus()</code> | Phần tử nhận được focus (phần tử được chọn). |
| <code>onkeydown()</code> | Một phím được nhấn (trái ngược với được nhả ra) trong khi phần tử đang được chọn. |
| <code>onkeypress()</code> | Một phím được nhấn khi phần tử đang được chọn. |
| <code>onkeyup()</code> | Một phím được nhả ra khi phần tử đang được chọn. |
| <code>onload()</code> | Phần tử được tải lên trang web (văn bản, frameset hoặc hình ảnh). |
| <code>onmousedown()</code> | Chuột được nhấn. |
| <code>onmousemove()</code> | Chuột di chuyển. |
| <code>onmouseout()</code> | Chuột di chuyển ra ngoài một phần tử. |
| <code>onmouseover()</code> | Chuột di chuyển trên một phần tử. |

| | |
|--------------------|--|
| <i>onmouseup()</i> | Khi nút bấm trên chuột máy tính được thả ra. |
| <i>onreset()</i> | Phần tử form được reset, chẳng hạn khi người dùng nhấn button reset trên form. |
| <i>onresize()</i> | Kích thước cửa sổ bị thay đổi. |
| <i>onselect()</i> | Nội dung của một phần tử trên form được chọn. |
| <i>onsubmit()</i> | Form được gửi đi. |
| <i>onunload()</i> | Văn bản hoặc frameset được gỡ bỏ. |

Các mô hình sự kiện mới: W3C và Internet Explorer

W3C phát triển một mô hình sự kiện cung cấp các hàm xử lý sự kiện hiệu quả, được hỗ trợ trong hầu hết các trình duyệt lớn trừ các phiên bản IE trước phiên bản 9. Vì mô hình sự kiện W3C và mô hình sự kiện Internet Explorer khác nhau nên khi lập trình JavaScript bạn cần chú ý đến cả hai mô hình này.

Về lý thuyết, quá trình xử lý sự kiện trong hai mô hình này tương tự nhau. Trước tiên, bạn đăng ký một sự kiện, sau đó liên kết sự kiện này với một hàm xử lý sự kiện. Khi sự kiện đăng ký diễn ra, hàm xử lý sự kiện sẽ được gọi. Tuy nhiên, vị trí xảy ra sự kiện lại là điểm khác biệt quan trọng giữa hai mô hình.

Để hiểu sự khác biệt này, hãy hình dung ra một tài liệu có phần tử `<body>` và phần tử `` nằm trong phần tử `<body>` này.. Nếu người dùng di chuột lên trên hình ảnh, sự kiện `onmouseover()` sẽ được xử lý trước trong thẻ `<body>` hay thẻ ``? Hai mô hình bất đồng trong việc xác định vị trí trước tiên nên xử lý sự kiện này.

W3C hỗ trợ hai hình thức xác định vị trí nên xử lý sự kiện: *Event Capture* và *Event Bubbling*. Đối với Event Capture, DOM tìm kiếm hàm xử lý bắt đầu từ cấp cao nhất (cấp tài liệu) và tiếp tục xuống các phần tử con trên cây DOM. Nếu bạn sử dụng Event Capture, phần tử `<body>` sẽ xử lý sự kiện trước, sau đó tới ``. Event Bubbling thì ngược lại, phần tử chứa sự kiện sẽ xử lý trước tiên, sau đó mới đến cấp cha của nó.

Mô hình W3C – cũng như tất cả các trình duyệt theo chuẩn W3C – có thể sử dụng cả hai hình thức xử lý sự kiện trên, trong khi các phiên bản Internet Explorer trước phiên bản 9 chỉ sử dụng Event Bubbling. Với mô hình W3C, bạn đăng ký một sự kiện bằng phương thức `addEventListener()`. Với mô hình Internet Explorer cũ, bạn dùng `attachEvent()` cho mục đích tương tự. Trên thực tế, điều này có nghĩa là bạn cần sử dụng cả hai phương thức trong mỗi đoạn mã để xử lý các sự kiện và lựa chọn phương thức phù hợp với trình duyệt sẽ diễn

ra tại thời điểm đoạn mã được thực thi. Cấu trúc cơ bản của phương thức *addEventListener()* là:

```
addEventListener(sự kiện, hàm, capture/bubble);
```

Trong đó, tham số *capture/bubble* là một giá trị Boolean, *true* tương đương với Event Capture và *false* tương đương với Event Bubbling. Đây là ví dụ điển hình về lời gọi phương thức *addEventListener()* cho button Submit. Lời gọi đăng ký sự kiện *submit* (gửi form), chỉ định một hàm có tên *myFunction()* (hàm này được định nghĩa ở một vị trí khác trong chương trình) và sử dụng cách bắt sự kiện Event Capture.

```
window.addEventListener("submit", myFunction(), true);
```

Để đăng ký sự kiện tương tự bằng Event Bubbling, viết như sau:

```
window.addEventListener("submit", myFunction(), false);
```

Mô hình Internet Explorer trước đây không yêu cầu tham số thứ ba khi gọi *attachEvent()* vì nó chỉ hỗ trợ Event Bubbling.

Sau đây là ví dụ đăng ký sự kiện *submit* trên các phiên bản Internet Explorer cũ và liên kết nó với phương thức *myFunction()* bằng cách gọi *attachEvent()*:

```
window.attachEvent("onsubmit", myFunction());
```

Bạn có thể nhận ra một khác biệt nhỏ trong tên của sự kiện cần xử lý - *submit* và *onsubmit* (trong DOM cấp 0). Nhiều sự kiện trong DOM cấp 2 đã được đổi tên. Bảng 11-2 trình bày tên của một vài sự kiện DOM cấp 0 và sự kiện tương ứng trong DOM cấp 2. Chú ý rằng các sự kiện trong DOM cấp 2 đơn giản loại bỏ từ “on” ra khỏi tên sự kiện của mô hình cũ. (Mô hình Internet Explorer cũ sử dụng tên sự kiện trong DOM cấp 0).

BẢNG 11-2 Các sự kiện trong DOM cấp 0 và DOM cấp 2.

| Sự kiện DOM cấp 0 | Sự kiện DOM cấp 2 |
|----------------------|-------------------|
| <i>onblur()</i> | <i>blur</i> |
| <i>onfocus()</i> | <i>focus</i> |
| <i>onchange()</i> | <i>change</i> |
| <i>onmouseover()</i> | <i>mouseover</i> |

| | |
|----------------------|------------------|
| <i>onmouseout()</i> | <i>mouseout</i> |
| <i>onmousemove()</i> | <i>mousemove</i> |
| <i>onmousedown()</i> | <i>mousedown</i> |
| <i>onmouseup()</i> | <i>mouseup</i> |
| <i>onclick()</i> | <i>click</i> |
| <i>ondbclick()</i> | <i>dbclick</i> |
| <i>onkeydown()</i> | <i>keydown</i> |
| <i>onkeyup()</i> | <i>keyup</i> |
| <i>onkeypress()</i> | <i>keypress</i> |
| <i>onsubmit()</i> | <i>submit</i> |
| <i>onload()</i> | <i>load</i> |
| <i>onunload()</i> | <i>unload</i> |

Cả mô hình W3C và Internet Explorer cũ đều có các phương thức để gỡ bỏ bộ lắng nghe sự kiện. Trong W3C, phương thức này là *removeEventListener()* và nhận vào ba tham số giống như *addEventListener()*:

`removeEventListener(sự kiện, hàm, capture/bubble)`

Trong các phiên bản cũ của Internet Explorer, phương thức được sử dụng là *detachEvent()*:

`detachEvent(sự kiện, hàm);`

Nếu bạn không muốn việc xử lý sự kiện tiếp tục diễn ra ở cấp cao hơn hoặc thấp hơn sau khi hàm xử lý sự kiện đầu tiên được thực thi, W3C hỗ trợ phương thức *stopPropagation()* còn Internet Explorer cũ dùng thuộc tính *cancelBubble*. Trong phần sau của chương “Mở, đóng và thay đổi kích thước cửa sổ” sẽ có một ví dụ về việc này.

Hàm xử lý sự kiện chung

Việc thêm nhiều bộ lắng nghe cho những sự kiện cần xử lý nhanh làm chương trình trở nên công kẽm. Trên thực tế, bạn chỉ cần sử dụng một hàm xử lý sự kiện chung (generic event handler) cho tất cả các trình duyệt nhằm giảm bớt sự không tương thích giữa các mô hình. Ví dụ 11-1 giới thiệu một hàm xử lý sự kiện chung. Bạn có thể tìm thấy đoạn mã này trong file ehandler.js trong thư mục mã

nguồn mẫu của Chương 11.

VÍ DỤ 11-1 Hàm xử lý sự kiện chung.

```
var EHandler = {};
    if (document.addEventListener) {
        EHandler.add = function(element, eType, eFunc) {
            if (eType == "load") {
                if (typeof window.onload =
                    var existingOnload
                    //onload s
                    window.onload = fu
                    existingOn
                    eFunc();
                } // kết thúc hàm
            } else {
                window.onload = eF
            }
        } else {
            element.addEventListener(e
        }
    };
    EHandler.remove = function(element, eType,
        element.removeEventListener(eType,
    );
}
else if (document.attachEvent) {
    EHandler.add = function(element, eType, eFunc) {
        if (eType == "load") {
            if (typeof window.onload == "funct
                var existingOnload = windo
                window.onload = function()
                    existingOnload();
                    eFunc();
                } // kết thúc hàm xử lý on
            } else {
                window.onload = eF
            }
        } else {
            element.attachEvent("on" + eType,
        }
    };
}
```

```
EHandler.remove = function(element, eType, eFunc)
    element.detachEvent("on" + eType, eFunc);
}
}
```

Hàm xử lý sự kiện chung tạo một đối tượng *EHandler* (viết tắt của *event handler*), sau đó thêm vào hai phương thức *add()* và *remove()*, cả hai đều có thể dùng được trong mô hình W3C và Internet Explorer cũ. Trong mỗi mô hình, phương thức *add* xác định xem đó có phải là sự kiện *Load* không, có nghĩa là hàm có cần được thực thi khi trang được tải lên không. Nếu đúng, *add* cần xác định xem liệu đã có bất kỳ hàm *onload* nào được định nghĩa chưa. Nếu có, hàm *onload* có sẵn phải được đưa vào thực thi cùng với hàm mới được định nghĩa.

Để đi tiếp chương này cũng như phần còn lại của quyển sách, tôi đề nghị bạn lưu đoạn mã trên vào một file JavaScript ngoài có tên là *ehandler.js* và thêm file này vào bất kỳ file JavaScript nào cần xử lý sự kiện. Xin nhắc lại cách thêm file JavaScript ngoài như sau:

```
<script type="text/javascript" data-src="ehandler.js"></script>
```

Một điều may mắn là, Windows Internet Explorer phiên bản 9 sử dụng mô hình tương thích với mô hình W3C. Do đó, khi phiên bản này trở nên phổ biến hơn trên thị trường, *attachEvent* sẽ dần được thay thế. Tuy nhiên, những trang web hỗ trợ các trình duyệt cũ sẽ vẫn cần sử dụng mô hình cũ của Microsoft trong nhiều năm tới.

Bạn có thể cải thiện đoạn mã xử lý sự kiện trên cho phù hợp với tình huống thực tế khi xây dựng các ứng dụng JavaScript mạnh hơn. (Tham khảo thêm thông tin về chủ đề này tại trang web của John Resig: <http://ejohn.org/blog/flexible-javascript-events/>). Tuy nhiên, giải pháp tốt nhất là sử dụng một framework hoặc thư viện JavaScript như jQuery để trừu tượng hóa mô hình sự kiện.

Thông tin bổ sung Tôi sẽ trình bày chi tiết hơn về jQuery và các framework của JavaScript trong Chương 19 "Sơ lược về AJAX". Trước mắt, đoạn mã xử lý sự kiện đơn giản đã trình bày ở trên cũng đủ cho các ví dụ trong sách này. Để tìm hiểu thêm về lập trình JavaScript xử lý sự kiện nâng cao, tôi khuyên bạn nên dùng một thư viện như jQuery thay vì cách xử lý đơn giản như đưa ra ở đây.

Phần còn lại của chương này áp dụng những kiến thức bạn đã thu được về sự kiện cũng như về Mô hình đối tượng trình duyệt trong Chương 9; Mô hình đối tượng tài liệu (DOM) trong Chương 10 và cú pháp ở phần I của cuốn sách “JavaScript là gì, dùng ở đâu, tại sao dùng và sử dụng như thế nào?”.

Nhận biết thông tin khách truy cập

Lập trình với JavaScript thường đòi hỏi phải *nhận biết trình duyệt* mà khách truy cập sử dụng để truy cập trang web. Trong nhiều năm, việc này được thực hiện chủ yếu bằng cách sử dụng thuộc tính *userAgent* mang thông tin về trình duyệt của khách truy cập trong đối tượng *navigator*. Tuy nhiên, cách làm này hiện nay không còn được khuyến khích bởi vì khách truy cập có thể dễ dàng giả mạo thông tin. Ngoài ra, việc duy trì chính xác thông tin *userAgent* cho tất cả các phiên bản của các trình duyệt cũng có nhiều khó khăn.

Bản thân tôi dùng năm phiên bản trình duyệt khác nhau. Vì vậy, việc cập nhật danh sách xác định trình duyệt nào hỗ trợ tính năng nào rất vất vả. Hãy tưởng tượng mọi chuyện sẽ thế nào khi bạn luôn đảm bảo ứng dụng JavaScript của mình sẽ tương thích với *mọi* trình duyệt và *mọi* phiên bản của trình duyệt đó. Điều này gần như không thể. Đây là còn chưa kể đến việc các chuỗi *userAgent* có thể bị giả mạo (bởi những đoạn mã độc hại hoặc cho các mục đích khác!).

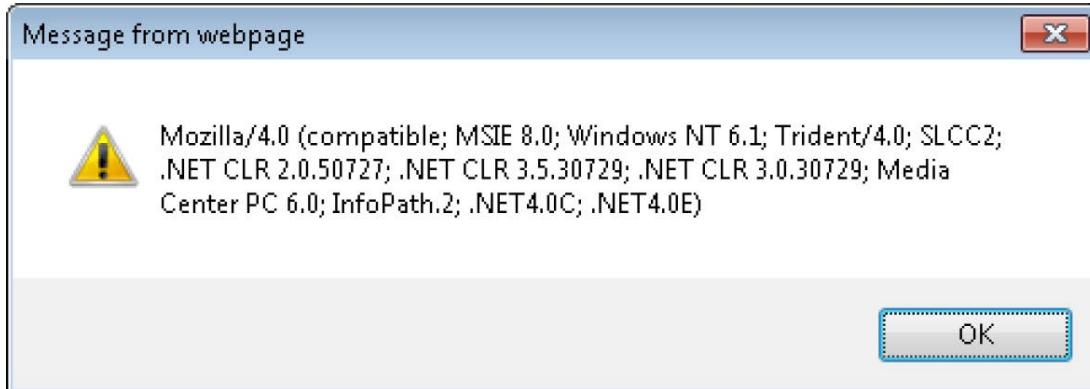
Vì vậy, sau đây tôi sẽ chỉ trình bày ngắn gọn về thuộc tính *userAgent* và sau đó chuyển sang trình bày những phương thức mới (tốt hơn nhiều) để xác định liệu đoạn mã JavaScript mà bạn đang sử dụng có hoạt động tốt trên trình duyệt của khách truy cập hay không. Phần này cũng xem xét các thuộc tính hữu dụng khác của đối tượng *navigator*.

Tóm lược thuộc tính *userAgent*

Như đã nói, chuỗi *userAgent* là một thuộc tính của đối tượng *navigator*. Nó chứa thông tin về trình duyệt của người dùng. Để xem thông tin *userAgent*, chỉ cần nhập dòng sau vào trình duyệt của bạn:

```
javascript:alert(navigator.userAgent);
```

Nếu đang sử dụng Windows Internet Explorer 7, bạn có thể thấy một hộp thoại như Hình 11-1.



HÌNH 11-1 Thuộc tính `userAgent` của đối tượng `navigator`

Các trình duyệt khác sẽ hiển thị thông tin theo cách khác. Ví dụ, trình duyệt Firefox sẽ hiển thị như sau:

Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.3) !

Chuỗi trên thay đổi theo từng phiên bản trình duyệt. Nếu cố công theo dõi từng phiên bản mới của tất cả các trình duyệt, sau đó lại tìm hiểu xem phiên bản nào hỗ trợ tính năng nào của JavaScript, bạn sẽ phí phạm thời gian của chính mình, của công ty và của cả khách hàng.

Cách tốt hơn cả để biết tính năng nào được hỗ trợ trên trình duyệt là kiểm tra tính năng, nội dung này sẽ được thảo luận ngay trong phần tiếp theo.

Kiểm tra tính năng

Kiểm tra tính năng (*feature testing*) hay *nhận biết đối tượng* (*object detection*) là kỹ thuật được sử dụng trong JavaScript nhằm xác định trình duyệt của người dùng có hỗ trợ một tính năng nhất định hay không.

Thật may là bạn không cần kiểm tra *tất cả* các hàm và các phương thức cần dùng. Mô hình DOM cấp 0 và các hàm JavaScript cũ được hỗ trợ rộng rãi và được triển khai gần như tương tự nhau trên các trình duyệt nên việc kiểm thử cho từng tính năng cụ thể là không cần thiết.

Tuy nhiên, bạn vẫn phải kiểm tra xem trình duyệt có hỗ trợ JavaScript hay không

bởi vì không phải tất cả các trình duyệt đều hỗ trợ JavaScript và không phải tất cả người dùng đều bật tính năng JavaScript trên trình duyệt của họ.

Toán tử `typeof` là cơ chế chủ yếu được dùng để kiểm tra tính năng. Toán tử này được sử dụng như sau:

```
if (typeof tenTinhNang != "undefined") {  
    // Làm gì đó thú vị với tính năng này  
}
```

Để kiểm tra phương thức `getElementById()` có được hỗ trợ không, hay nói cách khác trình duyệt có hỗ trợ DOM cấp độ cao hay không, bạn dùng đoạn mã sau:

```
if (typeof document.getElementById != "undefined") {  
    alert("Có hỗ trợ getElementById");  
} else {  
    alert("Không hỗ trợ getElementById");  
}
```

Cũng có thể bạn không muốn dùng `typeof` mà làm như sau:

```
if (document.getElementById) { ... }
```

Tuy nhiên, cách này không hoàn toàn đáng tin cậy như việc kiểm tra bằng `typeof`. Khi bạn không dùng `typeof`, phương thức hay thuộc tính được kiểm tra có thể trả về `0` hoặc `false` theo mặc định, làm cho điều kiện kiểm tra thất bại; nói cách khác, kết quả trả về là trình duyệt không hỗ trợ phương thức hay thuộc tính này trong khi thực tế là ngược lại. Vì vậy, sử dụng `typeof` là cách kiểm tra an toàn và đáng tin cậy hơn cả.

Một cách khác là dùng toán tử `điều kiện ba ngôi` để thiết lập cờ ở đầu đoạn mã. Sau đó sử dụng câu lệnh điều kiện `if` thông thường để kiểm tra cờ này:

```
//Kiểm tra getElementById, gán kết quả vào biến getElem  
var getElem = (typeof document.getElementById == "function")  
//bây giờ bạn có thể kiểm tra getElem  
If (getElem) {  
    //Chúng ta biết getElementById được hỗ trợ  
    //vì vậy chúng ta sẽ dùng nó  
}
```

Không dùng các trình duyệt cũ để chạy JavaScript

Một trong những vấn đề làm các lập trình viên web đau đầu là họ phải làm việc với các trình duyệt cũ. Việc viết một trang web kiểu mới cho các trình duyệt mới nhưng vẫn hoạt động tốt trên các trình duyệt cũ càng lúc càng trở nên khó khăn hơn. Thế nào là một trình duyệt cũ? Nếu hỏi ba chuyên gia thiết kế web, có thể bạn sẽ nhận được ba câu trả lời khác nhau. Với cá nhân tôi, trình duyệt cũ là trình duyệt từ ba năm tuổi, thậm chí là hai năm tuổi trở lên.

Theo định nghĩa này, Firefox 1.0 sẽ là trình duyệt cũ mặc dù hầu hết các trang web đều hiển thị tốt trên trình duyệt này và tất cả mã JavaScript cũng chạy tốt trên Firefox 1.0. Một tiêu chí tổng quát hơn được nhiều nhà thiết kế web dùng để định nghĩa trình duyệt cũ là các phiên bản trước phiên bản 5 của trình duyệt Internet Explorer hoặc Netscape.

Với thực tế là các phiên bản cũ hơn phiên bản Microsoft Internet Explorer 5.0 vẫn được nhiều người sử dụng, có tốt hơn cả là nên chấp nhận việc mã JavaScript sẽ không chạy trên các trình duyệt đó. Gần đây, tôi cài đặt bản Netscape 3 (ra đời năm 1997). Trình duyệt này gặp rất nhiều vấn đề với các đoạn mã JavaScript cũng như với việc hiển thị HTML và CSS trên các trang web cơ bản. Điều này cũng dễ hiểu vì Netscape3 ra đời trước khi các tiêu chuẩn hiện nay được giới thiệu. Một vấn đề nữa là dù bạn cố gắng thế nào đi nữa, trang web của bạn có thể sẽ không chạy trên các phiên bản cũ của trình duyệt. Tôi khuyên bạn nên chọn cấp trình duyệt thấp nhất để hỗ trợ thiết kế theo mục tiêu đó và luôn ghi nhớ rằng cấp trình duyệt càng cao thì số lượng khách thăm trang càng hạn chế. Mục đích là cân bằng giữa hai tiêu chí của một trang web: thân thiện và độc đáo.

Có hai kỹ thuật chính để tránh không dùng JavaScript trên các trình duyệt cũ: chèn ghi chú HTML bên trong JavaScript, sử dụng thẻ `<noscript>` `</noscript>`.

Để đưa chú thích HTML vào đoạn mã JavaScript, ta bao chúng trong cặp `<!--` và `-->` như ví dụ sau:

```
<script type="text/javascript">
<!-- //Bắt đầu chú thích
var helloelem = document.getElementById("hello");
```

```
alert(helloelem.innerHTML);  
// Kết thúc chú thích -->  
</script>
```

Không phải mọi trình duyệt đều chấp nhận các chú thích HTML, vì vậy đôi khi bạn vẫn gặp một số lỗi. Định dạng chú thích này ngày càng trở nên ít phổ biến hơn. Cho đến khi các trình duyệt cũ bị loại bỏ hoàn toàn thì cách làm này không còn cần thiết nữa.

Nội dung trong cặp thẻ `<noscript> </noscript>` sẽ hiển thị khi trình duyệt bị phát hiện không hỗ trợ JavaScript. Sau đây là một ví dụ về `<noscript>`:

```
<noscript>  
<p> Trang này yêu cầu JavaScript </p>  
</noscript>
```

Khi dùng trình duyệt không hỗ trợ JavaScript, người dùng sẽ thấy nội dung nằm giữa cặp thẻ `<noscript> </noscript>`. Trong ví dụ này, họ sẽ thấy dòng thông báo: “Trang này yêu cầu JavaScript”. Lưu ý rằng đoạn mã `<noscript>` không làm gián đoạn việc thực thi hoặc phân tích cú pháp, do đó đoạn mã HTML còn lại trong trang vẫn hiển thị như bình thường.

Tôi khuyên bạn chỉ sử dụng `<noscript>` trong các ứng dụng bắt buộc cần JavaScript chứ không phải trong những ứng dụng chỉ dùng JavaScript cho các hiệu ứng bề ngoài như ảnh cuộn. Việc lạm dụng JavaScript hoặc sử dụng không đúng cách sẽ gây trở ngại cho người sử dụng. Không có gì tệ hơn việc để người dùng truy cập vào trang web chỉ để thấy nó không hoạt động, không phản hồi hoặc bị khóa vì một vài đoạn mã JavaScript không cần thiết.

Mách nhỏ Bạn cần nhớ rằng có nhiều lý do khiến người dùng không thể sử dụng JavaScript, có thể họ phải sử dụng trình duyệt trợ giúp đặc biệt hoặc đơn giản là chương trình đọc văn bản. Vì vậy, hãy cố gắng tạo đầy đủ các chức năng văn bản cho trang web và cung cấp một sơ đồ trang tiện dụng.

Các phương thức và thuộc tính khác của đối tượng navigator

Mặc dù thuộc tính *userAgent* không được ưa dùng song đối tượng *navigator* thực sự cung cấp nhiều thông tin hữu ích mà người lập trình JavaScript có thể truy xuất. Chương 9 trình bày chi tiết về đối tượng này, bao gồm tất cả các thuộc tính của nó cũng như cách xác định liệu Java đã được bật trong trình duyệt hay chưa.

Chú ý Hãy thận trọng khi sử dụng đối tượng *navigator*. Đôi khi các kết quả có thể không hoàn toàn chính xác. Không những thế, *navigator* sẽ không khả dụng trên những trình duyệt không hỗ trợ JavaScript. Vì vậy, đừng để chức năng chính của trang web phụ thuộc vào thuộc tính của đối tượng *navigator*!

Mở, đóng và thay đổi kích thước cửa sổ



Một trong những tính năng ít được chào đón nhất của JavaScript là khả năng mở, đóng và thay đổi kích thước cửa sổ trình duyệt. Việc mở thêm một cửa sổ trên trình duyệt để đáp lại hoặc đóng vai trò như một phần của sự kiện *onLoad* là thao tác thường xuyên và phiền nhiễu của các nhà quảng cáo. Thiết lập mặc định trong Mozilla Firefox, Opera, và một số trình duyệt khác cho phép người dùng chặn tính năng này. IE 6.0 trên Service Pack 2 và các phiên bản sau cũng cung cấp lựa chọn đó.

Tôi chưa từng thấy một cửa sổ pop-up tự động nào thật sự hữu ích với người dùng mà không khiến họ cảm thấy khó chịu. Nếu bạn chắc chắn rằng trang web của mình cần một thành phần để mở một cửa sổ mới, tôi khuyên bạn nên xem xét lại việc điều hướng trước khi làm việc đó. Người dùng sẽ cảm kích không chỉ vì trang web đã được đơn giản hóa và trực quan hơn mà còn vì nó ít phụ thuộc vào JavaScript và sẽ hoạt động tốt trên cả những trình duyệt mà ngôn ngữ này đã bị vô hiệu hóa.

Mặc dù các cửa sổ gây phiền toái, nhưng đôi khi người dùng muốn mở cửa sổ mới khi họ nhấn chuột. Chẳng hạn, nhấn vào liên kết để mở một cửa sổ nhỏ cho phép người dùng lựa chọn từ một menu hoặc hiển thị thông tin trợ giúp...

Đối tượng *window* chứa một số phương thức hữu ích cho việc mở, đóng và thay đổi kích thước cửa sổ. Phương thức *open* dùng để mở cửa sổ trình duyệt mới. Cú pháp cơ bản của nó là:

```
window.open(url, name, features)
```

Tham số *url* là chuỗi đại diện cho đường dẫn (URL) sẽ được tải. Nếu tham số này để trống, trình duyệt sẽ mở một trang trống mặc định. Tham số *name* đại diện cho tên cửa sổ được mở. Nếu đã có một cửa sổ cùng tên được mở sẵn, phương thức này sẽ mở URL trong cửa sổ đó; nếu không, cửa sổ mới sẽ được mở ra.

Tham số *features* là một chuỗi chứa nhiều lựa chọn được phân tách với nhau bởi dấu phẩy đại diện cho những đặc tính mà bạn muốn cửa sổ mới phải có chẳng hạn như chiều dài, chiều rộng và thanh cuộn của cửa sổ. Bảng 11-3 liệt kê một số đặc tính có sẵn. Danh sách này không toàn diện vì các trình duyệt hỗ trợ những đặc tính khác nhau và chúng có tên khác nhau. Xem <http://msdn2.microsoft.com/en-us/library/ms536651.aspx> để có thêm thông tin về Internet Explorer và <https://developer.mozilla.org/en/DOM/window.open> để có thêm thông tin về Firefox và dòng Mozilla.

BẢNG 11-3 Một số đặc tính của cửa sổ được sử dụng trong phương thức *open()* của đối tượng *window*.

| Tên đặc tính | Mô tả |
|--------------------|---|
| <i>directories</i> | Xác định có hiển thị thanh công cụ cá nhân hoặc thanh công cụ đánh dấu trang trong cửa sổ mới hay không. Người dùng có thể cấu hình lại trong Firefox. |
| <i>height</i> | Độ cao tính bằng pixel của cửa sổ mới. |
| <i>left</i> | Khoảng cách tính bằng pixel từ cạnh trái của màn hình tới vị trí nơi cửa sổ mới sẽ được đặt. |
| <i>location</i> | Quy định thanh địa chỉ (location bar hay address bar) có được hiển thị hay không. Trong IE7 và các phiên bản mới hơn thanh công cụ luôn được hiển thị, còn Firefox thì có thể thay đổi để luôn hiển thị nó, vì thế tùy chọn này dần trở nên lỗi thời. |
| <i>menubar</i> | Quy định thanh menu có được hiển thị trong cửa sổ mới không. |
| <i>resizable</i> | Quy định người dùng có thể thay đổi kích thước cửa sổ hay không. Firefox luôn cho phép thay đổi kích thước các cửa sổ (thân thiện với người dùng). |
| <i>scrollbars</i> | Quy định thanh cuộn có được hiển thị hay không. |
| <i>status</i> | Quy định thanh trạng thái có được hiển thị trong cửa sổ mới hay không. Người dùng có thể tự cấu hình trong Firefox. |
| <i>toolbar</i> | Quy định thanh công cụ có xuất hiện trong cửa sổ mới hay không. |

| | |
|--------------|---|
| <i>top</i> | Khoảng cách tính bằng pixel từ cạnh trên của màn hình tới vị trí nơi cửa sổ mới được đặt. |
| <i>width</i> | Độ rộng tính bằng pixel của cửa sổ mới. |

Một số trình duyệt cho phép người dùng thiết lập hiệu lực cho các tùy chọn trong Bảng 11-3. Chẳng hạn, ẩn thanh địa chỉ trên cửa sổ mới bằng JavaScript sẽ không hiệu quả trên Internet Explorer hoặc Firefox nếu người dùng đã cấu hình để thanh này luôn hiển thị.

Phương thức *close()* của đối tượng *window* không có tham số. Cách dùng *close()* đơn giản như sau:

```
window.close()
```

Phương thức này không phải lúc nào cũng đáng tin cậy, vì vậy bạn đừng bao giờ mặc định rằng cửa sổ đã thực sự đóng. Hãy chỉ hi vọng nó đã được đóng.

Những kinh nghiệm tốt nhất về việc mở cửa sổ



Mặc dù bạn có thể mở thêm một cửa sổ mới với ít đặc tính hơn (như minh họa trên Hình 11-2) song tôi khuyên bạn không nên làm vậy ngoại trừ những trường hợp đặc biệt.



HÌNH 11-2 Một cửa sổ không có thanh menu và các chức năng khác thường thấy trên cửa sổ trình duyệt.

Bất kỳ cửa sổ mới mở nào cũng nên bao gồm menu, các phần tử điều hướng và thanh địa chỉ. Firefox và IE đều không cho phép JavaScript tắt các tính năng như thay đổi kích thước và các thành phần giao diện như thanh trạng thái. Những tính năng quan trọng này cho phép người dùng sử dụng trang web và ứng dụng để phục vụ nhu cầu của họ hơn là nhu cầu của nhà phát triển. Cách tốt nhất là thiết kế trang web với đầy đủ các tùy chọn sao cho người dùng có thể sử dụng thoải mái mà không bị ảnh hưởng bởi các thành phần giao diện có sẵn.

Bạn sẽ nhận thấy `window.open()` ngày càng trở nên không cần thiết. Với sự ra đời của trình duyệt web theo từng tab, `window.open()` không còn được ưa chuộng. Phần tiếp theo hướng dẫn bạn cách mở một tab mới mà không cần bất kỳ đoạn JavaScript nào.

Mở thêm tab mà không cần đến

JavaScript?

Trên thực tế, bạn không cần dùng JavaScript để mở một tab mới, đây thực sự là điều mà hầu hết các lập trình viên web mong muốn. Thay vào đó, bạn mở tab mới bằng cách sử dụng thuộc tính *target* của thẻ `<a>`. Cách làm này không ảnh hưởng đến thao tác của người dùng trong các trình duyệt như Firefox và IE7 hoặc mới hơn.

Sau đây là ví dụ về hoạt động của thuộc tính *target*:

```
<a target="Microsoft" href="http://www.microsoft.com" id="mslink"> ĐI ĐẾN WEBSITE CỦA MICROSOFT </a>
```

Ví dụ để mở tab mới không tên:

```
<a target="_blank" href="http://www.microsoft.com" id="mslink"> ĐI ĐẾN WEBSITE CỦA MICROSOFT </a>
```

Thay đổi kích thước và di chuyển cửa sổ

JavaScript cũng hỗ trợ thay đổi kích thước cửa sổ trình duyệt. Tuy nhiên, các trình duyệt như Firefox có một tùy chọn để ngăn chặn thay đổi kích thước cửa sổ bằng JavaScript. Vì lý do này, chúng ta không nên thay đổi kích thước cửa sổ bằng JavaScript và tôi sẽ chỉ giới thiệu ngắn gọn các phương thức và thuộc tính để thực hiện thao tác này. Để tìm hiểu thêm thông tin, tham khảo <http://support.microsoft.com/kb/287171>.

Trong Chương 9 mục “Lấy thông tin về màn hình” sẽ trình bày các thuộc tính của đối tượng con screen của window, bao gồm *availHeight* và *availWidth*. Các thuộc tính này đôi khi được sử dụng để hỗ trợ việc thay đổi kích thước cửa sổ trình duyệt. Các phương thức và thuộc tính hữu ích của đối tượng *window* liên quan đến việc di chuyển và thay đổi kích thước cửa sổ được liệt kê trong Bảng 11-4.

BẢNG 11-4 Các thuộc tính và phương thức liên quan đến di chuyển và thay đổi kích thước cửa sổ.

| Thuộc tính/Phương thức | Mô tả |
|------------------------|--|
| <i>moveBy(x, y)</i> | Di chuyển cửa sổ một đoạn x và y pixel theo trục tọa độ tương ứng. |
| <i>moveTo(x, y)</i> | Di chuyển cửa sổ đến tọa độ (x,y). |
| <i>resizeBy(x, y)</i> | Thay đổi kích thước cửa sổ một lượng bằng x, y theo pixel. |

resizeTo(x, y) Thay đổi kích thước cửa sổ thành x,y.

Bộ định thời

JavaScript chứa các hàm được gọi là *bộ định thời (timer)* dùng để đo thời gian xảy ra các sự kiện hoặc ngừng thực thi một đoạn mã trong một khoảng thời gian nhất định.

JavaScript có bốn hàm toàn cục có liên quan tới bộ định thời trong JavaScript:

- *setTimeout()*
- *clearTimeout()*
- *setInterval()*
- *clearInterval()*

Hai hàm cơ bản liên quan đến thiết lập bộ định thời là *setTimeout()* và *setInterval()*. Chúng nhận vào hai đối số: hàm được gọi và khoảng thời gian. Với *setTimeout()*, hàm đối số được gọi khi bộ định thời hết hạn. Với *setInterval()*, hàm đối số được gọi mỗi khi khoảng thời gian hẹn giờ trôi qua. Các hàm trả về một định danh mà bạn có thể sử dụng để xóa hoặc ngừng bộ định thời với hàm *clearTimeout()* và *clearInterval()*.

Các hàm liên quan đến bộ định thời hoạt động theo mili giây chứ không phải giây. Đó là điều cần nhớ khi sử dụng các hàm này. Không có gì tệ hơn việc thiết lập khoảng thời gian là 1 với hy vọng nó sẽ thực thi mỗi giây, chỉ để nhận ra nó sẽ thực thi 1000 lần một giây.

Mách nhỏ 1 giây = 1000 mili giây

Ví dụ 11-2 (có trong file listing11-2.htm của Tài nguyên đi kèm) minh họa cách sử dụng hàm *setTimeout()* để hiển thị một hộp thoại thông báo sau 3 giây.

VÍ DỤ 11-2 Ví dụ về *setTimeout()*.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title> Bộ định thời </title>  
<script type="text/javascript" data-src="eHandler.js"></sc  
</head>  
<body id="mainBody">  
<p> Xin chào </p>  
<script type="text/javascript">  
function sendAlert() {  
    alert("Xin chào");  
}  
function startTimer() {  
    var timerID = window.setTimeout(sendAlert, 3000);  
}  
var mainBody = document.getElementById("mainBody");  
EHandler.add(mainBody, "load", function() { startTimer();  
</script>  
</body>  
</html>
```

Ví dụ 11-2 bao gồm hai hàm `sendAlert()` và `startTimer()`. Sự kiện `onload` của trang gọi hàm `startTimer()`, trong đó có một dòng gọi hàm `setTimeout()`. Hàm `setTimeout()` tiếp tục gọi hàm `sendAlert()` sau 3 giây (3000 mili giây).

Biến `timerID` chứa định danh trả đến hàm `setTimeout()` được gọi. Bạn có thể sử dụng biến `timerID` này để hủy bộ định thời như sau:

```
cancelTimeout(timerID);
```

Hàm `setTimeout()` có thể được gọi bởi đoạn mã JavaScript thô thay vì lời gọi hàm. Tuy nhiên, bạn nên sử dụng lời gọi hàm vì cách dùng mã JavaScript có thể gây ra lỗi trong một số trình duyệt.

Bài tập

1. Tạo trang web chứa hàm xử lý sự kiện `onclick` kết nối đến một liên kết

bằng cách sử dụng sự kiện tại chỗ của DOM 0. Hàm xử lý sự kiện cần hiển thị hộp thoại thông báo "Bạn đã nhấn vào đây".

2. Thay đổi trang web được tạo trong Bài tập 1 để sử dụng kiểu xử lý sự kiện mới bằng cách sử dụng file ehandler.js (Xem Tài nguyên đi kèm), kết nối cùng sự kiện *click/onclick* để hiển thị hộp thoại thông báo đã tạo trong Bài tập 1.
3. Tạo một trang web chứa liên kết đến *http://www.microsoft.com*. Cài đặt để liên kết này được mở trong một tab mới.



Chương 12

Tạo và sử dụng cookie

Sau khi đọc xong chương này, bạn có thể:

- Hiểu rõ về cookie HTTP.
- Tạo cookie bằng JavaScript và gửi cookie tới trình duyệt.
- Nắm được cách thiết lập thời gian hết hạn cho cookie.
- Nắm được cách thiết lập đường dẫn và domain cho cookie.
- Đọc cookie từ trình duyệt và phân tách nội dung của cookie.

Tìm hiểu về cookie



Cookie HTTP (Hypertext Transfer Protocol) là những phần dữ liệu nhỏ được gửi nhận qua lại giữa client (thường là trình duyệt) và server. Cookie được sử dụng để theo dõi các thông tin trạng thái của ứng dụng (ví dụ như vị trí của bạn trong ứng dụng), từ thông tin về phiên làm việc tới các thông tin truy cập như ID người dùng (dù vậy có rất nhiều lý do không nên lưu ID người dùng hoặc các thông tin cá nhân khác trong cookie).

Cookie HTTP được mô tả chi tiết trong RFC 2965 – bạn có thể tìm thấy ở đây nhiều thông tin về cookie hơn cả những gì bạn cần. Bạn có thể tra cứu RFC 2965 tại địa chỉ http://www.rfc-editor.org/cgi-bin/rfcdoctype.pl?loc=RFC&letsgo=2965&type=ftp&file_format=txt.

Cookie và tính riêng tư

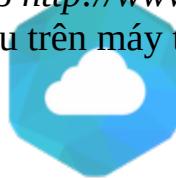
Mặc dù có thể có nhiều rắc rối tính riêng tư liên quan tới cookie, nhưng bản thân các cookie hoàn toàn vô hại. Lý tưởng nhất đó là các cookie được lưu trữ trên bộ nhớ truy cập ngẫu nhiên (RAM) trong khoảng thời gian mà khách truy cập vẫn còn mở trình duyệt. Còn trường hợp xấu nhất đó là

cookie được lưu trữ lâu dài trong các file text trên ổ cứng.

Cookie không gây ra các vấn đề về tính riêng tư ngoại trừ vấn đề liên quan đến thông tin lưu trong chúng. Có một thực tế là người quản trị trang web hoàn toàn có thể lưu trữ thông tin nhạy cảm trong cookie và họ cũng có thể lưu dữ liệu đó theo những cách không an toàn khác ngoài cookie. Cookie hoàn toàn vô hại - vấn đề là do người dùng sử dụng cookie để lưu dữ liệu nhạy cảm. Thông tin lưu trong cookie cũng được bảo mật tương tự như dữ liệu lưu trên máy tính. Do đó, tôi khuyên bạn đừng bao giờ lưu trữ thông tin xác thực cá nhân trong cookie. Hãy đảm bảo an toàn cho dữ liệu truy cập của bạn.

Cookie là một file text thông thường – định nghĩa này là cách đơn giản và cũng là thuận tiện nhất để hình dung về cookie. Cookie có thể chứa một vài phần tử khác nhau, nhưng phần tử chính của cookie là tập hợp các cặp *ten/giatri*, đa số do người quản trị hoặc lập trình viên thiết lập và là tùy chọn không bắt buộc.

Khi khách truy cập thăm trang web <http://www.braingia.org/> và vào đường liên kết tới blog, cookie có thể được lưu trên máy tính của người đó. Cookie này sẽ có nội dung tương tự như sau:



Name: cookie4blog
Content: sess_id02934235
Domain: www.braingia.org
Path: /
Send For: Any type of connection
Expires: Never

Trình duyệt lưu trữ cookie trên máy tính của khách truy cập cho tới khi cookie đó hết hạn (trong trường hợp trên, cookie không bao giờ hết hạn). Nếu vẫn người đó đó thăm lại trang web, trình duyệt sẽ gửi cookie tới server. Khi đó, server có thể nhận ra người dùng và sử dụng một số thiết lập mà người dùng từng chọn để tùy biến trải nghiệm của họ.

Một trong những đặc tính của cookie đó là cookie chỉ được gửi tới server nằm trên domain mà cookie đó được thiết lập. Vì vậy, cookie trong đoạn mã trước sẽ chỉ được gửi tới server khi domain mà trình duyệt ghé thăm là www.braingia.org. Ngoài ra, cookie có thể được thiết lập cờ *secure* (cookie trong ví dụ trên không có cờ *secure*). Nếu cờ *secure* được thiết lập, cookie chỉ có thể được gửi tới server thông qua một phiên làm việc sử dụng giao thức SSL (Secure

Sockets Layer), ví dụ như thông qua một kết nối HTTPS (Hypertext Transfer Protocol Secure).

Chú ý Các cookie bên thứ ba và thủ thuật để gửi nhận cookie ngoài domain không thuộc phạm vi thảo luận của cuốn sách này.

JavaScript vừa có thể tạo vừa có thể đọc cookie. Phần còn lại của chương này sẽ đề cập tới hai chức năng trên.

Tạo cookie với JavaScript

Bạn có thể dùng JavaScript để tạo cookie bằng cách sử dụng thuộc tính `document.cookie`. Phần này sẽ hướng dẫn cách tạo và gửi cookie tới trình duyệt của khách truy cập.

 **Mách nhỏ** Khi làm việc với cookie, việc sử dụng các giá trị chuỗi đơn giản rất quan trọng. Hãy tránh sử dụng ký tự trắng, dấu chấm câu cũng như các ký tự đặc biệt khác vì những ký tự đó không được phép. Bạn có thể sử dụng những ký tự không hợp lệ trên nhưng không phải ở dạng thông thường – bạn phải thực hiện thoát các ký tự đó, nếu không, chúng có thể gây ra lỗi cho cookie, trang web và trình duyệt. Nói chung cũng giống như các lỗi JavaScript và lỗi lập trình web khác, lỗi do các ký tự trên có thể rất tinh vi và thường khó phát hiện. Khi chưa chắc chắn, bạn chỉ nên sử dụng các ký tự chữ và số thông dụng.

Tìm hiểu một cookie đơn giản

Cookie phải có tên. Đoạn mã JavaScript ngắn dưới đây thực hiện tạo và gửi một cookie tới trình duyệt web:

```
var cookName = "testcookie";
var cookVal = "testvalue";
var myCookie = cookName + "=" + cookVal;
document.cookie = myCookie;
```

Khi bạn truy cập trang web chứa đoạn mã JavaScript trên, đoạn mã sẽ thiết lập một cookie có tên là *testcookie* trong trình duyệt của bạn. Giả sử bạn chấp nhận cookie này, khi đó trình duyệt sẽ lưu dữ liệu (*testvalue*) cũng như các thông tin khác về cookie để phục vụ cho mục đích sử dụng trong tương lai.

Khi sử dụng JavaScript để thiết lập cookie, cookie đó sẽ được thêm vào sau một cookie có sẵn bất kỳ. Bạn phải để ý tới đặc điểm này khi đọc cookie. Cần biết rằng, để thiết lập các thuộc tính trong một cookie cụ thể, bạn phải nối cặp *ten/giatri* của thuộc tính đó vào cookie khi tạo cookie. Ví dụ tiếp theo sẽ giúp bạn thấy rõ hơn khía cạnh này khi làm việc với cookie trong JavaScript.

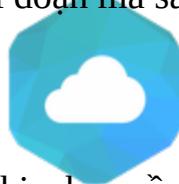
Thiết lập thời gian hết hạn cho cookie

Ví dụ trình bày ở phần trước đã tạo ra một cookie thông qua cặp *ten/giatri*:

```
var myCookie = cookName + "=" + cookVal;
```

Về cơ bản, đoạn mã trên giống với đoạn mã sau (cookie này chỉ có tên và tập dữ liệu):

```
testcookie=testvalue;
```



Để thêm thời gian hết hạn cho cookie, bạn cần thêm thuộc tính *expires*, lúc này cookie sẽ trở thành:

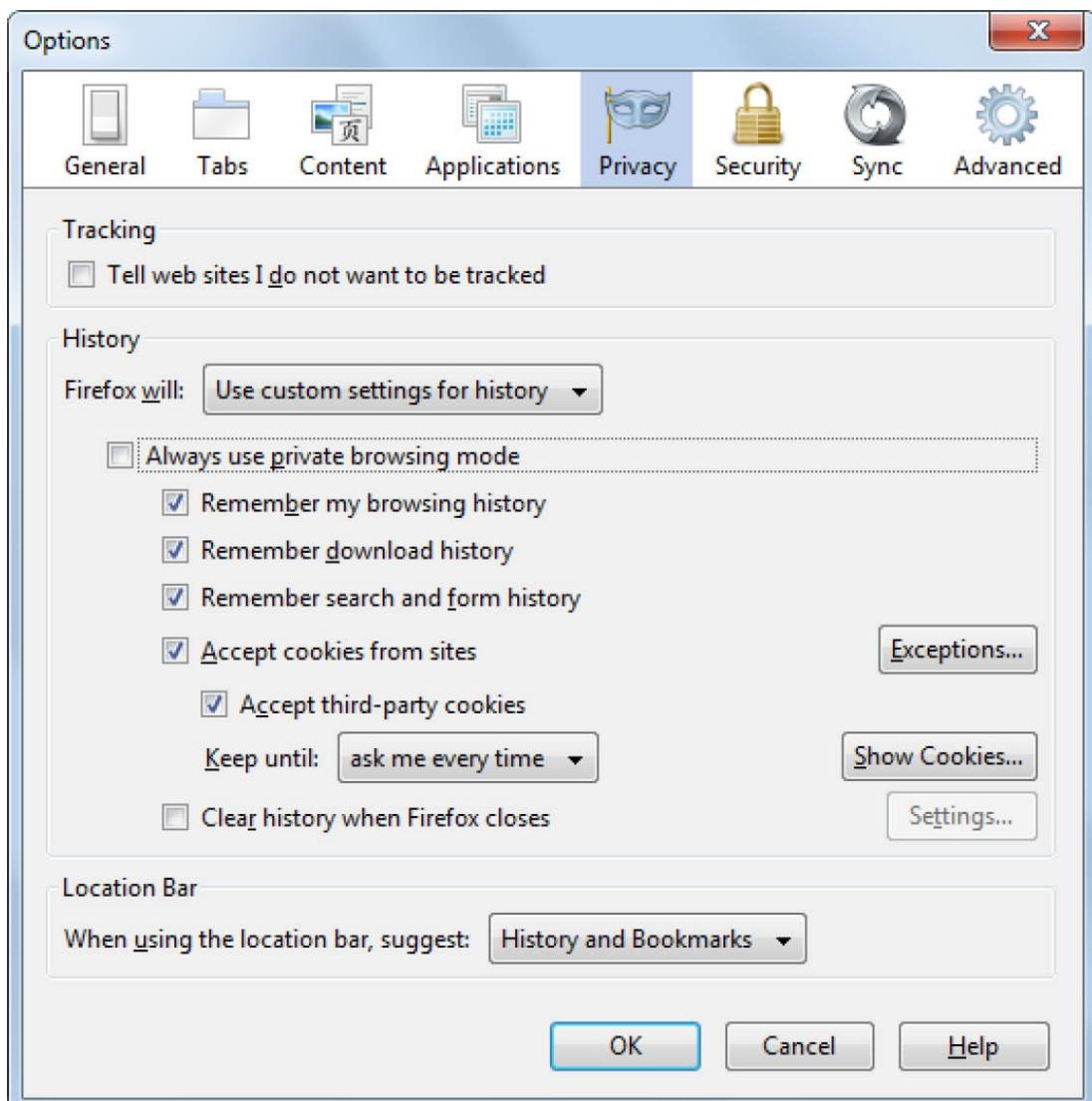
```
testcookie=testvalue; expires=Sat, 12 Mar 2011 17aad:51:50 GI
```

Việc thêm thời gian hết hạn cho cookie rất đơn giản, bạn chỉ cần gán thời gian hết hạn đó vào cuối cookie được gửi đi. Định dạng của thời gian hết hạn rất quan trọng. Để định dạng chính xác thời gian hết hạn, bạn cần sử dụng một số hàm JavaScript. Trong bài tập tiếp theo, bạn sẽ thực hành tạo và thiết lập thời gian hết hạn cho cookie.

Trước khi bắt đầu bài tập, bạn nên kích hoạt hộp thoại để hiển thị cookie trong trình duyệt. Điều này giúp việc gỡ lỗi trở nên dễ dàng hơn, vì mỗi khi server hay mã JavaScript gửi cookie tới trình duyệt, bạn sẽ được thông báo nội dung của cookie đó.

Tuy nhiên việc kích hoạt hộp thoại trong Windows Internet Explorer bị coi là không an toàn, do đó với bài tập này, bạn cần sử dụng Firefox. Trong Firefox

phiên bản Windows, nhấn vào menu Tools, sau đó chọn Options (trong phiên bản Firefox cho Linux, nhấn vào menu Edit và chọn Preferences). Trong hộp thoại Options, nhấn vào biểu tượng Privacy, sau đó chọn Use Custom Settings For History (sử dụng các thiết lập tùy chỉnh về lịch sử duyệt web) từ danh sách thả xuống. Bây giờ chọn Ask Me Every Time từ danh sách thả xuống Keep Until, giống như trong Hình 12-1.



HÌNH 12-1 Thay đổi thiết lập cookie trong Firefox.

Mách nhỏ Đừng quên thay đổi những thiết lập này về trạng thái ban đầu sau khi kiểm tra xong các vấn đề liên quan tới cookie. Những hộp thông báo liên tục có thể gây khó chịu vì hầu hết các trang web sử dụng một số lượng cookie nhất định.

Thêm thời gian hết hạn cho cookie

1. Sử dụng Microsoft Visual Studio, Eclipse, hoặc một trình soạn thảo khác chỉnh sửa file cookie-expire.htm trong thư mục mã nguồn mẫu Chương 12, phần Tài nguyên đi kèm.
2. Trong trang này, bổ sung thêm đoạn mã in đậm dưới đây (đoạn mã này có trong file cookie-expire.txt và bản hoàn chỉnh có trong file cookie.htm – hai file này đều nằm trong Tài nguyên đi kèm của cuốn sách):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Xin chào Cookie</title>
<script type="text/javascript">
var cookName = "testcookie";
var cookVal = "testvalue";
var myCookie = cookName + "=" + cookVal;
document.cookie = myCookie;
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>
```

3. Cài đặt trang web này lên một server, sau đó sử dụng trình duyệt web để mở, bạn sẽ nhận được một hộp thoại thông báo như sau:



Trong Firefox, nhấn vào nút Deny để đảm bảo rằng cookie này không được

lưu trên trình duyệt. Nếu bạn vô tình chấp nhận cookie này, hãy đóng và mở lại trình duyệt. Do đây là cookie theo phiên làm việc, nên cookie đó sẽ bị xóa khi bạn thoát khỏi trình duyệt. Xin chúc mừng – bạn đã hoàn thành xong bài tập sử dụng JavaScript để gửi một cookie tới trình duyệt!

4. Sửa lại đoạn mã trên để thêm giá trị thời gian hết hạn. Đoạn mã này phải được đặt trước dòng khai báo biến *myCookie* giống như sau:

```
var date = new Date();
date.setTime(date.getTime() + 604800000)
var expireDate = date.toGMTString();
```

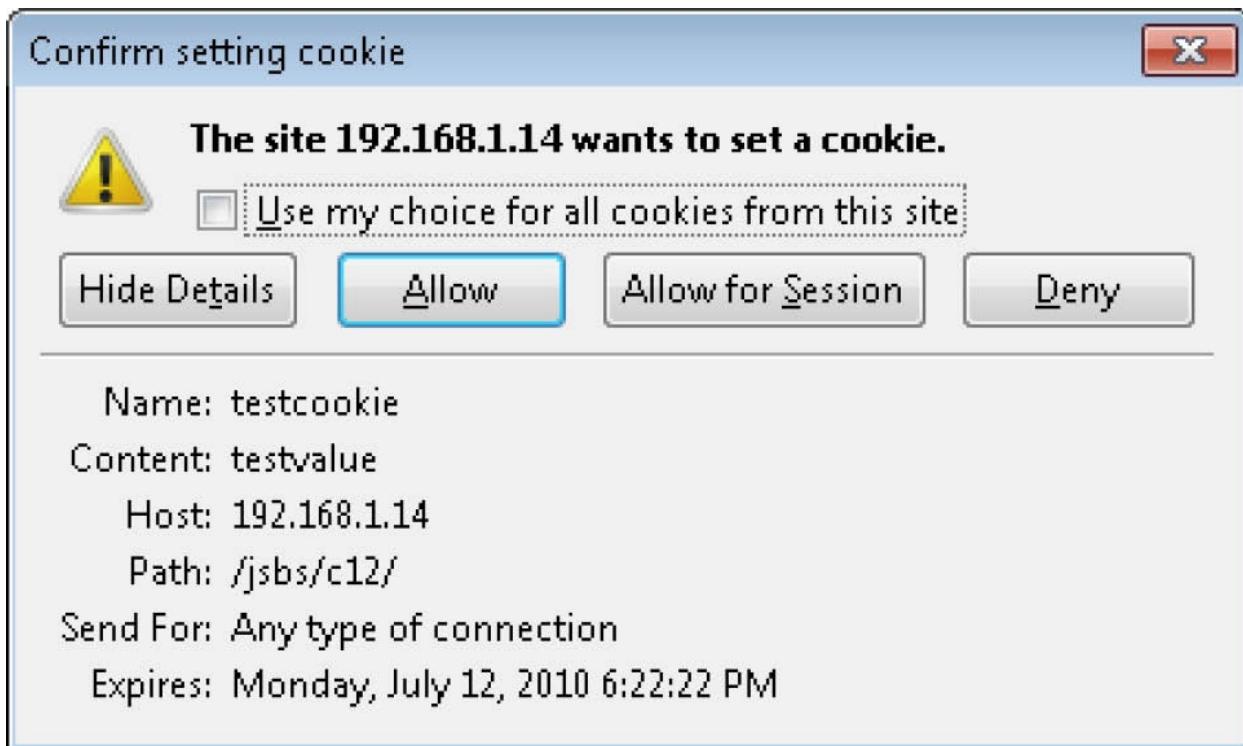
5. Sửa lại dòng khai báo biến *myCookie* để thêm thời gian hết hạn cho cookie:

```
var myCookie = cookName + "=" + cookVal + ";expires=" + e;
```

6. Toàn bộ đoạn mã lúc này sẽ giống như sau (các dòng bổ sung được in đậm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Xin chào Cookie</title>
<script type="text/javascript">
var cookName = "testcookie";
var cookVal = "testvalue";
var date = new Date();
date.setTime(date.getTime() + 604800000)
var expireDate = date.toGMTString();
var myCookie = cookName + "=" + cookVal + ";expires=" + e;
document.cookie = myCookie;
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>
```

7. Khi xem trên trình duyệt, đoạn mã JavaScript của trang web sẽ gửi tới trình duyệt một cookie cùng với thời gian hết hạn của cookie đúng bằng một tuần kể từ ngày bạn truy cập tới trang web. Hộp thoại bạn nhìn thấy sẽ giống như hình sau đây. Nhấn vào nút Show Details để mở rộng hộp thoại nếu cần:



Ở bài tập này, bốn dòng mã đã được thêm và điều chỉnh để thiết lập thời gian hết hạn cho cookie, điều này cho phép cookie được lưu trữ lâu hơn thay vì chỉ tồn tại trong thời gian hoạt động của trình duyệt. Ba dòng mã đầu tiên thiết lập giá trị thời điểm hết hạn. Dòng đầu tiên tạo ra một giá trị ngày tháng và lưu giá trị đó vào biến *date*. Tiếp theo, đoạn mã thiết lập giá trị ngày giờ bằng phương thức *setTime()*. Tham số của phương thức *setTime()* là một biểu thức trong đó gọi tới phương thức *getTime()*. Phương thức *getTime()* trả về giá trị ngày giờ hiện tại theo đơn vị mili giây, tính từ ngày 1/1/1970 . Giá trị hàm *getTime()* biểu diễn cho ngày giờ hiện tại, do đó để xác định thời điểm hết hạn cho cookie (trong trường hợp này là 1 tuần), bạn phải tính số giây trong 1 tuần (604.800) sau đó nhân số đó với 1000 để chuyển thời gian sang đơn vị mili giây. Cộng kết quả đó (604.800.000) với giá trị của hàm *getTime()*. Tiếp theo, bạn có thể chuyển kiểu của số này sang định dạng chuỗi GMT (Greenwich Mean Time) bằng cách sử dụng phương thức *toGMTString()* của đối tượng *date*.

Cuối cùng, đoạn mã thêm thuộc tính *expires* vào cookie. Kết quả cuối cùng sẽ như sau:

```
testcookie=testvalue;expires=Mon, 12 Jul 2010 23:22:22 GMT
```

Thiết lập đường dẫn cho cookie

Trong các ví dụ đã trình bày cho tới nay, trên máy tính chạy thử, trình duyệt gửi cookie tới server khi đường dẫn (path) của yêu cầu HTTP đúng bằng `/jsbs/c12/` vì server này chỉ phục vụ các trang web từ môi trường thử nghiệm này. Đường dẫn này sẽ khác trong các trường hợp khác, ví dụ như trong môi trường của bạn hoặc trên máy tính của bạn. Bạn có thể thay đổi đường dẫn này bằng cách thêm một tùy chọn nữa khi thiết lập cookie. Một cách làm phổ biến là thiết lập đường dẫn với dấu gạch chéo (`/`) để tất cả các yêu cầu của domain ban đầu đều có thể truy cập tới cookie đó.

Giống như tùy chọn `expires`, việc thiết lập tùy chọn đường dẫn đòi hỏi phải gán một cặp `ten/giatri` cho cookie.

Dưới đây là trang web giống với trang web ở các ví dụ đã xét, trang này chứa đoạn mã xác định thời gian hết hạn và thêm tùy chọn mới cho đường dẫn của cookie (đoạn mã này được in đậm và nằm trong file cookie-path.htm của Tài nguyên đi kèm):

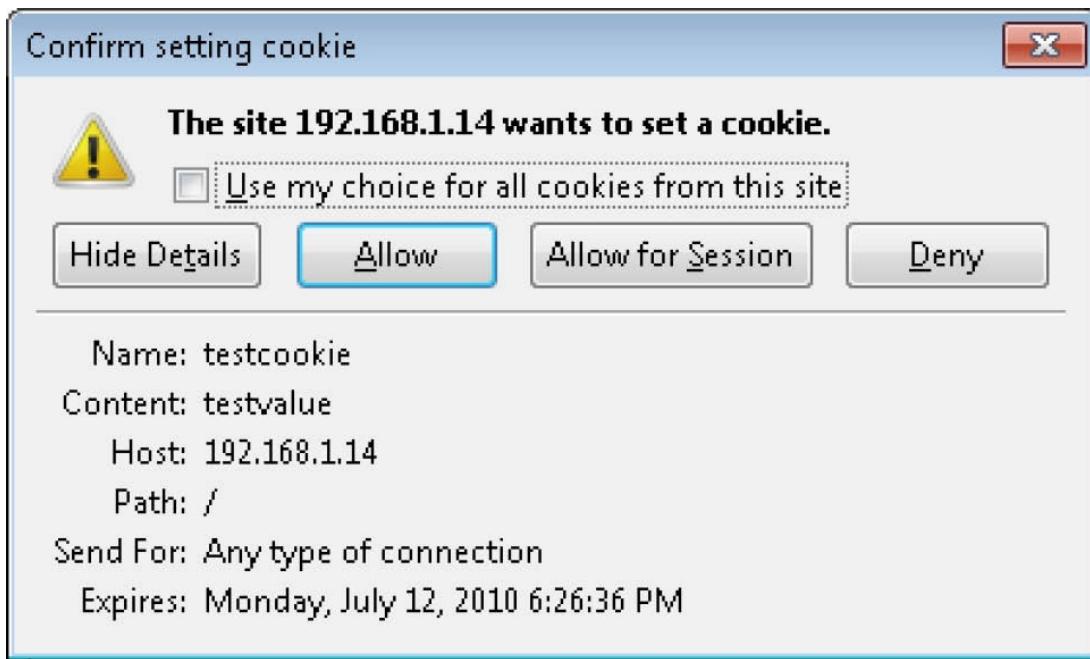
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Xin chào Cookie</title>
<script type="text/javascript">
var cookName = "testcookie";
var cookVal = "testvalue";
var date = new Date();
date.setTime(date.getTime() + 604800000)
var expireDate = date.toGMTString();
var path = ";path=/";
var myCookie = cookName + "=" + cookVal + ";expires=" + expireDate;
document.cookie = myCookie;
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>
```

Đoạn mã trên thêm một dòng mới cho tính năng đường dẫn và thay đổi định

nghĩa của biến *myCookie* bằng việc cộng thêm biến đường dẫn mới (biến path). Với đoạn mã mới trên, cookie được tạo ra sẽ giống như sau:

```
testcookie=testvalue;expires=Mon, 12 Jul 2010 23:22:22 GMT;path=/
```

Mở trang web vừa chỉnh sửa, bạn sẽ thấy một hộp thoại cookie tương tự như Hình 12-2



Thiết lập domain cho cookie

Các ví dụ từ trước tới nay chưa thiết lập thuộc tính *domain* cho cookie; theo mặc định, thuộc tính *domain* thiết lập host và domain cho server gửi cookie tới trình duyệt (tức server gửi đoạn mã JavaScript chứa cookie). Trong các ví dụ trên, domain là www.braigia.org. Tuy nhiên, nhiều trang (bao gồm cả www.braigia.org) có thể có nhiều host để phục vụ nội dung HTTP. Ví dụ, có thể có một server hoàn toàn riêng biệt tên là images.braigia.org chỉ để cung cấp hình ảnh cho trang [braingia.org](http://www.braigia.org). Sẽ rất thuận tiện khi thiết lập domain là [braingia.org](http://www.braigia.org) để các cookie có thể được dùng chung trên toàn bộ *domain*.

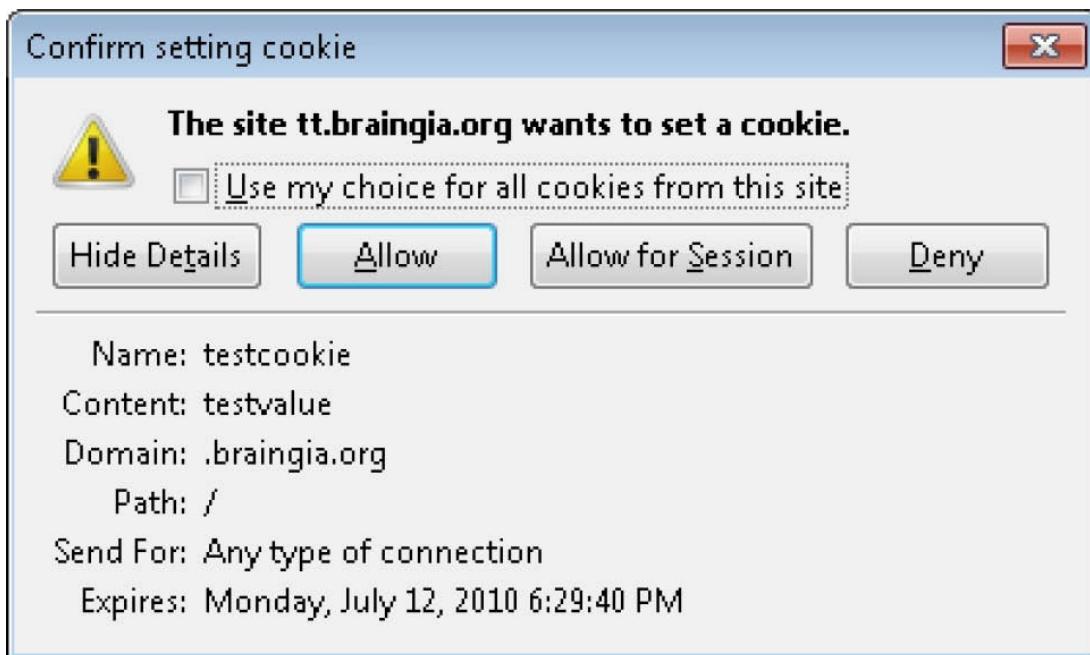
Việc thiết lập domain cho cookie cũng tương tự như thiết lập đường dẫn. Gán tùy chọn domain cho cookie sẽ thiết lập cho domain một giá trị xác định. Dưới đây là trang web tích hợp các thuộc tính đường dẫn, thời gian hết hạn và domain

cho cookie. Đoạn mã chỉ định và thêm thuộc tính *domain* vào biến *myCookie* được in đậm (và nằm trong file cookie-domain.htm của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Xin chào Cookie</title>
<script type="text/javascript">
var cookName = "testcookie";
var cookVal = "testvalue";
var date = new Date();
date.setTime(date.getTime() + 604800000)
var expireDate = date.toGMTString();
var path = ";path=/";
var domain = ";domain=braingia.org";
var myCookie = cookName + "=" + cookVal + ";expires=" + expireDate;
document.cookie = myCookie;
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>
```



Khi dùng trình duyệt mở trang web này, một hộp thoại như Hình 12-3 sẽ xuất hiện.



HÌNH 12-3 Thiết lập thuộc tính domain của cookie sao cho cookie đó có thể được đọc bất cứ đâu trong domain braingia.org

Mách nhỏ Bạn có thể thiết lập tên domain cụ thể cho cookie để tạo ra các cookie chỉ đọc được từ các domain *images.braingia.org* hoặc *someotherspecificcomputer.braingia.org*. Tuy nhiên, bạn không thể thiết lập một domain bên ngoài domain đang phục vụ nội dung trang web. Ví dụ, bạn không thể thay đổi giá trị domain của cookie thành *microsoft.com* nếu cookie đó đang được gửi đi bởi domain *braingia.org*. Trình duyệt sẽ bỏ qua những cookie như vậy.

Thiết lập bảo mật cho cookie

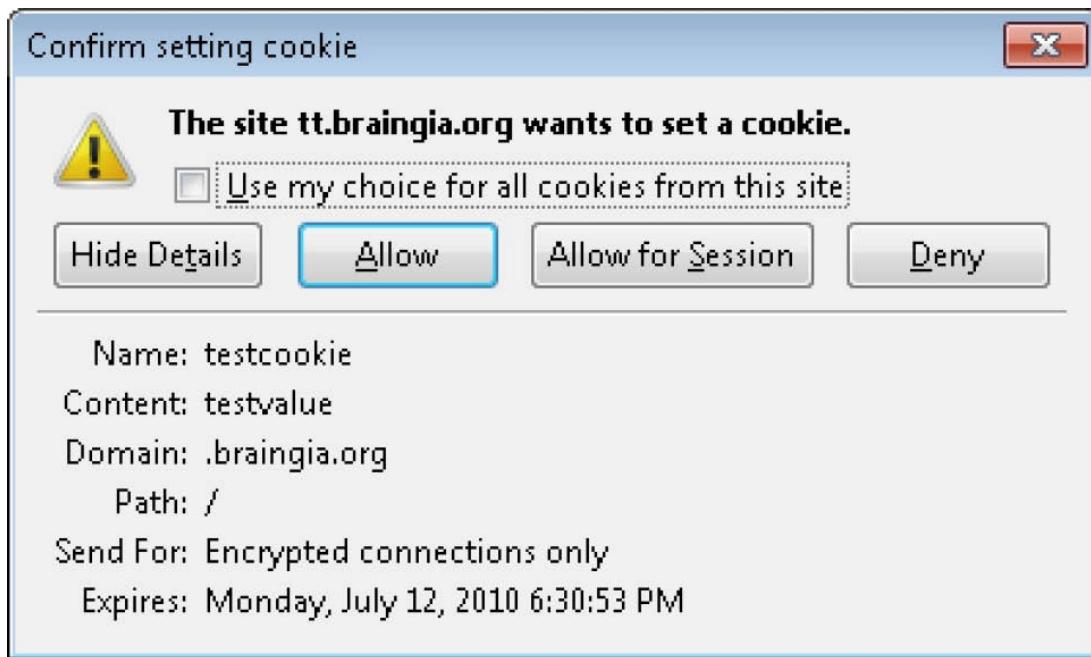
Việc thiết lập cờ *secure* trong một cookie biểu thị rằng cookie đó chỉ được gửi khi kết nối sử dụng SSL, ví dụ như kết nối HTTPS.

Đoạn mã để thêm cờ *secure* cho biến *myCookie* sẽ như sau:

```
var myCookie = cookName + "=" + cookVal + ";expires=" + expi
```

Khi dùng trình duyệt mở trang web này, ngay cả với kết nối HTTP chưa được mã hóa, đoạn mã JavaScript trên sẽ tạo ra một cookie như được hiển thị trong hộp thoại như trong Hình 12-4. Chú ý rằng tùy chọn Send For hiện là “*Encrypted connections only*” (Chỉ các kết nối mã hóa), trong khi các ví dụ

trước, giá trị này được thiết lập là “*Any type of connection*” (Bất kỳ kiểu kết nối nào).



HÌNH 12-4 Thiết lập cờ secure cho cookie.

Tới lúc này, bạn đã tìm hiểu tất cả những thứ cần thiết để tạo cookie bằng JavaScript. Phần cuối chương sẽ giúp bạn tìm hiểu cách đọc nội dung cookie với JavaScript.

Đọc nội dung của cookie với JavaScript

Tới lúc này, bạn đã tìm hiểu đoạn mã thực hiện việc gửi cookie tới trình duyệt khi bạn truy cập một trang web. Khi dùng trình duyệt truy cập tới trang web mà domain và đường dẫn của trang này khớp với cookie lưu trên máy client, trình duyệt sẽ gửi tất cả các cookie trùng khớp về server cùng với yêu cầu duyệt trang web. Mã JavaScript của trang web có thể truy cập cookie khi nội dung trang web được chuyển tới trình duyệt.

Vìệc đọc cookie trong mã JavaScript bao gồm việc lấy các cookie từ đối tượng *document.cookie*, sau đó *phân tách* những cookie đó thành các phần nhỏ có

thể quản lý được. Để thực hiện điều này, ta sẽ dùng tới phương thức *split* của đối tượng *document.cookie* vì các cookie được phân tách bởi dấu chấm phẩy, tương tự như cách thức phân tách thuộc tính của cookie (Xem lại mô tả trong phần trước). Dưới đây là ví dụ:

```
var incCookies = document.cookie.split(";;");
```

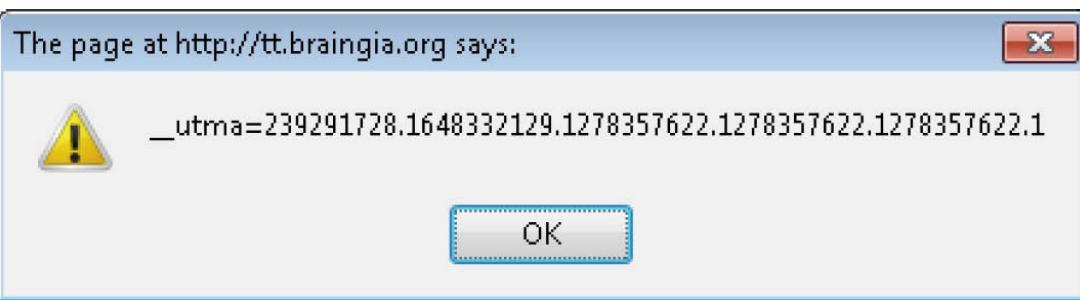
Với đoạn mã JavaScript ngắn này, tất cả các cookie sẽ được phân tách và sẵn sàng chờ truy cập. Với việc sử dụng vòng lặp for, bạn có thể duyệt qua tất cả các cookie sẵn dùng cho domain để tìm ra tên và dữ liệu của từng cookie, như sau đây (Chú ý rằng đoạn mã này không kiểm tra các thuộc tính của cookie):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Đọc Cookie</title>
<script type="text/javascript">
var incCookies = document.cookie.split(";;");
var cookLength = incCookies.length;
for (var c = 0; c < cookLength; c++) {
    alert(incCookies[c]);
}
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>
```

Đoạn mã trên (có trong file readcookie.htm) đọc các cookie có sẵn trên domain và gán vào biến *incCookies*. Các cookie này được phân tách bởi dấu chấm phẩy. Ví dụ, tôi đã thiết lập một số cookie cho domain *braingia.org* để phục vụ cho hoạt động thông thường của trang web và các ví dụ trong chương này. Bạn có thể dùng đoạn mã trên để đọc tất cả các cookie cho domain này. Hình 12-5 và Hình 12-6 hiển thị hai hộp thoại thông báo kết quả.



HÌNH 12-5 Cookie thứ nhất được đọc bởi vòng lặp.



HÌNH 12-6 Cookie thứ hai được đọc bởi vòng lặp.

Sử dụng vòng lặp *for*, bạn có thể chia nhỏ các cookie chứa dấu bằng (=) để tách phần tên và phần dữ liệu của cookie giống như đoạn mã dưới đây.

```
var incCookies = document.cookie.split(";");
var cookLength = incCookies.length
for (var c = 0; c < cookLength; c++) {
    var pairs = incCookies[c].split "=";
    var cookieName = pairs[0];
    var cookieValue = pairs[1];
    alert("Tên cookie: " + cookieName + " - " + "Giá trị
          + cookieValue);
}
```

Trong đoạn mã này (đoạn mã có trong file readcookie2.htm của Tài nguyên đi kèm), mỗi cookie *incCookie[c]* được phân tách tại vị trí dấu bằng (=) và đặt vào trong biến *pairs*. Chỉ số đầu tiên (0) của biến *pairs* là tên của cookie, chỉ số thứ hai (1) là dữ liệu của cookie đó. Hình 12-7 là kết quả của đoạn mã trên.



HÌNH 12-7 Phân tách cặp tên/giá trị tương ứng với tên và dữ liệu của cookie.

Loại bỏ cookie

Không có phương thức tích hợp sẵn để loại bỏ cookie bằng JavaScript hoặc cách khác. Để loại bỏ cookie, đơn giản chỉ cần xóa giá trị của nó và thiết lập thời điểm hết hạn của cookie đó bằng một thời điểm trong quá khứ.

Đoạn mã dưới đây đã được sử dụng trong ví dụ ở phần trước để tạo và thiết lập dữ liệu cho cookie:

```
var cookName = "testcookie";
var cookVal = "testvalue";
var date = new Date();
date.setTime(date.getTime() + 604800000)
var expireDate = date.toGMTString();
var path = ";path=/";
var domain = ";domain=braingia.org";
var myCookie = cookName + "=" + cookVal + ";expires=" + expireDate;
document.cookie = myCookie;
```

Bạn có thể xóa cookie này bằng cách thiết lập thời điểm hết hạn bằng một thời điểm trong quá khứ. Chú ý rằng các thành phần như *name*, *path*, và *domain* phải trùng với cookie đã được thiết lập. Cụ thể, bạn cần ghi đè cookie đang tồn tại bằng một cookie đã hết hạn như dưới đây:

```
var cookName = "testcookie";
var cookVal = "";
var date = new Date();
date.setTime(date.getTime() - 60)
var expireDate = date.toGMTString();
```

```
var path = ";path=/";
var domain = ";domain=braingia.org";
var myCookie = cookName + "=" + cookVal + ";expires=" + expi
document.cookie = myCookie;
```

Đoạn mã này (xem trong file cookie-delete.htm của phần Tài nguyên đi kèm) khiến dữ liệu của `testcookie` bị thiết lập bằng rỗng và thời gian hết hạn được thiết lập bằng một thời điểm trong quá khứ.

Chú ý Khi đoạn mã JavaScript trên thực thi, Firefox sẽ ngay lập tức xóa cookie có tên là `testcookie` đi; tuy nhiên, một số trình duyệt khác vẫn giữ lại cookie này cho tới khi đóng trình duyệt.

Bài tập

1. Tạo một trang web để gửi cookie tới trình duyệt. Thiết lập thời gian hết hạn của cookie đó tại thời điểm sau một ngày so với ngày giờ hiện tại. Kiểm tra lại xem đoạn mã JavaScript đã gửi cookie tới trình duyệt hay chưa bằng cách xem cookie trong khi cookie đó **đã** được thiết lập hoặc sau khi cookie được lưu trên máy tính. Bạn có thể thực hiện yêu cầu thứ hai của bài tập này bằng cách sử dụng JavaScript hoặc xem trong cookie lưu trên máy tính.
2. Tạo một trang web gửi cookie có thời gian hết hạn là một tuần và thiết lập cờ `secure`. Trang web này có thể giống như trang web trong Bài tập 1, nhưng phải đảm bảo cookie được gán một tên khác để hai cookie trong mỗi bài tập là hoàn toàn riêng biệt. Tương tự, hãy đảm bảo rằng chỉ có cờ `secure` của cookie trong bài tập này được kích hoạt còn cookie trong Bài tập 1 thì không.
3. Tạo một trang web để đọc cookie có thiết lập cờ `secure`. Bạn có nhận được cookie đó hay không? Nếu không, bạn cần làm gì để nhận được cookie đó?
4. Tạo một trang web để đọc cookie đã tạo trong Bài tập 1. Sử dụng vòng lặp `for` và câu lệnh điều kiện `if` để hiển thị hộp thoại `alert()` khi cookie có tên đúng như cookie trong Bài tập 1 được tìm thấy trong vòng lặp. Không hiển thị hộp thoại `alert()` đối với các cookie khác.

Chương 13

Làm việc với hình ảnh trong JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Nắm được các phương pháp tạo hiệu ứng rollover mới và cũ bằng JavaScript.
- Tải trước ảnh với JavaScript.
- Tạo slideshow ảnh.
- Cải tiến bản đồ ảnh với JavaScript.

Tạo hiệu ứng Rollover cho ảnh



Thuật ngữ hiệu ứng *rollover của ảnh* (*image rollover*) chỉ hiệu ứng làm thay đổi hình ảnh khi người dùng di chuột lên nó, đây là hiệu ứng cung cấp phản hồi trực quan về vị trí chuột trên màn hình. Mặc dù kỹ thuật này hầu như đã được thay thế bằng cách sử dụng CSS (cascading style sheets), nhưng vì đây là cuốn sách về JavaScript nên tôi sẽ trình bày cách tạo hiệu ứng rollover bằng ngôn ngữ này. Ngoài ra, ngay cả khi bạn sử dụng CSS thì kiến thức tạo hiệu ứng rollover bằng JavaScript luôn có ích cho bạn trong công việc.

Hiệu ứng rollover được kích hoạt dựa trên các sự kiện liên quan tới việc di chuột trên màn hình, chủ yếu là sự kiện *mouseover* và *mouseout*.

Hiệu ứng rollover đơn giản

Đặt hàm xử lý sự kiện *mouseover* và *mouseout* vào thẻ *img* để tạo các hiệu ứng rollover. Hàm xử lý các sự kiện này sẽ làm hiển thị những hình ảnh khác

nhau. Đoạn mã HTML dưới đây tạo hiệu ứng rollover bằng mô hình xử lý sự kiện DOM:

```
*. Đầu tiên hãy liên kết tới đoạn mã JavaScript eHandler.js:

```
<script type="text/javascript" data-src="eHandler.js"></script>
```

Tiếp theo, gọi hàm *rollover* để đáp ứng lại sự kiện *onLoad* bằng phương thức *add* của đối tượng *EHandler*. Thêm đoạn mã sau vào cuối đoạn mã HTML:

```
<script type="text/javascript">
var bodyEl = document.getElementsByTagName('body')[0];
EHandler.add(bodyEl, "load", function() { rollover(); });
</script>
```

Mặc dù có tính năng tương tự như ví dụ trước, nhưng đoạn mã trong Ví dụ 13-1 chưa thật sự linh động. (Bạn sẽ cải thiện đoạn mã này trong phần sau). Dưới đây là diễn giải cho đoạn mã này.

Hàm *rollover* truy xuất tất cả các phần tử *<img>* bằng cách sử dụng phương thức *getElementsByName()* của đối tượng *document* và đặt những phần tử đó vào biến *images*. Tiếp theo, đoạn mã truy xuất ra số phần tử chứa trong biến *images* sử dụng thuộc tính *length* và lưu kết quả trong biến *imgLength*. Đoạn mã sử dụng giá trị trong biến *imgLength* để duyệt qua tất cả các ảnh. Trong vòng lặp, các sự kiện *mouseover* và *mouseout* được liên kết tới các hàm xử lý tương ứng, đó là *mouseOver()* và *mouseOut()*.

Đoạn mã trên chưa hoàn toàn linh động là bởi vì trong các hàm xử lý sự kiện *mouseOver* và *mouseOut*, thuộc tính *src* được gán cố định bằng các tên file *box1.png* và *box2.png*. Điều này không ảnh hưởng gì nếu như bạn chỉ có một ảnh và một ảnh thay thế. Tuy nhiên, trên thực tế, một trang web thường chứa nhiều ảnh khác nhau nên đoạn mã trên không thể áp dụng được. Ngoài ra, từ khóa *this* dễ gây ra sự nhầm lẫn khi được gọi trong một hàm khác có liên quan tới hàm xử lý sự kiện. Vì thế, đoạn mã này cần được cải tiến.

Bài tập tiếp theo minh họa lý cho thuyết cơ bản để tạo ra hiệu ứng rollover. Vòng lặp duyệt qua các ảnh (được truy xuất bằng ID) và liên kết sự kiện *mouseover* và *mouseout* của những ảnh đó tới các hàm cụ thể, sau đó những hàm này sẽ gán thuộc tính *src* bằng tên của ảnh sử dụng cho sự kiện tương ứng. Bây giờ bạn sẽ làm cho hàm này trở nên linh hoạt hơn để có thể sử dụng lại trong tương lai.

Bài tập tiếp theo minh họa lý cho thuyết cơ bản để tạo ra hiệu ứng rollover. Vòng lặp duyệt qua các ảnh (được truy xuất bằng ID) và liên kết sự kiện *mouseover* và *mouseout* của những ảnh đó tới các hàm cụ thể, sau đó những hàm này sẽ gán thuộc tính *src* bằng tên của ảnh sử dụng cho sự kiện tương ứng. Bây giờ bạn sẽ làm cho hàm này trở nên linh hoạt hơn để có thể sử dụng lại trong tương lai.

## Tạo hiệu ứng rollover linh động

1. Dùng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác chỉnh sửa lại file *rollover.htm* trong thư mục Chương 13*rollover* của Tài nguyên đi kèm. Thư mục này chứa sáu ảnh: *homedefault.png*, *homeover.png*, *aboutdefault.png*, *aboutover.png*, *blogdefault.png* và *blogover.png*. Những file có tên chứa chuỗi *\_default* là các ảnh để hiển thị lúc đầu; những file có tên chứa chuỗi *\_over* là các ảnh thay thế.
2. Thay thế các đoạn mã được in đậm dưới đây vào các dòng chú thích TODO trong file *rollover.htm* (đoạn mã in đậm này có thể tìm thấy trong file *rollover.htm*).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Rollover</title>
<script type="text/javascript" data-src="eHandler.js"></script>
<script type="text/javascript">
function rollover() { //hàm tạo hiệu ứng rollover
 var images = document.getElementsByTagName("img");
 var imgLength = images.length;
 for (var i = 0; i < imgLength; i++) {
 EHandler.add(images[i],"mouseover", function()
 {images[i].src = "aboutover.png";});
 EHandler.add(images[i],"mouseout", function()
 {images[i].src = "aboutdefault.png";});}
}
</script>
<body>

</body>
</html>
```

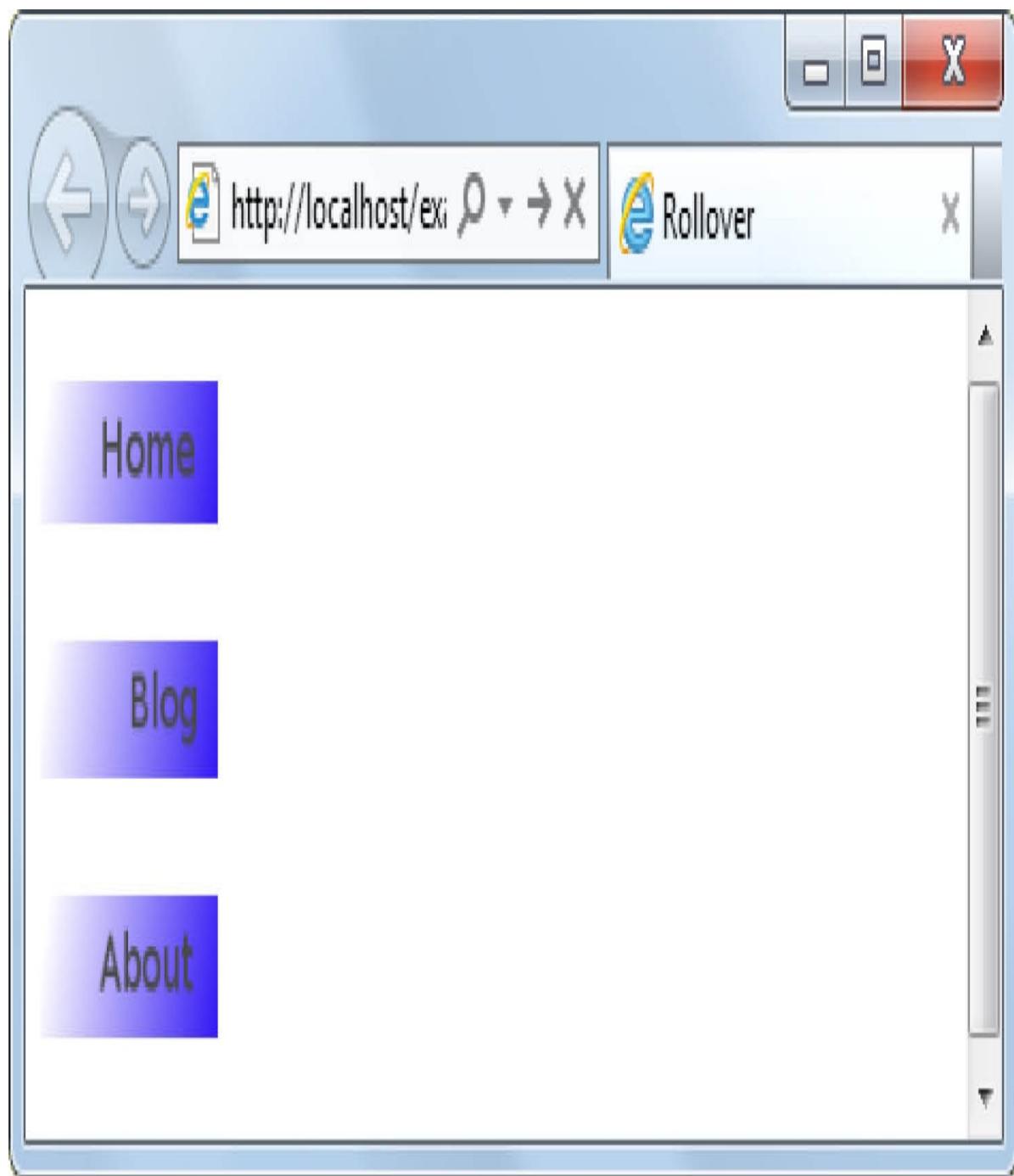
```

 return function() {
 images[i].src = images[i].
 };
 })(i));
EHandler.add(images[i],"mouseout", function()
 return function() {
 images[i].src = images[i].
 };
})(i));
}
</script>
</head>
<body>
<p>img id="home" name="img_home" data-src="home_default.pr
<p>img id="blog" name="img_blog" data-src="blog_default.pr
<p>img id="about" name="img_about" data-src="about_default
<script type="text/javascript">//gọi hàm tạo hiệu ứng rol
var bodyEl = document.getElementsByTagName("body")[0];
EHandler.add(bodyEl,"load", function() { rollover(); });
</script>
</body>
</html>

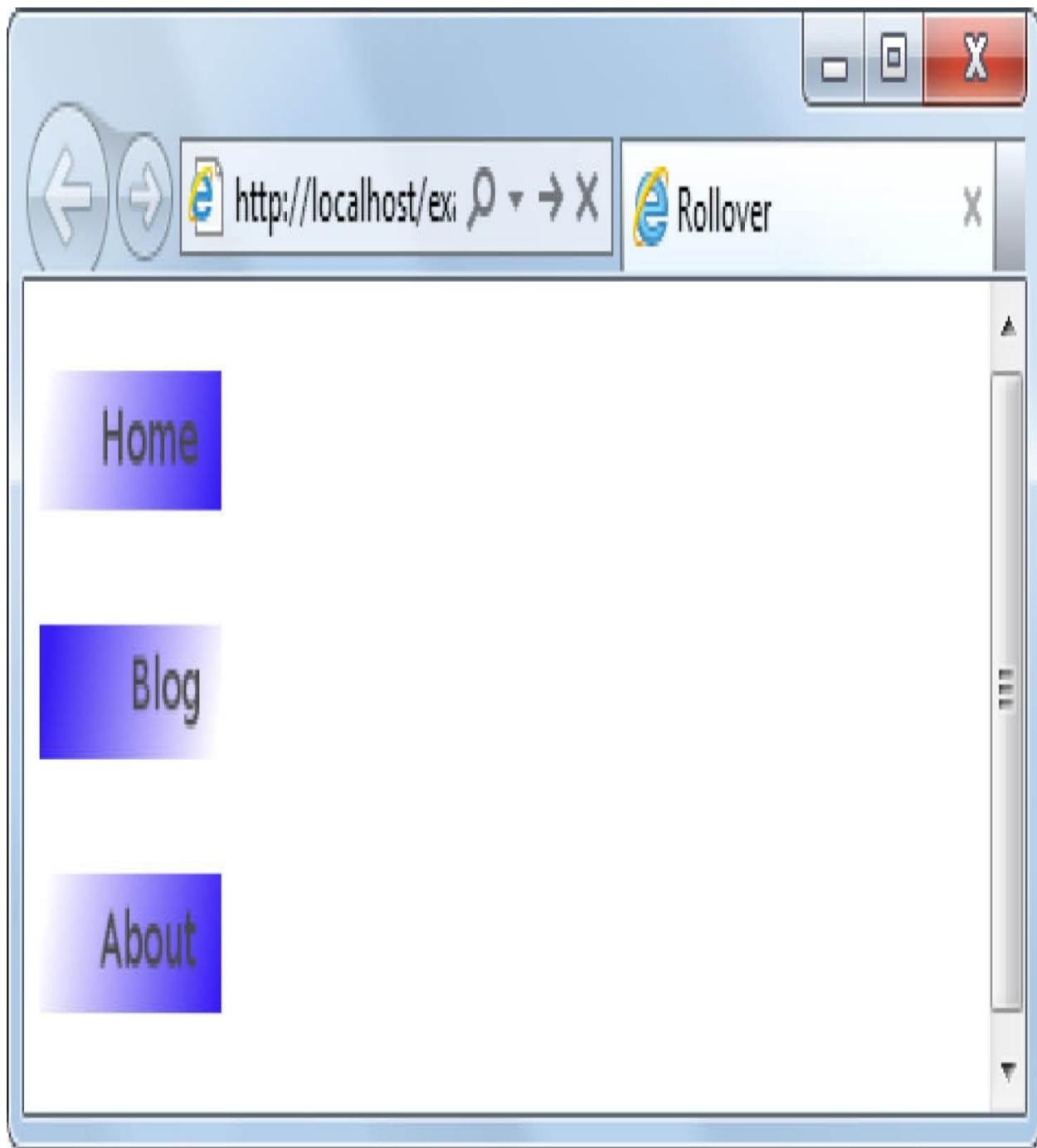
```



3. Dùng trình duyệt mở trang web. Bạn sẽ thấy một trang tương tự như ảnh chụp màn hình dưới đây. Nếu việc tải trang web gặp vấn đề, hãy đảm bảo rằng các hình ảnh phải được đặt trong thư mục hiện tại, bởi vì thư mục hiện tại là nơi mà thẻ <img> tìm kiếm các ảnh đó.



4. Di chuột trên các ảnh. Hình ảnh sẽ thay đổi thành các ảnh thay thế tương ứng. Dưới đây là ảnh màn hình khi di chuột lên ảnh Blog:



Ví dụ này trình bày một cách tốt hơn để tạo hiệu ứng rollover. Cũng giống như ví dụ trước, đoạn mã này tạo ra một hàm, sau đó gọi tới hàm đó trong sự kiện `window.onload`. Ngoài ra, đoạn mã này gom tất cả các ảnh cần dùng vào một biến có tên là `images`, sau đó duyệt qua từng ảnh và thêm các hàm xử lý sự kiện cho các ảnh đó:

```
function rollover() { // Dùng biến images để lưu các ảnh
```

```

var images = document.getElementsByTagName("img");
var imgLength = images.length;
for (var i = 0; i < imgLength; i++) {
 EHandler.add(images[i], "mouseover", function()
 return function() {
 images[i].src = images[i].id
 };
)(i);
 EHandler.add(images[i], "mouseout", function()
 return function() { // Truy xuất đến
 images[i].src = images[i].id
 };
)(i));
}

```

Một lần nữa, ví dụ này lại sử dụng đoạn script đăng ký sự kiện *EHandler* được nhắc tới trong Chương 11, gọi tới phương thức *Ehandler.add()* để đăng ký sự kiện *mouseover* và *mouseout* cho mỗi ảnh. Hàm này thường được sử dụng để đăng ký các sự kiện phức tạp bởi vì Windows Internet Explorer gặp một số vấn đề khi xử lý từ khóa *this* trong hàm xử lý sự kiện.

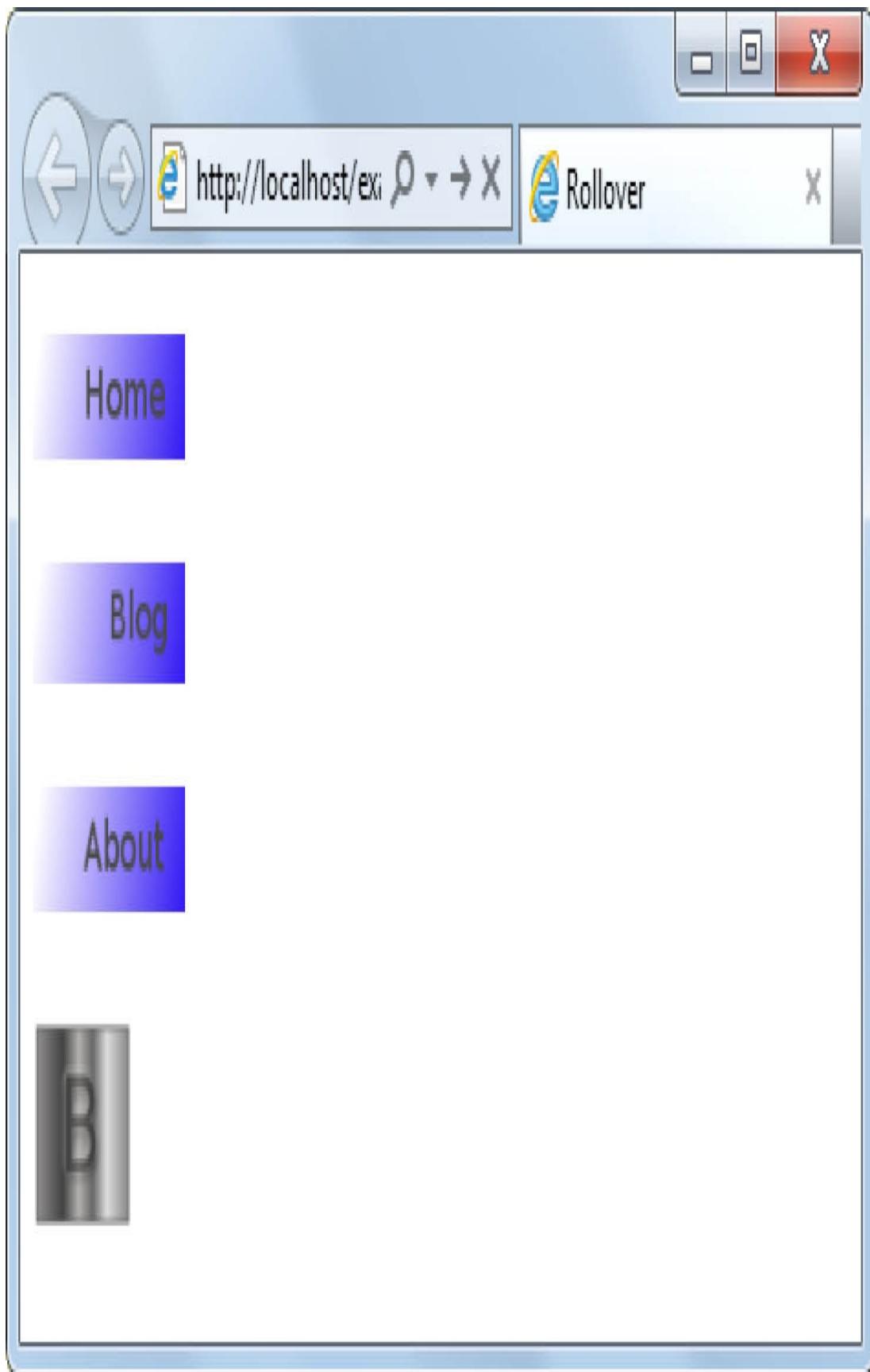
Trong các trình duyệt khác như Firefox, từ khóa *this* đại diện cho phần tử kích hoạt sự kiện - trong trường hợp này là *img*. Tuy nhiên, Internet Explorer không đăng ký sự kiện cho tới khi sự kiện đó được sử dụng, vì thế phần tử kích hoạt không được đăng ký trong quá trình đăng ký sự kiện (trong vòng lặp *for*). Đoạn mã cần phải nhảy qua một số vòng lặp để truyền tham chiếu tới phần tử kích hoạt sự kiện cho Internet Explorer. Trong trường hợp này, chỉ số *i* được truyền vào trong hàm *add()*, hàm này sau đó gọi tới một hàm không tên của chính nó.

Ví dụ này khác với Ví dụ 13-1 bởi vì đã bỏ đi định nghĩa của hàm *mouseOver()* và *mouseOut()*. Với ví dụ này, ID của mỗi bức ảnh được truy xuất bằng lời gọi *images[i].id* bên trong hàm. Sau đó, giá trị này được nối với chuỗi *"\_over.png"* hoặc *"\_default.png"* tùy thuộc vào hàm tương ứng.

Việc đảm bảo cho tên của file và thuộc tính *id* của thẻ *<img>* trùng khớp nhau là điều rất quan trọng. Dưới đây là thẻ *<img>* của ví dụ trên:

```
<p></p>
```

Trong đoạn mã mới, vòng lặp sẽ sử dụng một biểu thức chính quy thông qua phương thức *match()* để kiểm tra thuộc tính *id* có chứa từ *rollover* hay không. Nếu đúng, đoạn mã tiếp tục xử lý hiệu ứng rollover; ngược lại, đoạn mã đơn giản trả về giá trị hàm (*return*). Hình 13-3 là ví dụ về một trang web có bốn hình ảnh, ba trong số những ảnh đó có hiệu ứng rollover.



HÌNH 13-3 Đây là ví dụ cho thấy không phải tất cả các ảnh đều có hiệu ứng rollover.

Khi di chuột lên một hình ảnh bất kỳ trong ba hình ảnh ở phía trên cùng của trang web, hình ảnh thay thế sẽ được tải lên. Tuy nhiên, do ID của ảnh thứ tư không chứa từ *rollover* nên ảnh này không được gán hàm xử lý sự kiện *mouseover* và *mouseout*. Dưới đây là toàn bộ đoạn mã cho ví dụ này (chú ý rằng đoạn mã này vẫn cần một số cải tiến). Hãy chú ý tới tên mới của các file ảnh. (Bạn có thể xem đoạn mã trong thư mục *rolloverregexp* của Tài nguyên đi kèm.)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Rollover</title>
<script type="text/javascript" data-src="eHandler.js"></script>
<script type="text/javascript">
function rollover() {
 var images = document.getElementsByTagName("img");
 var imgLength = images.length; //số lượng ảnh trên trang
 for (var i = 0; i < imgLength; i++) { //chỉ lấy ảnh có id
 if (images[i].id.match(/rollover/)) {
 EHandler.add(images[i], "mouseover",
 return function() {
 images[i].src = images[i].dataSrc;
 });
 EHandler.add(images[i], "mouseout",
 return function() {
 images[i].src = images[i].dataSrc;
 });
 }
 }
}
</script>
</head>
<body>

<p></p>
<p></p>
<p></p>
```

```
<p>
<p>
<script type="text/javascript">
var bodyEl = document.getElementsByTagName("body")[0];
EHandler.addHandler(bodyEl,"load", function() { rollover(); });
</script>
</body>
</html>
```

Điểm khác biệt giữa đoạn mã này so với đoạn mã trước đó là sự thay đổi nhỏ trong hàm *rollover()* và phần tử img trong HTML. Trong hàm *rollover()*, một biểu thức chính quy được đặt trực tiếp trong phương thức *match()* để tìm kiếm chuỗi *rollover* trong thuộc tính id của ảnh. Nếu chuỗi *rollover* xuất hiện trong ID, các xử lý cho hiệu ứng *rollover* sẽ được cài đặt giống như trong các ví dụ trước. Nếu ngược lại, vòng lặp *for* sẽ tiếp tục.

## Tải trước ảnh



Có thể bạn đã để ý thấy một vấn đề khi làm việc với ví dụ về hiệu ứng *rollover* trong các phần trước. Lần đầu tiên hiển thị ảnh thay thế, phải mất một giây để ảnh đó hiển thị đầy đủ. Độ trễ này là do ảnh thay thế phải được tải về từ web server qua hệ thống mạng trước khi hiển thị trên trình duyệt.

Đây không phải là vấn đề lớn mà chỉ là chút phiền toái nhỏ khi mở trang web trên một kết nối mạng nhanh. Tuy nhiên, trên thực tế, độ trễ trở nên đáng kể trong các ứng dụng web, đặc biệt khi người dùng sử dụng kết nối dial-up tốc độ chậm. May mắn là bạn có thể tải trước các ảnh đó bằng cách sử dụng mã JavaScript. Việc *tải trước* sẽ lưu trữ các hình ảnh trong bộ nhớ đệm của trình duyệt, do đó những hình ảnh này gần như hiển thị ngay lập tức khi người truy cập di chuột trên ảnh.

Cơ sở lý thuyết của việc tải trước ảnh là tạo ra một đối tượng *image* sau đó gọi tới phương thức *src()* của đối tượng này để trả tới hình ảnh mà bạn muốn tải trước. Những gì bạn thao tác với đối tượng sau khi gọi phương thức *src()* không quan trọng. JavaScript không đồng bộ các lời gọi tải ảnh do đó phần còn lại của đoạn script vẫn được thực hiện trong khi hình ảnh tiếp tục được tải ở chế

độ chạy nền.

Khả năng tải trước ảnh không đồng bộ có ý nghĩa quan trọng khi bạn phải làm việc cùng lúc với nhiều ảnh: Bạn phải tạo một đối tượng *image* mới cho mỗi hình ảnh cần tải. Nếu có một loạt ảnh tạo hiệu ứng rollover, mỗi ảnh cần có đối tượng và lời gọi phương thức *src( )* riêng.

Phiên bản cuối cùng của đoạn mã tạo hiệu ứng rollover kết hợp kỹ thuật tải trước. Ví dụ 13-2 trình bày đoạn script tạo hiệu ứng rollover cùng với trang HTML; đoạn mã xử lý việc tải trước ảnh được in đậm. Đoạn mã này có trong Tài nguyên đi kèm, thư mục rolloverregexppreload.

### VÍ DỤ 13-2 Tải trước và hiệu ứng rollover.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Rollover</title>
<script type="text/javascript" data-src="eHandler.js"></script>
<script type="text/javascript">
function rollover() {
 var images = document.getElementsByTagName("img");
 var imgLength = images.length;
 var preLoad = []; //lưu các ảnh sẽ dùng
 for (var i = 0; i < imgLength; i++) {
 if (images[i].id.match(/rollover/)) { //tải
 preLoad[i] = new Image();
 preLoad[i].src = images[i].id + "_ov";
 EHandler.add(images[i],"mouseover", function() {
 images[i].src = images[i].id + "_ov";
 });
 EHandler.add(images[i],"mouseout", function() {
 images[i].src = images[i].id + "_or";
 });
 }
 }
}
```

```
</script>
</head>
<body>

<p></p>

<p></p>

<p></p>

<p></p>
<script type="text/javascript">
var bodyEl = document.getElementsByTagName("body")[0];
EHandler.add(bodyEl, "load", function() { rollover(); });
</script>
</body>
</html>
```



Đoạn mã trên sử dụng quy ước đặt tên cho các ảnh, vì vậy tất cả các quy ước về việc đồng bộ thuộc tính `id` trong mã JavaScript như đã được đề cập ở phần trước của chương này đều cần được tuân thủ.

## Làm việc với slideshow

Bạn có thể sử dụng JavaScript để tạo hiệu ứng “slideshow” (trình chiếu ảnh), hoán đổi ảnh bằng một ảnh khác trên cửa sổ trình duyệt. Trong phạm vi chương này, bạn sẽ xây dựng một slideshow cho phép người truy cập thay ảnh bằng cách nhấp vào các button để di chuyển sang ảnh sau hoặc ảnh trước đó. Hiệu ứng này khác với slideshow hẹn giờ - ứng dụng tự động hoán đổi các ảnh sau một khoảng thời gian nhất định.

### Tạo slideshow

Có nhiều cách để thực hiện tính năng slideshow với JavaScript. Một trong số đó

là dùng vòng lặp *for* để duyệt qua từng ảnh, phần này sẽ giới thiệu một phương pháp tạo slideshow đơn giản hơn.

Đa số các slideshow đều có thiết kế đơn giản mặc dù tôi từng thấy những slideshow rất phức tạp. Ví dụ 13-3 đưa ra một phiên bản slideshow chỉ có khả năng chuyển đổi sang ảnh tiếp theo.

### VÍ DỤ 13-3 Một slideshow đơn giản (chưa hoàn chỉnh).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ tạo slideshow</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
//mảng images lưu trữ các ảnh sẽ dùng
var images = ['home_default.png', 'about_default.png',
 'blog_default.png', 'logo.png']; //các ảnh sẽ dùng
function nextImage() { //chuyển đến ảnh tiếp theo
 var img = document.getElementById("slideimage");
 var imgname = img.name.split("_");
 var index = imgname[1];
 if (index == images.length - 1) {
 index = 0;
 } else {
 index++;
 }
 img.src = images[index];
 img.name = "image_" + index;
}
</script>
</head>
<body>
<p></p>
<form name="slideform">
<input type="button" id="nextbtn" value="Next">
</form>
<script type="text/javascript">
var nextBtn = document.getElementById("nextbtn");
EHandler.add(nextBtn, "click", function() { nextImage(); });

```

```
</script>
</body>
</html>
```

Tôi có thể giải thích về phần HTML trong đoạn mã trên :

```
<p>
<form name="slideform">
<input type="button" id="nextbtn" value="Next">
</form>
```

Đoạn mã HTML này hiển thị một ảnh, sau đó thiết lập thuộc tính ID và tên cho các giá trị cụ thể sẽ được sử dụng sau đó trong JavaScript. Tiếp theo, đoạn mã trên tạo ra một button có giá trị *Next*.

Phần JavaScript của đoạn mã đầu tiên thực hiện liên kết tới đối tượng *EHandler* đã được tạo ra trong Chương 11:

```
<script type="text/javascript" data-src="ehandler.js"></script>
```

Phần chính của đoạn mã JavaScript thực hiện di chuyển đến các ảnh tiếp theo trong slideshow:

```
//mảng images lưu trữ các ảnh sẽ dùng
var images = ['home_default.png', 'about_default.png', 'blog_default.png',
 'logo.png'];
function nextImage() { //chuyển đến ảnh tiếp theo
 var img = document.getElementById("slideimage");
 var imgname = img.name.split("_");
 var index = imgname[1];
 if (index == images.length - 1) {
 index = 0;
 } else {
 index++;
 }
 img.src = images[index];
 img.name = "image_" + index;
}
```

Đoạn mã này tạo ra một mảng các ảnh.

**Chú ý** Mảng lưu các ảnh được tạo trong đoạn mã trên chỉ chứa tên file vì thế các file ảnh phải được đặt trong cùng một thư mục với đoạn mã sẽ được thực thi. Nếu không, tên của các file ảnh trong mảng phải chứa đường dẫn chính xác.

Tiếp theo, một đoạn script trong phần thân của tài liệu sẽ kết nối hàm `nextImage()` cho sự kiện `click` của button Next thông qua phương thức `Ehandler.add()`:

```
<script type="text/javascript">
var nextBtn = document.getElementById("nextbtn");
EHandler.add(nextBtn, "click", function() { nextImage(); });
</script>
```

Lúc này, khi người dùng nhấp vào button Next, đoạn script sẽ gọi tới hàm `nextImage()`. Hàm `nextImage()` truy xuất đối tượng `image` từ thẻ HTML `<img>` thông qua hàm `getElementById()`. Tiếp đến, hàm này thực hiện phân tách thuộc tính `name` tại vị trí ký tự gạch dưới để nhận về ký tự số nằm ở cuối thuộc tính `name`. Ký tự số này được lưu trong biến `index`.

Phần tiếp theo của đoạn mã thực hiện một phép so sánh để kiểm tra xem giá trị của biến `index` có bằng độ dài của mảng `images` trừ đi 1 hay không. Nếu điều kiện này đúng, có nghĩa là người dùng đã xem tới ảnh cuối cùng của slideshow, do đó đoạn mã sẽ thiết lập lại giá trị `index` bằng 0 và bắt đầu lại từ đầu. Nếu slideshow chưa đi đến ảnh cuối, đoạn mã sẽ tăng giá trị biến `index` lên 1.

Hai dòng cuối của đoạn mã JavaScript thiết lập thuộc tính `src` cho ảnh mới và thiết lập tên tương ứng sao cho khi đoạn mã gọi tới hàm `nextImage()` trong lần tiếp theo thì chỉ số hiện tại có thể được xác định.

## Di chuyển ảnh theo chiều ngược lại

Có thể bạn cho rằng việc thêm một button mới cho phép di chuyển các ảnh theo chiều ngược lại trong slideshow đơn giản chỉ là sao chép đoạn mã vừa viết, sau đó chỉnh sửa một chút để tạo tính năng cho button Previous. Trong đa số trường hợp, suy nghĩ trên là đúng. Tuy nhiên, hãy xét một trường hợp đặc biệt khi bạn đang ở bức ảnh đầu tiên và muốn chuyển tới ảnh trước đó. Tình huống này làm cho việc sử dụng button Previous gặp chút khó khăn.

Bài tập tiếp theo sử dụng lại một số ảnh mà bạn đã thấy trong các bài tập và ví dụ trước. Những ảnh đó có thể làm cho slideshow trở nên nhảm chán, vì thế bạn hãy thoải mái thay thế chúng bằng các ảnh khác. Trong ví dụ này, tôi chọn ra bốn ảnh cho slideshow, bạn có thể sử dụng nhiều hơn nếu muốn. Tuy nhiên, hãy đảm bảo rằng bạn phải sử dụng ít nhất ba ảnh để kiểm thử đầy đủ tính năng chuyển tiếp và quay lui slideshow của đoạn mã JavaScript.

### Tạo button Previous

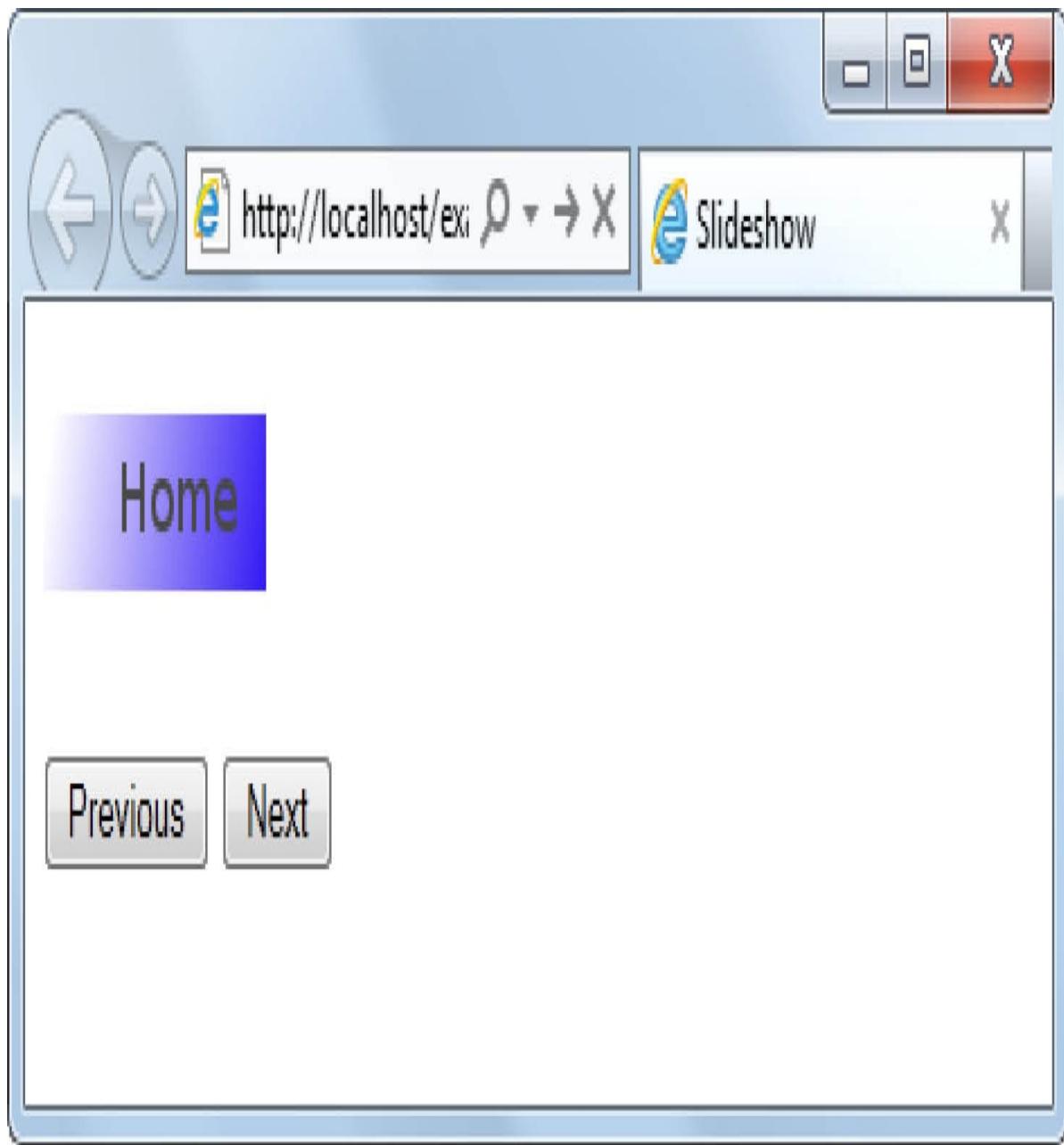
1. Dùng Visual Studio, Eclipse hoặc một trình soạn thảo khác chỉnh sửa lại nội dung của file slideshow.html trong thư mục Chương 13slideshow của Tài nguyên đi kèm.
2. Trong trang đó, thay thế dòng chú thích TODO trong file slideshow.txt bằng đoạn mã in đậm dưới đây:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ tạo slideshow</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
var images = ['home_default.png', 'about_default.png',
 'blog_default.png', 'logo.r
function nextImage() {
 var img = document.getElementById("slideimage");
 var imgname = img.name.split("_");
 var index = imgname[1];
 if (index == images.length - 1) {
 index = 0;
 } else {
 index++;
 }
 img.src = images[index];
 img.name = "image_" + index;
}
</script>
</head>
<body>
```

```
<p></p>
<form name="slideform">
<input type="button" id="prevbtn" value="Previous">
<input type="button" id="nextbtn" value="Next">
</form>
<script type="text/javascript">
var nextBtn = document.getElementById("nextbtn");
EHandler.add(nextBtn,"click",function() { nextImage(); });
</script>
</body>
</html>
```

3. Dùng trình duyệt mở trang web này. Bạn sẽ thấy một trang như sau:





4. Nhấn vào button Next để duyệt qua tất cả các ảnh. Chú ý rằng slideshow sẽ bắt đầu lại từ ảnh đầu tiên khi di chuyển tới ảnh cuối cùng.
5. Bây giờ hãy thay đổi đoạn mã trên để thêm vào button Previous (đoạn mã mới thêm vào được in đậm).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
```

```
<title>Ví dụ tạo slideshow</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
var images = ['home_default.png', 'about_default.png',
 'blog_default.png'];
function nextImage() { //chuyển đến ảnh tiếp theo
 var img = document.getElementById("slideimage");
 var imgname = img.name.split("_");
 var index = imgname[1];
 if (index == images.length - 1) {
 index = 0;
 } else {
 index++;
 }
 img.src = images[index];
 img.name = "image_" + index;
}
function prevImage() { //chuyển đến ảnh phía trước
 var img = document.getElementById("slideimage");
 var imgname = img.name.split("_");
 var index = imgname[1];
 if (index == 0) {
 index = images.length - 1;
 } else {
 index--;
 }
 img.src = images[index];
 img.name = "image_" + index;
}
</script>
</head>
<body>

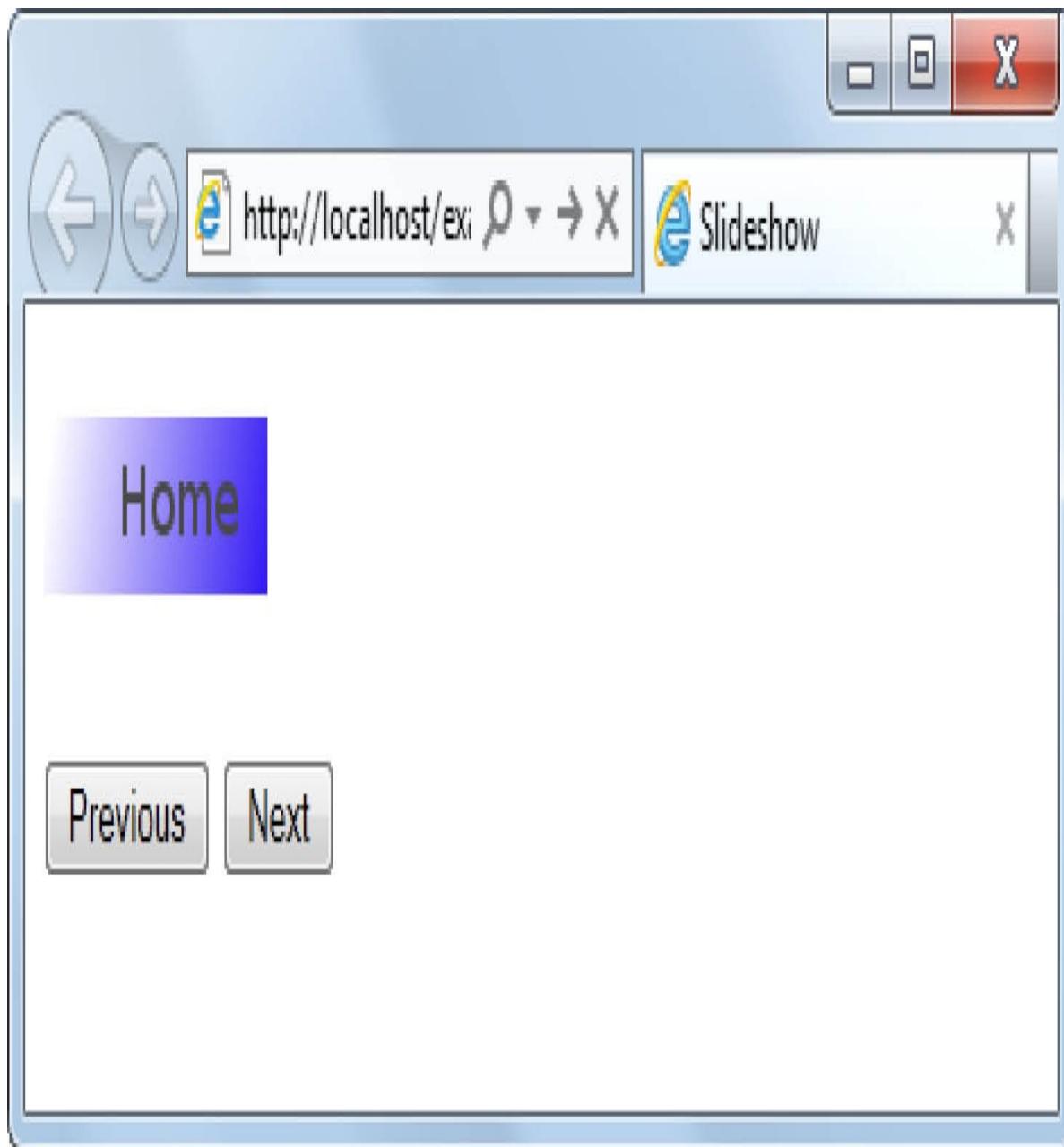
<p></p>

<form name="slideform">
<input type="button" id="prevbtn" value="Previous">
<input type="button" id="nextbtn" value="Next">
</form>
<script type="text/javascript">
var nextBtn = document.getElementById("nextbtn");

```

```
var prevBtn = document.getElementById("prevbtn");
EHandler.add(nextBtn,"click",function() { nextImage(); });
EHandler.add(prevBtn,"click",function() { prevImage(); });
</script>
</body>
</html>
```

6. Dùng trình duyệt mở trang web này. Bạn sẽ thấy button Previous xuất hiện.



7. Kiểm tra chức năng slideshow của trang web bằng cách sử dụng kết hợp cá

hai button để di chuyển tới các ảnh phía trước và phía sau trong slideshow.

Đoạn mã của bài tập này đã bổ sung thêm một button mới cho chức năng Previous trong trang HTML:

```
<input type="button" id="prevbtn" value="Previous">
```

Ngoài ra, đoạn mã JavaScript trên cũng thêm một hàm mới là *prevImage()* để chuyển đến các ảnh phía trước:

```
function prevImage() {
 var img = document.getElementById("slideimage");
 var imgname = img.name.split("_");
 var index = imgname[1];
 if (index == 0) {
 index = images.length - 1;
 } else {
 index--;
 }
 img.src = images[index];
 img.name = "image_" + index;
}
```



Đoạn mã này rất giống với hàm *nextImage()*, ngoại trừ phần điều kiện kiểm tra. Nếu index bằng 0, có nghĩa là slideshow đang ở bức ảnh đầu tiên, hàm *prevImage()* cần dịch chuyển đến bức ảnh cuối cùng. Ngược lại, đoạn mã sẽ dịch chuyển lùi lại một chỉ số để hiển thị ảnh trước đó.

## Làm việc với các bản đồ ảnh

Bản đồ ảnh là những ảnh có các vùng đặc biệt được cài đặt để thực hiện một số tính năng nhất định, ví dụ như liên kết tới một tài liệu khác. Bản đồ ảnh thường được sử dụng trong các bản đồ để chọn ra đất nước hoặc vùng mà khách truy cập lưu trú. Bản đồ ảnh cũng được sử dụng trong các menu mặc dù cách dùng này không còn phổ biến từ khi kỹ thuật CSS ra đời.

Không may, tôi không phải là một họa sĩ giỏi để có thể vẽ bản đồ của trái đất làm ví dụ. Thay vào đó, tôi đã tạo ra một bản đồ nhỏ biểu diễn một phần của bầu trời đêm kéo dài từ 44,52 độ vĩ Bắc cho tới -89,58 độ kinh Tây trong những

tháng mùa hè. Hình vẽ này được kèm theo đoạn mã nguồn mẫu của chương và được đặt tên là `nightskymapdefault.gif`.

Trong Hình 13-4 có bốn chòm sao: Ursa Minor, Cepheus, Draco và Cassiopeia.





HÌNH 13-4 Một phần của bầu trời đêm được quan sát từ Stevens Point, Wisconsin.

Tôi đã chuyển hình vẽ này thành bản đồ ảnh để khi người dùng nhấp vào một chòm sao bất kỳ, họ sẽ được liên kết tới trang Wikipedia giới thiệu về chòm sao đó. Đoạn mã cho bản đồ ảnh được trình bày trong Ví dụ 13-4 (đoạn mã này có trong file mã nguồn mẫu của thư mục Chương 13nightsky).

#### VÍ DỤ 13-4 Bản đồ ảnh bầu trời đêm.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Night Sky</title>
<script type="text/javascript">
</script>
</head>
<body>
<p>
<area coords="119,180,264,228" alt="Ursa Minor" shape="REC
href="http://en.wikipedia.org/wiki/Ursa_Minor">
<area coords="66,68,193,170" alt="Draco" shape="RECT"
href="http://en.wikipedia.org/wiki/Draco">
<area coords="36,170,115,246" alt="Draco" shape="RECT"
href="http://en.wikipedia.org/wiki/Draco">
<area coords="118,249,174,328" alt="Draco" shape="RECT"
href="http://en.wikipedia.org/wiki/Draco">
<area coords="201,47,298,175" alt="Draco" shape="RECT"
href="http://en.wikipedia.org/wiki/Cepheus_(constellation)">
<area coords="334,95,389,204" alt="Cassiopeia" shape="RECT"
href="http://en.wikipedia.org/wiki/Cassiopeia_(constellati
</map>
</body>
</html>
```

Đoạn mã HTML trên tạo ra một bản đồ ảnh đơn giản, sử dụng hệ tọa độ điểm ảnh biểu diễn các hình chữ nhật nhỏ tương ứng với mỗi chòm sao và ba hình chữ nhật để tạo thành hình dạng và phần đuôi của chòm sao Draco. Bạn có thể sử dụng JavaScript để cải thiện đoạn mã này.

Thẻ `<area>` của bản đồ ảnh hỗ trợ các sự kiện `mouseover` và `mouseout`. Sử

dụng những sự kiện này và mã JavaScript, bạn có thể làm cho bản đồ trở nên thú vị hơn. Ví dụ, khi người truy cập di chuyển chuột lên một vùng bản đồ, bạn có thể tải một ảnh mới để highlight chòm sao tương ứng. Đoạn mã trong Ví dụ 13-5 dưới đây thực hiện chức năng này bằng cách sử dụng sự kiện *mouseover* và *mouseout*. Đoạn mã này cũng có trong file mẫu của thư mục Chương 13nightsky, phần Tài nguyên đính kèm.

### VÍ DỤ 13-5 Một bản đồ ảnh đã thêm mã JavaScript.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Night Sky</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
function loadConst() {
 var areas = document.getElementsByTagName("area");
 var areaLength = areas.length; //lưu số vùng trên ảnh
 for (var i = 0; i < areaLength; i++) {
 EHandler.add(areas[i],"mouseover", function(i) {
 return function() {
 document.getElementById("nightsky_map")
 };
 }(i));
 EHandler.add(areas[i],"mouseout", function(i) {
 return function() {
 document.getElementById("nightsky_map")
 };
 }(i));
 } //kết thúc vòng lặp
} //kết thúc hàm loadConst
</script>
</head>
<body>
<p>img id="nightsky" name="nightsky" data-src="nightsky_map_
isMap useMap="#sky" alt="Bầu trời đêm"></p>
<map name="sky">
<area id="ursaminor" coords="119,180,264,228" alt="Ursa Minor">
<area id="draco" coords="66,68,193,170" alt="Draco" shape="Ri
```

```
<area id="draco" coords="36,170,115,246" alt="Draco" shape="I
<area id="draco" coords="118,249,174,328" alt="Draco" shape=
<area id="cepheus" coords="201,47,298,175" alt="Draco" shape:
href="http://en.wikipedia.org/wiki/Cepheus_(constellation)">
<area id="cassie" coords="334,95,389,204" alt="Cassiopeia" si
href="http://en.wikipedia.org/wiki/Cassiopeia_(constellation
</map>
<script type="text/javascript">
var bodyEl = document.getElementsByTagName("body")[0];
EHandler.add(bodyEl,"load", function() { loadConst(); });
</script>
</body>
</html>
```

Mở trang web này trên trình duyệt, khi bạn di chuyển chuột lên mỗi chòm sao thì hình vẽ phác thảo của chòm sao đó sẽ xuất hiện, giống như trong Hình 13-5. Đoạn mã JavaScript trên thực ra chỉ là một biến thể nhỏ của đoạn mã tạo hiệu ứng rollover mà bạn đã thấy ở các phần trước, nó truy xuất tới các phần tử `<area>` thay vì phần tử `<img>`.

```
var areas = document.getElementsByTagName("area");
```





HÌNH 13-5 Thêm đoạn mã JavaScript vào bản đồ ảnh để tạo hiệu ứng rollover.

Một cải tiến có thể áp dụng cho đoạn mã này đó là tải trước các ảnh thay thế để tạo hiệu ứng rollover cho bản đồ ảnh. (Bạn đã học điều này trong bài tập ở phần trước).

**Chú ý** Đoạn mã HTML trong ví dụ này không hoàn toàn hợp lệ theo chuẩn HTML 4.01 bởi vì các thẻ `<area>` của chòm sao Draco đều sử dụng chung một giá trị `id`. Để làm cho đoạn mã HTML này hợp lệ, mỗi thẻ phải có giá trị `id` riêng. Tuy nhiên, điều này làm cho đoạn mã JavaScript trở nên phức tạp bởi vì mỗi ID phải được phân tách hoặc phân tích để đảm bảo các ảnh phác họa của chòm sao Draco sẽ được tải; nếu không, bạn sẽ phải tải ba ảnh khác nhau hoặc tìm một phương pháp phù hợp khác.

## Bài tập



1. Tạo hiệu ứng rollover cho ảnh, tải trước ảnh và đảm bảo rằng các hàm JavaScript nằm tách biệt với phần mã HTML.
2. Sử dụng bản đồ ảnh có sẵn trong bài (hoặc một bản đồ ảnh khác mà bạn thích), hãy tải trước tất cả các ảnh sử dụng trong bản đồ để không cần tải những ảnh đó về khi người truy cập di chuột lên các vùng khác.

# Chương 14

## Sử dụng JavaScript với Web Form

Sau khi đọc xong chương này, bạn có thể:

- Nắm được cách sử dụng JavaScript để kiểm tra tính hợp lệ cho dữ liệu nhập vào web form (form trên trang web).
- Làm việc với các radio button, các select box và check box, lấy giá trị và thiết lập giá trị cho những phần tử này.
- Đưa ra các phản hồi dựa trên việc kiểm tra tính hợp lệ, thông qua hộp thoại thông báo `alert()` và thông qua dòng thông báo chèn vào ngay trên trang web.
- Nắm được những hạn chế trong việc kiểm tra tính hợp lệ trên form bằng JavaScript và tìm hiểu ví dụ về việc kiểm tra hợp lệ không đúng cách.

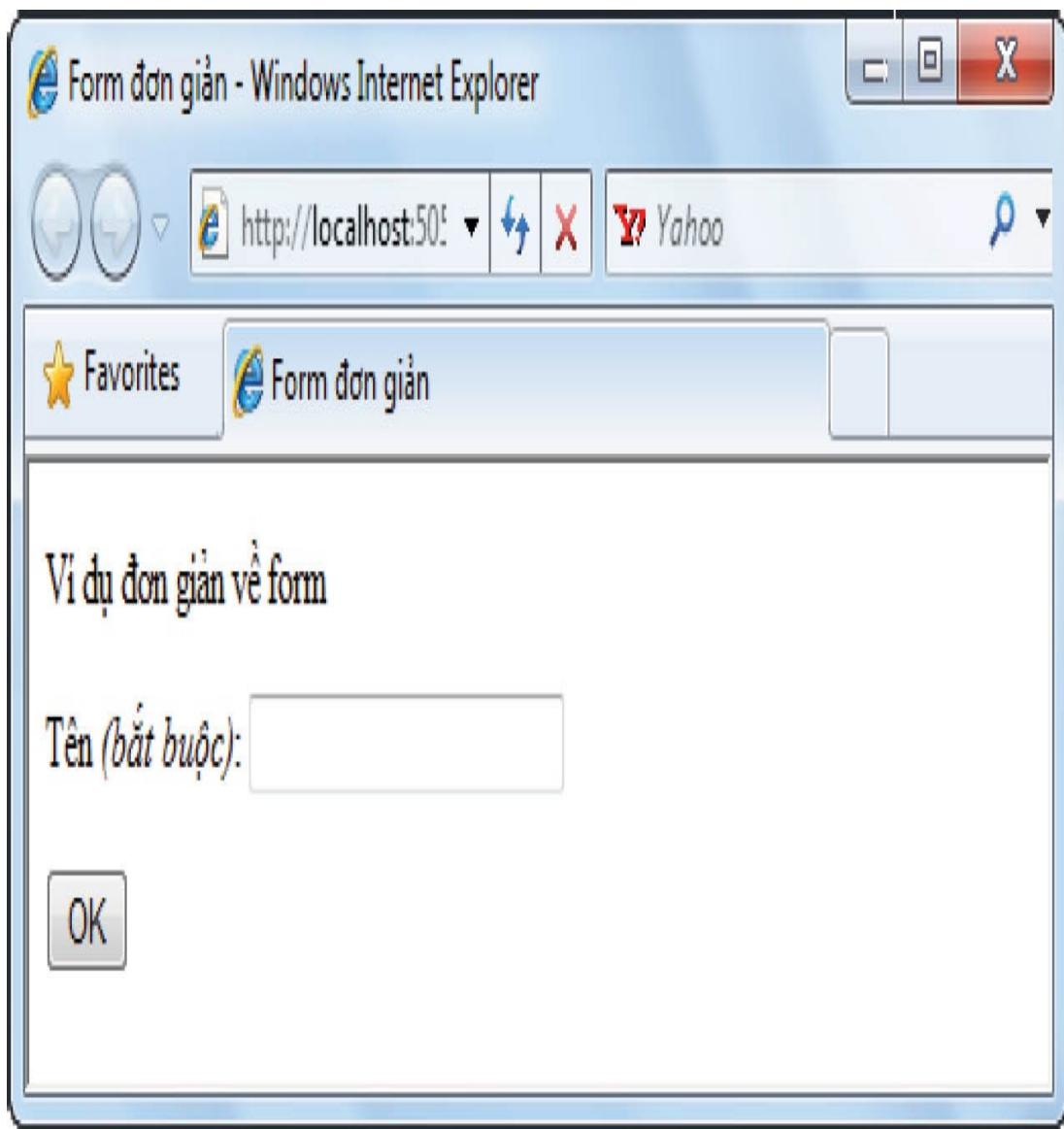
## JavaScript và Web Form

JavaScript đã được sử dụng với web form từ khá lâu – thường với mục đích kiểm tra xem liệu người dùng đã điền đầy đủ và chính xác các trường thông tin trên form trước khi gửi form đó tới server, đây được gọi là *quá trình kiểm tra hợp lệ phía client (client-side validation)*. Trước khi có JavaScript, trình duyệt phải gửi form và các thông tin trên form về server để đảm bảo tất cả các trường bắt buộc đã được điền đầy đủ, quá trình này được gọi là *quá trình kiểm tra hợp lệ phía server (server-side validation)*.

**Quan trọng** Khi sử dụng JavaScript, việc kiểm tra hợp lệ vẫn phải được thực hiện ở phía server phòng trường hợp người dùng tắt JavaScript hoặc người dùng cố ý có hành động phá hoại.

Hãy nhớ lại hàm `alert()` mà bạn đã tìm hiểu trong các chương trước, hàm này được sử dụng để minh họa cho các ví dụ đơn giản. Chúng ta sẽ gấp lại hàm `alert()` trong chương này. Hàm `alert()` thường được sử dụng để hiển thị thông tin phản hồi tới người dùng khi kiểm tra tính hợp lệ của form, mặc dù các kỹ thuật mới sử dụng mô hình đối tượng tài liệu (DOM) để hiển thị thông tin phản hồi theo cách thân thiện hơn.

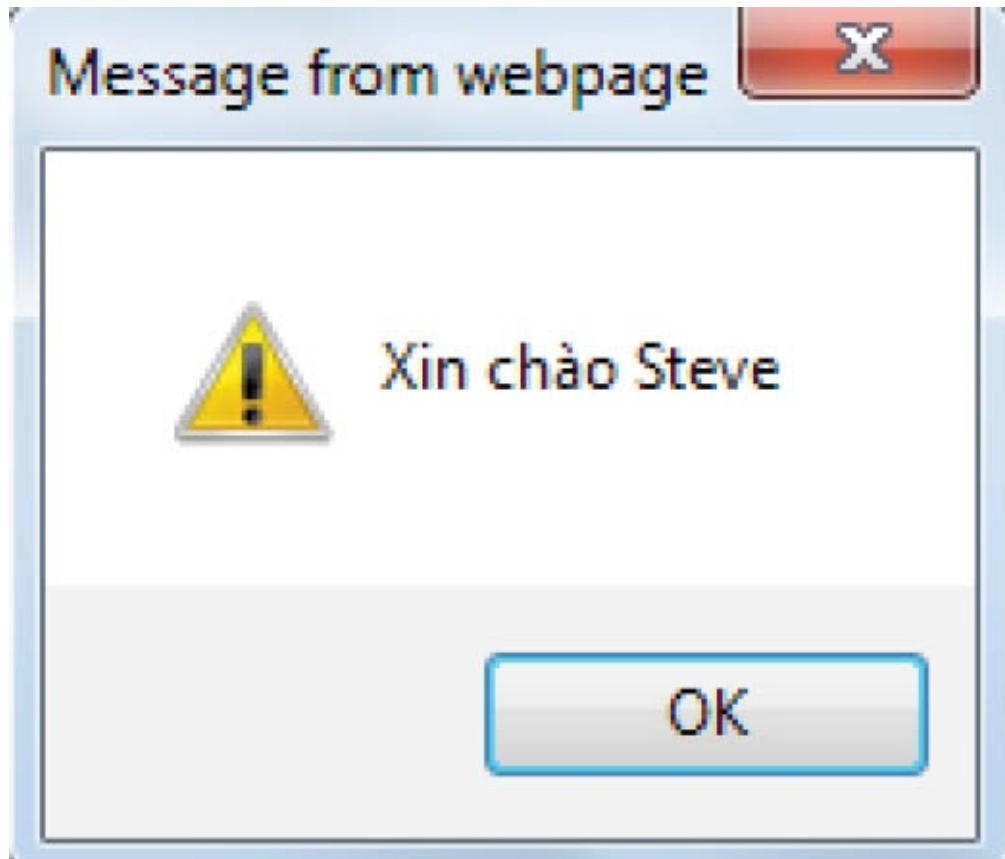
Một trang web với form đơn giản có dạng giống như Hình 14-1.



HÌNH 14-1 Một form đơn giản trên web.

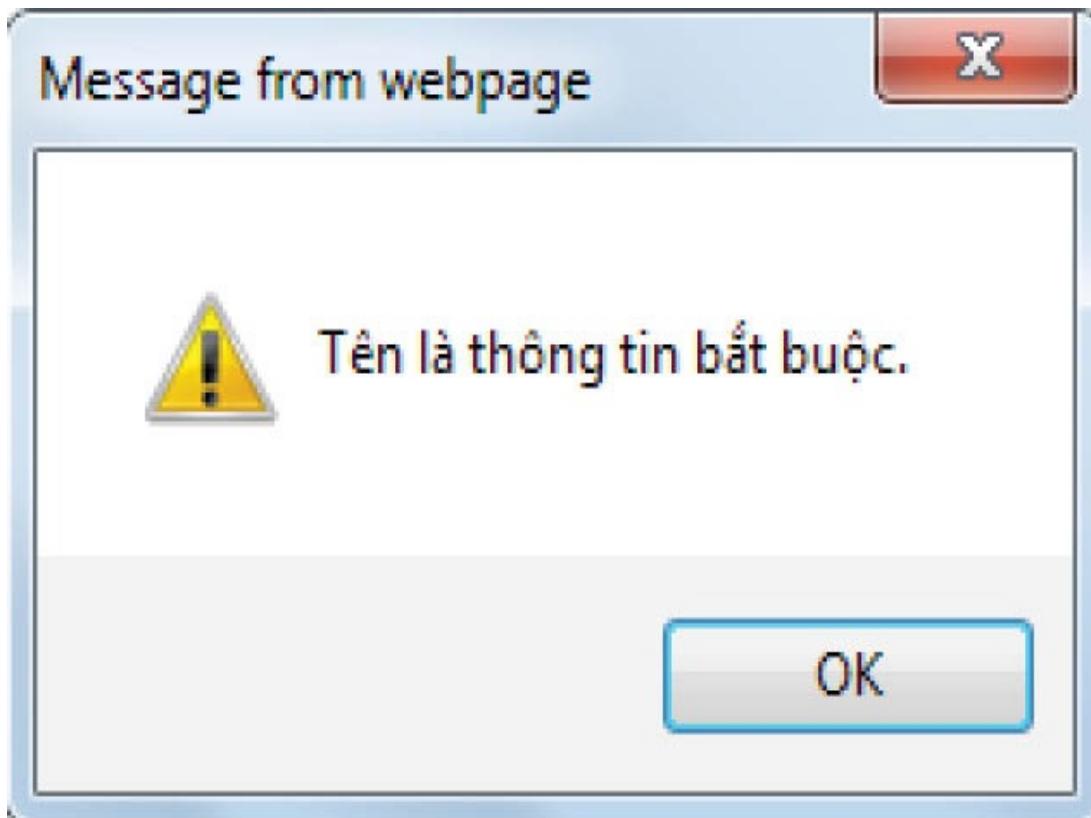
Khi người dùng gửi form này về server, mã JavaScript bên trong sẽ kiểm tra để

đảm bảo text box (ô văn bản) *Tên* đã được nhập thông tin. Khi text box *Tên* được nhập, ví dụ như “*Steve*”, trang web sẽ hiển thị tên được nhập như trong Hình 14-2.



HÌNH 14-2 Khi form được nhập thông tin, trang web hiển thị câu chào mừng.

Nếu người dùng không nhập dữ liệu vào text box *Tên*, đoạn mã sẽ hiển thị hộp thoại *alert()* thông báo rằng trường Tên chưa được nhập, như trong Hình 14-3.



HÌNH 14-3 Form hiển thị một hộp cảnh báo khi text box Tên bị bỏ trống.

Đoạn mã thực hiện chức năng trên được trình bày ở phần dưới đây. Bạn có thể tìm thấy đoạn mã này trong file formvalid.htm, phần Tài nguyên đi kèm. File này chứa mã HTML như sau:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Form đơn giản</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
function formValid(eventObj) {
 if (document.forms[0].textname.value.length == 0) {
 alert("Tên là thông tin bắt buộc.");
 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
 }
}
```

```

 } else {
 alert("Xin chào " + document.forms[0].textna
 return true;
 }
 }
</script>
</head>
<body>
<p>Ví dụ đơn giản về form</p>
<form action="#">
<p>Tên (bắt buộc): <input id="textbox1" name="textna
 type="text" /></p>
<p><input id="submitbutton1" type="submit" value="OK" /></p>
<script type="text/javascript">
var formEl = document.getElementsByTagName("form")[0];
EHandler.addHandler(formEl,"submit", function(eventObj) { formValid
</script>
</form>
</body>
</html>

```

Đầu tiên, đoạn mã JavaScript nằm trong phần tử `<head>` liên kết tới file `ehandler.js` chứa hàm xử lý sự kiện đã được xây dựng trong Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt”. Tiếp theo, đoạn mã này định nghĩa một hàm có tên là `formValid()` để xử lý dữ liệu nhập vào form:

```

function formValid(eventObj) {
 if (document.forms[0].textname.value.length == 0) {
 alert("Tên là thông tin bắt buộc.");
 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
 } else {
 alert("Xin chào " + document.forms[0]
 return true;
 }
}

```

Trong hàm `formValid()`, điều kiện `if` sử dụng mảng `document.forms[]`.

Với việc kiểm tra giá trị chỉ số đầu tiên ( $0$ ) của mảng này, đoạn mã chỉ xét tới form duy nhất của trang web. Điều kiện *if* kiểm tra độ dài của thuộc tính *textname.value* trên form có bằng  $0$  hay không. Nếu đúng, đoạn mã sẽ sử dụng hộp thoại *alert()* để thông báo lỗi. Ngược lại, đoạn mã hiển thị lời chào cùng với nội dung của thuộc tính *textname.value*.

Giá trị trả về của hàm *formValid()* rất quan trọng. Khi các hàm xử lý sự kiện *submit* hay *click* được gọi và trả về giá trị *false*, trình duyệt sẽ tạm dừng quá trình gửi form. Đó là lý do vì sao việc trả về giá trị *false* rất quan trọng khi việc kiểm tra tính hợp lệ thất bại. Nếu không trả về *false*, action mặc định vẫn sẽ tiếp tục và thực hiện gửi form. Bạn có thể dừng action mặc định này trong hầu hết các trình duyệt bằng cách gọi tới phương thức *preventDefault()*. Tuy nhiên, phương thức này không được hỗ trợ trong các phiên bản Windows Internet Explorer trước phiên bản 9, do đó đoạn mã trên đầu tiên thực hiện việc kiểm tra xem phương thức *preventDefault()* có được hỗ trợ hay không. Nếu được hỗ trợ, đoạn mã sẽ gọi phương thức *preventDefault()*; ngược lại, đoạn mã sẽ thiết lập giá trị thuộc tính *returnValue* của đối tượng *window.event* bằng *false*.

Một đoạn mã JavaScript ngắn xuất hiện trong phần tử HTML *<body>*, đoạn mã này gán sự kiện *submit* của form cho hàm xử lý sự kiện *EHandler*:

```
var formEl = document.getElementsByTagName("form")[0];
EHandler.add(formEl, "submit", function(eventObj) { formValid
```

Chú ý rằng để truy xuất form, hàm *formValid()* đã sử dụng giá trị của chỉ số đầu tiên của mảng *document.forms[]*, trong khi đó định nghĩa biến *formEl* sử dụng phương thức *getElementsByName*. Cả hai phương pháp này đều hoạt động tốt khi trên trang chỉ có một form duy nhất. Bạn sẽ thường xuyên bắt gặp các đoạn mã thực hiện truy xuất form thông qua tên của form như trong phần tiếp sau đây.

## Lấy dữ liệu từ form

Trước khi có thể đưa ra các phản hồi dựa trên dữ liệu form, bạn phải biết cách lấy dữ liệu đó. Ví dụ trong phần trước đã đưa ra cách truy cập dữ liệu form sử dụng mảng *document.forms[]* và hàm *getElementsByName*. Phần này

sẽ giới thiệu một cách khác để thực hiện thao tác này – đó là sử dụng thuộc tính *name* của form thay vì sử dụng chỉ số của form.

Cũng giống như các phần tử khác của trang HTML, bạn có thể thiết lập thuộc tính *id* của phần tử *form*. Dưới đây là ví dụ trong phần trước trong đó thuộc tính *id* của form đã được thêm vào:

```
<form action="#" name="testform">
<p>Tên (bắt buộc): <input id="textbox1" name="textname"
 type="text" /></p>
<p><input id="submitbutton1" type="submit" /></p>
</form>
```

Bây giờ bạn có thể truy cập form sử dụng thuộc tính *name* thay cho chỉ số như sau:

```
document.forms["testform"]
```

Sử dụng thuộc tính *name* rất hữu dụng, vì trong một số trường hợp bạn không thể biết chính xác chỉ số của form mà bạn muốn truy cập tới, điều này đôi lúc xảy ra khi đoạn mã phía server hoặc phía client tạo ra một form động, khi đó bạn phải tính toán (hoặc tồi tệ hơn là phải đoán) chỉ số của form trong tài liệu. Cách phù hợp nhất để đảm bảo rằng bạn tham chiếu đúng tới giá trị chỉ số đó là thiết lập thuộc tính *id* của form, sau đó truy cập tới form thông qua *id*. Bạn cũng có thể truy cập trực tiếp tới form theo cách không chuẩn tắc, đó là thông qua đối tượng *document*, nhưng tôi khuyên bạn không nên làm theo cách này:

```
document.testform
```

Phương pháp truy cập trực tiếp này không nhất quán giữa các trình duyệt và cũng không giúp giảm bớt việc viết mã so với cách viết chính tắc như sau:  
`document.forms["testform"]`

## Làm việc với thông tin của form

Bạn có thể truy cập tới từng phần tử của các form thông qua DOM. Phương thức chính xác để truy cập tới từng phần tử rất khác nhau, phụ thuộc vào kiểu phần tử. Với các text box và các danh sách chọn - select box (hay còn gọi là *menu thả xuống*), thuộc tính *value* chứa văn bản mà người truy cập nhập hoặc chọn. Với

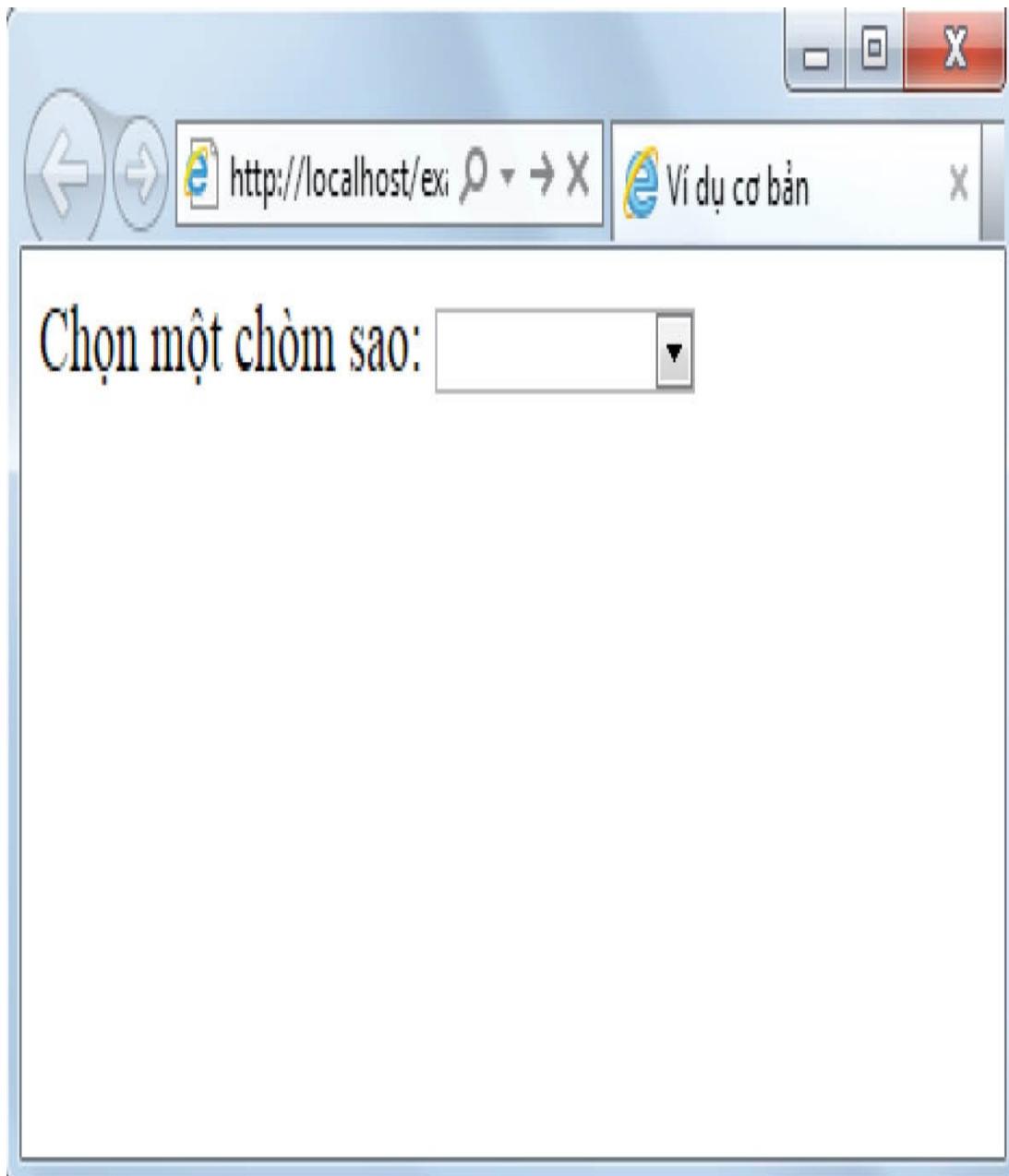
các radio button và check box, bạn phải sử dụng một phương pháp khác để xác định trạng thái của những phần tử này, đây cũng là nội dung của phần này.

## Làm việc với các Select Box

Select box chứa các nhóm tùy chọn. Dưới đây là ví dụ về đoạn mã HTML tạo select box (bạn có thể tìm thấy đoạn mã này trong file selectbox.txt của Tài nguyên đi kèm).

```
<form id="starform" action="">
Chọn một chòm sao:
<select name="startype" id="starselect">
<option selected="selected"> </option>
<option value="Aquila">Aquila</option>
<option value="Centaurus">Centaurus</option>
<option value="Canis Major">Canis Major</option>
<option value="Canis Minor">Canis Minor</option>
<option value="Corona Borealis">Corona Borealis</option>
<option value="Crux">Crux</option>
<option value="Cygnus">Cygnus</option>
<option value="Gemini">Gemini</option>
<option value="Lyra">Lyra</option>
<option value="Orion">Orion</option>
<option value="Taurus">Taurus</option>
<option value="Ursa Major">Ursa Major</option>
<option value="Ursa Minor">Ursa Minor</option>
</select>
</form>
```

Đoạn mã này tạo ra một select box như trong Hình 14-4.



HÌNH 14-4 Select box tạo ra từ đoạn mã HTML ở ví dụ trên.

Khi người dùng nhấn vào một tùy chọn, thuộc tính *value* của select box được thiết lập bằng giá trị tùy chọn đó. Trong ví dụ này, thuộc tính *value* của select box *startype* chưa bất kỳ giá trị nào được chọn bởi người dùng. Bạn có thể truy cập tới thuộc tính này theo cách như sau:

```
document.forms["starform"].startype.value
```

Với ví dụ cụ thể này, bạn cần liên kết một hàm xử lý sự kiện với sự kiện *change*

của select box, điều này được thực hiện với sự trợ giúp của đối tượng *EHandler* đã được tạo ra trong Chương 11. Sự kiện *change* kích hoạt một hàm mỗi khi tùy chọn trong select box thay đổi, ví dụ như khi người dùng chọn một tùy chọn từ menu thả xuống. Trang web gán sự kiện *change* cho ô *<select>* với sự trợ giúp của đoạn mã trong Ví dụ 14-1, đoạn mã này được thêm vào phần *<body>* của trang web.

**Chú ý** Hãy nhớ tham chiếu đoạn mã *EHandler* trong phần tử *<head>*. Xem lại Chương 11 để biết thêm thông tin.

**VÍ DỤ 14-1** Gán sự kiện *change* cho phần tử *<select>* sử dụng đoạn mã *EHandler* trong Chương 11.

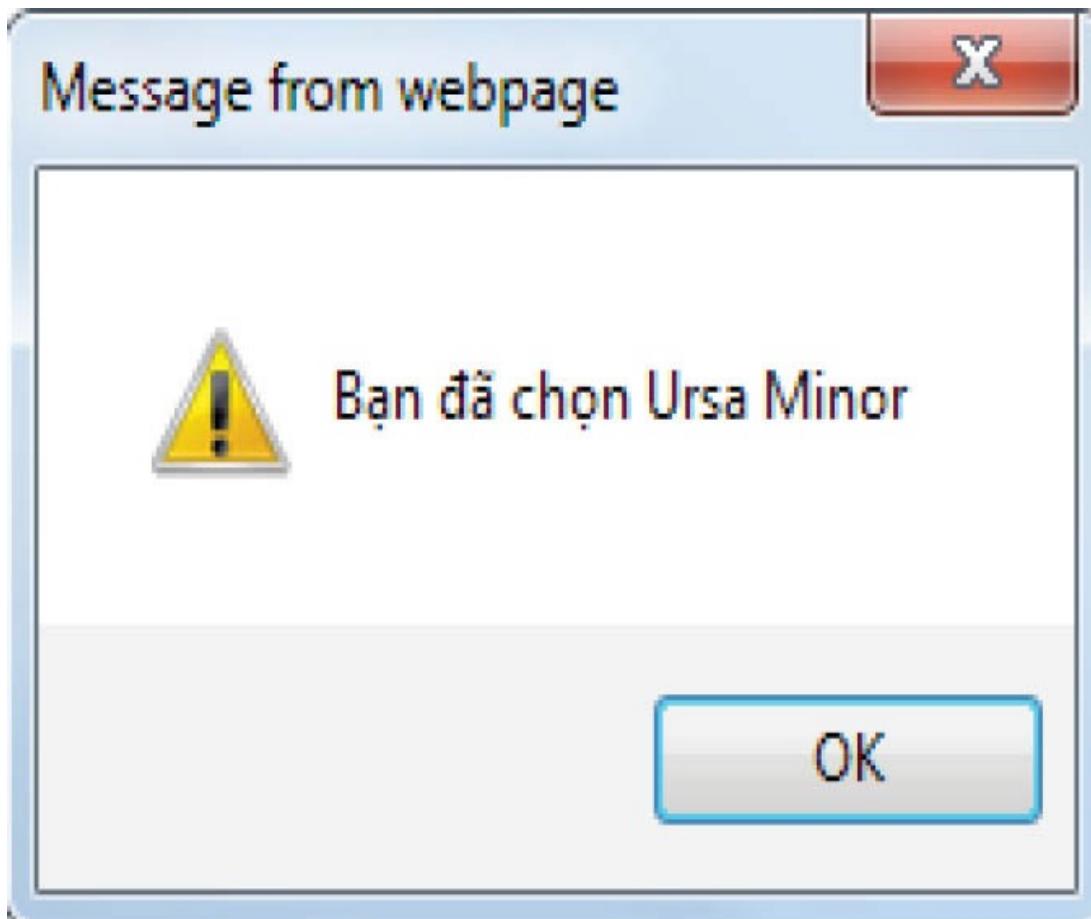
```
<script type="text/javascript">
var selEl = document.getElementById("starselect");
EHandler.add(selEl,"change", function() { displayValue();
</script>
```

Đoạn mã này sử dụng phương thức *EHandler.add()* để gán một hàm vào sự kiện *change* của phần tử *<select>*, phần tử này được truy xuất thông qua thuộc tính ID, đó là *starselect*. Trong trường hợp này, hàm xử lý được gán cho sự kiện *change* là một hàm người dùng định nghĩa, có tên là *displayValue()*, hàm này được trình bày trong Ví dụ 14-2.

**VÍ DỤ 14-2** Hàm *displayValue()* được gọi khi sự kiện *change* của form được kích hoạt.

```
//Hàm displayValue hiển thị chòm sao đã chọn
function displayValue(){
 var selected = document.forms["starform"].startype
 alert("Bạn đã chọn " + selected);
}
```

Đoạn mã JavaScript trên chỉ hiển thị giá trị được chọn từ menu thả xuống. Ví dụ, khi chọn Ursa Minor từ menu thả xuống, hộp thoại *alert()* như trong Hình 14-5 sẽ được hiển thị.



HÌNH 14-5 Chọn một chòm sao thông qua `form` sau đó hiển thị hộp thoại `alert()`

**Chú ý** Đoạn mã hoàn thiện cho ví dụ này có trong file sel.htm trong thư mục mã nguồn mẫu Chương 14 của Tài nguyên đi kèm.

Mã HTML của select box có chứa thuộc tính `selected`, thuộc tính này cho biết tùy chọn nào sẽ được hiển thị. Ví dụ này chọn trước một tùy chọn rỗng, do đó giá trị ban đầu của select box trống.

```
<option selected="selected"> </option>
```

Bạn cũng có thể sử dụng JavaScript và DOM để chọn. Cách thức chọn bằng lập trình thường được dùng với các form có nhiều trường nhập liệu sao cho khi người dùng lựa chọn trên form sẽ làm một số tùy chọn khác được chọn tự động.

Trong bài tập sau đây, bạn sẽ tạo form mà một nhà hàng pizza có thể sử dụng để nhận đơn đặt hàng. Nhà hàng này chỉ sản xuất một số loại pizza đặc biệt: pizza rau, pizza thịt và pizza theo phong cách Hawaii với thịt dăm bông và dứa. Nhà

hàng muốn tạo một trang web có ba button (nút bấm) cho phép nhân viên làm pizza theo dõi các loại pizza được đặt hàng. Những button này cho phép chọn trước lớp phủ cho bánh pizza.

## Chọn một tùy chọn với JavaScript

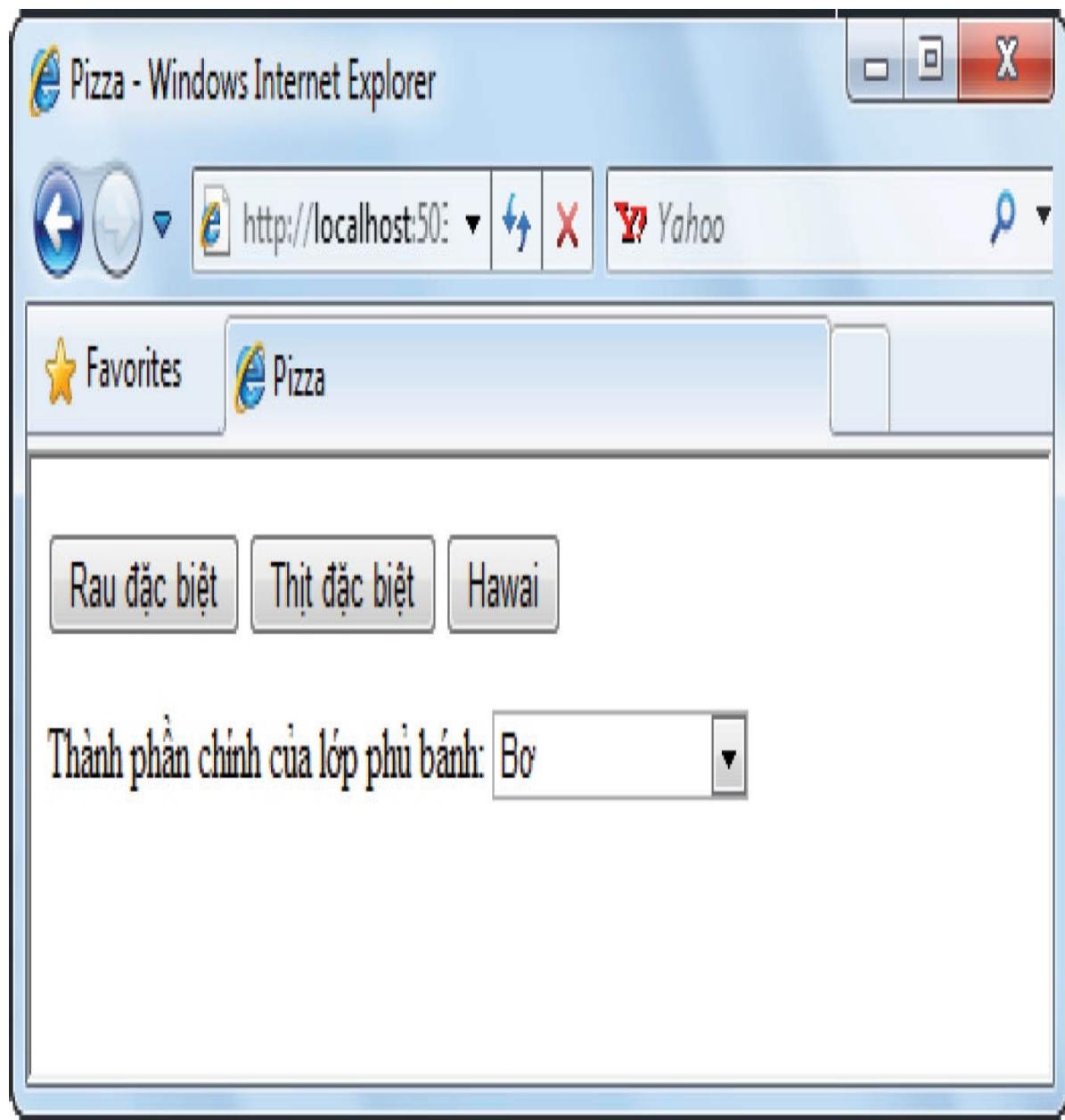
1. Dùng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác chỉnh sửa lại file pizza.htm trong thư mục Chương 14 (phần Tài nguyên đi kèm).
2. Hãy thêm đoạn mã được in đậm dưới đây vào trang này (đoạn mã này nằm trong file pizza.txt của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Pizza</title>
 <script type="text/javascript" data-src="ehandler">
 <script type="text/javascript">
 //Gán giá trị cho thành phần chính của bánh Pizza
 //loại Pizza
 function flip(pizzatype) {
 if (pizzatype.value == "Rau đặc biệt") {
 document.forms["pizzaform"].topping.value = "Rau đặc biệt";
 } else if (pizzatype.value == "Thịt đặc biệt") {
 document.forms["pizzaform"].topping.value = "Thịt đặc biệt";
 } else if (pizzatype.value == "Hawai") {
 document.forms["pizzaform"].topping.value = "Hawai";
 }
 }
 </script>
 </head>
 <form id="pizzaform" action="#">
 <p>
 <input id="vegbutton" type="button" name="veggiespecial"
 value="Rau đặc biệt">
 <input id="meatbutton" type="button" name="meatspecial"
 value="Thịt đặc biệt">
 <input id="hawbutton" type="button" name="hawaiian"
 value="Hawai">
 </p>
 </form>
</body>
```

```
Thành phần chính của lớp phủ bánh: <select name="topping">
<option value="cheese" selected="selected">Bơ</option>
<option value="veggies">Rau</option>
<option value="meat">Thịt</option>
<option value="hampineapple">Dăm bông & Dứa</option>
</select>
</form>
<script type="text/javascript">
//Xử lý sự kiện khi các nút được nhấn
var vegEl = document.getElementById("vegbutton");
var meatEl = document.getElementById("meatbutton");
var hawEl = document.getElementById("hawbutton");
EHandler.add(vegEl,"click",function() { flip(vegEl); });
EHandler.add(meatEl,"click",function() { flip(meatEl); });
</script>
</body>
</html>
```

3. Dùng trình duyệt mở trang web. Bạn sẽ nhìn thấy một trang giống như sau:





- Chọn một button. (Chú ý là select box cho “Thành phần chính” sẽ thay đổi tương ứng.)

Trọng tâm của ví dụ này là hàm *flip()*:

```
//Gán giá trị cho thành phần chính của bánh Pizza dựa vào loại pizza
function flip(pizzatype) {
 if (pizzatype.value == "Rau đặc biệt") {
 document.forms["pizzaform"].topping.value =
 } else if (pizzatype.value == "Thịt đặc biệt") {
```

```
 document.forms["pizzaform"].topping.value =
 } else if (pizzatype.value == "Hawaii") {
 document.forms["pizzaform"].topping.value =
 }
}
```

Hàm này kiểm tra giá trị của biến *pizzatype* được truyền vào, sau đó sử dụng một lệnh điều kiện để thay đổi tương ứng giá trị của select box *topping*.

Trong ví dụ này, phần *<head>* của trang web liên kết tới đoạn mã xử lý sự kiện *EHandler* và sử dụng phương thức *EHandler.add* để gán sự kiện *click* cho các nút, tương tự như đoạn mã bạn đã thấy trong các chương trước.

```
//Xử lý sự kiện khi các nút được nhấn
var vegEl = document.getElementById("vegbutton");
var meatEl = document.getElementById("meatbutton");
var hawEl = document.getElementById("hawbutton");
EHandler.add(vegEl,"click",function() { flip(vegEl); });
EHandler.add(meatEl,"click",function() { flip(meatEl); });
EHandler.add(hawEl,"click",function() { flip(hawEl); });
```

Ví dụ này cho thấy cách lấy thông tin từ một form và làm thế nào để thiết lập thông tin trong form. Form này trông còn khá đơn giản và nhà hàng vẫn chưa sản xuất nhiều loại pizza. Tuy nhiên, nhà hàng vẫn đang phát triển bởi vì pizza của nhà hàng rất được ưa chuộng. Các ví dụ tiếp theo sẽ tiếp tục mở rộng form này.

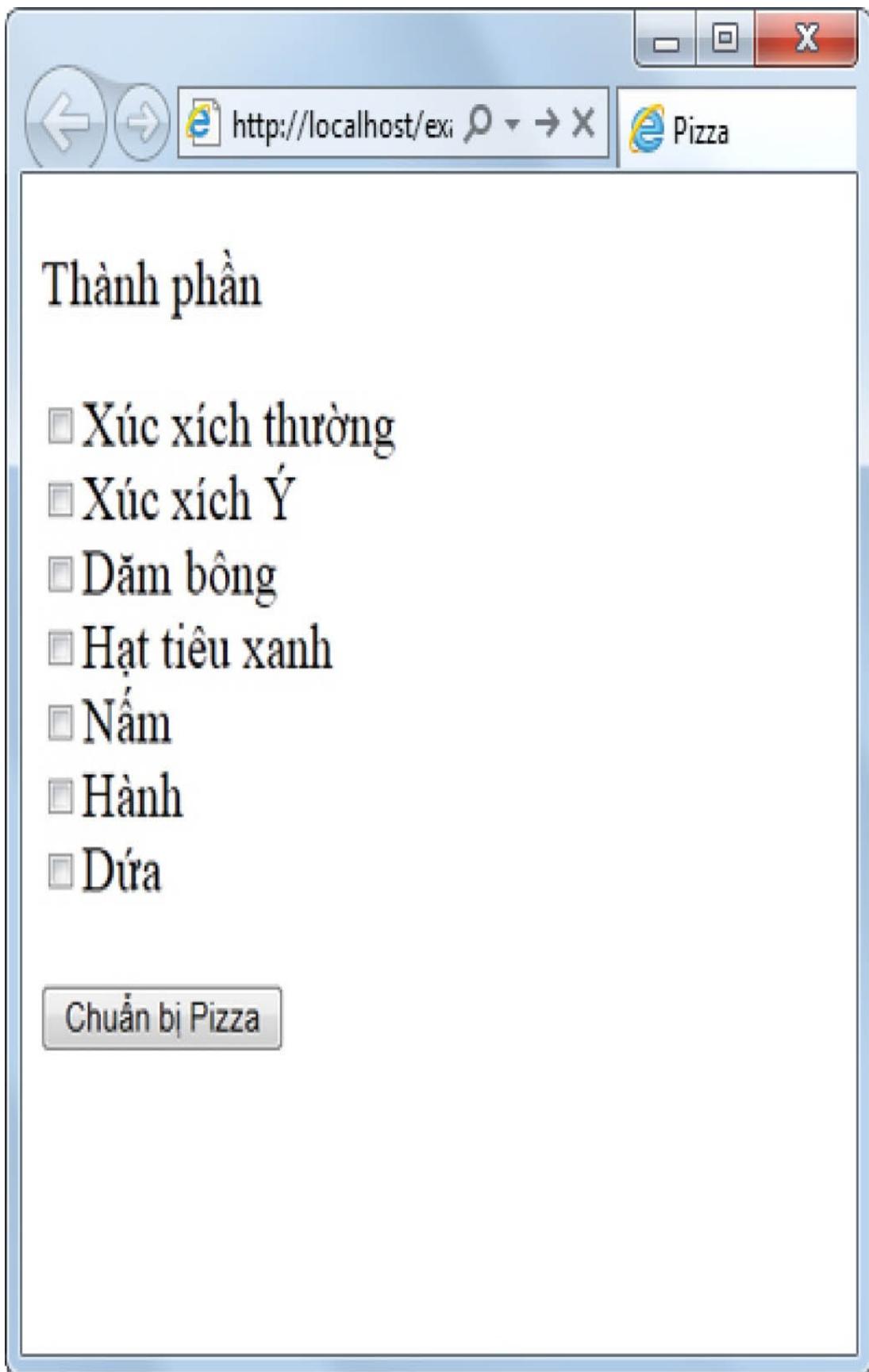
## Làm việc với Check Box

Ví dụ trước đã trình bày về các select box và bạn cũng đã thấy cách sử dụng các text box ở phần đầu chương. Một kiểu ô nhập liệu khác nữa đó là check box (hộp kiểm hay hộp đánh dấu), cho phép người dùng chọn cùng lúc nhiều phần tử. Trang web đặt hàng bánh pizza ở phần trước là ví dụ hay để minh họa cho check box.

Hãy nhớ lại trong phần trước, khi người dùng chọn một trong ba loại pizza thì select box *Thành phần chính của lớp phủ bánh*: thay đổi tương ứng với loại bánh được chọn. Tuy nhiên, trang web đó không cho phép người dùng chọn cùng lúc nhiều loại pizza.

Hình 14-6 đưa ra một form đặt pizza mới. Người đặt hàng có thể chọn nhiều thành phần với sự kết hợp tùy ý.





## Thành phần

- Xúc xích thường
- Xúc xích Ý
- Dăm bông
- Hạt tiêu xanh
- Nấm
- Hành
- Dứa

Chuẩn bị Pizza

HÌNH 14-6 Thay đổi form đặt pizza bằng cách sử dụng check box.

Khi chọn các thành phần của bánh pizza và nhấn vào button Chuẩn bị Pizza, các thành phần được chọn sẽ hiển thị trên màn hình như trong Hình 14-7.



The screenshot shows a web browser window with the URL `http://localhost/exi`. The title bar says "Pizza". The main content area is titled "Thành phần" (Ingredients). It lists several toppings with checkboxes:

- Xúc xích thường
- Xúc xích Ý
- Dăm bông
- Hạt tiêu xanh
- Nấm
- Hành
- Dứa

Below the ingredients is a button labeled "Chuẩn bị Pizza" (Prepare Pizza).

Text below the button states: "Chiếc bánh pizza này sẽ có:" (This pizza will have:)

- Hạt tiêu xanh
- Nấm
- Hành

HÌNH 14-7 Đặt bánh pizza thông qua form mới và thêm các phần tử sử dụng DOM.

Đoạn mã thực hiện chức năng này được trình bày trong Ví dụ 14-3 (Xem file listing14-3.htm trong Tài nguyên đi kèm).

### VÍ DỤ 14-3 Sử dụng check box trong form đặt hàng.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Pizza</title>
 <script type="text/javascript" data-src="ehandler.
 <script type="text/javascript">
 function prepza() {
 var checkboxes = document.forms["pizzaform"]
 var newelement = document.createElement("p")
 newelement.setAttribute("id", "orderheading")
 document.body.appendChild(newelement);
 newelement.appendChild(document.createTextNode(
 "Chiếc bánh pizza này sẽ có:"));
 for (var i = 0; i < checkboxes; i++) {
 if (document.forms["pizzaform"].to
 var newelement = document.createElement("p")
 newelement.setAttribute("id", "newe
 document.body.appendChild(neweleme
 newelement.appendChild(document.c
 document.forms["pizzaform"]
 }
 }
 }
 </script>
</head>
<body>
<form id="pizzaform" action="#">
 <p>Thành phần</p>
 <input type="checkbox" id="topping1" value="Xúc xích thường"
 name="toppingcheck">Xúc xích thường

 <input type="checkbox" id="topping2" value="Xúc xích Ý"
 name="toppingcheck">Xúc xích Ý

 <input type="checkbox" id="topping3" value="Dăm bông"
 name="toppingcheck">Dăm bông

```

```

<input type="checkbox" id="topping4" value="Hạt tiêu xanh"
 name="toppingcheck">Hạt tiêu xanh

<input type="checkbox" id="topping5" value="Nấm"
 name="toppingcheck">Nấm

<input type="checkbox" id="topping6" value="Hành"
 name="toppingcheck">Hành

<input type="checkbox" id="topping7" value="Dứa"
 name="toppingcheck">Dứa

<input type="button" id="prepBtn" name="prepBtn"
 value="Chuẩn bị Pizza"></p>
</form>
<script type="text/javascript">
var prepBtn = document.getElementById("prepBtn");
EHandler.addHandler(prepBtn,"click",function() { prepza(); });
</script>
</body>
</html>

```

Phần chính của trang web này là hàm *prepza()*, hàm này bắt đầu bằng việc thu thập số ô được đánh dấu chọn trong form *pizzaform*. Các ô này được nhóm lại với nhau sử dụng thuộc tính *toppingcheck* như sau:

```
var checkboxes = document.forms["pizzaform"].toppingcheck.le
```

Sau khi tạo ra phần tử *<p>* với tiêu đề, đoạn mã sử dụng vòng lặp *for* để duyệt qua từng check box. Từng check box được kiểm tra xem thuộc tính *checked* có được thiết lập hay không:

```
if (document.forms["pizzaform"].toppingcheck[i].checked) {
```

Nếu thuộc tính *checked* của check box được thiết lập, đoạn mã tạo ra một phần tử *<p>* mới và thêm vào tài liệu. Kết quả chính là trang web mà bạn thấy trong Hình 17-7. (Bạn đã thấy các ví dụ về cách tạo và thêm các phần tử trong Chương 10 “Mô hình đối tượng tài liệu - DOM”).

Hãy ghi nhớ ví dụ này, vì một trong các bài tập cuối chương sẽ yêu cầu bạn kết hợp ví dụ này với tính năng tự động chọn các thành phần cho bánh pizza khi người dùng nhấn vào button, giống như ví dụ về select box mà bạn đã thấy trong phần trước.

Sự kiện *click* của button Chuẩn bị Pizza được gán với phương thức *EHandler.add()* đã được nhắc đến trong Chương 11.

## Làm việc với Radio Button

Các radio button cũng tạo ra một nhóm các tùy chọn, nhưng không giống như check box, tại một thời điểm sẽ chỉ có một radio button trong nhóm được chọn. Trong tình huống của ví dụ nhà hàng pizza, người truy cập có thể sử dụng một radio button để chọn loại đế bánh: mỏng, dày hoặc loại thường. Vì mỗi chiếc pizza chỉ có thể có một kiểu đế bánh, nên việc sử dụng radio button là hoàn toàn hợp lý. Trang web sau khi thêm các radio button chọn loại đế bánh sẽ trông giống như trong Hình 14-8.



http://localhost:53505/HTMLPage1.htm... X

Yahoo!

Favorites http://localhost:53505/HTMLP...

Thành phần      Đế bánh

Xúc xích thường  Bình thường

Xúc xích Ý  Dày

Dăm bông  Mỏng

Hạt tiêu xanh

Nấm

Hành

Dứa

HÌNH 14-8 Thêm các radio button để chọn loại đế cho bánh pizza.

Dưới đây là đoạn mã HTML thực hiện thêm các radio button này cùng với một bảng chứa những button đó (đoạn mã này cũng nằm trong file radiobuttonhtml.txt, phần Tài nguyên đi kèm):

```
<table>
<tr><td>Thành phần</td><td>Đế bánh</td></tr>
<tr>
 <td><input type="checkbox" id="topping1" value="Xúc xích thường" name="toppingcheck">Xúc xích thường</td>
 <td><input type="radio" name="crust" value="Bình thường" checked="checked" id="radio1">Bình thường</td>
</tr>
<tr>
 <td><input type="checkbox" id="topping2" value="Xúc xích Ý" name="toppingcheck">Xúc xích Ý</td>
 <td><input type="radio" name="crust" value="Dày" id="radio2" />Dày</td>
</tr>
<tr>
 <td><input type="checkbox" id="topping3" value="Dăm bông" name="toppingcheck">Dăm bông</td>
 <td><input type="radio" name="crust" value="Mỏng" id="radio3">Mỏng</td>
</tr>
<tr>
 <td><input type="checkbox" id="topping4" value="Hạt tiêu xanh" name="toppingcheck">Hạt tiêu xanh</td>
 <td></td>
</tr>
<tr>
 <td><input type="checkbox" id="topping5" value="Nấm" name="toppingcheck">Nấm</td>
 <td></td>
</tr>
<tr>
 <td><input type="checkbox" id="topping6" value="Hành" name="toppingcheck">Hành</td>
 <td></td>
</tr>
<tr>
```

```

<td><input type="checkbox" id="topping7" value="Dứa"
 name="toppingcheck">Dứa</td>
<td></td>
</tr>
</table>

```

Đoạn mã xử lý các radio button cũng tương tự như đoạn mã xử lý các check box mà bạn đã thấy. Điểm khác biệt chủ yếu là các radio button có cùng chung một tên và nhóm logic, có nghĩa là các radio button được nhóm với nhau và tại một thời điểm chỉ có một radio button được chọn. Đoạn mã xử lý các radio button được thêm vào trong hàm *prepza()* trông như sau (Xem file radiobuttonjs.txt, phần Tài nguyên đi kèm):

```

//Biến crusttype - lưu kiểu đế bánh
var crusttype = document.forms["pizzaform"].crust; //crust -
var crustlength = crusttype.length;
for (var c = 0; c < crustlength; c++) {
 if (crusttype[c].checked) {
 var newelement = document.createElement("div");
 newelement.setAttribute("id", "crust" + c);
 document.body.appendChild(newelement);
 newelement.appendChild(document.createTextNode(
 + crusttype[c].value));
 }
}

```

## Kiểm tra tính hợp lệ của dữ liệu form

JavaScript thường xuyên được sử dụng để kiểm tra xem một trường trên form có được nhập dữ liệu chính xác hay không. Bạn đã gặp một ví dụ về kiểm tra tính hợp lệ trong chương này, trong đó form yêu cầu phải điền thông tin tên vào text box. Nếu không có thông tin nào được nhập vào trường yêu cầu, một cảnh báo lỗi sẽ xuất hiện. JavaScript là công cụ rất tốt để kiểm tra tính hợp lệ của dữ liệu ở phía client. Tuy nhiên, phía server lại hầu như không sử dụng JavaScript để thực hiện kiểm tra tính hợp lệ của dữ liệu.

Bạn đừng bao giờ giả định rằng những gì server nhận được là hợp lệ. Tôi không

thể đếm nổi có bao nhiêu lập trình viên web đã nói: “Chúng tôi đã sử dụng JavaScript để kiểm tra tính hợp lệ của dữ liệu, do đó không cần phải kiểm tra dữ liệu ở phía server nữa”. Giả định trên có lẽ đã không tính đến một số tình huống thực tế. Người dùng có thể tắt JavaScript trong trình duyệt hoặc họ cũng có thể gửi các dữ liệu định dạng POST và GET tới chương trình phía server mà không tuân theo sự điều hướng của giao diện trình duyệt. Dù bạn áp dụng những kỹ thuật gì ở phía client, đó cũng chỉ là những thủ thuật. Sẽ có người tìm được cách qua mặt những thủ thuật đó.

Điểm mấu chốt là bạn có thể và nên sử dụng JavaScript để kiểm tra trước tính hợp lệ của form. Việc kiểm tra trước tính hợp lệ rất hữu ích để phản hồi nhanh tới người dùng khi đoạn mã phát hiện ra những sai sót hiển nhiên ở dữ liệu đầu vào. Tuy vậy, bạn vẫn phải kiểm tra tất cả thông tin nhập vào ở phía server sau khi người dùng gửi dữ liệu.

Phần này chỉ bàn tới một số cách sử dụng JavaScript để kiểm tra tính hợp lệ phía client, nhưng trước khi trình bày, tôi sẽ đưa ra một số ví dụ minh họa sự nguy hiểm khi sử dụng JavaScript làm phương tiện duy nhất để kiểm tra tính hợp lệ của thông tin trên website.



## Vô hiệu hóa việc kiểm tra tính hợp lệ của JavaScript

Phần này sử dụng một chương trình phía server để tạo một hệ thống danh mục đơn hàng với ba thành phần đơn giản: sản phẩm, số lượng và giá. Mặt hàng là lá cỏ được lấy từ bãi cỏ nhà tôi. Nơi tôi ở vừa trải qua một mùa hè khô hạn, do đó thời điểm này số cỏ sót lại không còn nhiều - đa số là cây đại và cát, cơ bản nó không có vẻ gì là một bãi cỏ. Vì cỏ hiếm như vậy, nên các đơn hàng bị giới hạn, mỗi hộ gia đình chỉ được mua tối đa 3 lá cỏ và giá bán cũng rất cao. Tôi sẽ giới hạn số lượng mua này bằng cách sử dụng mã JavaScript.

Tôi đã tạo ra một trang web để bán cỏ. Khi dùng trình duyệt để mở, trang web trông giống như Hình 14-9.

## Đặt hàng - Windows Internet Explorer

http://localhost:50532/Chapti

Yahoo

Favorites

Đặt hàng

Đặt lá cỏ từ bãi cỏ của Steve Suehring



Mô tả: Steve rất ít chăm sóc bãi cỏ, vì thế bãi cỏ của ông không có nhiều cỏ. Số lượng cực kỳ hạn chế.

Giá: \$100.00 mỗi lá cỏ

Số lượng đặt hàng (giới hạn 3 lá cỏ mỗi hộ gia đình):

Đặt hàng

HÌNH 14-9 Một form danh mục đặt hàng.

Dưới đây là mã HTML và mã JavaScript để tạo trang web này (Bạn có thể xem thêm trong file validate.htm của Tài nguyên đi kèm). Hãy chú ý tới cách sử dụng `document.forms` (được in đậm) để truy cập tới giá trị được điền trong trường Số lượng. Ngoài ra, hãy để ý rằng bạn không thể gửi form về server vì action của form, file catalog.php, thực chất không tồn tại. Action của form không quan trọng trong ví dụ sau.

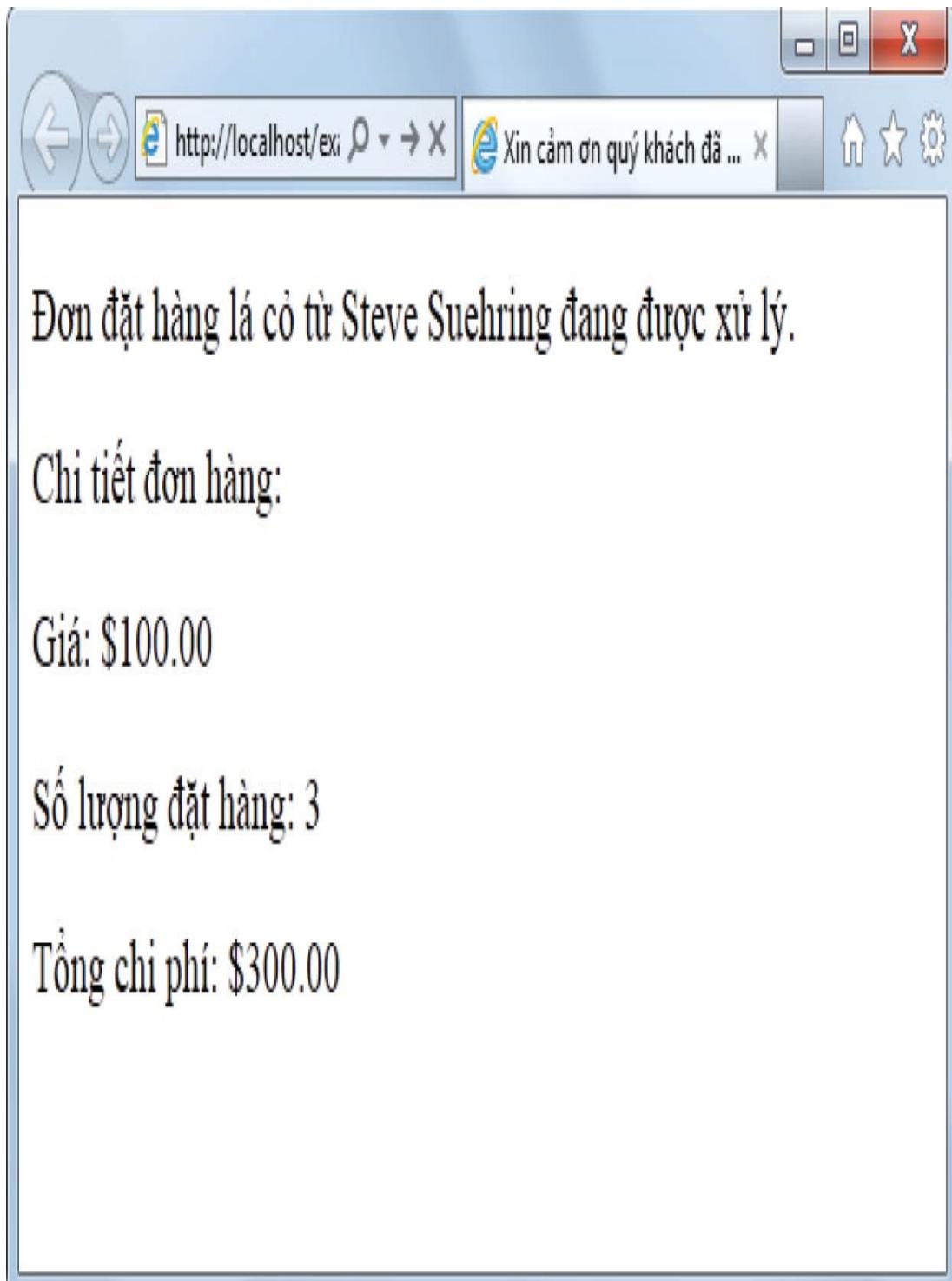
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Đặt hàng</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
 function formValid(eventObj) {
 if (document.forms["catalogform"]["quantity"]
 alert("Giới hạn 3 đơn vị/hộ gia đình");
 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
 } else {
 return true;
 }
 }
</script>
</head>
<body>
<form name="catalogform" id="catalogform" action="catalog.php"
 method="POST">
<p>Đặt lá cỏ từ bãi cỏ của Steve Suehring</p>
<div id="lawndiv"></div>
<p>Mô tả: Steve rất ít chăm sóc bãi cỏ, vì thế bãi cỏ của ông
không có nhiều cỏ. Số lượng cực kỳ hạn chế.</p>
<p>Giá: $100.00 mỗi lá cỏ</p>
<p>Số lượng đặt hàng (Giới hạn 3 lá cỏ mỗi gia đình): <input
type="text" name="quantity"></p>
```

```
<input type="submit" value="Đặt hàng">
</form>
<script type="text/javascript">
var formEl = document.getElementsByTagName("form")[0];
EHandler.add(formEl, "submit", function(eventObj) { formValid
</script>
</body>
</html>
```

**Chú ý** Bạn có thể cải tiến việc kiểm tra tính hợp lệ bằng cách đảm bảo người truy cập phải đặt hàng với số lượng tối thiểu là 1!

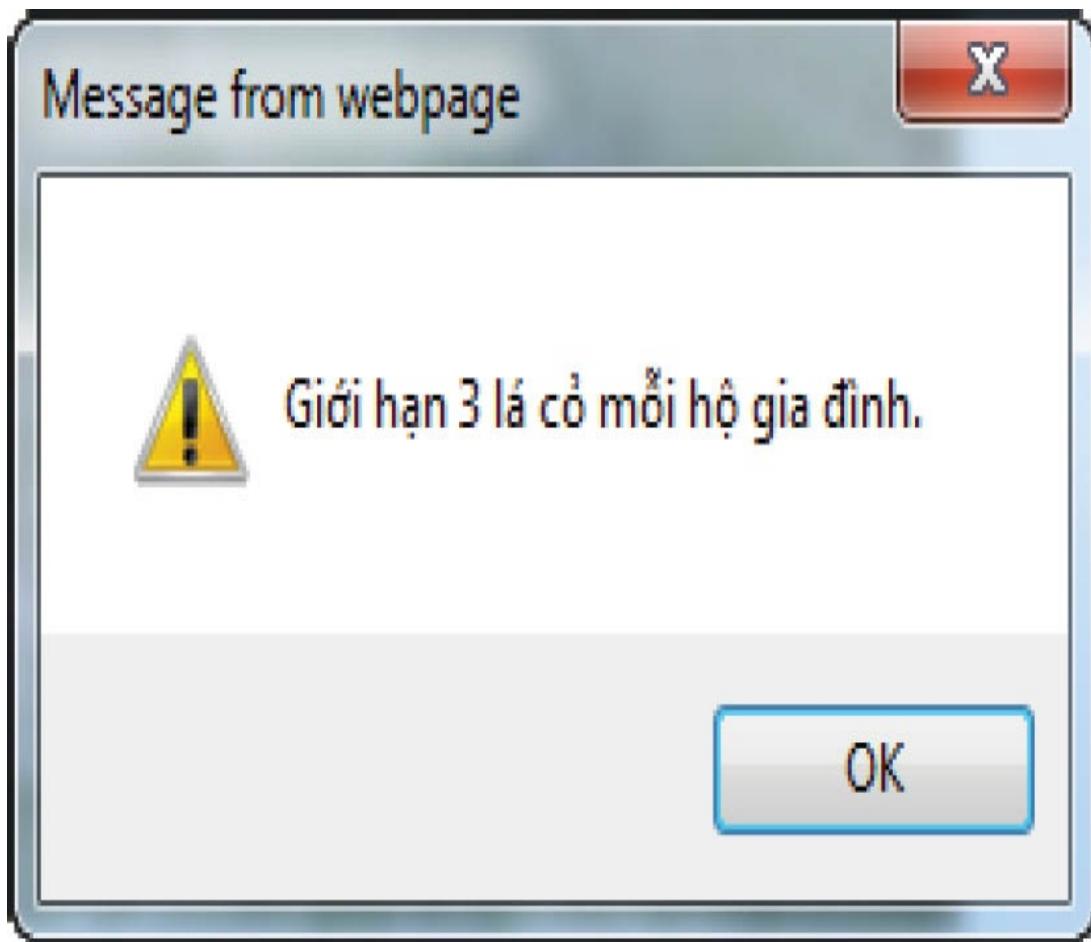
Khi JavaScript được kích hoạt trong trình duyệt, việc người dùng đặt hàng số lượng bằng 3 hoặc ít hơn là hoàn toàn hợp lệ, do đó form sẽ được gửi về kịch bản phía server để xử lý yêu cầu, sau đó trả về tổng số lượng đặt hàng, như trong Hình 14-10.





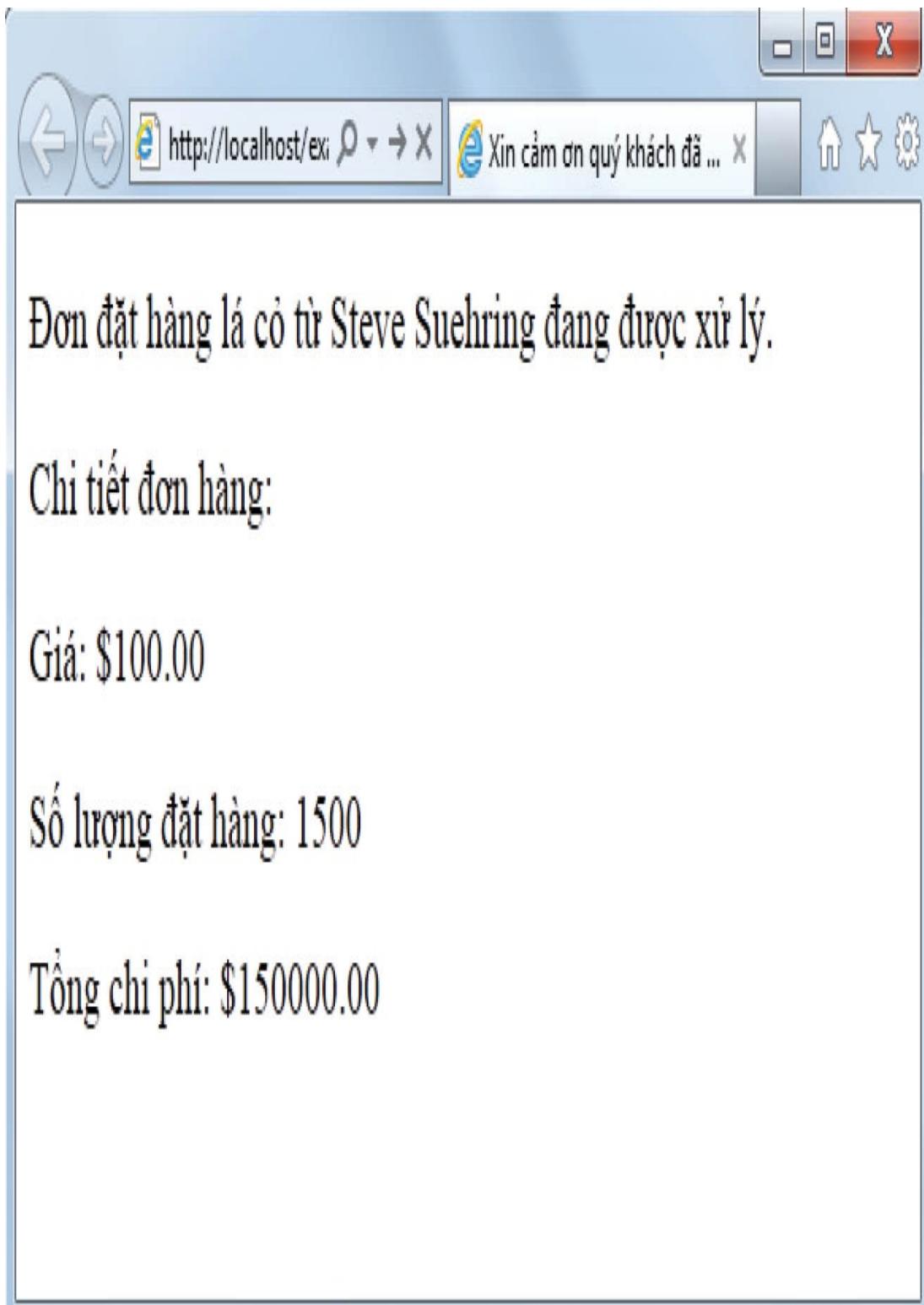
HÌNH 14-10 Đặt hàng số lượng 3 lá cỏ hoặc ít hơn là hợp lệ, kết quả trả về là tổng giá thành số sản phẩm đã đặt hàng.

Nếu người dùng quay trở lại trang web và JavaScrip vẫn đang được kích hoạt và thử đặt hàng số lượng 4 lá cỏ, khi đó một hộp thoại cảnh báo lỗi `alert()` sẽ xuất hiện giống như Hình 14-11 .



HINH 14-11 JavaScript thông báo lỗi nếu đặt hàng số lượng lớn hơn 3 lá cỏ.

Cho tới lúc này, trang web vẫn đang hoạt động rất tốt. Bây giờ hãy hình dung rằng tôi tắt JavaScript trong trình duyệt. Giao diện trang trông không có gì thay đổi khi tôi quay trở lại trang đặt hàng, trang vẫn giống như trong Hình 14-9. Tuy nhiên, bây giờ, tôi có thể đặt hàng số lượng 1500 sản phẩm. Đơn giản chỉ cần nhập 1500 vào ô số lượng và nhấn button Đặt hàng, kết quả trả về từ phía server đó là đơn hàng đã được xử lý, giống như trong Hình 14-12.



HÌNH 14-12 Vì JavaScript đã bị tắt, nên quá trình kiểm tra tính hợp lệ của đơn hàng bị bỏ qua trước khi dữ liệu được gửi tới server.

Do ở phía server không thực hiện kiểm tra tính hợp lệ của dữ liệu, nên giá trị đầu

vào mặc nhiên là hợp lệ, và đơn hàng được xử lý thành công. Vấn đề duy nhất ở đây đó là vườn nhà tôi không có đủ 1500 lá cỏ, do đó đơn hàng này không thể thực hiện.

Bạn có thể không đồng tình với kịch bản này, nhưng đây là một tình huống rất phổ biến trong các ứng dụng web. Trên thực tế, tình huống trong ví dụ này còn chưa nguy hiểm so với một số tình huống trong đó website cho phép người truy cập thay đổi đơn giá trong quá trình đặt hàng mà không quan tâm tới việc kiểm tra tính hợp lệ của dữ liệu – vì “không một ai làm như vậy”. Như vậy, người dùng có thể thoải mái thay đổi đơn giá sản phẩm nếu bạn không ngăn cản họ.

Bạn có thể đưa ra cách giải quyết cho vấn đề này đó là bắt buộc người truy cập phải kích hoạt JavaScript trước khi đặt hàng – tuy nhiên cách này sẽ không hiệu quả. Bạn có thể kiểm tra xem JavaScript được bật hay không, nhưng bạn sẽ không bao giờ có thể chắc chắn 100% về điều đó.

Cách duy nhất để giải quyết vấn đề này là kiểm tra tính hợp lệ của dữ liệu và thực thi các quy tắc kiểm tra ở phía server. Phía server phải kiểm tra các quy tắc nghiệp vụ để giới hạn số lượng đặt hàng. Thực hiện việc này không tốn nhiều thời gian, bạn chỉ cần làm một lần duy nhất - sau đó người dùng sẽ không thể đặt hàng vượt số lượng quy định.

Phần này đã cho thấy việc loại bỏ bước kiểm tra tính hợp lệ bằng JavaScript là rất dễ dàng, bạn chỉ cần tắt JavaScript trong trình duyệt. Phần tiếp theo sẽ trình bày cách sử dụng JavaScript để kiểm tra tính hợp lệ phía client. JavaScript chỉ nên được sử dụng để kiểm tra trước tính hợp lệ ở phía client và nó không nên được sử dụng làm phương thức duy nhất để kiểm tra tính hợp lệ của dữ liệu đầu vào.

## Kiểm tra tính hợp lệ của trường văn bản

Ở phần đầu chương, bạn đã gặp một ví dụ về cách kiểm tra tính hợp lệ của một trường văn bản. Nếu trường văn bản đó không được điền đầy đủ, hộp thoại thông báo `alert()` sẽ xuất hiện. Trong phần này, bạn sẽ học cách đưa ra các dòng phản hồi chèn trong tài liệu ngay cạnh trường văn bản thay vì sử dụng hộp thoại `alert()`.

Dưới đây là đoạn mã thực hiện điều này (bạn có thể tìm thấy đoạn mã này trong file mẫu catalog.htm của phần Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Đặt hàng</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
 function formValid(eventObj) {
 if (document.forms["catalogform"]["quantity"] > 3) {
 var submitbtn = document.forms["catalogform"].submit;
 var quantityp = document.getElementById("quantity");
 var errorel = document.createElement("div");
 errorel.appendChild(document.createTextNode(
 "Giới hạn 3 đơn vị/hộ gia đình"));
 quantityp.appendChild(errorel);
 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
 } else {
 return true;
 }
 }
</script>
</head>
<body>
<form name="catalogform" id="catalogform" action="catalog.php">
<p>Đặt lá cỏ từ bãi cỏ của Steve Suehring</p>
<div id="lawndiv"></div>
<p>Mô tả: Steve rất ít chăm sóc bãi cỏ, vì thế bãi cỏ của ông
 có nhiều cỏ. Số lượng cực kỳ hạn chế.</p>
<p>Giá: $100.00 mỗi lá cỏ</p>
<p>Số lượng đặt hàng (Giới hạn 3 lá cỏ mỗi gia đình): <input
 type="text" name="quantity"></p>
<input id="submitbutton" type="submit"
 value="Đặt hàng"></p>
</form>
<script type="text/javascript">
var formEl = document.getElementsByTagName("form")[0];
```

```
EHandler.add(formEl, "submit", function(eventObj) { formValid
</script>
</body>
</html>
```

**Mách nhỏ** Lưu ý rằng việc kiểm tra tính hợp lệ bằng JavaScript trong các ví dụ trên được kích hoạt bằng sự kiện *submit*. Sự kiện *submit* của toàn bộ form được ưu tiên hơn sự kiện *click* của button *Submit* vì nó được kích hoạt bất kể khi người truy cập nhấn vào button *Submit* hay nhấn phím Enter trên bàn phím.

Về cơ bản, đoạn mã trên giống hệt những gì bạn đã thấy trong các ví dụ trước. Đoạn mã này chỉ kiểm tra xem form có hợp lệ hay không. Nếu form không hợp lệ, đoạn mã tạo và thêm một phần tử HTML span cùng với dòng văn bản “Giới hạn 3 đơn vị/hộ gia đình” như trong Hình 14-13 thay vì hiển thị hộp thoại cảnh báo *alert()*.



## Đặt hàng - Windows Internet Explorer



http://localhost:50532/Chapter14



Yahoo



Favorites



Đặt lá cỏ từ bãi cỏ của Steve Suehring



Mô tả: Steve rất ít chăm sóc bãi cỏ, vì thế bãi cỏ của ông không có nhiều cỏ. Số lượng cực kỳ hạn chế.

Giá: \$100.00 mỗi lá cỏ

Số lượng đặt hàng (Giới hạn 3 lá cỏ mỗi hộ gia đình):

Giới hạn 3 lá cỏ mỗi hộ  
gia đình

HÌNH 14-13 Đưa ra phản hồi chèn vào trang web thay vì hiển thị hộp thoại thông báo `alert()`.

## Bài tập

1. Tạo một web form hiển thị hộp thoại thông báo `alert()` dựa trên dữ liệu người dùng chọn trên select box.
2. Thêm một số nhóm radio button vào form trong bài tập pizza ở phần đầu chương để có thêm ba kiểu kích thước pizza: cỡ nhỏ, cỡ trung bình và cỡ lớn. Hiển thị kết quả cùng với đơn đặt hàng pizza.
3. Thiết kế lại hệ thống đặt hàng bánh pizza với các button từ ví dụ pizza ban đầu, cho phép người đặt hàng có thể chọn các loại bánh Rau đặc biệt, Thịt đặc biệt hoặc Hawaii. Sau đó, những button này cần chọn ra chính xác thành phần chính tương ứng với loại bánh đã chọn trong các check box. Với bánh pizza Rau đặc biệt, chọn Hạt tiêu xanh, Nấm và Hành. Với bánh pizza Thịt đặc biệt, chọn Xúc xích thường, Xúc xích Ý và Dăm bông; còn với pizza Hawaii, chọn Dứa và Dăm bông.



# Chương 15

## JavaScript và CSS

Sau khi đọc xong chương này, bạn có thể:

- Nắm được nội dung cơ bản về CSS (Cascading Style Sheets).
- Nắm được quan hệ giữa JavaScript và CSS.
- Sử dụng JavaScript để thay đổi style của một phần tử đơn lẻ.
- Sử dụng JavaScript để thay đổi style của một nhóm phần tử.
- Sử dụng JavaScript để cung cấp phản hồi trực quan trên một trang web với CSS.

### CSS là gì?



CSS cho phép bạn có thể tùy chỉnh giao diện của một trang web. Sử dụng CSS, bạn có thể thay đổi màu sắc, font chữ và cách bố trí các thành phần trong trang web.

Hình 15-1 minh họa một trang web cơ bản. Trang web trông khá nhảm chán – ít nhất là ở bố cục.



HÌNH 15-1 Ví dụ về một trang web đơn giản chưa được áp dụng CSS.

Dùng CSS, bạn có thể thêm style, giúp tăng tính thẩm mỹ cho trang web trong Hình 15-1 mà không làm thay đổi nội dung của trang. Ví dụ, bạn có thể thay đổi font chữ cho tiêu đề và tạo điểm nhấn cho nội dung đặc biệt của trang bằng font chữ in đậm như Hình 15-2.



HÌNH 15-2 Trang web tương tự như trang trong Hình 15-1 nhưng đã được áp dụng style CSS.

## Sử dụng thuộc tính và bộ chọn

Cấu trúc cơ bản của CSS là tên thuộc tính CSS (property) sau là dấu hai chấm (:) và giá trị (value) của thuộc tính như sau:

tenthuoctinh: giatri

Bạn có thể thiết lập các thuộc tính style CSS. Trong Hình 15-2, thuộc tính *font-weight* tạo chữ in đậm ở dòng thứ hai. Bạn có thể tìm thấy danh sách đầy

đủ các thuộc tính và các giá trị hợp lệ tương ứng trên trang web của tổ chức World Wide Web Consortium (W3C):  
<http://www.w3.org/TR/CSS21/propidx.html>.

Bạn có thể áp dụng thuộc tính CSS cho một nhóm phần tử của tài liệu dựa trên kiểu phần tử (*<p>*, *<h1>*, *<a>*,...) hoặc cho từng phần tử riêng rẽ bằng cách chỉ định thuộc tính *class* hoặc thuộc tính *id* của phần tử. Những nhóm này được gọi chung là bộ chọn (*selector*).

Bộ chọn chỉ ra phần tử hay những phần tử cần được thiết lập các thuộc tính và giá trị cụ thể. Cấu trúc câu lệnh CSS cơ bản với bộ chọn có dạng như sau:

```
tenbochon { tenthuoctinh: giatri; }
```

Ví dụ, đoạn mã để chuyển toàn bộ các phần tử *<h1>* trong tài liệu sang font Arial sẽ như sau (xem file ex1.css, phần Tài nguyên đi kèm):

```
h1 { font-family: arial; }
```

Mặc dù việc áp dụng style cho toàn bộ kiểu phần tử nào đó thường có ích, nhưng bạn sẽ gặp những tình huống chỉ cần định kiểu trình bày cho một vài phần tử của một loại thẻ nhất định chứ không phải tất cả, hoặc cần định kiểu trình bày cho các phần tử thuộc cùng một loại thẻ theo các cách khác nhau. Bạn có thể thực hiện định kiểu trình bày chọn lọc bằng cách sử dụng thuộc tính *class* hoặc *id* của phần tử. Những thuộc tính này cho phép kiểm soát tận gốc cách hiển thị của từng phần tử trong tài liệu. Ví dụ, tài liệu có thể có nhiều phần tử *<p>*, nhưng bạn chỉ muốn một số phần tử *<p>* có font in đậm. Bằng cách sử dụng thuộc tính *class* với CSS thích hợp, bạn có thể áp dụng định dạng cụ thể cho các phần tử *<p>* thuộc lớp đó. Ví dụ, để tất cả phần tử thuộc lớp *boldParagraphs* được in đậm, bạn cần viết đoạn CSS như sau:

```
.boldParagraphs { font-weight: bold; }
```

Chú ý rằng bộ chọn cho một lớp CSS bắt đầu bằng dấu chấm. Style in đậm sau đó được áp dụng cho bất kỳ phần tử HTML nào có thuộc tính *class* chứa giá trị *boldParagraphs*:

```
class="boldParagraphs"
```

Đây là ví dụ một thẻ sẽ được áp dụng style trên:

< p class="boldParagraphs" > Đoạn này sẽ được in đậm. </ p >

Bạn thậm chí có thể kiểm soát chi tiết hơn với thuộc tính *id*, từ đó bạn có thể chọn một phần tử cụ thể với ID biết trước và áp dụng style cho nó, như ở Hình 15-2. Cả hai đoạn văn bản “JavaScript - Hướng dẫn học qua ví dụ” là một cuốn sách viết bởi Steve Suehring, do Microsoft Press xuất bản” và “Cuốn sách nhấn mạnh việc lập trình JavaScript theo chuẩn để mã JavaScript có thể chạy trên nhiều nền tảng, trên nhiều trình duyệt khác nhau” đều nằm trong phần tử *<p>*. Tuy nhiên, câu đầu tiên có ID là *tagline*, nhờ đó nó được in đậm bằng CSS. Sau đây là đoạn HTML:

< p id="tagline" > JavaScript - Hướng dẫn học qua ví dụ là một .

Và đoạn CSS (có trong file ex2.css, phần Tài nguyên đi kèm):

```
#tagline { font-weight: bold; }
```

Chú ý rằng bộ chọn cho *id* bắt đầu bằng một dấu thăng (#).

## Áp dụng CSS



Có một số cách áp dụng style cho một tài liệu với CSS. Bạn có thể:

- Áp dụng trực tiếp style cho phần tử HTML trong chính phần tử đó.
- Đưa khối thẻ *<style>* vào phần *<head>* của tài liệu.
- Liên kết đến một file CSS ngoài tương tự như cách liên kết đến một file JavaScript ngoài.

Cho đến nay, cách tốt nhất vẫn là sử dụng file CSS ngoài – tương tự như cách tốt nhất để lập trình với JavaScript là sử dụng file JavaScript ngoài. Sử dụng file CSS ngoài giúp tăng khả năng dùng lại và giúp đơn giản hóa đáng kể quá trình bảo trì trang web. Giả sử nhiệm vụ của bạn là quản lý website của một công ty có hàng trăm trang web và bây giờ bạn được yêu cầu thay đổi thiết kế website, ví dụ thay đổi font chữ phần tiêu đề của các trang. Nếu website của bạn sử dụng chung một file CSS ngoài, việc thay đổi sẽ rất nhanh chóng và dễ dàng vì bạn chỉ cần thay đổi trong một file. Nếu CSS nằm trong từng tài liệu, thì chỉ một thay đổi nhỏ cũng đòi hỏi khá nhiều thời gian.

Còn rất nhiều kiến thức về CSS so với nội dung mà một cuốn sách về JavaScript

có thể đề cập. Nếu bạn chưa từng làm việc với CSS, và muốn tìm hiểu thêm thông tin, bạn có thể đọc bài viết “CSS Overviews and Tutorials” (Tổng quan và hướng dẫn về CSS) trên trang web của Microsoft (<http://msdn2.microsoft.com/en-us/library/ms531212.aspx>).

## Mối quan hệ giữa JavaScript và CSS

Bạn có thể sử dụng JavaScript để tác động tới nhiều style của tài liệu bằng cách sử dụng Mô hình đối tượng tài liệu (DOM) (Xem lại Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt”). Với DOM, bạn có thể truy xuất một phần tử bằng tên thẻ hoặc ID của nó và thiết lập thuộc tính style của phần tử.

Ví dụ, đoạn tiêu đề trong Ví dụ 15-1 có phần tử `<h1>`. Nếu bạn đặt một ID đại diện cho phần tử `<h1>` đó, ví dụ như `heading`, bạn có thể truy xuất nó bằng phương thức `getElementById()` của JavaScript. Sau đó, bạn có thể sử dụng thuộc tính `style` của phần tử đó để truy xuất đối tượng `style`, đây là một cách thay đổi style cho phần tử bằng JavaScript. Sau đây là ví dụ chuyển style sang một font khác:

```
var heading = document.getElementById("heading");
heading.style.fontFamily = "arial";
```

### Thiết lập style cho phần tử thông qua ID

Sử dụng hàm `getElementById()` và đối tượng `style`, như trong ví dụ trên, là một cách dễ dàng và hiệu quả để thiết lập style cho phần tử. Bạn có thể áp dụng style bằng cách thiết lập các thuộc tính cho đối tượng `style`, tuy nhiên tên các thuộc tính đó không phải lúc nào cũng giống tên các thuộc tính tương ứng trong CSS. Khi tên thuộc tính style trong JavaScript là một từ đơn, nó thường giống tên thuộc tính style trong CSS chuẩn, ví dụ như `margin` (lề). Tuy nhiên, khi tên thuộc tính CSS có dấu gạch ở giữa, ví dụ `text-align` (căn chữ), tên thuộc tính trong JavaScript sẽ trở thành `textAlign`. Lưu ý dấu gạch ngang được bỏ đi và ký tự viết hoa được dùng để phân biệt từ chính với các từ còn lại. Cách đặt tên

thuộc tính như trên được gọi là *camelCase*.

**Bảng 15-1** Trình bày một số thuộc tính CSS và các thuộc tính tương ứng trong JavaScript.

| Thuộc tính CSS               | Thuộc tính tương ứng trong JavaScript                                                             |
|------------------------------|---------------------------------------------------------------------------------------------------|
| <i>background</i>            | <i>background</i>                                                                                 |
| <i>background-attachment</i> | <i>backgroundAttachment</i>                                                                       |
| <i>background-color</i>      | <i>backgroundColor</i>                                                                            |
| <i>background-image</i>      | <i>backgroundImage</i>                                                                            |
| <i>background-repeat</i>     | <i>backgroundRepeat</i>                                                                           |
| <i>border</i>                | <i>border</i>                                                                                     |
| <i>border-color</i>          | <i>borderColor</i>                                                                                |
| <i>color</i>                 | <i>color</i>                                                                                      |
| <i>font-family</i>           | <i>fontFamily</i>                                                                                 |
| <i>font-size</i>             | <i>fontSize</i>                                                                                   |
| <i>font-weight</i>           | <i>fontWeight</i>                                                                                 |
| <i>height</i>                | <i>height</i>                                                                                     |
| <i>left</i>                  | <i>left</i>                                                                                       |
| <i>list-style</i>            | <i>listStyle</i>                                                                                  |
| <i>list-style-image</i>      | <i>listStyleImage</i>                                                                             |
| <i>margin</i>                | <i>margin</i>                                                                                     |
| <i>margin-bottom</i>         | <i>marginBottom</i>                                                                               |
| <i>margin-left</i>           | <i>marginLeft</i>                                                                                 |
| <i>margin-right</i>          | <i>marginRight</i>                                                                                |
| <i>margin-top</i>            | <i>marginTop</i>                                                                                  |
| <i>padding</i>               | <i>padding</i>                                                                                    |
| <i>padding-bottom</i>        | <i>paddingBottom</i>                                                                              |
| <i>padding-left</i>          | <i>paddingLeft</i>                                                                                |
| <i>padding-right</i>         | <i>paddingRight</i>                                                                               |
| <i>padding-top</i>           | <i>paddingTop</i>                                                                                 |
| <i>position</i>              | <i>position</i>                                                                                   |
| <i>float</i>                 | <i>styleFloat</i> (trong Windows Internet Explorer); <i>cssFloat</i> (trong các trình duyệt khác) |
| <i>text-align</i>            | <i>textAlign</i>                                                                                  |
| <i>top</i>                   | <i>top</i>                                                                                        |
| <i>visibility</i>            | <i>visibility</i>                                                                                 |

---

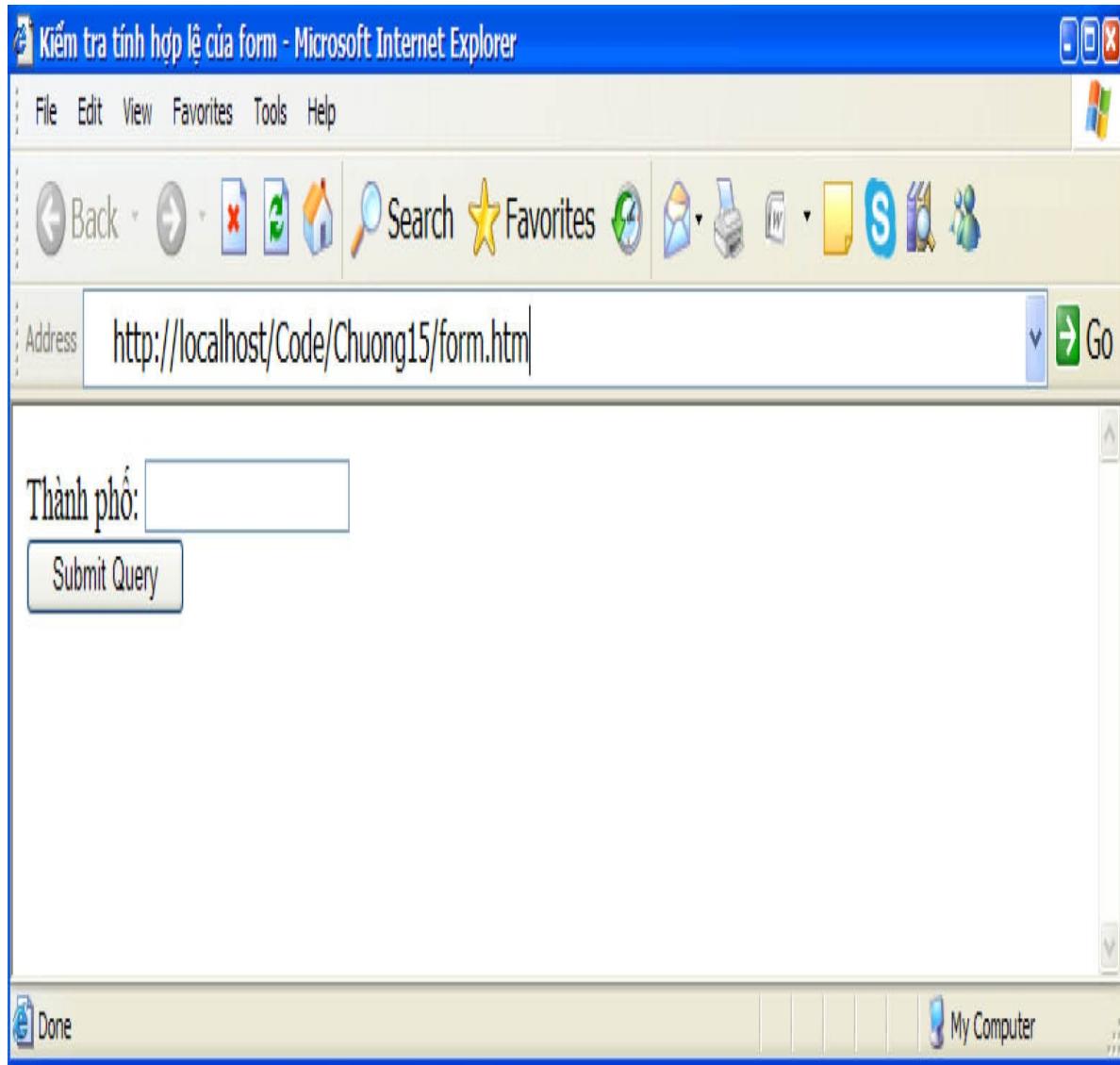
Một cách dùng JavaScript phổ biến là kiểm tra tính hợp lệ cho dữ liệu nhập vào form. Sử dụng CSS với JavaScript có thể giúp bạn tránh dùng các hộp thoại `alert()` và thay vào đó hiển thị phản hồi trực tiếp trên trang ngay cạnh phần thông tin nhập sai. Bài tập tiếp theo sẽ hướng dẫn bạn cách sử dụng tính năng này.

### Sử dụng CSS và JavaScript để kiểm tra tính hợp lệ của form

1. Dùng Visual Studio, Eclipse hoặc một trình soạn thảo khác thay đổi file form.html trong thư mục mã nguồn mẫu Chương 15 (Tài nguyên đi kèm).
2. Trong trang đó, thay dòng chú thích TODO bằng phần HTML in đậm dưới đây: (Bạn có thể tìm thấy đoạn mã này trong file form.txt trong Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Kiểm tra tính hợp lệ của form</title>
</head>
<body>
<form name="formexample" id="formexample" action="#">
<div id="citydiv">Thành phố: <input id="city" name="city">
<div><input id="submit" type="submit"></div>
</form>
</body>
</html>
```

3. Dùng trình duyệt để mở trang. Trang web sẽ như sau:



4. Tạo file JavaScript trong thư mục mà bạn đã lưu file form.html. Đặt tên file này là **form.js**.
5. Trong file form.js, thêm đoạn mã dưới đây. Nếu muốn, bạn có thể thay đổi giá trị Stevens Point đang được dùng để kiểm tra biến *cityField* bằng một giá trị khác. Lưu file.

```
function checkValid(eventObj) { //Kiểm tra tính hợp lệ
 var cityField = document.forms[0]["city"];
 if (cityField.value != "Stevens Point") {
 var cityDiv = document.getElementById("ci
 cityDiv.style.fontWeight = "bold";
 cityDiv.style.border = "1px solid black";
```

```

 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
 } else {
 return true;
 }
}

```

- Mở lại file form.html và tạo liên kết đến file JavaScript ngoài, sau đó tạo thêm sự kiện *submit* cho tài liệu. File form.html sẽ như dưới đây (những phần thay đổi được in đậm):

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Kiểm tra tính hợp lệ của form</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript" data-src="form.js"></script>
</head>
<body>
<form name="formexample" id="formexample" action="#">
<div id="citydiv">Thành phố: <input id="city" name="city">
<div><input id="submit" type="submit"></div>
</form>
<script type="text/javascript">
var formEl = document.getElementsByTagName("form")[0];
EHandler.add(formEl,"submit", function(eventObj) { checkVal...
</script>
</body>
</html>

```

- Tải lại trang form.html trên trình duyệt của bạn. Trong trường nhập dữ liệu *Thành phố*, nhập chữ **test** và nhấn Submit Query. Bạn sẽ thấy ngay nhữn Thành phố được in đậm và có viền xung quanh thẻ *div*.
- Nhập lại vào trường *Thành phố* giá trị **Stevens Point** (hoặc bất cứ giá trị nào đã thiết lập ở bước 5) và nhấn Submit Query. Màu nền được thay đổi và trường nhập trống. Nguyên nhân là vì form tiếp tục hoạt động sau khi gửi dữ

liệu đi (trong trường hợp này, form không thực hiện gì cả).

Đoạn mã trong ví dụ trên chỉ là một biến thể của đoạn mã được dùng trước đó có bổ sung thêm file JavaScript ngoài để thực hiện kiểm tra tính hợp lệ và trả về phản hồi bằng CSS. Trong file JavaScript là đoạn mã kiểm tra tính hợp lệ. Đầu tiên, đoạn mã truy xuất *đối tượng trường văn bản* từ form. Tiếp theo, giá trị nhập vào trường này sẽ được kiểm tra xem có trùng với Stevens Point hay không. Nếu giá trị không phải là Stevens Point, đoạn mã đổi thuộc tính định kiểu trình bày *font-weight* của trường văn bản thành in đậm (bold) và thêm thuộc tính viền (border).

Cách làm này có hạn chế là style CSS cho các phần tử lại được thiết lập trong đoạn mã JavaScript. Thực tế là, việc bảo trì sẽ dễ hơn rất nhiều khi bạn phân tách phần nội dung (HTML), style (CSS) và hành vi (JavaScript). Bạn có thể cải tiến ví dụ này bằng cách thiết lập style với bộ chọn dựa trên kiểu phần tử, hoặc bằng cách tạo một lớp bao lõi chung trong CSS và dùng mã JavaScript để áp dụng lớp đó. Phần tiếp theo sẽ lần lượt đề cập đến các phương thức này.

## Thiết lập style cho phần tử thông qua kiểu phần tử

Dù việc thiết lập style cho phần tử thông qua ID là cách phổ biến để thay đổi style trong JavaScript, bạn cũng cần biết cách thiết lập thuộc tính cho tất cả các phần tử thuộc một kiểu nào đó.

Hãy nhớ lại các hình chụp màn hình ở phần đầu của chương này, cụ thể là Hình 15-2. Ví dụ 15-1 là đoạn mã HTML cho trang đó.

### VÍ DỤ 15-1 Đoạn HTML cho Hình 15-2.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Lập trình với JavaScript</title>
<style type="text/css">
h1 { font-family: arial; }

#tagline { font-weight: bold; }
```

```
</style>
</head>
<body>

<h1 id="heading">JavaScript - Hướng dẫn học qua ví dụ</h1>
<p id="tagline">"JavaScript - Hướng dẫn học qua ví dụ" là cu
<p>Cuốn sách nhấn mạnh việc lập trình JavaScript theo chuẩn i
</body>
</html>
```

Chú ý đến hai phần tử *<p>* trong Ví dụ 15-1. Phần tử *<p>* đầu tiên được áp dụng style *font-weight: bold*. Bạn có thể sử dụng JavaScript để áp dụng thêm các style cho tất cả các phần tử *<p>*. Xem xét đoạn mã trong Ví dụ 15-2, trong đó có thêm một số đoạn mã JavaScript (được in đậm) để thay đổi kiểu font của tập hợp các phần tử *<p>*.

## VÍ DỤ 15-2 Sử dụng JavaScript và HTML để thay đổi style của phần tử.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Lập trình với JavaScript</title>
<style type="text/css">
h1 { font-family: arial; }

#tagline { font-weight: bold; }
</style>
</head>
<body>

<h1 id="heading">JavaScript - Hướng dẫn học qua ví dụ</h1>
<p id="tagline">"JavaScript - Hướng dẫn học qua ví dụ" là cu
<p>Cuốn sách nhấn mạnh việc lập trình JavaScript theo chuẩn i
<script type="text/javascript">
var pElements = document.getElementsByTagName("p");
for (var i = 0; i < pElements.length; i++) {
 pElements[i].style.fontFamily = "arial";
}
</script>
</body>
</html>
```

Khi xem trên trình duyệt web, trang này hiển thị hai phần tử `<p>` với font Arial như minh họa trong Hình 15-3.

Đoạn mã JavaScript trong ví dụ này khá đơn giản vì chỉ sử dụng các hàm mà bạn đã gặp. Đầu tiên, đoạn mã truy xuất phần tử `<p>` sử dụng phương thức `getElementsByName()` của DOM và lưu vào biến `pElements`. Sau đó, đoạn mã duyệt qua các phần tử trong biến `pElements` bằng vòng lặp `for`, thay đổi thuộc tính `style.fontFamily` của mỗi phần tử thành Arial.



HÌNH 15-3 Dùng JavaScript để cùng lúc thay đổi kiểu font của nhiều phần tử.

# Thiết lập các lớp CSS với JavaScript

Tuân theo đúng định hướng lập trình tách biệt nội dung và ngôn ngữ đánh dấu (HTML) khỏi phần style (CSS) và phần mã xử lý hành vi (JavaScript), một giải pháp tốt hơn nữa cho việc thay đổi style của các phần tử là tạo lớp trong CSS và dùng JavaScript áp dụng lớp đó khi cần, thay vì thay đổi các thuộc tính cụ thể như *font-weight* và *size* bằng JavaScript. Phần này hướng dẫn cách thêm cũng như gỡ bỏ các lớp CSS cho phần tử.

Hãy nhớ bạn đã từng tạo lớp CSS như dưới đây:

```
.errorClass {
 font-weight: bold;
 border: 1px solid black;
}
```

Bạn có thể áp dụng JavaScript cho lớp này bằng cách sử dụng thuộc tính *className* như sau:

```
//truy xuất phần tử tagline
var tagLineElement = document.getElementById("tagline");
tagLineElement.className += "errorClass";
```

Chú ý cách dùng toán tử `+=` trong đoạn mã. Toán tử này sẽ *thêm* lớp *errorClass* vào thuộc tính *className* nhưng không *ghi đè* lên các lớp đã tồn tại.

Để gỡ bỏ lớp khỏi phần tử, bạn cần dùng phương thức *replace()* và biểu thức chính quy. Bạn truy xuất phần tử như phần trước và truy xuất danh sách các lớp mà phần tử đó thuộc về bằng cách sử dụng thuộc tính *className*. Cuối cùng, bạn thay thế tên lớp muốn gỡ bỏ bằng một biểu thức chính quy:

```
//truy xuất phần tử tagline
var tagLineElement = document.getElementById("tagline");
tagLineElement.className = tagLineElement.className.replace
(/\berrorClass\b/, "");
```

Ví dụ này gỡ bỏ tên lớp *errorClass* khỏi thuộc tính *className* của phần tử *tagLineElement* bằng biểu thức chính quy. Biểu thức chính quy tìm biên của một từ (`\b`), theo sau là chuỗi *errorClass* và sau đó là biên của một từ khác (`\b`). Đoạn mã thay thế bất kỳ từ trùng khớp nào bằng một chuỗi trắng ("").

# Truy xuất style của phần tử bằng JavaScript

Bạn có thể truy xuất các style của phần tử bằng JavaScript; tuy nhiên, phương thức để truy xuất style trong Internet Explorer và các trình duyệt khác không giống nhau. Với các trình duyệt theo chuẩn W3C, có thể truy xuất style bằng phương thức `getComputedStyle()`; với Internet Explorer, bạn cần sử dụng thuộc tính mảng `currentStyle`. Style được truy xuất là style cuối cùng được áp dụng vì nó là style tổng hợp từ tất cả các nguồn CSS bao gồm các file CSS bên ngoài và trang style (CSS) bên trong tài liệu.

Ví dụ 15-3 hướng dẫn cách truy xuất thuộc tính CSS `color` đã tổng hợp và tính toán của một phần tử có ID là `heading`. Trong ví dụ này, tiêu đề là một phần tử `<h1>`:

```
<h1 style="font-family: arial; color: #0000FF;" id="heading">
```

Trong Ví dụ 15-3, kết quả được hiển thị trong hộp thoại thông báo `alert()`.

**VÍ DỤ 15-3** Dùng JavaScript để truy xuất thuộc tính CSS `color`.

```
var heading = document.getElementById("heading");
if (typeof heading.currentStyle != "undefined") {
 var curStyle = heading.currentStyle.color;
} else if (typeof window.getComputedStyle != "undefined") {
 var curStyle =
 document.defaultView.getComputedStyle(
 heading, null).getPropertyValue("colo
} alert(curStyle);
```

Khi dùng trình duyệt xem trang, bạn sẽ thấy một hộp thoại thông báo như Hình 15- 4. Phương thức `getComputedStyle()` nhận hai đối số: phần tử được truy xuất và phần tử giả. Trong hầu hết các trường hợp, chúng ta chỉ sử dụng duy nhất phần tử được truy xuất, do đó chúng ta có thể bỏ qua tham số thứ hai bằng cách không thiết lập giá trị cho nó, như trong ví dụ trên.



Hình 15-4 Style hiện đang được áp dụng cho một phần tử.

**Chú ý** Firefox trả về  $rgb(0, 0, 255)$  cho đoạn mã tương tự để biểu diễn giá trị màu (*color*).

## Chỉnh sửa style sheet bằng JavaScript

Các ví dụ trong phần đầu chương này hướng dẫn cách làm việc với các phần tử style độc lập thông qua đối tượng *style*. Tuy nhiên, có thể bạn muốn thay đổi toàn bộ style cho một hoặc nhiều phần tử, hay nói cách khác, bạn muốn thay đổi style sheet (file chứa định nghĩa của các style CSS) áp dụng cho một phần tử hoặc một lớp các phần tử. Việc này không dễ thực hiện như cách chỉnh sửa style đã được học ở những phần trước.

Khó khăn đầu tiên là bạn phải xác định trình duyệt của người truy cập có hỗ trợ việc truy xuất các style sheet đã tồn tại hay không. Bạn có thể làm việc này bằng cách kiểm tra thuộc tính *document.styleSheets* như dưới đây:

```
if (typeof document.styleSheets != "undefined") {
 //Trình duyệt hỗ trợ truy xuất style sheet.
```

}

Mảng `document.styleSheets` chứa các file style sheet CSS áp dụng cho một tài liệu, liệt kê theo thứ tự được áp dụng. Điều này có nghĩa là các style sheet CSS ngoài khi liên kết trong tài liệu sẽ được áp dụng theo thứ tự xuất hiện, bắt đầu từ chỉ số 0. Xem đoạn mã dưới đây:

```
<link rel="stylesheet" href="ex1.css" type="text/css" />
<link rel="stylesheet" href="ex2.css" type="text/css" />
```

Các style sheet này được đặt thứ tự tương ứng là `document.styleSheets[0]` và `document.styleSheets[1]`. Do đó, việc biết thứ tự các style sheet trong tài liệu là cần thiết nếu muốn truy xuất style áp dụng cho phần tử trong tài liệu đó.

Sau khi xác định trình duyệt có hỗ trợ style sheet hay không, bạn cần giải quyết sự khác biệt giữa Internet Explorer và các trình duyệt theo chuẩn W3C.

Internet Explorer cho phép truy xuất các định nghĩa style khai báo trong một style sheet bằng mảng `rules`, ngược lại, các trình duyệt theo chuẩn W3C truy xuất các định nghĩa này bằng mảng `cssRules`. Tương tự việc bạn phải lập trình để hỗ trợ sự khác biệt trong mô hình sự kiện giữa các trình duyệt, bạn cũng phải lập trình để xử lý sự khác biệt giữa các trình duyệt khi truy xuất style sheet CSS. Giả sử bạn có một định nghĩa style khai báo như sau:

```
h1 { font-family: arial; }
```

Ví dụ 15-4 chỉ ra cách truy xuất style sheet đầu tiên từ một tài liệu.

#### VÍ DỤ 15-4 Truy xuất style sheet CSS bằng JavaScript.

```
if (typeof document.styleSheets != "undefined") {
 var stylerules;
 if (typeof document.styleSheets[0].rules != "undefined")
 stylerules = document.styleSheets[0].rules
 } else {
 stylerules = document.styleSheets[0].cssRules
 }
}
```

Đoạn mã dưới đây, cùng với Ví dụ 15-4, thay đổi font của từng phần tử cụ thể trong CSS đó sang font khác:

```
stylerules[0].style.fontFamily = "courier";
```

Thay đổi *tất cả* các bộ chọn trong file style sheet CSS thành một thiết lập là cách làm không phổ biến. Duyệt qua file style sheet để tìm bộ chọn cụ thể thường hữu dụng hơn, như ở Ví dụ 15-5.

**VÍ DỤ 15-5** Duyệt style sheet để tìm bộ chọn *<h1>*.

```
for (var i = 0; i < stylerules.length; i++) {
 if (stylerules[i].selectorText.toLowerCase() == "h1") {
 stylerules[i].style.fontFamily = "courier"
 }
}
```

Dưới đây là ví dụ hoàn chỉnh hơn của tính năng này. Giả sử, chúng ta có một file style sheet CSS ngoài có tên là ex1.css cho ví dụ này:

```
h1 { font-family: arial; }
```

Ví dụ 15-6 chỉ ra trang HTML sử dụng hai style sheet ex1.css và ex2.css.

**VÍ DỤ 15-6** Thay đổi style của phần tử thông qua mảng *styleSheets*.

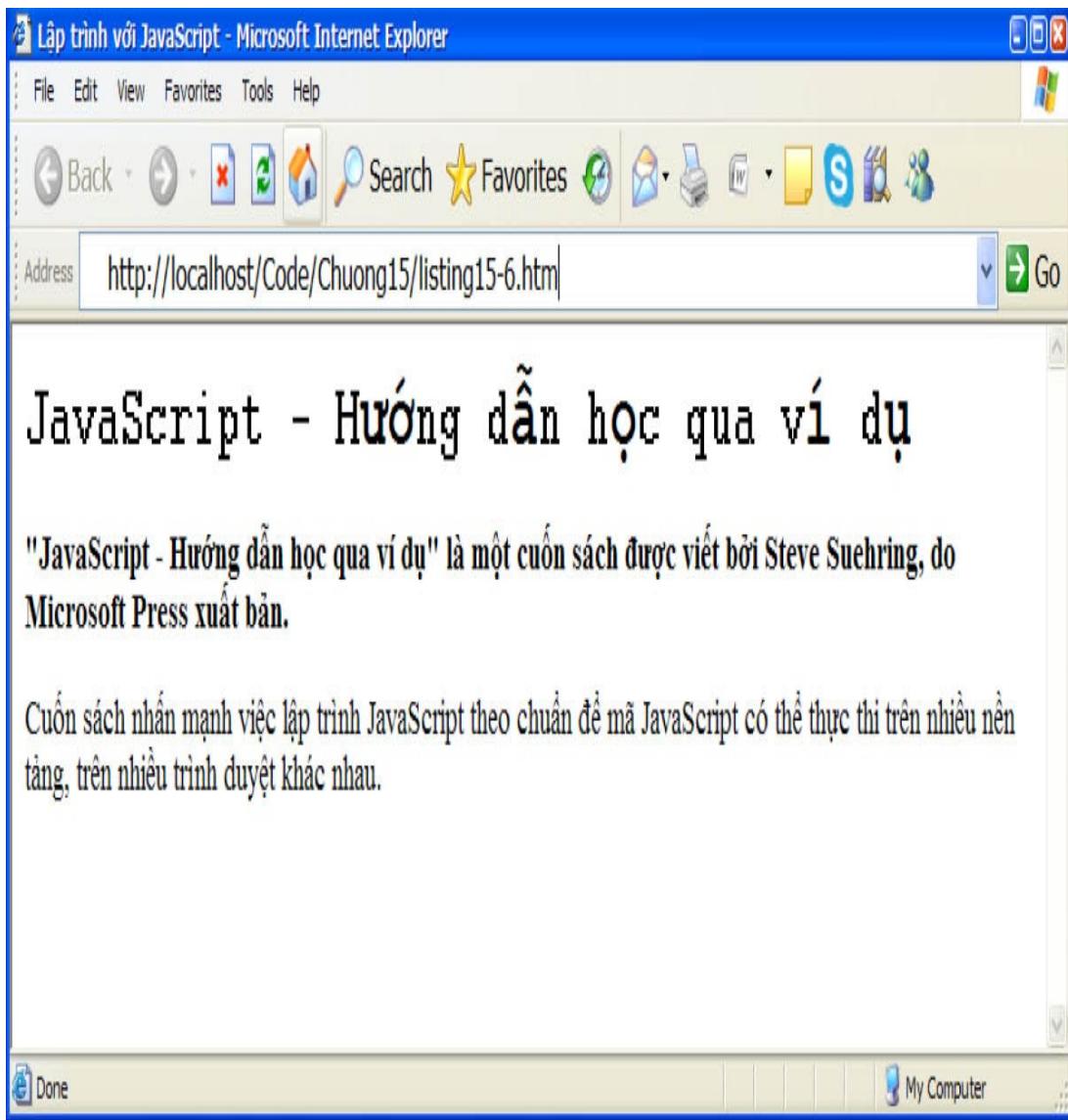
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Lập trình với JavaScript</title>
<link rel="stylesheet" href="ex1.css" type="text/css" />
<link rel="stylesheet" href="ex2.css" type="text/css" />
<script type="text/javascript">
if (typeof document.styleSheets != "undefined") {
 var stylerules;
 if (typeof document.styleSheets[0].rules != "undefined")
 stylerules = document.styleSheets[0].rules
 } else {
 stylerules = document.styleSheets[0].cssRules
 }
 for (var i = 0; i < stylerules.length; i++) {
 if (stylerules[i].selectorText.toLowerCase() == "h1")
 stylerules[i].style.fontFamily = "courier"
 }
}</script>
```

```
 }
 }
</script>
</head>
<body>

<h1 id="heading">JavaScript - Hướng dẫn học qua ví dụ</h1>
<p id="tagline">"JavaScript - Hướng dẫn học qua ví dụ" là
<p>Cuốn sách nhấn mạnh việc lập trình JavaScript theo chuẩn
</body>
</html>
```

Khi được xem trên trình duyệt, trang web sẽ hiển thị một tiêu đề với định dạng font *Courier*. Dòng chữ này được thay đổi nhờ đoạn mã JavaScript trong vòng lặp *for* như trong Ví dụ 15-6. Bạn có thể thấy kết quả ở Hình 15-5.





HÌNH 15-5 Sử dụng mảng `styleSheets` để truy cập bộ chọn.

## Bài tập

1. Tạo một tài liệu HTML đơn giản sử dụng một style sheet ngay trong tài liệu đó hoặc thông qua một file ngoài. Trang đó phải có ít nhất hai phần tử `<p>` và một phần tử `<h1>`. Đặt thuộc tính ID cho mỗi phần tử.
2. Dùng JavaScript để thay đổi style của một phần tử `<p>`, đổi thuộc tính `color` sang màu xanh dương.

3. Dùng JavaScript để thay đổi style của tất cả phần tử `<p>`, thay đổi thuộc tính hiển thị để ẩn chúng (Xem lại Bảng 15-1 để tìm hiểu về thuộc tính hiển thị).
4. Dùng JavaScript để truy xuất style hiện tại cho thuộc tính hiển thị của phần tử `<p>` và hiển thị giá trị thuộc tính hiển thị hiện tại bằng hộp thoại thông báo `alert()`.



# Chương 16

# Xử lý lỗi trong JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Nắm được các phương thức xử lý lỗi trong JavaScript: *try/catch* và *onerror*.
- Xử lý lỗi với câu lệnh *try/catch*.
- Sử dụng câu lệnh *try/catch/finally*.
- Xử lý sự kiện *onerror* của đối tượng window và image.

## Giới thiệu hai cách xử lý lỗi

Chương này giới thiệu hai cách chính để xử lý lỗi trong JavaScript: *try/catch* và *onerror*. Các ngôn ngữ khác như Microsoft Visual Basic .NET và Microsoft Visual C# cũng sử dụng *try/catch* để bắt và xử lý lỗi. Sự kiện *onerror* cho phép bạn thực thi một action khi gặp lỗi.

## Sử dụng *try/catch*

Phần *try* trong nhóm câu lệnh *try/catch* bao quanh một khối mã JavaScript. Khi đoạn script thực thi, bất kỳ ngoại lệ nào xuất hiện trong khối lệnh *try* đều được bắt lại bởi câu lệnh *catch*. Tiếp đó, bạn xử lý lỗi này bằng đoạn mã JavaScript trong khối *catch*. Cú pháp của *try/catch* như sau:

```
try {
 // Thực thi đoạn mã
}
catch(errorObject) {
```

```
// Viết mã để xử lý lỗi ở đây
}
```

Khi mã trong mệnh đề *try* thực thi, bất kỳ lỗi nào ảnh hưởng đến quá trình sẽ ngay lập tức được mệnh đề *catch* xử lý. Hãy xem Ví dụ 16-1 (Ví dụ này có trong file listing16-1.txt, phần Tài nguyên đi kèm):

### VÍ DỤ 16-1 Ví dụ *try/catch* đơn giản.

```
try {
 var numField = document.forms[0]["num"];
 if (isNaN(numField.value)) {
 throw "Không phải số";
 }
}
catch(errorObject) {
 alert(errorObject);
}
```

Khi giá trị của *numField.value* không phải là số, câu lệnh *throw* đưa ra một ngoại lệ do lập trình viên viết sẵn, đó là dòng thông báo “Không phải số”. Mệnh đề *catch* sau đó thực thi và trong trường hợp này nó trả về hộp thoại *alert()*. Chú ý sự khác nhau giữa ngoại lệ viết bởi lập trình viên (*throw*) và ngoại lệ được tạo bởi bộ thông dịch JavaScript tại thời điểm chạy, ví dụ như lỗi cú pháp. Khối lệnh *try/catch* không bắt lỗi cú pháp, do vậy chúng ta không thể dùng nó để xử lý những lỗi này.

Khi sử dụng mệnh đề *catch*, ta có thể thực thi nhiều công việc một lúc, ví dụ như gọi đến hàm khác để ghi lại lỗi hay xử lý tình trạng lỗi theo cách thức chung. Sử dụng *catch* đặc biệt có ích trong những đoạn mã phức tạp hay trong những đoạn mã dễ gây lỗi (ví dụ các đoạn mã xử lý dữ liệu do người dùng nhập vào).

Trong bài tập sau, bạn sẽ tạo một web form tương tự như form đã tạo trong Chương 15 “JavaScript và CSS”. Lần này, ngoài phản hồi trực quan trong trường văn bản, chúng ta cung cấp thêm phản hồi bằng chữ.

### Sử dụng *try/catch* với web form

- Sử dụng Microsoft Visial Studio, Eclipse hoặc trình soạn thảo khác sửa file number.htm trong thư mục mã nguồn mẫu của Chương 16, Tài nguyên đi kèm.
- Thay thế dòng chú thích TODO bằng đoạn mã bôi đậm dưới đây (Xem trong file number.txt của Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Try/Catch</title>
<script type="text/javascript" data-src="ehandler.js"></scr
<script type="text/javascript" data-src="number.js"></scr
</head>
<body>
<form name="formexample" id="formexample" action="#">
<div id="citydiv">Nhập một số trong khoảng từ 1 đến 100:<
<input id="num" name="num"> <
<div><input id="submit" type="submit"></div>
</form>
<script type="text/javascript">
var formEl = document.getElementsByTagName("form")[0];
EHandler.add(formEl,"submit", function(eventObj) { checkVa
</script>
</body>
</html>
```

- Tạo file JavaScript có tên là number.js (File này có trong Tài nguyên đi kèm).
- Chuyển đổi cách xử lý lỗi ở Chương 15 sang cách sử dụng *try/catch*. Câu lệnh *try/catch* không thực sự cần thiết cho ví dụ Chương 15, đây chỉ là minh họa cho cách dùng *try/catch*. Điền đoạn mã sau vào file number.js (mặc dù có thể kết hợp nhiều đoạn mã vào trong một câu lệnh *if*, tác giả vẫn chia ra nhiều câu vì chúng ta sẽ mở rộng những đoạn mã này ở bài tập sau).

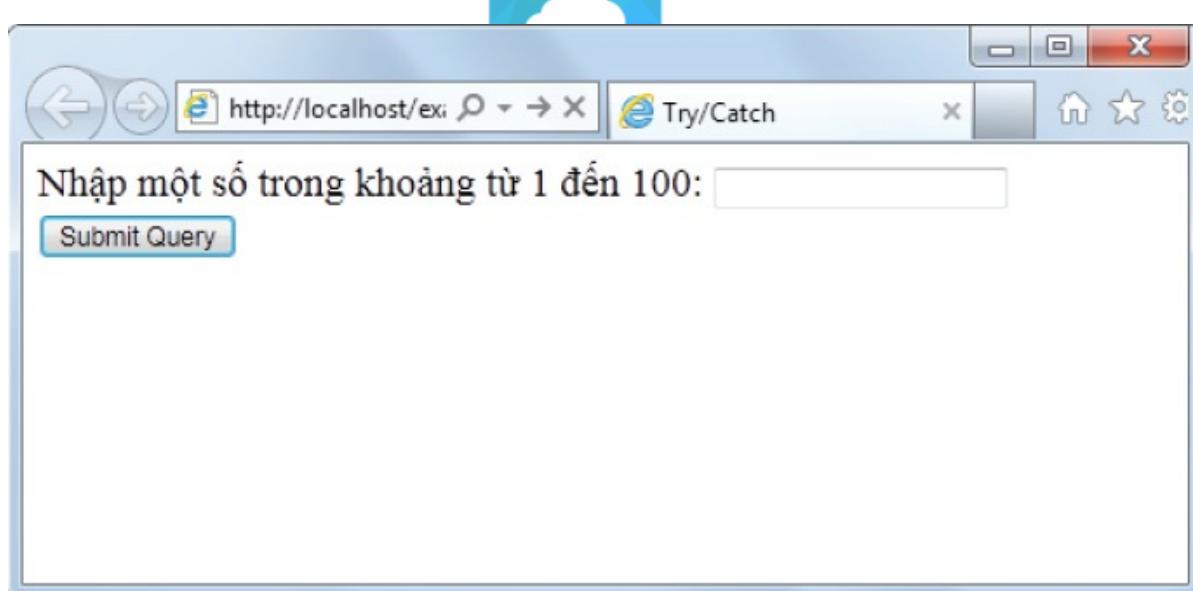
```
function checkValid(eventObj) {
 try {
 var numField = document.forms[0]["num"];
 if (isNaN(numField.value)) {
 throw numField;
```

```

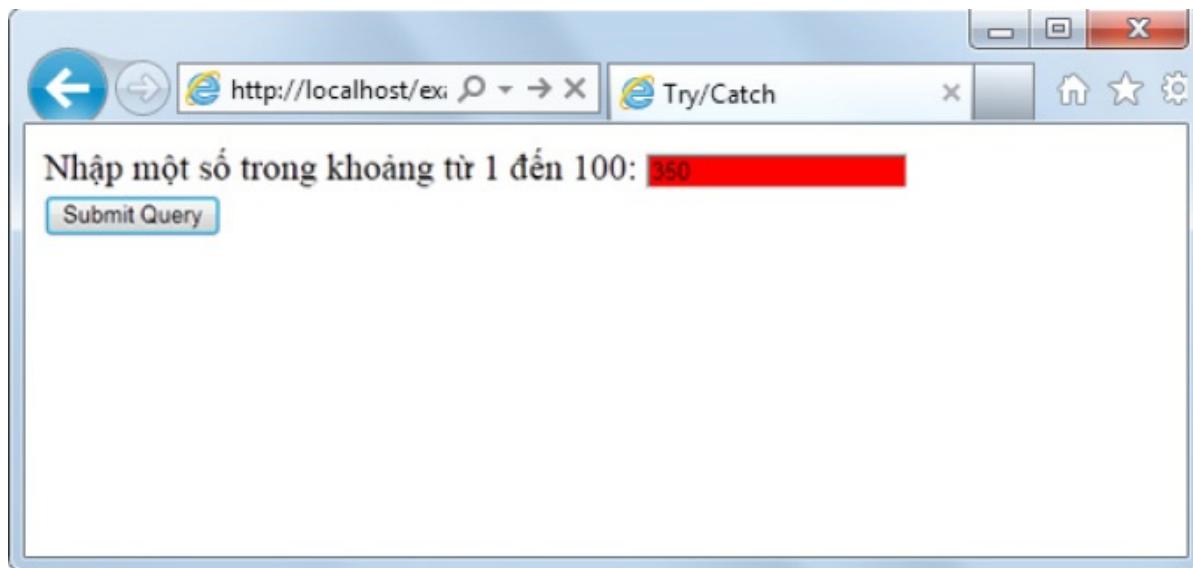
 }
 else if (numField.value > 100) {
 throw numField;
 }
 else if (numField.value < 1) {
 throw numField;
 }
 return true;
 }
 catch(errorObject) {
 errorObject.style.background = "#FF0000";
 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
 }
}

```

5. Dùng trình duyệt mở trang web. Cửa sổ như sau sẽ mở ra:



6. Kiểm tra chức năng của mệnh đề *try/catch*. Nhập một số lớn hơn 100 vào ô trống (ví dụ **350**) rồi nhấn Submit Query. Bạn sẽ thấy trang web như sau:



7. Nhập vào một cụm từ, nhấn Submit Query. Trường văn bản trong form vẫn có màu đỏ.
8. Nhập vào một số nhỏ hơn 1, ví dụ như **-2** rồi chọn Submit Query. Trường văn bản vẫn có màu đỏ.
9. Điền vào số **50** rồi chọn Submit Query. Lần này, form được gửi đi thành công, trả về form với trường văn bản để trống.
10. Sửa file number.js để thêm vào phản hồi bằng chữ. File hoàn chỉnh sẽ giống như sau:

```
function checkValid(eventObj) {
 try {
 var numField = document.forms[0]["num"];
 if (isNaN(numField.value)) {
 var err = new Array("Không phải là
 số");
 throw err;
 }
 else if (numField.value > 100) {
 var err = new Array("Số này lớn h
 ơn 100");
 throw err;
 }
 else if (numField.value < 1) {
 var err = new Array("Số này nhỏ h
 ơn 1");
 throw err;
 }
 }
}
```

```

 return true;
 }
 catch(errorObject) {
 var errorText = document.createTextNode(errorObject.message);
 var feedback = document.getElementById("feedback");
 var newspan = document.createElement("span");
 newspan.appendChild(errorText);
 newspan.style.color = "#FF0000";
 newspan.style.fontWeight = "bold";
 newspan.setAttribute("id", "feedback");
 var parent = feedback.parentNode;
 var newChild = parent.replaceChild(newspan, errorObject[1]);
 newspan.style.background = "#FF0000";
 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
 }
}

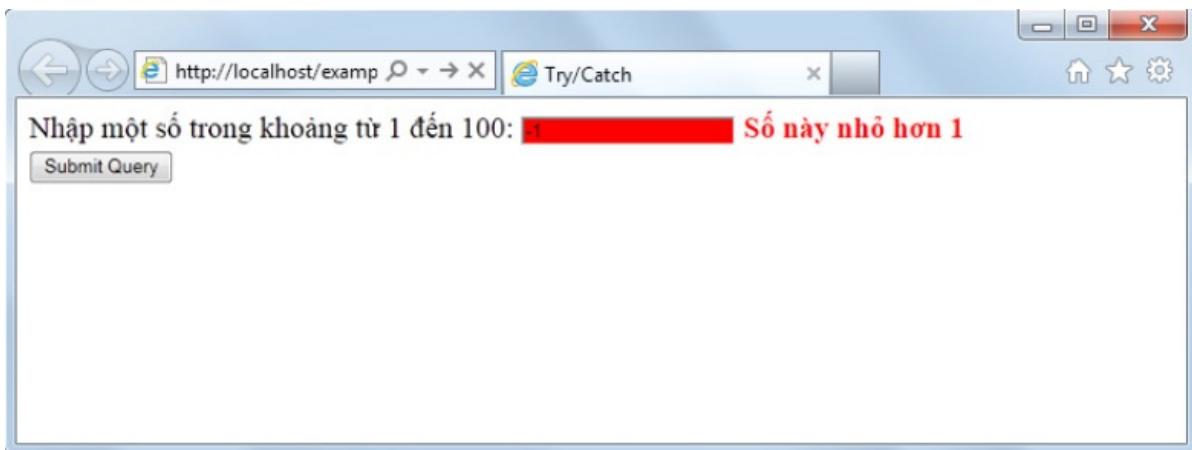
```



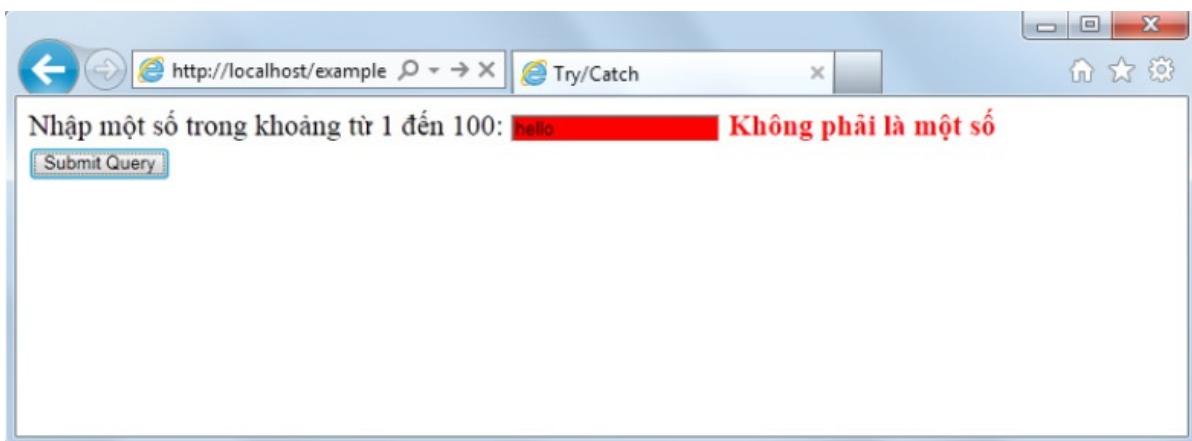
- [1]. Mở lại trang web để thực thi phiên bản script mới. Bạn chưa thấy sự thay đổi nào so với phiên bản trước.
- [2]. Nhập **350** vào ô văn bản và nhấn Submit Query. Bây giờ bạn nhận được trang như sau với phản hồi trả về cạnh ô văn bản:

Nhập một số trong khoảng từ 1 đến 100: 350 **Số này lớn hơn 100**

- [3]. Nhập **-1** vào ô văn bản và nhấn Submit Query. Bạn sẽ thấy trang sau:



4. Nhập vào một cụm từ và nhấn Submit Query. Bạn sẽ thấy trong hiển thị như sau:



5. Nhập vào một số từ 1 đến 100 (ví dụ 50) rồi gửi form đi. Form được gửi thành công và không có lỗi xảy ra.

Ví dụ này sử dụng một số phương thức đã được giải thích ở các chương trước để tạo phần tử mới và thêm vào trang web nhằm mục đích phản hồi. Phần đầu của ví dụ được lấy từ web form trong Chương 15 để sử dụng *try/catch* và nội dung mới. Khác biệt là ở chỗ form trong Chương 15 yêu cầu người dùng điền vào tên một thành phố trong khi form này yêu cầu nhập một số thuộc phạm vi xác định; bằng cách này, chúng ta có thể biểu diễn nhiều điều kiện phản hồi.

Từng điều kiện ném ra một lỗi với đối tượng *numField* là đối tượng lỗi. Trong câu lệnh catch, màu nền của *errorObject* chuyển sang đỏ và hàm trả về giá trị *false* nhằm thông báo form không hợp lệ. Đoạn mã thêm vào dùng một mảng để kết hợp cả lỗi viết bằng chữ và đối tượng *form* (*numField*). Đối tượng

*array* được đưa vào câu lệnh *catch*. Chỉ số đầu tiên (0) là đoạn văn bản hiển thị lỗi, chỉ số thứ hai (1) là đối tượng *numField* được biểu diễn như sau (hãy chú ý đến phần mã in đậm):

```
try {
 var numField = document.forms[0]["num"];
 if (isNaN(numField.value)) {
 var err = new Array("Không phải là số", numField);
 throw err;
 }
 else if (numField.value > 100) {
 var err = new Array("Lớn hơn 100", numField);
 throw err;
 }
 else if (numField.value < 1) {
 var err = new Array("Nhỏ hơn 1", numField);
 throw err;
 }
 else {
 return true;
 }
}
```



Trong ví dụ này câu lệnh *catch* thực thi một số nhiệm vụ. Đầu tiên, nó lấy về phần tử *<span>* để cung cấp phản hồi cho người dùng:

```
var feedback = document.getElementById("feedback");
```

Tiếp đến, nó tạo một nút văn bản mới thông qua đoạn văn bản báo lỗi của *errorObject*:

```
var errorText = document.createTextNode(errorObject[0]);
```

Sau đó, câu lệnh tạo một phần tử *span* mới để đặt vào tài liệu sau đó. Phần tử *span* trong đoạn mã là *newspan* được gán thêm đoạn văn bản báo lỗi màu đỏ với phần chữ đậm. Phần tử *span* mới này được đặt ID là *feedback* giống như phần tử *span* đã có:

```
var newspan = document.createElement("span");
newspan.appendChild(errorText);
newspan.style.color = "#FF0000";
newspan.style.fontWeight = "bold";
```

```
newspan.setAttribute("id", "feedback");
```

Đoạn mã truy xuất nút cha của đối tượng *feedback*, do đó nó có thể sử dụng phương thức *replaceChild()* để thay thế phần tử *span* cũ bằng phần tử mới như sau:

```
var parent = feedback.parentNode;
var newChild = parent.replaceChild(newspan, feedback);
```

Tiếp theo, đoạn mã đổi màu nền của trường nhập dữ liệu form thành màu đỏ:

```
errorObject[1].style.background = "#FF0000";
```

Những dòng cuối của đoạn mã, như bạn đã thấy ở các chương trước, ngăn các hành động mặc định của form để làm trình duyệt đứng tại trang mà không gửi form đi:

```
if (eventObj.preventDefault) {
 eventObj.preventDefault();
} else {
 window.event.returnValue = false;
}
return false;
```



**Mách nhỏ** Sử dụng *try/catch* như ví dụ trên giúp trừu tượng hóa việc xử lý ngoại lệ trong JavaScript. Tuy nhiên, sử dụng *try/catch* không thể ngăn chặn hay hỗ trợ việc kiểm tra những lỗi cú pháp trong mã.

## Xử lý nhiều ngoại lệ

Một số trình duyệt, trong đó có Firefox, cho phép người dùng có thể dễ dàng xử lý nhiều ngoại lệ. Ví dụ, xem đoạn mã dưới đây:

```
if (isNaN(numField.value)) {
 throw "Không phải số";
}
else if (numField.value > 100) {
 throw "Lớn hơn 100";
}
else if (numField.value < 1) {
```

```

 throw "Nhỏ hơn 1";
 }
Khối mã `catch` sẽ có dạng như sau:
catch(errorObject if errorObject == "Không phải số") {
 // Xử lý trường hợp không phải số
}
catch(errorObject if errorObject == "Lớn hơn 100") {
 // xử lý trường hợp > 100
}
catch(errorObject if errorObject == "Nhỏ hơn 1") {
 // xử lý trường hợp <1
}
catch(errorObject) {
 // xử lý các trường hợp khác
}

```

Trong đoạn mã này, mỗi ngoại lệ được bắt bởi những đoạn mã xử lý cho riêng từng trường hợp. Nếu không có ngoại lệ nào xảy ra, đoạn mã xử lý ngoại lệ chung tại phần cuối của khối lệnh *catch* sẽ được thực thi. Tuy nhiên, đáng tiếc là Windows Internet Explorer không hỗ trợ nền tính năng này, do đó bạn nên hạn chế sử dụng.



## Khối *finally*

Trong JavaScript, có một câu lệnh bổ trợ không bắt buộc, tên là *finally*, thường sử dụng cùng với *try/catch*. Câu lệnh *finally* chứa các đoạn mã sẽ được thực thi bất kể đoạn mã trong câu lệnh *try* có thực thi thành công hay không hoặc câu lệnh *catch* có chạy hay không. Thông thường, bạn nên sử dụng khối lệnh *finally* để đảm bảo một đoạn mã (ví dụ, đoạn mã có nhiệm vụ dọn dẹp) luôn được gọi.

Ví dụ 16-2 (có trong file listing16-2.txt của phần Tài nguyên đi kèm) đưa ra hàm *checkValid()* như bạn đã thấy trong những bài tập trước của chương, nhưng có bổ sung câu lệnh *finally*:

**VÍ DỤ 16-2** Thêm câu lệnh *finally* vào hàm *checkValid()*.

```
function checkValid(eventObj) {
```

```

try {
 var numField = document.forms[0]["num"];
 if (isNaN(numField.value)) {
 var err = new Array("Không phải là
 số");
 throw err;
 }
 else if (numField.value > 100) {
 var err = new Array("Số này lớn hơn
 100");
 throw err;
 }
 else if (numField.value < 1) {
 var err = new Array("Số này nhỏ hơn
 1");
 throw err;
 }
 return true;
}
catch(errorObject) {
 var errorText = document.createTextNode(er
 var feedback = document.getElementById("fe
 var newspan = document.createElement("span
 newspan.appendChild(errorText);
 newspan.style.color = "#FF0000";
 newspan.style.fontWeight = "bold";
 newspan.setAttribute("id", "feedback");
 var parent = feedback.parentNode;
 var newChild = parent.replaceChild(newspan
 errorObject[1].style.background = "#FF0000
 if (eventObj.preventDefault) {
 eventObj.preventDefault();
 } else {
 window.event.returnValue = false;
 }
 return false;
}
finally {
 alert("Sẽ luôn được gọi dù thao tác thành
 công");
}

```

## Sử dụng sự kiện *onerror*

Bạn có thể thấy sự kiện *onerror* được sử dụng trong chương trình xử lý các sự kiện lỗi, nhưng lập trình viên thường ít khi sử dụng phương pháp này vì họ có thể xử lý lỗi theo những cách thức tiện lợi hơn. Sự kiện *onerror* có thể gắn với đối tượng *window* và *image*.

## Gắn *onerror* vào đối tượng *window*

Để sử dụng sự kiện *onerror*, bạn cần gán chúng vào một hàm sẽ được gọi bất cứ khi nào lỗi JavaScript xảy ra. Sự kiện *onerror* khá hữu ích trong quá trình phát triển, dù vậy, giờ đây với những công cụ như Firebug, lập trình viên dần bớt lệ thuộc vào nó hơn trước.

Bạn có thể gán sự kiện *onerror* vào đối tượng *window* như sau:

```
window.onerror = myErrorHandler;
```

Biến *myErrorHandler* chỉ đến một hàm do người dùng định nghĩa để xử lý các lỗi. Trình thông dịch JavaScript tự động gửi đi ba đối số đến hàm xử lý lỗi:

- Mô tả lỗi.
- Địa chỉ URL mà lỗi xảy ra.
- Dòng mã ở đó lỗi xảy ra.



Khi hàm xử lý lỗi trả về giá trị *true*, JavaScript sẽ không xử lý lỗi đó; JavaScript sẽ hiểu rằng lỗi này đã được các hàm xử lý lỗi xử lý.

Tiếp theo, hãy xem Ví dụ 16-3 (Xem file listing16-3.txt, phần Tài nguyên đi kèm). Đây là ví dụ về JavaScript và hàm xử lý lỗi do người dùng tự định nghĩa.

**VÍ DỤ 16-3** Ví dụ về *onerror* của đối tượng *window*.

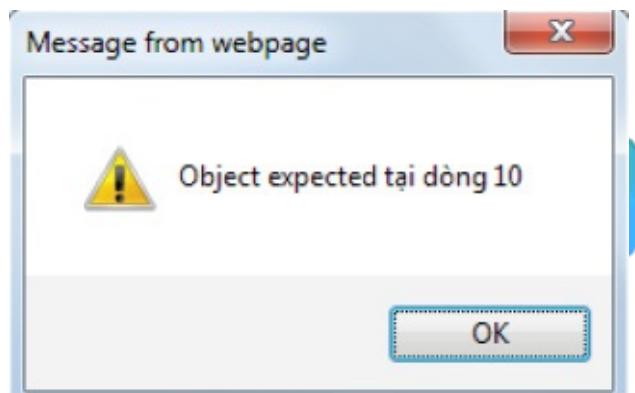
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>onerror</title>
</head>
<body>
<div id="mydiv">Hi</div>
```

```

<script type="text/javascript">
function init() {
 doSomething();
}
function errorHandler() {
 alert(arguments[0] + " tại dòng " + arguments[2]);
 return true;
}
window.onload = init;
window.onerror = errorHandler;
</script>
</body>
</html>

```

Khi bạn chạy các đoạn mã trong Ví dụ 16-3 trên trình duyệt, bạn sẽ thấy hộp thoại *alert()* như Hình 16-1.



HÌNH 16-1 Xử lý lỗi với sự kiện `onerror` của đối tượng `window`

Ví dụ 16-3 có chứa một hàm chủ định không được định nghĩa bên trong hàm *init()* khi trang web được tải. Trình thông dịch JavaScript ném ra lỗi khi nó tìm thấy một hàm không được định nghĩa và vì hàm *errorHandler* do người dùng định nghĩa được gán vào sự kiện *onerror*, nên hàm này được gọi. Hàm *errorHandler* hiển thị hộp thoại thông báo *alert()* và trả về giá trị *true* để ngăn xử lý lỗi. Hộp thoại *alert()* hiển thị chỉ số thứ nhất và thứ ba của mảng đối số (*arguments[0]* và *arguments[2]*). Mảng *arguments* chứa ba đối số được truyền vào hàm xử lý như đã được mô tả ở trên: Thông báo lỗi, địa chỉ và dòng mã xảy ra lỗi.

#### Tránh xử lý sự kiện theo cách gây bất tiện cho người truy cập

Ví dụ 16-3 xử lý lỗi theo cách hơi bất tiện: thông qua hộp thoại *alert()*.

Việc xử lý lỗi nên chạy ngầm bất cứ khi nào có thể thay vì đưa ra các hộp thoại thông báo. Nếu trang web có nhiều lỗi, việc sử dụng hộp thoại rất phiền phức cho người dùng vì họ phải nhấn từng thông báo để tắt chúng đi. Khi bạn đoán một lỗi có thể xảy ra, hãy cố gắng xử lý để đoạn mã vẫn hoạt động ở mức chấp nhận được – ví dụ, dùng một hàm thay thế hay hiển thị một thông báo lỗi thân thiện hơn với người dùng.

## BỎ QUA LỖI

Bạn có thể phớt lờ lỗi thay vì viết thêm mã để xử lý. Để làm được điều này, đơn giản chỉ cần trả về giá trị *true* trong hàm xử lý lỗi. Khi một hàm xử lý lỗi trả về giá trị *true*, trình duyệt hiểu rằng lỗi đã được xử lý. Do đó, có thể hiểu rằng bạn đã ngầm yêu cầu trình thông dịch bỏ qua lỗi.

Xét đoạn mã trong Ví dụ 16-4. Ví dụ này khá giống Ví dụ 16-3, tuy nhiên hàm *errorHandler* trả về giá trị *true* (như phần bôi đậm). Khi hàm không được định nghĩa *doSomething()* gây ra lỗi, lỗi này được bỏ qua.

Bạn có thể tìm thấy đoạn mã cho Ví dụ 16-4 trong Tài nguyên đi kèm, file listing16-4.htm.

### VÍ DỤ 16-4 Đoạn mã bỏ qua lỗi.

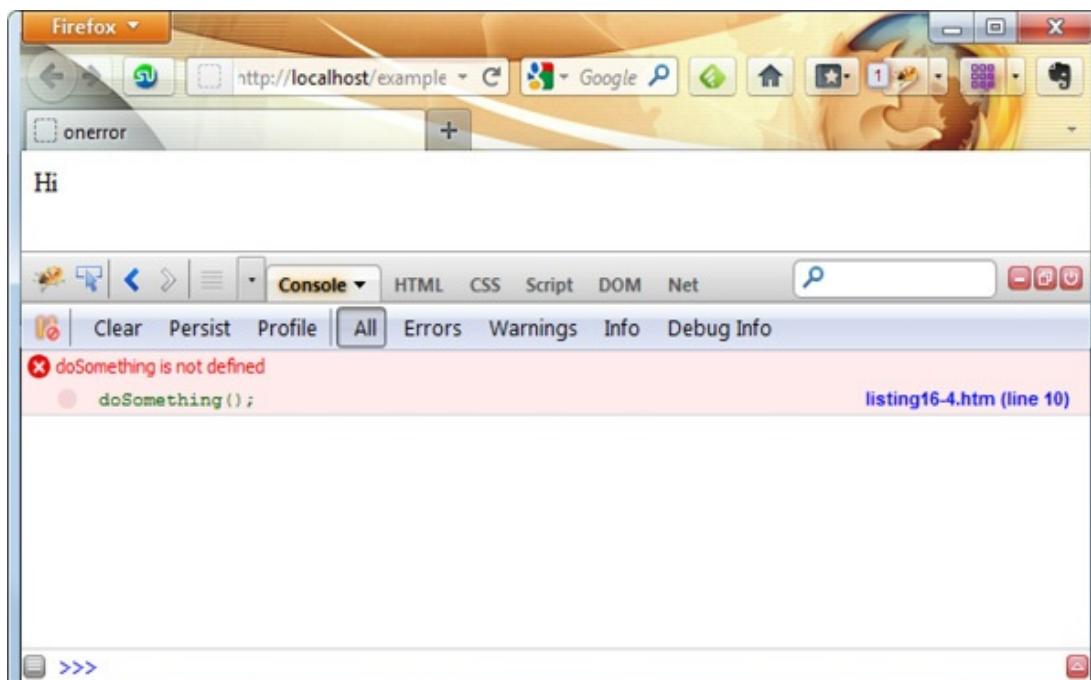
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>onerror</title>
</head>
<body>
<div id="mydiv">Hi</div>
<script type="text/javascript">
function init() {
 doSomething();
}
function errorHandler() {
 return true;
}
```

```
window.onload = init;
window.onerror = errorHandler;
</script>
</body>
</html>
```

Bạn sẽ hiểu cách bỏ qua lỗi khi chạy ví dụ trên Firefox có cài tiện tích Firebug. Mở file này trên trình duyệt, bạn sẽ không thấy lỗi nào xảy ra. Sau đó, chuyển câu lệnh *return true* trong hàm *errorHandler* thành chú thích như sau:

```
function errorHandler() {
 // return true;
}
```

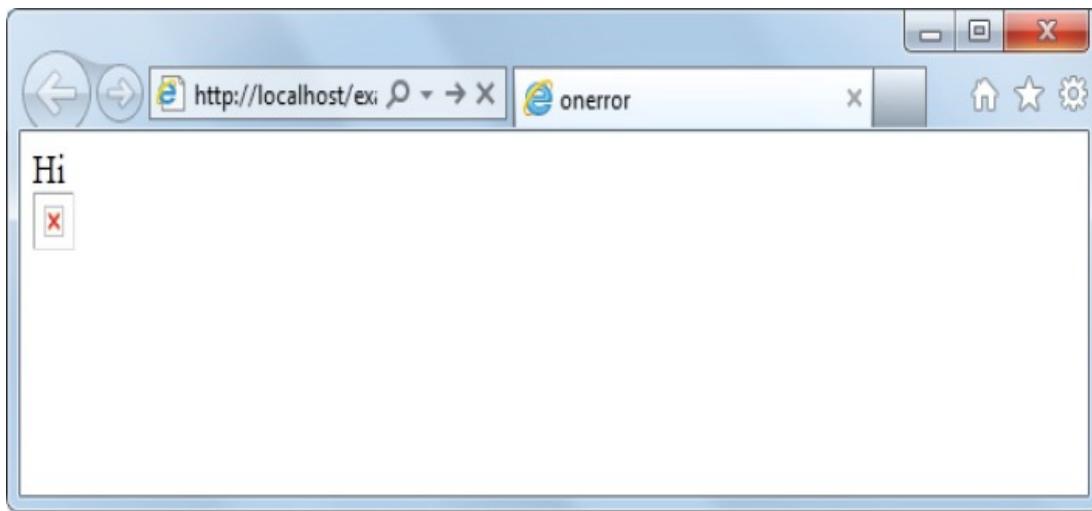
Khi bạn mở lại ví dụ, bạn sẽ thấy có một lỗi trong cửa sổ lỗi (*error console*) của Firebug như Hình 16-2.



HÌNH 16-2 Việc chuyển câu lệnh trả về giá trị của hàm xử lý thành chú thích sẽ gây ra lỗi như trong cửa sổ của Firebug.

## Gắn onerror vào đối tượng image

Bạn cũng có thể gắn sự kiện *onerror* vào các đối tượng *image*. Khi được đặt trong thẻ *<img>*, bạn có thể sử dụng những hàm xử lý sự kiện để xử lý những hình ảnh không tìm thấy. Ví dụ, Hình 16-3 là một trang web bị lỗi thiếu ảnh.



Hình 16-3 Lỗi thiếu file ảnh có thể tránh được bằng cách dùng JavaScript

Bạn có thể tìm thấy đoạn mã cho Ví dụ 16-5 trong Tài nguyên đi kèm, file listing16-5.htm.

### VÍ DỤ 16-5 Một trang web bị mất hình.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>onerror</title>
</head>
<body>
<div id="mydiv">Hi</div>

</body>
</html>
```



Bây giờ chúng ta sẽ xem xét đoạn mã với hàm xử lý *onerror* được chèn trong tài liệu. Hàm *onerror* chuyển hướng phần tử *<img>* đến một ảnh có tồn tại. Trong trường hợp này, nội dung của ảnh không quan trọng, điều cốt yếu là sử dụng hàm *onerror* ở đây sẽ giúp chúng ta tránh gặp biểu tượng “Image Not Found” (Không tìm thấy ảnh) trên trang web của mình. Bạn có thể tìm thấy đoạn mã cho Ví dụ 16-6 trong Tài nguyên đi kèm, file listing16-6.htm.

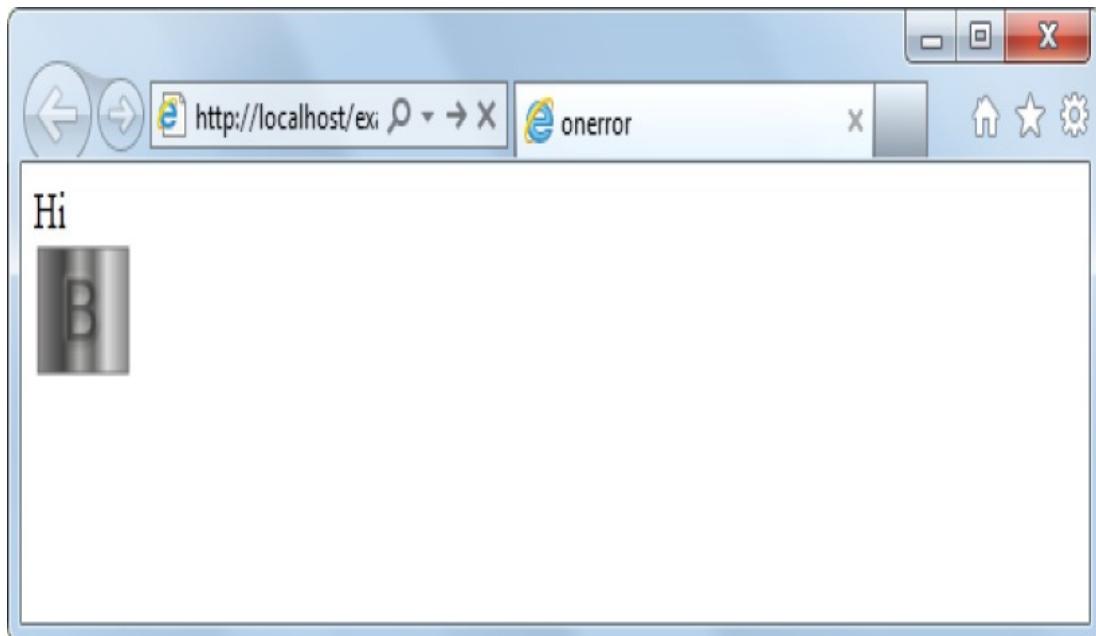
### VÍ DỤ 16-6 Thêm hàm xử lý *onerror()* cho hình ảnh.

```
<!DOCTYPE HTML PUBLIC ****-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
<head>
<title>onerror</title>
</head>
<body>
<div id="mydiv">Hi</div>

</body>
</html>
```

Khi bạn mở trang web trên trình duyệt, trang này không tìm thấy hình notfound.png, tuy nhiên trình duyệt lấy về và hiển thị file hình ảnh có tên logo.png như Hình 16-4.



HÌNH 16-4 File ảnh bị thiếu đã được thay thế nhờ vào hàm xử lý sự kiện `onerror`.

## Bài tập

1. Sử dụng hàm xử lý *onerror* gắn với đối tượng *window* để xử lý lỗi xảy ra khi có một hàm không được định nghĩa. Chú ý bạn có thể dùng đoạn mã trong Ví dụ 16-3, nhưng hàm xử lý nên hiển thị lỗi thân thiện với người dùng hơn là sử dụng hộp thoại *alert()*.
2. Tạo một web form và sử dụng *try/catch* để bắt trường hợp tên thành phố

điền vào không phải là “*Stockholm*”. Cung cấp phản hồi trực quan nếu tên thành phố không đúng.

3. Tạo một web form và sử dụng *try/catch/finally* để bắt lỗi trường hợp số điền vào lớn hơn 100. Luôn đưa ra lời cảm ơn người dùng mỗi khi họ nhập vào form (bất kể giá trị đúng hay sai).



# Phần 4

# AJAX và tích hợp phía server

[Chương 17: JavaScript và XML](#)

[Chương 18: Các ứng dụng JavaScript](#)

[Chương 19: Sơ lược về AJAX](#)

[Chương 20: Tìm hiểu thêm về AJAX](#)

## Chương 17

## JavaScript và XML



Sau khi đọc xong chương này, bạn có thể:

- Tìm hiểu các hàm dùng để mở tài liệu XML bằng JavaScript.
- Hiển thị dữ liệu XML dưới dạng bảng HTML.
- Xem tài liệu XML xuất ra từ Microsoft Office Excel 2007 bằng JavaScript.

## Sử dụng XML với JavaScript

XML là một ngôn ngữ bao gồm hầu hết các thẻ được định nghĩa bởi người dùng. Đặc điểm này khiến XML trở thành ngôn ngữ có tính tùy biến cao và thường được sử dụng để trao đổi dữ liệu. Một điểm quan trọng mà các lập trình viên JavaScript nên lưu ý là XML chính là chữ X trong cụm từ viết tắt AJAX (Asynchronous JavaScript and XML). AJAX hiện nay đã trở thành một công cụ được sử dụng phổ biến trong việc tạo các ứng dụng web có tính tương tác cao với người dùng. Bạn sẽ hiểu thêm về AJAX trong hai chương tiếp theo: Chương

19 “Sơ lược về AJAX” và Chương 20 “Tìm hiểu thêm về AJAX”.

XML là một chuẩn mở được định nghĩa bởi W3C và hiện đang ở phiên bản thứ tư. Phần này sẽ giới thiệu sơ lược về XML trong mối quan hệ với JavaScript. Bạn có thể tìm thêm thông tin về XML trên trang chủ của W3C (<http://www.w3.org/XML/Core>) hoặc trên trang web của Microsoft (<http://msdn.microsoft.com/xml/>).

## Một ví dụ về tài liệu XML

Tài liệu XML chứa các phần tử được đặt trong một cấu trúc tài liệu. Các phần tử này có quy tắc cú pháp riêng, bao gồm quy tắc một phần tử cần phải có thẻ bắt đầu (hay thẻ mở) và thẻ kết thúc (hay thẻ đóng). Đối với những người lập trình web, đoạn văn bản đặt giữa hai thẻ có thể quen thuộc. Sau đây là một tài liệu XML ví dụ (bạn có thể tìm thấy tài liệu này trong file books.xml trong thư mục mã nguồn mẫu Chương 17, trong Tài nguyên đi kèm):

```
<books>
<book>
 <title>MySQL Bible</title>
 <author>Steve Suehring</author>
 <isbn>9780764549328</isbn>
 <publisher>Wiley Publishing Inc.</publisher>
</book>
<book>
 <title>JavaScript - Hướng dẫn học qua ví dụ</title>
 <author>Steve Suehring</author>
 <isbn>9780735624498</isbn>
 <publisher>Microsoft Press</publisher>
</book>
</books>
```

Cấu trúc của một tài liệu XML cần thỏa mãn những tiêu chí nhất định để được công nhận là tài liệu đúng cú pháp (well-formed document). Như bạn đã thấy trong ví dụ trên, mỗi phần tử luôn có thẻ bắt đầu và thẻ kết thúc. Các phần tử cũng có thể lồng trong nhau. Có rất nhiều quy tắc giống các quy tắc của HTML.

Tài liệu XML cũng có thể chứa các thuộc tính, ví dụ như sau:

```
<?xml version="1.0"?>
<book title="JavaScript - Hướng dẫn học qua ví dụ" author="S...
```

# Tải một tài liệu XML bằng JavaScript

Bạn có thể tải và thao tác với các tài liệu XML bằng JavaScript. Phần này sẽ hướng dẫn bạn cách làm việc đó.

## Tải một tài liệu

Có hai cách để tải một tài liệu XML tùy thuộc vào trình duyệt mà bạn muốn hỗ trợ. Đối với các trình duyệt mới, như Chrome, Firefox và các phiên bản IE mới, bạn có thể sử dụng đối tượng `XMLHttpRequest()`, song với các phiên bản IE cũ, bạn phải dùng đối tượng `ActiveXObject`. Đoạn mã tiếp theo tải file `books.xml` trên tất cả các loại trình duyệt:

```
if (window.XMLHttpRequest) {
 var httpObj = new XMLHttpRequest();
} else {
 var httpObj = new ActiveXObject("Microsoft.XMLHTTP")
}
httpObj.open("GET", "books.xml", false);
httpObj.send();
var xmlDoc = httpObj.responseXML;
```

## Hiển thị tài liệu XML

Thông thường, chúng ta sử dụng định dạng bảng hoặc bảng tính để hiển thị dữ liệu XML. Hình 17-1 hiển thị file `books.xml` trong Excel 2007.

|    | A                                    | B              | C             | D                     | E | F | G |
|----|--------------------------------------|----------------|---------------|-----------------------|---|---|---|
| 1  | title                                | author         | isl           | publisher             |   |   |   |
| 2  | MySQL Bible                          | Steve Suehring | 9780764549328 | Wiley Publishing Inc. |   |   |   |
| 3  | JavaScript - Hướng dẫn học qua ví dụ | Steve Suehring | 9780735624498 | Microsoft Press       |   |   |   |
| 4  |                                      |                |               |                       |   |   |   |
| 5  |                                      |                |               |                       |   |   |   |
| 6  |                                      |                |               |                       |   |   |   |
| 7  |                                      |                |               |                       |   |   |   |
| 8  |                                      |                |               |                       |   |   |   |
| 9  |                                      |                |               |                       |   |   |   |
| 10 |                                      |                |               |                       |   |   |   |
| 11 |                                      |                |               |                       |   |   |   |
| 12 |                                      |                |               |                       |   |   |   |
| 13 |                                      |                |               |                       |   |   |   |
| 14 |                                      |                |               |                       |   |   |   |
| 15 |                                      |                |               |                       |   |   |   |
| 16 |                                      |                |               |                       |   |   |   |

HÌNH 17-1 File XML được hiển thị trên bảng tính.

Dữ liệu trong Hình 17-1 có thể hiển thị trên trình duyệt thông qua một phần tử bảng HTML. Hiển thị dữ liệu XML sử dụng JavaScript đòi hỏi một số kiến thức về Mô hình đối tượng tài liệu (DOM) kết hợp với các hàm và phương thức để tải về tài liệu mà bạn đã học.

Ví dụ tiếp theo tạo ra hàm *displayData()* hiển thị thông tin dưới dạng bảng.

### Thuộc tính *readyState*

Các phiên bản trước của cuốn sách này sử dụng thuộc tính *readyState* để xác định khi nào tài liệu XML được tải. Thuộc tính *readyState* là một số nguyên có năm giá trị tương ứng với trạng thái xử lý yêu cầu tải dữ liệu hiện tại. Bảng 17-1 liệt kê các giá trị này với chú giải tương ứng.

**BẢNG 17-1** Thuộc tính *readyState*\*\*

| Giá trị | Chú giải                                               |
|---------|--------------------------------------------------------|
| 0       | Chưa khởi tạo. Yêu cầu đã được mở nhưng chưa được gọi. |
| 1       | Mở. Yêu cầu được khởi tạo, nhưng chưa được gửi.        |
| 2       | Gửi. Yêu cầu đã được gửi.                              |
| 3       | Nhận. Đang nhận về phản hồi.                           |
| 4       | Tải. Đã nhận xong phản hồi.                            |

Bạn sẽ học thêm về thuộc tính *readyState* và sự kiện *onreadystatechange* trong Chương 18 “Các ứng dụng JavaScript.”

Việc hiển thị các nút và nút con bên trong một tài liệu XML đòi hỏi phải duyệt qua tất cả các cấp độ trong tài liệu và xây dựng tài liệu đầu ra. Hàm tiếp theo sẽ thực hiện công việc đó bằng cách duyệt qua cấu trúc của tài liệu XML và hiển thị dữ liệu trong đó dưới dạng bảng HTML. Đoạn mã này tiếp nối ví dụ tạo biến *xmlDocument* và nạp tài liệu XML (*books.xml*) vào biến *xmlDocument* ở trên.

```
function displayData(xmlDocument) {
 var xmlEl = xmlDocument.getElementsByTagName("book");
 var table = document.createElement("table");
 table.border = "1";
 var tbody = document.createElement("tbody");
 // Thêm phần thân cho bảng
 table.appendChild(tbody);
 // Tạo các dòng mới
 for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các cột cho dòng/td
 for (j = 0; j < xmlEl[i].childNodes.length;
 // Bỏ qua nếu kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeType
 continue;
 }
 // Chèn dữ liệu từ tài liệu XML.
 var td = document.createElement("td");
 var xmlData =
 document.createTextNode(xmlEl[i].chi-
 td.appendChild(xmlData);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}
```

```

 }
 document.getElementById("xmlData").appendChild(table)
 }

```

Để đưa đoạn mã trên vào trang web, chúng ta cần gắn các hàm tải và hiển thị file XML vào một sự kiện. Ví dụ 17 -1 (có trong file listing17-1.html trong tài nguyên đi kèm) tạo ra một hàm mới là *getXML* và gắn hàm này với sự kiện *load* của đối tượng *window*. Đoạn mã gắn sự kiện được in đậm.

### VÍ DỤ 17-1 Hiển thị dữ liệu XML trong bảng HTML.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Books</title>
<script type="text/javascript" data-src="eHandler.js"></sc
</head>
<body id="mainBody">
<div id="xmlData"></div>
<script type="text/javascript">
function displayData(xmlDocument) {
 var xmlEl = xmlDocument.getElementsByTagName("book");
 var table = document.createElement("table");
 table.border = "1";
 var tbody = document.createElement("tbody");
 // Thêm phần thân cho bảng
 table.appendChild(tbody);
 // Tạo dòng cho bảng
 for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các cột cho dòng/td
 for (j = 0; j < xmlEl[i].childNodes.length
 // Bỏ qua nếu kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeTyp
 continue;
 }
 // Chèn dữ liệu từ tài liệu XML.
 var td = document.createElement("td");
 var xmlData =
 document.createTextNode(xmlEl[i].c
 td.appendChild(xmlData);
 }
}

```

```

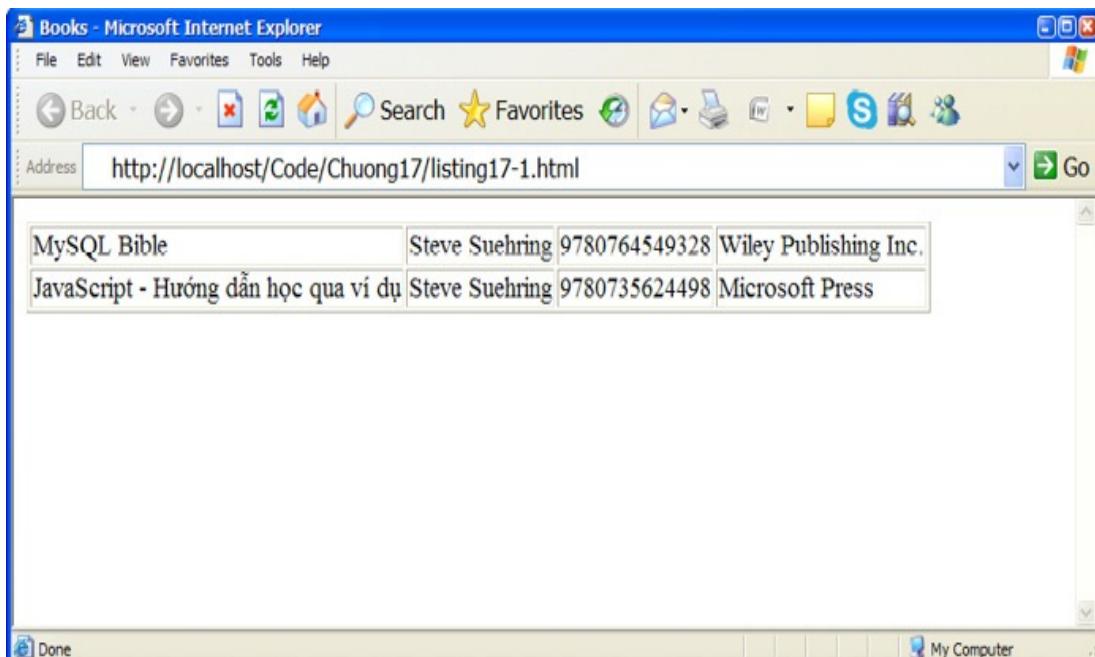
 row.appendChild(td);
 }
 tbody.appendChild(row);
}
document.getElementById("xmldata").appendChild(tab)
}

function getXML() {
 if (window.XMLHttpRequest) {
 var httpObj = new XMLHttpRequest();
 } else {
 var httpObj = new ActiveXObject("Microsoft
 }
 httpObj.open("GET", "books.xml", false);
 httpObj.send();
 var xmlDoc = httpObj.responseXML;
 displayData(xmlDoc);
}

var mainBody = document.getElementById("mainBody");
EHandler.addHandler(mainBody, "load", function() { getXML(); });
</script>
</body>
</html>

```

Khi xem trên trình duyệt, bảng dữ liệu **trống** giống bảng tính thông thường như trong Hình 17-2.



HINH 17-2 Hiển thị tài liệu books.xml trong bảng HTML.

Ví dụ 17-1 cho thấy một vòng lặp *for* được sử dụng để duyệt qua cấu trúc XML và lần lượt tạo ra các dòng cho bảng. Chú ý là vòng lặp này chỉ tìm các nút *Element* bên trong tài liệu XML bằng đoạn mã sau:

```
// Bỏ qua nếu kiểu khác 1
if (xmlEl[i].childNodes[j].nodeType != 1) {
 continue;
}
```

Kiểu nút (*nodeType*) bằng 1 chứng tỏ đó là nút *Element*. Nếu kiểu của nút hiện tại không phải là *Element*, đoạn mã sẽ chuyển đến phần tiếp theo trong tài liệu. Bạn có thể để ý thấy bảng dữ liệu trong Hình 17-2 không hề có tiêu đề cột. Để thêm tiêu đề, bạn chỉ cần thêm một đoạn mã. Phần tiếp theo sẽ hướng dẫn bạn cách thực hiện việc này.

### Thêm tiêu đề cột từ tài liệu XML

1. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo khác sửa file books.htm trong thư mục mã nguồn mẫu Chương 17 trong Tài nguyên đi kèm. (Lúc này, khi xem trên trình duyệt, trang books.htm sẽ vẫn hiển thị giống Hình 17-2).
2. Trong file books.htm, thêm đoạn mã in đậm dưới đây vào hàm *displayData()* (có thể tìm thấy đoạn mã này trong file books.txt) thay cho dòng chú thích TODO:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Books</title>
<script type="text/javascript" data-src="ehandler.js"></script>
</head>
<body id="mainBody">
<div id="xmldata"></div>
<script type="text/javascript">
function displayData(xmlDocument) {
 var xmlEl = xmlDocument.getElementsByTagName("book");
 var table = document.createElement("table");
 table.border = "1";
 var tr = document.createElement("tr");
 var td1 = document.createElement("td");
 td1.innerHTML = "Title";
 var td2 = document.createElement("td");
 td2.innerHTML = "Author";
 var td3 = document.createElement("td");
 td3.innerHTML = "Price";
 tr.appendChild(td1);
 tr.appendChild(td2);
 tr.appendChild(td3);
 table.appendChild(tr);
 for (var i = 0; i < xmlEl.length; i++) {
 var tr = document.createElement("tr");
 var td1 = document.createElement("td");
 td1.innerHTML = xmlEl[i].getElementsByTagName("title")[0].innerHTML;
 var td2 = document.createElement("td");
 td2.innerHTML = xmlEl[i].getElementsByTagName("author")[0].innerHTML;
 var td3 = document.createElement("td");
 td3.innerHTML = xmlEl[i].getElementsByTagName("price")[0].innerHTML;
 tr.appendChild(td1);
 tr.appendChild(td2);
 tr.appendChild(td3);
 table.appendChild(tr);
 }
 xmldata.innerHTML = "";
 xmldata.appendChild(table);
}
```

```
var tbody = document.createElement("tbody");
// Thêm phần thân cho bảng
table.appendChild("****tbody****");
var row = document.createElement("tr");
for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType == 1)
 continue;
}
var tableHead = document.createElement("th");
var colName =
 document.createTextNode(xmlEl[0].childNodes[colHead].nodeValue);
tableHead.appendChild(colName);
row.appendChild(tableHead);
}
// Chèn dòng vào bảng
tbody.appendChild(row);
// Tạo dòng cho bảng
for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các cột cho dòng/td
 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 if (xmlEl[i].childNodes[j].nodeType == 1)
 continue;
 }
 // Chèn dữ liệu từ tài liệu XML.
 var td = document.createElement("td");
 var xmlData =
 document.createTextNode(xmlEl[i].childNodes[j].nodeValue);
 td.appendChild(xmlData);
 row.appendChild(td);
}
tbody.appendChild(row);
}
document.getElementById("xmldata").appendChild(tbody);
}

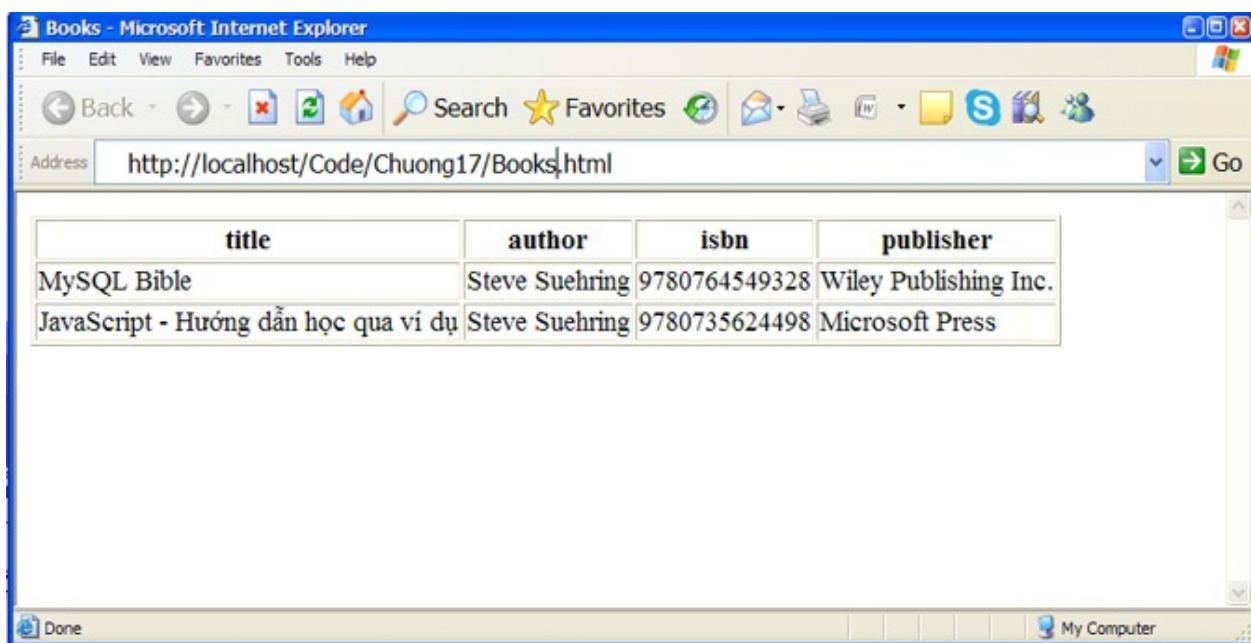
function getXML() {
 if (window.XMLHttpRequest) {
 var httpObj = new XMLHttpRequest();
 } else {
 var httpObj = new ActiveXObject("Microsoft.XMLHTTP");
 }
}
```

```

 httpObj.open("GET", "books.xml", false);
 httpObj.send();
 var xmlDoc = httpObj.responseXML;
 displayData(xmlDoc);
 }
 var mainBody = document.getElementById("mainBody");
 EHandler.add(mainBody, "load", function() { getXML(); });
</script>
</body>
</html>

```

3. Dùng trình duyệt mở trang web. Bạn sẽ nhận được kết quả như sau:



## Làm việc với dữ liệu XML xuất từ Excel 2007

Excel 2007 hỗ trợ nhiều tính năng để làm việc với dữ liệu XML. Bạn có thể nhập và xuất dữ liệu XML bằng Excel. Trên thực tế, Excel không thêm bất cứ thuộc tính độc quyền nào khi xuất dữ liệu ra tài liệu XML. Dưới đây là file books.xml được xuất ra từ Excel 2007 (file này có trong Tài nguyên đi kèm với tên là newbooks.xml):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <book>
 <title>MySQL Bible</title>
 <author>Steve Suehring</author>
 <isbn>9780764549328</isbn>
 <publisher>Wiley Publishing Inc.</publisher>
 </book>
 <book>
 <title>JavaScript - Hướng dẫn học qua ví dụ</title>
 <author>Steve Suehring</author>
 <isbn>9780735624498</isbn>
 <publisher>Microsoft Press</publisher>
 </book>
</books>
```

Nhờ mối liên hệ gần gũi giữa XML và Excel 2007, hàm *displayData()* ở trên có thể làm việc ngay với dữ liệu XML xuất ra từ Excel 2007 mà không phải thay đổi gì cả. Đối với các lập trình viên từng làm việc với các định dạng độc quyền trước đây, có thể nói đây là một điều ngạc nhiên đầy thú vị.



## Sơ lược về những nội dung tiếp theo

Mặc dù XML là chữ X trong cụm từ viết tắt AJAX, song AJAX có nhiều nội dung thú vị hơn là JavaScript và XML. AJAX có thể làm việc với các kiểu dữ liệu khác ngoài XML. Trong Chương 19, bạn sẽ làm việc với AJAX dựa trên những kiến thức nền tảng được giới thiệu ngắn gọn trong chương này. Chương 20 sẽ trình bày cách kết hợp JavaScript, AJAX và CSS để hiển thị dữ liệu truy xuất bằng JavaScript.

## Bài tập

1. Sử dụng đoạn mã ví dụ trong chương để hiển thị bảng dữ liệu khi người dùng nhấn vào một liên kết thay vì khi trang web được tải lên.

2. Sử dụng đoạn mã ví dụ trong chương để hiển thị bảng dữ liệu, kết hợp với DOM để đổi màu cho từng dòng trên bảng sao cho các dòng chẵn có màu xám. Gợi ý: #aaabba là dạng biểu diễn thập lục phân của màu xám.



# Chương 18

## Các ứng dụng JavaScript

Sau khi đọc xong chương này, bạn có thể:

- Hiểu các thành phần có trong ứng dụng JavaScript.

## Thành phần của ứng dụng JavaScript

Để xây dựng một ứng dụng trên nền web có giao diện phức tạp và đem lại cảm nhận như sử dụng ứng dụng desktop, bạn cần sử dụng JavaScript. Những ứng dụng này có tính năng và khả năng đáp ứng như một ứng dụng desktop, như thể ứng dụng đang được thực thi trên máy tính cục bộ thay vì chạy qua trình duyệt.

Chương này sẽ cung cấp cho bạn cái nhìn tổng quan về các thành phần cấu thành một ứng dụng dựa trên JavaScript. Mục tiêu của chương là giúp bạn hiểu các kiến trúc bên dưới và mức độ phức tạp khi xây dựng ứng dụng ở cấp doanh nghiệp.

### Ba yếu tố chính: Hiển thị, Hành vi, Dữ liệu

Có ba thành phần chính tồn tại trong một ứng dụng web:

- **Hiển thị** Giao diện của trang web.
- **Hành vi** Chức năng của giao diện ứng dụng, cụ thể là những gì sẽ xảy ra khi người dùng nhấn chuột vào một thành phần của trang web hay tương tác với nó.

- Dữ liệu** Thành phần phía server chứa dữ liệu và thực thi các thao tác, kết quả
- trả về được biểu diễn trên trang web.

Mã JavaScript chủ yếu xử lý hai thành phần đầu tiên – hiển thị và hành vi – để tác động đến giao diện hoặc phản hồi khi người dùng thực thi một hành động trên trang web. JavaScript cũng làm việc với những dữ liệu được server trả về, nhưng thường chỉ để thay đổi hình thức hiển thị theo cách nào đó. Ví dụ, một lời gọi đến dịch vụ web trả về nhiệt độ tại thời điểm hiện tại hoặc tình hình thời tiết có thể dùng JavaScript để thay đổi biểu tượng thời tiết (ví dụ: thời tiết có nắng). Nay giờ, chúng ta sẽ đi vào chi tiết của từng thành phần.

## Hiển thị: Bố cục trang web

Thành phần hiển thị của trang web bao gồm bố cục của trang và tất cả những gì liên quan đến giao diện của website hay trang web như màu sắc, hình ảnh, kiểu menu (bo tròn hay vuông...), vị trí các nút bấm và nội dung, màu chữ và cách sử dụng hình ảnh. Như bạn đã thấy trong các chương về CSS và việc kiểm tra tính hợp lệ của form JavaScript có thể tác động tới toàn bộ các phần trên. Những thành phần này là yếu tố cơ bản trong thiết kế web và được người dùng chú ý tới nhiều nhất. Vì vậy, bạn cần cân nhắc đến chúng khi xác định yêu cầu cho website của mình.



## Hành vi: Kiểm soát hành vi và thời điểm xảy ra hành vi

Một trong những yếu tố quan trọng nhất quyết định trải nghiệm người dùng cũng chính là điều thường bị bỏ qua nhất khi thiết kế ứng dụng web: hành vi của giao diện ứng dụng, thành phần quyết định điều gì xảy ra khi người dùng tương tác với một thành phần nào đó. Xét hai trường hợp sau:

- Khi người dùng nhấn nút Submit trên web form, nút Submit vẫn trong trạng thái kích hoạt hay đã bị vô hiệu?
- Khi người dùng chuyển focus đến một trường văn bản, trường này có đổi màu hay được làm nổi lên không?

Những hành vi dù nhỏ nhưng có thể cải thiện đáng kể trải nghiệm của người dùng nếu được thiết kế đúng cách. Tuy vậy, khi thiết kế website, người ta thường bỏ qua hay lược bớt để ưu tiên cho giao diện hiển thị hoặc thiết kế thô của trang web.

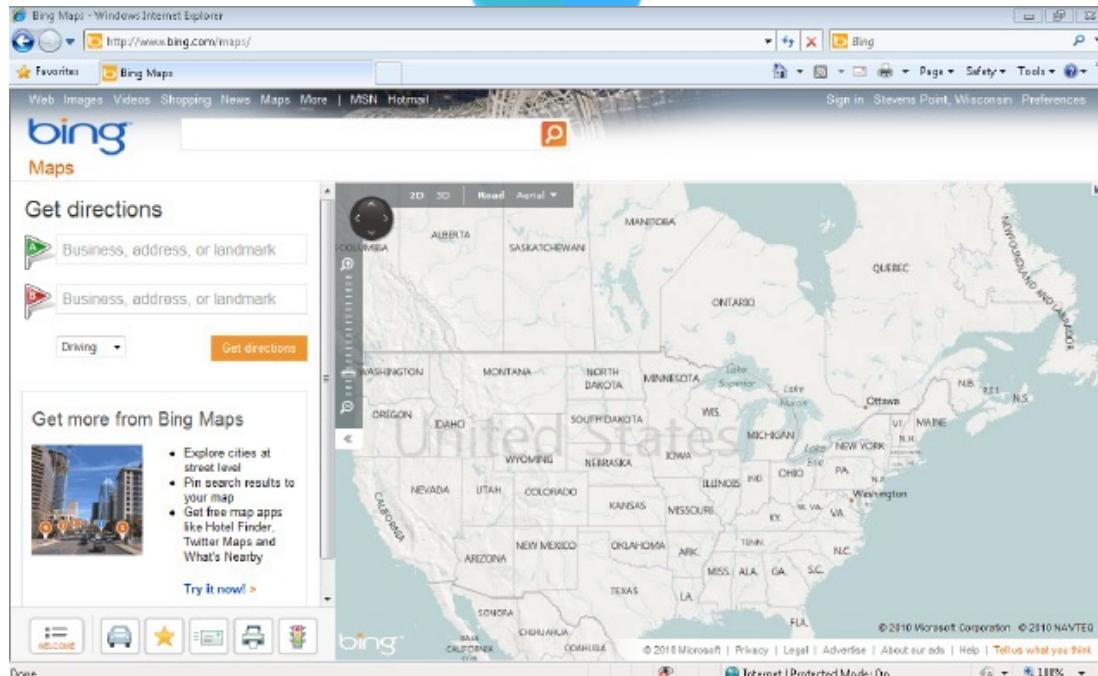
## Dữ liệu: Sử dụng, hiển thị và kiểm tra tính hợp lệ

JavaScript không tương tác trực tiếp với cơ sở dữ liệu hay server, ít nhất là nội dung này sẽ không được đề cập trong phạm vi cuốn sách này. Rõ ràng, JavaScript có thể làm được điều này nhờ kỹ thuật AJAX và thông qua dịch vụ web, tuy nhiên các quá trình này đòi hỏi đoạn mã phía server phải trả dữ liệu về cho đoạn mã JavaScript đã gọi.

Giống như thành phần hiển thị của trang web, thành phần dữ liệu phía server cũng cần được quan tâm đáng kể khi thiết kế ứng dụng web. Bao trọn từ thiết kế cơ sở dữ liệu đến lập trình logic nghiệp vụ, phần công việc hạ tầng này cần sự chú ý đặc biệt.

## JavaScript và giao diện web

Các lập trình viên sử dụng JavaScript để tạo các ứng dụng phía người dùng nhằm cung cấp những trải nghiệm chất lượng. Microsoft Bing Maps (tiền thân là Live Search Maps) là một ứng dụng web dựa nhiều vào JavaScript. Hình 18-1 là ví dụ về Microsoft Bing Maps, bạn có thể truy cập tại <http://www.bing.com/maps/>.

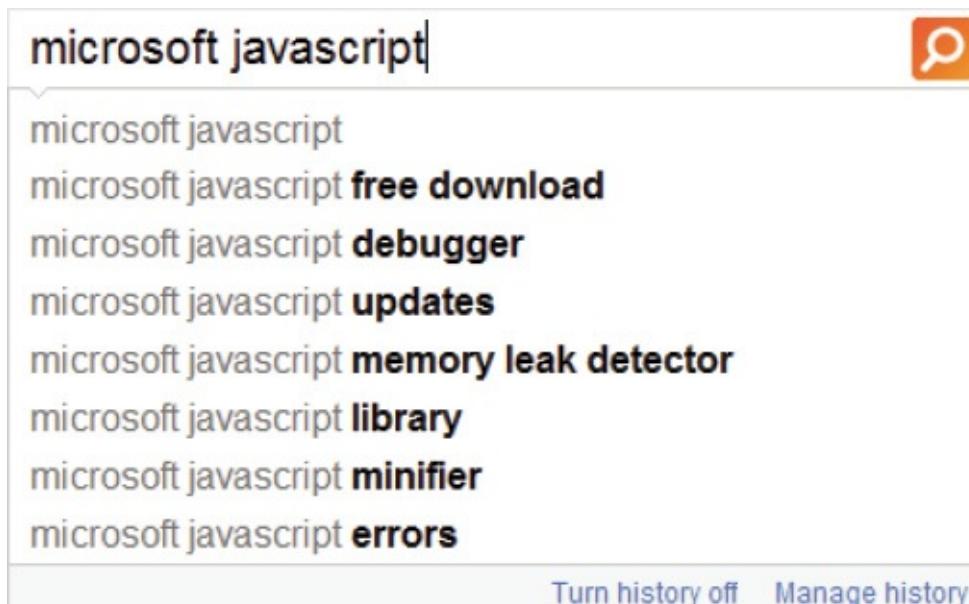


HÌNH 18-1 Giao diện Bing Maps sử dụng JavaScript để nâng cao tính tương tác.

Người dùng có thể kéo bản đồ ra các hướng y hệt như họ thao tác với một ứng

dụng desktop. Bản đồ được ghép từ nhiều ô hình với độ phân giải khác nhau. Khi người dùng thực hiện thao tác kéo chuột trên bản đồ, trình duyệt gửi yêu cầu HTTP đến web server Virtual Earth để yêu cầu trả về những ô hình mới, sau đó trình duyệt nhanh chóng hiển thị thêm những ô hình mới này.

Công cụ tìm kiếm Bing cũng sử dụng gợi ý tìm kiếm tương tự như các công cụ tìm kiếm khác, chẳng hạn Google. Khi bạn nhập từ khóa vào ô tìm kiếm của trang <http://www.bing.com>, trình duyệt ngay lập tức gửi yêu cầu HTTP đến server để tìm các từ khóa tương tự như Hình 18-2.



HÌNH 18-2 Khả năng gợi ý tìm kiếm sử dụng JavaScript để lấy về danh sách các từ khóa liên quan từ server.

Tất cả các thành phần của công cụ tìm kiếm Bing đều sử dụng JavaScript. Không thể kể hết các ứng dụng web sử dụng JavaScript để cải tiến trải nghiệm người dùng bằng cách điều chỉnh thành phần hành vi của trang web. Phần còn lại của cuốn sách sẽ cùng bạn xây dựng những ứng dụng kiểu này. Chương 19 “Sơ lược về AJAX” và Chương 20 “Tìm hiểu thêm về AJAX” sẽ hướng dẫn bạn cách tạo ra giao diện cung cấp chức năng tìm kiếm với gợi ý đơn giản bằng JavaScript. Hai chương này cũng giới thiệu về AJAX và đưa ra các ví dụ làm việc với dữ liệu để tạo ra ứng dụng. Chương 21 “Giới thiệu về các thư viện và Framework của JavaScript” và chương 22 “Giới thiệu về jQuery” sẽ giới thiệu các thư viện JavaScript, tập trung vào jQuery, cung cấp những kiến thức giúp bạn đơn giản hóa nhiều công việc khi tạo các ứng dụng chứa mã JavaScript phức tạp.

# Chương 19

## Sơ lược về AJAX

Sau khi đọc xong chương này, bạn có thể:

- Hiểu những điểm cơ bản về phương thức lập trình AJAX (Asynchronous JavaScript and XML).
- Hiểu rõ những điểm khác biệt giữa lời gọi AJAX đồng bộ và không đồng bộ.
- Sử dụng AJAX để truy xuất dữ liệu.
- Sử dụng AJAX với các phương thức HTTP (Hypertext Transfer Protocol) khác nhau để truy xuất phản hồi từ server.

### Giới thiệu về AJAX

AJAX mô tả phương thức lập trình kết hợp JavaScript và web server. Các lập trình viên sử dụng AJAX để tạo những ứng dụng web mang tính tương tác cao, ví dụ như Microsoft Virtual Earth.

Nếu không sử dụng AJAX, ứng dụng có thể bắt người truy cập đợi trong khi thu thập phản hồi từ server. Ứng dụng AJAX gửi các yêu cầu từ trình duyệt web tới web server ở chế độ chạy ngầm (chế độ không đồng bộ) trong khi người truy cập tương tác với ứng dụng. Điều này làm cho ứng dụng đáp ứng người dùng tốt hơn.

Trong ứng dụng AJAX, JavaScript xử lý các phản hồi và hiển thị dữ liệu tới người dùng. Khi kết hợp với CSS (Cascading Style Sheets) cùng với bố cục trang web hợp lý, ứng dụng AJAX cung cấp những tính năng sử dụng và sự linh động tuyệt vời mà chỉ ứng dụng web mới có thể có được.

Mặc dù có thể có một số ứng dụng AJAX phức tạp, nhưng quá trình gửi yêu cầu và xử lý phản hồi của những ứng dụng đó không quá phức tạp. Chương này sẽ

giúp bạn tìm hiểu cách gửi các yêu cầu và nhận phản hồi từ phía server bằng cách sử dụng một đối tượng cơ bản trong AJAX: *XMLHttpRequest*.

Một khái niệm trọng tâm trong AJAX là gọi tới các ứng dụng phía server để nhận về dữ liệu. Trong chương này, bạn sẽ được cung cấp kiến thức tổng quan về cách tạo ra các ứng dụng phía server với ASP.NET và PHP (PHP là từ viết tắt của PHP Hypertext Preprocessor - Bộ tiền xử lý siêu văn bản). Nếu cần thêm sự hỗ trợ để tạo mã chương trình phía server trong ứng dụng AJAX, bạn có thể tham khảo một số nguồn.

Nếu tạo ứng dụng phía server sử dụng công nghệ của Microsoft thì Microsoft Developer Network là một nguồn tham khảo tuyệt vời với rất nhiều tài liệu hướng dẫn và bài viết giới thiệu về AJAX (<http://msdn.microsoft.com/en-us/magazine/cc163363.aspx>). Microsoft Press cũng xuất bản một số cuốn sách xây dựng ứng dụng web rất hay. Một trong số đó là cuốn *Microsoft ASP.NET 3.5 Step By Step* (Microsoft Press, 2008). Với các cuốn sách khác, bạn có thể xem thêm thông tin tại <http://www.microsoft.com/mspress>.

Nếu bạn đang phát triển các ứng dụng phía server sử dụng công nghệ như LAMP (Linux, Apache, MySQL, Perl/PHP/Python), cách đơn giản nhất để nhanh chóng làm việc trên nền tảng này đó là tìm kiếm các bài hướng dẫn trên web. Cuốn sách *Learning Perl* (O'Reilly, 2005) là một tài liệu tham khảo rất tốt để tìm hiểu những kiến thức cơ bản về ngôn ngữ lập trình Perl.

**Chú ý** Nếu thích phong cách viết cuốn sách này, bạn có thể tham khảo cuốn *Beginning Perl Web Development* (Apress, 2005), tài liệu này tập trung vào việc sử dụng ngôn ngữ Perl để làm việc với các ứng dụng web.

Trang web chính thức của PHP (<http://www.php.net>) là nguồn tài liệu rất tốt để tìm hiểu với PHP, còn với ngôn ngữ Python, hãy xem chi tiết tại website <http://www.python.org>.

## Đối tượng XMLHttpRequest

Đối tượng *XMLHttpRequest* là trung tâm để xây dựng ứng dụng AJAX. Mặc dù, việc triển khai JavaScript có nhiều khác biệt nhưng tổ chức ECMAScript

cũng như tổ chức W3C (World Wide Web Consortium) đã tiến hành chuẩn hóa rất nhiều khía cạnh trong lập trình JavaScript, ngoại trừ đối tượng *XMLHttpRequest*, đối tượng này chưa bao giờ được chuẩn hóa. Dù vậy, kể từ khi phiên bản Windows Internet Explorer 7 ra đời, đối tượng *XMLHttpRequest* đã được sử dụng thống nhất trên tất cả các trình duyệt chính.

Microsoft hỗ trợ đối tượng *XMLHttpRequest* lần đầu tiên trong phiên bản Microsoft Internet Explorer 5.0. Nếu người truy cập sử dụng trình duyệt phiên bản cũ hơn, ứng dụng sử dụng đối tượng *XMLHttpRequest* sẽ không hoạt động được. Trong các phiên bản Internet Explorer trước phiên bản 7, đối tượng *XMLHttpRequest* được khởi tạo như một đối tượng *ActiveXObject*, trong khi các trình duyệt khác lại triển khai đối tượng *XMLHttpRequest* như một đối tượng JavaScript tích hợp sẵn trong trình duyệt. Điều này có nghĩa là nếu ứng dụng cần làm việc với các phiên bản Internet Explorer trước phiên bản 7, bạn phải khởi tạo đối tượng *XMLHttpRequest* theo cách khác, giống như tôi sẽ trình bày trong phần sau của chương này. Phần tiếp theo “Khởi tạo đối tượng *XMLHttpRequest*” sẽ trình bày cách kiểm tra sự tồn tại của đối tượng *XMLHttpRequest* và cách khởi tạo đối tượng này trong tất cả các phiên bản Internet Explorer.



## Khởi tạo đối tượng *XMLHttpRequest*

Đối với Internet Explorer 7 hoặc các phiên bản mới hơn và các trình duyệt phổ biến có hỗ trợ đối tượng *XMLHttpRequest*, việc khởi tạo đối tượng *XMLHttpRequest* được thực hiện tương tự nhau:

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

Còn với các phiên bản Internet Explorer trước phiên bản 7, bạn sẽ phải khởi tạo một đối tượng *ActiveXObject* thay thế. Tuy nhiên, cách thức khởi tạo đối tượng *ActiveXObject* cũng khác nhau tùy theo phiên bản của thư viện XMLHTTP được cài đặt trên máy client. Do đó, bạn cần viết thêm mã bổ trợ để khởi tạo đối tượng *XMLHttpRequest* trong các phiên bản Internet Explorer cũ.

Đoạn mã trong Ví dụ 19-1 trình bày hàm khởi tạo đối tượng *XMLHttpRequest* có thể sử dụng trên nhiều trình duyệt khác nhau.

**VÍ DỤ 19-1** Khởi tạo một đối tượng *XMLHttpRequest* trên nhiều trình duyệt khác nhau.

```

function readyAJAX() {
 try {
 return new XMLHttpRequest();
 } catch(e) {
 try {
 return new ActiveXObject("Msxml2.X
 } catch(e) {
 try {
 return new ActiveXObject("Microsoft
 } catch(e) {
 return "Bạn cần phải sử dụng
 }
 }
 }
}

```

Hàm được định nghĩa trong Ví dụ 19-1 sử dụng nhiều tầng khối lệnh *try/catch* để khởi tạo một đối tượng *XMLHttpRequest* bất kể người truy cập sử dụng trình duyệt nào. Nếu lời gọi ban đầu tới đối tượng *XMLHttpRequest* thất bại, điều đó có nghĩa là người truy cập đang sử dụng phiên bản trình duyệt Internet Explorer trước phiên bản 7. Trong trường hợp đó, lỗi sẽ được phát hiện và một trong các phương thức khởi tạo *XMLHttpRequest* dựa trên đối tượng *ActiveXObject* sẽ chạy. Nếu không có phương thức nào khởi tạo thành công, nguyên nhân có thể do phiên bản trình duyệt quá cũ và không hỗ trợ đối tượng *XMLHttpRequest*.

*Bài viết “About Native XMLHttpRequest” trên MSDN mô tả lịch sử và khía cạnh bảo mật của đối tượng XMLHttpRequest trong Internet Explorer. Bạn có thể tham khảo bài viết này tại địa chỉ <http://msdn2.microsoft.com/en-us/library/ms537505.aspx>.*

Hàm *readyAJAX()* trong Ví dụ 19-1 được gọi tới như sau:

```
var requestObj = readyAJAX();
```

Biến *requestObj* chứa đối tượng *XMLHttpRequest* trả về từ hàm *readAjax()*, trong trường hợp hàm này không khởi tạo thành công đối tượng *XMLHttpRequest*, biến *requestObj* sẽ chứa chuỗi “*Bạn cần sử dụng một trình duyệt mới hơn*”.

# Gửi yêu cầu AJAX

Với đối tượng `XMLHttpRequest` vừa được khởi tạo, bạn có thể sử dụng để gửi yêu cầu tới web server và nhận về các phản hồi. Để gửi yêu cầu tới web server, bạn phải sử dụng kết hợp hai phương thức `open()` và `send()` của đối tượng `XMLHttpRequest`.

Có hai cách cơ bản để gửi yêu cầu AJAX: đồng bộ và không đồng bộ. Khi gửi yêu cầu đồng bộ, đoạn mã gửi yêu cầu sẽ đợi phản hồi từ phía server - quá trình này được gọi là quá trình *chặn (blocking)*. Vì vậy, với yêu cầu đồng bộ, đoạn mã gửi yêu cầu sẽ chặn việc thực thi các mã JavaScript khác trong khi đợi phản hồi từ web server. Quá trình này bộc lộ những nhược điểm rõ ràng khi các yêu cầu hoặc phản hồi bị thất lạc trong quá trình truyền nhận hoặc đơn giản là bị chậm. Với yêu cầu không đồng bộ, dòng mã gửi yêu cầu không chặn các mã khác. Thay vào đó, nó sẽ kiểm tra trạng thái của yêu cầu để biết khi nào yêu cầu hoàn thành. Bạn sẽ tìm hiểu thêm về yêu cầu không đồng bộ trong phần sau của chương này còn trước tiên chúng ta sẽ làm việc với yêu cầu đồng bộ.

Trước khi gửi một yêu cầu tới server, bạn cần tạo ra nó. Phương thức `open` được sử dụng để tạo yêu cầu, phương thức này có ba đối số: phương thức gửi yêu cầu (`GET`, `POST`, `HEAD` hoặc khác), địa chỉ URL (Uniform Resource Locator) chứa địa chỉ của trang web mà bạn sẽ gọi tới và một giá trị Boolean `true` hoặc `false` cho biết bạn muốn gửi yêu cầu theo chế độ không đồng bộ hay đồng bộ.

Giả sử đối tượng `XMLHttpRequest` được truy xuất thành công thông qua hàm `readyAJAX()` và lưu vào biến `requestObj`, khi đó lời gọi không đồng bộ tới phương thức `open` sẽ như sau:

```
var url = "http://www.braingia.org/getdata.php";
requestObj.open("GET", url, true);
```

Tương tự, lời gọi đồng bộ sẽ như sau:

```
var url = "http://www.braingia.org/getdata.php";
requestObj.open("GET", url, false);
```

Quá trình gửi yêu cầu thực sự được thực hiện khi phương thức `send` được gọi:

```
requestObj.send();
```

**Chú ý** Nếu các tham số được gửi kèm với yêu cầu có chứa ký tự đặc biệt, ví dụ như ký tự trắng hoặc các ký tự đã được quy định trong URI RFC, trước tiên bạn phải thực hiện thoát những ký tự đó bằng cách sử dụng ký hiệu %. Kỹ thuật này được trình bày chi tiết trong RFC 3986, bạn có thể tham khảo tại địa chỉ <ftp://ftp.rfc-editor.org/in-notes/rfc3986.txt>. Hoặc bạn cũng có thể tham khảo tại [http://msdn2.microsoft.com/en-us/library/aa226544\(sql.80\)](http://msdn2.microsoft.com/en-us/library/aa226544(sql.80)).

### Làm thế nào trang web có thể làm việc chỉ với tối đa 500 từ

Giao thức truyền siêu văn bản (HTTP) là một ngôn ngữ web. Hiện tại, HTTP được định nghĩa trong tài liệu RFC 2616 và HTTP mô tả giao thức trao đổi thông tin thông qua yêu cầu từ được gửi từ phía client và phản hồi từ phía server.

Các yêu cầu từ phía client như trình duyệt chứa một tập cụ thể các header xác định phương thức truy xuất dữ liệu, đối tượng cần truy xuất và phiên bản giao thức được sử dụng. Các header khác chứa tên web server, ngôn ngữ yêu cầu, tên của trình duyệt và các thông tin liên quan khác mà client gửi kèm theo yêu cầu.

Dưới đây là một yêu cầu HTTP phiên bản 1.1 cơ bản, yêu cầu này chỉ chứa phần header quan trọng nhất:

```
GET / HTTP/1.1
Host: www.braingia.org
```

Yêu cầu trên xác định phương thức truy xuất tới tài liệu là phương thức *GET*, địa chỉ của tài liệu cần yêu cầu đặt tại thư mục / (gốc) và giao thức sử dụng là HTTP phiên bản 1.1. Dòng thứ hai, thường được gọi là header của host, là chuỗi URL *http://www.braingia.org*. Header này thông báo cho web server biết website nào đang được yêu cầu. Một yêu cầu có thể sử dụng nhiều phương thức khác nhau như: *GET*, *POST* và *HEAD*. Client và server cũng trao đổi các cookie HTTP chứa trong header. Các cookie được gửi kèm theo yêu cầu và được nhận về trong phản hồi.

Khi nhận được yêu cầu như trên, web server lưu trữ website <http://www.braingia.org> sẽ gửi các header phản hồi nhằm thông báo

yêu cầu đã được xử lý. Trong trường hợp này, web server sẽ gửi trả các header phản hồi tương tự như sau:

```
HTTP/1.1 200 OK
Date: Sat, 12 Mar 2011 01:04:34 GMT
Server: Apache/1.3.33 (Debian GNU/Linux) mod_perl/1.29 PHP/
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

Tài liệu được yêu cầu sẽ được gửi theo sau header. Header đầu tiên và cũng là header quan trọng nhất cho biết trạng thái phản hồi. Trong ví dụ này, mã của phản hồi là **200**, nghĩa là lời gọi không đồng bộ đã được xử lý thành công. Một số mã phản hồi thông dụng khác bao gồm: **404** (biểu thị rằng không tìm thấy tài liệu được yêu cầu), **302** (biểu thị sự chuyển hướng) và **500** (biểu thị rằng có lỗi xảy ra phía server).

Việc nắm kiến thức cơ bản về HTTP là rất quan trọng để có thể hiểu cách tạo ra một yêu cầu AJAX và cách gỡ lỗi cho các yêu cầu khi có lỗi xảy ra. Bạn có thể tìm hiểu thêm về HTTP, bao gồm các mã phản hồi, trong tài liệu RFC 2616 tại địa chỉ:

<ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt>



## Xử lý kết quả phản hồi của AJAX

Sẽ rất dễ dàng nếu làm việc với một yêu cầu đồng bộ vì toàn bộ mã JavaScript sẽ tạm ngừng thực thi cho đến khi có phản hồi. Biến requestObj cung cấp một phương thức hữu ích trong việc xử lý kết quả trả về bằng cách cung cấp luôn mã trạng thái và mô tả trạng thái gửi về từ server. Tuy vậy, bất kể là yêu cầu đồng bộ hay không đồng bộ, mã trạng thái cũng cần được kiểm soát để chắc chắn là toàn bộ quá trình đã diễn ra thành công (thông thường mã trạng thái có giá trị 200 nếu thành công).

Thuộc tính *responseText* chứa toàn bộ văn bản trả về từ web server.

Ví dụ, giả sử ứng dụng phía server thực hiện việc trả về tổng của hai số và lời gọi yêu cầu tính tổng của 2 và 56 là:

<http://www.braingia.org/addtwo.php?num1=2&num2=56>

Toàn bộ mã của yêu cầu kiểu đồng bộ và xử lý kết quả phản hồi trả về từ server như sau:

```
requestObj.open("GET", "http://www.braingia.org/addtwo.php?num1=2&num2=56");
requestObj.send();
if (requestObj.status == 200) {
 alert(requestObj.responseText);
} else {
 alert(requestObj.statusText);
}
```

Trong ví dụ này, *requestObj* là biến được tạo bởi hàm *readyAJAX()* được xây dựng tại phần trước. Tâm điểm của đoạn mã mà phương thức *open* xử dụng lệnh GET để yêu cầu đến địa chỉ *http://www.braingia.org/addtwo.php?num1=2&num2=56*, yêu cầu này được gửi theo kiểu đồng bộ (tham số tương ứng khi gọi phương thức *open* là *false*). Sau đó, phương thức *send* được gọi để thực sự gửi toàn bộ các yêu cầu đến web server.

Khi client nhận được phản hồi từ phía web server, nó sử dụng thuộc tính *status* để kiểm tra trạng thái của dữ liệu trả về. Nếu giá trị nhận được là 200, mọi thứ đều suôn sẻ, hàm *alert* được gọi để hiển thị văn bản lưu trong thuộc tính *responseText*, đó chính là dữ liệu phản hồi của web server cho client. Nếu *status* có giá trị khác 200, nội dung biến *statusText* sẽ được hiển thị.

Việc xử lý các yêu cầu kiểu không đồng bộ hơi phức tạp hơn một chút. Khi gửi một yêu cầu loại này, các đoạn mã lệnh phía sau vẫn tiếp tục chạy. Do đó, sẽ rất khó xác định thời điểm mà phản hồi được nhận đầy đủ. Để kiểm soát được chính xác trạng thái của quá trình phản hồi từ server, chúng ta có thể sử dụng sự kiện *onreadystatechange* để kích hoạt đoạn mã kiểm tra sự thay đổi của thuộc tính *readyState* của sự kiện này để xác định việc này. Trong Chương 17 đã nhắc đến thuộc tính *readyState* với năm trạng thái như bảng 19-1 dưới đây:

### BẢNG 19-1 Các giá trị của thuộc tính *readyState*.

| Giá trị | Mô tả                                              |
|---------|----------------------------------------------------|
| 0       | Chưa được khởi tạo. Đã mở nhưng chưa được gọi tới. |
| 1       | Mở. Đã được khởi tạo nhưng chưa được gửi.          |
| 2       | Đã gửi. Yêu cầu đã được gửi.                       |
| 3       | Đang nhận. Đang nhận phản hồi.                     |

Trên thực tế, trạng thái duy nhất của thuộc tính `readyState` mà người lập trình JavaScript và AJAX quan tâm đó là 4 – Đã được tải (loaded). Việc thực hiện xử lý các phản hồi có giá trị `readyState` khác 4 sẽ có thể gây ra lỗi.

Thông thường bạn sẽ sử dụng một hàm đồng bộ để xử lý sự kiện `onreadystatechange` đối với các lời gọi AJAX đồng bộ. Hàm này thực hiện kiểm tra thuộc tính `readyState` có bằng 4 hay không, sau đó kiểm tra để đảm bảo trạng thái phản hồi bằng 200, tức là yêu cầu thành công. Hàm này có dạng như sau:

```
requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {
 alert(requestObj.responseText);
 } else {
 alert(requestObj.statusText);
 }
 }
}
```



Trong bài tập tiếp theo, bạn sẽ tạo một đối tượng `XMLHttpRequest` và gửi yêu cầu tới web server để truy xuất một cuốn sách dựa vào mã ISBN. Bạn cần một web server và đoạn mã (chương trình) phía server để in ra các phản hồi vì các yêu cầu được gửi qua đối tượng `XMLHttpRequest` tuân thủ chính sách cùng nguồn gốc (Same Origin Policy).

*Chính sách cùng nguồn gốc* quy định rằng các yêu cầu chỉ được gửi tới server trong domain cung cấp đoạn mã đang gọi. Nói cách khác, nếu tôi đang xử lý đoạn mã trực tiếp từ web server tại địa chỉ `http://www.braingia.org`, đoạn mã của tôi chỉ có thể gọi tới máy chủ đó và nhận về một phản hồi. Nếu gọi tới URL trên một web server khác, chính sách cùng nguồn gốc sẽ ngăn không cho đoạn mã nhận phản hồi.

**Chú ý** Có một số cách để loại bỏ tính năng bảo mật của *chính sách cùng nguồn gốc* đó là sử dụng HTTP proxy hoặc viết các chương trình phía server để gửi yêu cầu đại diện cho chương trình gọi; tuy nhiên, cuốn sách này sẽ không đề cập tới những cách thức đó.

Trong bài tập tiếp theo, đoạn mã hay chương trình phía server cần trả về chuỗi "*JavaScript - Hướng dẫn học qua ví dụ*" khi nhận được yêu cầu *GET* với đối số là cặp tên/giá trị như sau:

isbn=9780735624498

Dưới đây là ví dụ về một chương trình phía server cơ bản nhất được viết bằng ngôn ngữ VBScript và cài đặt trong một trang ASP:

```
<%
dim isbn
isbn=Request.QueryString("isbn")
If isbn<>"" Then
 If isbn=="9780735624498" Then
 Response.Write("JavaScript - Hướng dẫn học qua ví dụ")
 End If
End If
%>
```

Chương trình với tính năng tương tự được viết bằng PHP sẽ có dạng như sau:

```
<?php
$isbn = $_GET['isbn'];
if (! $isbn) {
 print "Yêu cầu không hợp lệ.";
} else if ($isbn == "9780735624498") {
 print "JavaScript - Hướng dẫn học qua ví dụ";
}
?>
```



Trong bài tập tiếp theo, URL mà yêu cầu sẽ được gửi tới đã được định nghĩa sẵn; tuy nhiên, bạn phải thay thế URL đó bằng URL chứa đường dẫn tới chương trình phía server của bạn. Vì chính sách cùng nguồn gốc nên chương trình phía server phải nằm trong cùng domain với trang web gọi tới nó.

### Gửi yêu cầu và nhận phản hồi với XMLHttpRequest

1. Tạo chương trình phía server để trả về tựa sách khi nhận được đối số *isbn* có giá trị như trong phần trước. Chương trình này có thể được viết bằng ngôn ngữ bất kỳ. (Nếu cần, bạn có thể tham khảo hai ví dụ trên).
2. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo khác,

chỉnh sửa lại file isbn.htm trong thư mục mã nguồn mẫu Chương 19 (phần Tài nguyên đi kèm).

- Trong trang này, thay thế dòng chú thích TODO bằng đoạn mã in đậm sau đây (Xem trong file isbn.txt, phần Tài nguyên đi kèm). Hãy nhớ thay thế biến *url* bằng đường dẫn URL tương ứng tới chương trình phía server của bạn:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>ISBN</title>
</head>
<body>
<div id="data"></div>
<script type="text/javascript">
 function readyAJAX() {
 try {
 return new XMLHttpRequest();
 } catch(e) {
 try {
 return new ActiveXObject('Microsoft.XMLHTTP');
 } catch(e) {
 try {
 return new ActiveXObject('Microsoft.AXHTTP');
 } catch(e) {
 return "Bạn cần dùng một trình duyệt hỗ trợ JavaScript";
 }
 }
 }
 }
 var requestObj = readyAJAX();
 var url = "http://www.braingia.org/isbn.php?isbn=9780735624291";
 requestObj.open("GET", url, true);
 requestObj.send();
 requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {
 alert(requestObj.responseText);
 } else {
 alert(requestObj.statusText);**
 }
 }
 }
</script>
```

```
 }
 }
}
</script>
</body>
</html>
```

4. Lưu lại, sau đó dùng trình duyệt để mở trang này. Bạn sẽ nhận được một thông báo như trong hình dưới đây:



Xin chúc mừng! Bạn đã hoàn thành xong bài tập làm việc với đối tượng `XMLHttpRequest`.



## Xử lý phản hồi XML

Những ví dụ AJAX bạn đã thấy tới lúc này đều sử dụng các phản hồi định dạng HTML và văn bản thuần túy từ web server, do đó bạn có thể truy xuất tới những phản hồi đó thông qua phương thức `responseText` của đối tượng `XMLHttpRequest`. Tuy nhiên, ứng dụng phía server có thể trả về phản hồi định dạng XML, khi đó bạn phải sử dụng phương thức `responseXML` để xử lý những phản hồi này.

Trong phần trước của chương, “Làm thế nào trang web có thể làm việc chỉ với tối đa 500 từ”, đã có một ví dụ về định dạng phản hồi từ web server. Phản hồi đó có chứa header `Content-Type`:

`Content-Type: text/html; charset=iso-8859-1`

Để truy xuất phản hồi bằng phương thức `responseXML`, web server cần gửi header `Content-Type` có giá trị là `text/xml` hoặc `application/xml`:

Content-Type: application/xml.

Khi đối tượng `XMLHttpRequest` nhận được phản hồi XML, bạn có thể sử dụng các phương thức của DOM để xử lý phản hồi đó.

Phương thức `responseXML` tương đối bất ổn định, phương thức này có thể dẫn tới kết quả không như mong đợi, tùy thuộc vào trình duyệt và hệ điều hành được sử dụng. Ngoài ra, phương thức `responseXML` không được hỗ trợ rộng rãi như các phương thức JavaScript khác. Sử dụng phương thức `responseXML` có nghĩa là bạn phải kết hợp kỹ thuật làm việc với `XMLHttpRequest` (đã được giới thiệu ở phần đầu chương) với kỹ thuật phân tách XML được giới thiệu trong Chương 17. Ví dụ, hãy cùng xét file XML sau đây (xem file book.xml):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<book>
 <title>JavaScript - Hướng dẫn học qua ví dụ</title>
 <isbn>9780735624498</isbn>
</book>
```

Kết hợp đối tượng `XMLHttpRequest` và kỹ thuật phân tách XML dẫn tới đoạn mã dưới đây, đoạn mã này truy xuất và hiển thị giá trị ISBN trong file book.xml:

```
var requestObj = readyAJAX();
var url = "http://www.braingia.org/book.xml";
requestObj.open("GET", url, false);
requestObj.send();
if (requestObj.status == 200) {
 var xmldocument = requestObj.responseXML;
 alert(xmldocument.getElementsByTagName("isbn")[0].ch
} else {
 alert(requestObj.statusText);
}
```

Khi yêu cầu hoàn tất, phương thức `requestObj.responseXML` sẽ chứa file XML được yêu cầu (book.xml). Dòng mã

`xmldocument.getElementsByTagName("isbn")` truy xuất một mảng trong thẻ `<isbn>` của file book.xml. Chỉ có một thẻ duy nhất chứa giá trị `isbn` trong file; chỉ số `[0]` biểu thị thẻ đầu tiên. Phần `.childNodes[0]` của đoạn mã truy xuất nút con đầu tiên của thẻ `<isbn>`. Trong trường hợp này, đó là nút văn bản chứa số hiệu mã ISBN. Cuối cùng, phần `.nodeValue` của đoạn mã truy xuất tới giá trị của nút văn bản, sau đó hiển thị mã ISBN thông qua hộp thoại `alert`.

# Làm việc với JSON

JavaScript Object Notation (JSON) là cách thức truyền dữ liệu theo định dạng đối tượng và mảng thuần của JavaScript, thay vì mã hóa dữ liệu bên trong các phản hồi XML (hoặc HTML). JSON rất hiệu quả trong việc truyền dữ liệu từ server tới client. Việc phân tách XML sử dụng DOM khá phức tạp và chậm, trong khi đó việc phân tách dữ liệu theo định dạng JSON được thực hiện trực tiếp bằng JavaScript.

Hãy nhớ lại file book.xml trong ví dụ ở phần trước. Dữ liệu của file này theo định dạng JSON sẽ được biểu diễn như sau:

```
{
"book":
{
 "title": "JavaScript - Hướng dẫn học qua ví dụ",
 "isbn": "9780735624498"
}
}
```

Việc truy xuất tới một phần tử đơn lẻ trong định dạng JSON dễ dàng hơn so với trong định dạng XML. Bạn có thể sử dụng hàm JavaScript `eval()` để phân tách phản hồi JSON. Ví dụ dưới đây là đoạn mã truy xuất và hiển thị tựa sách (title):

```
var requestObj = readyAJAX();
var url = "http://www.braingia.org/json.php";
requestObj.open("GET", url, true);
requestObj.send();
if (requestObj.status == 200) {
 var xmldocument = eval('(' + requestObj.responseText)
 alert(xmldocument.book.title);
} else {
 alert(requestObj.statusText);
}
```

Việc sử dụng định dạng JSON có một nguy cơ mang tính cốt hữu về bảo mật, bởi bạn cần sử dụng hàm `eval()` để phân tách phản hồi từ server. Về bản chất, hàm `eval()` thực thi đoạn mã JavaScript nhận được, vì vậy nếu đoạn mã JavaScript đó chứa mã độc, đoạn mã sẽ được chạy trong môi trường của ứng dụng hiện hành. Nhiệm vụ của bạn là đảm bảo dữ liệu mà ứng dụng sử dụng với định dạng JSON phải hoàn toàn vô hại và không chứa mã độc có thể gây ra các sự cố khi

thực thi với hàm `eval()`.

Sử dụng framework JavaScript như jQuery có thể giảm được nguy cơ bảo mật vì jQuery sử dụng JSON theo chuẩn ECMA-262 phiên bản 5. Bạn sẽ tìm hiểu về jQuery và cách sử dụng jQuery để xử lý JSON trong Chương 22 “Giới thiệu về jQuery”.

## Xử lý header

Phương thức `HTTP HEAD` chỉ trả về phản hồi chứa header thay vì cả header và nội dung như phương thức `GET`. Phương thức `HEAD` rất hữu dụng khi xác định xem một nguồn tài nguyên có bị cập nhật hoặc thay đổi hay không.

Một trong những header HTTP thường xuyên được gửi từ server đó là `Expires` (*thời hạn*), header này cho biết khi nào client nên yêu cầu bản sao mới của tài liệu thay vì đọc tài liệu đó từ bộ nhớ đệm. Nếu server gửi header `Expires`, phương thức `HEAD` rất hiệu quả để xem và phân tách header `Expires` vì phương thức này chỉ truy xuất header thay vì toàn bộ phần thân (body) của tài liệu được yêu cầu.



Để chỉ yêu cầu phản hồi chỉ chứa header từ server dù sử dụng yêu cầu `HEAD` hoặc bất kỳ kiểu yêu cầu nào khác như `GET` hoặc `POST`, hãy sử dụng phương thức `getAllResponseHeaders()` của đối tượng `XMLHttpRequest` như sau:

```
requestObj.getAllResponseHeaders();
```

Ví dụ 19-2 trình bày cách thức truy xuất header trong phản hồi từ trang chủ website của tôi.

**VÍ DỤ 19-2** Truy xuất các header.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Header phản hồi</title>
</head>
<body>
<div id="data"></div>
```

```

<script type="text/javascript">
function readyAJAX() {
 try {
 return new XMLHttpRequest();
 } catch(e) {
 try {
 return new ActiveXObject("Msxml2.X
 } catch(e) {
 try {
 return new ActiveXObject("
 } catch(e) {
 return "Bạn cần sử dụng mộ
 }
 }
 }
}
var requestObj = readyAJAX();
var url = "http://www.braingia.org/";
requestObj.open("HEAD",url,true);
requestObj.send();
requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {
 alert(requestObj.getAllResponseHea
 } else {
 alert(requestObj.statusText);
 }
 }
}
</script>
</body>
</html>

```

**Trợ giúp** Chính sách cùng nguồn gốc mà bạn đã làm quen ở phần trước cũng được áp dụng tương tự cho phương thức *HEAD* trong Ví dụ 19-2. Khi viết Ví dụ 19-2, ban đầu tôi đã quên mất chính sách này và thiết lập biến *url* bằng *http://www.microsoft.com/*. Khi đó, tôi nghĩ rằng tôi có thể truy xuất đến trang chủ của website đó. Tuy nhiên, sau đó lỗi xảy ra, tôi đã tìm ra nguyên nhân và thay đổi biến *url* đúng bằng domain thực thi đoạn mã. Bạn cũng có thể gặp lỗi tương tự. Hãy nhớ thay đổi biến *url* trong Ví

độ 19-2 sao cho đúng với server mà đoạn mã thực thi.

## Sử dụng phương thức POST

Cho tới thời điểm này, các ví dụ mà bạn làm việc đều sử dụng phương thức *GET* và *HEAD* để truy xuất dữ liệu từ server. Khi gửi các yêu cầu thông qua giao thức HTTP, bạn thường sử dụng phương thức *POST*. Sử dụng phương thức *POST* với đối tượng *XMLHttpRequest* phức tạp hơn so với sử dụng phương thức *GET* hoặc *HEAD*. Tuy nhiên, phương thức *POST* có hai ưu điểm mà phương thức *GET* không có. Đầu tiên, các tham số gửi qua yêu cầu *POST* được chứa trong phần thân của yêu cầu thay vì trong *URL* như phương thức *GET*, do đó những tham số đó khó có thể bị phát hiện qua quan sát thông thường. Thứ hai, phương thức *POST* hỗ trợ những yêu cầu có kích thước lớn. Một số server thường giới hạn kích thước của yêu cầu *GET* trong một số lượng ký tự nhất định và mặc dù những server đó cũng giới hạn kích thước của yêu cầu *POST*, giới hạn đó vẫn lớn hơn nhiều so với yêu cầu *GET*.

Phương thức HTTP *POST* đòi hỏi một header bổ sung trong yêu cầu. Việc thiết lập header được thực hiện thông qua phương thức *setRequestHeader()*:

```
requestObj.setRequestHeader(header, value);
```

Ví dụ, để thiết lập header *Content-Type* cho một web form sử dụng yêu cầu *POST*, bạn có thể viết như sau:

```
requestObj.setRequestHeader("Content-type", "application/x-wl")
```

Trong ví dụ trước, khi yêu cầu AJAX sử dụng phương thức *GET*, URL sẽ bao gồm các đối số hoặc cặp *tên/giá trị* truyền cho ứng dụng, giống như sau:

<http://www.braingia.org/books/javascriptsbs/isbn.php?isbn=97>

Trong ví dụ trước, tham số *isbn* có giá trị bằng *9780735624498*. Tuy nhiên khi sử dụng yêu cầu *POST*, URL chỉ chứa tên tài liệu hoặc nguồn được yêu cầu - URL không chứa bất kỳ tham số nào. Do đó bạn phải gửi những tham số này tới server thông qua phương thức *send()*.

Ví dụ 19-3 biểu diễn một yêu cầu AJAX sử dụng phương thức *POST*, đoạn mã tương ứng được in đậm. Yêu cầu đó sử dụng hai tham số - bạn hãy thử xác

định những tham số đó.

### VÍ DỤ 19-3 Tạo một yêu cầu POST.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>POST</title>
</head>
<body>
<div id="xmldata"></div>
<script type="text/javascript">
function readyAJAX() {
 try {
 return new XMLHttpRequest();
 } catch(e) {
 try {
 return new ActiveXObject("Msxml2.X
 } catch(e) {
 try {
 return new ActiveXObject("Microsoft
 } catch(e) {
 return "Bạn cần sử dụng tr
 }
 }
 }
}
var requestObj = readyAJAX();
var url = "http://www.braingia.org/books/javascriptsbs/pos
var params = "num1=2&num2=2";
requestObj.open("POST",url,true);
requestObj.setRequestHeader("Content-type",
 "application/x-www-form-urlencoded");
requestObj.send(params);
requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {
 alert(requestObj.responseText);**
 } else {
 alert(requestObj.statusText);**
 }
 }
}
```

```
 }
 }
</script>
</body>
</html>
```

Ví dụ 19-3 tạo ra hai tham số và lưu vào biến *params*:

```
var params = "num1=2&num2=2";
```

Sau khi tạo ra đối tượng *XMLHttpRequest* (*requestObj*), các tham số được truyền vào dưới dạng đối số của phương thức *send()*:

```
requestObj.send(params);
```

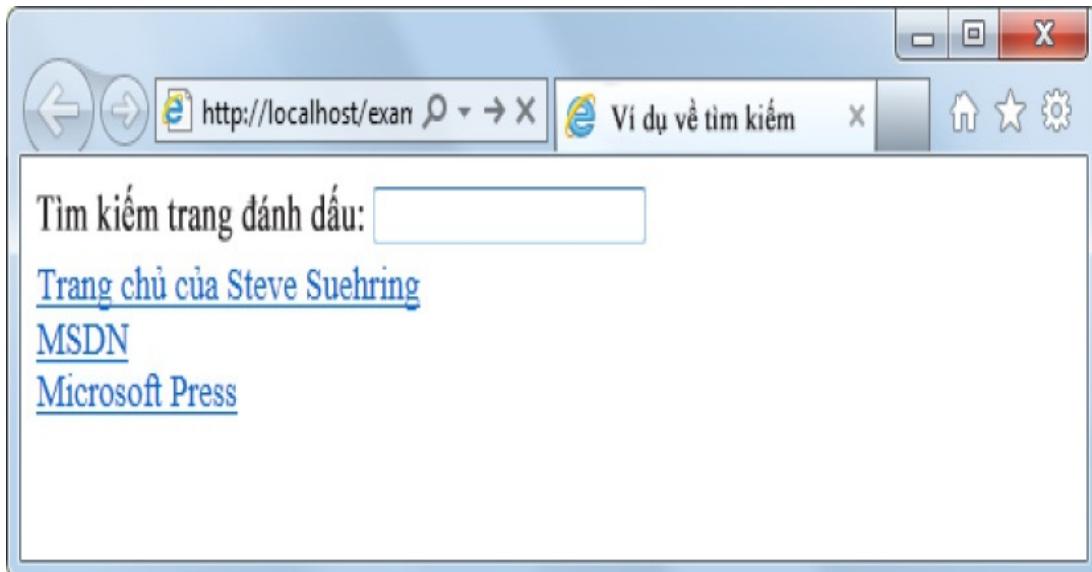
## Bài tập thực tế: Tìm kiếm và cập nhật thông tin trực tiếp

Có rất nhiều ví dụ về việc sử dụng JavaScript để hỗ trợ gợi ý hay đoán từ khóa và tìm kiếm như ví dụ đã trình bày trong Chương 18 “Các ứng dụng JavaScript”. Một ví dụ khác là tạo form tìm kiếm những địa chỉ email nằm trong danh sách đen hoặc danh sách trắng thông qua bộ lọc thư rác.

Công cụ tìm kiếm danh sách đen/trắng sử dụng AJAX để nhập file XML và đưa ra kết quả dựa trên dữ liệu mà người quản trị web nhập vào. Bạn có thể dễ dàng chỉnh sửa ứng dụng này để cung cấp tính năng tìm kiếm hoặc đánh dấu trang trực tiếp. Chương 20 “Tìm hiểu thêm về AJAX” sử dụng một phần trong ứng dụng của Chương 18 để tạo form tìm kiếm trực tiếp (live search form - hiển thị kết quả tìm kiếm ngay khi người dùng nhập liệu lên ô tìm kiếm), còn phần tiếp theo sẽ giới thiệu một phiên bản của ứng dụng đó để tạo danh sách đánh dấu trang (bookmark) sử dụng XML và có khả năng tìm kiếm trực tiếp.

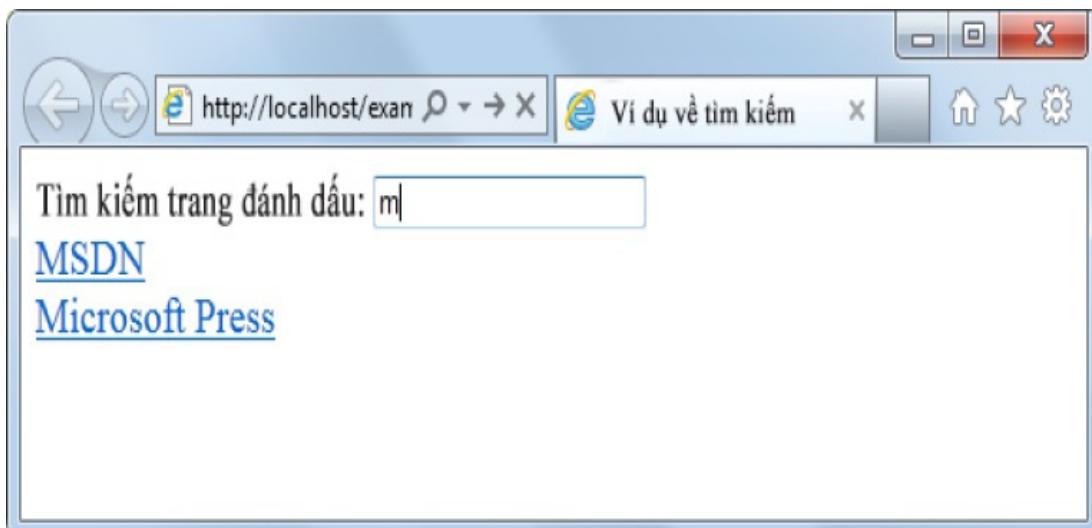
Việc truy cập vào danh sách đánh dấu trang của trình duyệt web từ nhiều máy tính là một tính năng cần thiết. Với ý tưởng đó, ví dụ sau sẽ hướng dẫn bạn cách tạo ứng dụng AJAX cung cấp danh sách đánh dấu trang. Trang web này quản lý các trang được đánh dấu sử dụng file XML nằm ở vị trí trung tâm. Đoạn mã truy xuất file XML và tạo ra một trang web hiển thị danh sách trang đánh dấu đồng thời cung cấp giao diện tìm kiếm.

Ứng dụng danh sách đánh dấu trang có giao diện như trong Hình 19-1. Trong hình, ứng dụng chỉ hiển thị ba trang được đánh dấu, nhưng cơ chế hoạt động của danh sách 3 hoặc 300 trang đánh dấu đều giống nhau. Ví dụ đơn giản này giúp trình bày vấn đề dễ hiểu hơn.



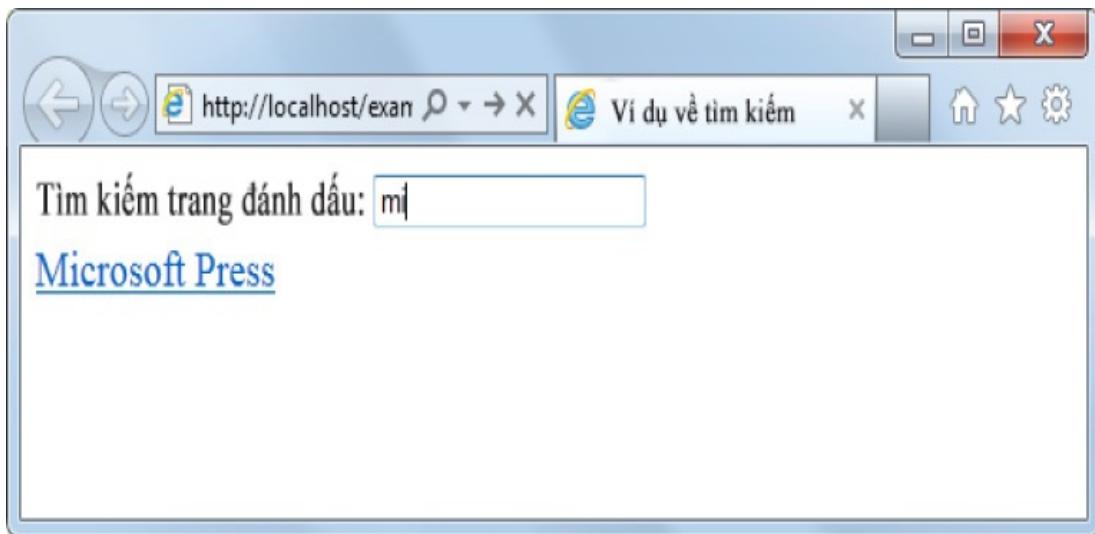
HÌNH 19-1 Ứng dụng tìm kiếm trang đánh dấu theo thời gian thực.

Ô tìm kiếm giúp lọc ra các trang đã được đánh dấu ngay khi người dùng nhập từ khóa vào ô tìm kiếm. Ví dụ, nếu người dùng gõ ký tự **m** vào ô tìm kiếm, trang web sẽ thay đổi sao cho chỉ những đánh dấu trang bắt đầu bằng ký tự *m* được hiển thị, giống như trong Hình 19-2.



HÌNH 19-2 Nhập ký tự *m* giới hạn danh sách chỉ hiển thị các trang đánh dấu bắt đầu bằng ký tự *m*.

Nếu nhập tiếp ký tự vào ô tìm kiếm, ví dụ nếu nhập thêm ký tự **i** để tạo ra chữ **mi**, kết quả tiếp tục được lọc như trong Hình 19-3.



HÌNH 19-3 Thêm các ký tự vào ô tìm kiếm để tiếp tục giới hạn các kết quả tìm kiếm.

Khi người dùng xóa chữ trong text box, trang web sẽ trở lại trạng thái mặc định (như trong Hình 19-1).

Dưới đây là mã XML của ứng dụng này (xem trong file bookmark.xml, phần Tài nguyên đi kèm):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bookmarks xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <bookmark>
 <title>Trang chủ của Steve Si...
 <url>http://www.braingia.org...
 </bookmark>
 <bookmark>
 <title>MSDN</title>
 <url>http://msdn.microsoft.co...
 </bookmark>
 <bookmark>
 <title>Microsoft Press</title>
 <url>http://www.microsoft.co...
 </bookmark>
</bookmarks>
```

Đoạn mã cho ứng dụng cùng với trang web được trình bày ở đây và có trong file bookmark.htm (phần Tài nguyên đi kèm):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ về tìm kiếm</title>
</head>
<body>
<form name="nameform" id="nameform" action="" method="post">
Tìm kiếm trang đánh dấu: <input id="textname" type="text" name="textname">
</form>
<div id="data"></div>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
function textSearch() {
 var textName = document.getElementById("textname");
 var dataNode = document.getElementById("data");
 while (dataNode.hasChildNodes()) {
 dataNode.removeChild(dataNode.firstChild);
 }
 listName(textName.value);
}
function readyAJAX() {
 try {
 return new XMLHttpRequest();
 } catch(e) {
 try {
 return new ActiveXObject("Msxml2.XMLHTTP");
 } catch(e) {
 try {
 return new ActiveXObject("Microsoft.XMLHTTP");
 } catch(e) {
 return "Bạn cần sử dụng trình duyệt hỗ trợ";
 }
 }
 }
}
function listName(text) {
 var xmlEl = AJAXresponse.getElementsByTagName("bookmarks");
 elLength = xmlEl.length;
 for (i = 0; i < elLength; i++) {
 var div = document.createElement("div");
 // Tạo các phần tử cho mỗi dòng
```

```

 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 // Bỏ qua phần tử có kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeType == 1)
 continue;
 }
 var url = new RegExp("http");
 if (!xmlEl[i].childNodes[j].firstChild)
 var pattern = "^" + text;
 var title = xmlEl[i].childNodes[j].firstChild.textContent;
 var nameRegexp = new RegExp(pattern);
 var existDiv = document.getElementById("data");
 if (!existDiv) {
 if (title.match(nameRegexp)) {
 var anchor = document.createElement("a");
 var xmlData = document.createDocumentFragment();
 document.createElement("div").appendChild(anchor);
 var urls = Array();
 anchor.setAttribute("href", url);
 anchor.append(xmlData);
 div.appendChild(anchor);
 }
 }
 }
 document.getElementById("data").appendChild(div);
 }
 }
}

var requestObj = readyAJAX();
var url = "http://www.braingia.org/books/javascriptsbs/bookmarks.html";
requestObj.open("GET", url, true);
requestObj.send();
var AJAXresponse;
requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {
 AJAXresponse = requestObj.responseXML;
 listName("");
 } else {
 alert(requestObj.statusText);
 }
 }
}

```



```

}
var textEl = document.getElementById("textname");
EHandler.add(textEl,"keyup", function() { textSearch(); });
</script>
</body>
</html>

```

Phần JavaScript của đoạn mã trên được chia thành một số hàm, tôi sẽ giải thích ngắn gọn từng hàm ngay sau đây. Đầu tiên, đoạn mã liên kết tới đoạn mã xử lý sự kiện ehandler.js, file này đã được tạo trong Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt”:

```
<script type="text/javascript" data-src="ehandler.js"></script>
```

Mã HTML của trang web chỉ chiếm một số ít dòng. Dưới đây là mã HTML của web form:

```
<form name="nameform" id="nameform" action="" method="post">
</form>
```

Và dưới đây là thẻ *div* chứa các trang đánh dấu:

```
<div id="data"></div>
```



Phần Javascript của đoạn mã trên khai báo một số hàm và thực thi đoạn mã sau đây trong khối lệnh chính. Đoạn mã này tương tự đoạn mã mà bạn đã gặp trong suốt chương này: đoạn mã sử dụng hàm *readyAJAX()* và gửi một yêu cầu AJAX tới server để truy xuất file XML chứa danh sách các trang đã được đánh dấu. Khi nhận được phản hồi, đoạn mã gọi tới hàm *listName()*.

Ngoài đoạn mã AJAX, một hàm xử lý sự kiện cũng được gắn cho text box trên web form. Sự kiện được xử lý là *keyup*, sự kiện này được kích hoạt khi một phím được nhấn và nhả trong text box. Đoạn mã này sẽ như sau:

```

var requestObj = readyAJAX();
var url = "http://www.braingia.org/books/javascriptsbs/bookmarks.xml";
requestObj.open("GET",url,true);
requestObj.send();
var AJAXresponse;
requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {

```

```

 AJAXresponse = requestObj.responseXML;
 listName("");
 } else {
 alert(requestObj.statusText);
 }
}
var textEl = document.getElementById("textname");
EHandler.add(textEl,"keyup", function() { textSearch(); });

```

Hàm xử lý sự kiện liên quan đến các thao tác phím trên form tìm kiếm nằm trong hai hàm *textSearch* và *listName*. Hàm *textSearch* chịu trách nhiệm loại bỏ các đánh dấu trang khỏi danh sách và gọi tới hàm *listName()*.

```

function textSearch() {
 var textName = document.getElementById("textname");
 var dataNode = document.getElementById("data");
 while (dataNode.hasChildNodes()) {
 dataNode.removeChild(dataNode.firstChild);
 }
 listName(textName.value);
}

```



Cuối cùng, hàm *listName()* chứa **đoạn** mã hiển thị trang đánh dấu có ít nhất một phần trùng khớp với những ký tự được nhập vào ô tìm kiếm. Nếu ô tìm kiếm trống, hàm *listName()* sẽ hiển thị tất cả danh sách các trang đã được đánh dấu.

```

function listName(text) {
 var xmlEl = AJAXresponse.getElementsByTagName("bookmarks");
 elLength = xmlEl.length;
 for (i = 0; i < elLength; i++) {
 var div = document.createElement("div");
 // ra các phần tử cho mỗi dòng
 for (j = 0; j < xmlEl[i].childNodes.length;
 j++) {
 // Bỏ qua các phần tử có kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeType
 != 1)
 continue;
 }
 var url = new RegExp("http");
 if (! xmlEl[i].childNodes[j].firstChild)
 var pattern = "^" + text;

```

```

 var title = xmlEl[i].childNodes[0].nodeValue;
 var nameRegexp = new RegExp('([a-zA-Z]+)([a-zA-Z0-9_]*)([a-zA-Z]+)');
 var existDiv = document.getElementById("data");
 if (!existDiv) {
 if (title.match(nameRegexp)) {
 var anchor = document.createElement("a");
 var xmlData = document.createDocumentFragment();
 document.createElement("div").appendChild(xmlData);
 var urls = Array();
 anchor.setAttribute("href", "http://www.google.com");
 anchor.appendChild(document.createTextNode(title));
 div.appendChild(anchor);
 div.appendChild(xmlData);
 }
 }
 }
}
document.getElementById("data").appendChild(div);
}
}

```



## Bài tập

1. Phương thức yêu cầu HTTP nào được đề cập trong chương này có tính bảo mật cao nhất? Vì sao?
2. Hãy nêu sự khác nhau giữa yêu cầu/phản hồi *XMLHttpRequest* sử dụng HTML, sử dụng XML và sử dụng JSON.
3. Tạo một chương trình phía server trả về tổng của hai số, hai số đó được client gửi tới chương trình dưới dạng tham số. Sau đó sử dụng đối tượng *XMLHttpRequest* và lời gọi không đồng bộ để gọi tới chương trình vừa tạo.

# Chương 20

## Tìm hiểu thêm về AJAX

Sau khi đọc xong chương này, bạn có thể:

- Nắm rõ cách kết hợp AJAX và CSS.
- Hiểu thêm về quan hệ giữa DOM, AJAX và CSS.
- Sử dụng AJAX và CSS để tạo và áp dụng style cho bảng HTML với dữ liệu XML.
- Sử dụng CSS tạo ô thả xuống trên nền AJAX.

Trong chương trước, bạn đã biết cách sử dụng đối tượng `XMLHttpRequest` để gửi, nhận, xử lý các yêu cầu và cách tạo ra một ứng dụng AJAX. Trong chương này, bạn sẽ học cách sử dụng CSS (Cascading Style Sheets) để hiển thị dữ liệu nhận về với AJAX.



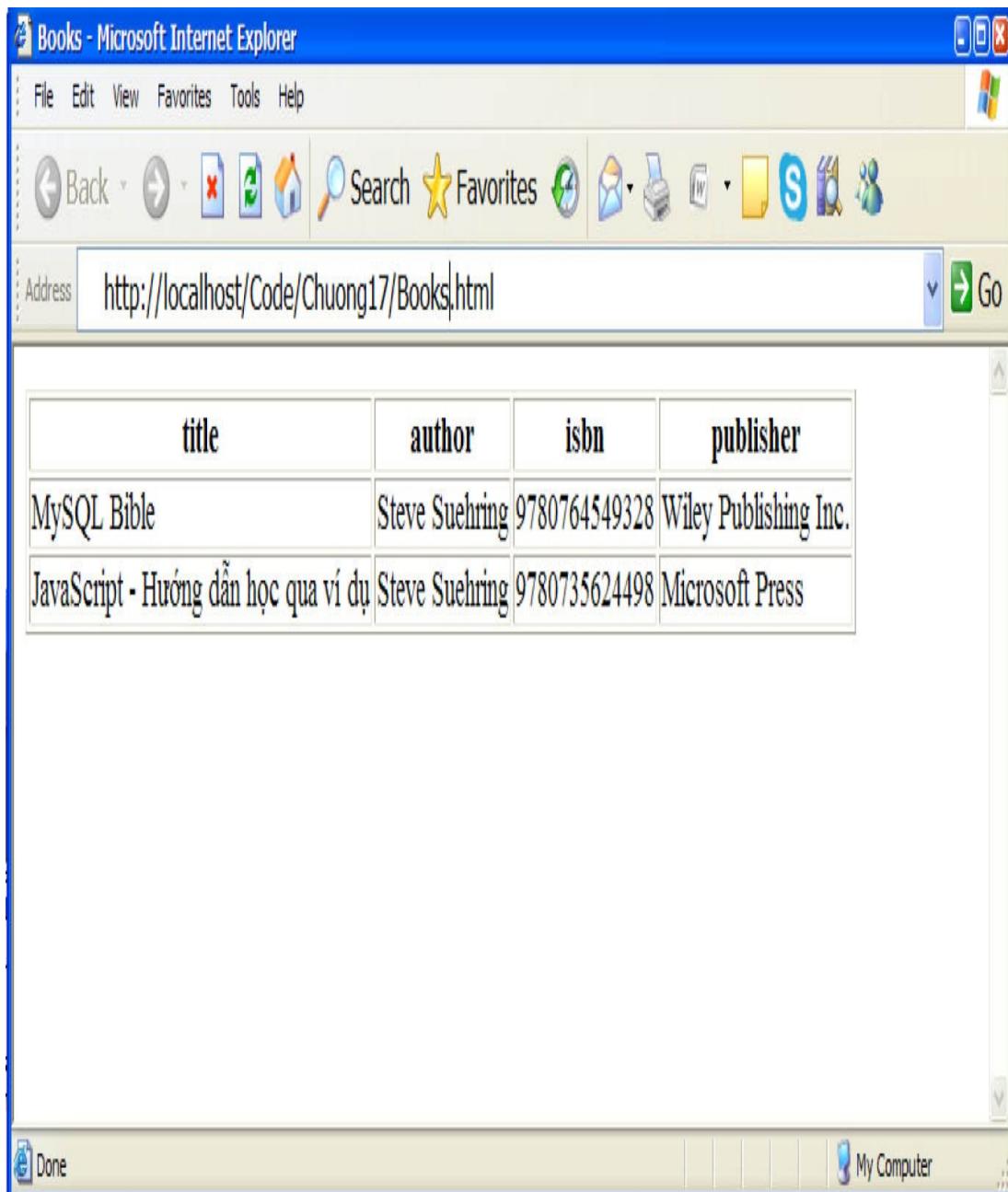
Mối quan hệ giữa JavaScript và CSS đã được đề cập trong Chương 15 “JavaScript và CSS”. Trong Chương 15, bạn đã học cách thay đổi style (định dạng trình bày) cho tài liệu bằng JavaScript. Chương 17 “JavaScript và XML” giới thiệu cách hiển thị dữ liệu XML dưới dạng bảng HTML và Chương 19 “Sơ lược về AJAX” giới thiệu cách sử dụng CSS và DOM để tạo một trang tìm kiếm các trang được đánh dấu (bookmark). Trong chương này, bạn sẽ học cách sử dụng CSS để áp dụng style cho bảng tạo ra trong Chương 17 và mở rộng ứng dụng tìm kiếm trong Chương 19 với cả CSS và JavaScript.

Qua chương này, tôi hy vọng có thể truyền tải một điều: sử dụng AJAX rất dễ dàng. Việc truy xuất và phân tích thông tin bằng đối tượng `XMLHttpRequest` cũng khá đơn giản, điều quan trọng là thao tác với dữ liệu. Đó là lý do tại sao CSS và DOM đóng vai trò quan trọng đến vậy. Có thể nói, AJAX là môi trường để bạn vận dụng tất cả các kiến thức về JavaScript học được trong cuốn sách này để tạo ra những ứng dụng lớn.

# Tạo bảng HTML với XML và CSS

Chương 17 đã trình bày ví dụ đọc dữ liệu từ file XML và sử dụng dữ liệu đó để tạo bảng HTML như minh họa trong Hình 20-1. Đoạn mã để tạo bảng dữ liệu này được trình bày trong Chương 17 và được cải tiến để hiển thị không chỉ dữ liệu mà còn cả các tiêu đề cột. Kết quả của đoạn mã cuối Chương 17 được minh họa trong hình vẽ sau đây:





HÌNH 20-1 Hiển thị dữ liệu XML trong bảng HTML.

Đoạn mã trong Chương 17 sử dụng các phương thức XML để lấy dữ liệu trực tiếp. Bài tập tiếp theo sẽ thay đổi đoạn mã đó để truy xuất file XML bằng cách sử dụng đối tượng *XMLHttpRequest*. Cũng giống như bài tập trong Chương 19, bài tập này yêu cầu file XML phải được lưu trên web server.

### Sử dụng *XMLHttpRequest* để truy xuất và hiển thị dữ liệu XML

1. Dùng file books.xml trong thư mục mã nguồn mẫu Chương 17 hoặc tạo một file books.xml chứa đoạn mã sau đây (đoạn mã có trong file books.xml của Tài nguyên đi kèm). Đặt file books.xml trên cùng web server với file HTML mà bạn sẽ tạo trong bước tiếp theo.

```
<books>
<book>
 <title>JavaScript - Hướng dẫn học qua ví dụ</title>
 <author>Steve Suehring</author>
 <isbn>9780735624498</isbn>
 <publisher>Microsoft Press</publisher>
</book>
<book>
 <title>MySQL Bible</title>
 <author>Steve Suehring</author>
 <isbn>9780764549328</isbn>
 <publisher>Wiley Publishing Inc.</publisher>
</book>
</books>
```

2. Sử dụng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác chỉnh sửa file ajaxbooks.htm trong thư mục mã nguồn mẫu Chương 20 của Tài nguyên đi kèm.
3. Trong trang ajaxbooks.htm, thay thế dòng chú thích TODO bằng đoạn mã in đậm sau đây (đoạn mã có trong file ajaxbooks.txt của Tài nguyên đi kèm). Hãy nhớ thay thế dòng URL *YOUR SERVER HERE* bằng URL tương ứng với web server của bạn. Lưu ý bạn chỉ thay đổi định nghĩa hàm và dòng đầu tiên của hàm *displayData()* so với mã gốc ở Chương 18.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Books</title>
</head>
<body>
<div id="xmldata"></div>
<script type="text/javascript">
function readyAJAX() {
 try {
 return new XMLHttpRequest();
```

```
 } catch(e) {
 try {
 return new ActiveXObject('Msxml2.')
 } catch(e) {
 try {
 return new ActiveXObject('
 } catch(e) {
 return "Bạn cần sử dụng m
 }
 }
 }
 }

var requestObj = readyAJAX();**
var url = "http://YOUR SERVER HERE/books.xml";
requestObj.open("GET",url,true);
requestObj.send();
var AJAXresponse;
requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {
 AJAXresponse = requestObj.responseText;
 displayData(AJAXresponse);
 } else {
 alert(requestObj.statusText);
 }
 }
}

function displayData(response) {
 var xmlEl = response.getElementsByTagName("book");
 var table = document.createElement("table");
 table.border = "1";
 var tbody = document.createElement("tbody");
 // Thêm phần body vào bảng
 table.appendChild(tbody);
 var row = document.createElement("tr");
 // Thêm một dòng vào bảng
 tbody.appendChild(row);
 for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType != 1) {
 continue;
 }
 var tableHead = document.createElement("th");
 tableHead.innerHTML = xmlEl[0].childNodes[colHead].nodeValue;
 row.appendChild(tableHead);
 }
}
```

```

 var colName = document.createTextNode(
 xmlEl[0].childNodes[colHead].nodeValue);
 tableHead.appendChild(colName);
 row.appendChild(tableHead);
 }
 tbody.appendChild(row);
 // Tạo các dòng cho bảng
 for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các ô cho mỗi dòng/td
 for (j = 0; j < xmlEl[i].childNodes.length;
 // Bỏ qua phần tử có kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeType == 1)
 continue;
 }
 // Thêm dữ liệu từ tài liệu XML
 var td = document.createElement("td");
 var xmlData = document.createTextNode(
 xmlEl[i].childNodes[j].nodeValue);
 td.appendChild(xmlData);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}
document.getElementById("xmldata").appendChild(tbody);
}
</script>
</body>
</html>

```

4. Dùng trình duyệt để mở, bạn sẽ thấy một trang như sau:

A screenshot of Microsoft Internet Explorer version 6.0. The window title is "Books - Microsoft Internet Explorer". The address bar shows the URL "http://localhost/Code/Chuong20/ajaxbooks.html". The main content area displays a table with four columns: title, author, isbn, and publisher. The data rows are: "MySQL Bible" by Steve Suehring, ISBN 9780764549328, published by Wiley Publishing Inc.; and "JavaScript - Hướng dẫn học qua ví dụ" by Steve Suehring, ISBN 9780735624498, published by Microsoft Press.

| title                                | author         | isbn          | publisher             |
|--------------------------------------|----------------|---------------|-----------------------|
| MySQL Bible                          | Steve Suehring | 9780764549328 | Wiley Publishing Inc. |
| JavaScript - Hướng dẫn học qua ví dụ | Steve Suehring | 9780735624498 | Microsoft Press       |

Bài tập này kết hợp đoạn mã từ hai bài tập trong Chương 18 và 19 để minh họa cách truy xuất, hiển thị dữ liệu XML bằng đối tượng `xmlHttpRequest` kết hợp với ứng dụng AJAX. Mặc dù ứng dụng XML ở Chương 18 đã được cải tiến để sử dụng đối tượng `xmlHttpRequest` song bảng kết quả hiển thị trông vẫn chưa thật đẹp. Đây là lúc vai trò của CSS được thể hiện.

## Sử dụng CSS để áp dụng style cho bảng

Hàm chủ đạo hiển thị bảng trong bài tập trước là `displayData()`. Trong hàm này, bạn có thể áp dụng các style CSS để bảng hiển thị đẹp hơn và giống bảng trong các ứng dụng web hiện đại hơn.

**Chú ý** Cách làm việc với CSS trong chương này là thay đổi trực tiếp các thuộc tính định dạng trong JavaScript. Tuy nhiên, bạn cần hiểu rằng đây chỉ là phương pháp phục vụ mục đích giảng dạy; phương pháp này không được ưa dùng trong thực tế vì nó làm cho việc gỡ lỗi các thuộc tính định dạng trở nên khó khăn do thông tin bị quy định trong cả mã CSS và mã JavaScript. Một phương pháp khác để làm việc với CSS sẽ được trình bày trong Chương 22 “Giới thiệu về jQuery”, trong đó việc thay đổi các style CSS được thực

hiện bằng cách chỉnh sửa style CSS áp dụng cho các phần tử HTML. Phương pháp này được ưa dùng hơn khi làm việc với CSS và JavaScript vì nó tách biệt phần hiển thị (CSS) với hành vi (mã JavaScript).

## Thay đổi các thuộc tính định dạng với JavaScript

Một trong những việc đầu tiên cần làm là loại bỏ dòng mã ở đầu hàm *displayData()* để xóa đường viền của bảng:

```
table.border = "1";
```

Trong hàm *displayData()* có hai vòng lặp chính: vòng lặp thứ nhất hiển thị các tiêu đề cột và vòng lặp thứ hai hiển thị dữ liệu của bảng. Vòng lặp hiển thị tiêu đề cột có dạng như sau:

```
for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType != 1) {
 continue;
 }
 var tableHead = document.createElement("th");
 var colName =
 document.createTextNode(xmlEl[0].childNodes[colHead].nodeValue);
 tableHead.appendChild(colName);
 row.appendChild(tableHead);
}
tbody.appendChild(row);
```

Vòng lặp thứ hai hiển thị dữ liệu trong bảng như sau:

```
for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các ô cho mỗi dòng/td
 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 // Bỏ qua dòng nếu kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeType != 1) {
 continue;
 }
 // Lấy dữ liệu từ tài liệu XML
 var td = document.createElement("td");
 td.appendChild(xmlEl[i].childNodes[j].nodeValue);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}
```

```

 var xmlData =
 document.createTextNode(xmlEl[i].childNodes[j]);
 td.appendChild(xmlData);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}

```

Những thay đổi về hiển thị dữ liệu chủ yếu được thực hiện trong hai vòng lặp này. Tôi sẽ in đậm những đoạn mã thay đổi.

Chẳng hạn bạn muốn thay đổi font chữ thành Arial thông qua việc sử dụng thuộc tính định dạng *fontFamily* trong JavaScript. Font chữ cần được thay đổi trong cả hai vòng lặp để toàn bộ văn bản trong bảng có cùng định dạng font chữ Arial. Sau khi thêm đoạn mã, hai vòng lặp sẽ như bên dưới (chú ý hai dòng mã mới được in đậm):

```

for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType != 1) {
 continue;
 }
 var tableHead = document.createElement("th");
 var colName =
 document.createTextNode(xmlEl[0].childNodes[colHead]);
 tableHead.style.fontFamily = "Arial";
 tableHead.appendChild(colName);
 row.appendChild(tableHead);
}
tbody.appendChild(row);
for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các ô cho mỗi dòng/td
 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 // Bỏ qua dòng nếu kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeType != 1) {
 continue;
 }
 // Lấy dữ liệu từ tài liệu XML
 var td = document.createElement("td");
 var xmlData =
 document.createTextNode(xmlEl[i].childNodes[j]);
 td.appendChild(xmlData);
 td.style.fontFamily = "Arial";
 row.appendChild(td);
 }
}

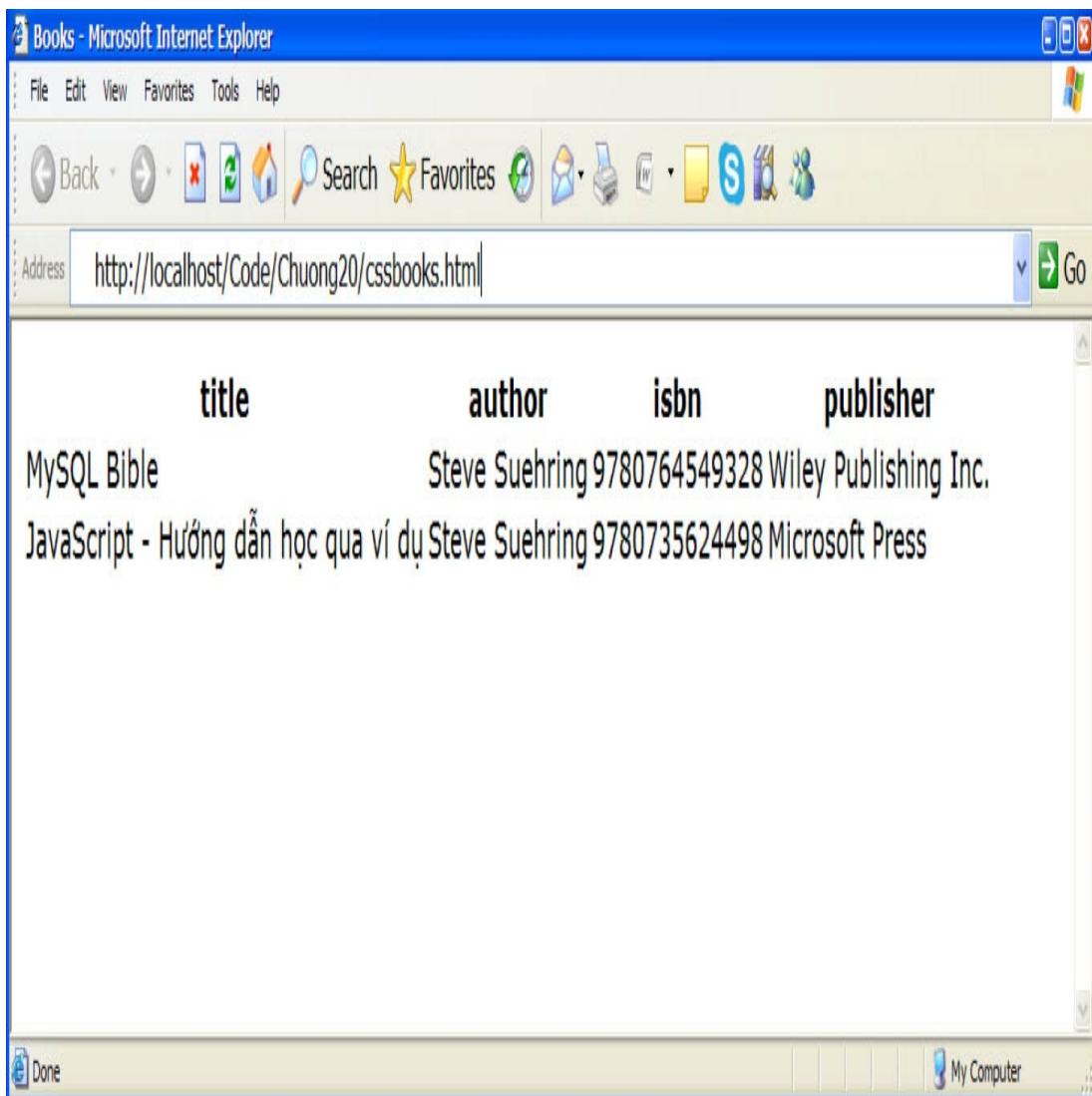
```

```

 td.appendChild(xmlData);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}

```

Kết quả của những thay đổi trên và việc loại bỏ đường viền của bảng sẽ tạo ra một bảng giống như trong Hình 20-2.



The screenshot shows a Microsoft Internet Explorer window titled "Books - Microsoft Internet Explorer". The address bar contains the URL "http://localhost/Code/Chuong20/cssbooks.htm". The page displays a table with the following data:

| <b>title</b>                         | <b>author</b>  | <b>isbn</b>   | <b>publisher</b>      |
|--------------------------------------|----------------|---------------|-----------------------|
| MySQL Bible                          | Steve Suehring | 9780764549328 | Wiley Publishing Inc. |
| JavaScript - Hướng dẫn học qua ví dụ | Steve Suehring | 9780735624498 | Microsoft Press       |

HÌNH 20-2 Bước đầu áp dụng style cho bảng với CSS.

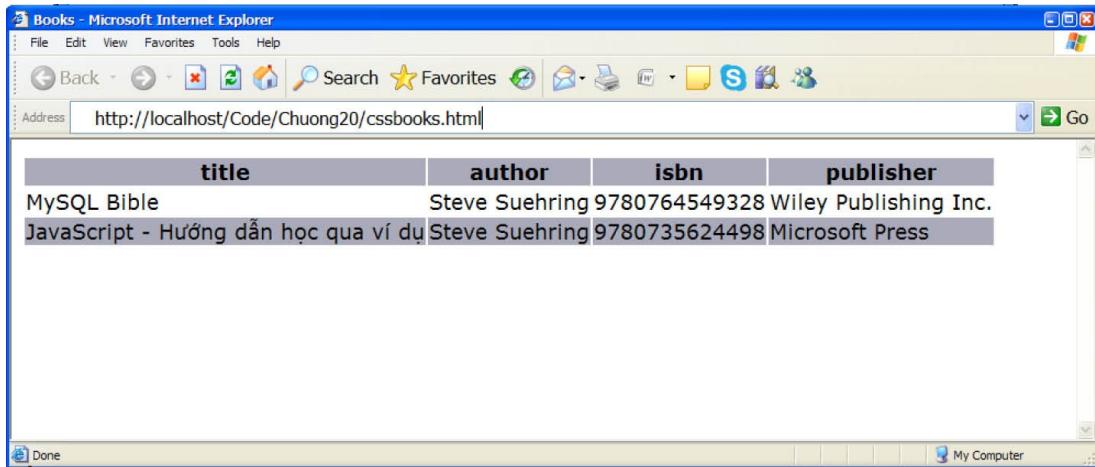
Thêm hiệu ứng màu sẽ làm cho bảng dễ đọc hơn, đặc biệt là khi có nhiều dòng dữ liệu được hiển thị. Thông thường các màu được bố trí xen kẽ trên mỗi hàng dữ liệu và có màu riêng cho phần tiêu đề. Dưới đây là hai vòng lặp sau khi đã thêm *màu nền* cho các dòng dữ liệu (chú ý, đoạn mã thêm vào được in đậm):

```

for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType != 1) {
 continue;
 }
 var tableHead = document.createElement("th");
 var colName =
 document.createTextNode(xmlEl[0].childNodes[colHead].nodeValue);
 tableHead.style.fontFamily = "Arial";
 tableHead.style.backgroundColor = "#aaabba";
 tableHead.appendChild(colName);
 row.appendChild(tableHead);
}
tbody.appendChild(row);
for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các ô cho mỗi dòng/td
 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 // Bỏ qua phần tử nếu có kiểu khác 1
 if (xmlEl[i].childNodes[j].nodeType != 1) {
 continue;
 }
 // Lấy dữ liệu từ tài liệu XML
 var td = document.createElement("td");
 var xmlData =
 document.createTextNode(xmlEl[i].childNodes[j].nodeValue);
 if (i % 2) {
 td.style.backgroundColor = "#aaabba";
 }
 td.style.fontFamily = "Arial";
 td.appendChild(xmlData);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}

```

Đoạn mã trên sử dụng toán tử mô-đun (%) để tô màu xám nhạt cho các dòng dữ liệu ở vị trí chẵn trong bảng. Vì chỉ có hai dòng dữ liệu nên dòng thứ hai được tô màu. Hình 20-3 hiển thị kết quả sau khi thêm màu nền cho bảng. Bạn có thể tìm thấy phiên bản hoàn thiện của trang web này trong thư mục CompleteCode Chương 20 của Tài nguyên đi kèm.



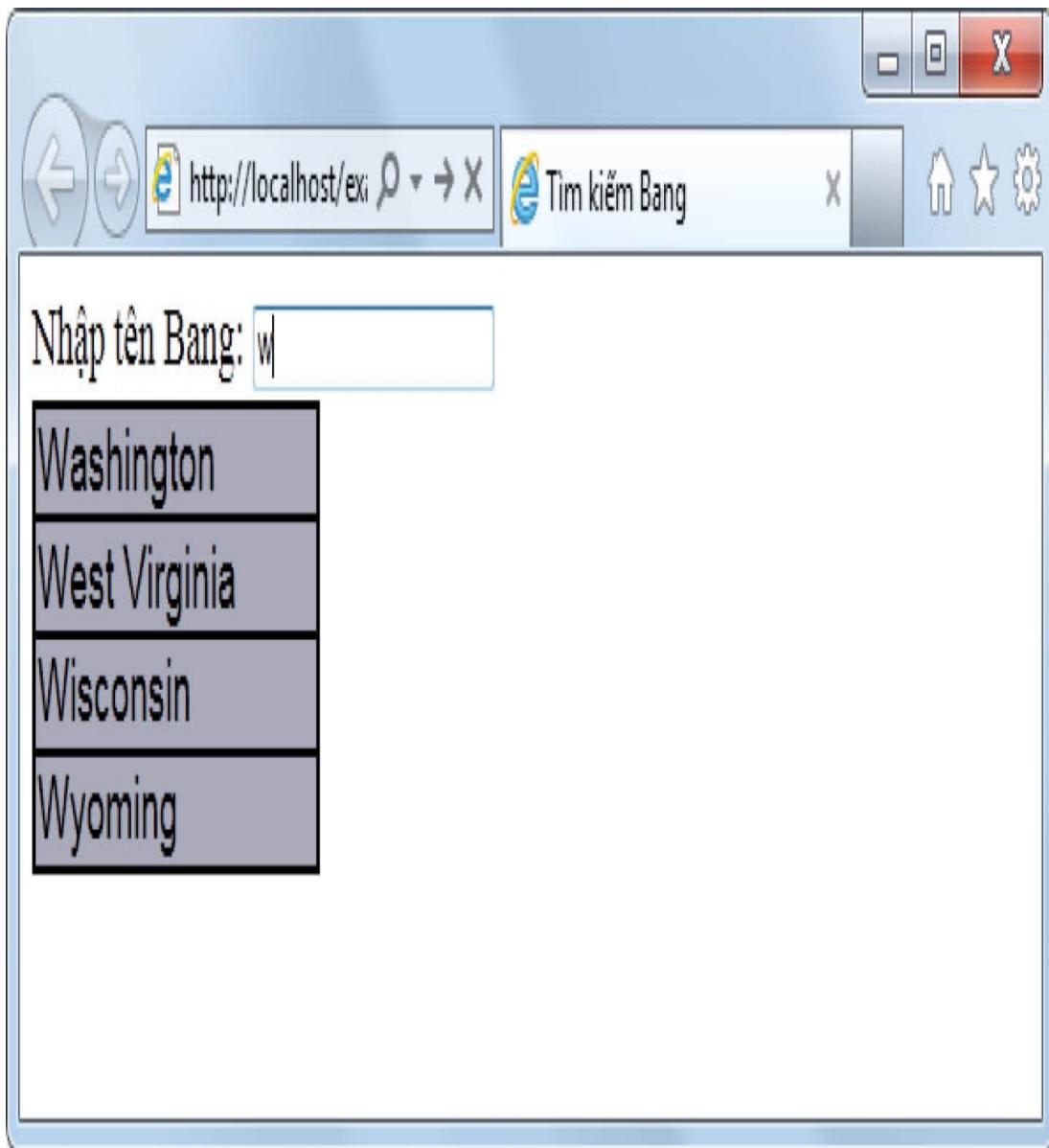
HÌNH 20-3 Thêm màu vào bảng sử dụng CSS.

## Tạo ô thả xuống động

Bạn có thể chỉnh sửa lại ứng dụng đánh dấu trang trong Chương 19 để tạo ra một ô thả xuống chứa dữ liệu dạng danh sách văn bản. Đôi khi tính năng này còn được biết đến với tên gọi “ô gợi ý”, vì khi bạn nhập dữ liệu vào ô, giao diện sẽ đưa ra một danh sách các từ gợi ý tương ứng với từ được nhập, giúp người dùng nhập liệu dễ dàng hơn. Google Suggest là một ứng dụng như vậy.

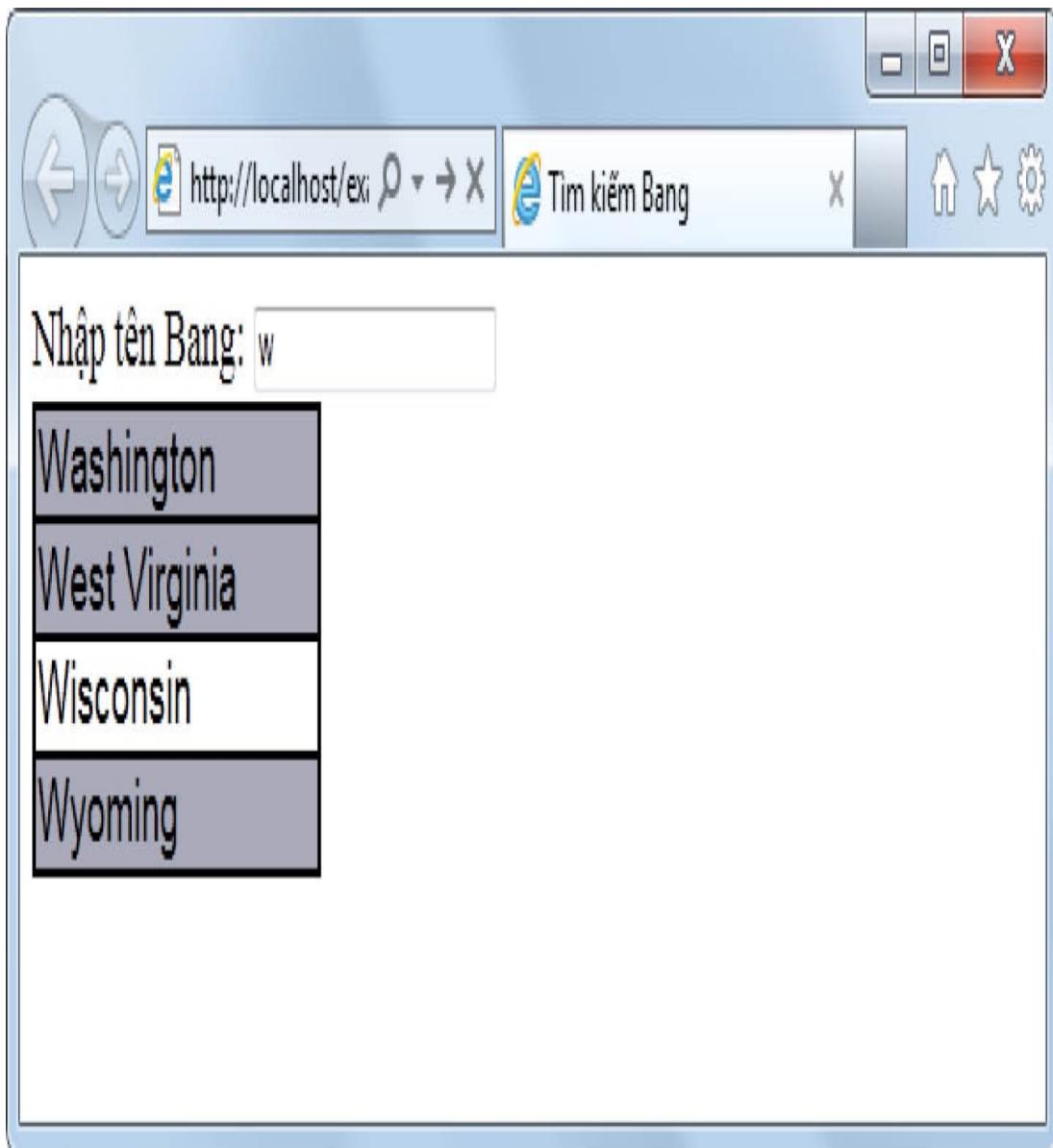
Một ví dụ áp dụng nguyên lý này đó là ô thả xuống hiển thị các nội dung khi người dùng nhập liệu (ví dụ như danh sách các Bang của Mỹ). Khác biệt lớn nhất trong ứng dụng này là tập dữ liệu truy xuất nhanh từ ô thả xuống được quản lý. Chẳng hạn, với tập dữ liệu là danh sách 50 bang của Mỹ, người dùng có thể dễ dàng và nhanh chóng truy xuất ra một bang trong đó. Ngược lại, nếu người dùng làm việc với tập dữ liệu trên 1.000.000 bản ghi thì việc truy xuất có thể rất lâu và không đáp ứng được yêu cầu. Một ví dụ khác là bạn có thể sử dụng ứng dụng này trong công việc để truy xuất tới danh sách tên nhân viên của một công ty.

Dưới đây là minh họa cho ứng dụng này. Sử dụng đối tượng `xmlHttpRequest` để truy xuất tới danh sách 50 bang của Mỹ. Khi người dùng nhập ký tự `w`, ứng dụng sẽ truy xuất tới tất cả các bang có tên bắt đầu bằng ký tự `w`, giống như trong Hình 20-4.



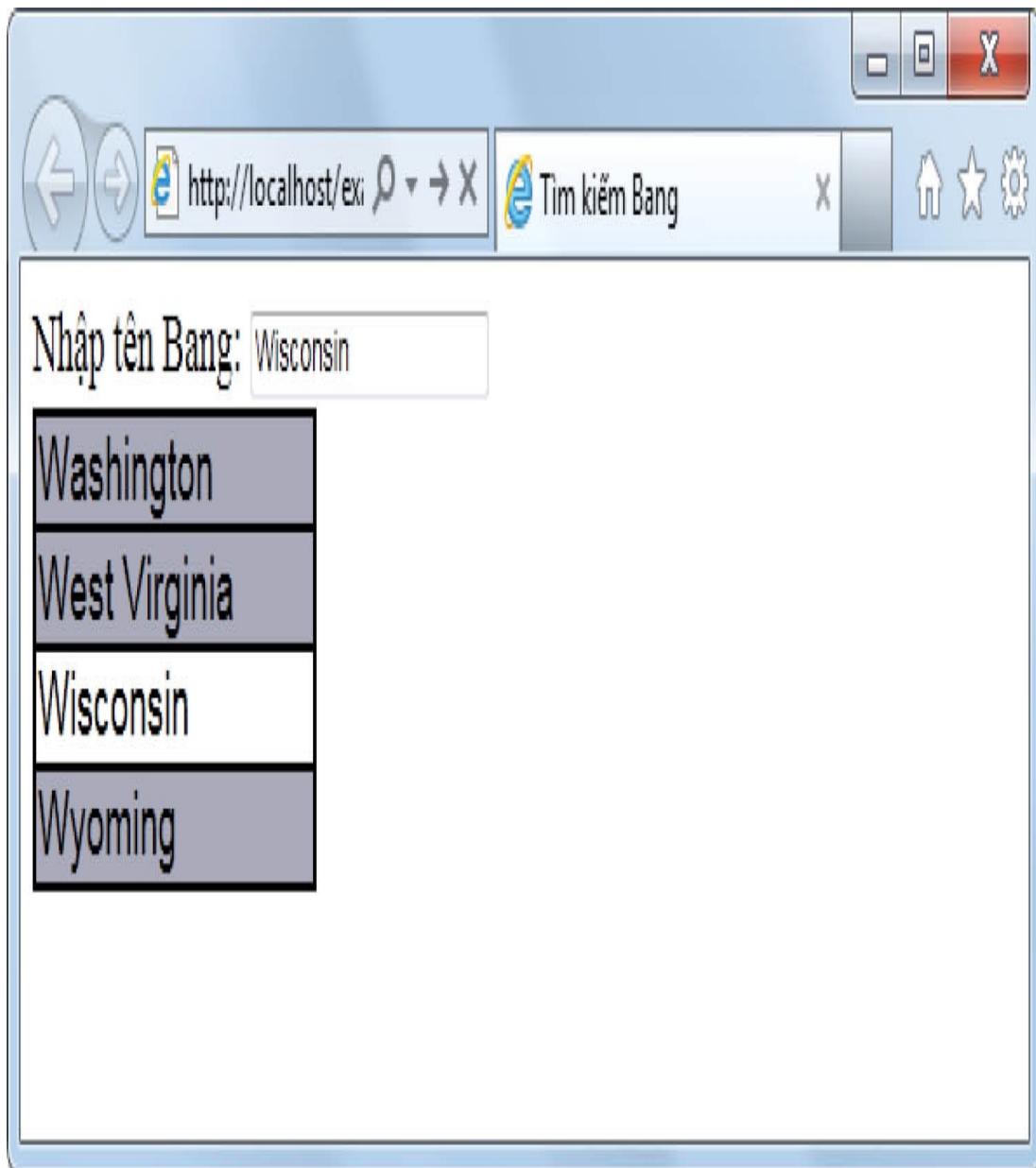
HÌNH 20-4 Truy xuất tới danh sách các bang có tên bắt đầu bằng ký tự `w`.

Màu nền của các phần tử trong danh sách sẽ thay đổi khi ta di chuột lên các bang khác nhau, giống như trong Hình 20-5, ở đây tôi đang di chuột lên bang Wisconsin (con trỏ chuột không hiển thị trong ảnh chụp màn hình).



HÌNH 20-5 Màu nền của các phần tử trong danh sách sẽ thay đổi khi di chuyển chuột lên các bang khác nhau.

Cuối cùng, khi nhấn chuột vào một bang bất kỳ, tên của bang đó sẽ được sao chép vào text box. Kết quả cuối cùng của ứng dụng được biểu diễn trong Hình 20-6. Từ đây, form có thể được gửi đi và xử lý dựa trên dữ liệu nhập vào.



HÌNH 20-6 Sao chép tên bang vào text box.

Đoạn mã này hoạt động tương tự như đoạn mã trong ứng dụng đánh dấu trang ở Chương 19, cung cấp tính năng cho phép người dùng nhập từ khóa và hiển thị một danh sách thu hẹp để lựa chọn. Hãy cùng xem xét trường hợp khi người dùng nhập ký tự **n**. Khi đó, text box sẽ hiển thị 8 bang trong danh sách thả xuống. Nếu tiếp tục nhập thêm ký tự mới – ví dụ nhập từ **new** – danh sách thả xuống sẽ thu hẹp chỉ hiển thị 4 bang và nếu tiếp tục nhập thêm các ký tự khác, danh sách thả xuống sẽ thu hẹp hơn nữa.

Đoạn mã cho ứng dụng này được trình bày trong Ví dụ 20-1.

## VÍ DỤ 20-1 Ứng dụng tìm kiếm.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Tìm kiếm Bang</title>
<script type="text/javascript" data-src="ehandler.js"></sc
</head>
</body>
<form name="nameform" id="nameform" action="" method="post
Nhập tên Bang: <input id="textname" type="text" name="text
</form>
<div id="data"></div>
<script type="text/javascript">
function textSearch() {
 var textName = document.getElementById("textname");
 var dataNode = document.getElementById("data");
 while (dataNode.hasChildNodes()) {
 dataNode.removeChild(dataNode.firstChild);
 }
 if (textName.value != "") {
 listName(textName.value);
 }
}
function readyAJAX() {
 try {
 return new XMLHttpRequest();
 } catch(e) {
 try {
 return new ActiveXObject('Msxml2.X
 } catch(e) {
 try {
 return new ActiveXObject('
 } catch(e) {
 return "Bạn cần sử
 }
 }
 }
}
function listName(text) {
 var nameList = AJAXresponse.split(",");
 var dataNode = document.getElementById("data");
 while (dataNode.hasChildNodes()) {
 dataNode.removeChild(dataNode.firstChild);
 }
 for (var i = 0; i < nameList.length; i++) {
 var item = document.createElement("div");
 item.innerHTML = nameList[i];
 dataNode.appendChild(item);
 }
}
```

```
var pattern = "^" + text;
var nameRegexp = new RegExp(pattern, "i");
for (var i = 0; i < nameList.length; i++) {
 var existDiv = document.getElementById(nameList[i]);
 if (!existDiv) {
 if (nameList[i].match(nameRegexp)) {
 var displayDiv = document.createElement("div");
 var newDiv = document.createElement("div");
 if (window.attachEvent) {
 newDiv.attachEvent("onload", function() {
 document.getElementById("list").appendChild(displayDiv);
 displayDiv.appendChild(newDiv);
 });
 newDiv.attachEvent("onmouseover", function(e) {
 e.srcElement.style.backgroundColor = "#000";
 e.srcElement.style.color = "#fff";
 });
 newDiv.attachEvent("onmouseout", function(e) {
 e.srcElement.style.backgroundColor = "#fff";
 e.srcElement.style.color = "#000";
 });
 } else {
 newDiv.addEventListener("load", function() {
 document.getElementById("list").appendChild(displayDiv);
 displayDiv.appendChild(newDiv);
 }, false);
 newDiv.addEventListener("mouseover", function(e) {
 this.style.backgroundColor = "#000";
 this.style.color = "#fff";
 }, false);
 newDiv.addEventListener("mouseout", function(e) {
 this.style.backgroundColor = "#fff";
 this.style.color = "#000";
 }, false);
 }
 newDiv.setAttribute("id", nameList[i]);
 newDiv.style.backgroundColor = "#fff";
 newDiv.style.color = "#000";
 newDiv.style.border = "solid 1px #ccc";
 newDiv.style.display = "block";
 newDiv.style.width = "175px";
 newDiv.appendChild(document.createTextNode(text));
 displayDiv.appendChild(newDiv);
 }
 }
}
}

var requestObj = readyAJAX();
var url = "http://YOUR SERVER HERE/statelist.php";
requestObj.open("GET", url, true);
requestObj.send();
var AJAXresponse;
```

```

requestObj.onreadystatechange = function() {
 if (requestObj.readyState == 4) {
 if (requestObj.status == 200) {
 AJAXresponse = requestObj.responseText
 } else {
 alert(requestObj.statusText);
 }
 }
}
var textEl = document.getElementById("textname");
EHandler.add(textEl,"keyup", function() { textSearch(); })
</script>
</body>
</html>

```

Đoạn mã trên chủ yếu sử dụng lại những đoạn mã đã được giới thiệu trong cuốn sách này, tuy nhiên tôi sẽ vẫn giải thích ngắn gọn.

Đoạn mã truy xuất tới danh sách các bang bằng cách gọi tới file *statelist.php* phía server. File này trả về một danh sách các bang phân tách nhau bởi dấu phẩy như sau:

*Alabama,Alaska,Arizona,California,Colorado,Delaware,Florida,Georgia, . . .*



File *statelist.php* phân tách danh sách bang dựa vào dấu phẩy và đặt các bang vào một mảng có tên là *nameList* như sau:

```
var nameList = AJAXresponse.split(",");
```

So với ứng dụng trong hai chương trước (Chương 18 và 19), đoạn mã bổ sung các style CSS để tạo các ô thả xuống cho phép người dùng nhấn chuột để chọn. Phần mã bổ sung được đưa vào hàm *listName()* mà bạn đã thấy trong Chương 19. Hàm *listName* sử dụng hàm lắng nghe sự kiện và áp dụng các style CSS cho các thẻ HTML *DIV* bằng đoạn mã in đậm dưới đây:

```

function listName(text) {
 var nameList = AJAXresponse.split(",");
 var pattern = "^" + text;
 var nameRegexp = new RegExp(pattern, "i");
 for (var i = 0; i < nameList.length; i++) {
 var existDiv = document.getElementById(nameL
 if (! existDiv) {

```

```
 if (nameList[i].match(nameRegexp)) {
 var displayDiv = document.get
 var newDiv = document.create
 if (window.attachEvent)
 newDiv.attach
 docui
 newDiv.attach
 e.sr
 newDiv.attach
 e.sr
 } else {
 newDiv.addEv
 docui
 newDiv.addEv
 this
 newDiv.addEv
 this
 }
 newDiv.setAttribute(
 newDiv.style.backgrou
 newDiv.style.color =
 newDiv.style.border :
 newDiv.style.display
 newDiv.style.width =
 newDiv.appendChild(d
 displayDiv.appendChild
}
}
}
}
```



## Tiếp nhận dữ liệu đầu vào từ người dùng và AJAX

Bước tiếp theo trong quá trình phát triển ứng dụng AJAX là tiếp nhận và xử lý dữ liệu đầu vào. Nói tới ứng dụng AJAX là nói tới những ứng dụng cung cấp

một giao diện mang tính tương tác cao dựa trên các hành động của người dùng. Tuy nhiên, để tạo ra những ứng dụng đó, bạn cần tìm hiểu thêm về các ứng dụng xử lý dữ liệu đầu vào phía server trong khi nội dung đó lại nằm ngoài phạm vi của cuốn sách nhập môn này.

Hy vọng trong khuôn khổ của cuốn sách này, bạn có thể nhận thấy việc xây dựng ứng dụng AJAX không có gì hơn ngoài việc cung cấp những ứng dụng thân thiện và có tính tương tác cao và hầu hết công việc đều xoay quanh việc sử dụng JavaScript chứ không chỉ riêng đối tượng `XMLHttpRequest`. `XMLHttpRequest` đơn giản chỉ là phương tiện để lấy dữ liệu trong chương trình. Tầng hoạt động của `XMLHttpRequest` nằm bên dưới tầng giao diện tạo nên trang web. Do đó, người dùng không bao giờ nhìn thấy quá trình xử lý của đối tượng `XMLHttpRequest`, họ chỉ thấy những gì được hiển thị bởi ứng dụng.

Hai chương còn lại của cuốn sách này sẽ dựa trên những kiến thức và ví dụ mà bạn đã tìm hiểu cho tới thời điểm này, nhưng sẽ sử dụng một thư viện JavaScript mới, đó là thư viện jQuery.

## Bài tập



1. Tạo hàm xử lý sự kiện `submit` cho ví dụ hiển thị danh sách các bang để hiển thị bang được chọn khi người dùng gửi form.
2. Tạo ứng dụng sử dụng đối tượng `XMLHttpRequest`, trả về danh sách tên (ví dụ như danh sách nhân viên). Bạn có thể sử dụng file văn bản thường hoặc file XML để lưu dữ liệu.

# Phần 5

# jQuery

[Chương 21: Giới thiệu về các thư viện và Framework của JavaScript](#)

[Chương 22: Giới thiệu về jQuery](#)

[Chương 23: Các hiệu ứng và plug-in cho jQuery](#)

## Chương 21

# Giới thiệu về các thư viện và Framework của JavaScript.

Sau khi đọc xong chương này, bạn có thể:

- Hiểu vai trò của các thư viện và framework JavaScript.
- Biết cách tự tạo thư viện JavaScript.
- Hiểu vai trò của các thư viện và framework JavaScript bên thứ ba và cách tìm kiếm thêm thông tin về chúng.

## Tìm hiểu các thư viện lập trình

Trong thuật ngữ lập trình, *thư viện (library)* là một tập hợp các đoạn mã cung cấp những tính năng thông dụng hoặc tính năng bổ sung cho ứng dụng. Thông thường, thư viện chứa một hoặc nhiều file trình bày những đối tượng và chức năng nhất định. Trong một ứng dụng, người lập trình tích hợp hoặc gọi tới thư viện để sử dụng các đối tượng và chức năng này. Như vậy, các thư viện và

framework JavaScript giúp bảo trì và phát triển các tính năng nâng cao hoặc bổ sung. Nhờ đó, những khó khăn do môi trường phát triển đa trình duyệt cũng được giảm bớt.

Trọng tâm của chương này là các thư viện JavaScript, bao gồm việc tự định nghĩa thư viện JavaScript và tìm hiểu một số thư viện và framework JavaScript phổ biến.

# Tự định nghĩa thư viện JavaScript

Dù làm việc với bất cứ ngôn ngữ nào, các lập trình viên cũng rơi vào tình huống phải viết đi viết lại một tính năng nào đó, do vậy họ tạo ra những thư viện cá nhân lưu những tính năng hữu ích cho các dự án trong tương lai.

Đoạn mã xử lý sự kiện ở Chương 11, ehandler.js, hướng dẫn cách tạo thư viện bằng cách tạo namespace.

```
var EHandler = {};
```



Trong namespace, thư viện *EHandler* bổ sung hai hàm. Đầu tiên là:

```
EHandler.add = ...
```

Thứ hai là:

```
EHandler.remove = ...
```

Mặc dù ngắn nhưng *EHandler* là một thư viện hoàn chỉnh. Như vậy, thư viện không nhất thiết phải lớn mới hữu ích. Bạn đã thấy rất nhiều ví dụ sử dụng *EHandler* trong phần đầu của cuốn sách này. Với ý niệm về thư viện *Ehandler* sẵn có trong đầu, trong ví dụ tiếp theo, bạn sẽ tự tạo một thư viện JavaScript.

## Tạo thư viện

1. Dùng Microsoft Visual Studio, Eclipse hoặc trình soạn thảo khác mở file library.js trong thư mục mã nguồn mẫu Chương 21 thuộc phần Tài nguyên đi kèm.
2. Trong file library.js, thêm vào đoạn mã sau (thay thế cho dòng chú thích

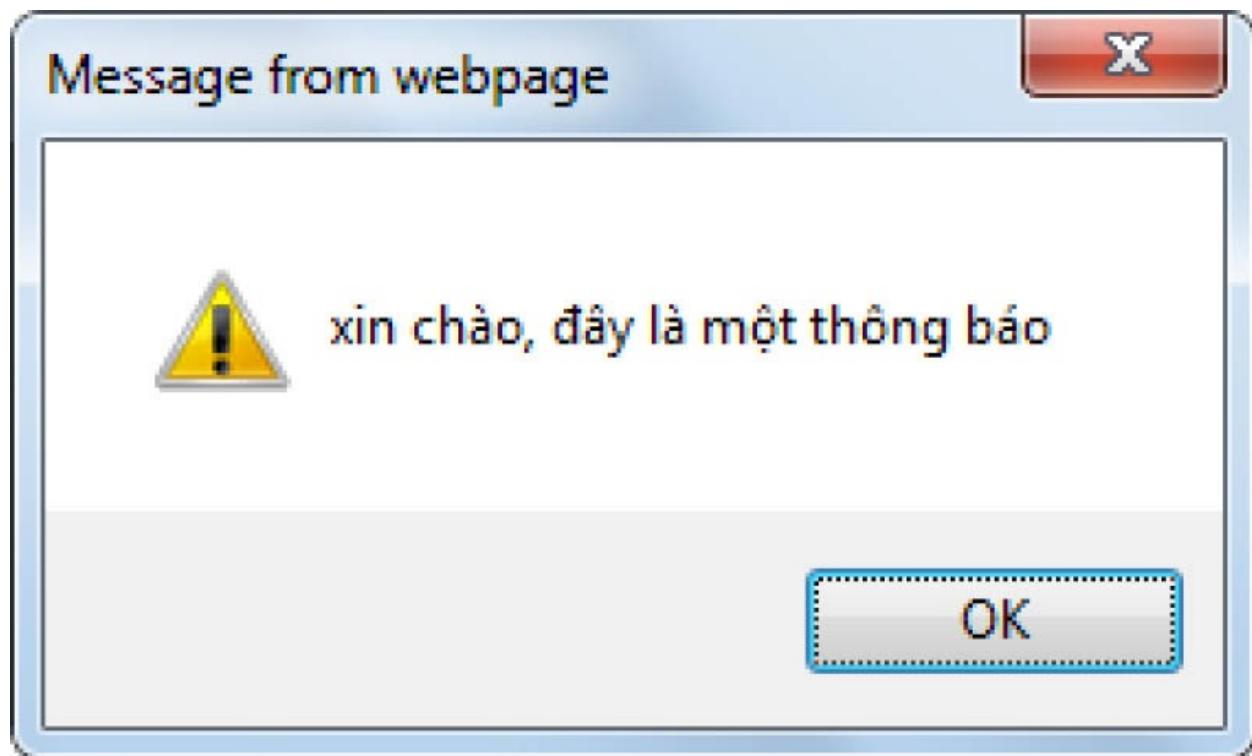
TODO) để tạo namespace và sau đó tạo một hàm:

```
var MyLibrary = {};
MyLibrary.sendAlert = function(mesg, elm) {
 alert(mesg);
};
```

3. Lưu và đóng file.
4. Mở file librarypage.htm. Trong file librarypage.htm, thêm dòng mã in đậm dưới đây (thay thế cho dòng chú thích TODO):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ đơn giản</title>
<script type="text/javascript" data-src="library.js"></sc
</head>
<body>
<script type="text/javascript">
MyLibrary.sendAlert("xin chào, đây là một thông báo");
</script>
</body>
</html>
```

5. Dùng trình duyệt mở trang librarypage.htm. Bạn sẽ thấy một hộp thoại thông báo như sau:



**Xử lý sự cố** Nếu không thấy hộp thoại thông báo như trên, hãy kiểm tra lại đường dẫn tới file library.js. Trong ví dụ về trang librarypage.htm trước đó giả định rằng file library.js nằm trong cùng thư mục với file HTML.

Khi tạo và sử dụng các thư viện, hãy lưu ý tránh xung đột hoặc trùng lặp với các hàm đã tồn tại và từ khóa của chuẩn ECMA-262. Ngoài ra, nếu bạn dùng một thư viện ngoài như jQuery hoặc YUI, cần đảm bảo thư viện bạn tạo không xung đột với quy ước đặt tên của các thư viện đó.

## Sơ lược về các thư viện và framework JavaScript phổ biến

Có nhiều thư viện và framework dùng chung phục vụ cho việc lập trình JavaScript. Mục đích của chúng là xử lý những tác vụ khó và giúp lập trình viên phát triển ứng dụng web JavaScript dễ dàng hơn.

Các lập trình viên web thường mất nhiều thời gian để làm cho trang web có giao diện và vận hành như nhau trên các loại trình duyệt. Một ưu điểm quan trọng khi sử dụng các thư viện và framework JavaScript là chúng giải quyết những vấn đề đau đầu liên quan đến sự tương thích giữa các trình duyệt. Tất cả các thư viện và framework JavaScript phổ biến đều hỗ trợ để các tính năng của riêng chúng có thể chạy trên nhiều trình duyệt khác nhau.

## jQuery

jQuery có danh sách tính năng phong phú với nhiều tùy chọn mạnh, khả năng mở rộng và sự hỗ trợ tuyệt hảo từ cộng đồng người dùng. jQuery được đóng gói hoàn chỉnh trong một file JavaScript. Với jQuery, bạn có thể tạo hiệu ứng cho trang web, tăng cường tính khả dụng và giúp thao tác với dữ liệu bằng AJAX trở nên dễ dàng hơn. Một điểm cộng nữa là jQuery đi kèm cùng bộ Visual Studio 2010 của Microsoft. Chương 22 “Giới thiệu về jQuery” và Chương 23 “Các hiệu ứng và plugin cho jQuery” sẽ giới thiệu chi tiết hơn về jQuery. Bạn có thể tìm hiểu thêm thông tin về jQuery tại địa chỉ <http://jquery.com>.



## Yahoo! User Interface (Giao diện người dùng Yahoo!)

Giao diện người dùng Yahoo! (YUI) cung cấp cả JavaScript và CSS, giúp đơn giản hóa công việc phát triển ứng dụng web. Giống như jQuery, YUI có những tính năng giúp nâng cao tính khả dụng và cải thiện ứng dụng web. Một ưu điểm nữa là tài liệu đặc tả của YUI rất đầy đủ. Bạn có thể tìm hiểu thêm về YUI tại địa chỉ <http://developer.yahoo.com/yui/>.

## MooTools

MooTools là một thư viện rất nhỏ nhưng hết sức tối ưu cho JavaScript. MooTools khác YUI và jQuery ở chỗ nó tập trung vào việc tối ưu quá trình xử lý JavaScript, trong khi YUI và jQuery tập trung vào các hiệu ứng, CSS và các tương tác trực tiếp với người dùng. Dĩ nhiên, như thế không có nghĩa là MooTools không có các hiệu ứng – MooTools cũng cung cấp nhiều hiệu ứng như accordion menu (menu có khả năng mở rộng hoặc thu gọn các mục menu con) và thanh trượt slider tương tự như YUI và jQuery. MooTools được khuyên

dùng cho những ứng dụng JavaScript bậc trung đến cao cấp; Bạn có thể tham khảo thêm thông tin chi tiết về MooTools tại địa chỉ <http://mootools.net/>.

## Các thư viện khác

Có nhiều thư viện và framework khác cho JavaScript – số lượng lớn đến độ không thể trình bày hay nhắc đến tất cả trong cuốn sách này. Bạn có thể ghé thăm trang [http://en.wikipedia.org/wiki/Comparisonof\\_JavaScriptframeworks](http://en.wikipedia.org/wiki/Comparisonof_JavaScriptframeworks) để tìm hiểu thêm về các framework JavaScript.

## Bài tập

1. Nghiên cứu từng thư viện và framework có trong chương này. Theo bạn, thư viện hay framework nào dễ học nhất đối với người mới lập trình JavaScript? Tại sao?
2. Tạo một thư viện JavaScript với file JavaScript ngoài. Tham chiếu file đó trong một trang HTML và gọi tới file đó.

# Chương 22

## Giới thiệu về jQuery

Sau khi đọc xong chương này, bạn có thể:

- Nắm được cách liên kết tới jQuery trong HTML.
- Nắm rõ các khái niệm và cú pháp quan trọng của jQuery.
- Sử dụng jQuery trong trang web.

## Thông tin cơ bản về jQuery

jQuery là một framework JavaScript rất được ưa dùng và dễ sử dụng. jQuery giúp việc lập trình JavaScript trở nên dễ dàng hơn vì nó giúp gỡ bỏ rào cản không tương thích JavaScript trên các trình duyệt khác nhau.

Toàn bộ thư viện jQuery được gói gọn trong một file JavaScript, điều này giúp đơn giản hóa việc đưa jQuery vào mã JavaScript. Cú pháp jQuery cũng rất dễ hiểu, jQuery sử dụng một namespace đơn giản và tính năng thống nhất. Khi được kết hợp với tiện ích jQuery User Interface (UI) (sẽ được đề cập trong Chương 23 “Các hiệu ứng và plug-in cho jQuery”), jQuery giúp bạn tạo ra những ứng dụng web mạnh, có tính tương tác cao.

Chương này sẽ giới thiệu những tính năng cơ bản nhất của jQuery, trong đó có việc tải và sử dụng jQuery trong JavaScript.

## Sử dụng jQuery

Bạn có thể tải về jQuery tại địa chỉ <http://www.jquery.com/>. Trong phần này, bạn sẽ tìm hiểu cách tải và tích hợp jQuery vào trang web.

## Hai phiên bản jQuery

Trên trang chủ của jQuery, có hai phiên bản jQuery: một phiên bản sản phẩm và một phiên bản phát triển. Bạn nên tải và sử dụng phiên bản sản phẩm nếu không có ý định phát triển các plug-in cho jQuery hoặc muốn tìm hiểu sâu hơn về phần lõi của jQuery.

Ngoài ra, còn có một lựa chọn khác, đặc biệt là để thực hiện các bài thực hành trong chương này, đó là sử dụng mạng phân phối nội dung (Content Delivery Network - CDN) để truy cập tới phiên bản jQuery được lưu trữ trên server. Google lưu trữ jQuery và các thư viện khác thông qua trang website API của hãng. Điều này cho phép bạn sử dụng jQuery trong trang web và sử dụng các chương trình JavaScript mà không cần có file thư viện jQuery trên server cục bộ. Xem thêm thông tin tại địa chỉ <http://code.google.com/apis/libraries/devguide.html>.

**Chú ý** Nếu mới bắt đầu làm việc với jQuery, tôi khuyên bạn hãy tải và lưu trữ file thư viện jQuery trên máy cục bộ. Sử dụng phiên bản cục bộ giúp ứng dụng web chạy nhanh và tin cậy hơn so với việc sử dụng phiên bản CDN. Ví dụ, nếu sử dụng phiên bản CDN và kết nối mạng bị mất, khi đó các ứng dụng sử dụng thư viện CDN sẽ không thể hoạt động. Tuy nhiên, với các ví dụ và bài tập trong chương này, việc sử dụng thư viện CDN là hoàn toàn phù hợp.

Để hoàn thành các bài tập trong chương này đòi hỏi bạn phải có một thư viện jQuery trên máy tính cục bộ hoặc phải kết nối tới một thư viện jQuery trên website CDN.

## Liên kết với jQuery

Cách liên kết jQuery vào trang web cũng giống như cách liên kết các file JavaScript ngoài khác - đó là sử dụng thẻ `<script>` để trỏ tới file nguồn. Hãy xét đoạn mã trong Ví Dụ 22-1.

**VÍ DỤ 22-1** Liên kết jQuery vào trang web.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
```

```

"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Tích hợp jQuery</title>
<script type="text/javascript" data-src="jquery-1.4.3.min.
</head>
<body>
</body>
</html>

```

Sau khi, bạn đã tải phiên bản thư viện jQuery về máy tính hoặc tham chiếu tới nó trên website CDN và nắm được cách liên kết file thư viện đó vào trong trang web thông qua ví dụ trên, giờ là lúc để tìm hiểu các cú pháp của jQuery.

**Quan trọng** Phiên bản 1.4.3 là phiên bản jQuery mới nhất tại thời điểm viết cuốn sách. Tuy nhiên, phiên bản này có thể sẽ khác tại thời điểm cuốn sách đến với bạn, do đó bạn cần thay đổi lại thuộc tính *src* cho tương ứng với phiên bản jQuery được tải về.



## Cú pháp jQuery cơ bản

Khi đã liên kết thư viện jQuery vào trang web, jQuery sẽ thêm vào hàm *jquery()*. Bạn có thể hình dung đơn giản rằng việc gọi tới các hàm jQuery được thực hiện thông qua giao diện hàm *jquery()*, nhưng có một cách viết ngắn hơn để gọi hàm *jquery()*, đó là: *\$()*. Thay vì phải nhập **jquery**, bạn chỉ cần sử dụng ký tự đô-la theo sau là cặp dấu ngoặc đơn giống như các ví dụ trong Bảng 22-1.

**BẢNG 22-1** Một số bộ chọn jQuery

| Cú pháp                       | Mô tả                                                                                                                             |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <code>\$("a")</code>          | Chọn tất cả các phần tử <i>a</i> trong tài liệu.                                                                                  |
| <code>\$(document)</code>     | Chọn toàn bộ tài liệu, thường được dùng để truy cập tới hàm <i>ready()</i> , hàm này sẽ được trình bày trong phần sau của chương. |
| <code>\$("#elementID")</code> | Chọn phần tử có thuộc tính ID bằng <i>elementID</i> .                                                                             |
| <code>\$(".className")</code> | Chọn phần tử hoặc các phần tử có thuộc tính class bằng <i>className</i> .                                                         |

Bạn sẽ tìm hiểu thêm về các bộ chọn và các hàm liên quan trong phần sau của chương này.

Giống như mã JavaScript, câu lệnh jQuery được kết thúc bằng dấu chấm phẩy (;). Một điểm cần lưu ý nữa đó là bạn có thể sử dụng dấu nháy kép hoặc nháy đơn để đánh dấu các bộ chọn trong jQuery. Ví dụ, hai câu lệnh sau là tương đương nhau:

```
$("a")
$('a')
```

Trên thực tế, dấu nháy đơn và nháy kép thường được sử dụng thay thế cho nhau. Các ví dụ trong chương này sẽ sử dụng kết hợp cả hai ký hiệu để giúp bạn làm quen với cách viết lệnh jQuery; tuy nhiên, bạn nên chọn cho mình một cách và tuân thủ theo cách viết đó.

## Kết nối jQuery tới sự kiện Load

Một trong những cách phổ biến nhất để làm việc với jQuery đó là kết nối các phần tử trong sự kiện *load* (hoặc *onload*) của trang web. (Các sự kiện và hàm sẽ được đề cập chi tiết trong phần cuối chương). Trong jQuery, bạn thực hiện điều này thông qua hàm phụ trợ `.ready()` của phần tử *document*.

Hãy nhớ lại ví dụ ở phần trước, jQuery truy cập các phần tử thông qua cú pháp `$( )`. Hãy nhớ rằng bạn có thể truy cập tới phần tử *document* bằng cú pháp:

```
$(document)
```

Do đó, hàm *ready()* được truy cập như sau:

```
$(document).ready()
```

Bài tập tiếp theo yêu cầu bạn tải và cài đặt file thư viện jQuery trên máy tính cá nhân hoặc sử dụng phiên bản CDN. Ví dụ này sử dụng phiên bản jQuery 1.4.3, tuy nhiên số hiệu phiên bản có thể khác tại thời điểm bạn làm bài tập này.

### Sử dụng hàm Ready của Document

1. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo khác chỉnh sửa lại file docready.html trong thư mục mã nguồn mẫu Chương 22

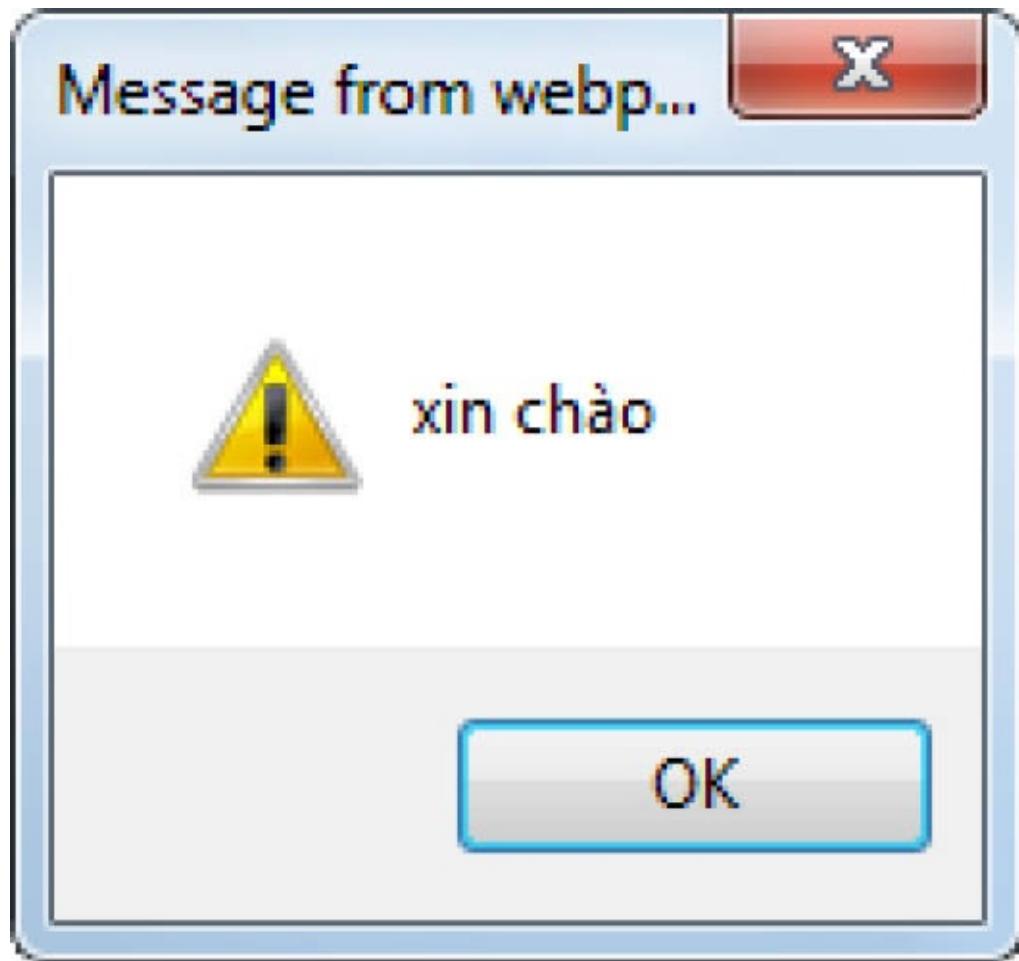
(Tài nguyên đi kèm).

- Trong file, thay thế dòng chú thích TODO bằng đoạn mã được in đậm dưới đây:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Document Ready</title>
<script type="text/javascript" data-src="jquery-1.4.3.min.js">
</head>
<body>
<script type="text/javascript">
$(document).ready(alert('xin chào'));
</script>
</body>
</html>
```

- Lưu và dùng trình duyệt mở trang web. Bạn sẽ thấy một hộp thoại thông báo giống như sau:





Đoạn mã trong bài tập này kết hợp mã jQuery (hàm `$(document).ready()`) với đoạn mã JavaScript thông thường, được đại diện bởi hàm `alert()`. Kết hợp jQuery và JavaScript là một khái niệm quan trọng bạn cần nắm rõ: sử dụng jQuery để bổ sung cho JavaScript. jQuery làm cho những công việc khó thực hiện với JavaScript trở nên dễ dàng hơn, nhờ đó bạn có thể dành thời gian xây dựng tính năng thay vì lo lắng về tính tương thích của chương trình khi chạy trên các trình duyệt khác nhau.

Với hàm `$(document).ready()`, bạn không cần sử dụng sự kiện `load` của trình duyệt hay thêm lời gọi hàm trong sự kiện `load`. Với hàm này, tất cả mọi phần tử của DOM đều được nạp trước khi gọi hàm `.ready()`.

**Mách nhỏ** Hàm `$(document).ready()` là hàm trọng tâm trong lập trình jQuery.

# Sử dụng các bộ chọn

Bộ chọn là chìa khóa để làm việc với jQuery và DOM. Bạn sử dụng bộ chọn để xác định và nhóm các phần tử, sau đó hàm jQuery sẽ làm việc với các tập hợp này. Trong Bảng 22-1, bạn sử dụng bộ chọn để thu thập tất cả các phần tử trong một thẻ, một ID hoặc một lớp CSS xác định. Bạn cũng có thể sử dụng bộ chọn theo những cách nâng cao, ví dụ chọn một số lượng xác định các phần tử hoặc chọn các phần tử ở cấp bậc cụ thể, ví dụ chọn những thẻ `<p>` theo sau thẻ `<div>`. Phần này sẽ giới thiệu chi tiết về các bộ chọn.

**Mách nhỏ** Các bộ chọn (*selector*) và cách thức hoạt động của chúng trong jQuery đều dựa trên các bộ chọn trong CSS. Nếu đã quen với bộ chọn trong CSS (Xem lại Chương 15 "JavaScript và CSS"), bạn sẽ thấy việc sử dụng bộ chọn trong jQuery cũng rất dễ dàng.

## Chọn phần tử theo ID

Ví dụ trong Bảng 22-1 giới thiệu cú pháp thông thường để chọn một phần tử dựa vào thuộc tính ID:

```
$("#elementID")
```

Ví dụ, hãy xét đoạn mã HTML sau:

```
Liên kết
```

Nếu sử dụng JavaScript, bạn sẽ truy cập phần tử này như sau:

```
getElementById("linkOne")
```

Còn với jQuery, việc truy cập tới phần tử được thực hiện qua cú pháp:

```
$("#linkOne")
```

## Chọn phần tử theo lớp

Chúng ta có thể chọn các phần tử thông qua lớp bằng cách thêm dấu chấm(.) vào trước tên của lớp đó. Cú pháp chọn sẽ như sau:

```
$(".className")
```

Ví dụ, dưới đây là một thẻ div được áp dụng thuộc tính class (lớp):

```
<div class="specialClass">
```

Phần tử này được truy cập thông qua jQuery như sau:

```
$(".specialClass")
```

Hãy nhớ rằng, theo cách này, bạn không thể chỉ truy cập một phần tử; bộ chọn theo lớp truy cập *tất cả* các phần tử thuộc lớp đó. Hay nói cách khác, nếu một số phần tử trong trang web có cùng thuộc tính class là “*specialClass*”, khi đó jQuery sẽ truy cập tới tất cả các phần tử đó thông qua bộ chọn `$(".specialClass")`. Bạn sẽ hiểu thêm về tính năng này trong các phần tiếp theo khi các hàm duyệt qua từng phần tử được truy xuất bởi bộ chọn theo lớp.

## Chọn phần tử theo kiểu phần tử

Bạn cũng có thể sử dụng bộ chọn để truy cập các phần tử theo kiểu, ví dụ như các phần tử `<div>`, `<a>`,... Ví dụ, bạn có thể truy cập tới các phần tử `<div>` trong tài liệu như sau:

```
$('div')
```

Tương tự, để truy cập các phần tử `<a>`, bạn có thể viết:

```
$('a')
```

Sử dụng bộ chọn theo kiểu cho phép truy cập tất cả các phần tử thuộc cùng kiểu trong trang web. Giống như bộ chọn theo lớp, bộ chọn theo kiểu có thể trả về nhiều phần tử cùng lúc.

## Chọn phần tử theo cấp bậc

Như đã đề cập từ trước, bạn có thể chọn các phần tử dựa vào vị trí của phần tử đó trong quan hệ với các phần tử khác trong trang. Ví dụ, để chọn tất cả các phần

tử `<a>` nằm bên trong phần tử `<div>`, bạn có thể sử dụng cú pháp sau:

```
$("div a")
```

Bạn còn có thể chọn các phần tử chi tiết hơn. Ví dụ, nếu muốn chọn các phần tử `<a>` nằm sau một thẻ `div` xác định, bạn có thể kết hợp bộ chọn theo kiểu cùng với cú pháp bộ chọn theo ID. Hãy xem xét đoạn mã HTML sau:

```
<div id="leftNav">
Liên kết 1
Liên kết 2
</div>
```

Sau đây là cú pháp bộ chọn trong jQuery để truy xuất tới hai phần tử `<a>` trong thẻ `div leftNav`.

```
$("#leftNav a")
```

Tổng quát hơn, nếu muốn chọn trực tiếp các phần tử “con cháu” của một phần tử xác định, bạn có thể sử dụng ký hiệu dấu lớn hơn (`>`), như sau:

```
$("div > p")
```



Cú pháp này chọn ra các phần tử `<p>` là con cháu trực tiếp của thẻ `div` nhưng không bao gồm các phần tử `<p>` khác nằm bên trong các phần tử `<p>` được chọn.

Bạn cũng có thể chọn phần tử con thứ *n* trong một tập hợp bằng bộ chọn `:nth-child()`. Ví dụ này chọn ra phần tử con thứ ba:

```
$("p:nth-child(3)")
```

*Ngoài ra còn có một số bộ chọn theo cấp bậc khác. Bạn có thể tìm hiểu thêm trong tài liệu tham khảo về bộ chọn trong jQuery tại địa chỉ <http://api.jquery.com/category/selectors/>.*

## Chọn phần tử theo vị trí

Như bạn đã thấy, các bộ chọn trong jQuery rất “tham lam”. Ví dụ, `$( 'a' )` sẽ chọn tất cả các thẻ `<a>`. jQuery cung cấp những cách thức cho phép tìm kiếm các phần tử cụ thể trong một tập hợp. Một trong những cách đó là sử dụng bộ chọn `first` và `last`. Đoạn mã dưới đây chọn ra phần tử `<p>` đầu tiên trong

trang web:

```
$("p:first")
```

Tương tự, phần tử `<p>` cuối cùng được chọn bằng cú pháp sau:

```
$("p:last")
```

Bạn cũng có thể chọn các phần tử trực tiếp theo vị trí. Ví dụ, hãy xét đoạn mã HTML sau:

```
<p>Phần tử p đầu tiên</p>
<p>Phần tử p thứ hai</p>
<p>Phần tử p thứ ba</p>
```

Để chọn phần tử `<p>` thứ hai, bạn sử dụng cú pháp:

```
$("p")[1]
```

Chú ý rằng mảng phần tử bắt đầu từ chỉ số 0, do đó phần tử đầu tiên của mảng có chỉ số bằng 0, phần tử thứ hai có chỉ số là 1,... Sử dụng cú pháp này khá nguy hiểm, vì việc chọn phụ thuộc vào vị trí của các phần tử trong cấu trúc trang web. Nếu thêm thẻ `<p>` vào trang web ngay trước phần tử bạn muốn chọn, khi đó chỉ số của phần tử đó sẽ thay đổi, dẫn tới việc chọn không còn chính xác. Nếu có thể, tốt hơn hết hãy sử dụng bộ chọn theo ID để xác định phần tử cụ thể thay vì dựa vào chỉ số của phần tử đó.

Một cách khác để chọn phần tử theo chỉ số đó là sử dụng cú pháp `:eq`. Ví dụ, để chọn đoạn văn bản thứ ba, bạn có thể viết:

```
$("p:eq(3)")
```

Cuối cùng, đôi khi việc sử dụng bộ chọn theo vị trí chẵn lẻ cũng rất hữu ích khi chọn các phần tử khác nhau trong một tập hợp:

```
$("p:even")
```

Các bộ chọn `even` (*chẵn*) và `odd` (*lẻ*) rất hữu ích khi làm việc với các dữ liệu dạng bảng để thay đổi màu sắc của các dòng. Ví dụ 22-2 minh họa cho việc sử dụng bộ chọn `odd` để tạo sự khác biệt về màu nền giữa các dòng chẵn lẻ cho một bảng.

**Chú ý** Đoạn mã trong Ví dụ 22-2 sử dụng hai phần tử chưa được giới thiệu chính thức: đó là một hàm người dùng tự định nghĩa và hàm `.css()`. Đừng chú ý tới chúng. Bạn sẽ tìm hiểu chi tiết từng hàm trong phần sau của chương.

## VÍ DỤ 22-2 Dữ liệu dạng bảng và jQuery.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Kiểm thử với Bảng</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.
</head>
<body>
<table>
 <tr>
 <td>Dòng 1 Cột 1 của bảng</td>
 <td>Dòng 1 Cột 2 của bảng</td>
 </tr>
 <tr>
 <td>Dòng 2 Cột 1 của bảng</td>
 <td>Dòng 2 Cột 2 của bảng</td>
 </tr>
 <tr>
 <td>Dòng 3 Cột 1 của bảng</td>
 <td>Dòng 3 Cột 2 của bảng</td>
 </tr>
 <tr>
 <td>Dòng 4 Cột 1 của bảng</td>
 <td>Dòng 4 Cột 2 của bảng</td>
 </tr>
 <tr>
 <td>Dòng 5 Cột 1 của bảng</td>
 <td>Dòng 5 Cột 2 của bảng</td>
 </tr>
 <tr>
 <td>Dòng 6 Cột 1 của bảng</td>
 <td>Dòng 6 Cột 2 của bảng</td>
 </tr>
</table>
```

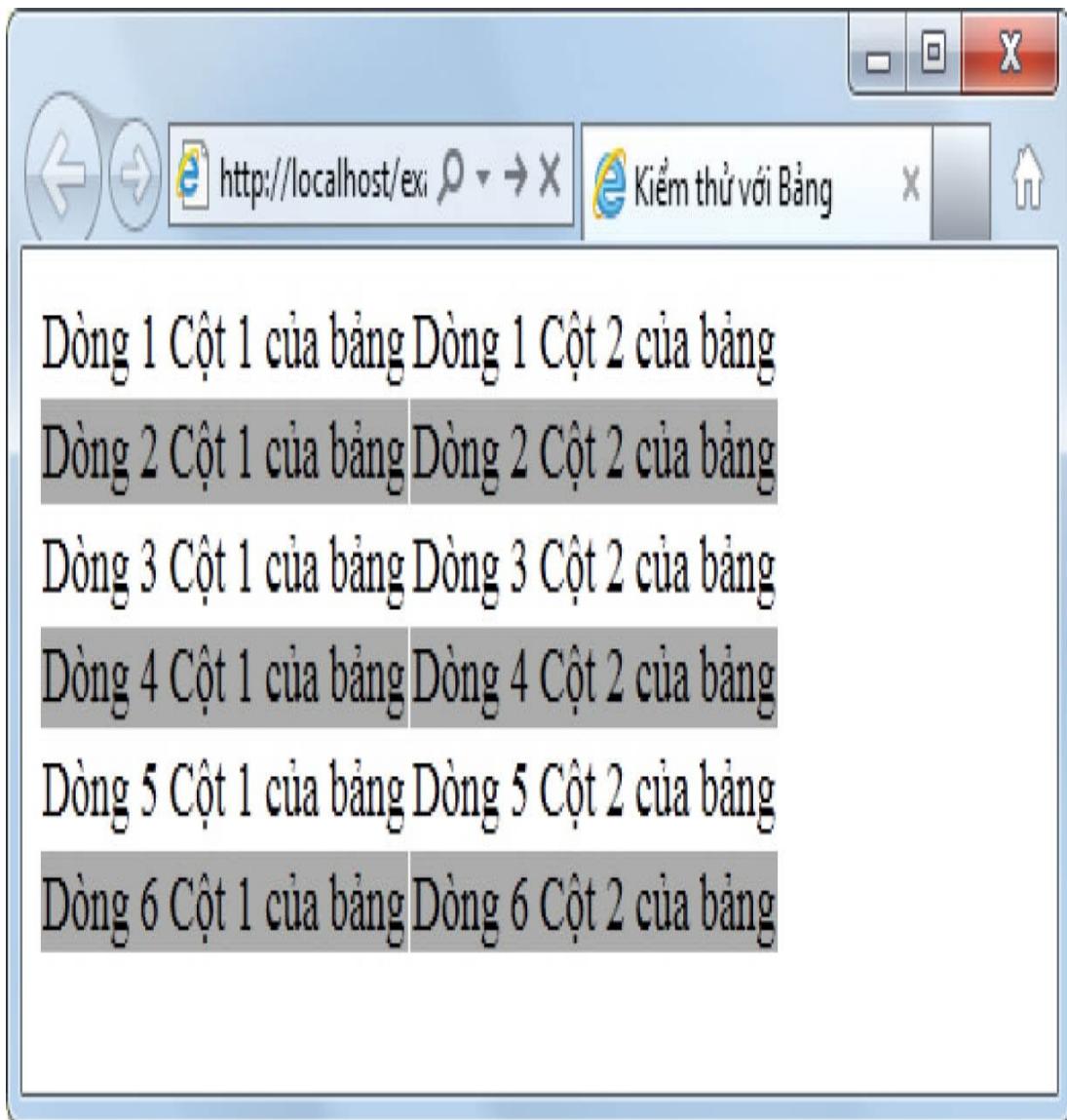
```
<script type="text/javascript">
$(document).ready(function() {
 $('tr:odd').css("background-color", "#abac
});
</script>
</body>
</html>
```

Phần chính của đoạn mã trên nằm trong đoạn mã JavaScript thuộc phần thân (body) của HTML:

```
$(document).ready(function() {
 $('tr:odd').css("background-color", "#abacab");
});
```

Đoạn mã sử dụng hàm `$(document).ready()` cùng với bộ chọn `:odd` để thiết lập màu nền bằng màu có mã hệ 16 là `#abacab` - tương ứng với màu xám nhạt. Hình 22-1 hiển thị kết quả của ví dụ này.





HÌNH 22-1 Bảng sau khi được tô màu với sự trợ giúp của jQuery.

Bạn vừa tìm hiểu những bộ chọn theo vị trí phổ biến nhất, tuy nhiên còn có rất nhiều bộ chọn theo vị trí khác. Tham khảo thêm tại địa chỉ <http://api.jquery.com/category/selectors/>.

## Chọn phần tử theo thuộc tính

Nếu bạn vẫn chưa cảm thấy chắc chắn với việc chọn phần tử theo lớp, jQuery cho phép bạn chọn các phần tử chỉ chứa một thuộc tính hoặc các phần tử chứa một thuộc tính với giá trị xác định. Ví dụ, để chọn tất cả các phần tử ảnh có thuộc tính `alt`, cú pháp chọn sẽ như sau:

```
$("img[alt]")
```

Nếu muốn chọn chỉ những phần tử ảnh với thuộc tính *alt* có giá trị xác định, bạn có thể viết:

```
$("img[alt='alternate text']")
```

Đoạn mã trên chỉ chọn ra phần tử ảnh nếu thuộc tính *alt* của ảnh đó đúng bằng *alternate text*. Hãy chú ý tới cách sử dụng dấu nháy đơn và nháy kép trong ví dụ này. Bộ chọn được đặt trong dấu nháy kép còn giá trị của thuộc tính *alt* được đặt trong dấu nháy đơn, tuy nhiên hai dấu nháy này có thể đảo ngược vị trí cho nhau, dấu nháy đơn dùng cho bộ chọn còn dấu nháy kép cho thuộc tính của bộ chọn:

```
$('img[alt="alternate text"]')
```

Bạn cũng có thể chỉ dùng một loại dấu nháy, tuy nhiên bạn phải thực hiện thoát ký tự nháy kép bằng cách thêm ký tự gạch chéo giống như sau:

```
$("img[alt=\"alternate text\"]")
```

Điều quan trọng cần chú ý đó là kiểu bộ chọn này đòi hỏi thuộc tính phải chính xác. Trong ví dụ này, thuộc tính *alt* phải đúng bằng chuỗi "*alternate text*". Các chuỗi khác như "*alternate text 2*" hoặc " *alternate text*" đều không phù hợp và không được chọn.

jQuery cũng chứa các bộ chọn đại diện, không đòi hỏi thuộc tính chọn phải chính xác hoàn toàn. Hãy xem một số ví dụ trong Bảng 22-2.

## BẢNG 22-2 Bộ chọn đối chiếu theo thuộc tính.

| Cú pháp                  | Mô tả                                                                                                                                                |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>attribute*=value</i>  | Chọn tất cả các phần tử chứa thuộc tính mà giá trị của nó có chứa chuỗi con đang đối chiếu.                                                          |
| <i>attribute~=value</i>  | Chọn tất cả các phần tử chứa thuộc tính mà giá trị của nó có chứa từ đang đối chiếu.                                                                 |
| <i>attribute!=value</i>  | Chọn tất cả các phần tử hoặc không chứa thuộc tính đang đối chiếu, hoặc chứa thuộc tính đó nhưng giá trị của nó không khớp với chuỗi đang đối chiếu. |
| <i>attribute\$=value</i> | Chọn các phần tử chứa thuộc tính có giá trị kết thúc bằng chuỗi đang đối chiếu.                                                                      |
| <i>attribute^=value</i>  | Chọn các phần tử chứa thuộc tính có giá trị bắt đầu bằng chuỗi đang đối chiếu.                                                                       |

# Chọn các phần tử trên form

jQuery hỗ trợ sẵn một số bộ chọn cho phép chọn các phần tử trên form. Bảng 22-3 liệt kê một số bộ chọn này, một vài bộ chọn trong số này sẽ được sử dụng trong phần còn lại của chương.

BẢNG 22-3 Các bộ chọn liên quan tới form.

| Cú pháp                | Mô tả                                                                |
|------------------------|----------------------------------------------------------------------|
| <code>:checkbox</code> | Chọn tất cả check box.                                               |
| <code>:checked</code>  | Chọn tất cả các phần tử được đánh dấu chọn, ví dụ như các check box. |
| <code>:input</code>    | Chọn tất cả các phần tử input trên trang web.                        |
| <code>:password</code> | Chọn tất cả các phần tử input kiểu password.                         |
| <code>:radio</code>    | Chọn tất cả các radio button.                                        |
| <code>:reset</code>    | Chọn tất cả các phần tử input có kiểu reset.                         |
| <code>:selected</code> | Chọn tất cả các phần tử đang được chọn trên form.                    |
| <code>:submit</code>   | Chọn tất cả phần tử input kiểu <code>submit</code> .                 |
| <code>:text</code>     | Chọn tất cả các phần tử input có kiểu text.                          |

## Các bộ chọn khác

Còn có rất nhiều kiểu bộ chọn khác trong jQuery, ví dụ như những bộ chọn chọn tất cả các phần tử ẩn (`:hidden`) hoặc chọn các phần tử hiển thị (`:visible`) hoặc chọn các phần tử được kích hoạt, phần tử bị vô hiệu hóa, v.v... Tham khảo thêm tại địa chỉ <http://api.jquery.com/category/selectors/> để tìm hiểu đầy đủ về các bộ chọn trong jQuery.

**Mách nhỏ** Thay vì nghĩ ra những cú pháp chọn phức tạp và dễ dẫn tới sai sót, bạn hãy tham khảo các bộ chọn trong jQuery (<http://api.jquery.com/category/selectors/>) để tìm hiểu các ví dụ và giải pháp có sẵn cho việc sử dụng bộ chọn.

# Hàm trong jQuery

Cho tới giờ, đã có khá nhiều ví dụ về cách chọn phần tử trong jQuery; tuy nhiên, chỉ có một số ít ví dụ minh họa những gì bạn có thể làm với các phần tử sau khi chọn. jQuery sử dụng hàm để thực hiện các hành động trên các phần tử được chọn. Hàm có thể là hàm có sẵn trong jQuery hoặc hàm người dùng tự định nghĩa. Thông thường, bạn luôn sử dụng kết hợp cùng lúc cả hai kiểu hàm này.

## Duyệt các phần tử trong DOM

Về bản chất, việc lập trình web sử dụng JavaScript và jQuery thường yêu cầu duyệt qua một số phần tử - ví dụ, hàm `.each()` làm việc với một danh sách phần tử được chọn, duyệt qua từng phần tử và thực hiện thao tác nào đó (hoặc không gì cả) với mỗi phần tử. jQuery chứa rất nhiều hàm hỗ trợ chức năng lặp. Quá trình lặp này trong jQuery được gọi là *duyệt (traversing)*. Bạn có thể tìm thêm thông tin về các hàm duyệt tại địa chỉ <http://api.jquery.com/category/traversing/>.

Khi sử dụng các hàm duyệt, hầu như bạn luôn kết hợp với một hàm tự định nghĩa và bộ chọn `$(this)`. Cũng như từ khóa `this` trong lập trình hướng đối tượng, bộ chọn `$(this)` tham chiếu tới đối tượng hiện tại – trong trường hợp này đó là phần tử đang được duyệt.

Chúng ta hãy cùng xét ví dụ sau. Đoạn mã HTML dưới đây tạo ra một trang web xếp hạng các đội trong một giải bóng chuyền hư cấu:

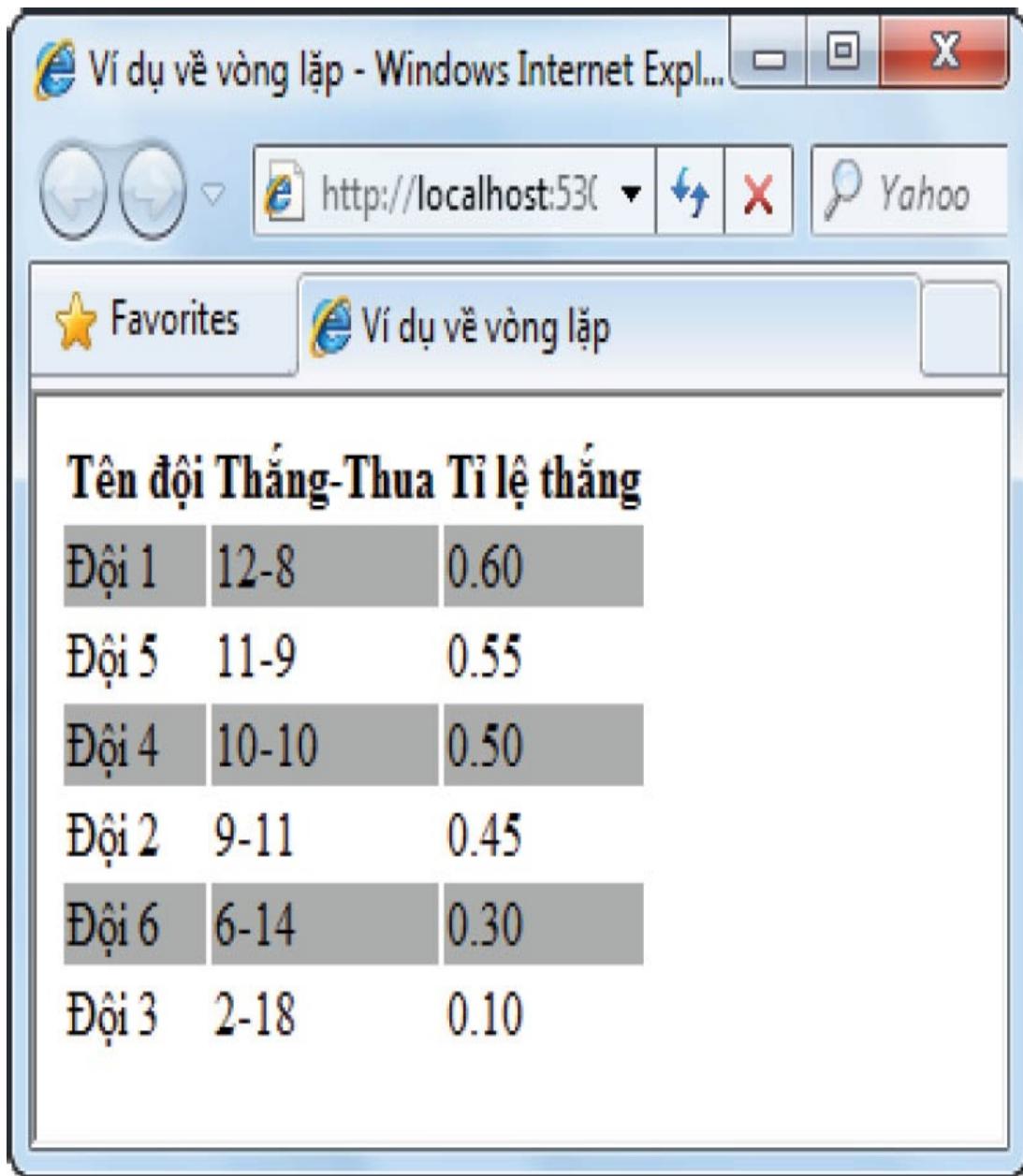
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ về vòng lặp</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
</head>
<body>
<table>
 <thead>
 <th>Tên đội</th>
 <th>Thắng-Thua</th>
 <th>Tỉ lệ thắng</th>
 </thead>
 <tbody>
 <tr>
 <td>Đội 1</td>
 <td>12-8</td>
 <td class="percentage">0.60</td>
```

```

</tr>
<tr>
 <td>Đội 5</td>
 <td>11-9</td>
 <td class="percentage">0.55</td>
</tr>
<tr>
 <td>Đội 4</td>
 <td>10-10</td>
 <td class="percentage">0.50</td>
</tr>
<tr>
 <td>Đội 2</td>
 <td>9-11</td>
 <td class="percentage">0.45</td>
</tr>
<tr>
 <td>Đội 6</td>
 <td>6-14</td>
 <td class="percentage">0.30</td>
</tr>
<tr>
 <td>Đội 3</td>
 <td>2-18</td>
 <td class="percentage">0.10</td>
</tr>
</table>
<script type="text/javascript">
$(document).ready(function() {
 $('tr:odd').css("background-color", "#abacab");
});
</script>
</body>
</html>

```

Khi xem trên trình duyệt, trang web sẽ trông như Hình 22-2.



HÌNH 22-2 Trang web xếp hạng các đội trong giải bóng chuyền.

Ví dụ này lặp qua các phần tử chứa thuộc tính *class* bằng *percentage* – lớp này được áp dụng cho các ô trong cột Tỉ lệ thắng. Trong ví dụ này, các đội có tỉ lệ thắng trên 0.5 (có nghĩa là thắng trên một nửa số trận đấu) sẽ được hiển thị font chữ in đậm trong cột Tỉ lệ Thắng. Bạn có thể thực hiện điều này với đoạn mã jQuery sau đây, đoạn mã này được thêm vào ngay sau đoạn mã jQuery đã có trong trang:

```
$('.percentage').each(function() {
if ($(this).text() >= .5) {
```

```
$(this).css('font-weight', 'bold');
}
});
```

Đoạn mã trên sử dụng một bộ chọn để thu thập các phần tử có thuộc tính *class* bằng *percentage*. Sau đó, nó truy xuất tới từng phần tử thông qua hàm *.each()* của jQuery. Trong hàm *.each()*, một hàm tự định nghĩa sẽ thực hiện phép kiểm tra điều kiện để xác định giá trị trong cột Tỉ lệ thắng có lớn hơn hoặc bằng 0.5 hay không. Nếu lớn hơn, đoạn mã sẽ gọi tới hàm *.css()* để thêm thuộc tính *font-weight* được thiết lập là *bold* cho phần tử. Sau khi thêm đoạn mã này vào trang web, kết quả sẽ giống như trong Ví dụ 22-3.

**VÍ DỤ 22-3** Thêm jQuery vào trang web xếp hạng giải bóng chuyền.

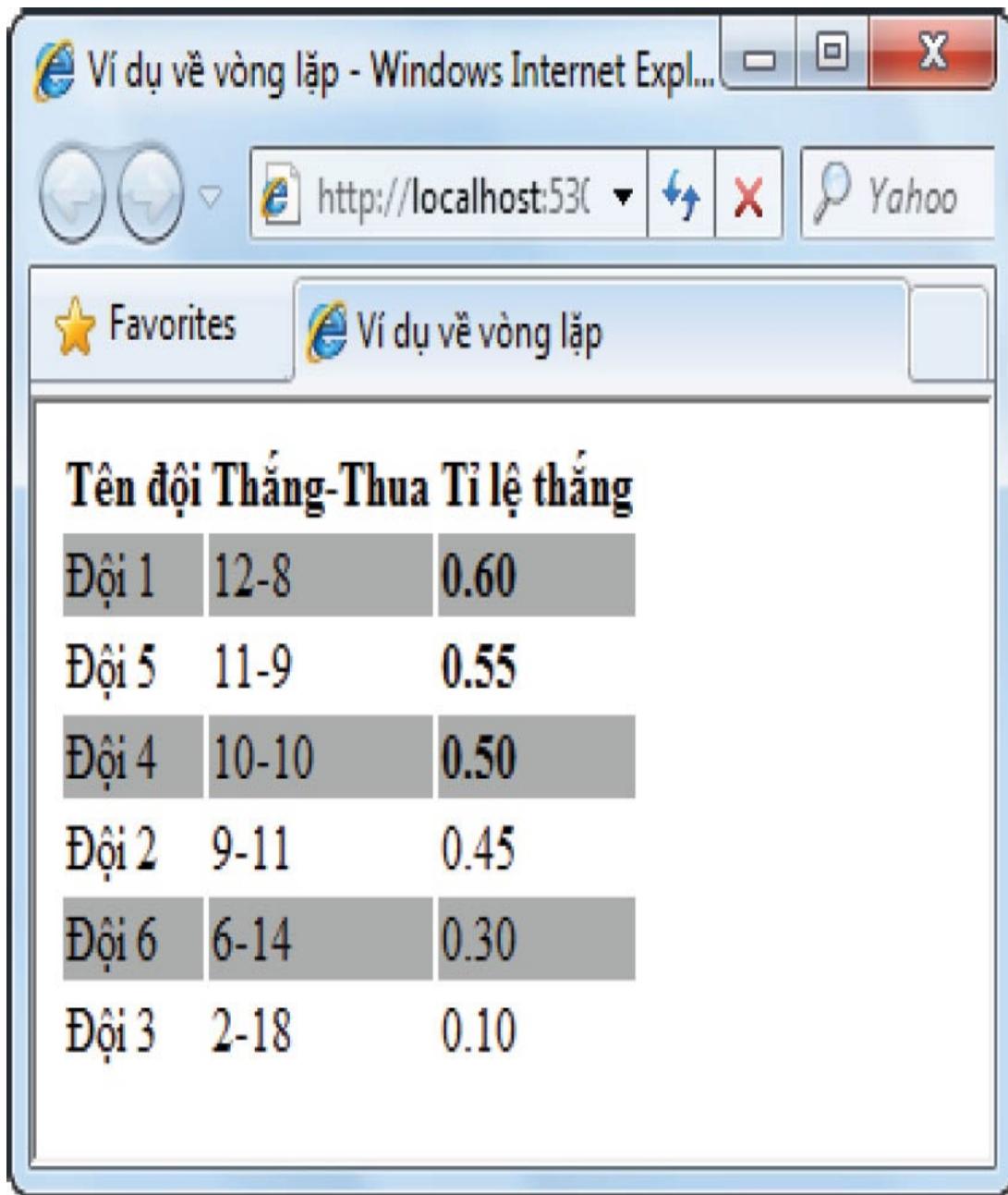
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ về vòng lặp</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.
</head>
<body>
<table>
 <th>Tên đội</th>
 <th>Thắng-Thua</th>
 <th>Tỉ lệ thắng</th>
 <tr>
 <td>Đội 1</td>
 <td>12-8</td>
 <td class="percentage">0.60</td>
 </tr>
 <tr>
 <td>Đội 5</td>
 <td>11-9</td>
 <td class="percentage">0.55</td>
 </tr>
 <tr>
 <td>Đội 4</td>
 <td>10-10</td>
 <td class="percentage">0.50</td>
 </tr>
```

```

<tr>
 <td>Đội 2</td>
 <td>9-11</td>
 <td class="percentage">0.45</td>
</tr>
<tr>
 <td>Đội 6</td>
 <td>6-14</td>
 <td class="percentage">0.30</td>
</tr>
<tr>
 <td>Đội 3</td>
 <td>2-18</td>
 <td class="percentage">0.10</td>
</tr>
</table>
<script type="text/javascript">
$(document).ready(function() {
 $('tr:odd').css("background-color", "#abacab");
 $('.percentage').each(function() {
 if ($(this).text() >= .5) {
 $(this).css('font-weight', 'bold')
 }
 });
});
</script>
</body>
</html>

```

Khi bạn dùng trình duyệt mở trang, những dòng tương ứng với các đội có tỉ lệ trận thắng trên 0.5 trong cột Tỉ lệ thắng sẽ được in đậm, giống như trong Hình 22-3.



HÌNH 22-3 Cột Tỉ lệ thắng được in đậm với sự trợ giúp của jQuery.

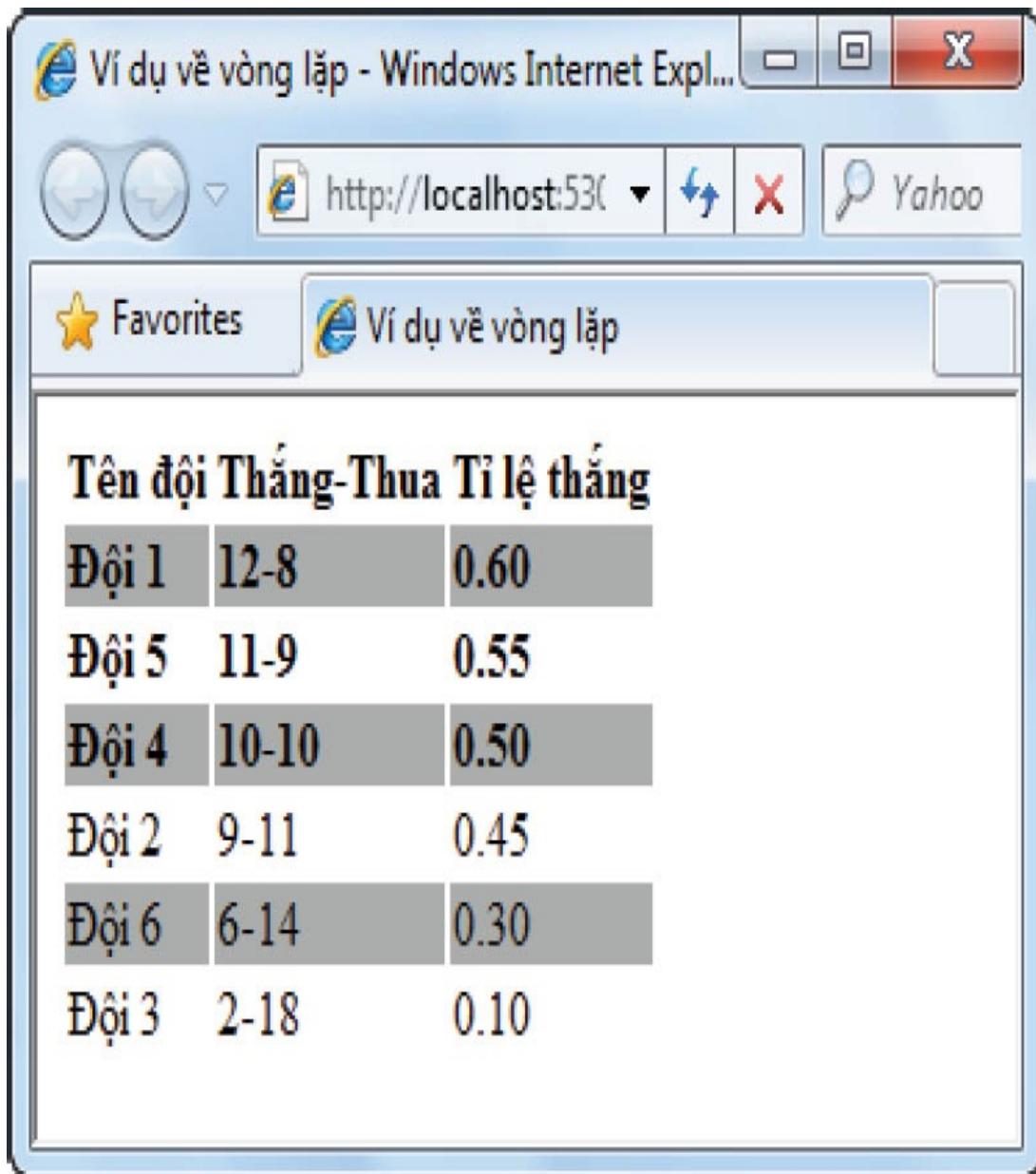
Nhìn vào kết quả trong Hình 22-3, việc áp dụng font in đậm cho toàn bộ dòng dữ liệu sẽ làm cho bảng hiển thị đẹp hơn là chỉ bôi đậm trên cột Tỉ lệ thắng. Điều này có vẻ khó thực hiện vì đoạn mã đã duyệt qua các phần tử dòng trong bảng khi điều kiện kiểm tra được áp dụng để tìm giá trị trong ô Tỉ lệ thắng. May mắn là jQuery có một hàm giúp thực hiện việc này: hàm `.parent()`. (Trên thực tế, có thể có một vài cách khác nhau và hàm `.parent()` chỉ là một trong số các cách đó).

Về cơ bản hàm `.parent()` sẽ duyệt ngược lại cây phân cấp DOM để tìm thẻ `<tr>` gần nhất chứa phần tử `<td>`. Bằng cách áp dụng định dạng CSS cho phần tử `<tr>` bạn có thể làm cho toàn bộ dòng chuyển thành kiểu in đậm. Đoạn mã mới sẽ giống như sau, phần thay đổi được in đậm:

```
$('.percentage').each(function() {
 if ($(this).text() >= .5) {
 $(this).**parent()**.css('font-weight', 'bold');
 }
});
```

Khi thêm đoạn mã trong Ví dụ 22-3, kết quả sẽ tương tự như trong Hình 22-4.





HÌNH 22-4 Áp dụng định dạng CSS cho các dòng trong bảng.

**Chú ý** Đoạn mã sửa đổi này có trong file listing22-4 .html của Tài nguyên đi kèm.

Sử dụng hàm `.parent()` đưa ra một khái niệm mới đó là chuỗi hàm. *Chuỗi hàm (chaining)* là một cấu trúc rất mạnh trong jQuery, nó cho phép chọn các phần tử ở những cấp độ khác nhau cũng như cho phép xử lý đa cấp trong hàm. Trong ví dụ này, bộ chọn `$(this)` gọi tới hàm `.parent()`, hàm này thực hiện

chọn phần tử cha của `$(this)`. Sau đó đoạn mã gọi tới hàm `.css()`.

Chuỗi hàm có thể gây ra một số vấn đề nguy hiểm. Sử dụng chuỗi hàm có thể khiến đoạn mã trở nên khó đọc và khó bảo trì. Ngoài ra, chuỗi hàm còn tạo ra các đoạn mã dễ bị lỗi, nhất là khi các phần tử được chọn trong chuỗi thay đổi. Chuỗi hàm thực sự rất hữu dụng – và theo tôi, bạn nên sử dụng chuỗi hàm nếu có thể – tuy nhiên, bạn không thể hy sinh tính dễ đọc hay dễ bảo trì của mã.

Các ví dụ trong chương này cho tới giờ đều sử dụng JavaScript để truy cập và thay đổi trực tiếp thuộc tính CSS. Như đã nói trong Chương 15, trên thực tế, chúng ta không thường dùng JavaScript để thay đổi style hay định dạng trình bày của trang web. Cách tốt hơn đó là áp dụng hoặc loại bỏ các style thông qua CSS thay vì thay đổi trực tiếp các thuộc tính. Trong jQuery có một số phương thức làm việc với các lớp CSS như các hàm `.hasClass()`, `.addClass()`, `.removeClass()`, và `.toggleClass()`. Xem thêm thông tin về cách làm việc với lớp CSS sử dụng các hàm này tại địa chỉ <http://api.jquery.com/category/css/>.

## Làm việc với các thuộc tính

Ngoài những hàm liên quan tới thuộc tính `class`, jQuery còn có các hàm làm việc với các thuộc tính của DOM. Một trong các hàm phổ biến nhất đó là hàm `.attr()`, ngoài ra còn có một số hàm khác cũng rất hữu dụng như `.html()` và `.val()`. Phần này sẽ trình bày hàm `attr()`, còn hàm `.html()`, `.val()`, và một số hàm khác sẽ được đề cập sơ lược trong phần cuối.

Hàm `.attr()` được sử dụng để truy xuất cũng như thiết lập các thuộc tính. Ví dụ, bạn có thể vừa truy xuất vừa thiết lập thuộc tính `alt` của một phần tử ảnh bằng cú pháp sau đây:

```
// Truy xuất thuộc tính alt:
$("#myImageID").attr("alt")
// Thiết lập thuộc tính alt:
$("#myImageID").attr("alt", "đoạn văn bản mới")
```

**Chú ý** Không nhất thiết phải truy xuất giá trị của phần tử rồi mới thiết lập giá trị.

# Thay đổi văn bản và HTML

Bạn có thể viết lại toàn bộ một trang web bằng cách sử dụng các hàm như `.text()` và `.val()`. Tất nhiên, có khả năng không có nghĩa là tốt. Tuy vậy, đôi khi chúng ta cần viết lại một phần HTML trong trang hoặc thay đổi các giá trị văn bản.

Hàm `.html()` cho phép truy xuất và thay đổi toàn bộ HTML trong phần tử được chọn. Ví dụ, hãy xét đoạn mã HTML sau:

```
<div id="myDiv">Đây là một thẻ div, nó khá đẹp mắt</div>
```

Và đây là mã jQuery:

```
$("#myDiv").html('Đây là nội dung mới
```

Kết quả của mã jQuery này là thẻ `<div>` có id là `myDiv` chứa một phần tử `<span>` cùng với dòng văn bản mới như trong đoạn mã ví dụ. Ví dụ trên tương đối đơn giản, tuy nhiên thẻ `<div>` có thể chứa những nội dung phức tạp. Sử dụng jQuery, bạn có thể viết lại toàn bộ phần tài liệu HTML nếu muốn.

Cũng giống như hàm `.html()`, hàm `text()` hỗ trợ truy xuất và thiết lập giá trị văn bản cho phần tử được chọn. Tuy nhiên, hàm này chỉ cho phép thay đổi văn bản, do đó không thể sử dụng nó để thay đổi HTML.

```
<div id="myDiv">Đây là một thẻ div, nó khá đẹp mắt</div>
$("#myDiv").text('Đây là nội dung mới của thẻ div');
```

Trong ví dụ trên, chỉ có phần văn bản được thay đổi; đoạn mã không thêm phần tử `<span>` hay áp dụng style cho phần tử được chọn.

## Chèn thêm các phần tử

Bạn có thể dễ dàng sử dụng jQuery để chèn thêm các phần tử vào trang web. Hai hàm cơ bản để thực hiện điều này là hàm `:after()` và `:before()`. Đúng như tên gọi, hai hàm này chèn tương ứng các phần tử vào trước và vào sau phần tử được chọn.

Ví dụ, vẫn là thẻ `div` như trên:

```
<div id="myDiv">Đây là một thẻ div, nó khá đẹp mắt</div>
```

Dưới đây là mã jQuery thêm một thẻ *div* khác vào trước thẻ *div* hiện tại:

```
$("#myDiv").before("<div>Đây là thẻ div mới</div>");
```

Hàm *:after()* thực thi tương tự:

```
$("#myDiv").after("<div>Đây là thẻ div mới, nó xuất hiện sau
```

Khi chạy, trang web chứa đoạn mã này sẽ có ba thẻ *<div>*:

```
<div>Đây là thẻ div mới</div>
<div id="myDiv">Đây là một thẻ div, nó khá đẹp mắt</div>
<div>Đây là thẻ div mới, nó xuất hiện sau myDiv</div>
```

Các ví dụ trên minh họa việc chèn thêm các phần tử *<div>* - nhưng dĩ nhiên bạn có thể sử dụng những những hàm này cho bất kỳ phần tử hợp lệ nào khác.

## Hàm callback



Đôi khi bạn cần chạy một hàm khi *hàm khác* hoặc một phần của một hàm hoàn thành, hàm đó được gọi là *hàm callback*. Hàm callback thực thi sau khi hàm cha của nó hoàn thành. jQuery sử dụng rất nhiều hàm callback, đặc biệt trong AJAX. Bạn đã được thấy một ví dụ về hàm callback khi thực hiện lặp sử dụng hàm *.each()*.

Xem thêm về hàm callback tại địa chỉ [http://docs.jquery.com/Tutorials:How jQueryWorks](http://docs.jquery.com/Tutorials:How_jQueryWorks).

Bạn sẽ tìm hiểu thêm một số ví dụ về hàm callback trong phần còn lại của chương này. Nếu mới bắt đầu lập trình JavaScript, bạn đừng nên nghĩ phức tạp về hàm callback. Hàm callback đơn giản chỉ là một khối mã lệnh được gọi bên trong một hàm khác.

## Các sự kiện trong jQuery

Cho tới giờ, bạn đã gặp một số ví dụ về bộ chọn và các hàm cơ bản trong jQuery.

Phần cuối của chương giới thiệu về jQuery sẽ đề cập tới các sự kiện. Cũng tương tự như việc xử lý sự kiện trong JavaScript, jQuery cho phép chương trình phản hồi lại các sự kiện nhấn chuột, gửi form, nhấn phím, v.v... Khác JavaScript, tính tương thích trình duyệt trong xử lý sự kiện của jQuery rất tuyệt vời. jQuery có tính tương thích cao với mọi trình duyệt. Điều này đặc biệt đúng với xử lý sự kiện, bạn không cần bận tâm tới việc trình duyệt xử lý hàm bạn viết như thế nào.

## Gắn và gỡ hàm xử lý sự kiện

Hàm `.bind()` kết nối hàm xử lý sự kiện với sự kiện, ví dụ như sự kiện nhấn chuột:

```
.bind(event, data, handler)
```

Trong ví dụ này, `event` là một sự kiện bạn muốn đáp ứng, `data` là đối tượng không bắt buộc, chứa dữ liệu được truyền vào hàm xử lý sự kiện và `handler` là hàm bạn muốn thực thi khi sự kiện được kích hoạt.

Ví dụ:

```
Liên kết
$("#myLink").bind("click", function() {
 alert("Đã nhấn vào liên kết");
});
```



Kết quả của đoạn mã trên là khi người dùng nhấn chuột vào thẻ `<a>`, sự kiện `click` được kích hoạt và trang web sẽ hiển thị một hộp thoại thông báo. Chú ý rằng ví dụ này không sử dụng tới tham số `data` trong hàm `.bind()`.

Bạn có thể gắn kết các sự kiện sau đây trong hàm `.bind()`:

- beforeunload
- blur
- change
- click
- dblclick
- error

focus

- ▪ focusin
- focusout
- hover
- keydown
- keypress
- keyup
- load
- mousedown
- mouseenter
- mouseleave
- mousemove
- mouseout
- mouseover
- mouseup
- resize
- scroll
- select
- submit
- toogle
- unload



Trong các chương trước, bạn đã biết cách bắt sự kiện bằng đoạn mã ehandler.js đã được xây dựng trong Chương 11 “Các sự kiện trong JavaScript và làm việc với trình duyệt”. Đoạn mã ehandler.js cung cấp một hàm cho phép gán hàm xử lý sự kiện với khả năng tương thích với nhiều trình duyệt. Về cơ bản, chức năng của hàm `.bind()` trong jQuery cũng tương tự như vậy. Điểm khác nhau đó là hàm `.bind()` của jQuery có tính tương thích cao hơn và mạnh hơn so với đoạn mã ehandler.js.

Dù bạn có thể sử dụng hàm `.bind()` để gán hàm xử lý sự kiện, song jQuery còn cung cấp một hàm tắt cũng với chức năng tương tự như hàm `.bind()`. Thay vì viết `.bind("click", function())...` bạn chỉ cần viết

.click(function()... Ví dụ, bạn có thể viết lại hàm .bind() ở ví dụ trước như sau:

```
$("#myLink").click(function() {
 alert("Đã nhấn vào liên kết");
});
```

Không chỉ phản hồi lại các sự kiện như nhấn vào đường liên kết, bạn còn có thể kích hoạt các sự kiện. Ví dụ, hãy xem đoạn mã trong Ví dụ 22-5:

### VÍ DỤ 22-5 Phản hồi lại các sự kiện.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
<html>
<head>
<title>Ví dụ về xử lý sự kiện</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.
</head>
<body>
<div id="myDiv">
Đây là một văn bản.

Văn bản nằm trong thẻ div này

</div>
<p>
 Steve
</p>
<script type="text/javascript">
$(document).ready(function() {
//Xử lý sự kiện khi nhấn vào liên kết
$('#braingiaLink').click(function() {
 alert("xin chào");
 return true;
});
//Xử lý sự kiện khi nhấn vào thẻ div
$('#myDiv').click(function() {
 $('#braingiaLink').click();
});
});
</script>
</body>
</html>
```

Khi trang web được nạp vào trình duyệt, nếu nhấp vào thẻ `<div>`, sự kiện `click` của phần tử `<a>` sẽ được kích hoạt như khi bạn nhấp vào phần tử đó.

Để ngừng xử lý một sự kiện, bạn có thể sử dụng hàm `unbind()`, hàm này có hai đối số:

```
.unbind(event, function)
```

Đối số `event` là sự kiện bạn muốn ngừng xử lý, còn đối số `function` là hàm hiện tại đang gắn với sự kiện.

**Chú ý** Bạn có thể gắn kết nhiều hàm xử lý sự kiện tới cùng một sự kiện bằng cách gọi tới hàm `.bind()` nhiều lần.

## Các sự kiện của chuột và hàm Hover

Trong ví dụ trước, bạn đã được thấy cách gắn kết (bind) và xử lý sự kiện `click`. Bạn cũng có thể làm việc với các sự kiện chuột khác như `mouseover` và `mouseout`. Một ví dụ rất thú vị đó là bạn có thể làm cho các phần tử biến mất và xuất hiện trở lại bằng thao tác di chuột (Bạn không nên áp dụng hiệu ứng này cho các trang web trực tuyến vì nó có thể gây khó chịu cho người dùng). Ví dụ 22-6 trình bày một vài dòng mã để làm cho phần tử `<a>` biến mất khi di chuột lên đoạn văn bản bên trong.

**VÍ DỤ 22-6** Làm việc với các sự kiện chuột.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ về xử lý sự kiện</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
<style type="text/css">

#braingiaLink {
 border: solid 1px black;
 padding: 3px;
}
```

```

#wrapperP {
 padding: 50px;
}
</style>
</head>
<body>
<div id="myDiv">
Đây là một văn bản.

Văn bản nằm trong thẻ div này

</div>
<p id="wrapperP">
 Steve
</p>
<script type="text/javascript">
$(document).ready(function() {
//Xử lý sự kiện khi nhấn chuột vào liên kết
$('#braingiaLink').click(function() {
 alert("xin chào");
 return true;
});
//Xử lý sự kiện khi nhấn vào thẻ div
$("#myDiv").click(function() {
 $('#braingiaLink').click();
});
//Xử lý sự kiện khi di chuột lên #wrapperP
$('#wrapperP').mouseover(function() {
 $('#braingiaLink').hide();
});
//Xử lý sự kiện khi di chuột ra khỏi #wrapperP
$('#wrapperP').mouseout(function() {
 $('#braingiaLink').show();
});
});
});
</script>
</body>
</html>

```

Phần chính trong đoạn mã trên là các hàm xử lý sự kiện `.mouseover()` và `.mouseout()`, trong đó sử dụng hai hàm bổ trợ của jQuery đó là `.hide()` và `.show()`. Các sự kiện `.mouseover()` và `.mouseout()` được gắn kết với đoạn văn bản có ID là `wrapperP`. Khi di chuột lên trên đoạn văn bản này, phần tử

`<a>` có id là `braingiaLink` sẽ biến mất và chỉ xuất hiện trở lại khi bạn di chuột ra khỏi vùng văn bản. Cần chú ý là đường liên kết đó vẫn có thể được kích hoạt bằng bàn phím. Hãy nhớ có rất nhiều cách để làm việc với trang web.

Ngoài ra jQuery còn có hàm `.hover()`, hàm này hoạt động gần giống với các sự kiện `.mouseover()` và `.mouseout()`. Xem thêm về hàm `.hover()` tại địa chỉ <http://api.jquery.com-hover/>.

## Các hàm xử lý sự kiện khác

Như đã trình bày trong bảng trước đó, jQuery có rất nhiều hàm xử lý sự kiện và chương này không thể đề cập tất cả những sự kiện đó. Các bạn có thể tham khảo thêm về sự kiện trong jQuery tại địa chỉ <http://api.jquery.com/category/events/>.

# AJAX và jQuery



Hai chương trước đã giới thiệu cách viết và sử dụng AJAX. jQuery cũng có các hàm hỗ trợ làm việc với AJAX. Và giống như các tác vụ JavaScript khác, jQuery khiến việc sử dụng AJAX trở nên dễ dàng hơn. Phần này sẽ minh họa cách sử dụng AJAX trong jQuery.

jQuery hỗ trợ một vài hàm làm việc với dữ liệu gửi đi và nhận về từ server. Trong số đó phải kể tới các hàm `.load()`, `.post()` và hàm `.get()`. jQuery cũng bao gồm một hàm AJAX có tên là `.ajax()`.

Sử dụng hàm này, bạn có thể thiết lập các tham số, bao gồm phương thức HTTP mà hàm nên sử dụng (`GET` hoặc `POST`), thời gian timeout và cách xử lý khi có lỗi xảy ra (cũng như khi mã thực thi thành công).

**Thông tin bổ sung** Xem thêm tại địa chỉ <http://api.jquery.com/jQuery.ajax/> để biết đầy đủ về các tham số sử dụng trong hàm `.ajax()`.

Hàm `ajax()` có cú pháp cơ bản như sau:

```
$.ajax({
 parameter: value
});
```

Bạn có thể truyền nhiều cặp tham số/giá trị *parameter : value* tới hàm *.ajax()*; tuy nhiên, thông thường đó là phương thức HTTP, URL và hàm callback. Bạn cũng thường phải xác định rõ kiểu dữ liệu trả về, chế độ lưu trữ phản hồi trong bộ nhớ lưu trữ đệm (cache), dữ liệu truyền tới server và hàm xử lý khi có lỗi xảy ra.

**Chú ý** Hàm *.ajaxSetup()* cho phép thiết lập các tham số AJAX mặc định như chế độ lưu trữ đệm, phương thức và hàm xử lý sự kiện.

Sau đây là một ví dụ sử dụng hàm *.ajax()* trong thực tế:

```
$.ajax({
 url: "testajax.aspx",
 success: function(data) {
 alert("Tải thành công.");
 }
});
```



jQuery cũng giới thiệu hàm *.getJSON()* cho phép thực hiện chức năng giống như các hàm AJAX trên; tuy nhiên, hàm này còn có khả năng làm việc với dữ liệu JSON được mã hóa từ server. Hàm *.getJSON()* tương đương lời gọi hàm *.ajax()* với tham số *dataType* được thiết lập bằng '*json*'.

Ví dụ, hãy xét danh sách các bang theo định dạng JSON sau đây:

```
["Wisconsin", "California", "Colorado", "Illinois", "Minnesota",
```

Trong ví dụ này, giả định rằng dữ liệu mã hóa theo định dạng JSON được trả về khi file *json.php* được gọi trên server. Dưới đây là đoạn mã sử dụng hàm *.ajax()* để truy xuất dữ liệu và gọi tới hàm *showStates* nếu thành công:

```
$.ajax({
 type: "GET",
 url: "json.php",
 dataType: "json",
 success: showStates
});
```

```
});
```

Hàm `showStates` tạo ra một danh sách và thêm danh sách này vào select box `<select>` trên form.

## Sử dụng AJAX với jQuery

Để hoàn thành bài tập này, bạn cần có sẵn file json.php trên cùng thư mục với file mà bạn sẽ sử dụng (file json.php có trong Tài nguyên đi kèm). Tương tự các bài tập khác về AJAX, file json.php phải nằm trên cùng domain với file tạo yêu cầu AJAX.

1. Chỉnh sửa file ajax.html (file này cũng nằm trong Tài nguyên đi kèm cuốn sách) bằng một trình soạn thảo bất kỳ.
2. Chỉnh sửa file ajax.html (file này cũng nằm trong Tài nguyên đi kèm cuốn sách) bằng một trình soạn thảo bất kỳ.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Kiểm tra AJAX</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js"></script>
</head>
<body>
<div id="states">
</div>
<script type="text/javascript">
$(document).ready(function() {
$.ajax({
type: "GET",
url: "json.php",
dataType: "json",
success: showStates
});
function showStates(data,status) {
$.each(data, function(item) {
$("#states").append("<div>" + data[item] + "</div>");
});
```

```
};
});
</script>
</body>
</html>
```

3. Lưu file và dùng trình duyệt mở lại. Danh sách các bang được hiển thị như trong hình dưới đây:





# Các lỗi AJAX và thời gian timeout

Hàm `.ajax()` cho phép xử lý các lỗi và thời gian timeout theo cách thân thiện với người dùng. Ngoài hàm xử lý trường hợp thành công, bạn có thể xác định hàm xử lý các lỗi với tham số `error`. Giá trị của tham số `error` thường là tên của một hàm callback. Trong ví dụ dưới đây, tham số `error` được in đậm:

```
$ajax({
 type: "GET",
 url: "json.php",
 dataType: "json",
 success: successFunction,
 error: errorFunction
});
```

Hàm callback được dùng để xử lý lỗi (hàm `errorFunction`) nhận ba đối số: đối tượng `XMLHttpRequest`, chuỗi biểu diễn lỗi được phát hiện và đối tượng `exception` (ngoại lệ), nếu có ngoại lệ xảy ra. Do đó, hàm xử lý lỗi cần chấp nhận cả ba đối số này và thực hiện tác vụ đổi với kết quả trả về. Ví dụ sau đây sẽ hiển thị thông báo:

```
function errorFunction(xhr, statusMessage, exceptionObj) {
 alert("Có lỗi xảy ra: " + statusMessage);
}
```

Việc thiết lập thời gian timeout cho yêu cầu AJAX cũng rất cần thiết. Bạn có thể thiết lập thời gian timeout toàn cục thông qua hàm mặc định `$.ajaxSetup`, tuy nhiên bạn cũng có thể thiết lập giá trị thời gian `timeout` cho từng lời gọi cụ thể. Sau đây là ví dụ:

```
$ajax({
 type: "GET",
 url: "json.php",
 dataType: "json",
 success: successFunction,
 error: errorFunction
 timeout: 5000
});
```

Một điều quan trọng có thể thấy đó là thời gian timeout được tính bằng mili giây.

Như vậy, ví dụ trên thiết lập thời gian timeout bằng 5 giây.

## Gửi dữ liệu tới Server

Trong lời gọi AJAX, bạn không chỉ nhận dữ liệu từ server mà còn cần gửi dữ liệu tới server và nhận về phản hồi. Hàm `.ajax()` được sử dụng với tham số `data` để thực hiện việc gửi dữ liệu tới server bằng phương thức `GET` hoặc `POST`.

Bạn có thể định dạng dữ liệu dưới dạng cặp giá trị `key=value` phân tách bằng dấu `&` (`key1=value1&key2=value2`) hoặc dưới dạng các cặp ánh xạ `{key1: value1, key2: value2}`. Ví dụ sau sử dụng định dạng `key=value`, định dạng này còn được gọi là chuỗi truy vấn.

Ví dụ này gọi tới một chương trình phía server có tên là `statefull.php`, chương trình này nhận đối số đầu vào là hai ký tự viết tắt của tên bang và trả về tên đầy đủ của bang tương ứng.

```
$.ajax({
 type: "POST",
 url: "statefull.php",
 dataType: "json",
 success: successFunction,
 data: "state=WI"
});
```



## Các tùy chọn quan trọng khác

Có rất nhiều tùy chọn trong hàm `.ajax()`. Bạn đã thấy cách sử dụng của đa số các tùy chọn này, ở đây tôi sẽ nhấn mạnh thêm hai tùy chọn nữa đó là:

- `async`
- `cache`

Tùy chọn `asyn` có giá trị mặc định bằng `true`, tùy chọn này cho biết liệu đoạn mã có phải đợi (và chặn thông tin nhập vào trình duyệt) khi giao dịch AJAX được gửi tới server và được xử lý không. Khi được thiết lập bằng `true`, giao dịch AJAX sẽ được thực thi theo cách không đồng bộ, do đó đoạn mã sẽ không thực hiện quá trình chặn.

Thiết lập *cache* có giá trị mặc định là *true* trong hầu hết các trường hợp, tùy chọn này điều khiển việc lưu trữ đệm cho giao dịch AJAX của jQuery. Điều này rất hữu ích khi dữ liệu nhận về không thay đổi thường xuyên, vì việc lưu trữ đệm giúp tăng tốc giao dịch, tuy nhiên nó cũng gây ra những rắc rối khi ứng dụng sử dụng dữ liệu đã cũ, chưa được cập nhật dù trên server dữ liệu đó đã bị thay đổi. Việc thiết lập tùy chọn này bằng *false* rất hữu dụng để không lưu trữ đệm những phản hồi từ server, đặc biệt trong trường hợp bạn phát hiện thấy dữ liệu không được nạp lại.

## Tìm hiểu thêm về jQuery

Cho tới lúc này, bạn mới chỉ thấy một phần nhỏ những gì jQuery có thể làm được. Khi đã hiểu sâu hơn về JavaScript và cách sử dụng JavaScript để hỗ trợ cho trang web, bạn có thể xem xét sử dụng jQuery hoặc một thư viện JavaScript khác để phát triển ứng dụng một cách nhanh chóng và dễ dàng.

*Tham khảo thêm các tài liệu về jQurery tại địa chỉ <http://www.jquery.com>.*



## Bài tập

1. Sử dụng đoạn mã trong file ajax.html (trong bài tập “Sử dụng Ajax với jQuery”), hãy thêm một style CSS để tạo màu nền xanh cho các thẻ *<div>* khi di chuột lên tên trong các bang trong danh sách. Gợi ý: Có rất nhiều cách để thực hiện điều này.
2. Tạo một chương trình phía server trả về dữ liệu khi truyền tham số sử dụng hàm *\$.ajax()*. Hiển thị dữ liệu nhận được bằng hộp thoại thông báo alert hoặc ghi ra trang web. Ví dụ, bạn có thể viết một chương trình phía server trả về tổng của hai số, hoặc giống ví dụ đã được trình bày trả về tên đầy đủ của một bang nếu chương trình nhận được tên viết tắt tương ứng.

# Chương 23

## Các hiệu ứng và plug-in cho jQuery

Sau khi đọc xong chương này, bạn sẽ có thể:

- Hiểu và biết cách sử dụng các hiệu ứng của jQuery.
- Nắm được jQuery UI.
- Sử dụng jQuery UI.

### Các tính năng chính giúp tăng cường tính khả dụng

Với jQuery, bạn có thể dễ dàng thực hiện các hiệu ứng cải tiến tính khả dụng như kéo thả, làm các phần tử hiện dần hay mờ dần, và dịch chuyển các phần tử. Nếu những tính năng này vẫn chưa làm bạn hoàn toàn hài lòng, jQuery còn có khả năng mở rộng và có cả một cộng đồng người dùng hỗ trợ.

Một trong những đóng góp của cộng đồng người dùng jQuery là các plug-in. Plug-in cung cấp các tính năng bổ sung không có trong gói jQuery lõi. Bạn có thể tìm hiểu thêm thông tin về các plug-in và danh sách các plug-in hiện có tại trang web về Plug-in của jQuery <http://plugins.jquery.com/>.

Chương này cung cấp cái nhìn tổng thể về một số hiệu ứng có trong gói jQuery lõi cũng như trong jQuery UI.

### Các hiệu ứng có sẵn của JQuery

Như đã đề cập trong phần giới thiệu của chương này, jQuery cung cấp một số

chức năng giúp tăng cường tính khả dụng của ứng dụng web, như hiện và ẩn các phần tử, khiến các phần tử hiện dần hay mờ dần. Phần này trình bày một số hiệu ứng có sẵn trong gói jQuery. Cần lưu ý tại thời điểm ra đời cuốn sách, jQuery và jQuery UI đang được cập nhật để tương thích với Windows Internet Explorer 9, vì thế một số ví dụ trong chương này có thể không chạy trên bản beta của Internet Explorer 9. Tuy vậy, nhiều khả năng là khi bạn đọc chương sách này, jQuery và jQuery UI đã được cập nhật và Internet Explorer 9 đã được phát hành.

## Hiệu ứng hiện, ẩn và đảo trạng thái

Hàm `.show()` và `.hide()` lần lượt làm hiện và ẩn các phần tử trên trang web. Các hàm này thiết lập thuộc tính `display` của CSS. Để ẩn một phần tử, cần gán thuộc tính `display` bằng `none`. Lưu ý rằng việc gán thuộc tính `display` bằng `none` không xóa mất phần tử khỏi DOM, vì thế bạn vẫn có thể làm hiện phần tử đó bằng hàm `.show()`. Ví dụ 23-1 chỉ ra cách sử dụng hàm `.hide()`.

### VÍ DỤ 23-1 Ẩn một phần tử.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ẩn phần tử</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.
<style type="text/css">
.removeBtn {
 color: #0000CC;
}
</style>
</head>
<body>

<li id="option1">Lựa chọn 1
<li id="option2">Lựa chọn 2
<li id="option3">Lựa chọn 3
<li id="option4">Lựa chọn 4

<script type="text/javascript">
$(document).ready(function() {
$('#option1').click(function() {
 $('#option1').hide();
```

```

});
$('#option2').click(function() {
 $('#option2').hide();
});
$('#option3').click(function() {
 $('#option3').hide();
});
$('#option4').click(function() {
 $('#option4').hide();
});
});
</script>
</body>
</html>

```

Tuy nhiên, đoạn mã trong Ví dụ 23-1 chưa được tối ưu hóa vì nó đòi hỏi phải tạo thêm phần xử lý sự kiện cho từng phần tử option. (Bạn sẽ không gặp kiểu lập trình như thế này trong một sản phẩm thương mại; tuy vậy, ví dụ này chỉ phục vụ cho mục đích minh họa). Một giải pháp tốt hơn là xử lý các phần tử option bằng chính các hàm của chúng. Ví dụ 23-2 trình bày cách thức tốt hơn để thực hiện tính năng tương tự bằng jQuery mà không dùng các phần tử option làm mã cứng.

### VÍ DỤ 23-2 Ẩn một phần tử, cải tiến bằng jQuery.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://
<html>
<head>
<title>Ẩn phần tử</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.
<style type="text/css">
.removeBtn {
 color: #0000CC;
}
</style>
</head>
<body>

<li id="option1">Lựa chọn 1 <span class="removeBtn"
 id="remove1">(x)
<li id="option2">Lựa chọn 2 <span class="removeBtn"
 id="remove2">(x)
<li id="option3">Lựa chọn 3 <span class="removeBtn"

```

```

 id="remove3">(x)
<li id="option4">Lựa chọn 4 <span class="removeBtn"
 id="remove4">(x)

<script type="text/javascript">
$(document).ready(function() {
 $('.removeBtn').each(function(elm) {
 $(this).click(function() {
 $(this).parent().hide();
 });
 });
});
</script>
</body>
</html>

```

Thay đổi duy nhất trong đoạn mã ở Ví dụ 23-2 là đoạn:

```

$('.removeBtn').each(function() {
 $(this).click(function() {
 $(this).parent().hide();
 });
});

```



Đoạn mã này gán một hàm xử lý sự kiện nhấn chuột (click) cho từng phần tử có thuộc tính *class* là *removeBtn*. Hàm xử lý sự kiện gọi hàm *.hide()*, nhưng vì khi sự kiện xảy ra, *(this)* trỏ tới phần tử *removeBtn* (trong trường hợp này là phần tử *<span>*), bạn cần đi ngược cây phân cấp để tìm nút cha, trong trường hợp này là phần tử *<li>*.

Hàm *.toggle()* – đảo trạng thái – giúp làm hiện hoặc ẩn phần tử dựa trên trạng thái hiện tại của phần tử. Ví dụ, khi phần tử đang hiển thị, việc gọi hàm *.toggle()* sẽ ẩn nó đi. Tương tự, khi một phần tử đang ẩn, phần tử đó sẽ hiện trở lại khi bạn gọi hàm *.toggle()*.

Cả ba hàm *.show()*, *.hide()* và *.toggle()* đều nhận hai đối số: khoảng thời gian và hàm callback. Lưu ý đoạn mã trong Ví dụ 23-1 và 23-2, khi bị ẩn, phần tử biến mất ngay lập tức. Việc thêm khoảng thời gian vào hàm *.hide()* khiến phần tử ẩn đi sau khoảng thời gian đã định. Giống những hàm khác trong jQuery, khoảng thời gian được tính bằng mili giây. Ngoài ra, bạn có thể sử dụng hằng số “*fast*” và “*slow*”, tương đương với 200 và 600 mili giây.

Hàm callback thực thi sau khi thao tác ẩn hoặc hiện phần tử được hoàn tất. Một ứng dụng của hàm callback là hiển thị button Undo (hoàn tác) sau khi ẩn một phần tử, ứng dụng này cho phép người dùng hiển thị lại phần tử vừa bị ẩn đi.

## Thêm khoảng thời gian

1. Sử dụng Microsoft Visual Studio, Eclipse hoặc một trình soạn thảo văn bản khác mở file duration.html, bạn có thể tìm thấy file này trong thư mục mã nguồn mẫu Chương 23 của Tài nguyên đi kèm.
2. Trong file này, thêm đoạn mã in đậm sau:

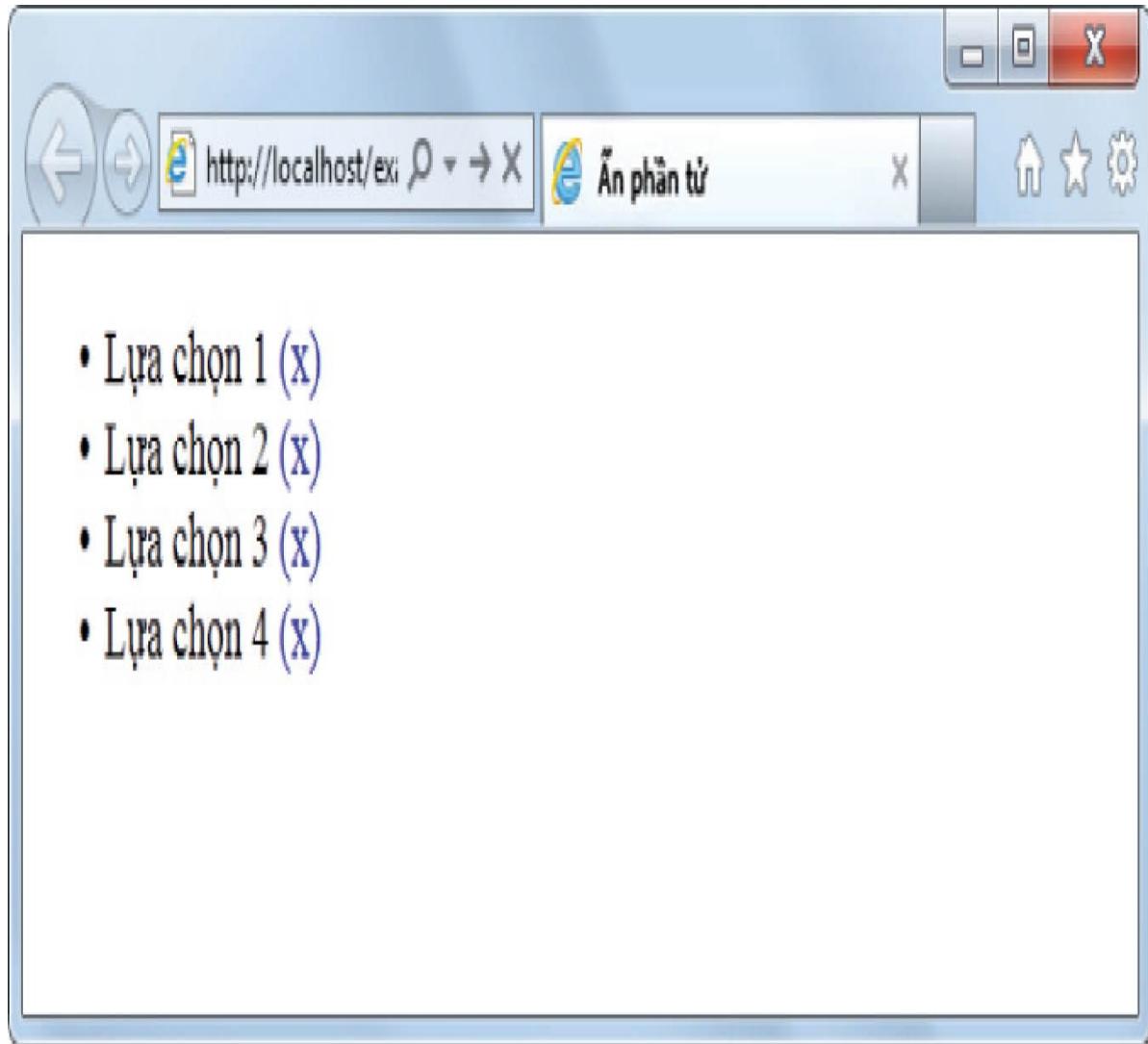
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ẩn phần tử</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
<style type="text/css">
.removeBtn {
 color: #0000CC;
}
</style>
</head>
<body>

<li id="option1">Lựa chọn 1
<li id="option2">Lựa chọn 2
<li id="option3">Lựa chọn 3
<li id="option4">Lựa chọn 4

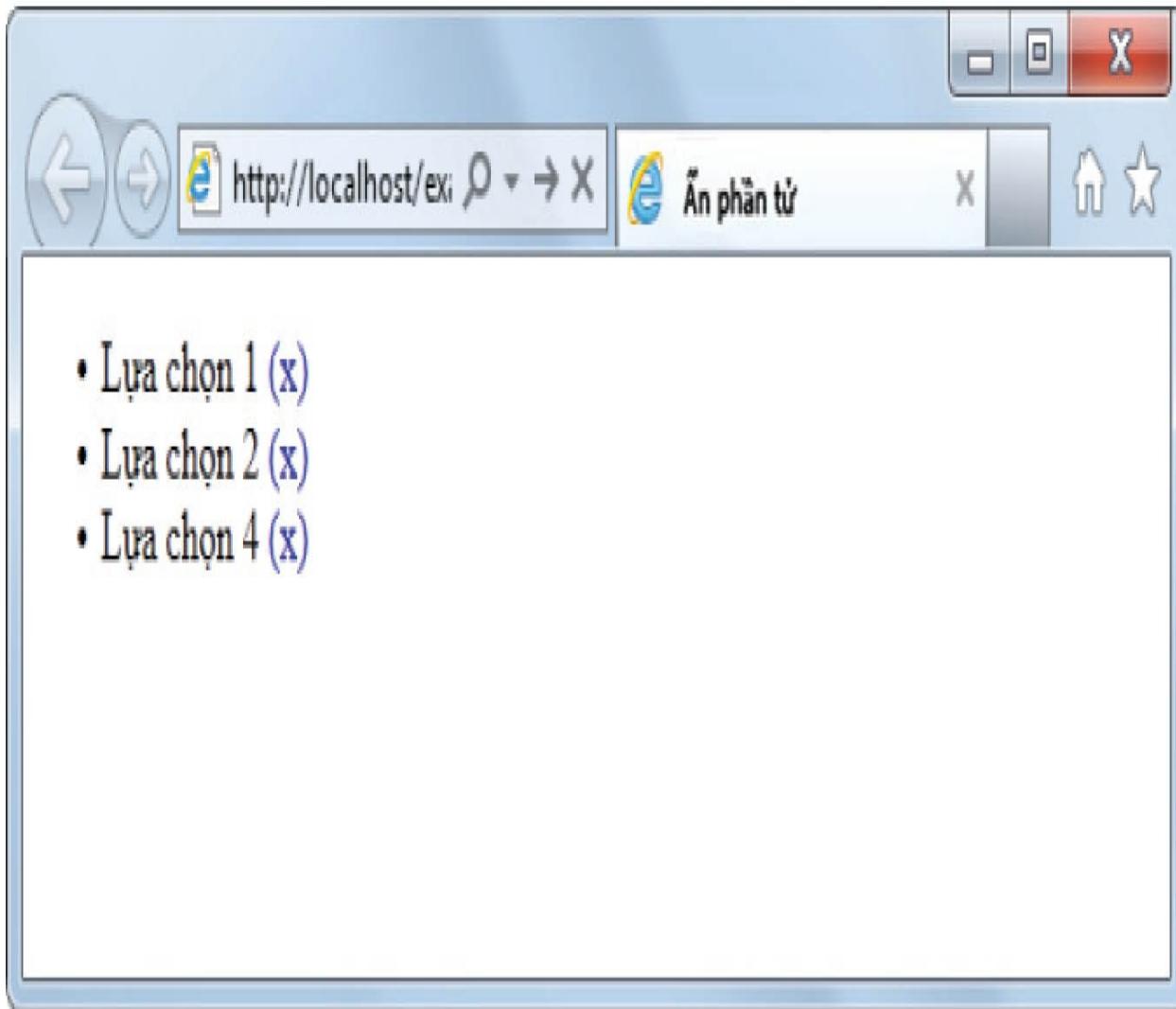
<script type="text/javascript">
$(document).ready(function() {
$('.removeBtn').each(function(elm) {
 $(this).click(function() {
 $(this).parent().hide(500);
 });
});
});
</script>
</body>
```

```
</html>
```

3. Lưu file và dùng trình duyệt mở trang web. Bạn sẽ thấy một trang web như sau:



4. Nhấn chuột vào chữ (x) bên cạnh phần tử bất kỳ, phần tử đó sẽ biến mất và bạn sẽ thấy trang web hiển thị như dưới đây:



Nếu đang dùng trình duyệt với tiện ích gỡ lỗi như tiện ích Firebug của Firefox, bạn có thể nhấn chuột phải vào một trong các phần tử option còn lại và chọn Inspect Element (kiểm tra phần tử), bạn sẽ thấy thuộc tính *display* của phần tử đã ẩn có giá trị bằng *none*.

Hiệu ứng hiện dần và mờ dần

Thêm đối số khoảng thời gian vào các hàm *.show()*, *.hide()* và *.toggle()* làm thay đổi độ mờ đục (opacity) của phần tử cho đến khi phần tử hiện hẵn hoặc ẩn hẵn. Tính năng tương tự có thể được thực hiện bằng hàm *.fadeIn()* và *.fadeOut()* với chức năng tương ứng là làm hiện dần và làm mờ dần. Xem <http://api.jquery.com/category/effects/fading> để tìm hiểu thêm về hai hàm này cũng như hàm *.fadeTo()*.

## Hiệu ứng trượt

Một phương thức khác để thiết lập thuộc tính *display* bằng *none* là sử dụng hàm *.slideUp()* và *.slideDown()*. Các hàm này tạo hiệu ứng trượt khiến phần tử dường như di chuyển trước khi biến mất, hoặc di chuyển trước khi xuất hiện. Hàm *.slideUp()* làm ẩn phần tử và *.slideDown()* làm phần tử hiện trở lại.

Thay đổi đoạn mã trong phần đầu chương để sử dụng hàm *.slideUp()* thay cho hàm *.hide()*:

```
$('.removeBtn').each(function(elm) {
 $(this).click(function() {
 $(this).parent().slideUp();
 });
});
```

**Chú ý** Khi đọc tên hàm, đừng nhầm lẫn rằng bạn có thể chọn trượt các phần tử lên hoặc trượt xuống. Hàm *.slideUp()* luôn làm ẩn các phần tử trong khi *.slideDown()* luôn làm hiện chúng.

## jQuery UI

jQuery UI được xây dựng dựa trên lõi của jQuery và cung cấp các tính năng mở rộng liên quan đến giao diện người dùng. Một số thành phần trong jQuery UI cung cấp các widget và hành vi cụ thể, bao gồm widget chọn ngày, hàm tạo hiệu ứng accordion và tự động điền theo từ gợi ý (auto-complete). Phần này trình bày một số tiện ích trong jQuery UI.

**Thông tin bổ sung** Để biết thêm thông tin về các tiện ích hiện có trong jQuery UI, hãy xem tại <http://jqueryui.com/>.

## Sử dụng jQuery UI

jQuery UI đòi hỏi phải tham chiếu một file JavaScript ngoài vào mã của bạn. Bạn có thể tải file này từ <http://jqueryui.com/>, tại đây bạn có thể tải bản đầy đủ và ổn định hoặc tự lựa chọn các thành phần cần thiết cho website. Chương này sử dụng bản phát hành đầy đủ của jQuery UI, nhưng với hầu hết các website, bạn nên tùy chỉnh gói jQuery UI tải về theo nhu cầu cụ thể sao cho nó chỉ bao gồm các thành phần cần thiết cho các hiệu ứng dùng trong website đó.

jQuery UI được đóng gói trong một file zip chứa cả jQuery lõi và mã jQuery UI cùng với các file CSS liên quan tới jQuery UI. Bạn cần giải nén file này vào thư mục mà web server có thể truy cập được, như thư mục *htdocs* hoặc *publichtml\_*. Về cơ bản, các file đó cần được lưu trong cùng thư mục với mã JavaScript và HTML mà bạn sử dụng suốt cuốn sách.

**Chú ý** Việc sử dụng các hàm của jQuery UI tiềm ẩn các vấn đề về khả năng truy cập, trừ khi bạn cung cấp cách thức thay thế để thực hiện cùng một tính năng.

## Hiệu ứng kéo thả



jQuery UI cung cấp hai hàm hỗ trợ việc di chuyển các phần tử bằng cách kéo thả, lần lượt là *.draggable()* và *.droppable()*. Hàm *.draggable()* cho phép người dùng sử dụng chuột để di chuyển một phần tử bên trong trang web. Xem đoạn mã trong Ví dụ 23-3 dưới đây.

### VÍ DỤ 23-3 Sử dụng hàm *.draggable()*.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Kéo phần tử</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
<script type="text/javascript" data-src="js/jquery-ui-1.8.4..>
</script>
<style type="text/css">

#container span {
 border: solid 1px black;
```

```

 padding: 3px;
 }
</style>
</head>
<body>
<div id="container">
Kéo tôi đi
</div>
<script type="text/javascript">
$(document).ready(function() {
 $('#container > span').draggable();
});
</script>
</body>
</html>

```

Đoạn mã tạo một phần tử `<p>` hỗ trợ tính năng kéo chuột bằng cách gọi hàm `.draggable()`:

```
$('#container > span').draggable();
```

**Chú ý** Ví dụ 23-3 giả định rằng bạn đã tải jQuery và các file jQuery UI xuống thư mục làm việc hiện hành – cùng thư mục lưu file HTML cho Ví dụ 23-3. Khi đó, mã jQuery UI sẽ được tải vào từ thư mục con có tên là `js/`.

Thử chạy Ví dụ 23-3 trên một trình duyệt. Bạn sẽ thấy có thể nhấn chuột và kéo phần tử `<p>` ở trên di chuyển xung quanh.

Bạn có thể sử dụng kết hợp hàm `.droppable()` với hàm `.draggable()` để tạo đích cho phần tử được kéo, như vậy bạn sẽ có thể thực hiện thao tác kéo thả các phần tử ngay trên màn hình. Ví dụ 23-4 mở rộng đoạn mã sử dụng hàm `.draggable()` ở Ví dụ 23-3 để bổ sung một phần tử `<div>` làm đích thả.

#### VÍ DỤ 23-4 Sử dụng hàm `.droppable()`.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Thả phần tử</title>

```

```

<script type="text/javascript" data-src="jquery-1.4.2.min.
<script type="text/javascript" data-src="js/jquery-ui-1.8.
<style type="text/css">

#container span {
 border: solid 1px black;
 padding: 3px;
}

#targetContainer {
 height: 200px;
 width: 200px;
 border: solid 1px black;
 background-color: #abacab;
 margin: 50px;
}
</style>
</head>
<body>
<div id="container">
Kéo và thả tôi
</div>
<div id="targetContainer">
</div>
<script type="text/javascript">
$(document).ready(function() {
 $('#container > span').draggable();
 $('#targetContainer').droppable({
 drop: function(event,ui) {
 alert("Phần tử được thả: " + ui.draggable.
 }
});
});
</script>
</body>
</html>

```

Điểm mấu chốt của đoạn mã là cách sử dụng hàm `.droppable()` với phần tử `<div>` có ID là `targetContainer`:

```

$('#targetContainer').droppable({
 drop: function(event,ui) {
 alert("Phần tử được thả: " + ui.draggable.te

```

```
});
}
```

Hàm `.droppable()` có thể xử lý một số sự kiện, cho phép bạn phản hồi khi một phần tử được kéo lên trên phần tử có thể làm đích thả (*over*), di chuyển ra ngoài phần tử có thể làm đích thả (*out*), khi một phần tử được thả xuống (*drop*, như trong ví dụ) và khi một phần tử được kích hoạt - được chọn hay nhận được `focus`, hoặc bỏ kích hoạt - bỏ chọn hay mất focus (*activate* và *deactivate*). Xem ví dụ về các sự kiện này tại <http://jqueryui.com/demos/droppable/>.

## Accordion

jQuery UI hỗ trợ tạo hiệu ứng accordion – hiệu ứng khiến các phần tử cuộn lên hoặc cuộn xuống chồng lên nhau. Yếu tố then chốt để tạo hiệu ứng accordion là tài liệu HTML phải đúng cú pháp và có bố cục phù hợp. Một ví dụ thường áp dụng hiệu ứng accordion là một nhóm các phần tử hay lựa chọn tương tự nhau.

Ví dụ 23-5 trình bày cách tạo hiệu ứng accordion đơn giản bằng HTML và JavaScript.

**VÍ DỤ 23-5** Hiệu ứng Accordion tạo bằng jQuery.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Accordion</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
<script type="text/javascript" data-src="js/jquery-ui-1.8.1.custom.min.js">
<style type="text/css">

#container {
 border: solid 1px black;
 padding: 3px;
}
.optionHead {
 border: solid 1px black;
 background-color: #abacab;
}
.optionDiv {
 border: solid 1px black;
 background-color: #eefeff;
 padding: 5px;
}
#optionContent {
 border: solid 1px black;
 background-color: #eefeff;
 padding: 5px;
}
#optionContent p {
 margin: 0;
}
#optionContent a {
 color: inherit;
 text-decoration: none;
}
```

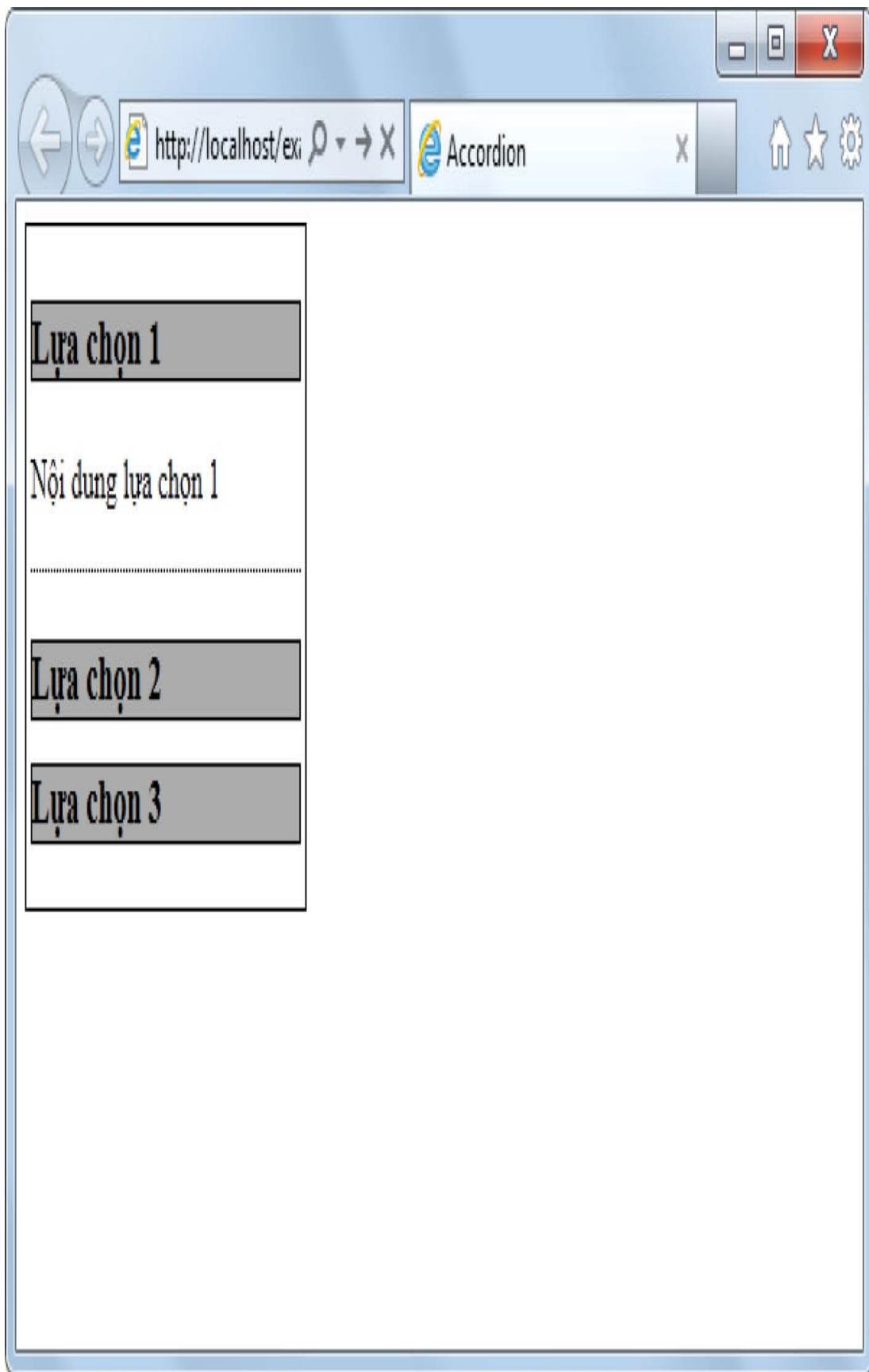
```
 border-bottom: dotted 1px black;
 }
</style>
</head>
<body>
<div id="container">

<h3 class="optionHead">Lựa chọn 1</h3>
<div class="optionDiv" id="option1">
<p>Nội dung lựa chọn 1</p>
</div>

<h3 class="optionHead">Lựa chọn 2</h3>
<div class="optionDiv" id="option2">
<p>Nội dung lựa chọn 2</p>
</div>

<h3 class="optionHead">Lựa chọn 3</h3>
<div class="optionDiv" id="option3">
<p>Nội dung lựa chọn 3</p>
</div>
</div>
<script type="text/javascript">
$(document).ready(function() {
$('#container').accordion();
});
</script>
</body>
</html>
```

Chạy đoạn mã trong Ví dụ 23-5 trên trình duyệt web sẽ cho một trang web như Hình 23-1.



## HÌNH 23-1 Hiệu ứng accordion cơ bản với jQuery.

Khi tạo accordion với jQuery, bạn sẽ thấy lựa chọn đầu tiên luôn mở sẵn khi trang web được tải. Tùy theo nhu cầu về bố cục và cách dùng accordion, bạn có thể muốn lựa chọn khác được mở sẵn khi trang web được tải, hoặc tất cả các lựa chọn được thu gọn tại thời điểm tải. Hàm `.accordion()` có một số tùy chọn để điều chỉnh hành vi của nó. Tùy chọn `active`, khi kết hợp với tùy chọn `collapsible` bằng `true` sẽ khởi tạo `accordion` trong trạng thái thu gọn hoặc với một lựa chọn nào đó được chọn sẵn.

Trong bài tập tiếp theo, bạn sẽ tạo hiệu ứng accordion cho trang web và thiết lập trạng thái mặc định cho hiệu ứng đó.

### Thiết lập trạng thái mặc định cho hiệu ứng accordion

1. Mở file `accordion.html` bằng một trình soạn thảo như Microsoft Visual Studio hoặc Eclipse. (Bạn có thể tìm thấy file này trong Tài nguyên đi kèm).
2. Trong file `accordion.html`, thêm đoạn mã in đậm trong đoạn mã sau:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ về hiệu ứng Accordion</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
<script type="text/javascript" data-src="js/jquery-ui-1.8.1.custom.min.js">
<style type="text/css">
```

```
\container {
 border: solid 1px black;
 padding: 3px;
}
.optionHead {
 border: solid 1px black;
 background-color: #abacab;
}
.optionDiv {
 border-bottom: dotted 1px black;
}
```

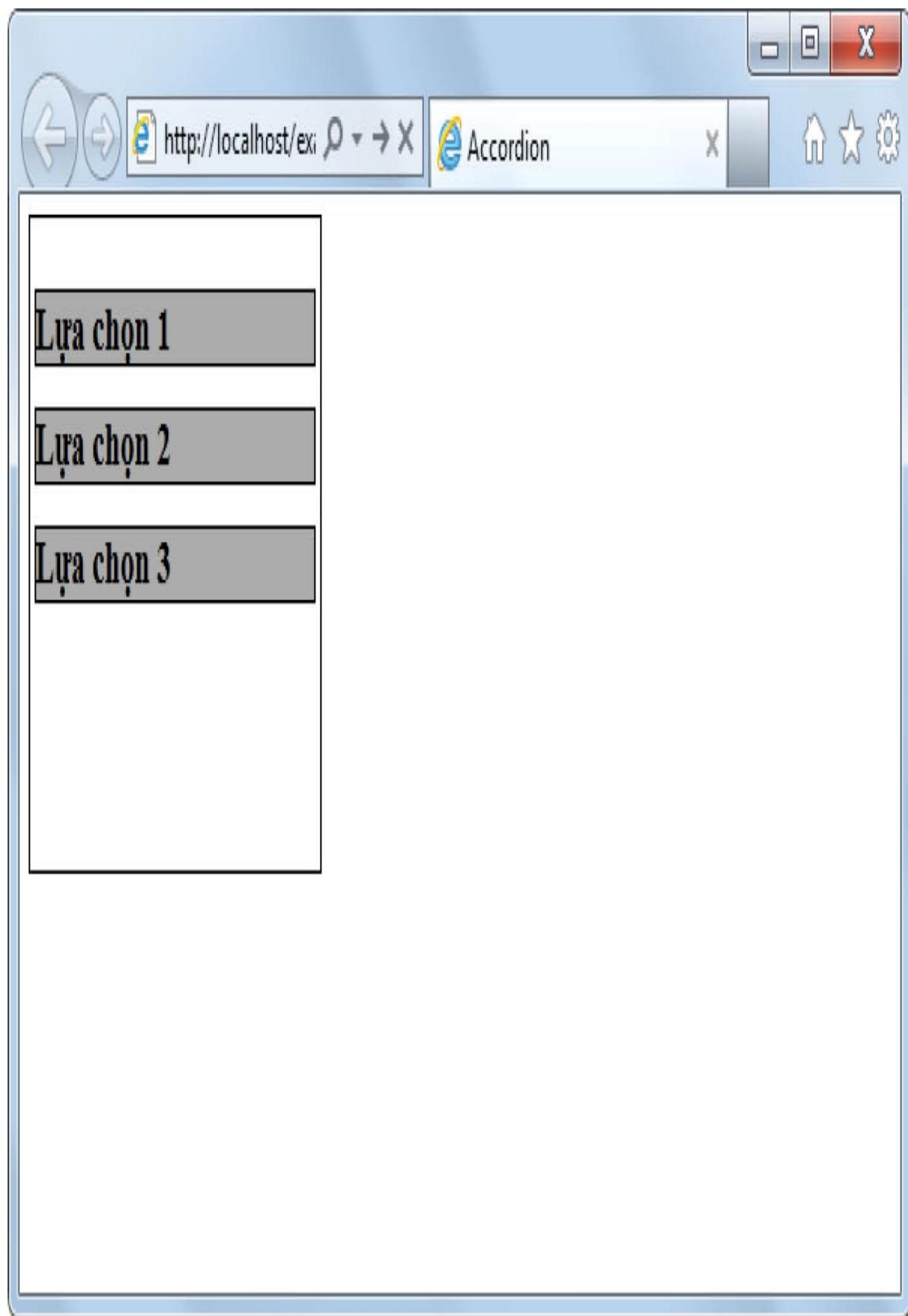
```
</style>
</head>
<body>
<div id="container">

<h3 class="optionHead">Lựa chọn 1</h3>
<div class="optionDiv" id="option1">
<p>Nội dung lựa chọn 1</p>
</div>

<h3 class="optionHead">Lựa chọn 2</h3>
<div class="optionDiv" id="option2">
<p>Nội dung lựa chọn 2</p>
</div>

<h3 class="optionHead">Lựa chọn 3</h3>
<div class="optionDiv" id="option3">
<p>Nội dung lựa chọn 3</p>
</div>
</div>
<script type="text/javascript">
$(document).ready(function() {
$('#container').accordion({
 collapsible: true,
 active: false
});
});
</script>
</body>
</html>
```

3. Lưu file và dùng trình duyệt mở trang web. Lưu ý rằng hiệu ứng accordion xuất hiện trong trạng thái thu gọn, như trong hình sau:



- Thay đổi giá trị của thuộc tính `active` thành 2 thay vì `false`. Đoạn mã sẽ giống như sau:

```
$('#container').accordion({
 collapsible: true,
 active: 2
});
```

- Nạp lại trang web trong trình duyệt. Bạn sẽ thấy lựa chọn thứ ba trên màn hình xuất hiện với trạng thái được mở rộng. Nguyên nhân là do chỉ số bắt đầu từ 0, vì vậy lựa chọn thứ nhất (hiển thị là Lựa chọn 1) thực chất có chỉ số bằng 0.

*Để tìm hiểu thêm các tùy chọn và sự kiện hiện có trong hàm tạo accordion, hãy xem tại <http://jqueryui.com/demos/accordion/>.*

## Tìm hiểu thêm về jQuery UI

Ngoài nội dung được trình bày trong chương này, còn rất nhiều thông tin về jQuery UI như khả năng tạo theme (chủ đề) CSS phức tạp để chọn các kiểu màu đồng bộ cho các hiệu ứng và tiện ích trong cả trang web. Có nhiều cuốn sách chỉ viết về jQuery UI. Bạn có thể tìm hiểu thêm về theme CSS và jQuery UI tại <http://jqueryui.com/themeroller/>. Ngoài ra, trang web <http://jqueryui.com> cũng có nhiều thông tin và bài hướng dẫn về jQuery UI.

## Bài tập

- Sử dụng đoạn mã trong Ví dụ 23-2 làm cơ sở, thêm một liên kết hoặc một button để làm hiển thị lại tất cả các lựa chọn bằng hàm `.show()`.
- Sử dụng kiến thức ở chương này và các chương trước để tạo hiệu ứng accordion bằng jQuery với các tùy chọn được tải qua AJAX.

## **ĐÔI NÉT VỀ FPT POLYTECHNIC**

Thành lập tháng 07/2010, FPT Polytechnic thuộc Đại học FPT, đào tạo Hệ Cao đẳng thực hành và cấp bằng Cao đẳng nghề theo Quyết định của Tổng cục dạy nghề.

[SỨ MỆNH][4]

**FPT Polytechnic ra đời với sứ mệnh cung cấp dịch vụ đào tạo tốt trên các tiêu chí: phù hợp với năng lực học tập của sinh viên; đáp ứng nhu cầu lớn của doanh nghiệp; và cung cấp dịch vụ đào tạo chuẩn mực dựa trên các chuẩn đã được công nhận.**

[Phù hợp với năng lực học tập của sinh viên ][5]

[Tất cả sinh viên đều có quyền học tập và có quyền được cung cấp một học vấn, kỹ năng phù hợp với năng lực. FPT Polytechnic cung cấp một chương trình học tập thiên về thực hành với mục tiêu chỉ cần sinh viên chăm chỉ, có ý thức học hỏi, cầu tiến thì sẽ đáp ứng được nhu cầu việc làm của doanh nghiệp. ][2]

[Đáp ứng nhu cầu lớn của doanh nghiệp ][6]

[Các chuyên ngành đào tạo của FPT Polytechnic đều nhắm tới nhu cầu xã hội lớn. Với nhận định Việt Nam có hàng trăm nghìn doanh nghiệp, mỗi doanh nghiệp đều cần có nhân viên kế toán, nhân viên tiếp thị và bán hàng. Các doanh nghiệp đều cần có website quảng bá sản phẩm và giao dịch do đó đều cần một nhân viên thiết kế, quản trị website. Ngoài ra, doanh nghiệp có hệ thống máy tính đều cần nhân viên để xây dựng các ứng dụng hữu ích và đảm bảo hệ thống được vận hành hiệu quả.] [2]

[Cung cấp dịch vụ đào tạo chuẩn mực ][7]

[Chương trình đào tạo của FPT Polytechnic tuân theo chuẩn khung chương trình của BTEC, Vương quốc Anh. Sách giáo trình, tài liệu học tập được chuyển ngữ sang Tiếng Việt từ các bộ sách uy tín của các Nhà xuất bản lớn trên thế giới như Pearson, Cengage, McGraw-Hill, ... Học liệu được các cán bộ có uy tín của Trường Đại học FPT thiết kế, biên tập và chuyển tải trên việc áp dụng hệ thống công nghệ thông tin.] [2]

[TRIẾT LÝ ĐÀO TẠO ][4]

FPT Polytechnic áp dụng triết lý đào tạo “Thực học – Thực nghiệp” với việc coi người học như là nhân viên. Trải nghiệm học tập sẽ được coi như trải nghiệm làm việc nhằm giúp cho doanh nghiệp làm quen được với công việc thực tiễn và phương pháp giải quyết vấn đề dựa trên công việc. Phương pháp đào tạo qua dự án “project based training” sẽ đưa các yêu cầu thực tế của công việc truyền đạt dưới dạng dự án giao cho sinh viên. Việc kỷ luật học tập cũng được áp dụng chặt chẽ cùng với các khóa đào tạo kỹ năng mềm nhằm đảm bảo sinh viên ra trường có được ý thức cũng như kỹ năng mềm trong công việc.

[CHIẾN LƯỢC TRONG VIỆC PHÁT TRIỂN GIÁO TRÌNH ][4]

Hiện nay FPT Polytechnic đã có bản quyền dịch và phát hành nhiều bộ sách giáo trình cho các chuyên ngành liên quan đến Công nghệ thông tin, Kế toán và Quản trị kinh doanh. Ngoài ra, FPT Polytechnic cũng đã phát triển các gói học liệu cho các giáo trình này. FPT Polytechnic sẵn sàng chia sẻ và cùng phát triển giáo trình, tài liệu để cung cấp chất lượng đào tạo tốt hơn cho sinh viên Việt Nam.

[![][photo1]][10]

[XIN VUI LÒNG LIÊN HỆ ][6]

[Văn phòng FPT Polytechnic Việt Nam ][6]

[Địa chỉ: Tòa nhà FPT Polytechnic, Đường Hàm Nghi, KĐT Mỹ Đình I, Từ Liêm, Hà Nội Điện thoại: (04) 7 305 9886 - (04) 7 308 0898 Email: caodang@fpt.edu.vn][2]

[Cơ sở tại Hà Nội ][5]

[Địa chỉ: Tòa nhà FPT Polytechnic, Đường Hàm Nghi, KĐT Mỹ Đình I, Từ Liêm, Hà Nội Điện thoại: (04) 8 582 0808 - (04) 6 287 1911 Email: caodangfpt.hn@fpt.edu.vn][2]

[Cơ sở tại Đà Nẵng ][5]

[Địa chỉ: 137 Nguyễn Thị Thập, Phường Hòa Minh, Quận Liên Chiểu, TP. Đà Nẵng Điện thoại: (0511) 3 710 999 Email: caodangfpt.dn@fpt.edu.vn][2]

[Cơ sở tại Tây Nguyên ][5]

[Tầng 2 tòa nhà VIB 27 Nguyễn Tất Thành, Phường Tân Lợi, TP. Buôn Ma Thuột, Tỉnh Đăk Lăk Điện thoại: (0500) 355 5678 Email: caodangfpt.daklak@fpt.edu.vn][2]

[Cơ sở tại TP. Hồ Chí Minh ][5]

[Địa chỉ: 391A Nam Kỳ Khởi Nghĩa, Q. 3, TP. HCM Điện thoại: (08) 3526 8799 Email: caodangfpt.hcm@fpt.edu.vn][2]

# Phụ lục

## Đáp án bài tập trong chương

Phụ lục này cung cấp đáp án và lời giải cho các bài tập được đưa ra trong cuốn sách này. Trong nhiều trường hợp, có nhiều hơn một cách giải quyết vấn đề. Vì vậy, trừ khi câu hỏi yêu cầu xử lý vấn đề theo cách thức cụ thể, bất cứ giải pháp nào cũng được chấp nhận. Lưu ý tên các hàm của bạn có thể sẽ khác với tên trong đáp án.

## Chương 1

1. Sai. Dù chuẩn JavaScript được soạn thảo bởi một tổ chức chuyên phát triển các chuẩn, ECMA International, song JavaScript vẫn không được hỗ trợ bởi tất cả các trình duyệt web. Mức độ hỗ trợ cũng khác biệt (đôi khi rất lớn) giữa các trình duyệt.
2. Sai. Có nhiều lý do khiến JavaScript bị vô hiệu hóa trên máy tính của khách truy cập website. Trình duyệt của họ không hỗ trợ JavaScript, họ dùng phần mềm chuyên dụng nào đó không hỗ trợ JavaScript, hoặc họ đơn thuần muốn tắt JavaScript vì lý do cá nhân. Bạn nên cố gắng để website vẫn hoạt động khi không có JavaScript.
3. Một khối lệnh định nghĩa JavaScript về cơ bản giống như sau:

```
<script type="text/javascript">
// Viết mã JavaScript ở đây
</script>
```
4. Sai. Phiên bản JavaScript không được khai báo trong định nghĩa DOCTYPE. Trên thực tế, chúng ta không mấy khi khai báo phiên bản JavaScript sẽ dùng. Đúng. Mã JavaScript có thể xuất hiện ở cả khối `<head>` và khối `<body>` của một tài liệu HTML.

# Chương 2

- Đoạn mã trong trang mysecondpage.htm sẽ tương tự như sau, dù trang của bạn có thể sẽ khác một chút (và dĩ nhiên nó sẽ chứa tên bạn thay vì tên tác giả!):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Trang web thứ hai của tôi</title>
<script type="text/javascript">
alert("Steve Suehring");
</script>
</head>
<body>
<p>Trang web thứ hai của tôi</p>
</body>
</html>
```

- Dưới đây là đoạn mã mới với phần thay đổi được in đậm:



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Trang web thứ hai của tôi</title>
<script type="text/javascript">
function callAlert() {
 alert("Steve Suehring");
}
</script>
</head>
<body>
<script type="text/javascript">
callAlert();
</script>
<p>Trang web thứ hai của tôi</p>
</body>
</html>
```

3. Tạo một file có tên là 3.htm và một file có tên là 3.js có nội dung như dưới đây. (Đoạn tham chiếu trong file 3.htm tới file 3.js được in đậm).

3.js:

```
function callAlert() {
 alert("Steve Suehring");
}
```

3.htm:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Trang web thứ hai của tôi</title>
<script type="text/javascript" data-src="3.js"> </script>
</head>
<body>
<script type="text/javascript">
callAlert();
</script>
<p>Trang web thứ hai của tôi</p>
</body>
</html>
```



## Chương 3

1. Các câu lệnh đúng là a, b, c và d. Câu lệnh sai duy nhất là e vì câu lệnh này sử dụng một từ khóa, *case*, làm tên biến.
2. Sai. Không phải tất cả câu lệnh JavaScript đều phải kết thúc bằng dấu chấm phẩy. Trên thực tế, dấu chấm phẩy thường không bắt buộc.
3. Biến *orderTotal* bị thay đổi sau khi người dùng nhận được hộp thoại thông báo số lượng hàng đã đặt, nhưng trước giá trị trả về của hàm. Bài học ở đây là bạn không được tùy tiện thay đổi giá trị hoặc nội dung của biến. Người dùng muốn đặt một số lượng hàng nhất định, nhưng đoạn mã đã thay đổi số lượng đó sau khi thông báo cho người dùng chính xác số lượng hàng

được đặt!

## Chương 4

1. Khai báo biến:

```
var first = 120;
var second = "5150";
var third = "Hai trăm ba mươi";
```

2. Mảng (các giá trị của bạn có thể khác, nhưng cần chú trọng kiểu dữ liệu và cú pháp):

```
var newArray = new Array(10, 20, 30, "chuoi thu nhat", "ct
```

3. Chuỗi đã thêm ký tự thoát:

```
alert("Phản ứng của Steve thật \"dễ thương!\"");
```

4. Đây là bài tập mở rộng, không có lời giải cụ thể.



## Chương 5

1. Các hộp thoại thông báo (giá trị của bạn có thể khác, nhưng cần chú trọng kiểu dữ liệu và cú pháp):

```
var num1 = 1;
var num2 = 1;
var num3 = 19;
var fourthvar = "84";
var name1 = "Jakob";
var name2 = "Edward";
alert(num1 + num2);
alert(num3 + fourthvar);
alert(name1 + name2);
```

2. Toán tử chèn sau:

```
var theNum = 1;
alert(theNum);
alert(theNum++);
alert(theNum);
```

Toán tử chèn trước:

```
var theNum = 1;
alert(theNum);
alert(++theNum);
alert(theNum);
```

3. Đoạn mã như sau:

```
var num1 = 1;
var num2 = 1;
var num3 = 19;
var fourthvar = "84";
var name1 = "Jakob";
var name2 = "Edward";
alert(typeof num1);
alert(typeof num2);
alert(typeof num3);
alert(typeof fourthvar);
alert(typeof name1);
alert(typeof name2);
```

Đoạn mã này sẽ hiển thị ba hộp thoại thông báo với từ *number*, theo sau là ba hộp thoại khác với từ *string*.

1. Sai. Các toán tử một ngôi xuất hiện khá thường xuyên trong JavaScript, đặc biệt trong vòng lặp *for*, biến đếm được tăng giá trị bằng toán tử `++` chèn sau.
2. Sai. Dù việc tiết kiệm một vài byte có thể có ích, đặc biệt cho ứng dụng web, giải pháp được ưu tiên hơn vẫn là tận dụng vài byte đó để mã dễ đọc và dễ bảo trì hơn. Tuy nhiên, phần lớn đây là vấn đề của thói quen và chuẩn lập trình. Trong phần sau, bạn sẽ được làm quen với jQuery. Phiên bản "thu nhỏ" của thư viện jQuery là một ví dụ cho việc tiết kiệm kích cỡ hết mức.

# Chương 6

- Thay thế *YOUR NAME* trong đoạn mã sau bằng nội dung thích hợp:

```
var inputName = prompt("Vui lòng nhập tên của bạn:");
switch(inputName) {
 case "YOUR NAME":
 alert("Chào mừng " + inputName);
 break;
 case "Steve":
 alert("Đi đi");
 break;
 default:
 alert("Vui lòng trở lại sau, " + inputName)
}
```

- Đoạn mã như sau:

```
var temp = prompt("Nhập nhiệt độ hiện tại:");
if (temp > 100) {
 alert("Vui lòng giảm nhiệt độ!");
} else if (temp < 20) {
 alert("Hãy tăng nhiệt độ lên!");
}
```

Lưu ý sẽ là một ý tưởng không tồi khi đưa ra hành động mặc định trong trường hợp nhiệt độ nằm trong khoảng từ 20 đến 100!

- Bài tập này thực chất không thể thực hiện được như yêu cầu. Vì toán tử ba ngôi chỉ chấp nhận một điều kiện kiểm tra duy nhất, trong khi bài tập 2 yêu cầu hai điều kiện. Do đó, toán tử ba ngôi không thể thực hiện chính xác yêu cầu đó. Đoạn mã sau tạo hộp thoại thông báo yêu cầu người dùng giảm nhiệt độ khi nhiệt độ cao hơn 100 và tăng nhiệt độ khi nhiệt độ thấp hơn hoặc bằng 100.

```
var temp = prompt("Nhập vào nhiệt độ hiện tại:");
temp > 100 ? alert("Hãy giảm nhiệt độ") : alert("Hãy tăng
```

- Đoạn mã như sau:

```

for (var i = 1; i < 101; i++) {
 if (i == 99) {
 alert("Giá trị số là " + i);
 }
}

```

Lưu ý: Vì biến *i* bắt đầu từ 1 (trong câu lệnh *for*), biến đếm này cần tăng đến 101 để đạt yêu cầu đếm từ 1 đến 100.

5. Đoạn mã như sau:

```

var i = 1;
while (i < 101) {
 if (i == 99) {
 alert("Giá trị số là " + i);
 }
 i++;
}

```

Chú ý việc đặt toán tử tăng chèn sau cho biến *i* trong vòng lặp. Bạn có thể sử dụng *i=i+1*, nhưng toán tử chèn sau là cách được ưa dùng hơn.



## Chương 7

1. Lưu ý rằng đoạn mã này dùng hàm *isNaN* để kiểm tra xem giá trị đầu vào có phải là số hay không. Cách lập trình này rất hay nhưng không phải lúc nào cũng được sử dụng. Cách làm khác cũng cho ra kết quả tương tự là sử dụng câu lệnh *return theNumber++* thay cho câu lệnh cuối cùng. Đoạn mã như sau.

```

<head>
 <title>Chương 7 Bài tập 1</title>
<script type = "text/javascript" >
function incrementNum(theNumber) {
 if (isNaN(theNumber)) {
 alert("Xin lỗi, " + theNumber + " không phải là số");
 return;
 }
 return theNumber + 1;
}

```

```
</script>
</head>
<body>
<script type = "text/javascript" >
alert(incrementNum(3));
</script>
</body>
```

2. Đoạn mã như sau:

```
function addNums(firstNum, secondNum) {
 if ((isNaN(firstNum)) || (isNaN(secondNum))) {
 alert("Xin lỗi, cả hai đối số đều phải là
 return;
 }
 else if (firstNum > secondNum) {
 alert(firstNum + "lớn hơn " + secondNum);
 }
 else {
 return firstNum + secondNum;
 }
}
```



3. Mục đích của bài tập này là chỉ ra vấn đề phạm vi của biến. Chú ý giá trị của biến *result* thay đổi như thế nào bên ngoài hàm – dù thay đổi chỉ được thực hiện bên trong hàm. Hai câu lệnh hiển thị hộp thoại thông báo được in đậm trong đoạn mã:

```
function addNumbers() {
 firstNum = 4;
 secondNum = 8;
 result = firstNum + secondNum;
 return result;
}
result = 0;
alert(result);
result = addNumbers();
alert(result);
```

4. Đoạn mã như sau:

```
<head>
<title>Chương 7 Bài tập 4</title>
```

```

<script type="text/javascript">
var stars = ["Polaris", "Aldebaran", "Deneb", "Vega", "Altair",
 "Dubhe", "Regulus"];
var constells = ["Ursa Minor", "Taurus", "Cygnus", "Lyra", "Aquila",
 "Ursa Major", "Leo"];
function searchStars(star) {
 var starLength = stars.length;
 for (var i = 0; i < starLength; i++) {
 if (stars[i] == star) {
 return constells[i];
 }
 }
 return "Không tìm thấy sao " + star;
}
</script>
</head>
<body>
<script type = "text/javascript" >
var inputStar = prompt("Nhập tên ngôi sao: ");
alert(searchStars(inputStar));
</script>
<p>Danh sách các ngôi sao</p>
</body>

```



## Chương 8

1. Đoạn mã như sau:

```

var star = ["Polaris", "Deneb", "Vega", "Altair"];
var starLength = star.length;
for (var i = 0; i < starLength; i++) {
 alert(star[i]);
}

```

2. Đây là một cách giải:

```

function Song(artist,length,title) {
 this.artist = artist;
 this.length = length;
 this.title = title;
}

```

```
}

song1 = new Song("Nghệ sĩ thứ nhất", "3:30", "Tên bài hát t"
song2 = new Song("Nghệ sĩ thứ hai", "4:11", "Tên bài hát thi"
song3 = new Song("Nghệ sĩ thứ ba", "2:12", "Tên bài hát thứ
```

3. Giả định bạn đang dùng đoạn mã cho sẵn, đoạn mã trong phần thân nối tên tất cả đối tượng vào một chuỗi, như sau:

```
var names = new Array;
for (var propt in star) {
 names += propt;
}
alert(names);
```

Đoạn mã để phân tách tên các đối tượng bằng dấu phẩy sẽ như sau:

```
var names = new Array;
for (var propt in star) {
 if (names != "") {
 names += "," + propt;
 } else {
 names = propt;
 }
}
alert(names);
```



## Chương 9

1. Đoạn mã như sau:

```
if (screen.availHeight < 768) {
 alert("Chiều cao khả dụng: " + screen.availHeight)
}
if (screen.availWidth < 1024) {
 alert("Chiều rộng khả dụng: " + screen.availWidth)
}
```

2. Đoạn mã hoàn chỉnh được đưa ra dưới đây, bao gồm cả đoạn mã trong bài tập. Phần mã thêm vào được in đậm. Lưu ý cách sử dụng hàm *unescape()* để loại bỏ ký tự mã hóa %20 (ký tự trống) theo kiểu URL khỏi tên quốc gia.

Điều này là cần thiết vì tên quốc gia trong bài tập này phải được giải mã khỏi định dạng URL để được xử lý trong yêu cầu HTTP GET.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Location, Location, Location</title>
</head>
<body>
 <script type="text/javascript">
 var body = document.getElementsByTagName('
 for (var prop in location) {
 var elem = document.createElement('
 var text = document.createTextNode('
 elem.appendChild(text);
 body.appendChild(elem);

 }
 if (location.search) {
 var querystring = location.search;
 var splits = querystring.split('&
 for (var i = 0; i < splits.length;
 var splitpair = splits[i];
 var elem = document.createElement('
 var text = document.createTextNode('
 if (splitpair[0] == "count") {
 switch(unescape(splitpair[1])) {
 case "Brazil":
 a
 case "Argentina":
 b
 case "Greece":
 a
 case "Bulgaria":
 b
 }
 }
 elem.appendChild(text);
 body.appendChild(elem);
 }
 }
 </script>
</body>
</html>
```

3. Bài tập này không có lời giải cụ thể. Bạn có thể cài đặt User Agent Switcher để hoàn thành bài tập này.

# Chương 10

1. Đoạn mã như sau:

```
var newelement = document.createElement("p");
newelement.setAttribute("id", "pelement");
document.body.appendChild(newelement);
newelement.appendChild(document.createTextNode(
 "Đây là một đoạn văn bản, dù ngắn."));
var anchorelem = document.createElement("a");
anchorelem.setAttribute("id", "aelement");
anchorelem.setAttribute("href", "http://www.braingia.org/");
document.body.appendChild(anchorelem);
anchorelem.appendChild(document.createTextNode(
 "Đi tới website của Steve Suehring."));
```

2. Đoạn mã như sau:



```
`// tạo các phần tử ban đầu (nếu bạn sử dụng một file HTML
var newelement = document.createElement("p");
newelement.setAttribute("id", "pelement");
document.body.appendChild(newelement);
newelement.appendChild(document.createTextNode(
 "Đây là một đoạn văn bản, dù ngắn."));
var anchorelem = document.createElement("a");
anchorelem.setAttribute("id", "aelement");
anchorelem.setAttribute("href", "http://www.braingia.org/");
document.body.appendChild(anchorelem);
anchorelem.appendChild(document.createTextNode("Nhấn vào đây để
// thực hiện thay đổi
var existingp = document.getElementById("pelement");
existingp.firstChild.nodeValue="Đây là văn bản mới.";
var newanchor = document.getElementById("aelement");
newanchor.setAttribute("href", "http://www.microsoft.com/");`
```

3. Đoạn mã như sau:

```

<head>
<title> Bài tập Chương 10 </title>
</script>
</head>
<body>
<div id="thetable"></div>
<script type = "text/javascript" >
var table = document.createElement("table");
table.border = "1";
var tbody = document.createElement("tbody");
// Nối thêm phần body vào bảng
table.appendChild(tbody);
var row = document.createElement("tr");
// Tạo các dòng trong bảng
for (i = 1; i < 3; i++) {
 var row = document.createElement("tr");
 // Tạo các cột/td
 for (j = 1; j < 3; j++) {
 // Chèn dữ liệu thực từ tài liệu XML
 var td = document.createElement("td");
 var data = document.createTextNode("Xin chào bạn, bạn đang ở hàng " + i + ", Cột " + j);
 td.appendChild(data);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}
document.getElementById("thetable").appendChild(table);
</script>
</body>

```

# Chương 11

1. Đoạn mã như sau:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Onclick</title>

```

```
<script type="text/javascript">
function handleclick() {
 alert("Bạn đã nhấp chuột vào đây");
 return false;
}
</script>
</head>
<body>
<p>Nhấp vào đây</p>
</body>
</html>
```

2. Đoạn mã như sau:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Onlick</title>
<script type="text/javascript" data-src="ehandler.js"></script>
<script type="text/javascript">
function handleclick() {
 alert("Bạn đã nhấp chuột vào đây");
 return false;
}
</script>
</head>
<body>
<p>Nhấp vào đây</p>
<script type="text/javascript">
var aLink = document.getElementById("clickMe");
EHandler.add(aLink, "click", function() { handleclick(); })
</script>
</body>
</html>
```

3. Bài tập này không cần dùng JavaScript. Đoạn mã HTML sẽ giống như sau:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Tab mới</title>
```

```
</head>
<body>
<p>
</body>
</html>
```

## Chương 12

- Đây là biến thể của một ví dụ trong chương này:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Xin chào Cookie</title>
<script type = "text/javascript">
var cookName = "cookie1";
var cookVal = "testvalue";
var date = new Date();
date.setTime(date.getTime() + 86400000); // một ngày, tính từ
var expireDate = date.toGMTString();
var myCookie = cookName + "=" + cookVal + ";expires=" + e
document.cookie = myCookie;
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>
```

- Đoạn mã này về cơ bản giống với đoạn mã trong Bài tập 1, một vài thay đổi được in đậm:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Xin chào Cookie</title>
<script type = "text/javascript">
var cookName = "cookie2";
```

```

var cookVal = "testvalue";
var date = new Date();
date.setTime(date.getTime() + 86400000); // một ngày, tính 1
var expireDate = date.toGMTString();
var myCookie =
 cookName + "=" + cookVal + ";expires=" + expireDate;
document.cookie = myCookie;
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>

```

3. Trừ khi bạn đang sử dụng một kết nối dạng Secure Sockets Layer (SSL), nếu không, bạn sẽ không thể đọc cookie có cờ *secure*.
4. Bài tập 1 có sử dụng một cookie tên là *cookie1*; vì vậy, bài tập này sẽ chỉ hiển thị thông tin của cookie đó. Đoạn mã như sau:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Đọc Cookie</title>
<script type = "text/javascript">
var incCookies = document.cookie.split(";");
for (var c = 0; c < incCookies.length; c++) {
 var splitCookies = incCookies[c].split("=");
 if (splitCookies[0] == "cookie1") {
 alert(incCookies[c]);
 }
}
</script>
</head>
<body>
<p>Xin chào</p>
</body>
</html>

```

# Chương 13

1. Tham khảo Ví dụ 13-2 để thực hiện bài tập này.
2. Tham khảo Ví dụ 13-2 để thực hiện việc tải trước ảnh. Bạn có thể sử dụng cùng đoạn mã đó cho bản đồ ảnh bạn thực hiện trong bài tập này.

# Chương 14

1. Thao khảo phần “Làm việc với các Select Box ” trong Chương 11 để thực hiện bài tập này.
2. Dựa trên ví dụ pizza.htm, phần `<head>` của đoạn mã sẽ giống như sau với phần thay đổi được in đậm:

```
<head>
 <title>Pizza</title>
 <script type = "text/javascript">
 function prepza() {
 var checkboxes = document.forms["pizzaform"];
 var crusttype = document.forms["pizzaform"];
 var size = document.forms["pizzaform"].size;
 var crustlength = crusttype.length;
 var sizelength = crusttype.length;
 var newelement = document.createElement("p");
 newelement.setAttribute("id", "orderheading");
 document.body.appendChild(newelement);
 newelement.appendChild(document.createTextNode(
 "Chiếc bánh pizza này sẽ có:"));
 for (var c = 0; c < crustlength; c++) {
 if (crusttype[c].checked) {
 var newelement = document.createElement("p");
 newelement.setAttribute("id", "crusttype");
 document.body.appendChild(newelement);
 newelement.appendChild(document.createTextNode(
 "Đế bánh: " +crusttype[c].value));
 }
 }
 }
 </script>

```

```

 }
 for (var s = 0; s < size.length; s++) {
 if (size[s].checked) {
 var newelement = document.createElement("p");
 document.body.appendChild(newelement);
 newelement.appendChild(document.createTextNode("Cỡ " + size[s].value));
 }
 }
 for (var i = 0; i < checkboxes; i++) {
 if (document.forms[0].checkboxes[i].checked) {
 var newelement = document.createElement("p");
 document.body.appendChild(newelement);
 newelement.appendChild(document.createTextNode("Làm " + checkboxes[i].value));
 }
 }
 }
}
</script>
</head>

```



Đoạn HTML có dạng như sau. Cách giải này sử dụng một bảng có ba cột, lưu ý rằng đây không phải cách làm duy nhất. Phần thêm vào được in đậm:

```

<form id="pizzaform" action="#">
<table>
<tr><td>Thành phần</td><td>Đế bánh</td><td>Kích thước</td>
<tr>
<td><input type="checkbox" id="topping1" value="Xúc xích 1" checked="" name="toppingcheck"/>Xúc xích thường</td>
<td><input type="radio" name="crust" value="Bình thường" checked="checked" id="radio1"/>Bình thường</td>
<td><input type="radio" name="size" value="Nhỏ" checked="checked" id="radiosize1"/>Nhỏ</td>
</tr>
<tr>
<td><input type="checkbox" id="topping2" value="Xúc xích 2" name="toppingcheck"/>Xúc xích Ý</td>
<td><input type="radio" name="crust" value="Dày" id="radio2"/>Dày</td>
<td><input type="radio" name="size" value="Trung bình" id="radiosize2"/>Trung bình</td>

```

```

</tr>
<tr>
<td><input type="checkbox" id="topping3" value="Dăm bông"
 name="toppingcheck" />Dăm bông</td>
<td><input type="radio" name="crust" value="Mỏng" id="rad:</td>
<td><input type="radio" name="size" value="Lớn" id="radios:</td>
</tr>
<tr>
<td><input type="checkbox" id="topping4" value="Hạt tiêu"
 name="toppingcheck"/>Hạt tiêu xanh</td>
<td></td>
<td></td>
</tr>
<tr>
<td><input type="checkbox" id="topping5" value="Nấm"</td>
<td></td>
<td></td>
</tr>
<tr>
<td><input type="checkbox" id="topping6" value="Hành"
 name="toppingcheck" />Hành</td>
<td></td>
<td></td>
</tr>
<tr>
<td><input type="checkbox" id="topping7" value="Dứa" />Dứa</td>
<td></td>
<td></td>
</tr>
</table>
<p><input type="submit" id="prepBtn" name="prepBtn" value="Tạo pizza" /></p>
</form>

```



3. Thêm đoạn mã sau vào phần `<head>` của ứng dụng pizza trong bài tập trước:

```

function flip(pizzatype) {
 if (pizzatype == "veggiespecial") {
 document.getElementById("peppers").checked = true;
 document.getElementById("onions").checked = true;
 document.getElementById("mushrooms").checked = true;
 } else if (pizzatype == "meatspecial") {
 document.getElementById("sausage").checked = true;
 document.getElementById("pepperoni").checked = true;
 }
}

```

```

 document.getElementById("ham").checked = 'checked'
 } else if (pizzatype == "hawaiian") {
 document.getElementById("ham").checked = 'checked'
 document.getElementById("pineapple").checked = 'checked'
 }
}

```

Sử dụng form HTML sau. (Lưu ý phần thêm ba button và phần thay đổi thuộc tính *id* của mỗi thành phần).

```

<form id="pizzaform" action="#">

<input type="button" id="veggiespecial" name="veggiespecial"
 value="Rau đặc biệt" />
<input type="button" id="meatspecial" name="meatspecial"
 value="Thịt đặc biệt" />
<input type="button" id="hawaiian" name="hawaiian" value='checked' />

<table>
<tr><td>Thành phần</td><td>Đế bánh</td><td>Kích thước</td>
<tr>
<td><input type="checkbox" id="sausage" value="Xúc xích tẩm ướp" name="toppingcheck" />Xúc xích thường</td>
<td><input type="radio" name="crust" value="Bình thường" checked="checked" id="radio1" />Bình thường</td>
<td><input type="radio" name="size" value="Nhỏ" id="radiosize1" />Nhỏ</td>
</tr>
<tr>
<td><input type="checkbox" id="pepperoni" value="Xúc xích Ý" name="toppingcheck" />Xúc xích Ý</td>
<td><input type="radio" name="crust" value="Dày" id="radio2" />Dày</td>
<td><input type="radio" name="size" value="Trung bình" id="radiosize2" />Trung bình</td>
</tr>
<tr>
<td><input type="checkbox" id="ham" value="Dăm bông" name="toppingcheck" />Dăm bông</td>
<td><input type="radio" name="crust" value="Mỏng" id="radio3" />Mỏng</td>
<td><input type="radio" name="size" value="Lớn" id="radiosize3" />Lớn</td>
</tr>
<tr>
<td><input type="checkbox" id="peppers" value="Hạt tiêu xanh" name="toppingcheck" />Hạt tiêu xanh</td>
<td><input type="radio" name="crust" value="Tròn" id="radio4" />Tròn</td>
<td><input type="radio" name="size" value="Huge" id="radiosize4" />Huge</td>
</tr>

```

```

<td></td>
</tr>
<tr>
<td><input type="checkbox" id="mushrooms" value="Nấm"
 name="toppingcheck"/>Nấm</td>
<td></td>
<td></td>
</tr>
<tr>
<td><input type="checkbox" id="onions" value="Hành"
 name="toppingcheck" />Hành</td>
<td></td>
<td></td>
</tr>
<tr>
<td><input type="checkbox" id="pineapple" value="Dứa"
 name="toppingcheck"/>Dứa</td>
<td></td>
<td></td>
</tr>
</table>

<input type="submit" id="prepBtn" name="prepBtn"
 value="Chuẩn bị Pizza" onclick="prepza();"/>
</form>

```



**Thêm các hàm xử lý sự kiện vào đoạn mã JavaScript trong phần <body> của trang web:**

```

var veggieBtn = document.getElementById("veggiespecial");
EHandler.add(veggieBtn,"click",function() { flip("veggiespecial") });
var meatBtn = document.getElementById("meatspecial");
EHandler.add(meatBtn,"click",function() { flip("meatspecial") });
var hawaiiBtn = document.getElementById("hawaiispecial");
EHandler.add(hawaiiBtn,"click",function() { flip("hawaiispecial") });

```

# Chương 15

- Đây là một trang ví dụ; có nhiều cách để thực hiện bài tập này:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>CSS</title>
<link href="exercise1.css" rel="stylesheet" type="text/css">
</head>
<body>
<h1 id="h1element">Tiêu đề</h1>
<p id="firstelement">Phần tử thứ nhất</p>
<p id="secondelement">Phần tử thứ hai</p>
</body>
</html>
```

Sau đây là file style sheet exercise1.css:

```
#h1element {
background-color: #abacab;
}
```

**firstelement {**

```
color: red;
}
```



**secondelement {**

```
color: blue;
}
```

- Đoạn mã này thay đổi phần tử có tên *firstelement* để phần tử đó có font chữ màu xanh dương:

```
<script type="text/javascript">
var element1 = document.getElementById("firstelement");
element1.style.color = "#0000FF";
</script>
```

- Đoạn mã này ẩn tất cả phần tử *<p>* bằng thuộc tính *visibility* của CSS:

```
<script type="text/javascript">
var pelements = document.getElementsByTagName("p");
var pelmLength = pelements.length;
for (var i = 0; i < pelmLength; i++) {
 pelements[i].style.visibility = "hidden";
}
</script>
```

- Đoạn mã này cho thấy thuộc tính hiển thị (*visibility*) trước và sau khi bị thay đổi trong đoạn mã. Khi chạy đoạn mã, bạn sẽ thấy thông báo trống trước khi thuộc tính được thiết lập giá trị.

```
<script type="text/javascript">
var pelements = document.getElementsByTagName("p");
var pelmLength = pelements.length;
for (var i = 0; i < pelmLength; i++) {
 alert(pelements[i].style.visibility);
 pelements[i].style.visibility = "hidden";
 alert(pelements[i].style.visibility);
}
</script>
```



## Chương 16

- Ví dụ 16-3 là một cách giải cho bài tập này.
- Hộp thoại thông báo *alert* cung cấp phản hồi trực quan và cũng là lời giải cho bài tập này. Sau đây là cách giải cơ bản cho bài tập này.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Try/Catch</title>
<script type="text/javascript">
</script>
</head>
<body>
<form name="formexample" id="formexample" action="#">
<div id="citydiv">Nhập tên một thành phố: <input id="city'>
```

```

<div><input id="submit" type="submit"></div>
</form>
<script type="text/javascript">
function checkValid() {
 try {
 var cityField = document.forms[0]["city"];
 if (cityField.value != "Stockholm") {
 throw "Không phải là Stockholm";
 }
 }
 catch(errorObject) {
 alert(errorObject);
 }
}
function init() {
 document.forms[0].onsubmit = function() { return c
 }
}
window.onload = init;
</script>
</body>
</html>

```



3. Đây là một cách giải cho bài tập này. Còn có những cách giải khác trong đó có cách dùng file ehandler.js:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Try/Catch</title>
<script type="text/javascript">
</script>
</head>
<body>
<form name="formexample" id="formexample" action="#">
<div id="citydiv">Nhập một số trong khoảng từ 1 đến 100:
<input id="num" name="num"></div>
<div><input id="submit" type="submit"></div>
</form>
<script type="text/javascript">
function checkValid() {
 try {
 var numField = document.forms[0]["num"];

```

```

 if (isNaN(numField.value)) {
 throw "Không phải số";
 }
 if ((numField.value > 100) || (numField.value < 0)) {
 numField.style.background = "#FF0000";
 return false;
 }
 else {
 numField.style.background = "#FFFF00";
 return true;
 }
 }
 catch(errorObject) {
 alert(errorObject);
 }
 finally {
 alert("Cảm ơn bạn đã tham gia.");
 }
}
function init() {
 document.forms[0].onsubmit = function() { return true; }
}
window.onload = init;
</script>
</body>
</html>

```



## Chương 17

- Bài giải này dùng lại hai file books.htm và books.xml trong Chương 17 và chỉ thay đổi file books.htm. Các thay đổi được in đậm.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script type="text/javascript" data-src="ehandler.js"></script>
<title>Books</title>
</head>
<body>

```

```
<div id="xmldata"></div>
<p>Hiển thị bảng</p>
<script type="text/javascript">
function displayData() {
 var xmlEl = docObj.getElementsByTagName("book");
 var table = document.createElement("table");
 table.border = "1";
 var tbody = document.createElement("tbody");
 // Nối thêm phần thân vào bảng
 table.appendChild(tbody);
 var row = document.createElement("tr");
 for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType != 1)
 continue;
 }
 var tableHead = document.createElement("th");
 var colName = document.createTextNode(
 xmlEl[0].childNodes[colHead].nodeValue);
 tableHead.appendChild(colName);
 row.appendChild(tableHead);
}
tbody.appendChild(row);
// Tạo dòng trong bảng
for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo cột cho bảng /td
 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 // Bỏ qua nếu kiểu không phải là 1
 if (xmlEl[i].childNodes[j].nodeType != 1)
 continue;
 }
 // Chèn văn bản/dữ liệu thực từ tài liệu
 var td = document.createElement("td");
 var xmlData = document.createTextNode(
 xmlEl[i].childNodes[j].nodeValue);
 td.appendChild(xmlData);
 row.appendChild(td);
}
tbody.appendChild(row);
}
document.getElementById("xmldata").appendChild(table);
```

```

function getXML()
{
 tablelink.style.visibility = "hidden";
 if (typeof document.implementation.createDocument
 {
 docObj = document.implementation.createDoc
 docObj.onload = displayData;
 }
 else if (window.ActiveXObject)
 {
 docObj = new ActiveXObject("Microsoft.XMLD
 docObj.onreadystatechange = function () {
 if (docObj.readyState == 4) displa
 };
 }
 docObj.load("books.xml");
}
var tablelink = document.getElementById("displaytable");
EHandler.add("tablelink","click",function() { getXML(); })
</script>
</body>
</html>

```



**Mở rộng:** Đoạn mã sau thêm liên kết "*Hiển thị bảng*" để khi bảng hiển thị, sẽ có một liên kết "*Ẩn bảng*". Đây không phải là yêu cầu của bài tập.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<script type="text/javascript" data-src="ehandler.js"></sc
<title>Books</title>
</head>
<body>
<div id="xmldata"></div>
<p>Hiển thị bảng</p>
<script type="text/javascript">
function displayData() {
 var xmlel = docObj.getElementsByTagName("book");
 var table = document.createElement("table");
 table.setAttribute("id", "bookstable");
 table.border = "1";

```

```
var tbody = document.createElement("tbody");
// Nối thêm phần body vào bảng
table.appendChild(tbody);
var row = document.createElement("tr");
for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType == 1)
 continue;
}
var tableHead = document.createElement("thead");
var colName = document.createTextNode(xmlEl[0].childNodes[0].nodeValue);
tableHead.appendChild(colName);
row.appendChild(tableHead);
}
tbody.appendChild(row);
// Tạo dòng trong bảng
for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các cột cho bảng/td
 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 // Bỏ qua nếu kiểu không phải là text
 if (xmlEl[i].childNodes[j].nodeType == 1)
 continue;
 }
 // Chèn văn bản/dữ liệu thực từ tài liệu XML
 var td = document.createElement("td");
 var xmlData = document.createTextNode(xmlEl[i].childNodes[j].nodeValue);
 td.appendChild(xmlData);
 row.appendChild(td);
}
tbody.appendChild(row);
}
var tableanchor = document.createElement("a");
var tableanchortext = document.createTextNode("Ẩn");
tableanchor.setAttribute("id","hidetable");
tableanchor.setAttribute("href","");
tableanchor.appendChild(tableanchortext);
if (typeof window.addEventListener != "undefined")
 tableanchor.addEventListener("click",hideTable);
} else {
 tableanchor.attachEvent("onclick",hideTable);
}
```

```

}
document.getElementById("xmldata").appendChild(tableanchor);
document.getElementById("xmldata").appendChild(table); **

}

function hideTable() {
var bookstable = document.getElementById("bookstable");
bookstable.style.display = "none";
tablelink.style.display = "";
var tableanchor = document.getElementById("hidetable");
tableanchor.style.display = "none";
}
function getXML()
{
tablelink.style.display = "none";
if (typeof document.implementation.createDocument != "undefined")
{
docObj = document.implementation.createDocument("", "", null);
docObj.onload = displayData;
}
else if (window.ActiveXObject)
{
docObj = new ActiveXObject("Microsoft.XMLDOM");
docObj.onreadystatechange = function () {
if (docObj.readyState == 4) displayData()
};
}
docObj.load("books.xml");
}
var tablelink = document.getElementById("displaytable");
EHandler.add("tablelink","click",function() { getXML(); });
</script>
</body>
</html>

```



2. Cách giải này cũng cần đến file books.xml. Đoạn mã này gần giống đoạn mã hoàn chỉnh cho file books.htm trong Chương 17, các phần thay đổi được in đậm:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
```

```
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Books</title>
</head>
<body>
<div id="xmlData"></div>
<script type="text/javascript">
window.onload = getXML;
function displayData() {
 var xmlEl = docObj.getElementsByTagName("book");
 var table = document.createElement("table");
 table.border = "1";
 var tbody = document.createElement("tbody");
 // Nối thêm phần body vào bảng
 table.appendChild(tbody);
 var row = document.createElement("tr");
 for (colHead = 0; colHead < xmlEl[0].childNodes.length; colHead++) {
 if (xmlEl[0].childNodes[colHead].nodeType == 1)
 continue;
 }
 var tableHead = document.createElement("thead");
 var colName = document.createTextNode(xmlEl[0].childNodes[0].nodeValue);
 tableHead.appendChild(colName);
 row.appendChild(tableHead);
}
tbody.appendChild(row);
// Tạo dòng trong bảng
for (i = 0; i < xmlEl.length; i++) {
 var row = document.createElement("tr");
 // Tạo các cột trong bảng/td
 for (j = 0; j < xmlEl[i].childNodes.length; j++) {
 // Bỏ qua nếu kiểu không phải là text
 if (xmlEl[i].childNodes[j].nodeType == 1)
 continue;
 }
 // Chèn văn bản/dữ liệu thực từ tài liệu XML
 var td = document.createElement("td");
 if (i % 2) {
 td.style.backgroundColor = "#a6c9e9";
 }
 row.appendChild(td);
}
tbody.appendChild(row);
}

```

```
 var xmlData = document.createTextNode(xmlEl[i].text);
 td.appendChild(xmlData);
 row.appendChild(td);
 }
 tbody.appendChild(row);
}
document.getElementById("xmldata").appendChild(tbody);
}

function getXML()
{
 if (typeof document.implementation.createDocument
 {
 docObj = document.implementation.createDocument("", "", "xml");
 docObj.onload = displayData;
 }
 else if (window.ActiveXObject)
 {
 docObj = new ActiveXObject("Microsoft.XMLDOM");
 docObj.onreadystatechange = function () {
 if (docObj.readyState == 4) displayData();
 };
 docObj.load("books.xml");
 }
}
</script>
</body>
</html>
```



## Chương 18

Không có bài tập trong Chương 18.

## Chương 19

1. Không có phương thức HTTP nào đề cập trong chương này cung cấp khả năng bảo mật tốt hơn. Chỉ có việc bổ sung SSL mới cung cấp một tầng bảo

mật lên trên các phương thức HTTP. Cần lưu ý rằng việc sử dụng phương thức *POST* không thực sự làm ẩn dữ liệu đầu vào và SSL chỉ nên dùng với phương thức *POST* vì phương thức *GET* đặt các tham số trực tiếp vào URL và luôn hiện bất kể SSL có được dùng hay không.

2. Các phản hồi sử dụng HTML chuẩn có thể được truy xuất thông qua phương thức *responseText* và có thể chứa bất kỳ nội dung nào được truyền qua HTTP. Các phản hồi XML được truy xuất thông qua phương thức *responseXML* và phải được server xử lý dưới dạng XML. Các phản hồi JSON là các phản hồi dạng JavaScript; vì thế, chúng có những ưu thế nhất định về hiệu suất so với các phương thức khác.
3. Đáp án cho bài tập đã được trình bày trong nội dung chương, nhưng trong lời giải dưới đây, chúng ta sẽ sử dụng lời gọi hàm không đồng bộ (thay thế giá trị *YOUR SERVER* tương ứng với môi trường của bạn):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Không đồng bộ</title>
</head>
<body>
<div id="xmldata"></div>
<script type="text/javascript">
function readyAJAX() {
try {
return new XMLHttpRequest();
} catch(e) {
try {
return new ActiveXObject('Msxml2.XMLHTTP');
} catch(e) {
try {
return new ActiveXObject('Microsoft.XMLHTTP');
} catch(e) {
return "Bạn cần dùng một trình duyệt mới hơn.";
}
}
}
}
```



```

}

var requestObj = readyAJAX();
var url ="http://YOUR SERVER/sum.php?num1=2&num2=2";
requestObj.open("GET",url,true);
requestObj.send();
requestObj.onreadystatechange = function() {
if (requestObj.readyState == 4) {
if (requestObj.status == 200) {
alert(requestObj.responseText);
} else {
alert(requestObj.statusText);
}
}
}
}

</script>
</body>
</html>

```

File sum.php là một chương trình phía server viết bằng PHP, rất nhỏ và không có bảo mật:



```

<?php
print $_GET['num1'] + $_GET['num2'];
?>

```

## Chương 20

- Cách giải này sử dụng Ví dụ 20-1 và bổ sung thêm button Submit vào form. Form sẽ giống như sau:

```

<form name="nameform" id="nameform" action="" method="GET"
Nhập tên Bang: <input id="textname" type="text" name="text"
<input type="submit" name="submit" id="statesubmit">
</form>

```

Ngoài việc liên kết đến file ehandler.js, chúng ta cần khai báo một hàm xử lý sự kiện và hàm mới. Dưới đây là đoạn mã thêm vào đoạn JavaScript đã tồn tại trên trang.

```
var formSubmit = document.getElementById("nameform");
EHandler.add("formSubmit", "submit", function() { showstate();
function showstate() {
 alert(document.forms[0].textname.value);
}
}
```

2. Cách giải cho bài tập này gần giống cách giải của bài tập trước và các bài tập của Chương 19. Chương trình phía server cần trả về danh sách nhân viên trong công ty phân tách bằng dấu phẩy, tương tự như việc trả về danh sách các bang trong bài tập trên.

## Chương 21

1. Cả jQuery và MooTools đều dễ dùng đối với người mới bắt đầu sử dụng, PrototypeJS cũng tương đối dễ học. Dojo không hướng tới lập trình JavaScript cơ bản và có nhiều lập trình viên từng gặp rắc rối với YUI, dù các công cụ này có tài liệu đặc tả rất đầy đủ. Tuy nhiên, mọi người đều có những phương pháp học tập khác nhau, vì vậy tôi khuyên bạn thử dùng từng thư viện thay vì nghe theo ý kiên riêng của tôi.
2. Bài tập này cung cấp ví dụ về việc tự tạo thư viện và tham chiếu đến nó trong trang web.

## Chương 22

1. Một cách để thực hiện điều này là sử dụng hàm `.hover()` cùng với hàm `.css()` để thay đổi màu nền. Chúng được đặt trong hàm `showStates` trong file ajax.html:

```
function showStates(data, status) {
 $.each(data, function(item) {
 $("#states").append("<div>" + data[item] -
 $('#states').children().hover(
 function() {
 $(this).css('background-color', '#ccc');
 },
 function() {
 $(this).css('background-color', '#fff');
 }
);
 });
}
```

```

 function() {
 $(this).css('background-color',
 }
);
}

```

2. Đây là đoạn mã phía server được viết bằng PHP. Đoạn mã này trả về chữ "Wisconsin" khi đổi số nhận vào là từ viết tắt "WI":

```

<?php
$stateAbbrev = trim($_POST['state']);
if ($stateAbbrev == "WI") {
 print json_encode("Wisconsin");
}
?>

```

Sau đây là đoạn mã HTML, JavaScript và jQuery tạo ra kết quả dựa trên lời gọi AJAX:



```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Kiểm tra Dữ liệu AJAX</title>
<script type="text/javascript" data-src="jquery-1.4.1.js">
</head>
<body>
<div id="state">
</div>
<script type="text/javascript">
$(document).ready(function() {
$.ajax({
 type: "POST",
 url: "statefull.php",
 dataType: "json",
 success: showStateFull,
 data: "state=WI"
});
function showStateFull(data, status) {
 $("#state").text(data);
}
});

```

```
</script>
</body>
</html>
```

## Chương 23

1. Bạn có thể thêm tính năng để hiển thị tất cả các lựa chọn theo nhiều cách. Ví dụ này thêm một phần tử `<a>` với ID là `showAll`. ID đó sẽ được gắn với một hàm trong đoạn mã jQuery/JavaScript. Hàm này duyệt qua từng lựa chọn, và nếu thuộc tính `display` của lựa chọn đang là `none`, đoạn mã thực thi hàm `.show()` cho lựa chọn đó. Một phương án khác – có thể tốt hơn – là thêm một lớp `hidden` vào phần tử `<li>` khi hàm `.hide()` được thực thi. Việc tìm phần tử bị ẩn sau đó sẽ dễ dàng hơn nhiều vì bạn chỉ cần tìm bất cứ phần tử `<li>` nào bị ẩn bên trong thẻ `<ul>`. Có một cách khác là dùng bộ chọn `:hidden` của `jQuery`; tuy nhiên, tôi đã gặp một vài vấn đề khi sử dụng bộ chọn `:hidden` trên các trình duyệt khác nhau, đó là lý do tôi áp dụng cách làm khác trong bài tập này. Đoạn mã sẽ có dạng như sau:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ẩn phần tử</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
<style type="text/css">
.removeBtn {
 color: #0000CC;
}
</style>
</head>
<body>

<li id="option1">Lựa chọn 1
<li id="option2">Lựa chọn 2
<li id="option3">Lựa chọn 3
<li id="option4">Lựa chọn 4

Show All
<script type="text/javascript">
```

```

$(document).ready(function() {
 $('.removeBtn').each(function(elm) {
 $(this).click(function() {
 $(this).parent().hide();
 });
 });
 $('#showAll').click(function() {
 $('.removeBtn').each(function(elm) {
 if ($(this).parent().attr("display") == "none") {
 $(this).parent().show();
 }
 });
 });
});
</script>
</body>
</html>

```

2. Chìa khóa của bài tập này là sử dụng tùy chọn `async` trong hàm `.ajax()` của jQuery. Không có tùy chọn `asyn`, hàm `.accordion()` sẽ thực thi trước khi các tùy chọn của accordion được nạp và xử lý bởi AJAX. Đoạn mã mẫu sử dụng lời gọi JSON để lấy về danh sách các bang đã dùng trong Chương 22. Sau đó, danh sách này sẽ được đặt vào accordion. Đoạn mã thiết lập `async` là `false` trong lời gọi tới hàm `.ajax()` và sau đó thêm một đoạn xử lý vào hàm `showStates` có trong Chương 22. Đây chỉ là biến thể của ví dụ accordion có trong Chương 23.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Ví dụ về hiệu ứng Accordion</title>
<script type="text/javascript" data-src="jquery-1.4.2.min.js">
<script type="text/javascript" data-src="js/jquery-ui-1.8.1.custom.min.js">
<style type="text/css">

```

```

#container {
 width: 350px;
 height: 700px;
 border: solid 2px black;
 padding: 3px;
}

```

```
 }
 .optionHead {
 border: solid 1px black;
 background-color: #abacab;
 }
 .optionDiv {
 border-bottom: dotted 1px black;
 }

```

</style>

</head>

<body>

<div id="container">

</div>

<script type="text/javascript">

```
$(document).ready(function() {
$.ajax({
 type: "GET",
 url: "json.php",
 dataType: "json",
 success: showStates,
 async: false
});
function showStates(data,status) {
 $.each(data, function(item) {
 $("#container").append(
 "<h3 class=\"optionHead\">" +
 "<input checked="" type=\"checkbox\" value=" +
 item.state +
 "> " +
 item.state +
 "</h3>" +
 "<div class=\"optionDiv\">" +
 "<ul style=" +
 "list-style-type: none;" +
 "padding-left: 0;" +
 ">" +
 "" +
 "<img alt=" +
 item.flag +
 " style=" +
 "width: 20px;" +
 ">" +
 item.flag +
 "" +
 "" +
 item.capital +
 "" +
 "" +
 item.state +
 "" +
 "" +
 "</div>" +
 "
"
);
 });
 $('#container').accordion({
 collapsible: true,
 active: false
 });
});
```

</script>

</body>

</html>

# JavaScript

## Hướng dẫn học qua ví dụ

PHIÊN BẢN LẦN 2

### Học kiến thức lập trình JavaScript cơ sở thông qua các hướng dẫn thực hành từng bước

Cuốn sách hướng dẫn bạn từng bước để lập trình JavaScript. Lý tưởng cho người học làm quen với những kỹ năng lập trình cơ bản, cuốn sách này cung cấp các chỉ dẫn rõ ràng, những ví dụ thực hành thực tế, cần thiết để tạo hoặc tùy chỉnh các ứng dụng Web giàu tính tương tác bằng cách sử dụng kỹ thuật và các tính năng chính của JavaScript.

#### Bạn sẽ có khả năng:

- Viết và triển khai mã JavaScript sử dụng Microsoft® Visual Studio® 2010, Eclipse IDE, hoặc các trình soạn thảo văn bản
- Làm việc với cú pháp và các kiểu dữ liệu trong JavaScript
- Sử dụng DOM để truy xuất, tạo và thay đổi các phần tử HTML
- Tạo hiệu ứng rollover cho ảnh và hiệu ứng trình chiếu ảnh
- Kiểm tra tính hợp lệ và cung cấp phản hồi cho dữ liệu người dùng nhập vào Web form
- Thao tác với các style CSS và đáp ứng lại các sự kiện trình duyệt
- Sử dụng AJAX để phát triển các ứng dụng Web giàu tính tương tác
- Tiết kiệm công sức lập trình bằng cách sử dụng các framework JavaScript như jQuery

#### Giới thiệu tác giả

Steve Suehring là một kỹ sư công nghệ thông tin, ông từng viết sách về lập trình, bảo mật, quản trị mạng và quản trị hệ thống, các hệ điều hành và các đề tài thuộc những chuyên ngành khác. Ông cũng là diễn giả tại các hội thảo, các nhóm người dùng và là biên tập viên cho một tạp chí công nghệ nổi tiếng.

#### SÁCH THAM KHẢO

##### Hướng dẫn từng bước dành cho lập trình viên

- Bài hướng dẫn thực hành cho các kỹ thuật và tính năng cơ sở
- Các bài tập thực hành
- Chuẩn bị và cung cấp thông tin về các chủ đề mới dành cho các lập trình viên



##### Sách tham khảo dành cho lập trình viên

- Các chủ đề chuyên sâu
- Các ví dụ lập trình bao quát, thực tế
- Thành thạo công nghệ của Microsoft



##### Các chủ đề trọng tâm

- Tìm hiểu sâu về khả năng và các kỹ thuật nâng cao
- Các ví dụ lập trình thích ứng, bao quát
- Lâm chủ hoàn toàn công nghệ của Microsoft



Tủ sách bản quyền FPT Polytechnic không ngừng phát hành thêm những đầu sách mới về Công nghệ thông tin và Quản trị kinh doanh có bản quyền, được tuyển dịch cẩn trọng. Để có thêm thông tin, mời các bạn ghé thăm trang web <http://books.poly.edu.vn>



Giá: 190.000 VND

FPT  
FPT University  
TRƯỜNG ĐẠI HỌC FPT

Microsoft®  
Visual Studio 2010

Microsoft®