Linux Professional Institute

Web Development Essentials

Phiên bản 1.0
Tiếng Việt

030

Table of Contents

Linux Professional Institute

Chủ đề 031: hát triển Phần mềm và Công nghệ Web

031.1 Khái niệm cơ bản về Phát triển Phần mềm

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 031.1

Khối lượng

1

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Hiểu mã nguồn là gì
- Hiểu về các nguyên tắc của trình biên dịch và trình thông dịch
- Hiểu về khái niệm thư viện
- Hiểu về các khái niệm về lập trình chức năng, thủ tục và hướng đối tượng
- Nhận thức về các tính năng phổ biến của các trình chỉnh sửa mã nguồn và môi trường phát triển tích hợp (IDE)
- Nhận thức về hệ thống kiểm soát phiên bản
- Nhận thức về tác vụ kiểm tra phần mềm
- Nhận thức về các ngôn ngữ lập trình quan trọng (C, C++, C#, Java, JavaScript, Python, PHP)

Linux Professional Institute

031.1 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	031 Phát triển Phần mềm và Công nghệ Web
Mục tiêu:	031.1 Khái niệm cơ bản về Phát triển Phần mềm
Bài:	1 trên 1

Giới thiệu

Những chiếc máy tính đầu tiên đã được lập trình từ quá trình gán các cáp vào các ổ nối đầy cực nhọc. Từ đây, các nhà khoa học máy tính đã không ngừng tìm kiếm những cách dễ dàng nhất để có thể ra lệnh cho máy tính. Chương này sẽ giới thiệu cho người đọc về các công cụ lập trình. Chúng ta sẽ thảo luận về những cách chính mà các chỉ thị bằng văn bản—tức ngôn ngữ lập trình—đại diện cho các tác vụ mà một lập trình viên muốn hoàn thành cũng như về các công cụ dùng để biến các chương trình thành một dạng được gọi là *ngôn ngữ của máy* mà máy tính có thể chạy.

NOTE Trong văn bản này, thuật ngữ *chương trình* và *ứng dụng* có thể được sử dụng thay thế lẫn nhau.

Mã Nguồn

Một lập trình viên thường sẽ phát triển một ứng dụng bằng cách viết một bản mô tả bằng văn bản—hay còn được gọi là *mã nguồn*--của các tác vụ mong muốn. Mã nguồn sẽ ở dưới dạng *ngôn ngữ lập trình* được xác định một cách nghiêm ngặt để thể hiện những gì máy tính có thể thực hiện được ở mức độ trừu tượng cao mà con người có thể hiểu được. Nhiều công cụ cũng đã được phát triển để cho phép các lập trình viên cũng như những người không phải là lập trình viên diễn đạt

suy nghĩ của họ một cách trực quan; tuy vậy, mã nguồn vẫn là cách lập trình chủ yếu nhất.

Cũng giống như trong một ngôn ngữ tự nhiên có danh từ, động từ và các cấu trúc để diễn đạt ý tưởng theo những cách cụ thể, các từ và dấu chấm câu trong ngôn ngữ lập trình là các ký hiệu tượng trưng cho các thao tác sẽ được thực hiện trên máy.

Về mặt này, mã nguồn không khác nhiều so với bất kỳ loại văn bản nào khác mà trong đó tác giả sử dụng các quy tắc đã được thiết lập từ lâu trong ngôn ngữ tự nhiên để giao tiếp với người đọc. Với mã nguồn, "người đọc" chính là máy móc; vì vậy, văn bản không thể chứa những điểm mơ hồ hoặc mâu thuẫn dù là nhỏ nhất.

Và giống như bất kỳ một văn bản thảo luận sâu về một chủ đề nào đó, mã nguồn cũng cần được cấu trúc tốt và tổ chức một cách hợp lý khi dùng để phát triển các ứng dụng phức tạp. Các chương trình đơn giản và các ví dụ mô phạm có thể được lưu trữ chỉ trong một vài dòng của một tệp văn bản chứa tất cả mã nguồn của chương trình. Các chương trình phức tạp hơn có thể được chia nhỏ thành hàng nghìn tệp, mỗi tệp có hàng nghìn dòng.

Mã nguồn của các ứng dụng chuyên nghiệp nên được sắp xếp thành các thư mục khác nhau có liên quan tới những mục đích cụ thể. Ví dụ như một chương trình tán gẫu có thể được tổ chức thành hai thư mục: một thư mục chứa các tệp mã xử lý việc truyền và nhận tin nhắn qua mạng và một thư mục khác chứa các tệp xây dựng giao diện và các phản ứng với thao tác của người dùng. Thật vậy, thông thường sẽ có nhiều thư mục và thư mục con chứa các tệp mã nguồn dành riêng cho các tác vụ rất cụ thể trong một ứng dụng.

Hơn nữa, mã nguồn không phải lúc nào cũng được tách biệt trong các tệp riêng được viết bằng một ngôn ngữ duy nhất. Ví dụ: trong các ứng dụng web, tài liệu HTML có thể nhúng mã JavaScript để thêm vào tài liệu một số chức năng bổ sung.

Trình Soạn Mã và IDE

Sự đa dạng các cách viết mã nguồn có thể khiến cho các nhà phát triển trở nên lúng túng. Do đó, nhiều nhà phát triển đã tận dụng các công cụ hỗ trợ việc viết và kiểm tra các chương trình.

Tệp mã nguồn cuối cùng cũng vẫn chỉ là một tệp văn bản thuần túy. Vì thế, nó có thể được chỉnh sửa bởi bất kỳ trình soạn thảo văn bản nào dù đơn giản đến đâu. Để dễ dàng phân biệt giữa mã nguồn và một văn bản thuần túy, mỗi ngôn ngữ sẽ sử dụng phần mở rộng tên tệp tự giải thích: `.c` cho ngôn ngữ C, `.py` cho Python, `.js` cho JavaScript, v.v. Các trình soạn đa dụng đều hiểu rõ mã nguồn của các ngôn ngữ phổ biến đủ để thêm chữ nghiêng, màu sắc và thụt đầu dòng nhằm làm cho mã trở dễ hiểu hơn.

Không phải nhà phát triển nào cũng sẽ chọn chỉnh sửa mã nguồn trong những trình soạn đa dụng. Một *Môi trường Phát triển Tích hợp* (IDE - Integrated Development Environment) sẽ cung cấp một

trình soạn thảo văn bản cùng với các công cụ để giúp lập trình viên tránh các lỗi cú pháp và không nhất quán rõ ràng. Những trình soạn thảo này đặc biệt được khuyên dùng cho những lập trình viên ít kinh nghiệm, nhưng những lập trình viên nhiều kinh nghiệm cũng vẫn sử dụng chúng.

Các IDE phổ biến như Visual Studio, Eclipse và Xcode sẽ theo dõi một cách thông minh những gì lập trình viên nhập, thường xuyên gợi ý các từ để sử dụng (tự động hoàn thành) và xác minh mã trong thời gian thực. Các IDE thậm chí có thể cung cấp tác vụ sửa lỗi và kiểm thử tự động để xác định các vấn đề bất cứ khi nào mã nguồn thay đổi.

Một số lập trình viên có kinh nghiệm hơn sẽ chọn các trình soạn ít trực quan hơn (ví dụ như Vim) hỗ trợ tính linh hoạt cao hơn và không yêu cầu cài đặt các gói bổ sung. Những lập trình viên này sử dụng công cụ độc lập bên ngoài để thêm các tính năng được tích hợp sẵn khi bạn sử dụng IDE.

Bảo trì Mã

Cho dù trong IDE hay khi sử dụng các công cụ độc lập, điều quan trọng là phải sử dụng một số loại *hệ thống kiểm soát phiên bản* (VCS - Version Control System). Mã nguồn luôn không ngừng phát triển vì các lỗi không lường trước được cần phải được sửa và các cải tiến cần phải được kết hợp. Hệ quả tất yếu của quá trình phát triển này là các bản sửa lỗi và cải tiến có thể sẽ ảnh hưởng đến các phần khác của ứng dụng trong một cơ sở mã lớn. Các công cụ kiểm soát phiên bản như Git, Subversion và Mercurial sẽ ghi lại tất cả các thay đổi được thực hiện đối với mã cũng như người thực hiện thay đổi đó, cho phép bạn theo dõi và sau cùng là khôi phục sau một phiên sửa đổi không thành công.

Hơn nữa, các công cụ quản lý phiên bản cho phép mỗi nhà phát triển trong nhóm có thể làm việc trên một bản sao của các tệp mã nguồn mà không can thiệp vào công việc của các lập trình viên khác. Khi các phiên bản mới của mã nguồn đã sẵn sàng và đã được thử nghiệm, các chỉnh sửa hoặc cải tiến trên một bản sao có thể được kết hợp bởi các thành viên khác trong nhóm.

Git là một hệ thống kiểm soát phiên bản phổ biến nhất hiện nay; nó cho phép nhiều bản sao độc lập của kho lưu trữ được duy trì bởi những người khác nhau muốn chia sẻ những thay đổi của họ theo ý muốn. Tuy nhiên, cho dù sử dụng hệ thống kiểm soát phiên bản phân tán hay tập trung thì hầu hết các nhóm đều sẽ phải duy trì một kho lưu trữ đáng tin cậy có mã nguồn và tài nguyên có thể dựa vào. Có nhiều dịch vụ trực tuyến cung cấp kho lưu trữ mã nguồn, phổ biến nhất trong số các dịch vụ này là GitHub và GitLab. Savannah của dự án GNU cũng rất đáng được nhắc đến.

Ngôn ngữ Lập trình

Có rất nhiều ngôn ngữ lập trình tồn tại; mỗi thập kỷ trôi qua chúng ta đều chứng kiến sự ra đời của những ngôn ngữ lập trình mới. Mỗi ngôn ngữ lập trình có các quy tắc riêng và được khuyến

nghị cho các mục đích riêng. Mặc dù các ngôn ngữ thể hiện sự khác biệt nhất định về cú pháp và từ khóa nhưng điều thực sự phân biệt các ngôn ngữ với nhau là cách tiếp cận khái niệm sâu sắc mà chúng thể hiện, được gọi là *mô hình* (paradigm).

Mô hình

Các mô hình sẽ xác định tiền đề mà ngôn ngữ lập trình dựa vào, đặc biệt liên quan đến cách cấu trúc mã nguồn.

Nhà phát triển sẽ bắt đầu từ mô hình ngôn ngữ để hình thành các tác vụ mà máy cần thực hiện. Những tác vụ này được thể hiện một cách tượng trưng bằng các từ và cấu trúc cú pháp do ngôn ngữ cung cấp.

Ngôn ngữ lập trình đều tuân theo *thủ tục* khi các hướng dẫn được trình bày trong mã nguồn được thực thi theo thứ tự tuần tự, giống như một kịch bản phim. Nếu mã nguồn được phân thành các chức năng hoặc chương trình con thì một quy trình chính sẽ đảm nhận việc gọi các chức năng theo thứ tự.

Đoạn mã sau là một ví dụ về ngôn ngữ thủ tục. Được viết bằng C, nó định nghĩa các biến thể hiện độ dài cạnh (side), diện tích (area) và thể tích (volume) của các hình dạng địa lý. Giá trị của biến `side` được gán trong `main()` là hàm sẽ được gọi khi chương trình thực thi. Các biến `area` và `volume` được tính toán trong các chương trình con là `square()` và `cube()` đứng trước hàm chính:

```
#include <stdio.h>

float side;
float area;
float volume;

void square(){ area = side * side; }

void cube(){ volume = area * side; }

int main(){
  side = 2;
  square();
  cube();
  printf("Volume: %f\n", volume);
  return 0;
}
```

Thứ tự của các hành động được xác định trong `main()` sẽ xác định trình tự các trạng thái của

chương trình, đặc trưng bởi giá trị của các biến `side`, `area` và `volume`. Ví dụ sẽ kết thúc sau khi hiển thị giá trị của `volume` với câu lệnh `printf`.

Mặt khác, mô hình *lập trình hướng đối tượng* (OOP - Object-Oriented Programming) có đặc điểm chính là tách trạng thái chương trình thành các trạng thái con độc lập. Các trạng thái con và hoạt động liên quan này là *đối tượng* (được gọi như vậy vì chúng ít nhiều có sự tồn tại độc lập trong chương trình và có các mục đích cụ thể).

Các mô hình riêng biệt không nhất thiết phải hạn chế loại tác vụ mà một chương trình thực hiện. Mã từ ví dụ trước có thể được viết lại theo mô hình OOP bằng ngôn ngữ C++:

```
#include <iostream>

class Cube {
  float side;
  public:
  Cube(float s){ side = s; }
  float volume() { return side * side * side; }
};

int main(){
  float side = 2;
  Cube cube(side);
  std::cout << "Volume: " << cube.volume() << std::endl;
  return 0;
}
```

Hàm `main()` vẫn còn. Nhưng bây giờ ta lại có một từ mới là `class` (hạng) đưa ra định nghĩa về một đối tượng. Hạng được định nghĩa (có tên là `Cube`) sẽ chứa các biến và chương trình con của chính nó. Trong OOP, một biến còn được gọi là *thuộc tính* và một chương trình con còn được gọi là *phương thức*.

Việc giải thích toàn bộ mã C++ trong ví dụ trên nằm ngoài phạm vi của chương này. Điều quan trọng ở đây là chúng ta thấy được `Cube` chứa thuộc tính `side` và hai phương thức. Phương thức `volume()` sẽ tính toán thể tích của khối lập phương.

Ta có thể tạo một số đối tượng độc lập từ cùng một hạng và các hạng này có thể bao gồm các hạng khác nữa.

Hãy nhớ rằng các tính năng tương tự này có thể được ghi bằng những cách khác nhau và các ví dụ trong chương này đã được đơn giản hóa đi nhiều. C và C++ có nhiều tính năng phức tạp hơn cho phép xây dựng các cấu trúc mang tính thực tế và phức tạp hơn rất nhiều.

Hầu hết các ngôn ngữ lập trình sẽ không nhất quyết phải áp đặt một mô hình nhất định mà sẽ cho phép các lập trình viên lựa chọn các khía cạnh khác nhau của một mô hình nào đó. Ví dụ như JavaScript kết hợp các khía cạnh của nhiều các mô hình khác nhau. Lập trình viên có thể phân tách toàn bộ chương trình thành các chức năng không chia sẻ một trạng thái chung với nhau:

```
function cube(side){
  return side*side*side;
}

console.log("Volume: " + cube(2));
```

Mặc dù ví dụ này cũng tương tự như lập trình thủ tục nhưng hãy lưu ý rằng hàm sẽ nhận một bản sao của tất cả thông tin cần thiết để thực thi và luôn tạo ra một kết quả giống nhau cho cùng một tham số, bất kể có những thay đổi nào xảy ra bên ngoài phạm vi của hàm. Mô hình này, hay còn được gọi là *mô hình chức năng*, bị ảnh hưởng mạnh mẽ bởi chủ nghĩa hình thức toán học mà trong đó, mọi phép toán đều tự hoạt động.

Một mô hình khác bao gồm các ngôn ngữ *khai báo*; nó mô tả các trạng thái của hệ thống mà bạn muốn. Một ngôn ngữ khai báo có thể tìm ra cách đạt được các trạng thái đã chỉ định. SQL là một ngôn ngữ chung để truy vấn cơ sở dữ liệu, đôi khi vẫn được gọi là ngôn ngữ khai báo dù nó thực sự chiếm một vị trí độc nhất trong quần thể lập trình.

Không có một mô hình phổ quát nào có thể được áp dụng cho mọi hoàn cảnh. Việc lựa chọn ngôn ngữ cũng có thể bị hạn chế bởi loại ngôn ngữ được hỗ trợ trên mỗi nền tảng hoặc môi trường thực thi nơi chương trình sẽ được sử dụng.

Ví dụ như một ứng dụng web được trình duyệt sử dụng sẽ cần phải được viết bằng JavaScript vì đây là ngôn ngữ được toàn bộ các trình duyệt hỗ trợ (một vài ngôn ngữ khác cũng có thể được sử dụng vì chúng có cung cấp trình chuyển đổi để tạo JavaScript). Vì vậy, đối với trình duyệt web—đôi khi được gọi là *phía khách hàng* hoặc *phía người dùng* (front end) của ứng dụng web—nhà phát triển sẽ phải sử dụng các mô hình được cho phép trong JavaScript. Phía máy chủ (back end) của ứng dụng là nơi xử lý các yêu cầu từ trình duyệt và thường được lập trình bằng một ngôn ngữ khác; PHP là ngôn ngữ phổ biến nhất cho mục đích này.

Bất kể là trong mô hình nào, mọi ngôn ngữ đều sẽ có *các thư viện* chức năng được xây sẵn và có thể tích hợp được vào mã. Các hàm toán học—như các hàm được minh họa trong mã ví dụ—không cần phải triển khai từ đầu vì ngôn ngữ đã có hàm sẵn để sử dụng. Ví dụ như JavaScript cung cấp cho đối tượng `Math` các phép toán phổ biến nhất.

Thậm chí nhiều chức năng chuyên biệt hơn thường sẽ có sẵn từ nhà cung cấp ngôn ngữ hoặc nhà phát triển bên thứ ba. Các thư viện tài nguyên bổ sung này có thể ở dạng mã nguồn, tức là trong

các tệp bổ sung được tích hợp vào với tệp mà chúng sẽ được sử dụng. Trong JavaScript, việc nhúng được thực hiện với `import from`:

```
import { OrbitControls } from 'modules/OrbitControls.js';
```

Trong cách nhập này, tài nguyên được nhúng cũng là một tệp mã nguồn; cách nhập này thường được sử dụng nhiều nhất trong cái được gọi là *ngôn ngữ thông dịch*. *Các ngôn ngữ biên dịch* sẽ cho phép kết hợp các tính năng được biên dịch sẵn trong ngôn ngữ máy (tức các thư viện *đã được biên dịch*) bên cạnh nhiều yếu tố khác. Phần tiếp theo của bài học sẽ giải thích sự khác biệt giữa các loại ngôn ngữ này.

Trình biên dịch và Trình Thông dịch

Như chúng ta đã biết, mã nguồn là một biển diễn tượng trưng của một chương trình cần được dịch sang ngôn ngữ máy để chạy.

Nói một cách đại khái, có hai cách khả thi để thực hiện dịch: chuyển đổi mã nguồn trước để thực hiện trong tương lai, hoặc chuyển đổi mã tại thời điểm thực thi. Ngôn ngữ của phương thức thứ nhất được gọi là *ngôn ngữ biên dịch* và ngôn ngữ của phương thức thứ hai được gọi là *ngôn ngữ thông dịch*. Một số ngôn ngữ thông dịch sẽ cung cấp tính năng biên dịch dưới dạng tùy chọn để chương trình có thể bắt đầu một cách nhanh hơn.

Trong các ngôn ngữ biên dịch có sự phân biệt rõ ràng giữa mã nguồn của chương trình và chính chương trình sẽ được thực thi bởi máy tính. Sau khi được biên dịch, chương trình thường sẽ chỉ hoạt động trên hệ điều hành và nền tảng mà nó được biên dịch.

Trong một ngôn ngữ thông dịch, bản thân mã nguồn sẽ được coi là chương trình và quá trình chuyển đổi sang ngôn ngữ máy sẽ rất rõ ràng đối với người lập trình. Đối với một ngôn ngữ thông dịch, người ta thường gọi mã nguồn là *tệp lệnh*. Trình thông dịch sẽ dịch tệp lệnh sang ngôn ngữ máy cho hệ thống mà nó đang chạy.

Biên dịch và Trình Biên dịch

Ngôn ngữ lập trình C là một trong những ví dụ nổi tiếng nhất về ngôn ngữ biên dịch. Điểm mạnh nhất của ngôn ngữ C là tính linh hoạt và hiệu suất của nó. Cả siêu máy tính hiệu suất cao và bộ vi điều khiển trong thiết bị gia dụng đều có thể được lập trình bằng ngôn ngữ C. Các ví dụ khác về ngôn ngữ biên dịch phổ biến là C++ và C# (C sharp). Như tên của chúng đã cho hay, các ngôn ngữ này được lấy cảm hứng từ C nhưng sẽ bao gồm các tính năng hỗ trợ mô hình hướng tới đối tượng.

Cùng một chương trình được viết bằng C hoặc C++ có thể được biên dịch cho nhiều nền tảng khác nhau mà chỉ cần rất ít hoặc thậm chí là không có sự thay đổi trong mã nguồn. Trình biên dịch mới

là yếu tố xác định nền tảng đích của chương trình. Có các trình biên dịch dành riêng cho một nền tảng nhất định cũng như các trình biên dịch đa nền tảng như GCC (viết tắt của *GNU Compiler Collection*) có thể tạo các chương trình nhị phân cho nhiều kiến trúc riêng biệt.

NOTE

Ngoài ra còn có các công cụ nhằm tự động hóa quá trình biên dịch. Thay vì gọi trực tiếp trình biên dịch, lập trình viên sẽ tạo một tệp chỉ định các bước biên dịch khác nhau để thực hiện tự động. Công cụ truyền thống được sử dụng cho mục đích này là `make` nhưng một số công cụ mới hơn như Maven và Gradle cũng đang được sử dụng rộng rãi. Toàn bộ quá trình xây dựng sẽ được tự động hóa khi bạn sử dụng IDE.

Quá trình biên dịch không phải lúc nào cũng tạo ra một chương trình nhị phân bằng ngôn ngữ máy. Có những ngôn ngữ biên dịch sẽ tạo ra một chương trình ở định dạng thường được gọi là *bytecode*. Giống như một tệp lệnh, bytecode không phải là ngôn ngữ dành riêng cho một nền tảng nhất định; do đó, nó yêu cầu một chương trình thông dịch dịch nó sang ngôn ngữ máy. Trong trường hợp này, chương trình thông dịch sẽ được gọi đơn giản là *thời gian chạy*.

Ngôn ngữ Java sử dụng cách tiếp cận này để các chương trình được biên dịch viết bằng Java có thể được sử dụng trên các hệ điều hành khác nhau. Mặc dù tên là Java nhưng nó không có liên quan gì đến JavaScript.

Bytecode sát với ngôn ngữ máy hơn mã nguồn; vì vậy, tốc độ thực thi của nó có xu hướng nhanh hơn. Bởi vì vẫn còn một quá trình chuyển đổi trong quá trình thực thi bytecode nên rất khó để đạt được hiệu suất giống như chương trình tương đương được biên dịch thành ngôn ngữ máy.

Thông dịch và Trình Thông dịch

Trong các ngôn ngữ thông dịch như JavaScript, Python và PHP, chương trình không cần phải được biên dịch trước nên việc phát triển và sửa đổi chương trình sẽ trở nên dễ dàng hơn. Thay vì được biên dịch, tệp lệnh sẽ được thực thi bởi một chương trình khác gọi là trình thông dịch. Thông thường, trình thông dịch của một ngôn ngữ được đặt tên theo chính ngôn ngữ đó. Ví dụ: trình thông dịch của tệp lệnh Python là một chương trình có tên `python`. Trình thông dịch của JavaScript thường là trình duyệt web, nhưng các tệp lệnh cũng có thể được thực thi bởi chương trình `node` bên ngoài trình duyệt. Vì được chuyển đổi sang chỉ thị nhị phân mỗi khi được thực thi, một chương trình với ngôn ngữ được thông dịch sẽ có xu hướng chạy chậm hơn so với ngôn ngữ được biên dịch tương đương.

Một ứng dụng hoàn toàn có thể có các thành phần được viết bằng các ngôn ngữ khác nhau. Nếu cần, các thành phần này có thể giao tiếp với nhau thông qua *giao diện lập trình ứng dụng* (API - Application Programming Interface) chung.

Ví dụ như ngôn ngữ Python có khả năng khai thác dữ liệu và tổng hợp các bảng dữ liệu phức tạp.

Nhà phát triển có thể chọn Python để viết các phần của chương trình xử lý một số khía cạnh nhất định và một ngôn ngữ khác (chẳng hạn như C++) để thực hiện xử lý các quy trình số học nặng hơn. Người ta có thể áp dụng chiến lược này ngay cả khi không có API (API cho phép giao tiếp trực tiếp giữa hai thành phần). Ví dụ: mã được viết bằng Python có thể tạo tệp ở định dạng phù hợp để chương trình được viết bằng C++ có thể sử dụng được.

Mặc dù có thể viết hầu hết mọi chương trình bằng bất kỳ ngôn ngữ nào nhưng nhà phát triển vẫn nên áp dụng một ngôn ngữ phù hợp nhất với mục đích của ứng dụng. Khi làm như vậy, bạn có thể được hưởng lợi từ việc tái sử dụng các thành phần đã được thử nghiệm và ghi chép đầy đủ.

Bài tập Hướng dẫn

1. Những loại chương trình nào có thể được sử dụng để chỉnh sửa mã nguồn?

2. Loại công cụ nào có thể giúp tích hợp công việc của các nhà phát triển khác nhau vào cùng một cơ sở mã?

Bài tập Mở rộng

1. Giả sử bạn muốn viết một trò chơi 3D để chơi trên trình duyệt. Các ứng dụng web và trò chơi đều được lập trình bằng JavaScript. Mặc dù có thể viết tất cả các chức năng đồ họa từ đầu, nhưng sẽ hiệu quả hơn nếu bạn sử dụng một thư viện có sẵn cho mục đích này. Thư viện của bên thứ ba nào có thể cung cấp khả năng cho hoạt hoạ 3D trong JavaScript?

2. Ngoài PHP, những ngôn ngữ nào khác có thể được sử dụng ở phía máy chủ của ứng dụng web?

Tóm tắt

Bài học này đã trình bày các khái niệm thiết yếu nhất về phát triển phần mềm. Nhà phát triển phải nhận thức được về các ngôn ngữ lập trình quan trọng và trường hợp sử dụng phù hợp cho từng ngôn ngữ. Bài học này đã nói về các khái niệm và quy trình sau:

- Mã nguồn là gì.
- Trình soạn mã nguồn và các công cụ liên quan.
- Các mô hình lập trình thủ tục, hướng đối tượng, chức năng và khai báo.
- Đặc điểm của ngôn ngữ biên dịch và thông dịch.

Đáp án Bài tập Hướng dẫn

1. Những loại chương trình nào có thể được sử dụng để chỉnh sửa mã nguồn?

   Trên nguyên tắc, câu trả lời có thể là bất kỳ chương trình nào có khả năng chỉnh sửa văn bản thuần túy.

2. Loại công cụ nào có thể giúp tích hợp công việc của các nhà phát triển khác nhau vào cùng một cơ sở mã?

   Hệ thống quản lý nguồn hoặc phiên bản, chẳng hạn như Git.

Đáp án Bài tập Mở rộng

1. Giả sử bạn muốn viết một trò chơi 3D để chơi trên trình duyệt. Các ứng dụng web và trò chơi đều được lập trình bằng JavaScript. Mặc dù có thể viết tất cả các chức năng đồ họa từ đầu, nhưng sẽ hiệu quả hơn nếu bạn sử dụng một thư viện có sẵn cho mục đích này. Thư viện của bên thứ ba nào có thể cung cấp khả năng cho hoạt hoạ 3D trong JavaScript?

   Có nhiều lựa chọn về thư viện đồ họa 3D cho JavaScript, chẳng hạn như threejs và BabylonJS.

2. Ngoài PHP, những ngôn ngữ nào khác có thể được sử dụng ở phía máy chủ của ứng dụng web?

   Bất kỳ ngôn ngữ nào được hỗ trợ bởi ứng dụng máy chủ HTTP được sử dụng trên máy chủ lưu trữ. Một số ví dụ chính là Python, Ruby, Perl và JavaScript.

031.2 Kiến trúc của Ứng dụng Web

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 031.2

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu nguyên tắc máy tính máy khách và máy chủ
- Hiểu về vai trò của trình duyệt web và nhận biết các trình duyệt web thường được sử dụng
- Hiểu về vai trò của máy chủ web và máy chủ ứng dụng
- Hiểu về các tiêu chuẩn và công nghệ phát triển web phổ biến
- Hiểu về các nguyên tắc của API
- Hiểu về nguyên tắc của cơ sở dữ liệu quan hệ và phi quan hệ (NoSQL)
- Nhận thức về các hệ thống quản lý cơ sở dữ liệu nguồn mở thường được sử dụng
- Nhận thức về REST và GraphQL
- Nhận thức về các ứng dụng một trang
- Nhận thức về phương thức đóng gói ứng dụng web
- Nhận thức về WebAssembly
- Nhận thức về hệ thống quản lý nội dung

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Chrome, Edge, Firefox, Safari, Internet Explorer
- HTML, CSS, JavaScript

- SQLite, MySQL, MariaDB, PostgreSQL
- MongoDB, CouchDB, Redis

031.2 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	031 Phát triển Phần mềm và Công nghệ Web
Mục tiêu:	031.2 Kiến trúc của Ứng dụng Web
Bài:	1 trên 1

Giới thiệu

Từ *ứng dụng* mang một ý nghĩa rất rộng trong phạm vi thuật ngữ công nghệ. Khi ứng dụng là một chương trình truyền thống được thực thi cục bộ và tự túc theo mục đích của nó thì cả giao diện vận hành của ứng dụng và các thành phần xử lý dữ liệu đều sẽ được tích hợp trong một "gói" duy nhất. Một *ứng dụng web* lại là một khái niệm khác vì nó sử dụng mô hình máy khách/máy chủ; phần máy khách của nó được dựa trên HTML lấy từ máy chủ và được hiển thị bởi một trình duyệt.

Máy Khách và Máy Chủ

Trong mô hình máy khách/máy chủ, một phần công việc sẽ được thực hiện cục bộ ở *phía máy khách* và một phần công việc còn lại sẽ được thực hiện từ xa từ *phía máy chủ*. Việc nhiệm vụ nào sẽ được thực hiện bởi bên nào sẽ tùy theo mục đích của ứng dụng nhưng nhìn chung, máy khách sẽ phải cung cấp giao diện cho người dùng và bố cục nội dung một cách đẹp mắt. Máy chủ chịu trách nhiệm chạy phần kinh doanh của ứng dụng, xử lý và đáp ứng các yêu cầu được thực hiện

bởi máy khách. Ví dụ: trong ứng dụng mua sắm, máy khách sẽ hiển thị giao diện để người dùng chọn và thanh toán sản phẩm, nhưng nguồn dữ liệu và hồ sơ giao dịch đều được lưu trên máy chủ từ xa và được truy cập qua mạng. Các ứng dụng web thực hiện giao tiếp này qua Internet và thường là thông qua Giao thức Truyền Siêu Văn Bản (HTTP - Hypertext Transfer Protocol).

Sau khi được tải bởi trình duyệt, phía máy khách của ứng dụng sẽ bắt đầu tương tác với máy chủ bất cứ khi nào cần thiết hoặc thuận tiện. Các máy chủ ứng dụng web sẽ cung cấp *giao diện lập trình ứng dụng* (API) để xác định các yêu cầu khả dụng và cách thực hiện các yêu cầu đó. Do đó, máy khách sẽ tạo một yêu cầu theo định dạng do API xác định và gửi nó đến máy chủ, máy chủ này sẽ kiểm tra mọi điều kiện tiên quyết của yêu cầu và gửi lại những phản hồi thích hợp.

Trong khi máy khách (ở dạng ứng dụng dành cho thiết bị di động hoặc trình duyệt trên máy tính để bàn) là một chương trình độc lập liên quan đến giao diện người dùng và hướng dẫn giao tiếp với máy chủ thì trình duyệt phải lấy trang HTML và các thành phần liên quan—chẳng hạn như hình ảnh, CSS và JavaScript—để xác định giao diện và hướng dẫn giao tiếp với máy chủ.

Các ngôn ngữ lập trình và nền tảng được sử dụng bởi máy khách và máy chủ là độc lập với nhau nhưng lại sử dụng một giao thức giao tiếp cả hai bên đều có thể hiểu được. Phần máy chủ hầu như luôn được thực hiện bởi một chương trình không có giao diện đồ họa và chạy trong môi trường máy tính có tính sẵn sàng cao để luôn đảm bảo đáp ứng các yêu cầu. Ngược lại, phần máy khách thường chạy trên mọi thiết bị có khả năng hiển thị giao diện HTML, chẳng hạn như điện thoại thông minh.

Ngoài việc cần thiết cho các mục đích nhất định, việc áp dụng mô hình máy khách/máy chủ cho phép ứng dụng tối ưu hóa một số khía cạnh của quá trình phát triển và bảo trì vì mỗi phần đều có thể được thiết kế cho mục đích nhất định của nó. Ví dụ như một ứng dụng hiển thị bản đồ và tuyến đường sẽ không cần phải lưu trữ cục bộ tất cả các bản đồ. Chỉ những bản đồ liên quan đến vị trí mà người dùng quan tâm mới được yêu cầu, vì thế mà chỉ những bản đồ đó mới được yêu cầu từ máy chủ trung tâm.

Các nhà phát triển có quyền kiểm soát trực tiếp máy chủ, vì vậy họ cũng có thể sửa đổi máy khách do nó cung cấp. Điều này cho phép các nhà phát triển cải thiện ứng dụng ở mức độ cao hơn hoặc thấp hơn mà không cần người dùng phải mất công cài đặt các phiên bản mới.

Phía Máy Khách

Một ứng dụng web nên chạy theo một cách cố định trên bất kỳ trình duyệt phổ biến nhất nào, miễn là trình duyệt đó đã được cập nhật. Một số trình duyệt có thể không tương thích với những sửa đổi mới nhất, nhưng chỉ có những ứng dụng thử nghiệm mới sử dụng các tính năng chưa được áp dụng rộng rãi.

Trước đây thường xảy ra các vấn đề không tương thích bởi mỗi trình duyệt đều có một *công cụ kết xuất* riêng và giữa các trình duyệt ít có sự hợp tác trong việc xây dựng và áp dụng các tiêu chuẩn. Công cụ kết xuất là thành phần chính của trình duyệt vì nó chịu trách nhiệm chuyển đổi HTML và các thành phần liên quan khác thành các yếu tố trực quan và tương tác của giao diện. Một số trình duyệt, đặc biệt là Internet Explorer, cần được xử lý đặc biệt trong mã để không làm "hỏng" hoạt động của các trang.

Ngày nay, có rất ít sự khác biệt giữa các trình duyệt lớn và rất hiếm khi xảy ra hiện tượng không tương thích. Trên thực tế, trình duyệt Chrome và Edge đều sử dụng cùng một công cụ kết xuất (được gọi là Blink). Trình duyệt Safari và các trình duyệt khác được cung cấp trên iOS App Store sử dụng công cụ WebKit. Firefox sử dụng một công cụ được gọi là Gecko. Ba công cụ này có thể nói là đang "thầu" tất cả các trình duyệt được sử dụng ngày nay. Mặc dù được phát triển riêng biệt, ba công cụ này là những dự án mã nguồn mở và có sự hợp tác giữa các nhà phát triển của chúng. Điều này tạo điều kiện thuận lợi cho tính năng tương thích, việc bảo trì và áp dụng các tiêu chuẩn.

Do các nhà phát triển trình duyệt đã nỗ lực rất nhiều để duy trì tính tương thích nên máy chủ thường sẽ không liên kết với một loại máy khách duy nhất. Về nguyên tắc, một máy chủ HTTP có thể giao tiếp với bất kỳ máy khách nào cũng có khả năng giao tiếp qua HTTP. Ví dụ như trong ứng dụng bản đồ, máy khách có thể là ứng dụng di động hoặc trình duyệt tải giao diện HTML từ máy chủ.

Các loại Máy Khách trong Mạng

Có các ứng dụng dành cho thiết bị di động và máy tính để bàn có giao diện được hiển thị từ HTML và, giống như trình duyệt, có thể sử dụng JavaScript làm ngôn ngữ lập trình. Tuy nhiên, không giống như một máy khách được tải trong trình duyệt, HTML và các thành phần cần thiết để máy khách gốc hoạt động phải có mặt cục bộ kể từ khi cài đặt ứng dụng. Trên thực tế, một ứng dụng hoạt động theo cách này hầu như sẽ giống với một trang HTML (cả hai thậm chí sẽ có khả năng được hiển thị bởi cùng một công cụ). Ngoài ra còn có *ứng dụng web lũy tiến* (PWA) là một cơ chế cho phép bạn đóng gói các máy khách ứng dụng web để sử dụng ngoại tuyến giới hạn ở các chức năng không yêu cầu giao tiếp ngay lập tức với máy chủ. Về những gì ứng dụng có thể làm: không có sự khác biệt giữa việc chạy trình duyệt và việc được đóng gói trong PWA ngoại trừ việc sau này nhà phát triển sẽ có nhiều quyền kiểm soát hơn đối với những gì được lưu trữ cục bộ.

Việc kết xuất các giao diện từ HTML là một hoạt động xuyên đến mức công cụ dành cho tác vụ này thường là một thành phần phần mềm riêng biệt có trong hệ điều hành. Nó hiện diện như một thành phần riêng biệt cho phép các ứng dụng khác nhau kết hợp với nó mà không cần phải nhúng nó vào gói ứng dụng. Mô hình này cũng ủy thác việc bảo trì công cụ kết xuất cho hệ điều hành để tạo điều kiện cho việc cập nhật. Một điều rất quan trọng ở đây là luôn cập nhật thành phần quan trọng này để tránh các lỗi hệ thống có thể xảy ra.

Bất kể phương thức triển khai của chúng là gì, các ứng dụng được viết bằng HTML sẽ chạy trên một lớp trừu tượng do công cụ tạo ra và lớp này sẽ hoạt động như một môi trường thực thi biệt lập. Cụ thể hơn, trong trường hợp máy khách chạy trên trình duyệt, ứng dụng chỉ có thể tùy ý sử dụng những tài nguyên do trình duyệt cung cấp. Các tính năng cơ bản (chẳng hạn như tương tác với các thành phần trang và yêu cầu tệp qua HTTP) sẽ luôn khả dụng. Các tài nguyên có thể chứa các thông tin nhạy cảm như quyền truy cập vào tệp cục bộ, vị trí địa lý, máy ảnh và micrô cần có sự cho phép rõ ràng của người dùng trước khi ứng dụng có thể sử dụng chúng.

Ngôn ngữ của Máy Khách trong Mạng

Yếu tố trọng tâm của máy khách ứng dụng web chạy trong máy chủ là tài liệu HTML. Ngoài việc trình bày các thành phần giao diện mà trình duyệt hiển thị theo một cách có cấu trúc, tài liệu HTML còn chứa các địa chỉ cho tất cả các tệp cần thiết để trình bày và vận hành máy khách một cách chính xác.

Chỉ riêng HTML sẽ không có nhiều tính linh hoạt để xây dựng các giao diện phức tạp hơn và không có các tính năng lập trình cho mục đích chung. Vì lý do này, một tài liệu HTML sẽ hoạt động như một máy khách luôn đi kèm với một hoặc nhiều bộ CSS và JavaScript.

CSS có thể được cung cấp dưới dạng một tệp riêng biệt hoặc trực tiếp trong chính tệp HTML. Mục đích chính của CSS là điều chỉnh giao diện và bố cục của các phần tử trong giao diện HTML. Mặc dù không thực sự cần thiết nhưng các giao diện phức tạp thường yêu cầu sửa đổi các thuộc tính CSS của các phần tử để phù hợp với nhu cầu của chúng.

JavaScript là một thành phần không thể thiếu. Các quy trình được viết bằng JavaScript sẽ phản hồi tới các sự kiện trong trình duyệt. Những sự kiện này có thể do người dùng gây ra hoặc do việc không tương tác. Không có JavaScript, tài liệu HTML thực tế chỉ giới hạn ở dạng văn bản và hình ảnh. Việc sử dụng JavaScript trong tài liệu HTML sẽ cho phép bạn mở rộng khả năng tương tác ngoài các siêu liên kết và biểu mẫu, làm cho trang được trình duyệt hiển thị trông giống như một giao diện ứng dụng thông thường.

JavaScript là ngôn ngữ lập trình đa dụng, nhưng mục đích sử dụng chính của nó là ở trong các ứng dụng web. Các tính năng của môi trường thực thi trình duyệt có thể được truy cập thông qua các từ khóa JavaScript được sử dụng trong tệp lệnh để thực hiện thao tác mong muốn. Ví dụ: thuật ngữ `document` (tài liệu) được sử dụng trong mã JavaScript là để chỉ tài liệu HTML được liên kết với mã JavaScript. Trong ngữ cảnh của ngôn ngữ JavaScript, `document` là một *đối tượng toàn cầu* với các thuộc tính và phương thức có thể được sử dụng để lấy thông tin từ bất kỳ phần tử nào trên tài liệu HTML. Quan trọng hơn, ta có thể sử dụng đối tượng `document` để sửa đổi các phần tử của nó và liên kết chúng với các hành động tùy chỉnh được viết bằng JavaScript.

Ứng dụng khách dựa trên công nghệ web là một ứng dụng đa nền tảng bởi nó có thể chạy trên bất

kỳ thiết bị nào có trình duyệt web tương thích.

Tuy nhiên, việc bị giới hạn trong trình duyệt sẽ đặt ra các hạn chế đối với các ứng dụng web so với các ứng dụng gốc. Sử dụng trình duyệt làm trung gian sẽ cho phép ta lập trình ở cấp cao hơn và tăng tính bảo mật, nhưng nó cũng làm tăng mức tiêu thụ bộ nhớ và quy trình xử lý.

Các nhà phát triển đang liên tục cải tiến các trình duyệt để cung cấp nhiều tính năng hơn và cải thiện hiệu suất của các ứng dụng JavaScript, nhưng có những khía cạnh nội tại đối với việc thực thi các tệp lệnh như JavaScript sẽ gây bất lợi cho chúng so với các chương trình gốc cho cùng phần cứng.

Một tính năng giúp cải thiện đáng kể hiệu suất của các ứng dụng JavaScript chạy trên trình duyệt là *WebAssembly*. WebAssembly là một loại JavaScript được biên dịch để tạo mã nguồn được viết bằng ngôn ngữ cấp thấp hơn và hiệu quả hơn, chẳng hạn như ngôn ngữ C. WebAssembly có thể tăng tốc các hoạt động chủ yếu sử dụng bộ xử lý nhiều vì nó tránh được phần lớn quá trình dịch do trình duyệt thực hiện khi chạy một chương trình được viết bằng JavaScript thông thường.

Bất kể chi tiết triển khai của ứng dụng là gì, tất cả các tệp mã HTML, CSS, JavaScript và đa phương tiện trước tiên phải được lấy từ máy chủ. Trình duyệt lấy các tệp này giống như một trang Internet, tức là với một địa chỉ được truy cập bởi trình duyệt.

Một trang web hoạt động như một giao diện cho một ứng dụng web cũng sẽ giống như một tài liệu HTML thuần tuý nhưng có thêm các hành vi bổ sung. Trên các trang thông thường, người dùng sẽ được chuyển hướng đến một trang khác sau khi nhấp vào liên kết. Các ứng dụng web có thể hiển thị giao diện của chúng và phản hồi các sự kiện của người dùng mà không cần tải các trang mới trong cửa sổ trình duyệt. Việc sửa đổi hành vi tiêu chuẩn này trong các trang HTML được thực hiện thông qua lập trình JavaScript.

Ví dụ: một ứng dụng email trên web sẽ hiển thị thư và chuyển đổi giữa các thư mục thư mà không cần rời khỏi trang. Điều này có thể thực hiện được là bởi máy khách sử dụng JavaScript để phản ứng lại các hành động của người dùng và đưa ra các yêu cầu phù hợp với máy chủ. Nếu người dùng nhấp vào chủ đề của thư trong hộp thư đến, mã JavaScript được liên kết với sự kiện này sẽ yêu cầu nội dung của thư đó từ máy chủ (sử dụng lệnh gọi API tương ứng). Ngay khi máy khách nhận được phản hồi, trình duyệt sẽ hiển thị thông báo trong phần thích hợp trong cùng một trang. Các ứng dụng webmail khách khác nhau có thể áp dụng các chiến lược khác nhau, nhưng tất cả đều sử dụng cùng một nguyên tắc này.

Do đó, ngoài việc cung cấp các tệp tạo nên máy khách cho trình duyệt, máy chủ cũng phải có khả năng xử lý các yêu cầu chẳng hạn như yêu cầu của ứng dụng webmail khi nó yêu cầu nội dung của một thư cụ thể. Mọi yêu cầu mà máy khách có thể thực hiện đều có một quy trình được xác định trước để phản hồi trên máy chủ. API của máy chủ có thể chỉ định các phương thức khác nhau để xác định quy trình mà yêu cầu đề cập đến. Các phương thức phổ biến nhất là:

- Địa chỉ, thông qua Bộ định vị tài nguyên thống nhất (URL)
- Các trường trong tiêu đề HTTP
- Phương thức GET/POST
- WebSockets

Một phương thức này có thể phù hợp hơn phương thức kia, tùy thuộc vào mục đích của yêu cầu và các tiêu chí khác được nhà phát triển tính đến. Nói chung, các ứng dụng web sẽ sử dụng kết hợp các phương thức, mỗi phương thức sẽ dành cho một trường hợp cụ thể.

Phương thức *Chuyển Trạng thái Biểu hiện* (REST - Representational State Transfer) được sử dụng rộng rãi để liên lạc trong các ứng dụng web vì nó dựa trên các phương thức cơ bản có sẵn trong HTTP. Tiêu đề của yêu cầu HTTP sẽ bắt đầu bằng một từ khóa xác định thao tác cơ bản sẽ được thực hiện: `GET`,`POST`, `PUT`,`DELETE`, v.v. kèm theo một URL tương ứng nơi hành động sẽ được áp dụng. Nếu ứng dụng yêu cầu các thao tác cụ thể hơn với mô tả chi tiết hơn về thao tác được yêu cầu thì giao thức GraphQL có thể sẽ là lựa chọn phù hợp hơn.

Các ứng dụng được phát triển bằng mô hình máy khách/máy chủ có thể sẽ không được ổn định trong giao tiếp. Vì lý do này, máy khách phải luôn áp dụng các chiến lược truyền dữ liệu hiệu quả để tạo sự nhất quán và không gây hại cho trải nghiệm của người dùng.

Phía Máy Chủ

Mặc dù là nhân vật chính trong một ứng dụng web, máy chủ lại là phía thụ động trong giao tiếp và chỉ đáp ứng các yêu cầu của máy khách. Trong từ điển về web, *máy chủ* có thể mang nghĩa là máy nhận yêu cầu, chương trình xử lý cụ thể các yêu cầu HTTP hoặc tệp lệnh nhân tạo phản hồi cho yêu cầu. Định nghĩa cuối cùng này là phù hợp nhất trong ngữ cảnh kiến trúc của ứng dụng web, nhưng tất cả trong số chúng cũng đều có liên quan chặt chẽ tới nhau. Mặc dù chỉ là một phần trong phạm vi của nhà phát triển máy chủ ứng dụng nhưng ta không thể bỏ qua máy, hệ điều hành và máy chủ HTTP vì chúng thường giao nhau và là nền tảng để chạy máy chủ ứng dụng.

Xử lý Đường dẫn theo Yêu cầu

Máy chủ HTTP (chẳng hạn như Apache và NGINX) thường yêu cầu các thay đổi cấu hình cụ thể để đáp ứng nhu cầu của ứng dụng. Theo mặc định, các máy chủ HTTP truyền thống sẽ liên kết trực tiếp đường dẫn được chỉ ra trong yêu cầu với một tệp trên hệ thống tệp cục bộ. Ví dụ: nếu máy chủ HTTP của một trang web giữ các tệp HTML của nó trong thư mục `/srv/www`, một yêu cầu có đường dẫn `/en/about.html` sẽ nhận được nội dung của tệp `/srv/www/en/ about.html` dưới dạng phản hồi trong trường hợp tệp có tồn tại. Các trang web phức tạp hơn và đặc biệt là các ứng dụng web sẽ cần các phương pháp xử lý tùy chỉnh cho các loại yêu cầu khác nhau. Trong trường

hợp này, một phần của việc triển khai ứng dụng sẽ sửa đổi cài đặt máy chủ HTTP để đáp ứng các yêu cầu của ứng dụng.

Ngoài ra, có những khuôn khổ sẽ cho phép ta tích hợp việc quản lý các yêu cầu HTTP và triển khai mã ứng dụng ở cùng một nơi, cho phép nhà phát triển tập trung nhiều hơn vào mục đích của ứng dụng hơn là chi tiết nền tảng. Ví dụ: trong Node.js Express, tất cả ánh xạ yêu cầu và lập trình tương ứng đều được triển khai bằng JavaScript. Vì việc lập trình máy khách thường được thực hiện bằng JavaScript nên nhiều nhà phát triển coi việc sử dụng cùng một ngôn ngữ cho máy khách và máy chủ từ góc độ bảo trì mã là một ý tưởng hợp lý. Các ngôn ngữ khác thường được sử dụng để triển khai phía máy chủ (kể cả là trong khuôn khổ hay trong máy chủ HTTP truyền thống) là PHP, Python, Ruby, Java và C#.

Hệ thống Quản lý Cơ sở Dữ liệu

Nhóm phát triển có toàn quyền quyết định cách mà máy khách nhận được hoặc yêu cầu dữ liệu được lưu trữ trên máy chủ, nhưng có những nguyên tắc chung áp dụng cho hầu hết các trường hợp. Việc giữ nội dung tĩnh—hình ảnh, mã JavaScript và CSS không thay đổi trong thời gian ngắn—dưới dạng tệp thông thường khá là tiện lợi dù là trên hệ thống tệp riêng của máy chủ hay được phân phối trên *mạng phân phối nội dung* (CDN). Các loại nội dung khác (chẳng hạn như email trong ứng dụng webmail, chi tiết sản phẩm trong ứng dụng mua sắm và nhật ký giao dịch) được lưu trữ thuận tiện hơn trong *hệ thống quản lý cơ sở dữ liệu* (DBMS).

Loại hệ thống quản lý cơ sở dữ liệu truyền thống nhất là *cơ sở dữ liệu quan hệ*. Trong đó, người thiết kế ứng dụng sẽ định nghĩa các bảng dữ liệu và định dạng đầu vào được chấp nhận bởi mỗi bảng. Tập hợp các bảng trong cơ sở dữ liệu sẽ chứa tất cả các dữ liệu động được ứng dụng sử dụng và tạo ra. Ví dụ: một ứng dụng mua sắm có thể có một bảng chứa một mục với chi tiết của từng sản phẩm trong cửa hàng và một bảng ghi lại các mặt hàng mà người dùng đã mua. Bảng các mặt hàng đã mua sẽ chứa tham chiếu đến các mục trong bảng sản phẩm và tạo nên mối quan hệ giữa các bảng. Cách tiếp cận này có thể tối ưu hóa việc lưu trữ và truy cập dữ liệu cũng như cho phép truy vấn trong các bảng kết hợp bằng ngôn ngữ được hệ thống quản lý cơ sở dữ liệu chấp nhận. Ngôn ngữ cơ sở dữ liệu quan hệ phổ biến nhất là *Ngôn ngữ Truy vấn có Cấu trúc* (SQL - Structured Query Language) được sử dụng bởi các cơ sở dữ liệu mã nguồn mở SQLite, MySQL, MariaDB và PostgreSQL.

Một giải pháp thay thế cho cơ sở dữ liệu quan hệ là một dạng cơ sở dữ liệu không yêu cầu cấu trúc cứng nhắc cho dữ liệu. Những cơ sở dữ liệu này được gọi là *cơ sở dữ liệu phi quan hệ* hoặc đơn giản là *NoSQL*. Mặc dù chúng có thể kết hợp một số tính năng tương tự như những tính năng được tìm thấy trong cơ sở dữ liệu quan hệ nhưng trọng tâm của chúng sẽ là cho phép ta linh hoạt hơn trong việc lưu trữ và truy cập vào dữ liệu được lưu trữ, chuyển giao nhiệm vụ xử lý dữ liệu đó cho chính ứng dụng. MongoDB, CouchDB và Redis là những hệ thống quản lý cơ sở dữ liệu phi quan hệ phổ biến.

Bảo trì Nội dung

Bất kể là mô hình cơ sở dữ liệu nào được áp dụng, các ứng dụng phải thêm dữ liệu và có thể sẽ phải cập nhật các dữ liệu đó trong suốt vòng đời của ứng dụng. Trong một số ứng dụng (chẳng hạn như webmail), người dùng sẽ tự cung cấp dữ liệu cho cơ sở dữ liệu khi sử dụng máy khách để gửi và nhận thư. Trong các trường hợp khác (chẳng hạn như trong ứng dụng mua sắm), điều quan trọng là phải cho phép người bảo trì ứng dụng sửa đổi cơ sở dữ liệu mà không cần phải lập trình. Do đó, nhiều tổ chức áp dụng một số loại *hệ thống quản lý nội dung* (CMS) cho phép người dùng không có kỹ thuật có thể quản lý ứng dụng. Vì vậy, đối với hầu hết các ứng dụng web, người ta cần triển khai ít nhất hai loại máy khách: máy khách không có đặc quyền được sử dụng bởi người dùng thông thường và máy khách có đặc quyền được sử dụng bởi người dùng đặc biệt để duy trì và cập nhật thông tin do ứng dụng trình bày.

Bài tập Hướng dẫn

1. Ngôn ngữ lập trình nào được sử dụng cùng với HTML để tạo các máy khách ứng dụng web?

2. Truy xuất ứng dụng web khác với truy xuất ứng dụng gốc như thế nào?

3. Ứng dụng web khác với ứng dụng gốc như thế nào khi truy cập vào phần cứng cục bộ?

4. Hãy trích dẫn một đặc điểm của máy khách ứng dụng web khiến nó khác biệt so với một trang web thông thường.

Bài tập Mở rộng

1. Các trình duyệt hiện đại cung cấp tính năng nào để khắc phục hiệu suất kém của các máy khách ứng dụng web sử dụng nhiều CPU?

2. Nếu một ứng dụng web sử dụng mô hình REST để liên lạc giữa máy khách và máy chủ, phương thức HTTP nào sẽ được sử dụng khi máy khách yêu cầu máy chủ xóa một tài nguyên cụ thể?

3. Hãy kể tên 5 ngôn ngữ lập trình máy chủ được máy chủ HTTP Apache hỗ trợ.

4. Tại sao cơ sở dữ liệu phi quan hệ lại được coi là dễ bảo trì và nâng cấp hơn so với cơ sở dữ liệu quan hệ?

Tóm tắt

Bài học này đã đề cập tới các khái niệm và tiêu chuẩn trong công nghệ và kiến trúc phát triển web. Nguyên tắc ở đây rất đơn giản: trình duyệt web chạy máy khách, giao tiếp với ứng dụng cốt lõi đang chạy trong máy chủ. Mặc dù khá đơn giản về nguyên tắc, các ứng dụng web luôn phải kết hợp nhiều công nghệ để áp dụng nguyên tắc tính toán máy khách và máy chủ trên web. Bài học này đã nhắc tới các khái niệm sau:

- Vai trò của trình duyệt web và máy chủ web.
- Các tiêu chuẩn và công nghệ phát triển web phổ biến.
- Làm thế nào để máy khách ứng dụng web có thể giao tiếp với máy chủ.
- Các loại máy chủ web và cơ sở dữ liệu máy chủ.

Đáp án Bài tập Hướng dẫn

1. Ngôn ngữ lập trình nào được sử dụng cùng với HTML để tạo các máy khách ứng dụng web?

   JavaScript.

2. Truy xuất ứng dụng web khác với truy xuất ứng dụng gốc như thế nào?

   Một ứng dụng web là một ứng dụng không được cài đặt. Thay vào đó, các phần của nó chạy trên máy chủ và giao diện máy khách sẽ chạy trong một trình duyệt web thông thường.

3. Ứng dụng web khác với ứng dụng gốc như thế nào khi truy cập vào phần cứng cục bộ?

   Tất cả các quyền truy cập vào các tài nguyên cục bộ (chẳng hạn như bộ lưu trữ, máy ảnh hoặc micrô) đều do trình duyệt làm trung gian và yêu cầu phải có sự cho phép rõ ràng của người dùng để có thể hoạt động.

4. Hãy trích dẫn một đặc điểm của máy khách ứng dụng web khiến nó khác biệt so với một trang web thông thường.

   Sự tương tác với các trang web truyền thống về cơ bản sẽ bị hạn chế đối với các siêu liên kết và biểu mẫu gửi, trong khi các máy khách ứng dụng web sẽ gần với giao diện ứng dụng thông thường hơn.

Đáp án Bài tập Mở rộng

1. Các trình duyệt hiện đại cung cấp tính năng nào để khắc phục hiệu suất kém của các máy khách ứng dụng web sử dụng nhiều CPU?

   Các nhà phát triển có thể sử dụng WebAssembly để triển khai các phần tiêu tốn nhiều CPU của máy khách. Mã WebAssembly nói chung có hiệu suất tốt hơn JavaScript truyền thống vì mã này yêu cầu dịch các chỉ dẫn ít hơn.

2. Nếu một ứng dụng web sử dụng mô hình REST để liên lạc giữa máy khách và máy chủ, phương thức HTTP nào sẽ được sử dụng khi máy khách yêu cầu máy chủ xóa một tài nguyên cụ thể?

   REST dựa trên các phương thức HTTP tiêu chuẩn; do đó, ta nên sử dụng phương thức `DELETE` tiêu chuẩn trong trường hợp này.

+. Hãy kể tên 5 ngôn ngữ lập trình máy chủ được máy chủ HTTP Apache hỗ trợ. PHP, Go, Perl, Python, và Ruby.

1. Tại sao cơ sở dữ liệu phi quan hệ được coi là dễ bảo trì và nâng cấp hơn so với cơ sở dữ liệu quan hệ?

   Không giống như cơ sở dữ liệu quan hệ, cơ sở dữ liệu phi quan hệ không yêu cầu dữ liệu phải khớp với các cấu trúc cứng nhắc được xác định từ trước, giúp dễ dàng thực hiện các thay đổi trong cấu trúc dữ liệu mà không ảnh hưởng đến dữ liệu hiện có.

031.3 Khái niệm cơ bản về HTTP

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 031.3

Khối lượng

3

Các lĩnh vực kiến thức chính

- Hiểu về các phương thức HTTP GET và POST, mã trạng thái, đề mục và loại nội dung
- Hiểu về sự khác biệt giữa nội dung tĩnh và động
- Hiểu về URL HTTP
- Hiểu về cách các URL HTTP được ánh xạ tới các đường dẫn hệ thống tệp
- Tải tệp lên thư mục gốc của máy chủ web
- Hiểu về bộ nhớ đệm
- Hiểu về cookie
- Nhận thức về các phiên làm việc và việc chiếm quyền điều khiển phiên
- Nhận thức về các máy chủ HTTP thường được sử dụng
- Nhận thức về HTTPS và TLS
- Nhận thức về ổ nối web
- Nhận thức về máy chủ ảo
- Nhận thức về các máy chủ HTTP phổ biến
- Nhận thức về các yêu cầu và giới hạn về băng thông mạng và độ trễ

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- GET, POST
- 200, 301, 302, 401, 403, 404, 500
- Apache HTTP Server (httpd), NGINX

Linux Professional Institute

031.3 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	031 Phát triển Phần mềm và Công nghệ Web
Mục tiêu:	031.3 Khái niệm cơ bản về HTTP
Bài:	1 trên 1

Giới thiệu

Giao thức truyền Siêu Văn bản (HTTP - HyperText Transfer Protocol) sẽ xác định cách mà máy khách yêu cầu máy chủ cung cấp một tài nguyên cụ thể. Nguyên tắc hoạt động của nó khá đơn giản: máy khách sẽ tạo một thông báo yêu cầu xác định tài nguyên mà nó cần và chuyển tiếp thông báo đó đến máy chủ thông qua mạng. Đổi lại, máy chủ HTTP sẽ xem xét vị trí để trích xuất tài nguyên được yêu cầu và gửi thông báo phản hồi lại cho máy khách. Thông báo trả lời sẽ chứa thông tin chi tiết về tài nguyên được yêu cầu, theo sau chính là tài nguyên đó.

Cụ thể hơn, HTTP là tập hợp các quy tắc xác định cách máy khách định dạng các thông báo *yêu cầu* sẽ được gửi đến máy chủ. Sau đó, máy chủ sẽ tuân theo các quy tắc HTTP để diễn giải yêu cầu và định dạng thông báo *phản hồi*. Ngoài việc yêu cầu hoặc chuyển nội dung được yêu cầu, các thông báo HTTP sẽ chứa các thông tin bổ sung về máy khách và máy chủ có liên quan, về bản thân nội dung và thậm chí là về tính không khả dụng của nội dung đó. Nếu không thể gửi tài nguyên, một mã trong phản hồi sẽ giải thích lý do không khả dụng và sẽ cho biết nơi tài nguyên đã được chuyển tới nếu có thể.

Phần thông báo xác định chi tiết tài nguyên và thông tin ngữ cảnh khác được gọi là *tiêu đề* của thông báo. Phần theo sau tiêu đề sẽ chứa nội dung của tài nguyên tương ứng và được gọi là *tải*

trọng của thông báo. Cả thông báo yêu cầu và thông báo phản hồi đều có thể có tải trọng, nhưng trong hầu hết các trường hợp thì chỉ có thông báo phản hồi là có tải trọng.

Yêu cầu của Máy Khách

Giai đoạn đầu tiên của một phiên trao đổi dữ liệu HTTP giữa máy khách và máy chủ sẽ được bắt đầu bởi máy khách khi nó viết một thông báo yêu cầu tới máy chủ. Lấy ví dụ một tác vụ phổ biến của trình duyệt: tải trang HTML từ máy chủ lưu trữ trang web (chẳng hạn như `https://learning.lpi.org/en/`). Địa chỉ (hay còn gọi là URL) sẽ cung cấp một số thông tin liên quan. Có ba mẩu thông tin xuất hiện trong ví dụ cụ thể này:

- Giao thức: Giao thức Truyền Siêu Văn bản an toàn (`https`) - một phiên bản được mã hóa của HTTP.
- Tên mạng của máy chủ web (`learning.lpi.org`)
- Vị trí của tài nguyên được yêu cầu trên máy chủ (thư mục `/en/`--trong trường hợp này là phiên bản tiếng Anh của trang chủ).

NOTE

Bộ Định vị Tài nguyên Thống nhất (URL - Uniform Resource Locator) là một địa chỉ trỏ đến một tài nguyên trên Internet. Tài nguyên này thường là một tệp có thể được sao chép từ một máy chủ từ xa, nhưng các URL cũng có thể biểu thị các luồng dữ liệu và nội dung được tạo ở trạng thái động.

Cách Máy Khách xử lý URL

Trước khi liên hệ với máy chủ, máy khách sẽ cần chuyển đổi `learning.lpi.org` thành địa chỉ IP tương ứng của nó. Máy khách sẽ sử dụng một dịch vụ Internet khác là *Hệ thống Tên Miền* (DNS) để yêu cầu địa chỉ IP của tên máy chủ từ một hoặc nhiều máy chủ DNS được xác định trước (máy chủ DNS thường được Nhà cung cấp dịch vụ Internet hay ISP tự động xác định).

Với địa chỉ IP của máy chủ, máy khách sẽ cố gắng kết nối với cổng HTTP hoặc HTTPS. Các cổng mạng là các số nhận dạng được *Giao thức Điều khiển Truyền dẫn* (TCP) xác định để đan xen và xác định các kênh liên lạc riêng biệt trong một kết nối máy khách/máy chủ. Theo mặc định, máy chủ HTTP sẽ nhận yêu cầu trên cổng TCP 80 (HTTP) và 443 (HTTPS).

NOTE

Có các giao thức khác được các ứng dụng web sử dụng để thực hiện giao tiếp máy khách/máy chủ. Ví dụ: đối với các cuộc gọi thoại và video, sẽ phù hợp hơn khi sử dụng WebSockets - một giao thức cấp thấp hơn và hiệu quả hơn HTTP trong việc truyền luồng dữ liệu theo cả hai hướng.

Định dạng của thông báo yêu cầu mà máy khách gửi đến máy chủ trong HTTP và HTTPS là giống

nhau. HTTPS hiện nay đã được sử dụng rộng rãi hơn HTTP bởi tất cả các trao đổi dữ liệu giữa máy khách và máy chủ đều được mã hóa; đây là một tính năng không thể thiếu để thúc đẩy quyền riêng tư và bảo mật trên các mạng công cộng. Kết nối được mã hóa sẽ được thiết lập giữa máy khách và máy chủ ngay cả trước khi bất kỳ thông báo HTTP nào được trao đổi sử dụng giao thức mã hóa *Bảo mật Tầng Vận tải* (TLS). Bằng cách này, tất cả các giao tiếp HTTPS sẽ được gói gọn bởi TLS. Sau khi được giải mã, yêu cầu hoặc phản hồi được truyền qua HTTPS sẽ không khác gì yêu cầu hoặc phản hồi được thực hiện riêng qua HTTP.

Phần tử thứ ba của URL trong ví dụ (`/en/`) sẽ được máy chủ hiểu là vị trí hoặc đường dẫn cho tài nguyên được yêu cầu. Nếu đường dẫn không được cung cấp trong URL, vị trí mặc định `/` sẽ được sử dụng. Việc triển khai máy chủ HTTP đơn giản nhất sẽ liên kết các đường dẫn trong URL với các tệp trên hệ thống tệp nơi máy chủ đang chạy, nhưng đây chỉ là một trong nhiều tùy chọn có sẵn trên các máy chủ HTTP phức tạp.

Thông báo Yêu cầu

HTTP hoạt động thông qua một kết nối đã được thiết lập giữa máy khách và máy chủ thường được triển khai trong TCP và được mã hóa bằng TLS. Trên thực tế, một khi kết nối đáp ứng được các yêu cầu do máy chủ đặt ra đã sẵn sàng, một yêu cầu HTTP được nhập bằng tay ở dạng văn bản thuần túy sẽ có thể tạo ra phản hồi từ máy chủ. Tuy nhiên, trên thực tế, các lập trình viên hiếm khi cần triển khai các quy trình để soạn thông báo HTTP vì hầu hết các ngôn ngữ lập trình đều cung cấp các cơ chế tự động hóa việc tạo thông báo HTTP. Trong trường hợp của URL trong ví dụ (`https://learning.lpi.org/en/`), thông báo yêu cầu đơn giản nhất có thể sẽ có nội dung như sau:

```
GET /en/ HTTP/1.1
Host: learning.lpi.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: text/html
```

Từ đầu tiên của dòng đầu tiên sẽ xác định *phương thức* HTTP. Nó sẽ xác định hoạt động nào mà máy khách muốn thực hiện trên máy chủ. Phương thức GET sẽ thông báo cho máy chủ rằng máy khách đang yêu cầu tài nguyên theo sau nó, tức `/en/`. Cả máy khách và máy chủ đều có thể hỗ trợ nhiều hơn một phiên bản của giao thức HTTP; vì vậy, phiên bản được sử dụng trong quá trình trao đổi dữ liệu cũng sẽ được cung cấp trong dòng đầu tiên là `HTTP/1.1`.

NOTE

Phiên bản mới nhất của giao thức HTTP là HTTP/2. Ngoài một số khác biệt chung, các tin nhắn được viết bằng HTTP/2 sẽ được mã hóa theo cấu trúc nhị phân, trong khi các tin nhắn được viết bằng HTTP/1.1 sẽ được gửi ở dạng văn bản thuần túy. Sự thay đổi này giúp tối ưu hóa tốc độ truyền tải dữ liệu, nhưng về cơ bản thì nội dung

> các tin nhắn vẫn không thay đổi.

Tiêu đề có thể chứa nhiều dòng hơn sau dòng đầu tiên để bối cảnh hóa và giúp xác định yêu cầu tới máy chủ. Ví dụ như trường tiêu đề `Host` (máy chủ) trông có vẻ hơi dư thừa vì máy chủ của máy chủ rõ ràng đã được máy khách xác định để thiết lập kết nối, và việc cho rằng máy chủ phải biết được danh tính của chính nó cũng là điều hợp lý. Tuy nhiên, điều quan trọng là phải thông báo cho máy chủ về tên máy chủ dự kiến trong tiêu đề của yêu cầu vì việc sử dụng cùng một máy chủ HTTP để lưu trữ nhiều trang web là khá thông dụng (trong trường hợp này, mỗi máy chủ cụ thể sẽ được gọi là *máy chủ ảo*). Do đó, trường `Host` sẽ được máy chủ HTTP sử dụng để xác định máy chủ mà yêu cầu đề cập đến.

Trường tiêu đề `User-Agent` sẽ chứa thông tin chi tiết về chương trình máy khách gửi yêu cầu. Trường này có thể được máy chủ sử dụng để điều chỉnh phản hồi theo nhu cầu của một máy khách cụ thể, nhưng nó lại thường được sử dụng để tạo số liệu thống kê về các máy khách sử dụng máy chủ.

Trường `Accept` (Chấp nhận) sẽ có giá trị ngay tức thì vì nó sẽ thông báo cho máy chủ về định dạng của tài nguyên được yêu cầu. Nếu máy khách không quan tâm đến định dạng tài nguyên, trường `Accept` có thể chỉ định `*/*` làm định dạng.

Có nhiều trường tiêu đề khác có thể được sử dụng trong thông báo HTTP, nhưng các trường được hiển thị trong ví dụ đã là đủ để yêu cầu tài nguyên từ máy chủ.

Ngoài các trường trong tiêu đề của yêu cầu, máy khách có thể thêm vào dữ liệu bổ sung khác trong yêu cầu HTTP được gửi đến máy chủ. Nếu dữ liệu này chỉ bao gồm các tham số văn bản đơn giản ở định dạng `name=value` thì chúng có thể được thêm vào đường dẫn của phương thức GET. Các tham số sẽ được nhúng trong đường dẫn sau một dấu chấm hỏi và sẽ được phân tách bằng ký tự (`&`):

```
GET /cgi-bin/receive.cgi?name=LPI&email=info@lpi.org HTTP/1.1
```

Trong ví dụ này, `/cgi-bin/receive.cgi` là đường dẫn đến tệp lệnh trên máy chủ; máy chủ sẽ xử lý và có thể sử dụng các tham số `name` và `email` thu được từ đường dẫn của yêu cầu. Chuỗi tương ứng với các trường ở định dạng `name=LPI&email=info@lpi.org` được gọi là *chuỗi truy vấn* và sẽ được cung cấp cho tệp lệnh `receive.cgi` bởi máy chủ HTTP nhận yêu cầu.

Khi dữ liệu được tạo thành từ nhiều trường văn bản ngắn thì việc gửi dữ liệu đó trong phần tải trọng của tin nhắn sẽ phù hợp hơn. Trong trường hợp này, phương thức HTTP POST phải được sử dụng để máy chủ nhận và xử lý tải trọng của thông báo theo thông số kỹ thuật được chỉ ra trong tiêu đề của yêu cầu. Khi phương thức POST được sử dụng, tiêu đề của yêu cầu phải cung cấp kích thước của tải trọng sẽ được gửi tiếp theo và cách định dạng phần thân:

```
POST /cgi-bin/receive.cgi HTTP/1.1
Host: learning.lpi.org
Content-Length: 1503
Content-Type: multipart/form-data; boundary=------------------------405f7edfd646a37d
```

Trường `Content-Length` sẽ cho biết kích thước được tính bằng byte của tải trọng và trường `Content-Type` sẽ cho biết định dạng của nó. Định dạng `multipart/form-data` là định dạng được sử dụng phổ biến nhất trong các biểu mẫu HTML truyền thống sử dụng phương thức POST. Ở định dạng này, mỗi trường được chèn trong tải trọng của yêu cầu sẽ được phân tách bằng mã được chỉ định bởi từ khóa `boundary`. Ta chỉ nên sử dụng phương thức POST trong các trường hợp thích hợp vì nó sử dụng lượng dữ liệu lớn hơn một chút so với yêu cầu tương đương được thực hiện bằng phương thức GET. Do phương thức GET sẽ gửi các tham số trực tiếp trong tiêu đề của thông báo yêu cầu nên toàn bộ quá trình trao đổi dữ liệu sẽ có độ trễ thấp hơn do giai đoạn kết nối để truyền nội dung của thông báo có thể được bỏ qua.

Tiêu đề của Phản hồi

Sau khi máy chủ HTTP nhận được tiêu đề của thông báo yêu cầu, máy chủ sẽ trả lại thông báo phản hồi cho máy khách. Một yêu cầu tệp HTML thường sẽ có tiêu đề phản hồi như sau:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 18170
Content-Type: text/html
Date: Mon, 05 Apr 2021 13:44:25 GMT
Etag: "606adcd4-46fa"
Last-Modified: Mon, 05 Apr 2021 09:48:04 GMT
Server: nginx/1.17.10
```

Dòng đầu tiên cho biết phiên bản của giao thức HTTP được sử dụng trong thông báo phản hồi và phiên bản này phải tương ứng với phiên bản được sử dụng trong tiêu đề yêu cầu. Sau đó, vẫn ở dòng đầu tiên, sẽ là mã trạng thái của phản hồi cho biết cách máy chủ diễn giải và tạo phản hồi cho yêu cầu.

Mã trạng thái là một số có ba chữ số, trong đó chữ số ngoài cùng bên trái xác định hạng phản hồi. Có năm hạng mã trạng thái, được đánh số từ 1 đến 5, mỗi hạng biểu thị một loại hành động được thực hiện bởi máy chủ:

1xx (Thông tin)

Yêu cầu đã được nhận, tiếp tục quá trình.

2xx (Thành công)

Yêu cầu đã được nhận, hiểu và chấp nhận thành công.

3xx (Chuyển hướng)

Cần thực hiện hành động tiếp theo để hoàn thành yêu cầu.

4xx (Lỗi máy khách)

Yêu cầu chứa cú pháp sai hoặc không thể thực hiện được.

5xx (Lỗi máy chủ)

Máy chủ không thực hiện được yêu cầu dù yêu cầu có vẻ hợp lệ.

Chữ số thứ hai và thứ ba được sử dụng để chỉ ra các chi tiết bổ sung. Ví dụ như mã 200 sẽ chỉ ra rằng yêu cầu có thể được trả lời mà không gặp bất kỳ sự cố nào. Như được minh họa trong ví dụ, bạn cũng có thể cung cấp một mô tả văn bản ngắn gọn sau mã phản hồi (`OK`). Một số mã cụ thể sẽ được đặc biệt quan tâm để đảm bảo rằng máy khách HTTP có thể truy cập tài nguyên trong các tình huống bất lợi hoặc để giúp xác định lý do thất bại trong trường hợp yêu cầu không thành công:

`301 Moved Permanently`

Tài nguyên đích đã được chỉ định một URL vĩnh viễn mới, được cung cấp bởi trường tiêu đề `Location` (vị trí) trong phản hồi.

`302 Found`

Tài nguyên đích đang tạm thời nằm dưới một URL khác.

`401 Unauthorized`

Yêu cầu chưa được áp dụng vì thiếu thông tin xác thực hợp lệ cho tài nguyên đích.

`403 Forbidden`

Phản hồi `Forbidden` chỉ ra rằng, mặc dù yêu cầu hợp lệ nhưng máy chủ được cấu hình để không cung cấp nó.

`404 Not Found`

Máy chủ gốc không tìm thấy biểu diễn hiện tại cho tài nguyên đích hoặc không sẵn sàng tiết lộ biểu diễn có sẵn.

`500 Internal Server Error`

Máy chủ gặp phải tình trạng không mong muốn khiến nó không thể thực hiện được yêu cầu.

`502 Bad Gateway`

Máy chủ trong khi đóng vai trò là cổng hoặc trung gian đã nhận được phản hồi không hợp lệ từ máy chủ đầu vào mà nó đã truy cập trong khi cố gắng thực hiện yêu cầu.

Mặc dù đã cho biết rằng yêu cầu không thể thực hiện được nhưng mã trạng thái `4xx` và `5xx` ít nhất cũng cho ta thấy rằng máy chủ HTTP đang chạy và có khả năng nhận yêu cầu. Các mã `4xx` yêu cầu thực hiện một hành động đối với máy khách vì URL hoặc thông tin xác thực của nó sai. Ngược lại, mã `5xx` cho biết có điều gì đó không ổn ở phía máy chủ. Do đó, trong ngữ cảnh của các ứng dụng web, hai loại mã trạng thái này chỉ ra rằng nguồn gốc của lỗi nằm trong chính ứng dụng (hoặc của máy khách hoặc của máy chủ) chứ không phải ở trong cơ sở hạ tầng.

Nội dung Tĩnh và Động

Máy chủ HTTP sử dụng hai cơ chế cơ bản để thực hiện nội dung theo yêu cầu của khách. Cơ chế đầu tiên cung cấp *nội dung tĩnh*: nghĩa là đường dẫn được chỉ ra trong thông báo yêu cầu sẽ tương ứng với một tệp trên hệ thống tệp cục bộ của máy chủ. Cơ chế thứ hai cung cấp *nội dung động*: nghĩa là máy chủ HTTP sẽ chuyển tiếp yêu cầu tới một chương trình khác (có thể là một tệp lệnh) để tạo phản hồi từ các nguồn khác nhau, chẳng hạn như cơ sở dữ liệu và các tệp khác.

Mặc dù có các máy chủ HTTP khác nhau nhưng tất cả chúng đều sử dụng cùng một giao thức truyền thông HTTP và áp dụng ít nhiều các quy ước giống nhau. Một ứng dụng không có nhu cầu cụ thể có thể được triển khai với bất kỳ máy chủ truyền thống nào, chẳng hạn như Apache hoặc NGINX. Cả hai đều có khả năng tạo nội dung động và cung cấp nội dung tĩnh, nhưng sẽ có sự khác biệt nhỏ trong cấu hình của từng loại.

Ví dụ như vị trí của các tệp tĩnh cần được cung cấp được xác định theo các cách khác nhau trong Apache và NGINX. Quy ước ở đây là phải giữ các tệp này trong một thư mục cụ thể cho mục đích này, thư mục đó phải có tên liên quan tới máy chủ lưu trữ, ví dụ như `/var/www/learning.lpi.org/`. Trong Apache, đường dẫn này được xác định bởi chỉ thị cấu hình `DocumentRoot /var/www/learning.lpi.org` trong phần xác định máy chủ ảo. Trong NGINX, lệnh được sử dụng là `root /var/www/learning.lpi.org` trong phần `server` của tệp cấu hình.

Cho dù bạn chọn máy chủ nào, các tệp tại `/var/www/learning.lpi.org/` cũng sẽ được cung cấp qua HTTP theo cách gần như giống nhau. Một số trường trong tiêu đề phản hồi và nội dung của chúng có thể khác nhau giữa hai máy chủ, nhưng các trường như `Content-Type` (Loại Nội dung) sẽ phải có trong tiêu đề phản hồi và phải nhất quán trên mọi máy chủ.

Lưu trữ vào Bộ nhớ Đệm

HTTP được thiết kế để hoạt động trên mọi loại kết nối Internet dù là nhanh hay chậm. Hơn nữa, hầu hết các giao thức trao đổi HTTP đều phải đi qua nhiều nút mạng do kiến trúc phân tán của Internet. Do đó, điều quan trọng là phải áp dụng một số chiến lược lưu trữ nội dung trên bộ nhớ đệm để tránh việc truyền dư thừa nội dung đã tải xuống trước đó. Truyền HTTP có thể hoạt động với hai loại bộ nhớ đệm cơ bản: *chia sẻ* và *riêng tư*.

Bộ nhớ đệm chia sẻ sẽ được sử dụng bởi nhiều máy khách. Ví dụ: nhà cung cấp nội dung lớn có thể sử dụng bộ nhớ đệm trên các máy chủ được phân phối theo khu vực để máy khách lấy dữ liệu từ máy chủ gần nhất với chúng. Khi một máy khách đã đưa ra yêu cầu và phản hồi của nó được lưu trữ trong bộ nhớ đệm chia sẻ, các máy khách khác thực hiện cùng một yêu cầu trong cùng khu vực đó cũng sẽ nhận được phản hồi đã được lưu trong bộ nhớ đệm.

Bộ nhớ đệm riêng tư được tạo bởi chính máy khách để sử dụng độc quyền. Đây là loại lưu trữ vào bộ nhớ đệm mà trình duyệt web sẽ thực hiện đối với hình ảnh, tệp CSS, JavaScript hoặc chính tài liệu HTML; vì vậy, chúng sẽ không cần phải tải xuống lại nếu được yêu cầu trong tương lai gần.

NOTE

Không phải yêu cầu HTTP nào cũng phải được lưu vào bộ nhớ đệm. Ví dụ: một yêu cầu sử dụng phương thức POST sẽ đi kèm một phản hồi dành riêng cho yêu cầu cụ thể đó; vì vậy, ta không nên sử dụng lại nội dung phản hồi đó nữa. Theo mặc định, chỉ các phản hồi đối với các yêu cầu được thực hiện bằng phương thức GET mới được lưu vào bộ nhớ đệm. Hơn nữa, chỉ những phản hồi có mã trạng thái kết luận như 200 (OK), 206 (Partial Content), 301 (Moved Permanently) và 404 (Not Found) mới phù hợp để lưu vào bộ nhớ đệm.

Cả hai chiến lược bộ nhớ đệm chia sẻ và riêng tư đều sử dụng các tiêu đề HTTP để kiểm soát cách nội dung đã tải xuống sẽ được lưu vào bộ nhớ đệm như thế nào. Đối với bộ nhớ đệm riêng tư, máy khách sẽ tham khảo tiêu đề phản hồi và xác minh xem liệu nội dung trong bộ nhớ đệm cục bộ có còn tương ứng với nội dung từ xa hiện tại hay không. Nếu tương ứng, máy khách sẽ từ bỏ việc chuyển tải trọng phản hồi và sử dụng phiên bản cục bộ.

Tính hợp lệ của tài nguyên được lưu trong bộ nhớ đệm có thể được đánh giá theo nhiều cách. Máy chủ có thể cung cấp ngày hết hạn trong tiêu đề phản hồi cho yêu cầu đầu tiên để máy khách loại bỏ tài nguyên được lưu trong bộ nhớ đệm khi kết thúc thời hạn và yêu cầu lại để lấy phiên bản cập nhật. Tuy nhiên, không phải lúc nào máy chủ cũng có thể xác định ngày hết hạn của tài nguyên; do đó, người ta thường sử dụng trường tiêu đề phản hồi `ETag` để xác định phiên bản của tài nguyên, ví dụ như `Etag: "606adcd4-46fa"`.

Để xác minh xem liệu tài nguyên được lưu trong bộ nhớ đệm có cần cập nhật hay không, máy khách sẽ chỉ yêu cầu tiêu đề phản hồi của nó từ máy chủ. Nếu trường `ETag` khớp với trường trong

phiên bản được lưu trữ cục bộ, máy khách sẽ sử dụng lại nội dung được lưu trong bộ nhớ đệm. Nếu không khớp, nội dung cập nhật của tài nguyên sẽ được tải xuống từ máy chủ.

Phiên HTTP

Trong một trang web hoặc ứng dụng web thông thường, các tính năng xử lý kiểm soát phiên sẽ dựa trên các tiêu đề HTTP. Ví dụ như máy chủ không thể giả định rằng tất cả các yêu cầu đến từ cùng một địa chỉ IP là từ cùng một máy khách. Phương pháp truyền thống nhất cho phép máy chủ liên kết các yêu cầu khác nhau với một máy khách là sử dụng *cookies*, tức một loại thẻ nhận dạng được máy chủ cấp cho máy khách và được cung cấp trong tiêu đề HTTP.

Cookies cho phép máy chủ lưu giữ thông tin về một máy khách cụ thể ngay cả khi người điều hành máy khách đó không định danh bản thân họ. Cookies có thể giúp triển khai các phiên mà trong đó thông tin đăng nhập, giỏ hàng, tùy chọn, v.v., được lưu giữ giữa các yêu cầu khác nhau được thực hiện cho cùng một máy chủ đã cung cấp chúng. Cookies cũng được sử dụng để theo dõi quá trình duyệt web của người dùng; vì vậy, điều quan trọng ở đây là phải yêu cầu sự đồng ý trước khi gửi chúng.

Máy chủ sẽ đặt cookies trong tiêu đề phản hồi bằng cách sử dụng trường `Set-Cookie`. Giá trị trường là một cặp `name=value` được chọn để biểu thị một số thuộc tính được liên quan tới một máy khách cụ thể. Ví dụ: máy chủ có thể tạo số nhận dạng cho máy khách yêu cầu tài nguyên lần đầu tiên và chuyển nó cho máy khách trong tiêu đề phản hồi:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Set-Cookie: client_id=62b5b719-fcbf
```

Nếu máy khách cho phép sử dụng cookies, các yêu cầu mới gửi đến cùng một máy chủ này sẽ có trường cookies trong tiêu đề:

```
GET /en/ HTTP/1.1
Host: learning.lpi.org
Cookie: client_id=62b5b719-fcbf
```

Với số nhận dạng này, máy chủ có thể truy xuất các định nghĩa cụ thể cho máy khách và tạo một phản hồi tùy chỉnh. Cũng có thể sử dụng nhiều trường `Set-Cookie` để gửi các cookie khác nhau cho cùng một máy khách. Bằng cách này, có thể có nhiều định nghĩa được lưu giữ ở phía máy khách.

Cookies gây ra các vấn đề về cả quyền riêng tư lẫn các lỗ hổng bảo mật tiềm ẩn bởi có khả năng

chúng sẽ được chuyển sang một máy khách khác có thể được máy chủ xác định là máy khách ban đầu. Các cookies được sử dụng để duy trì các phiên có thể cấp quyền truy cập vào các thông tin nhạy cảm từ máy khách ban đầu. Do đó, điều quan trọng đối với các máy khách là áp dụng các cơ chế bảo vệ cục bộ để ngăn cookise của họ bị trích xuất và tái sử dụng trong khi không được cho phép.

Bài tập Hướng dẫn

1. Thông báo yêu cầu sau sử dụng phương thức HTTP nào?

```
POST /cgi-bin/receive.cgi HTTP/1.1
Host: learning.lpi.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: */*
Content-Length: 27
Content-Type: application/x-www-form-urlencoded
```

2. Khi một máy chủ HTTP lưu trữ nhiều trang web, làm cách nào để nó có thể xác định yêu cầu nào là dành cho trang nào?

3. Tham số nào sẽ được cung cấp bởi chuỗi truy vấn của URL `https://www.google.com/search?q=LPI`?

4. Tại sao yêu cầu HTTP sau đây lại không phù hợp với bộ nhớ đệm?

```
POST /cgi-bin/receive.cgi HTTP/1.1
Host: learning.lpi.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: */*
Content-Length: 27
Content-Type: application/x-www-form-urlencoded
```

Bài tập Mở rộng

1. Làm thế nào để có thể sử dụng trình duyệt web để theo dõi các yêu cầu và phản hồi được thực hiện bởi một trang HTML?

2. Các máy chủ HTTP cung cấp nội dung tĩnh thường ánh xạ đường dẫn được yêu cầu thành một tệp trong hệ thống tệp của máy chủ. Điều gì sẽ xảy ra khi đường dẫn trong yêu cầu trỏ đến một thư mục?

3. Nội dung của các tệp được gửi qua HTTPS được bảo vệ bằng mã hóa; vì vậy, các máy tính giữa máy khách và máy chủ không thể đọc được chúng. Mặc dù vậy, những máy tính ở giữa này có thể xác định được tài nguyên nào mà máy khách đã yêu cầu từ máy chủ không?

Tóm tắt

Bài học này đã trình bày các kiến thức cơ bản về HTTP, giao thức chính được máy khách sử dụng để yêu cầu tài nguyên từ máy chủ web. Bài học đã nhắc tới các khái niệm sau:

- Thông báo yêu cầu, trường tiêu đề và phương thức.
- Mã trạng thái phản hồi.
- Cách máy chủ HTTP tạo phản hồi.
- Các tính năng HTTP hữu ích cho bộ nhớ đệm và việc quản lý phiên.

Đáp án Bài tập Hướng dẫn

1. Thông báo yêu cầu sau sử dụng phương thức HTTP nào?

```
POST /cgi-bin/receive.cgi HTTP/1.1
Host: learning.lpi.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: */*
Content-Length: 27
Content-Type: application/x-www-form-urlencoded
```

   Phương thức POST.

2. Khi một máy chủ HTTP lưu trữ nhiều trang web, làm cách nào để nó có thể xác định yêu cầu nào là dành cho trang nào?

   Trường `Host` trong tiêu đề yêu cầu sẽ cho biết trang web mục tiêu.

3. Tham số nào sẽ được cung cấp bởi chuỗi truy vấn của URL `https://www.google.com/search?q=LPI`?

   Tham số có tên `q` với giá trị `LPI`.

4. Tại sao yêu cầu HTTP sau đây lại không phù hợp với bộ nhớ đệm?

```
POST /cgi-bin/receive.cgi HTTP/1.1
Host: learning.lpi.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0
Accept: */*
Content-Length: 27
Content-Type: application/x-www-form-urlencoded
```

   Bởi vì các yêu cầu được thực hiện bằng phương thức POST ám chỉ một thao tác ghi trên máy chủ nên chúng không được lưu vào bộ nhớ đệm.

Đáp án Bài tập Mở rộng

1. Làm thế nào để có thể sử dụng trình duyệt web để theo dõi các yêu cầu và phản hồi được thực hiện bởi một trang HTML?

   Bên cạnh nhiều các công cụ khác thì tất cả các trình duyệt phổ biến đều cung cấp các *công cụ phát triển* có thể hiển thị tất cả các giao dịch mạng đã được thực hiện bởi trang hiện tại.

2. Các máy chủ HTTP cung cấp nội dung tĩnh thường ánh xạ đường dẫn được yêu cầu thành một tệp trong hệ thống tệp của máy chủ. Điều gì sẽ xảy ra khi đường dẫn trong yêu cầu trỏ đến một thư mục?

   Việc này phụ thuộc vào cách máy chủ được cấu hình. Theo mặc định, hầu hết các máy chủ HTTP sẽ tìm kiếm một tệp có tên `index.html` (hoặc một cái tên được xác định trước khác) trong cùng thư mục đó và sẽ gửi nó dưới dạng phản hồi. Nếu tệp không có ở đó, máy chủ sẽ đưa ra phản hồi `404 Not Found`.

3. Nội dung của các tệp được gửi qua HTTPS được bảo vệ bằng mã hóa; vì vậy, các máy tính giữa máy khách và máy chủ không thể đọc được chúng. Mặc dù vậy, những máy tính ở giữa này có thể xác định được tài nguyên nào mà máy khách đã yêu cầu từ máy chủ không?

   Không, bởi vì bản thân tiêu đề của các yêu cầu và phản hồi HTTP cũng sẽ được mã hóa bởi TLS.

Linux Professional Institute

Chủ đề 032: Đánh dấu Tài liệu HTML

032.1 Cấu trúc Tài liệu HTML

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 032.1

Khối lượng

2

Các lĩnh vực kiến thức chính

- Tạo một tài liệu HTML đơn giản
- Hiểu về vai trò của HTML
- Hiểu về khung HTML
- Hiểu về cú pháp HTML (thẻ, thuộc tính, chú thích)
- Hiểu về đề mục HTML
- Hiểu thẻ meta
- Hiểu về mã hóa ký tự

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `<!DOCTYPE html>`
- `<html>`
- `<head>`
- `<body>`
- `<meta>`, bao gồm cả bộ ký tự (UTF-8), thuộc tính tên và nội dung

032.1 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	032 Đánh dấu Tài liệu HTML
Mục tiêu:	032.1 Cấu trúc tài liệu HTML
Bài:	1 trên 1

Giới thiệu

HTML (*HyperText Markup Language* - Ngôn ngữ đánh dấu siêu văn bản) là ngôn ngữ đánh dấu được sử dụng để trình duyệt web biết được cách cấu trúc và hiển thị các trang web. Phiên bản hiện tại của nó là 5.0 và được phát hành vào năm 2012. Cú pháp HTML được xác định bởi *Tổ chức Quốc tế World Wide Web* (W3C).

HTML là một kỹ năng cơ bản trong phát triển web bởi nó xác lập cấu trúc và phần lớn giao diện của một trang web. Nếu bạn muốn phát triển sự nghiệp trong lĩnh vực phát triển web, HTML hiển nhiên là một điểm khởi đầu tốt dành cho bạn.

Cấu trúc của một Tài liệu HTML

Một trang HTML cơ bản sẽ có cấu trúc như sau:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My HTML Page</title>
```

```
    <!-- This is the Document Header -->
  </head>

  <body>
    <!-- This is the Document Body -->
  </body>
</html>
```

Hãy cùng phân tích nó một cách chi tiết ngay sau đây.

Thẻ HTML

HTML sử dụng các *phần tử* và *thẻ* để mô tả và định dạng nội dung. Thẻ bao gồm các dấu ngoặc nhọn xung quanh tên thẻ, ví dụ như `<title>`. Tên thẻ không phân biệt chữ hoa và chữ thường dù Tổ chức Quốc tế World Wide Web (W3C) có khuyến nghị người dùng sử dụng các chữ cái viết thường trong các phiên bản HTML hiện tại. Các thẻ HTML này được sử dụng để xây dựng các phần tử HTML. Thẻ `<title>` là ví dụ của một thẻ *mở* trong một phần tử HTML xác định tiêu đề của tài liệu HTML. Tuy nhiên, một phần tử còn có hai thành phần nữa. Một phần tử `<title>` đầy đủ sẽ trông như thế này:

```
<title>My HTML Page</title>
```

Ở đây, `My HTML Page` đóng vai trò là thành phần *nội dung*, trong khi `</title>` đóng vai trò là thẻ *đóng* cho biết rằng phần tử này đã hoàn thiện.

NOTE Không phải tất cả các phần tử HTML đều cần phải đóng; trong những trường hợp như vậy, nó được gọi là các phần tử trống hoặc phần tử tự đóng.

Sau đây là các phần tử HTML còn lại ở trong ví dụ trước:

`<html>`

Kèm theo toàn bộ tài liệu HTML. Nó chứa tất cả các thẻ tạo nên trang. Nó cũng chỉ ra rằng nội dung của tệp này được viết bằng ngôn ngữ HTML. Thẻ đóng tương ứng của nó là `</html>`.

`<head>`

Vùng chứa tất cả các thông tin meta liên quan đến trang. Thẻ đóng tương ứng của phần tử này là `</head>`.

`<body>`

Vùng chứa nội dung của trang và biểu diễn cấu trúc của nó. Thẻ đóng tương ứng của nó là `</body>`.

Các thẻ `<html>`, `<head>` (phần mở đầu), `<body>` (phần thân) và `<title>` (tiêu đề) được gọi là *thẻ skeleton* và dùng để cung cấp cấu trúc cơ bản cho tài liệu HTML. Đặc biệt, chúng sẽ cho trình duyệt web biết rằng nó đang đọc một trang HTML.

NOTE Trong số các phần tử HTML này, phần tử duy nhất cần có để tài liệu HTML được xác thực là thẻ `<title>`.

Như có thể thấy, mỗi trang HTML là một tài liệu được cấu trúc cẩn thận và thậm chí có thể được gọi là một cây, trong đó phần tử `<html>` đại diện cho gốc tài liệu và các phần tử `<head>` và `<body>` là những nhánh đầu tiên. Ví dụ đã cho ta thấy rằng các phần tử có thể được lồng ghép vào với nhau. Ví dụ: phần tử `<title>` được lồng bên trong phần tử `<head>`, phần tử này lại được lồng bên trong phần tử `<html>`.

Để đảm bảo rằng mã HTML của bạn có thể đọc và duy trì được, hãy đảm bảo rằng tất cả các phần tử HTML được đóng đúng cách và theo thứ tự. Các trình duyệt web vẫn có thể hiển thị trang web của bạn như mong đợi, nhưng việc lồng các phần tử và thẻ của chúng không chính xác là một việc dễ gây ra lỗi.

Cuối cùng, phải đặc biệt đề cập đến khai báo *doctype* ở trên cùng của cấu trúc tài liệu ví dụ. `<!DOCTYPE>` không phải là một thẻ HTML mà là một hướng dẫn dành cho trình duyệt web chỉ định phiên bản HTML được sử dụng trong tài liệu. Trong cấu trúc tài liệu HTML cơ bản được hiển thị trước đó, `<!DOCTYPE html>` đã được sử dụng và nó chỉ định rằng HTML5 đang được sử dụng trong tài liệu này.

Chú thích HTML

Khi tạo một trang HTML, bạn nên chèn các chú thích vào mã để cải thiện khả năng đọc và mô tả mục đích của các khối mã lớn. Một chú thích có thể được chèn vào giữa các thẻ `<!--` và `-->` như minh họa trong ví dụ sau:

```
<!-- This is a comment. -->

<!--
    This is a
    multiline
    comment.
-->
```

Ví dụ này cho thấy rằng các chú thích HTML có thể được đặt trong một dòng và cũng có thể trải dài trên nhiều dòng. Trong tất cả mọi trường hợp, văn bản ở giữa `<!--` và `-->` sẽ được trình duyệt web bỏ qua và do đó không được hiển thị trong trang HTML. Dựa trên những cân nhắc này, bạn

có thể suy luận rằng trang HTML cơ bản được hiển thị trong phần trước không hề hiển thị bất kỳ văn bản nào, bởi vì các dòng `<!-- This is the Document Header -->` và `<!-- This is the Document Body -->` chỉ đơn giản là hai chú thích.

WARNING Các chú thích không thể được lồng vào nhau.

Thuộc tính HTML

Các thẻ HTML có thể bao gồm một hoặc nhiều *thuộc tính* để chỉ định chi tiết của phần tử HTML. Một thẻ đơn giản với hai thuộc tính sẽ có dạng sau:

```
<tag attribute-a="value-a" attribute-b="value-b">
```

Các thuộc tính phải luôn được đặt trên thẻ mở.

Một thuộc tính bao gồm tên (cho biết thuộc tính nào sẽ được đặt), dấu bằng và giá trị mong muốn trong dấu trích dẫn kép. Cả dấu trích dẫn đơn và kép đều có thể được chấp nhận, nhưng ta nên sử dụng một trong hai loại dấu một cách nhất quán trong suốt dự án. Điều quan trọng là không trộn lẫn dấu trích dẫn đơn và dấu trích dẫn kép cho một giá trị thuộc tính bởi trình duyệt web sẽ không nhận các kết hợp lẫn lộn giữa các dấu trích dẫn là một đơn vị.

NOTE Bạn có thể sử dụng một loại dấu trích dẫn ở bên trong loại kia mà không gặp vấn đề gì. Ví dụ: nếu bạn cần sử dụng `'` trong một giá trị thuộc tính, bạn có thể bao giá trị đó trong `"`. Tuy nhiên, nếu bạn muốn sử dụng cùng một loại dấu trích dẫn bên trong giá trị mà bạn đang sử dụng để bao giá trị , bạn cần sử dụng `&quot;` cho `"` và `&apos;` cho `'`.

Các thuộc tính có thể được phân loại thành *thuộc tính cốt lõi* và *thuộc tính cụ thể* như sẽ được giải thích trong các phần sau đây.

Thuộc tính Cốt lõi

Thuộc tính cốt lõi là thuộc tính có thể được sử dụng trên bất kỳ phần tử HTML nào. Chúng bao gồm:

`title`

Mô tả nội dung của phần tử. Giá trị của nó thường được hiển thị dưới dạng chú giải công cụ (tooltip) được hiển thị khi người dùng di chuyển con trỏ qua phần tử.

`id`

Liên kết một mã định danh duy nhất với một phần tử. Số nhận dạng này phải là duy nhất trong

tài liệu và tài liệu sẽ không hợp lệ khi nhiều thành phần chia sẻ cùng một `id`.

`style`

Gán các thuộc tính đồ họa (kiểu CSS) cho phần tử.

`class`

Chỉ định một hoặc nhiều hạng cho phần tử trong danh sách tên hạng được phân tách bằng dấu cách. Các hạng này có thể được tham chiếu trong biểu định kiểu CSS.

`lang`

Chỉ định ngôn ngữ của nội dung thành phần bằng mã ngôn ngữ hai ký tự tiêu chuẩn ISO-639.

NOTE

Nhà phát triển có thể lưu trữ thông tin tùy chỉnh về một phần tử bằng cách xác định cái gọi là thuộc tính `data-` được biểu thị bằng cách thêm `data-` vào trước tên mong muốn (như trong `data-additionalinfo`). Bạn có thể gán cho thuộc tính này một giá trị giống như bất kỳ thuộc tính nào khác.

Thuộc tính Riêng

Các thuộc tính còn lại chỉ dành riêng cho từng phần tử HTML. Ví dụ: thuộc tính `src` của phần tử HTML `<img>` chỉ định URL của hình ảnh. Còn nhiều thuộc tính riêng khác nữa sẽ được đề cập tới trong các bài học sau.

Tiêu đề của Tài liệu

Tiêu đề của tài liệu sẽ xác định thông tin meta liên quan đến trang và được mô tả bởi phần tử `<head>`. Theo mặc định, thông tin trong tiêu đề của tài liệu sẽ không được trình duyệt web hiển thị. Mặc dù có thể sử dụng phần tử `<head>` để chứa các phần tử HTML có thể được hiển thị trên trang nhưng trên thực tế ta lại không nên làm như vậy.

Title

Tiêu đề của tài liệu sẽ được xác định bằng phần tử `<title>`. Tiêu đề được xác định giữa các thẻ sẽ xuất hiện trong thanh tiêu đề của trình duyệt web và sẽ là tên gợi ý cho thẻ đánh dấu trang khi bạn muốn đánh dấu một trang. Nó cũng sẽ được hiển thị trong kết quả của công cụ tìm kiếm dưới dạng tiêu đề của trang.

Sau đây là một ví dụ về phần tử này:

```
<title>My test page</title>
```

Thẻ `<title>` được yêu cầu trong tất cả các tài liệu HTML và chỉ xuất hiện một lần trong mỗi tài liệu.

NOTE Đừng nhầm lẫn tiêu đề của tài liệu với tiêu đề của trang được đặt trong phần thân.

Siêu Dữ liệu

Phần tử `<meta>` được sử dụng để chỉ định thông tin meta nhằm mô tả thêm nội dung của tài liệu HTML. Nó được gọi là phần tử tự đóng, tức là nó không có thẻ đóng. Ngoài các thuộc tính cốt lõi hợp lệ cho mọi phần tử HTML, phần tử `<meta>` còn sử dụng các thuộc tính sau:

`name`

Xác định siêu dữ liệu nào sẽ được mô tả trong phần tử này. Nó có thể được đặt thành bất kỳ giá trị được xác định tùy chỉnh nào, nhưng các giá trị thường được sử dụng là `author` (tác giả), `description` (mô tả) và `keywords` (từ khóa).

`http-equiv`

Cung cấp tiêu đề HTTP cho giá trị của thuộc tính `content` (nội dung). Một giá trị phổ biến là `refresh` (làm mới); giá trị này sẽ được giải thích sau. Nếu thuộc tính này được đặt thì ta không nên đặt thuộc tính `name`.

`content`

Cung cấp giá trị được liên quan tới thuộc tính `name` hoặc `http-equiv`.

`charset`

Chỉ định mã ký tự cho tài liệu HTML, ví dụ như `utf-8` để đặt nó thành Định dạng Chuyển đổi Unicode — 8-bit.

Thêm Thông tin về Tác giả, Mô tả và Từ khóa

Bằng cách sử dụng thẻ `<meta>`, ta có thể chỉ định thông tin bổ sung về tác giả của trang HTML và mô tả nội dung trang như sau:

```
<meta name="author" content="Name Surname">
<meta name="description" content="A short summary of the page content.">
```

Hãy thử cho thêm một loạt các từ khóa liên quan đến nội dung của trang trong mô tả. Mô tả này thường sẽ là điều đầu tiên người dùng nhìn thấy khi điều hướng bằng công cụ tìm kiếm.

Nếu bạn cũng muốn cung cấp các từ khóa bổ sung liên quan đến trang web cho các công cụ tìm kiếm, bạn có thể thêm phần tử này:

```
<meta name="keywords" content="keyword1, keyword2, keyword3, keyword4, keyword5">
```

NOTE

Trước đây, những kẻ gửi thư rác đã nhập hàng trăm từ khóa và mô tả không liên quan đến nội dung thực tế của trang để nó cũng xuất hiện trong các kết quả tìm kiếm không liên quan đến cụm từ mà mọi người đã tìm kiếm. Ngày nay, các thẻ `<meta>` đã trở thành thứ yếu và chỉ được sử dụng để hợp nhất các chủ đề có trong trang web; do đó, người ta không còn có thể sử dụng nó để đánh lừa các thuật toán của các công cụ tìm kiếm mới và tinh vi.

Chuyển hướng trang HTML và Xác định khoảng thời gian để Tài liệu tự làm mới

Bằng cách sử dụng thẻ `<meta>`, ta có thể tự động làm mới trang HTML sau một khoảng thời gian nhất định (ví dụ: sau 30 giây) theo cách này:

```
<meta http-equiv="refresh" content="30">
```

Ngoài ra, bạn có thể chuyển hướng một trang web đến một trang web khác sau cùng một khoảng thời gian với đoạn mã sau:

```
<meta http-equiv="refresh" content="30; url=http://www.lpi.org">
```

Trong ví dụ này, người dùng sẽ được chuyển hướng từ trang hiện tại đến `http://www.lpi.org` sau 30 giây. Các giá trị có thể là bất cứ thứ gì bạn muốn. Ví dụ: nếu bạn chỉ định `content="0; url=http://www.lpi.org"`, trang sẽ được chuyển hướng ngay lập tức.

Chỉ định Mã hóa Ký tự

Thuộc tính `charset` (bộ ký tự) chỉ định mã hóa ký tự cho tài liệu HTML. Một ví dụ phổ biến là:

```
<meta charset="utf-8">
```

Phần tử này chỉ định rằng mã hóa ký tự của tài liệu là `utf-8`; đây là một bộ ký tự chung thực tế bao gồm bất kỳ ký tự nào của bất kỳ ngôn ngữ nào của con người. Do đó, bằng cách sử dụng nó, ta sẽ tránh được các sự cố khi hiển thị một số ký tự ta có thể gặp phải khi sử dụng các bộ ký tự khác, chẳng hạn như ISO-8859-1 (bảng chữ cái Latinh).

Các Ví dụ hữu ích khác

Hai ứng dụng hữu ích khác của thẻ `<meta>` là:

- Thiết lập cookies để theo dõi khách truy cập trang web.
- Kiểm soát chế độ xem (khu vực hiển thị của trang web bên trong cửa sổ trình duyệt web) tùy thuộc vào kích thước màn hình của thiết bị người dùng (ví dụ như điện thoại di động hoặc máy tính).

Tuy nhiên, hai ví dụ này nằm ngoài phạm vi của bài kiểm tra và việc nghiên cứu chúng sẽ chỉ dành cho những độc giả có trí tò mò và muốn khám phá trong phạm vị rộng hơn.

Bài tập Hướng dẫn

1. Đối với mỗi thẻ sau, hãy cho biết thẻ đóng tương ứng:

`<body>`	
`<head>`	
`<html>`	
`<meta>`	
`<title>`	

2. Sự khác biệt giữa thẻ và phần tử là gì? Tham khảo mục sau đây:

```
<title>HTML Page Title</title>
```

3. Các thẻ nào có thể có chú thích đặt giữa chúng?

4. Hãy giải thích thuộc tính là gì và cho một số ví dụ về thẻ `<meta>`.

Bài tập Mở rộng

1. Hãy tạo một tài liệu HTML phiên bản 5 đơn giản với tiêu đề `My first HTML document` và chỉ có một đoạn trong phần nội dung chứa đoạn văn bản `Hello World`. Hãy s dụng thẻ đoạn văn `<p>` trong phần thân.

2. Hãy thêm tác giả (`Kevin Author`) và mô tả (`This is my first HTML page.`) của tài liệu HTML.

3. Hãy thêm các từ khóa sau liên quan đến tài liệu HTML: `HTML`, `Example`, `Test` và `Metadata`.

4. Hãy thêm phần tử `<meta charset="ISO-8859-1">` vào tiêu đề của tài liệu và thay đổi văn bản `Hello World` thành tiếng Nhật (□□□□□□□). Điều gì sẽ xảy ra? Bạn sẽ giải quyết vấn đề này như thế nào?

5. Sau khi thay đổi văn bản đoạn trở lại thành `Hello World`, hãy chuyển hướng trang HTML đến `https://www.google.com` sau 30 giây và thêm một chú thích để giải thích điều này trong tiêu đề của tài liệu.

Tóm tắt

Trong bài học này, bạn đã học về:

- Vai trò của HTML
- Skeleton trong HTML
- Cú pháp HTML (thẻ, thuộc tính, chú thích)
- Phần đầu của HTML
- Các thẻ meta
- Cách tạo một tài liệu HTML đơn giản

Các thuật ngữ sau đây đã được thảo luận trong bài học này:

`<!DOCTYPE html>`
: Thẻ khai báo.

`<html>`
: Vùng chứa cho tất cả các thẻ tạo nên trang HTML.

`<head>`
: Vùng chứa cho tất cả các phần tử phần đầu.

`<body>`
: Vùng chứa cho tất cả các phần tử phần thân.

`<meta>`
: Thẻ dành cho siêu dữ liệu được sử dụng để chỉ định thông tin bổ sung cho trang HTML (chẳng hạn như tác giả, mô tả và mã hóa ký tự).

Đáp án Bài tập Hướng dẫn

1. Đối với mỗi thẻ sau, hãy cho biết thẻ đóng tương ứng:

`<body>`	`</body>`
`<head>`	`</head>`
`<html>`	`</html>`
`<meta>`	None
`<title>`	`</title>`

2. Sự khác biệt giữa thẻ và phần tử là gì? Tham khảo mục sau đây:

   `<title>HTML Page Title</title>`

   Một phần tử HTML sẽ bao gồm thẻ mở, thẻ đóng và mọi thứ nằm ở giữa chúng. Thẻ HTML được sử dụng để đánh dấu phần đầu hoặc phần cuối của một phần tử. Do đó, `<title>HTML Page Title</title>` là một thành phần HTML, trong khi `<title>` và `</title>` lần lượt là thẻ mở và thẻ đóng.

3. Các thẻ nào có thể có chú thích đặt giữa chúng?

   Một chú thích sẽ được chèn vào giữa các thẻ `<!--` và `-->` và có thể được đặt trên một hoặc nhiều dòng.

4. Hãy giải thích thuộc tính là gì và cho một số ví dụ về thẻ `<meta>`.

   Đây là một thuộc tính được sử dụng để xác định chính xác hơn một phần tử HTML. Ví dụ: thẻ `<meta>` sử dụng cặp thuộc tính `name` và `content` để thêm tác giả và mô tả của trang HTML. Thay vào đó, bằng cách sử dụng thuộc tính `charset`, bạn có thể chỉ định mã hóa ký tự cho tài liệu HTML.

Đáp án Bài tập Mở rộng

1. Hãy tạo một tài liệu HTML phiên bản 5 đơn giản với tiêu đề `My first HTML document` và chỉ có một đoạn trong phần nội dung chứa đoạn văn bản `Hello World`. Hãy s dụng thẻ đoạn văn `<p>` trong phần thân.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first HTML document</title>
  </head>

  <body>
    <p>Hello World</p>
  </body>
</html>
```

2. Hãy thêm tác giả (`Kevin Author`) và mô tả (`This is my first HTML page.`) của tài liệu HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first HTML document</title>
    <meta name="author" content="Kevin Author">
    <meta name="description" content="This is my first HTML page.">
  </head>

  <body>
    <p>Hello World</p>
  </body>
</html>
```

3. Hãy thêm các từ khóa sau liên quan đến tài liệu HTML: `HTML`, `Example`, `Test` và `Metadata`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first HTML document</title>
    <meta name="author" content="Kevin Author">
    <meta name="description" content="This is my first HTML page.">
```

```
    <meta name="keywords" content="HTML, Example, Test, Metadata">
  </head>

  <body>
    <p>Hello World</p>
  </body>
</html>
```

4. Hãy thêm phần tử `<meta charset="ISO-8859-1">` vào tiêu đề của tài liệu và thay đổi văn bản `Hello World` thành tiếng Nhật (□□□□□□□□). Điều gì sẽ xảy ra? Bạn sẽ giải quyết vấn đề này như thế nào?

   Nếu ví dụ được thực hiện như mô tả, văn bản tiếng Nhật sex không được hiển thị chính xác. Điều này là do ISO-8859-1 đại diện cho mã hóa ký tự cho bảng chữ cái Latinh. Để xem văn bản, bạn cần thay đổi mã hóa ký tự, chẳng hạn nhw sử dụng UTF-8 (`<meta charset="utf-8">`).

5. Sau khi thay đổi văn bản đoạn trở lại thành `Hello World`, hãy chuyển hướng trang HTML đến `https://www.google.com` sau 30 giây và thêm một chú thích để giải thích điều này trong tiêu đề của tài liệu.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first HTML document</title>
    <meta name="author" content="Kevin Author">
    <meta name="description" content="This is my first HTML page.">
    <meta name="keywords" content="HTML, Example, Test, Metadata">
    <meta charset="utf-8">
    <!-- The page is redirected to Google after 30 seconds -->
    <meta http-equiv="refresh" content="30; url=https://www.google.com">
  </head>

  <body>
    <p>Hello World</p>
  </body>
</html>
```

032.2 Ngữ nghĩa trong HTML và Phân cấp Tài liệu

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 032.2

Khối lượng

2

Các lĩnh vực kiến thức chính

- Tạo đánh dấu cho nội dung trong tài liệu HTML
- Hiểu về cấu trúc văn bản HTML phân cấp
- Phân biệt giữa các phần tử HTML khối và nội tuyến
- Hiểu về các phần tử cấu trúc ngữ nghĩa quan trọng trong HTML

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`
- `<p>`
- `<ul>`, `<ol>`, `<li>`
- `<dl>`, `<dt>`, `<dd>`
- `<pre>`
- `<blockquote>`
- `<strong>`, `<em>`, `<code>`
- `<b>`, `<i>`, `<u>`
- `<span>`
- `<div>`

- `<main>`, `<header>`, `<nav>`, `<section>`, `<footer>`

Linux Professional Institute

032.2 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	032 Đánh dấu Tài liệu HTML
Mục tiêu:	032.1 Ngữ nghĩa trong HTML và Phân cấp Tài liệu
Bài:	1 trên 1

Giới thiệu

Trong bài học trước, chúng ta đã biết HTML là một ngôn ngữ đánh dấu có thể mô tả về mặt ngữ nghĩa nội dung của một trang web. Tài liệu HTML chứa cái được gọi là khung xương bao gồm các phần tử HTML `<html>`, `<head>` và `<body>`. Trong khi phần tử `<head>` mô tả một khối thông tin meta cho tài liệu HTML không hiển thị đối với khách truy cập vào trang web thì phần tử `<body>` có thể chứa nhiều các phần tử khác giúp xác định cấu trúc và nội dung của tài liệu HTML.

Trong bài học này, chúng ta sẽ tìm hiểu về định dạng văn bản, các phần tử HTML ngữ nghĩa cơ bản, mục đích của chúng cũng như cách cấu trúc một tài liệu HTML. Chúng ta sẽ sử dụng một danh sách mua sắm làm ví dụ.

NOTE Tất cả các ví dụ mã tiếp theo sẽ nằm trong phần tử `<body>` của tài liệu HTML có một khung xương hoàn chỉnh. Để dễ theo dõi, khung xương của HTML sẽ không nhất thiết phải được trình bày đầy đủ trong mọi ví dụ của bài học này.

Văn bản

Trong HTML, không có khối văn bản nào nên được để trống và đứng bên ngoài một phần tử. Ngay cả một đoạn văn bản ngắn cũng phải được bao bọc ở bên trong các thẻ HTML `<p>` là viết tắt của *paragraph* (đoạn).

```
<p>Short text element spanning only one line.</p>
<p>A text element containing much longer text that may span across multiple lines, depending
on the size of the web browser window.</p>
```

Khi được mở bằng trình duyệt web, mã HTML này sẽ tạo ra kết quả được hiển thị trong Figure 1.

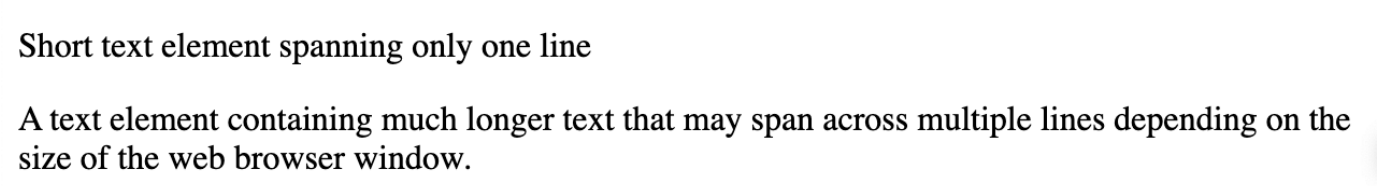

Figure 1. Phần trình bày trên trình duyệt web của mã HTML sẽ hiển thị hai đoạn văn bản. Đoạn đầu sẽ rất ngắn. Đoạn thứ hai dài hơn một chút và kết thúc thành một dòng thứ hai.

Theo mặc định, các trình duyệt web sẽ thêm khoảng cách vào trước và sau các phần tử `<p>` để tiện theo dõi. Vì lý do này, `<p>` còn được gọi là *phần tử khối*.

Đề mục

HTML xác định sáu cấp độ đề mục để mô tả và cấu trúc nội dung của tài liệu HTML. Các tiêu đề này sẽ được đánh dấu bằng các thẻ HTML `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` và `<h6>`.

```
<h1>Heading level 1 to uniquely identify the page</h1>
<h2>Heading level 2</h2>
<h3>Heading level 3</h3>
<h4>Heading level 4</h4>
<h5>Heading level 5</h5>
<h6>Heading level 6</h6>
```

Một trình duyệt web sẽ hiển thị mã HTML này như được hiển thị trong Figure 2.

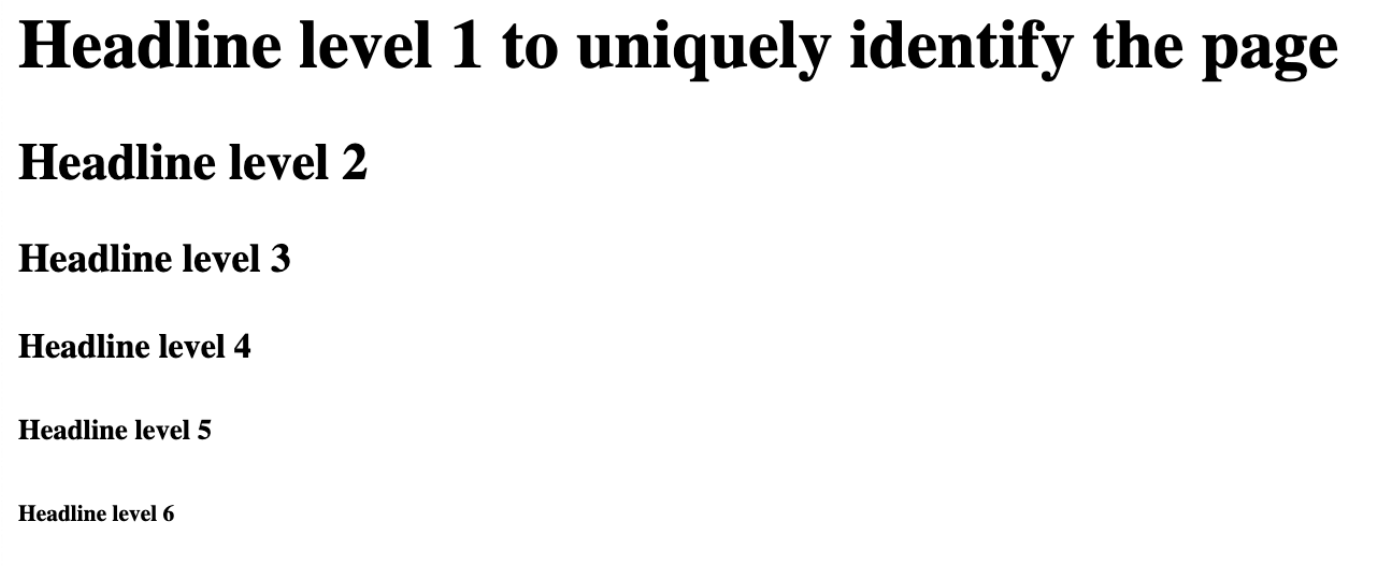

Figure 2. Phần trình bày trên trình duyệt web của mã HTML sẽ hiển thị các cấp độ đề mục khác nhau trong tài liệu HTML. Thứ bậc của các đề mục sẽ được biểu thị thông qua kích thước của văn bản.

Nếu đã quen thuộc với các trình xử lý văn bản như LibreOffice hoặc Microsoft Word, bạn có thể sẽ nhận ra được một số điểm tương đồng trong cách tài liệu HTML sử dụng các cấp độ đề mục khác nhau và cách chúng được hiển thị trong trình duyệt web. Theo mặc định, HTML sẽ sử dụng kích thước để biểu thị thứ bậc và tầm quan trọng của các tiêu đề, đồng thời thêm khoảng trắng vào trước và sau mỗi tiêu đề để tách biệt nó khỏi nội dung một cách trực quan.

Đề mục sử dụng phần tử `<h1>` nằm ở đầu phân cấp và do đó được coi là đề mục quan trọng nhất giúp xác định nội dung của trang. Chir tính trong phạm vi nội dung của tài liệu HTML, nó có thể so sánh với phần tử `<title>` (tiêu đề) đã được thảo luận trong bài học trước. Các phần tử đề mục tiếp theo có thể được sử dụng để cấu trúc thêm nội dung. Hãy đảm bảo rằng bạn không bỏ qua các cấp đề mục ở giữa. Hệ thống phân cấp tài liệu nên bắt đầu bằng `<h1>`, tiếp tục với `<h2>`, rồi `<h3>`, v.v. Bạn không cần sử dụng mọi phần tử đề mục cho đến `<h6>` nếu nội dung của bạn không yêu cầu điều này.

NOTE

Đề mục là công cụ quan trọng để cấu trúc một tài liệu HTML cả về mặt ngữ nghĩa và lẫn khía cạnh trực quan. Tuy nhiên, các đề mục không được sử dụng để tăng kích thước của văn bản không quan trọng về mặt cấu trúc. Theo nguyên tắc tương tự, ta không nên in đậm hoặc in nghiêng một đoạn văn bản ngắn để làm cho nó trông giống như một đề mục; hãy sử dụng các thẻ đề mục để đánh dấu các đề mục.

Hãy bắt đầu tạo tài liệu HTML cho danh sách mua sắm bằng cách xác định dàn ý của nó. Chúng ta sẽ tạo một phần tử `<h1>` để chứa tiêu đề trang (trong trường hợp này là `Garden Party`), theo sau là một đoạn thông tin ngắn được bao bọc trong phần tử `<p>`. Ngoài ra, ta sẽ sử dụng hai phần tử

`<h2>` để giới thiệu hai phần nội dung `Agenda` (Nội dung Chương trình) và `Please bring` (Hãy mang theo).

```
<h1>Garden Party</h1>
<p>Invitation to John's garden party on Saturday next week.</p>
<h2>Agenda</h2>
<h2>Please bring</h2>
```

Khi được mở bằng trình duyệt web, mã HTML này sẽ tạo ra kết quả được hiển thị trong Figure 3.

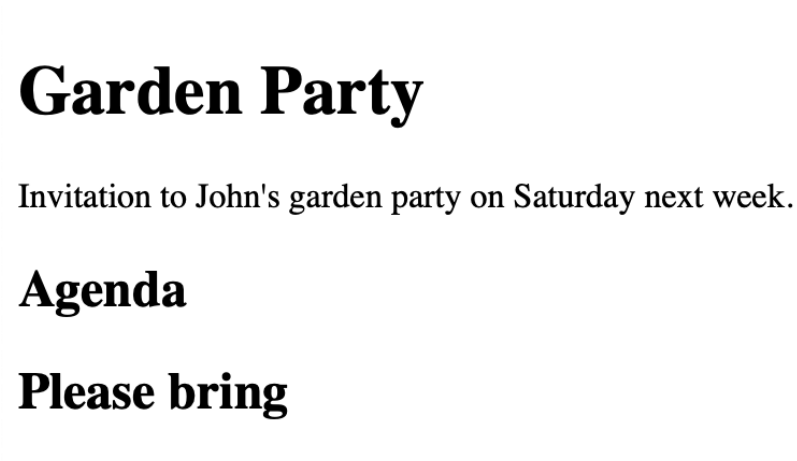

Figure 3. Phần trình bày trên trình duyệt web của mã HTML đang hiển thị một tài liệu ví dụ đơn giản mô tả lời mời dự tiệc ngoài vườn cùng với hai tiêu đề cho chương trình của bữa tiệc và danh sách những thứ nên mang theo.

Ngắt Dòng

Đôi khi chúng ta sẽ cần phải *ngắt dòng* mà không chèn thêm một phần tử `<p>` khác hoặc bất kỳ phần tử khối tương tự nào. Trong những trường hợp như vậy, ta có thể sử dụng phần tử `<br>` tự đóng. Hãy lưu ý rằng nó chỉ nên được sử dụng để chèn ngắt dòng trong nội dung như thơ, lời bài hát hoặc địa chỉ. Nếu nội dung được phân tách theo ý nghĩa, tốt hơn hết là nên sử dụng phần tử `<p>`.

Ví dụ: chúng ta có thể tách văn bản trong đoạn thông tin từ ví dụ trước như sau:

```
<p>
  Invitation to John's garden party.<br>
  Saturday, next week.
</p>
```

Khi được mở bằng trình duyệt web, mã HTML này sẽ tạo ra kết quả được hiển thị trong Figure 4.

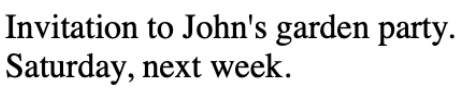

Figure 4. Phần trình bày trên trình duyệt web của mã HTML đang hiển thị một tài liệu ví dụ đơn giản với một ngắt dòng bắt buộc.

Dòng Ngang

Phần tử `<hr>` xác định một dòng ngang hay còn được gọi là *quy tắc ngang*. Theo mặc định, nó sẽ kéo dài toàn bộ chiều rộng của phần tử mẹ của nó. Phần tử `<hr>` có thể giúp bạn xác định một thay đổi theo chủ đề trong nội dung hoặc tách các phần của tài liệu. Phần tử này có thể tự đóng và vì thế mà nó không có thẻ đóng.

Trong ví dụ, chúng ta có thể tách hai đề mục:

```
<h1>Garden Party</h1>
<p>Invitation to John's garden party on Saturday next week.</p>
<h2>Agenda</h2>
<hr>
<h2>Please bring</h2>
```

Figure 5 sẽ hiển thị kết quả của mã này.

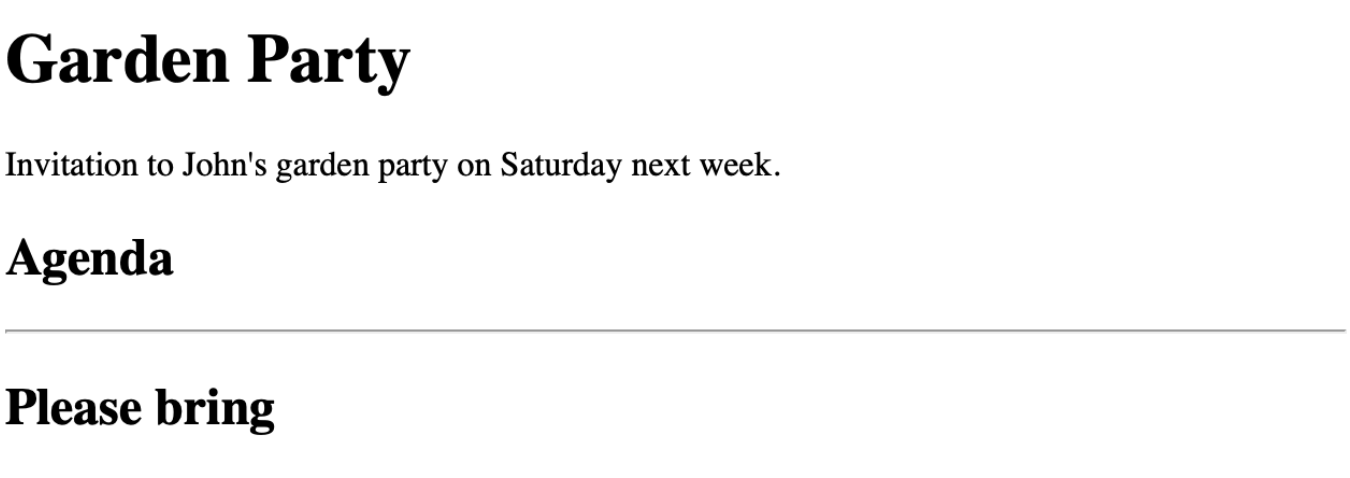

Figure 5. Phần trình bày trên trình duyệt web của mã HTML đang hiển thị một tài liệu ví dụ đơn giản mô tả một danh sách mua sắm với hai phần được phân tách bằng một dòng ngang.

Danh sách HTML

Trong HTML, bạn có thể xác định được ba loại danh sách:

Danh sách có thứ tự
: mà trong đó thứ tự của các thành phần trong danh sách là một yếu tố quan trọng

Danh sách không có thứ tự
: mà trong đó thứ tự của các thành phần trong danh sách không đặc biệt quan trọng

Danh sách mô tả
: để mô tả chính xác hơn một số thuật ngữ

Mỗi danh sách có thể chứa bất kỳ số *danh mục* nào. Chúng ta sẽ mô tả từng loại danh sách ngay sau đây.

Danh sách có Thứ tự

Một *danh sách có thứ tự* trong HTML (được biểu thị bằng phần tử HTML `<ol>`) là một tập hợp gồm bất kỳ một số lượng *danh mục* nào. Điều làm cho phần tử này trở nên đặc biệt là thứ tự của các phần tử trong danh sách này. Để nhấn mạnh điều này, các trình duyệt web sẽ hiển thị các chữ số trước các danh mục con theo mặc định.

NOTE Phần tử `<li>` là phần tử con hợp lệ duy nhất trong phần tử `<ol>`.

Trong ví dụ trên, ta có thể điền vào nội dung chương trình cho bữa tiệc ngoài vườn bằng cách sử dụng phần tử `<ol>` với mã sau:

```
<h2>Agenda</h2>
<ol>
  <li>Welcome</li>
  <li>Barbecue</li>
  <li>Dessert</li>
  <li>Fireworks</li>
</ol>
```

Khi được mở bằng trình duyệt web, mã HTML này sẽ tạo ra kết quả được hiển thị trong Figure 6.

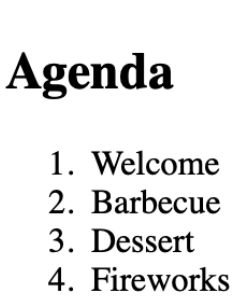

Figure 6. Phần trình bày trên trình duyệt web của mã HTML đang hiển thị một tài liệu ví dụ đơn giản chứa đề mục cấp hai, theo sau là danh sách có thứ tự với bốn mục mô tả nội dung chương trình cho một bữa tiệc ngoài vườn.

Tuỳ chọn

Như có thể thấy trong ví dụ này, các danh mục đã được đánh số bằng chữ số thập phân bắt đầu từ 1 theo mặc định. Tuy nhiên, ta có thể thay đổi hành vi này bằng cách chỉ định thuộc tính `type` của thẻ `<ol>`. Các giá trị hợp lệ cho thuộc tính này là `1` cho chữ số thập phân, `A` cho chữ hoa, `a` cho chữ thường, `I` cho chữ số La Mã viết hoa và `i` cho chữ số La Mã viết thường.

Nếu muốn, chúng ta cũng có thể xác định giá trị bắt đầu bằng cách sử dụng thuộc tính `start` của thẻ `<ol>`. Thuộc tính `start` luôn nhận một giá trị là số thập phân ngay cả khi thuộc tính `type` đã đặt một kiểu đánh số khác.

Ví dụ: chúng ta có thể điều chỉnh danh sách có thứ tự trong ví dụ trước để các mục trong danh sách sẽ được bắt đầu bằng chữ in hoa và bắt đầu bằng chữ C như được minh họa trong ví dụ sau:

```
<h2>Agenda</h2>
<ol type="A" start="3">
  <li>Welcome</li>
  <li>Barbecue</li>
  <li>Dessert</li>
  <li>Fireworks</li>
</ol>
```

Trong trình duyệt web, mã HTML này sẽ được hiển thị dưới dạng Figure 7.

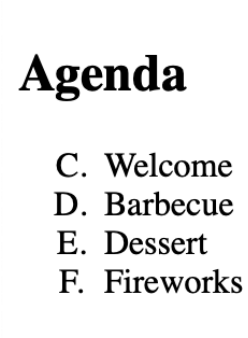

Figure 7. Phần trình bày trên trình duyệt web của mã HTML đang hiển thị một tài liệu ví dụ đơn giảnchứa đề mục cấp hai, theo sau là danh sách có thứ tự với các danh mục có tiền tố là chữ in hoa bắt đầu bằng chữ C.

Thứ tự của các danh mục trong danh sách cũng có thể được đảo ngược bằng cách sử dụng thuộc tính `reversed` mà không đi kèm với một giá trị nào.

NOTE Trong một danh sách có thứ tự, ta cũng có thể đặt giá trị ban đầu của một danh mục cụ thể bằng cách sử dụng thuộc tính `value` của thẻ `<li>`. Các danh mục theo sau sẽ tăng từ con số đó. Thuộc tính `value` sẽ luôn nhận một giá trị số thập phân.

Danh sách không có Thứ tự

Một *Danh sách không có Thứ tự* sẽ chứa một loạt các danh mục và, khác với các danh mục trong danh sách có thứ tự, các danh mục này sẽ không có một thứ tự hoặc trình tự đặc biệt nào. Phần tử HTML cho danh sách này là `<ul>`. Một lần nữa, `<li>` là phần tử HTML để đánh dấu các danhn mục trong danh sách của nó.

NOTE Phần tử `<li>` là phần tử con hợp lệ duy nhất trong phần tử `<ul>`.

Đối với trang web trong ví dụ, chúng ta có thể sử dụng danh sách không có thứ tự để liệt kê các vật phẩm để khách mang đến bữa tiệc. Chúng ta có thể đạt được điều này với mã HTML sau:

```
<h2>Please bring</h2>
<ul>
  <li>Salad</li>
  <li>Drinks</li>
  <li>Bread</li>
  <li>Snacks</li>
  <li>Desserts</li>
</ul>
```

Trong trình duyệt web, mã HTML này sẽ tạo ra phần được hiển thị trong Figure 8.

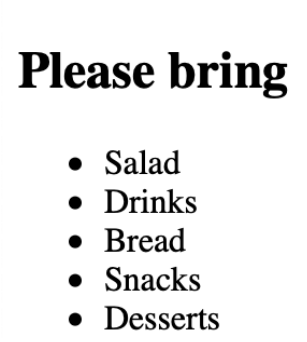

Figure 8. Phần trình bày trên trình duyệt web của mã HTML đang hiển thị một tài liệu ví dụ đơn giản chứa một đề mục cấp hai, theo sau là một danh sách không có thứ tự với tên các vật phẩm liên quan đến thực phẩm mà khách được yêu cầu mang đến bữa tiệc ngoài vườn.

Theo mặc định, mỗi danh mục sẽ được biểu thị bằng ký hiệ đĩa đàu dòng. Chúng ta sẽ có thể thay đổi giao diện của nó bằng CSS (sẽ được thảo luận trong các bài học sau).

Lồng Danh sách

Các danh sách có thể được lồng vào với nhau, chẳng hạn như danh sách có thứ tự có thể được lồng trong danh sách không có thứ tự và ngược lại. Để đạt được điều này, danh sách lồng phải là một phần của phần tử danh sách `<li>` vì `<li>` là phần tử con hợp lệ duy nhất của cả danh sách không có thứ tự và có thứ tự. Khi lồng chúng với nhau, hãy cẩn thận để không chồng chéo lẫn lộn các thẻ HTML.

Trong ví dụ đang được sử dụng, chúng ta có thể thêm một số thông tin về nội dung chương trình đã được tạo trước đây, ví dụ như sau:

```
<h2>Agenda</h2>
<ol type="A" start="3">
  <li>Welcome</li>
  <li>
    Barbecue
    <ul>
      <li>Vegetables</li>
      <li>Meat</li>
      <li>Burgers, including vegetarian options</li>
    </ul>
  </li>
  <li>Dessert</li>
  <li>Fireworks</li>
</ol>
```

Trình duyệt web sẽ hiển thị mã như được hiển thị trong Figure 9.

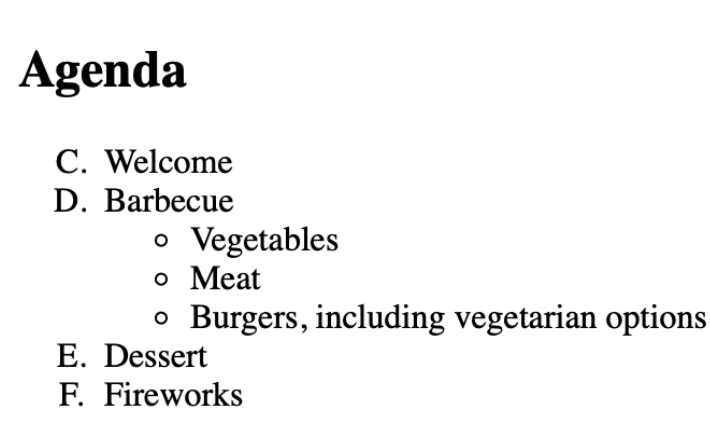

Figure 9. Phần trình bày trên trình duyệt web của mã HTML đang hiển thị một danh sách không có thứ tự được lồng trong một danh sách có thứ tự để thể hiện nội dung của chương trình dành cho một bữa tiệc ngoài vườn.

Bạn cũng có thể tiến xa hơn nữa và lồng nhiều cấp độ sâu hơn nữa. Về mặt lý thuyết, không có giới hạn về số lượng các danh sách có thể lồng vào nhau. Tuy nhiên, hãy cân nhắc về khả năng đọc của khách truy cập khi làm điều này.

Danh sách Mô tả

Một *danh sách mô tả* sẽ được xác định bằng cách sử dụng phần tử `<dl>` và sẽ đại diện cho một từ điển của các *khoá* và *giá trị*. Khóa là một thuật ngữ hoặc một cái tên mà bạn muốn mô tả và giá trị là phần mô tả. Danh sách mô tả có thể bao gồm các cặp khóa-giá trị từ đơn giản cho đến những định nghĩa mở rộng.

Khóa (hoặc *thuật ngữ*) sẽ được xác định bằng phần tử `<dt>`, còn mô tả của nó thì được xác định bằng phần tử `<dd>`.

Một ví dụ cho danh sách mô tả như vậy có thể là danh sách các loại trái cây kỳ lạ (exotic fruits) và các phần giải thích về việc chúng trông như thế nào.

```
<h3>Exotic Fruits</h3>
<dl>
  <dt>Banana</dt>
  <dd>
    A long, curved fruit that is yellow-skinned when ripe. The fruit's skin
    may also have a soft green color when underripe and get brown spots when
    overripe.
  </dd>
```

```
  <dt>Kiwi</dt>
  <dd>
    A small, oval fruit with green flesh, black seeds, and a brown, hairy
    skin.
  </dd>

  <dt>Mango</dt>
  <dd>
    A fruit larger than a fist, with a green skin, orange flesh, and one big
    seed. The skin may have spots ranging from green to yellow or red.
  </dd>
</dl>
```

Trong trình duyệt web, điều này sẽ tạo ra kết quả được hiển thị trong Figure 10.

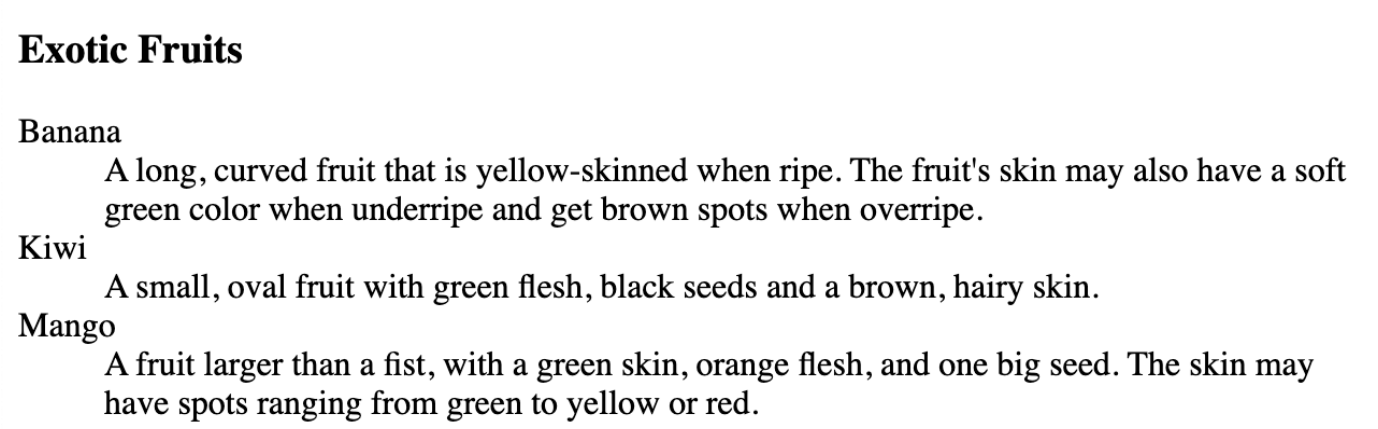

Figure 10. Một ví dụ về danh sách mô tả HTML sử dụng các loại trái cây kỳ lạ. Danh sách đã mô tả hình dạng của ba loại trái cây khác nhau.

NOTE Trái ngược với danh sách có thứ tự và danh sách không có thứ tự, trong danh sách mô tả, bất kỳ phần tử HTML nào cũng có thể được coi là một phần tử con trực tiếp hợp lệ. Điều này cho phép ta nhóm các phần tử và thiết kế cho chúng cả ở nơi khác thông qua việc sử dụng CSS.

Định dạng Văn bản Nội tuyến

Trong HTML, chúng ta có thể sử dụng các phần tử định dạng để thay đổi hình thức của văn bản. Các phần tử này có thể được phân loại thành *phần tử trình bày* hoặc *phần tử cụm từ*.

Phần tử Trình bày

Các phần tử trình bày cơ bản sẽ thay đổi phông chữ hoặc hình thức của văn bản. Chúng là `<b>`, `<i>`, `<u>` và `<tt>`. Những phần tử này đã được xác định ngay từ ban đầu trước khi CSS cho phép văn bản có thể được in đậm, in nghiêng, v.v. Hiện nay đã có nhiều cách hay hơn để thay đổi hình thức của văn bản, nhưng đôi khi bạn vẫn sẽ thấy những phần tử này.

Văn bản in đậm

Để in đậm văn bản, chúng ta sẽ gói văn bản đó vào trong phần tử `<b>` như được minh họa trong ví dụ sau. Kết quả sẽ hiển thị trong Figure 11.

```
This <b>word</b> is bold.
```

This **word** is bold.

Figure 11. Thẻ `<b>` dùng để in đậm văn bản.

Theo như đặc tính của HTML 5, ta chỉ nên sử dụng phần tử `<b>` khi không còn thẻ nào thích hợp hơn. Một phần tử khác cũng tạo ra cùng một kết quả trực quan nhưng lại bổ sung thêm tầm quan trọng về mặt ngữ nghĩa cho văn bản được đánh dấu là `<strong>`.

Văn bản in nghiêng

Để in nghiêng văn bản, chúng ta sẽ gói văn bản đó trong phần tử `<i>` như được minh họa trong ví dụ sau. Kết quả sẽ hiển thị trong Figure 12.

```
This <i>word</i> is in italics.
```

This *word* is in italics.

Figure 12. Thẻ `<i>` dùng để in nghiêng văn bản.

Theo như đặc tính của HTML 5, ta chỉ nên sử dụng phần tử `<i>` khi không còn thẻ nào thích hợp hơn.

Văn bản được gạch chân

Để gạch chân văn bản, chúng ta sẽ gói văn bản đó trong phần tử `<u>` như được minh họa trong ví dụ sau. Kết quả sẽ hiển thị trong Figure 13.

```
This <u>word</u> is underlined.
```

This word is underlined.

Figure 13. Thẻ `<u>` dùng để gạch chân văn bản.

Theo như đặc tính của HTML 5, ta chỉ nên sử dụng phần tử `<u>` khi không có cách nào tốt hơn để gạch chân văn bản. CSS có cung cấp một giải pháp thay thế hiện đại hơn.

Phông chữ Cố định Chiều rộng hoặc Đơn cách

Để hiển thị văn bản ở phông chữ đơn cách (có độ rộng cố định) thường được sử dụng để hiển thị mã máy tính, chúng ta sẽ sử dụng phần tử `<tt>` như được minh họa trong ví dụ sau. Kết quả sẽ hiển thị trong Figure 14.

```
This <tt>word</tt> is in fixed-width font.
```

This `word` is in fixed-width font.

Figure 14. Thẻ `<tt>` được sử dụng để hiển thị văn bản ở phông chữ có độ rộng cố định.

Thẻ `<tt>` không được hỗ trợ trong HTML 5. Tuy các trình duyệt web vẫn sẽ hiển thị nó nhưng ta vẫn nên sử dụng các thẻ thích hợp hơn là `<code>`, `<kbd>`, `<var>` và `<samp>`.

Phần tử Cụm từ

Các phần tử cụm từ không chỉ thay đổi hình thức của văn bản mà còn bổ sung tầm quan trọng về mặt ngữ nghĩa cho một từ hoặc một cụm từ. Bằng cách sử dụng chúng, ta có thể nhấn mạnh một từ hoặc đánh dấu nó là quan trọng. Trái ngược với các phần tử trình bày, các phần tử này được trình đọc màn hình nhận diện và từ đó giúp khách truy cập khiếm thị dễ tiếp cận với văn bản hơn cũng như cho phép các công cụ tìm kiếm đọc và đánh giá nội dung trang tốt hơn. Các phần tử cụm từ chúng ta sẽ sử dụng trong suốt bài học này là `<em>`, `<strong>` và `<code>`.

Văn bản được Nhấn mạnh

Để nhấn mạnh văn bản, chúng ta có thể gói văn bản đó trong thành phần `<em>` như được minh họa trong ví dụ sau:

```
This <em>word</em> is emphasized.
```

Figure 15. Thẻ `<em>` *dùng để nhấn mạnh văn bản.*

Như có thể thấy, các trình duyệt web đã hiển thị `<em>` theo một cách tương tự như `<i>`, nhưng `<em>` sẽ bổ sung tầm quan trọng về mặt ngữ nghĩa với tư cách là một phần cụm từ, giúp cải thiện khả năng theo dõi cho khách truy cập khiếm thị.

Văn bản Quan trọng

Để đánh dấu văn bản là quan trọng, chúng ta sẽ gói văn bản đó trong phần tử `<strong>` như được minh họa trong ví dụ sau. Kết quả sẽ được hiển thị trong Figure 16.

```
This <strong>word</strong> is important.
```

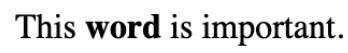

Figure 16. Thẻ `<strong>` *được dùng để đánh dấu văn bản quan trọng.*

Như có thể thấy, các trình duyệt web sẽ hiển thị `<strong>` theo một cách tương tự như `<b>`, nhưng `<strong>` bổ sung tầm quan trọng về mặt ngữ nghĩa với tư cách là một phần tử cụm từ, giúp cải thiện khả năng theo dõi cho khách truy cập khiếm thị.

Mã Máy tính

Để chèn một đoạn mã máy tính, chúng ta sẽ gói đoạn mã đó trong phần tử `<code>` như được minh họa trong ví dụ sau. Kết quả sẽ được hiển thị trong Figure 17.

```
The Markdown code <code># Heading</code> creates a heading at the highest level in the
hierarchy.
```

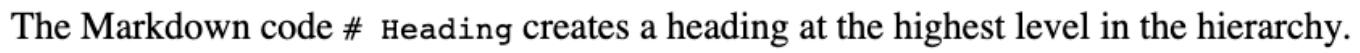

Figure 17. Thẻ `<code>` *dùng để chèn một đoạn mã máy tính.*

Văn bản được đánh dấu

Để đánh dấu văn bản với nền màu vàng (ương tự như kiểu đánh dấu tô sáng khi dùng bút dạ quang), chúng ta sẽ sử dụng phần tử `<mark>` như được minh họa trong ví dụ sau. Kết quả sẽ được hiển thị trong Figure 18.

```
This <mark>word</mark> is highlighted.
```

This word is highlighted.

Figure 18. Thẻ `<mark>` dùng để đánh dấu văn bản với nền màu vàng.

Định dạng Văn bản của Danh sách Mua sắm HTML

Dựa trên các ví dụ của chúng ta, hãy cùng chèn một số phần tử cụm từ để thay đổi hình thức của văn bản, đồng thời tăng thêm tầm quan trọng về mặt ngữ nghĩa của chúng. Kết quả sẽ được hiển thị trong Figure 19.

```
<h1>Garden Party</h1>
<p>
  Invitation to <strong>John's garden party</strong>.<br>
  <strong>Saturday, next week.</strong>
</p>

<h2>Agenda</h2>
<ol>
  <li>Welcome</li>
  <li>
    Barbecue
    <ul>
      <li><em>Vegetables</em></li>
      <li><em>Meat</em></li>
      <li><em>Burgers</em>, including vegetarian options</li>
    </ul>
  </li>
  <li>Dessert</li>
  <li><mark>Fireworks</mark></li>
</ol>

<hr>

<h2>Please bring</h2>
<ul>
  <li>Salad</li>
  <li>Drinks</li>
  <li>Bread</li>
  <li>Snacks</li>
  <li>Desserts</li>
```

```
</ul>
```

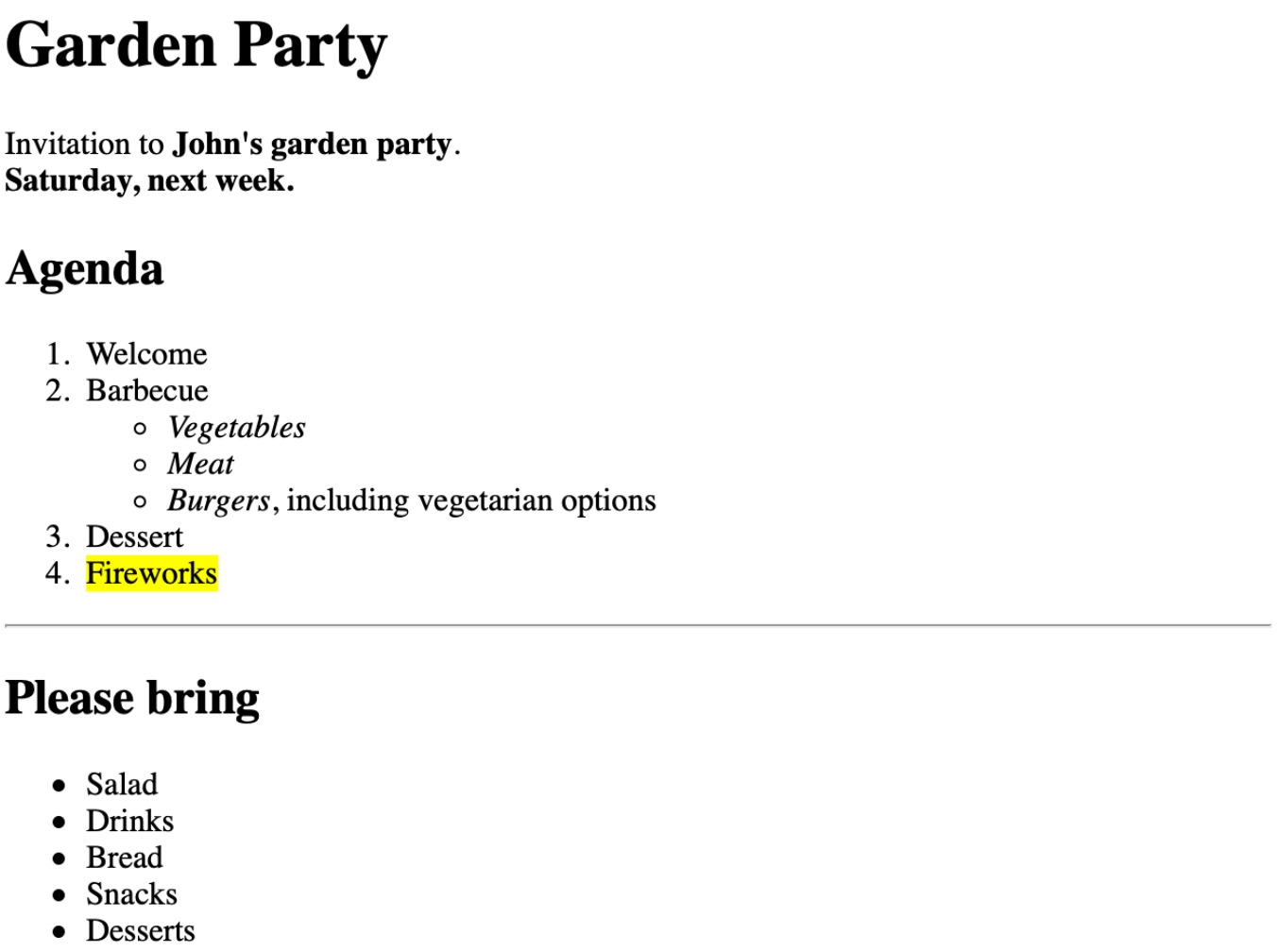

Figure 19. Trang HTML với một số yếu tố định dạng.

Trong tài liệu HTML mẫu này, thông tin quan trọng nhất liên quan đến bữa tiệc trong vườn được đánh dấu là quan trọng bằng cách sử dụng phần tử `<strong>`. Các loại thực phẩm có sẵn cho tiệc nướng được nhấn mạnh bằng phần tử `<em>`. Pháo hoa được làm nổi bật bằng cách sử dụng phần tử `<mark>`.

Như một bài tập thực hành, bạn cũng có thể thử định dạng các đoạn văn bản khác bằng cách sử dụng các phần tử định dạng khác.

Văn bản định dạng sẵn

Trong hầu hết các phần tử HTML, khoảng trắng thường được giảm xuống thành một khoảng cách

bình thường hoặc thậm chí bị bỏ qua hoàn toàn. Tuy nhiên, có một phần tử HTML có tên `<pre>` sẽ cho phép ta xác định cái được gọi là văn bản *định dạng sẵn*. Khoảng trắng trong nội dung của phần tử này (bao gồm các khoảng cách và khoảng ngắt dòng) sẽ được giữ nguyên và hiển thị trong trình duyệt web. Ngoài ra, văn bản sẽ được hiển thị bằng phông chữ có chiều rộng cố định, tương tự như phần tử `<code>`.

```
<pre>
field() {
  shift $1 ; echo $1
}
</pre>
```

```
field() {
  shift $1 ; echo $1
}
```

Figure 20. Phần trình bày trên trình duyệt web của mã HTML minh họa cách phần tử HTML `<pre>` *duy trì khoảng trắng.*

Nhóm các Phần tử

Theo quy ước, các phần tử HTML sẽ được chia thành hai loại:

Phần tử Cấp độ Khối

Chúng xuất hiện trên một dòng mới và chiếm toàn bộ chiều rộng có sẵn. Ví dụ về các phần tử cấp độ khối mà chúng ta đã thảo luận là `<p>`, `<ol>` và `<h2>`.

Các Phần tử Cấp độ Nội tuyến

Chúng xuất hiện trên cùng một dòng với các phần tử và văn bản khác và chỉ chiếm phần không gian cần thiết đủ cho nội dung của chúng. Ví dụ về các phần tử cấp độ nội tuyến là `<strong>`, `<em>` và `<i>`.

NOTE HTML5 đã giới thiệu các danh mục phần tử chính xác hơn để tránh khỏi nhầm lẫn với khối CSS và hộp nội tuyến. Để đơn giản hơn, ở đây chúng ta sẽ tập trung vào việc chia nhỏ thành các phần tử khối và nội tuyến truyền thống.

Các phần tử cơ bản để nhóm nhiều phần tử lại với nhau là các phần tử `<div>` và `<span>`.

Phần tử `<div>` là vùng chứa cấp khối của các phần tử HTML khác và sẽ không tự thêm giá trị ngữ nghĩa. Chúng ta có thể sử dụng phần tử này để chia tài liệu HTML thành các phần và cấu trúc nội

dung để tăng khả năng theo dõi mã cũng như áp dụng các kiểu CSS cho một nhóm phần tử ta sẽ trong bài học sau.

Theo mặc định, trình duyệt web sẽ luôn chèn dấu ngắt dòng trước và sau mỗi phần tử `<div>` để mỗi phần tử đều sẽ được hiển thị trên một dòng riêng.

Ngược lại, phần tử `<span>` được sử dụng làm vùng chứa văn bản HTML và thường được sử dụng để nhóm các phần tử nội tuyến khác nhằm áp dụng các kiểu CSS cho một phần văn bản nhỏ hơn.

Phần tử `<span>` hoạt động giống như văn bản thông thường và sẽ không bắt đầu trên một dòng mới. Do đó, nó là một phần tử nội tuyến.

Ví dụ sau đây sẽ so sánh biểu diễn trực quan của phần tử ngữ nghĩa `<p>` và các phần tử nhóm `<div>` và `<span>`:

```
<p>Text within a paragraph</p>
<p>Another paragraph of text</p>
<hr>
<div>Text wrapped within a <code>div</code> element</div>
<div>Another <code>div</code> element with more text</div>
<hr>
<span>Span content</span>
<span>and more span content</span>
```

Trình duyệt web sẽ hiển thị mã này như trong Figure 21.

Figure 21. Phần trình bày trên trình duyệt web của một tài liệu thử nghiệm cho thấy sự khác biệt giữa các phần tử span, div và paragraph trong HTML.

Chúng ta đã thấy rằng theo mặc định, trình duyệt web sẽ thêm khoảng cách trước và sau các phần tử `<p>`. Khoảng cách này không được áp dụng cho cả hai phần tử nhóm `<div>` và `<span>`. Tuy nhiên, các phần tử `<div>` sẽ được định dạng dưới dạng các khối riêng của chúng, trong khi văn bản trong các phần tử `<span>` sẽ được hiển thị trên cùng một dòng.

Cấu trúc Trang HTML

Chúng ta đã thảo luận về cách sử dụng các phần tử HTML để mô tả nội dung của trang web theo ngữ nghĩa – nói cách khác - để truyền đạt ý nghĩa và ngữ cảnh cho văn bản. Một nhóm phần tử khác được thiết kế với mục đích mô tả *cấu trúc ngữ nghĩa* của một trang web, một biểu thức hoặc cấu trúc của nó. Các phần tử này là phần tử khối, tức là . hành vi trực quan của chúng cũng tương tự như phần tử `<div>`. Mục đích của chúng là xác định cấu trúc ngữ nghĩa của trang web bằng cách chỉ định các khu vực được xác định rõ như tiêu đề, chân trang và nội dung chính của trang. Các phần tử này cho phép ta nhóm nội dung theo ngữ nghĩa để máy tính cũng có thể hiểu được nội dung đó, bao gồm cả công cụ tìm kiếm và trình đọc màn hình.

Phần tử `<header>`

Phần tử `<header>` (đầu trang) sẽ chứa các thông tin giới thiệu về phần tử ngữ nghĩa xung quanh trong tài liệu HTML. Một đầu trang khác với một đề mục, nhưng một đầu trang sẽ thường bao gồm một thành phần đề mục (`<h1>`, ... , `<h6>`).

Trong thực tế, phần tử này thường được sử dụng nhiều nhất để biểu diễn phần đầu trang, chẳng hạn như biểu ngữ có logo. Nó cũng có thể được sử dụng để giới thiệu nội dung cho bất kỳ phần tử nào sau đây: `<body>`, `<section>`, `<article>`, `<nav>` hoặc `<aside>`.

Một tài liệu có thể có nhiều phần tử `<header>`, nhưng một phần tử `<header>` lại không thể được lồng trong một phần tử `<header>` khác. Phần tử `<footer>` cũng không thể được sử dụng lồng trong `<header>`. Ví dụ: để thêm một đầu trang vào tài liệu mẫu, ta có thể thực hiện như sau:

```
<header>
  <h1>Garden Party</h1>
</header>
```

Sẽ không có một thay đổi rõ ràng nào đối với tài liệu HTML, vì `<h1>` (giống như tất cả các phần tử đề mục khác) là một phần tử cấp độ khối không có thuộc tính trực quan nào khác.

Phần tử Nội dung `<main>`

Phần tử `<main>` (phần chính) là vùng chứa nội dung trung tâm của trang web. Không được có nhiều hơn một phần tử `<main>` trong tài liệu HTML.

Trong tài liệu ví dụ của chúng ta, tất cả các mã HTML mà chúng ta đã viết từ trước cho tới nay sẽ được đặt bên trong phần tử `<main>`.

```
<main>
  <header>
    <h1>Garden Party</h1>
  </header>
  <p>
    Invitation to <strong>John's garden party</strong>.<br>
    <strong>Saturday, next week.</strong>
  </p>

  <h2>Agenda</h2>
  <ol>
    <li>Welcome</li>
    <li>
      Barbecue
      <ul>
        <li><em>Vegetables</em></li>
        <li><em>Meat</em></li>
        <li><em>Burgers</em>, including vegetarian options</li>
      </ul>
    </li>
    <li>Dessert</li>
    <li><mark>Fireworks</mark></li>
  </ol>

  <hr>

  <h2>Please bring</h2>
  <ul>
    <li>Salad</li>
    <li>Drinks</li>
    <li>Bread</li>
    <li>Snacks</li>
    <li>Desserts</li>
  </ul>
</main>
```

Giống như phần tử `<header>`, phần tử `<main>` sẽ không gây ra bất kỳ thay đổi trực quan nào trong ví dụ của chúng ta.

Phần tử `<footer>`

Phần tử `<footer>` (chân trang) sẽ chứa các chú thích (ví dụ như thông tin tác giả, thông tin liên hệ hoặc tài liệu liên quan) được đặt ở cuối trang cho các phần tử ngữ nghĩa xung quanh nó, ví dụ như

`<section>`, `<nav>` hoặc `<aside>`. Một tài liệu có thể có nhiều phần tử `<footer>` cho phép bạn mô tả chính xác hơn các phần tử ngữ nghĩa. Tuy nhiên, phần tử `<footer>` không thể được lồng trong một phần tử `<footer>` khác, cũng như không thể sử dụng phần tử `<header>` ở bên trong `<footer>`.

Trong ví dụ của chúng ta, ta có thể thêm thông tin liên hệ về chủ nhà (John) như trong ví dụ dưới đây:

```
<footer>
  <p>John Doe</p>
  <p>john.doe@example.com</p>
</footer>
```

Phần tử `<nav>`

Phần tử `<nav>` (điều hướng) mô tả một đơn vị điều hướng chính (chẳng hạn như menu) có chứa một loạt các siêu liên kết.

NOTE Không phải siêu liên kết nào cũng đều phải được gói trong một phần tử `<nav>`. Nó chỉ hữu ích trong việc liệt kê một nhóm các liên kết.

Bởi vì các siêu liên kết vẫn chưa được nhắc đến nên các phần tử điều hướng sẽ không có trong ví dụ của bài học này.

Phần tử `<aside>`

Phần tử `<aside>` (phần bên cạnh) là nơi chứa những nội dung không cần thiết theo thứ tự của nội dung trang chính, nhưng lại thường sẽ có liên quan một cách gián tiếp hoặc bổ sung. Phần tử này thường được sử dụng cho các thanh/ cột bên để hiển thị thông tin thứ cấp, chẳng hạn như bảng thuật ngữ.

Trong ví dụ của chúng ta, ta có thể thêm thông tin về địa chỉ và hành trình (tức những thông tin chỉ liên quan gián tiếp đến phần nội dung còn lại) bằng cách sử dụng phần tử `<aside>`.

```
<aside>
  <p>
    10, Main Street<br>
    Newville
  </p>
  <p>Parking spaces available.</p>
</aside>
```

Phần tử `<section>`

Phần tử `<section>` (phần) xác định một phần logic trong tài liệu - tức một phần của phần tử ngữ nghĩa xung quanh nhưng sẽ không hoạt động như một nội dung độc lập (chẳng hạn như một chương).

Trong tài liệu mẫu, chúng ta ta có thể gói các phần nội dung cho nội dung chương trình và đưa vào các phần danh sách như trong ví dụ sau:

```
<section>
  <header>
    <h2>Agenda</h2>
  </header>
  <ol>
    <li>Welcome</li>
    <li>
      Barbecue
      <ul>
        <li><em>Vegetables</em></li>
        <li><em>Meat</em></li>
        <li><em>Burgers</em>, including vegetarian options</li>
      </ul>
    </li>
    <li>Dessert</li>
    <li><mark>Fireworks</mark></li>
  </ol>
</section>

<hr>

<section>
  <header>
    <h2>Please bring</h2>
  </header>
  <ul>
    <li>Salad</li>
    <li>Drinks</li>
    <li>Bread</li>
    <li>Snacks</li>
    <li>Desserts</li>
  </ul>
</section>
```

Ví dụ này cũng đã bổ sung thêm các phần tử `<header>` trong các phần, sao cho mỗi phần đều nằm trong phần tử `<header>` của chính nó.

Phần tử `<article>`

Phần tử `<article>` (bài viết) xác định nội dung độc lập có ý nghĩa riêng mà không cần các phần còn lại của trang. Nội dung của nó có khả năng phân phối lại hoặc tái sử dụng trong những ngữ cảnh khác. Một số ví dụ điển hình cho phần tử `<article>` là một bài đăng trên blog, danh sách sản phẩm cho một cửa hàng và một quảng cáo sản phẩm. Sau đó, quảng cáo có thể tự tồn tại một mình hoặc ở trong một trang lớn hơn.

Trong ví dụ của chúng ta, ta có thể thay thế phần tử `<section>` đầu tiên bao quanh nội dung chương trình bằng một phần tử `<article>`.

```
<article>
  <header>
    <h2>Agenda</h2>
  </header>
  <ol>
    <li>Welcome</li>
    <li>
      Barbecue
      <ul>
        <li><em>Vegetables</em></li>
        <li><em>Meat</em></li>
        <li><em>Burgers</em>, including vegetarian options</li>
      </ul>
    </li>
    <li>Dessert</li>
    <li><mark>Fireworks</mark></li>
  </ol>
</article>
```

Phần tử `<header>` mà chúng ta đã thêm trong ví dụ trước cũng có thể được dùng ở đây vì các phần tử `<article>` có thể có các phần tử `<header>` của riêng chúng.

Ví dụ cuối cùng

Kết hợp tất cả các ví dụ trước, tài liệu HTML cuối cùng của lời mời sẽ trông như sau:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <title>Garden Party</title>
</head>

<body>
  <main>
    <h1>Garden Party</h1>
    <p>
      Invitation to <strong>John's garden party</strong>.<br>
      <strong>Saturday, next week.</strong>
    </p>

    <article>
      <h2>Agenda</h2>
      <ol>
        <li>Welcome</li>
        <li>
          Barbecue
          <ul>
            <li><em>Vegetables</em></li>
            <li><em>Meat</em></li>
            <li><em>Burgers</em>, including vegetarian options</li>
          </ul>
        </li>
        <li>Dessert</li>
        <li><mark>Fireworks</mark></li>
      </ol>
    </article>

    <hr>

    <section>
      <h2>Please bring</h2>
      <ul>
        <li>Salad</li>
        <li>Drinks</li>
        <li>Bread</li>
        <li>Snacks</li>
        <li>Desserts</li>
      </ul>
    </section>
  </main>

  <aside>
```

```
      <p>
        10, Main Street<br>
        Newville
      </p>
      <p>Parking spaces available.</p>
    </aside>

    <footer>
      <p>John Doe</p>
      <p>john.doe@example.com</p>
    </footer>
  </body>
</html>
```

Trong trình duyệt web, toàn bộ trang sẽ được hiển thị như trong Figure 22.

Garden Party

Invitation to **John's garden party**.
Saturday, next week.

Agenda

1. Welcome
2. Barbecue
    - *Vegetables*
    - *Meat*
    - *Burgers*, including vegetarian options
3. Dessert
4. Fireworks

Please bring

- Salad
- Drinks
- Bread
- Snacks
- Desserts

10, Main Street
Newville

Parking spaces available.

John Doe

john.doe@example.com

Figure 22. Phần trình bày trên trình duyệt web của tài liệu HTML cuối cùng kết hợp tất cả các ví dụ trước đó. Trang đại diện cho lời mời dự tiệc ngoài vườn và mô tả nội dung chương trình cho buổi tối cũng như danh sách đồ ăn các vị khách cần mang theo.

Bài tập Hướng dẫn

1. Đối với mỗi thẻ sau, hãy chỉ ra thẻ đóng tương ứng:

`<h5>`	
` `	
`<ol>`	
`<dd>`	
`<hr>`	
`<strong>`	
`<tt>`	
`<main>`	

2. Đối với mỗi thẻ sau, hãy cho biết liệu nó đánh dấu phần đầu của một phần tử khối hay một phần tử nội tuyến:

`<h3>`	
`<span>`	
`<b>`	
`<div>`	
`<em>`	
`<dl>`	
`<li>`	
`<nav>`	
`<code>`	
`<pre>`	

3. Có những loại danh sách nào mà bạn có thể tạo trong HTML? Nên sử dụng thẻ nào cho mỗi loại danh sách đó?

4. Những thẻ nào có thể gói các phần tử khối mà bạn có thể sử dụng để cấu trúc một trang HTML?

Bài tập Mở rộng

1. Hãy tạo một trang HTML cơ bản với tiêu đề “Form Rules”. Bạn sẽ sử dụng trang HTML này cho tất cả các bài tập mở rộng, mỗi bài đều sẽ dựa trên các bài trước. Sau đó, hãy thêm tiêu đề cấp 1 có nội dung “To receive the PDF document with the complete HTML course, it is necessary to fill in the following fields:” và một danh sách không có thứ tự với các danh mục sau: “Name”, “Surname”, “Email Address”, “Nation”, “Country” và “Zip/Postal Code”.

2. Hãy đặt ba trường đầu tiên (“Name”, “Surname”, and “Email Address”) bằng chữ in đậm, đồng thời tăng thêm tầm quan trọng về mặt ngữ nghĩa. Sau đó, hãy thêm tiêu đề cấp 2 có nội dung “Required fields” và một đoạn có nội dung “Bold fields are mandatory.”.

3. Hãy thêm một tiêu đề cấp 2 khác có nội dung “Steps to follow”, một đoạn văn bản có nội dung “There are four steps to follow:” và một danh sách có thứ tự với các danh mục sau: “Fill in the fields”, “Click the Submit button”, “Check your e-mail and confirm your request by clicking on the link you receive” và “Check your e-mail - You will receive the full HTML course in minutes”.

4. Hãy sử dụng `<div>` và tạo một khối cho mỗi một phần bắt đầu bằng tiêu đề cấp 2.

5. Hãy sử dụng `<div>` và tạo một khối khác cho phần bắt đầu bằng tiêu đề cấp 1. Sau đó, hãy tách phần này với hai phần còn lại bằng một đường ngang.

6. Hãy thêm phần tử đầu trang có nội dung “Form Rules - 2021” và phần tử chân trang có nội dung “Copyright Note - 2021”. Cuối cùng, hãy thêm phần tử chính có chứa ba khối `<div>`.

Tóm tắt

Trong bài học này, bạn đã học về:

- Cách tạo đánh dấu cho nội dung trong tài liệu HTML
- Cấu trúc văn bản HTML phân cấp
- Sự khác biệt giữa các phần tử HTML khối và nội tuyến
- Cách tạo tài liệu HTML có cấu trúc ngữ nghĩa

Các thuật ngữ sau đây đã được nhắc tới trong bài học này:

`<h1>`, **`<h2>`**, **`<h3>`**, **`<h4>`**, **`<h5>`**, **`<h6>`**
: Các thẻ Đề mục.

`<p>`
: Thẻ đoạn văn bản.

`<ol>`
: Thẻ danh sách có thứ tự.

`<ul>`
: Thẻ danh sách không có thứ tự.

`<li>`
: Thẻ danh mục.

`<dl>`
: Thẻ danh sách mô tả.

`<dt>`, **`<dd>`**
: Các thẻ của mỗi thuật ngữ và phần mô tả cho một danh sách mô tả.

`<pre>`
: Thẻ định dạng bảo toàn.

`<b>`, **`<i>`**, **`<u>`**, **`<tt>`**, **`<em>`**, **`<strong>`**, **`<code>`**, **`<mark>`**
: Các thẻ định dạng.

`<div>`, **`<span>`**
: Các thẻ nhóm.

`<header>`, `<main>`, `<nav>`, `<aside>`, `<footer>`

Các thẻ được sử dụng để cung cấp cấu trúc và bố cục đơn giản cho trang HTML.

Đáp án Bài tập Hướng dẫn

1. Đối với mỗi thẻ sau, hãy chỉ ra thẻ đóng tương ứng:

`<h5>`	`</h5>`
` `	Does not exist
`<ol>`	`</ol>`
`<dd>`	`</dd>`
`<hr>`	Does not exist
`<strong>`	`</strong>`
`<tt>`	`</tt>`
`<main>`	`</main>`

2. Đối với mỗi thẻ sau, hãy cho biết liệu nó đánh dấu phần đầu của một phần tử khối hay một phần tử nội tuyến:

`<h3>`	Phần tử Khối
`<span>`	Phần tử Ngoại tuyến
`<b>`	Phần tử Ngoại tuyến
`<div>`	Phần tử Khối
`<em>`	Phần tử Ngoại tuyến
`<dl>`	Phần tử Khối
`<li>`	Phần tử Khối
`<nav>`	Phần tử Khối
`<code>`	Phần tử Ngoại tuyến
`<pre>`	Phần tử Khối

3. Có những loại danh sách nào mà bạn có thể tạo trong HTML? Nên sử dụng thẻ nào cho mỗi loại danh sách đó?

   Trong HTML, bạn có thể tạo ba loại danh sách: danh sách có thứ tự bao gồm một loạt các danh mục được đánh số, danh sách không có thứ tự bao gồm một loạt các danh mục không có thứ tự hoặc trình tự đặc biệt, và cuối cùng là danh sách mô tả đại diện cho các mục như trong từ điển hoặc bách khoa toàn thư . Một danh sách có thứ tự sẽ được đặt giữa các thẻ `<ol>` và `</ol>`, một

danh sách không có thứ tự sẽ được đặt giữa các thẻ `<ul>` và `</ul>` và một danh sách mô tả sẽ được đặt giữa các thẻ `<dl>` và `</dl>`. Mỗi một mục trong danh sách có thứ tự hoặc không có thứ tự sẽ được đặt giữa các thẻ `<li>` và `</li>`, trong khi mỗi thuật ngữ trong danh sách mô tả sẽ được đặt giữa các thẻ `<dt>` và `</dt>` và phần mô tả của nó được đặt giữa các thẻ `<dd>` và `</dd>`.

4. Những thẻ nào có thể gói các thành phần khối mà chúng ta sẽ sử dụng để cấu trúc một trang HTML?

   Các thẻ `<header>` và `</header>` gói phần đầu trang, các thẻ `<main>` và `</main>` gói nội dung chính của trang HTML, các thẻ `<nav>` và `</nav>` gói phần điều hướng, các thẻ `<aside>` và `</aside>` gói các thanh bên và các thẻ `<footer>` và `</footer>` gói phần chân trang.

Đáp án Bài tập Mở rộng

1. Hãy tạo một trang HTML cơ bản với tiêu đề “Form Rules”. Bạn sẽ sử dụng trang HTML này cho tất cả các bài tập mở rộng, mỗi bài đều sẽ dựa trên các bài trước. Sau đó, hãy thêm tiêu đề cấp 1 có nội dung “To receive the PDF document with the complete HTML course, it is necessary to fill in the following fields:” và một danh sách không có thứ tự với các danh mục sau: “Name”, “Surname”, “Email Address”, “Nation”, “Country” và “Zip/Postal Code”.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Rules</title>
  </head>

  <body>
    <h1>How to fill in the request form</h1>
    <p>
      To receive the PDF document with the complete HTML course, it is necessary
      to fill in the following fields:
    </p>
    <ul>
      <li>Name</li>
      <li>Surname</li>
      <li>Email Address</li>
      <li>Nation</li>
      <li>Country</li>
      <li>Zip/Postal Code</li>
    </ul>
  </body>
</html>
```

2. Hãy đặt ba trường đầu tiên (“Name”, “Surname”, and “Email Address”) bằng chữ in đậm, đồng thời tăng thêm tầm quan trọng về mặt ngữ nghĩa. Sau đó, hãy thêm tiêu đề cấp 2 có nội dung “Required fields” và một đoạn có nội dung “Bold fields are mandatory.”.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Rules</title>
  </head>

  <body>
```

```
    <h1>How to fill in the request form</h1>
    <p>
      To receive the PDF document with the complete HTML course, it is necessary
      to fill in the following fields:
    </p>
    <ul>
      <li><strong> Name </strong></li>
      <li><strong> Surname </strong></li>
      <li><strong> Email Address </strong></li>
      <li>Nation</li>
      <li>Country</li>
      <li>Zip/Postal Code</li>
    </ul>

    <h2>Required fields</h2>
    <p>Bold fields are mandatory.</p>
  </body>
</html>
```

3. Hãy thêm một tiêu đề cấp 2 khác có nội dung “Steps to follow”, một đoạn văn bản có nội dung “There are four steps to follow:” và một danh sách có thứ tự với các danh mục sau: “Fill in the fields”, “Click the Submit button”, “Check your e-mail and confirm your request by clicking on the link you receive” và “Check your e-mail - You will receive the full HTML course in minutes”.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Rules</title>
  </head>

  <body>
    <h1>How to fill in the request form</h1>
    <p>
      To receive the PDF document with the complete HTML course, it is necessary
      to fill in the following fields:
    </p>
    <ul>
      <li><strong> Name </strong></li>
      <li><strong> Surname </strong></li>
      <li><strong> Email Address </strong></li>
      <li>Nation</li>
      <li>Country</li>
      <li>Zip/Postal Code</li>
```

```
    </ul>

    <h2>Required fields</h2>
    <p>Bold fields are mandatory.</p>

    <h2>Steps to follow</h2>
    <p>There are four steps to follow:</p>
    <ol>
      <li>Fill in the fields</li>
      <li>Click the Submit button</li>
      <li>
        Check your e-mail and confirm your request by clicking on the link you
        receive
      </li>
      <li>
        Check your e-mail - You will receive the full HTML course in minutes
      </li>
    </ol>
  </body>
</html>
```

4. Hãy sử dụng `<div>` và tạo một khối cho mỗi một phần bắt đầu bằng tiêu đề cấp 2.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Rules</title>
  </head>

  <body>
    <h1>How to fill in the request form</h1>
    <p>
      To receive the PDF document with the complete HTML course, it is necessary
      to fill in the following fields:
    </p>
    <ul>
      <li><strong> Name </strong></li>
      <li><strong> Surname </strong></li>
      <li><strong> Email Address </strong></li>
      <li>Nation</li>
      <li>Country</li>
      <li>Zip/Postal Code</li>
    </ul>
```

```
    <div>
      <h2>Required fields</h2>
      <p>Bold fields are mandatory.</p>
    </div>

    <div>
      <h2>Steps to follow</h2>
      <p>There are four steps to follow:</p>
      <ol>
        <li>Fill in the fields</li>
        <li>Click the Submit button</li>
        <li>
          Check your e-mail and confirm your request by clicking on the link you
          receive
        </li>
        <li>
          Check your e-mail - You will receive the full HTML course in minutes
        </li>
      </ol>
    </div>
  </body>
</html>
```

5. Hãy sử dụng `<div>` và tạo một khối cho mỗi một phần bắt đầu bằng tiêu đề cấp 2.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Rules</title>
  </head>

  <body>
    <div>
      <h1>How to fill in the request form</h1>
      <p>
        To receive the PDF document with the complete HTML course, it is
        necessary to fill in the following fields:
      </p>
      <ul>
        <li><strong> Name </strong></li>
        <li><strong> Surname </strong></li>
        <li><strong> Email Address </strong></li>
```

```
        <li>Nation</li>
        <li>Country</li>
        <li>Zip/Postal Code</li>
      </ul>
    </div>

    <hr>

    <div>
      <h2>Required fields</h2>
      <p>Bold fields are mandatory.</p>
    </div>

    <div>
      <h2>Steps to follow</h2>
      <p>There are four steps to follow:</p>
      <ol>
        <li>Fill in the fields</li>
        <li>Click the Submit button</li>
        <li>
          Check your e-mail and confirm your request by clicking on the link you
          receive
        </li>
        <li>
          Check your e-mail - You will receive the full HTML course in minutes
        </li>
      </ol>
    </div>
  </body>
</html>
```

6. Hãy thêm phần tử đầu trang có nội dung “Form Rules - 2021” và phần tử chân trang có nội dung “Copyright Note - 2021”. Cuối cùng, hãy thêm phần tử chính có chứa ba khối `<div>`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Rules</title>
  </head>

  <body>
    <header>
      <h1>Form Rules - 2021</h1>
```

```
  </header>

  <main>
    <div>
      <h1>How to fill in the request form</h1>
      <p>
        To receive the PDF document with the complete HTML course, it is
        necessary to fill in the following fields:
      </p>
      <ul>
        <li><strong> Name </strong></li>
        <li><strong> Surname </strong></li>
        <li><strong> Email Address </strong></li>
        <li>Nation</li>
        <li>Country</li>
        <li>Zip/Postal Code</li>
      </ul>
    </div>

    <hr>

    <div>
      <h2>Required fields</h2>
      <p>Bold fields are mandatory.</p>
    </div>

    <div>
      <h2>Steps to follow</h2>
      <p>There are four steps to follow:</p>
      <ol>
        <li>Fill in the fields</li>
        <li>Click the Submit button</li>
        <li>
          Check your e-mail and confirm your request by clicking on the link
          you receive
        </li>
        <li>
          Check your e-mail - You will receive the full HTML course in minutes
        </li>
      </ol>
    </div>
  </main>

  <footer>
```

```
      <p>Copyright Note - 2021</p>
    </footer>
  </body>
</html>
```

032.3 Tham chiếu HTML và Tài nguyên Nhúng

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 032.3

Khối lượng

2

Các lĩnh vực kiến thức chính

- Tạo liên kết đến các tài nguyên bên ngoài và neo trang
- Thêm hình ảnh vào tài liệu HTML
- Hiểu về các thuộc tính chính của các định dạng tệp phương tiện phổ biến như PNG, JPG và SVG
- Biết về iframe

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Thuộc tính `id`
- `<a>`, bao gồm các thuộc tính `href` và `target` (`_blank`, `_self`, `_parent`, `_top`)
- `<img>`, bao gồm các thuộc tính `src` và `alt`

032.3 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	032 Đánh dấu Tài liệu HTML
Mục tiêu:	032.3 Tham chiếu HTML và Tài nguyên Nhúng
Bài:	1 trên 1

Giới thiệu

Rất hiếm có một trang web hiện đại nào mà chỉ bao gồm văn bản. Nó thường sẽ bao gồm nhiều loại nội dung khác, chẳng hạn như hình ảnh, âm thanh, video và thậm chí là cả các tài liệu HTML khác nữa. Cùng với nội dung bên ngoài, các tài liệu HTML có thể chứa các liên kết đến các tài liệu khác, điều này làm cho trải nghiệm duyệt Internet trở nên đơn giản hơn rất nhiều.

Nội dung Nhúng

Người ta có thể trao đổi tệp qua Internet mà không cần các trang web được viết bằng HTML, vậy tại sao HTML lại là định dạng được chọn cho các tài liệu web chứ không phải là PDF hay bất kỳ định dạng xử lý văn bản nào khác? Một lý do quan trọng là HTML có khả năng giữ tài nguyên đa phương tiện của nó trong các tệp riêng biệt. Trong một môi trường như Internet nơi thông tin thường xuyên trùng lặp và được phân tán ở nhiều vị trí khác nhau, điều quan trọng là phải tránh việc truyền những dữ liệu không cần thiết. Trong hầu hết mọi trường hợp, các phiên bản mới của trang web sẽ sử dụng cùng một số các hình ảnh và tệp hỗ trợ khác giống như các phiên bản trước đó; vì vậy, trình duyệt web có thể sử dụng các tệp đã có trước đó thay vì sao chép lại mọi thứ. Hơn nữa, việc giữ các tệp riêng biệt sẽ tạo điều kiện thuận lợi cho việc tùy chỉnh nội dung đa phương tiện theo các đặc điểm của máy khách, chẳng hạn như vị trí, kích thước màn hình và tốc độ kết nối

của chúng.

Hình ảnh

Loại nội dung nhúng phổ biến nhất là hình ảnh đi kèm với văn bản. Hình ảnh sẽ được giữ riêng và được tham chiếu bên trong tệp HTML bằng thẻ `<img>` (hình ảnh):

```
<img src="logo.png">
```

Thẻ `<img>` không cần có thẻ đóng. Thuộc tính `src` cho biết vị trí nguồn của tệp hình ảnh. Trong ví dụ này, tệp hình ảnh `logo.png` phải được đặt trong cùng thư mục với tệp HTML; nếu không, trình duyệt sẽ không thể hiển thị nó. Thuộc tính vị trí nguồn chấp nhận các đường dẫn tương đối; do đó, ký hiệu *dấu chấm* có thể được sử dụng để chỉ ra đường dẫn đến hình ảnh:

```
<img src="../logo.png">
```

Hai dấu chấm cho biết hình ảnh nằm ở bên trong thư mục mẹ tương liên quan tới thư mục chứa tệp HTML. Nếu tên tệp `../logo.png` được sử dụng bên trong tệp HTML có URL là `http://example.com/library/periodicals/index.html` thì trình duyệt sẽ yêu cầu tệp hình ảnh tại địa chỉ `http://example.com/library/logo.png`.

Ký hiệu dấu chấm cũng sẽ được áp dụng nếu tệp HTML không phải là tệp thực trong hệ thống tệp; trình duyệt HTML diễn giải URL như một đường dẫn đến tệp, nhưng công việc của máy chủ HTTP là quyết định xem đường dẫn đó đề cập đến tệp hay nội dung được tạo động. Miền và đường dẫn thích hợp sẽ tự động được thêm vào tất cả các yêu cầu tới máy chủ trong trường hợp tệp HTML đến từ một yêu cầu HTTP. Tương tự như vậy, trình duyệt sẽ mở đúng hình ảnh nếu tệp HTML được mở trực tiếp từ hệ thống tệp cục bộ.

Vị trí nguồn bắt đầu bằng dấu gạch chéo (`/`) được coi là đường dẫn tuyệt đối. Đường dẫn tuyệt đối có thông tin đầy đủ về vị trí của hình ảnh; vì vậy, chúng có thể hoạt động bất kể vị trí của tài liệu HTML. Nếu tệp hình ảnh được đặt tại một máy chủ khác trong trường hợp khi *Mạng phân phối nội dung* (CDN) được sử dụng thì sẽ phải có cả tên miền.

NOTE Mạng Phân phối Nội dung bao gồm các máy chủ được phân bố theo vị trí địa lý lưu trữ nội dung tĩnh cho các trang web khác. Chúng giúp cải thiện hiệu suất và tính khả dụng cho các trang web có lượng truy cập lớn.

Nếu không thể tải được hình ảnh, trình duyệt HTML sẽ hiển thị văn bản được cung cấp bởi thuộc tính `alt` thay vì hình ảnh. Ví dụ:

```
<img src="logo.png" alt="The Company logo">
```

Thuộc tính `alt` cũng rất quan trọng đối với khả năng truy cập. Trình duyệt chỉ hỗ trợ văn bản và trình đọc màn hình đều sử dụng nó làm mô tả cho hình ảnh tương ứng.

Các loại Hình ảnh

Trình duyệt web có thể hiển thị tất cả các loại hình ảnh phổ biến, chẳng hạn như JPEG, PNG, GIF và SVG. Kích thước của hình ảnh có thể được xác định ngay khi hình ảnh được tải, nhưng chúng cũng có thể được xác định trước bằng các thuộc tính `width` (chiều ngang) và `height` (chiều cao):

```
<img src="logo.png" alt="The Company logo" width="300" height="100">
```

Lý do duy nhất để thêm các thuộc tính kích thước cho thẻ `<img>` là để tránh phá vỡ bố cục khi hình ảnh mất quá nhiều thời gian để tải hoặc khi không thể tải được hình ảnh. Việc sử dụng thuộc tính `width` và `height` để thay đổi kích thước ban đầu của hình ảnh có thể dẫn đến những kết quả không mong muốn:

- Hình ảnh sẽ bị biến dạng khi kích thước ban đầu nhỏ hơn kích thước mới hoặc khi tỷ lệ mới khác so với tỷ lệ ban đầu.
- Việc thu nhỏ kích thước hình ảnh lớn sẽ tốn thêm băng thông, dẫn đến thời gian tải lâu hơn.

SVG là định dạng duy nhất không bị ảnh hưởng bởi các thuộc tính này vì tất cả thông tin đồ họa của nó đều được lưu trữ ở tọa độ số rất phù hợp cho việc chia tỷ lệ và kích thước mà không ảnh hưởng đến kích thước tệp (do đó mà nó có tên là *Đồ họa Véc tơ có thể mở rộng*). Ví dụ: chỉ cần có thông tin về vị trí, kích thước cạnh và màu để có thể vẽ một hình chữ nhật trong SVG. Giá trị cụ thể cho từng pixel sẽ được hiển thị động sau đó. Trên thực tế, hình ảnh SVG cũng tương tự như tệp HTML ở việc các thành phần đồ họa của chúng cũng được xác định bởi các thẻ trong tệp văn bản. Các tệp SVG được dùng để thể hiện các bản vẽ sắc nét, chẳng hạn như biểu đồ hoặc sơ đồ.

Hình ảnh không phù hợp với các tiêu chí này sẽ được lưu trữ dưới dạng *ảnh mành hoá* (bitmaps). Không giống như các định dạng hình ảnh dựa trên vectơ, ảnh mành hoá sẽ lưu trữ thông tin màu cho từng pixel của hình ảnh trước đó. Việc lưu trữ giá trị màu cho từng pixel trong hình ảnh sẽ tạo ra một lượng dữ liệu rất lớn; vì thế, ảnh mành hoá thường được lưu trữ ở định dạng nén, chẳng hạn như JPEG, PNG hoặc GIF.

Định dạng JPEG được khuyên dùng cho các bức ảnh vì thuật toán nén của nó có thể tạo ra các kết quả tốt về sắc thái và hậu cảnh mờ. Đối với những hình ảnh có phần lớn là màu sắc đặc, định dạng PNG sẽ phù hợp hơn. Do đó, định dạng PNG nên được chọn khi cần chuyển đổi hình ảnh từ vectơ thành ảnh mành hoá.

Định dạng GIF cung cấp chất lượng hình ảnh thấp nhất trong tất cả các định dạng ảnh mành hoá phổ biến. Tuy nhiên, nó vẫn được sử dụng rộng rãi vì nó hỗ trợ hoạt ảnh. Thật vậy, có rất nhiều trang web sử dụng các tệp GIF để hiển thị các video ngắn, nhưng trên thực tế thì có nhiều cách tốt hơn để hiển thị các nội dung video.

Âm thanh và Video

Nội dung âm thanh và video có thể được thêm vào tài liệu HTML theo cách tương tự với hình ảnh. Không có gì lạ khi thẻ để thêm âm thanh là `<audio>` và thẻ để thêm video là `<video>`. Rõ ràng là các trình duyệt chỉ hỗ trợ văn bản không thể phát nội dung đa phương tiện; vì vậy, các thẻ `<audio>` và `<video>` phải sử dụng thẻ đóng để giữ văn bản được sử dụng như một phần dự phòng cho các phần tử không thể được hiển thị. Ví dụ:

```
<audio controls src="/media/recording.mp3">
<p>Unable to play <em>recording.mp3</em></p>
</audio>
```

Nếu trình duyệt không hỗ trợ thẻ `<audio>`, dòng “Unable to play recording.mp3” (Không thể phát bản ghi.mp3) sẽ được hiển thị. Việc sử dụng các thẻ đóng `</audio>` hoặc `</video>` cho phép trang web có những nội dung thay thế phức tạp hơn so với dòng văn bản đơn giản được cho phép bởi thuộc tính `alt` của thẻ `<img>`.

Thuộc tính `src` cho thẻ `<audio>` và `<video>` hoạt động theo cách tương tự như đối với thẻ `<img>`, nhưng nó cũng sẽ chấp nhận URL trỏ tới luồng trực tiếp. Trình duyệt đảm nhiệm việc đệm, giải mã và hiển thị nội dung khi nhận được. Thuộc tính `controls` sẽ hiển thị phần điều khiển các mục phát. Không có nó, khách truy cập sẽ không thể tạm dừng, tua hoặc kiểm soát quá trình phát.

Nội dung chung

Một tài liệu HTML này có thể được lồng vào một tài liệu HTML khác (tương tự như việc chèn hình ảnh vào tài liệu HTML) bằng cách sử dụng thẻ `<iframe>`:

```
<iframe name="viewer" src="gallery.html">
<p>Unsupported browser</p>
</iframe>
```

Các trình duyệt chỉ hỗ trợ văn bản đơn giản sẽ không hỗ trợ thẻ `<iframe>` và thay vào đó sẽ hiển thị văn bản đính kèm. Với các thẻ đa phương tiện, thuộc tính `src` sẽ đặt vị trí nguồn của tài liệu được lồng vào. Có thể thêm các thuộc tính `width` và `height` để thay đổi kích thước mặc định của phần tử `iframe`.

Thuộc tính `name` cho phép nhà phát triển tham chiếu đến iframe và thay đổi tài liệu được lồng vào. Không có thuộc tính này, tài liệu được lồng vào sẽ không thể thay đổi được. Phần tử `anchor` có thể được sử dụng để tải tài liệu từ một vị trí khác bên trong iframe thay vì cửa sổ trình duyệt hiện tại.

Các Liên kết

Một phần tử trang thường được gọi là *liên kết* web còn được gọi là *liên kết neo* (trong tiếng Anh là "anchor", do đó mà thẻ của nó là `<a>`) theo ngôn ngữ kỹ thuật. Liên kết neo sẽ dẫn đến một vị trí khác, có thể là bất kỳ địa chỉ nào được trình duyệt hỗ trợ. Vị trí được chỉ định bởi thuộc tính `href` (*tham chiếu siêu liên kết* - hyperlink reference):

```
<a href="contact.html">Contact Information</a>
```

Vị trí có thể được viết dưới dạng đường dẫn tương đối hoặc tuyệt đối như với nội dung nhúng đã thảo luận trước đây. Chỉ nội dung văn bản đính kèm (ví dụ như `Contact Information`) mới hiển thị với khách truy cập, thường mặc định ở dưới dạng văn bản màu xanh được gạch chân và có thể nhấp vào được. Tuy vậy, mục được hiển thị qua liên kết cũng có thể là bất kỳ nội dung hiển thị nào khác, chẳng hạn như hình ảnh:

```
<a href="contact.html"><img src="contact.png" alt="Contact Information"></a>
```

Các tiền tố đặc biệt có thể được thêm vào vị trí để cho trình duyệt biết cách mở nó. Ví dụ: nếu liên kết neo trỏ đến một địa chỉ email thì thuộc tính `href` của nó phải bao gồm tiền tố `mailto:`:

```
<a href="mailto:info@lpi.org">Contact by email</a>
```

Tiền tố `tel:` cho biết số điện thoại. Nó đặc biệt hữu ích đối với khách truy cập xem trang trên thiết bị di động:

```
<a href="tel:+123456789">Contact by phone</a>
```

Khi nhấp vào liên kết, trình duyệt sẽ mở nội dung của vị trí với ứng dụng được liên kết.

Tác dụng phổ biến nhất của liên kết neo là tải các tài liệu web khác. Theo mặc định, trình duyệt sẽ thay thế tài liệu HTML hiện tại bằng nội dung tại vị trí mới. Hành vi này có thể được sửa đổi bằng cách sử dụng thuộc tính `target` (mục tiêu). Ví dụ: mục tiêu `_blank` yêu cầu trình duyệt mở vị trí đã cho trong cửa sổ mới hoặc tab trình duyệt mới, tùy thuộc vào tùy chọn của khách truy cập:

```
<a href="contact.html" target="_blank">Contact Information</a>
```

Mục tiêu `_self` sẽ là mặc định khi thuộc tính `targer` không được cung cấp. Nó sẽ làm cho tài liệu được tham chiếu thay thế tài liệu hiện tại.

Các loại mục tiêu khác có liên quan đến phần tử `<iframe>`. Để tải tài liệu được tham chiếu bên trong phần tử `<iframe>`, thuộc tính `target` phải trỏ đến tên của phần tử iframe:

```
<p><a href="gallery.html" target="viewer">Photo Gallery</a></p>

<iframe name="viewer" width="800" height="600">
<p>Unsupported browser</p>
</iframe>
```

Phần tử iframe hoạt động như một cửa sổ trình duyệt riêng biệt; vì vậy, mọi liên kết được tải từ tài liệu bên trong iframe sẽ chỉ thay thế nội dung của iframe. Để thay đổi hành vi đó, các phần tử liên kết neo bên trong tài liệu khuôn cũng có thể sử dụng thuộc tính `target`. Mục tiêu `_parent` khi được sử dụng bên trong tài liệu khuôn sẽ khiến vị trí được tham chiếu thay thế tài liệu gốc chứa thẻ `<iframe>`. Ví dụ: tài liệu `gallery.html` được nhúng có thể chứa một liên kết neo tự tải cùng lúc thay thế tài liệu gốc:

```
<p><a href="gallery.html" target="_parent">Open as parent document</a></p>
```

Các tài liệu HTML hỗ trợ nhiều mức độ lồng với thẻ `<iframe>`. Việc sử dụng mục tiêu `_top` trong một liên kết neo bên trong tài liệu khuôn sẽ khiến vị trí được tham chiếu thay thế tài liệu chính trong cửa sổ trình duyệt, bất kể đó là phân tử mẹ hay họ hàng xa hơn nữa trong chuỗi của `<iframe>` tương ứng.

Vị trí bên trong Tài liệu

Địa chỉ của tài liệu HTML có thể tùy chọn chứa một *mảnh* được sử dụng để xác định tài nguyên bên trong tài liệu. Mảnh này (còn được gọi là *URL neo*) là một chuỗi theo sau dấu thăng # ở cuối URL. Ví dụ: từ `History` là phần neo trong URL `https://en.wikipedia.org/wiki/Internet#History`.

Khi URL có một phần neo, trình duyệt sẽ cuộn đến phần tử tương ứng trong tài liệu: nghĩa là phần tử có thuộc tính `id` giống với phần neo trong URL. Trong trường hợp URL đã cho (`https://en.wikipedia.org/wiki/Internet#History`), trình duyệt sẽ chuyển thẳng đến phần “History”. Khi kiểm tra mã HTML của trang, chúng ta sẽ phát hiện ra rằng tiêu đề của phần

cũng có thuộc tính `id` tương ứng:

```
<span class="mw-headline" id="History">History</span>
```

Các phần neo của URL có thể được sử dụng trong thuộc tính `href` của thẻ `<a>` cả khi chúng trỏ đến các trang bên ngoài lẫn các vị trí ở bên trong trang hiện tại. Trong trường hợp các vị trí ở bên trong trang hiện tại, ta chỉ cần bắt đầu bằng dấu thăng với đoạn URL như trong `<a href="#History">History</a>` là đủ.

WARNING Thuộc tính `id` không được chứa khoảng trắng (dấu cách, tab, v.v.) và phải là duy nhất trong tài liệu.

Có nhiều cách để tùy chỉnh cách trình duyệt phản ứng với các liên kết URL. Ví dụ: có thể viết một hàm JavaScript lắng nghe sự kiện cửa sổ *thay đổi giá trị băm* (hashchange) và kích hoạt một hành động tùy chỉnh, chẳng hạn như hoạt ảnh hoặc yêu cầu HTTP. Tuy nhiên, điều đáng chú ý là đoạn URL sẽ không bao giờ được gửi đến máy chủ cùng với URL; vì vậy, nó không thể được máy chủ HTTP sử dụng làm mã định danh.

Bài tập Hướng dẫn

1. Tài liệu HTML tại `http://www.lpi.org/articles/linux/index.html` có thẻ `<img>` có thuộc tính `src` trỏ tới `../logo.png`. Đường dẫn tuyệt đối hoàn chỉnh đến hình ảnh này là gì?

2. Hãy nêu hai lý do tại sao thuộc tính `alt` lại quan trọng trong các thẻ `<img>`.

3. Định dạng hình ảnh nào mang lại chất lượng hình ảnh tốt và giữ cho kích thước tệp nhỏ khi nó được sử dụng cho các bức ảnh có nhiều điểm mờ, màu sắc và mức màu?

4. Thay vì sử dụng nhà cung cấp bên thứ ba như Youtube, thẻ HTML nào cho phép bạn nhúng tệp video vào tài liệu HTML chỉ bằng các tính năng HTML tiêu chuẩn?

Bài tập Mở rộng

1. Giả sử rằng tài liệu HTML có siêu liên kết `<a href="pic1.jpeg">First picture</a>` và phần tử iframe `<iframe name="gallery"></iframe>`. Làm cách nào để có thể sửa đổi thẻ siêu liên kết để hình ảnh mà thẻ trỏ tới sẽ tải bên trong phần tử iframe đã cho sau khi người dùng nhấp vào liên kết?

2. Điều gì sẽ xảy ra khi khách truy cập nhấp vào siêu liên kết trong tài liệu bên trong iframe và siêu liên kết có thuộc tính `target` được đặt thành `_self`?

3. Bạn nhận thấy rằng phần neo URL dành cho phần thứ hai của trang HTML không hoạt động. Nguyên nhân của lỗi này có thể là gì?

Tóm tắt

Bài học này đề cập tới cách thêm hình ảnh và các nội dung đa phương tiện khác bằng cách sử dụng các thẻ HTML thích hợp. Hơn nữa, người đọc đã được tìm hiểu các cách khác nhau mà siêu liên kết có thể được sử dụng để tải các tài liệu khác và trỏ đến các vị trí cụ thể bên trong một trang. Bài học này đã đi qua các khái niệm và quy trình sau:

- Thẻ `<img>` và các thuộc tính chính của nó: `src` và `alt`.
- Đường dẫn URL tương đối và tuyệt đối.
- Các định dạng hình ảnh phổ biến cho Web và đặc điểm của chúng.
- Thẻ đa phương tiện `<audio>` và `<video>`.
- Cách chèn tài liệu lồng nhau với thẻ <iframe>.
- Thẻ siêu liên kết `<a>`, thuộc tính `href` của nó và các mục tiêu đặc biệt.
- Cách sử dụng các mảnh URL hay còn gọi là neo băm.

Đáp án Bài tập Hướng dẫn

1. Tài liệu HTML tại `http://www.lpi.org/articles/linux/index.html` có thẻ `<img>` có thuộc tính `src` trỏ tới `../logo.png`. Đường dẫn tuyệt đối hoàn chỉnh đến hình ảnh này là gì?

   `http://www.lpi.org/articles/logo.png`

2. Hãy nêu hai lý do tại sao thuộc tính `alt` lại quan trọng trong các thẻ `<img>`.

   Các trình duyệt chỉ hỗ trợ văn bản có thể hiển thị mô tả về hình ảnh bị thiếu. Trình đọc màn hình sử dụng thuộc tính `alt` để mô tả hình ảnh.

3. Định dạng hình ảnh nào mang lại chất lượng hình ảnh tốt và giữ cho kích thước tệp nhỏ khi nó được sử dụng cho các bức ảnh có nhiều điểm mờ, màu sắc và mức màu?

   Định dạng JPEG.

4. Thay vì sử dụng nhà cung cấp bên thứ ba như Youtube, thẻ HTML nào cho phép bạn nhúng tệp video vào tài liệu HTML chỉ bằng các tính năng HTML tiêu chuẩn?

   Thẻ `<video>`.

Đáp án Bài tập Mở rộng

1. Giả sử rằng tài liệu HTML có siêu liên kết `<a href="pic1.jpeg">First picture</a>` và phần tử iframe `<iframe name="gallery"></iframe>`. Làm cách nào để có thể sửa đổi thẻ siêu liên kết để hình ảnh mà thẻ trỏ tới sẽ tải bên trong phần tử iframe đã cho sau khi người dùng nhấp vào liên kết?

   Sử dụng thuộc tính `target` của thẻ `a`: `<a href="pic1.jpeg" target="gallery">First picture</a>`.

2. Điều gì sẽ xảy ra khi khách truy cập nhấp vào siêu liên kết trong tài liệu bên trong iframe và siêu liên kết có thuộc tính `target` được đặt thành `_self`?

   Tài liệu sẽ được tải bên trong cùng một iframe, đây là hành vi mặc định.

3. Bạn nhận thấy rằng phần neo URL dành cho phần thứ hai của trang HTML không hoạt động. Nguyên nhân của lỗi này có thể là gì?

   Mảnh URL sau dấu thăng không khớp với thuộc tính `id` trong phần tử tương ứng với phần thứ hai hoặc không có thuộc tính `id` của phần tử.

032.4 Biểu mẫu HTML

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 032.4

Khối lượng

2

Các lĩnh vực kiến thức chính

- Tạo biểu mẫu HTML đơn giản
- Hiểu về các phương thức biểu mẫu HTML
- Hiểu về các thành phần và loại đầu vào HTML

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `<form>`, bao gồm các thuộc tính `method` (get, post), `action` và `enctype`
- `<input>, ao gồm thuộc tính `type` (`text`, `email`, `password`, `number`, `date`, `file`, `range`, `radio`, `checkbox`, `hidden`)
- `<button>`, bao gồm thuộc tính `type` (`submit`, `reset`, `hidden`, `button`)
- Thuộc tính thành phần biểu mẫu phổ biến (`name`, `value`, `id`)
- `<label>, bao gồm thuộc tính `for`

Linux Professional Institute

032.4 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	032 Đánh dấu Tài liệu HTML
Mục tiêu:	032.4 Biểu mẫu HTML
Bài:	1 trên 1

Giới thiệu

Biểu mẫu web cung cấp một cách đơn giản và hiệu quả để có thể yêu cầu thông tin về khách truy cập từ trang HTML. Nhà phát triển giao diện người dùng có thể sử dụng các thành phần khác nhau như trường văn bản, hộp kiểm, nút bấm và nhiều thành phần khác để xây dựng giao diện gửi dữ liệu đến máy chủ theo một cách có cấu trúc.

Biểu mẫu HTML đơn giản

Trước khi nói đến mã đánh dấu cụ thể cho các biểu mẫu, hãy cùng bắt đầu với một tài liệu HTML trống đơn giản và không có bất kỳ nội dung nào:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Working with HTML Forms</title>
</head>
<body>
```

```
<!-- The body content goes here -->

</body>
</html>
```

Hãy lưu mẫu mã dưới dạng tệp văn bản thô có phần mở rộng là `.html` (như là `form.html`) và sử dụng trình duyệt yêu thích của bạn để mở nó. Sau khi thay đổi, hãy nhấn nút tải lại trên trình duyệt để hiển thị các sửa đổi.

Cấu trúc biểu mẫu cơ bản sẽ được cung cấp bởi chính thẻ `<form>` và các thành phần bên trong của nó:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Working with HTML Forms</title>
</head>
<body>

<!-- Form to collect personal information -->

<form>

<h2>Personal Information</h2>

<p>Full name:</p>
<p><input type="text" name="fullname" id="fullname"></p>

<p><input type="reset" value="Clear form"></p>
<p><input type="submit" value="Submit form"></p>

</form>

</body>
</html>
```

Dấu trích dẫn kép không phải là bắt buộc đối với các thuộc tính một từ như `type`; vì vậy, `type=text` cũng sẽ hoạt động như `type="text"`. Nhà phát triển có thể chọn sử dụng bất kỳ kiểu quy ước nào mà họ muốn.

Hãy lưu nội dung mới và tải lại trang trong trình duyệt. Bạn sẽ thấy kết quả hiển thị trong Figure 23.

Figure 23. Một biểu mẫu cơ bản.

Bản thân thẻ `<form>` sẽ không tạo ra bất kỳ kết quả đáng chú ý nào ở trên trang. Các phần tử bên trong thẻ `<form>...</form>` sẽ xác định các trường và các hỗ trợ trực quan khác được hiển thị cho khách truy cập.

Mã ở trong ví dụ có chứa cả thẻ HTML chung (`<h2>` và `<p>`) và thẻ `<input>` dành riêng cho biểu mẫu. Trong khi các thẻ chung có thể xuất hiện ở bất kỳ đâu trong tài liệu thì các thẻ dành riêng cho biểu mẫu chỉ nên được sử dụng bên trong phần tử `<form>`, nghĩa là giữa thẻ `<form>` mở và thẻ `</form>` đóng.

NOTE HTML chỉ cung cấp các thẻ và thuộc tính cơ bản để sửa đổi giao diện tiêu chuẩn của biểu mẫu. CSS sẽ cung cấp các cơ chế phức tạp để sửa đổi giao diện của biểu mẫu; vì vậy, khuyến nghị ở đây là dùng mã HTML cho việc xử lý các khía cạnh chức năng của biểu mẫu và sửa đổi giao diện của biểu mẫu bằng CSS.

Như được minh họa trong ví dụ, thẻ đoạn văn bản `<p>` có thể được sử dụng để mô tả trường cho khách truy cập. Tuy nhiên, không có cách rõ ràng nào để trình duyệt có thể liên kết mô tả trong thẻ `<p>` với phần tử đầu vào tương ứng. Thẻ `<label>`(nhãn) sẽ phù hợp hơn trong những trường hợp này (từ giờ trở đi, hãy xem xét tất cả các mẫu mã nằm trong phần nội dung của tài liệu HTML):

```
<form>
```

```
<h2>Personal Information</h2>

<label for="fullname">Full name:</label>
<p><input type="text" name="fullname" id="fullname"></p>

<p><input type="reset" value="Clear form"></p>
<p><input type="submit" value="Submit form"></p>

</form>
```

Thuộc tính `for` trong thẻ `<label>` chứa `id` của phần tử đầu vào tương ứng. Nó sẽ làm cho trang trở nên dễ truy cập hơn vì trình đọc màn hình sẽ có thể đọc nội dung của phần tử label khi phần tử đầu vào được đặt vào tiêu điểm. Ngoài ra, khách truy cập có thể nhấp vào nhãn để đặt tiêu điểm vào trường nhập dữ liệu tương ứng.

Thuộc tính `id` thực hiện cùng một công việc cho các phần tử biểu mẫu như với bất kỳ phần tử nào khác trong tài liệu. Nó cung cấp một mã định danh cho phần tử là duy nhất trong toàn bộ tài liệu. Thuộc tính `name` cũng có mục đích tương tự, nhưng nó được sử dụng để xác định thành phần đầu vào trong ngữ cảnh của biểu mẫu. Trình duyệt sẽ sử dụng thuộc tính `name` để xác định trường nhập khi gửi dữ liệu biểu mẫu đến máy chủ; vì vậy, điều quan trọng là phải sử dụng các thuộc tính `name` một cách có ý nghĩa và duy nhất ở bên trong của biểu mẫu.

Thuộc tính `type` là thuộc tính chính của phần tử đầu vào vì nó kiểm soát loại dữ liệu mà phần tử chấp nhận và cách trình bày trực quan của nó đối với khách truy cập. Nếu thuộc tính `type` không được cung cấp, đầu vào sẽ hiển thị một hộp văn bản theo mặc định. Các loại đầu vào sau đây hiện được hỗ trợ bởi các trình duyệt hiện đại:

Table 1. Các loại đầu vào của Biểu mẫu

Giá trị của `type`	**Kiểu dữ liệu**	**Kiểu hiển thị**
`hidden`	Một chuỗi tùy ý	Không áp dụng
`text`	Văn bản không ngắt dòng	Kiểm soát văn bản
`search`	Văn bản không ngắt dòng	Kiểm soát tìm kiếm
`tel`	Văn bản không ngắt dòng	Kiểm soát văn bản
`url`	URL tuyệt đối	Kiểm soát văn bản
`email`	Địa chỉ email hoặc danh sách địa chỉ email	Kiểm soát văn bản

Giá trị của `type`	**Kiểu dữ liệu**	**Kiểu hiển thị**
`password`	Văn bản không ngắt dòng (thông tin nhạy cảm)	Kiểm soát văn bản che khuất mục nhập
`date`	Một ngày (năm, tháng, ngày) không có múi giờ	Kiểm soát ngày
`month`	Một ngày bao gồm một năm và một tháng không có múi giờ	Kiểm soát một tháng
`week`	Ngày bao gồm số tuần trong năm và số tuần không có múi giờ	Kiểm soát một tuần
`time`	Thời gian (giờ, phút, giây, giây thập phân) không có múi giờ	Kiểm soát thời gian
`datetime-local`	Ngày và giờ (năm, tháng, ngày, giờ, phút, giây, phần giây) không có múi giờ	Kiểm soát ngày và giờ
`number`	Một giá trị số	Kiểm soát văn bản hoặc kiểm soát tăng giảm
`range`	Một giá trị số với ngữ nghĩa bổ sung mà giá trị chính xác là không quan trọng	Kiểm soát thanh trượt hoặc tương tự
`color`	Một màu sRGB với các thành phần đỏ, lục và lam 8 bit	Bộ chọn màu
`checkbox`	Một tập hợp gồm 0 hoặc nhiều giá trị từ danh sách được xác định trước	Một hộp kiểm (cung cấp các lựa chọn và cho phép nhiều lựa chọn có thể được chọn)
`radio`	Một giá trị liệt kê	Nút radio (cung cấp các lựa chọn và chỉ cho phép chọn một lựa chọn)
`file`	Không hoặc nhiều tệp, mỗi tệp có loại MIME và tên tệp tùy chọn	Một nhãn và một nút bấm
`submit`	Một giá trị liệt kê kết thúc quá trình nhập liệu và làm cho biểu mẫu được gửi	Một nút bấm

Giá trị của `type`	**Kiểu dữ liệu**	**Kiểu hiển thị**
`image`	Một tọa độ liên quan đến kích thước của một hình ảnh cụ thể kết thúc quá trình nhập liệu và khiến biểu mẫu được gửi	Một hình ảnh có thể nhấp hoặc một nút bấm
`button`	Không áp dụng	Một nút chung
`reset`	Không áp dụng	Nút có chức năng đặt lại tất cả các trường khác về giá trị ban đầu của chúng

Hình thức của các loại đầu vào `password`, `search`, `tel`, `url` và `email` sẽ không khác so với hình thức của `text` tiêu chuẩn. Mục đích của chúng là để đưa ra các gợi ý cho trình duyệt về nội dung dự định cho trường đầu vào đó, để trình duyệt hoặc tệp lệnh chạy ở phía máy khách có thể thực hiện các hành động tùy chỉnh cho một loại đầu vào cụ thể. Ví dụ như sự khác biệt duy nhất giữa kiểu nhập văn bản và kiểu trường mật khẩu là nội dung của trường mật khẩu không được hiển thị khi khách truy cập nhập chúng vào. Trong các thiết bị màn hình cảm ứng nơi văn bản được nhập bằng biểu tượng trên màn hình bàn phím, trình duyệt sẽ chỉ có thể bật lên bàn phím số khi đầu vào thuộc loại `tel` được chọn làm tiêu điểm. Một hành động khả thi khác là đề xuất một danh sách các địa chỉ email đã biết khi đầu vào thuộc loại `email` được chọn làm tiêu điểm.

Loại `number` cũng xuất hiện dưới dạng kiểu nhập văn bản đơn giản nhưng sẽ có các mũi tên tăng/giảm ở bên cạnh. Việc sử dụng nó sẽ làm cho bàn phím số xuất hiện trên các thiết bị màn hình cảm ứng khi nó trở thành tiêu điểm.

Các phần tử đầu vào khác cũng có giao diện và hành vi riêng. Ví dụ: loại `date` sẽ được hiển thị theo cài đặt định dạng ngày địa phương và lịch sẽ được hiển thị khi trường trở thành tiêu điểm:

```
<form>

<p>
  <label for="date">Date:</label>
  <input type="date" name="date" id="date">
</p>

</form>
```

[img-fig02] cho thấy cách mà phiên bản máy tính để bàn của Firefox đang hiển thị trường này.

#img-fig07] .Loại đầu vào ngày tháng. image::./images/fig07.png[]

NOTE Các phần tử có thể xuất hiện hơi khác nhau trong các trình duyệt hoặc hệ điều hành khác nhau, nhưng chức năng và cách sử dụng của chúng sẽ luôn giống nhau.

Đây là một tính năng tiêu chuẩn trong tất cả các trình duyệt hiện đại và không cần có các tùy chọn hoặc lập trình bổ sung.

Bất kể loại đầu vào là gì, nội dung của trường đầu vào được gọi là *giá trị* của nó. Tất cả các giá trị của trường đều sẽ trống theo mặc định, nhưng ta có thể sử dụng thuộc tính `value` để đặt giá trị mặc định cho trường. Giá trị cho loại `date` phải sử dụng định dạng *NNNN-TT-NN*. Giá trị mặc định trong trường ngày sau là vào ngày 6 tháng 8 năm 2020:

```
<form>

<p>
  <label for="date">Date:</label>
  <input type="date" name="date" id="date" value="2020-08-06">
</p>

</form>
```

Các loại đầu vào cụ thể sẽ hỗ trợ khách truy cập điền vào các trường, nhưng chúng sẽ không ngăn khách truy cập bỏ qua các hạn chế và nhập các giá trị tùy ý vào bất kỳ trường nào. Đó là lý do tại sao điều quan trọng ở đây là các giá trị trường phải được xác thực khi chúng đến máy chủ.

Các thành phần biểu mẫu có giá trị phải được nhập bởi khách truy cập có thể sẽ có các thuộc tính đặc biệt để hỗ trợ điền chúng. Thuộc tính `placeholder` (giữ chỗ) sẽ chèn một giá trị ví dụ vào phần tử đầu vào:

```
<p>Address: <input type="text" name="address" id="address" placeholder="e.g. 41 John St.,
Upper Suite 1"></p>
```

Phần giữ chỗ sẽ xuất hiện bên trong phần tử đầu vào như được hiển thị trong Figure 24.

Figure 24. Ví dụ về thuộc tính `placeholder`.

Một khi khách truy cập bắt đầu nhập vào trường, văn bản giữ chỗ sẽ biến mất. Văn bản giữ chỗ sẽ không được gửi dưới dạng giá trị trường nếu khách truy cập để trống trường.

Thuộc tính `required` (bắt buộc) yêu cầu khách truy cập điền giá trị cho trường tương ứng trước khi gửi biểu mẫu:

```
<p>Address: <input type="text" name="address" id="address" required placeholder="e.g. 41
John St., Upper Suite 1"></p>
```

Thuộc tính `required` là một thuộc tính Boolean; vì vậy, nó có thể được đặt một mình (mà không có dấu bằng). Việc đánh dấu các trường bắt buộc là rất quan trọng; nếu không, khách truy cập sẽ không thể biết trường nào bị thiếu và sẽ không gửi được biểu mẫu.

Thuộc tính `autocomplete` (tự hoàn thành) sẽ cho biết liệu giá trị của phần tử đầu vào có thể được trình duyệt tự động hoàn thành hay không. Nếu được đặt thành `autocomplete="off"` thì trình duyệt sẽ không đề xuất các giá trị đã từng xuất hiện trước đó để điền vào. Các phần tử đầu vào dành cho thông tin nhạy cảm (chẳng hạn như số thẻ tín dụng) phải đặt thuộc tính `autocomplete` thành `off`.

Đầu vào cho văn bản cỡ lớn: textarea

Không giống như trường văn bản chỉ có thể chèn một dòng văn bản, phần tử `textarea` (vùng văn bản) sẽ cho phép khách truy cập nhập nhiều hơn một dòng văn bản. Vùng văn bản là một phần tử riêng biệt và không dựa trên phần tử đầu vào:

```
<p> <label for="comment">Type your comment here:</label> <br>

<textarea id="comment" name="comment" rows="10" cols="50">
My multi-line, plain-text comment.
</textarea>

</p>
```

Giao diện điển hình của vùng văn bản là Figure 25.

Figure 25. Phần tử `textarea`.

Một điểm khác biệt khác so với phần tử đầu vào là phần tử `textarea` có thẻ đóng (`</textarea>`); vì vậy, nội dung của nó (tức giá trị của nó) sẽ nằm giữa hai thẻ này. Thuộc tính `rows` (hàng) và `cols` (cột) sẽ không giới hạn số lượng văn bản; chúng chỉ được sử dụng để xác định bố cục. Vùng văn bản cũng có một thanh điều khiển ở góc dưới cùng bên phải, cho phép khách truy cập thay đổi kích thước của nó.

Danh sách Tùy chọn

Một số loại kiểm soát biểu mẫu có thể được sử dụng để hiển thị danh sách các tùy chọn cho khách truy cập: phần tử `<select>` (chọn) và các loại đầu vào `radio` (phát thanh) và `checkbox` (hộp kiểm).

Phần tử `<select>` là một bảng điều khiển thả xuống với một danh sách các mục đã được xác định trước:

```
<p><label for="browser">Favorite Browser:</label>
<select name="browser" id="browser">
<option value="firefox">Mozilla Firefox</option>
<option value="chrome">Google Chrome</option>
<option value="opera">Opera</option>
<option value="edge">Microsoft Edge</option>
</select>
</p>
```

Thẻ `<option>` (tuỳ chọn) đại diện cho một mục duy nhất trong phần điều khiển `<select>` tương ứng. Toàn bộ danh sách sẽ xuất hiện khi khách truy cập chạm hoặc bấm vào điều khiển như được hiển thị trong Figure 26.

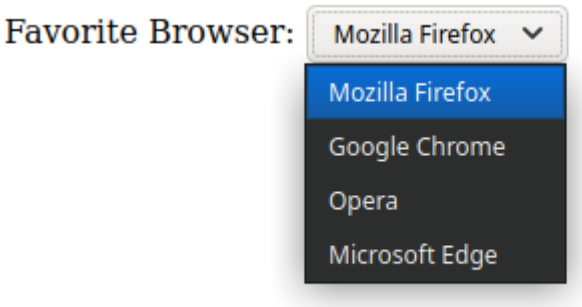

Figure 26. Phần tử biểu mẫu `select`.

Mục nhập đầu tiên trong danh sách sẽ được chọn theo mặc định. Để thay đổi hành vi này, ta có thể thêm thuộc tính `selected` (đã chọn) vào một mục nhập khác để nó được chọn khi tải trang.

Loại đầu vào `radio` cũng tương tự như phần tử điều khiển `<select>`, nhưng thay vì danh sách thả xuống, nó sẽ hiển thị tất cả các mục để khách truy cập có thể đánh dấu một trong số chúng. Kết quả của đoạn mã sau sẽ được hiển thị trong Figure 27.

```
<p>Favorite Browser:</p>

<p>
    <input type="radio" id="browser-firefox" name="browser" value="firefox" checked>
  <label for="browser-firefox">Mozilla Firefox</label>
</p>

<p>
    <input type="radio" id="browser-chrome" name="browser" value="chrome">
  <label for="browser-chrome">Google Chrome</label>
</p>

<p>
    <input type="radio" id="browser-opera" name="browser" value="opera">
  <label for="browser-opera">Opera</label>
</p>

<p>
    <input type="radio" id="browser-edge" name="browser" value="edge">
  <label for="browser-edge">Microsoft Edge</label>
```

```
</p>
```

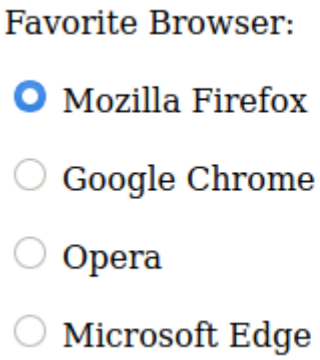

Figure 27. Các phần tử đầu vào của loại `radio`.

Hãy lưu ý rằng tất cả các loại đầu vào `radio` trong cùng một nhóm đều có cùng thuộc tính `name`. Tất cả đều là độc quyền; vì vậy, thuộc tính `value` tương ứng cho mục nhập đã chọn sẽ là thuộc tính được liên kết với thuộc tính `name` được chia sẻ. Thuộc tính `checked` (đã chọn) hoạt động giống như thuộc tính `selected` của phần tử điều khiển `<select>`. Nó sẽ đánh dấu mục tương ứng khi trang tải lần đầu tiên. Thẻ `<label>` sẽ đặc biệt hữu ích cho các mục nhập radio vì nó cho phép khách truy cập kiểm tra mục nhập bằng cách nhấp hoặc nhấn vào văn bản tương ứng bên cạnh phần điều khiển.

Trong khi các phần điều khiển `radio` chỉ dành cho việc chọn một mục duy nhất trong danh sách thì loại đầu vào `checkbox` sẽ cho phép khách truy cập chọn nhiều mục:

```
<p>Favorite Browser:</p>

<p>
    <input type="checkbox" id="browser-firefox" name="browser" value="firefox" checked>
    <label for="browser-firefox">Mozilla Firefox</label>
</p>

<p>
    <input type="checkbox" id="browser-chrome" name="browser" value="chrome" checked>
    <label for="browser-chrome">Google Chrome</label>
</p>

<p>
    <input type="checkbox" id="browser-opera" name="browser" value="opera">
    <label for="browser-opera">Opera</label>
</p>
```

```
<p>
    <input type="checkbox" id="browser-edge" name="browser" value="edge">
    <label for="browser-edge">Microsoft Edge</label>
</p>
```

Các hộp kiểm cũng có thể sử dụng thuộc tính `checked` để để chọn mặc định các mục. Thay vì phần điều khiển hình tròn của đầu vào radio, hộp kiểm sẽ được hiển thị dưới dạng các phần điều khiển hình vuôn như trong Figure 28.

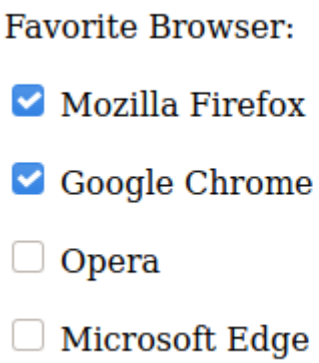

Figure 28. Loại đầu vào `checkbox`.

Nếu nhiều mục nhập được chọn, trình duyệt sẽ gửi chúng dưới cùng một tên và yêu cầu nhà phát triển phụ trợ viết mã cụ thể để đọc chính xác dữ liệu biểu mẫu có chứa các hộp kiểm.

Để cải thiện khả năng sử dụng, các trường nhập liệu có thể được nhóm lại với nhau bên trong thẻ `<fieldset>` (tập hợp trường):

```
<fieldset>
<legend>Favorite Browser</legend>

<p>
    <input type="checkbox" id="browser-firefox" name="browser" value="firefox" checked>
    <label for="browser-firefox">Mozilla Firefox</label>
</p>

<p>
    <input type="checkbox" id="browser-chrome" name="browser" value="chrome" checked>
    <label for="browser-chrome">Google Chrome</label>
</p>

<p>
    <input type="checkbox" id="browser-opera" name="browser" value="opera">
```

```
    <label for="browser-opera">Opera</label>
</p>

<p>
    <input type="checkbox" id="browser-edge" name="browser" value="edge">
    <label for="browser-edge">Microsoft Edge</label>
</p>
</fieldset>
```

Thẻ <legend> sẽ chứa văn bản được đặt ở đầu khung mà thẻ <fieldset> bao xung quanh các điều khiển (Figure 29).

Figure 29. Nhóm các phần tử với thẻ `fieldset`.

Thẻ `<fieldset>` sẽ không thay đổi cách các giá trị của trường được gửi tới máy chủ, nhưng nó sẽ cho phép nhà phát triển giao diện người dùng kiểm soát các phần điều khiển được lồng với nhau một cách dễ dàng hơn. Ví dụ: đặt thuộc tính `disabled` (vô hiệu hoá) bên trong thuộc tính `<fieldset>` sẽ làm cho tất cả các thành phần bên trong của nó không khả dụng đối với khách truy cập.

Loại Phần tử Ẩn (`hidden`)

Có những tình huống mà nhà phát triển muốn đưa một số thông tin vào biểu mẫu mà không muốn khách truy cập tương tác với những thông tin này - tức là gửi một giá trị do nhà phát triển chọn mà không hiển thị trường biểu mẫu nơi khách truy cập có thể nhập hoặc thay đổi giá trị. Ví dụ như nhà phát triển có thể muốn có mã thông báo xác thực trong một biểu mẫu cụ thể mà khách truy cập không cần nhìn thấy. Một phần tử biểu mẫu ẩn sẽ được mã hóa như trong ví dụ sau:

```
<input type="hidden" id="form-token" name="form-token" value="e730a375-b953-4393-847d-
```

```
2dab065bbc92">
```

Giá trị của trường nhập ẩn thường được thêm vào tài liệu ở phía máy chủ khi kết xuất tài liệu. Các đầu vào ẩn sẽ được coi như các trường thông thường khi trình duyệt gửi chúng đến máy chủ; máy chủ này cũng sẽ đọc chúng như đọc các trường đầu vào thông thường.

Loại Đầu vào Tệp

Ngoài dữ liệu văn bản được nhập hoặc chọn từ danh sách, các biểu mẫu HTML cũng có thể gửi các tệp tùy ý đến máy chủ. Loại đầu vào tệp (`file`) cho phép khách truy cập chọn một tệp từ hệ thống tệp cục bộ và gửi trực tiếp từ trang web:

```
<p>
<label for="attachment">Attachment:</label><br>
<input type="file" id="attachment" name="attachment">
</p>
```

Thay vì một trường biểu mẫu để ghi vào hoặc chọn một giá trị từ đó, loại đầu vào `file` sẽ hiển thị nút `browse` (duyệt) dùng để mở hộp thoại tệp. Loại tệp đầu vào `file` chấp nhận bất kỳ loại tệp nào, nhưng nhà phát triển phía máy chủ có thể sẽ hạn chế các loại tệp được phép và kích thước tối đa của chúng. Việc xác minh loại tệp cũng có thể được thực hiện ở giao diện người dùng bằng cách thêm thuộc tính `accept` (chấp nhận). Ví dụ: để chỉ chấp nhận hình ảnh JPEG và PNG, thuộc tính `accept` sẽ phải là `accept="image/jpeg, image/png"`.

Nút Hành động

Theo mặc định, biểu mẫu sẽ được nộp khi khách truy cập nhấn phím Enter tại bất kỳ trường nhập liệu nào. Để làm cho mọi thứ trực quan hơn, ta nên thêm một nút nộp với loại đầu vào `submit` (nộp):

```
<input type="submit" value="Submit form">
```

Văn bản trong thuộc tính `value` được hiển thị trên nút như trong Figure 30.

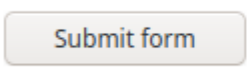

Figure 30. Một nút nộp tiêu chuẩn.

Một nút hữu ích khác để đưa vào biểu mẫu phức tạp là nút `reset` (đặt lại). Nút này sẽ xóa và đưa biểu mẫu về lại trạng thái ban đầu:

```
<input type="reset" value="Clear form">
```

Giống như nút nộp, văn bản trong thuộc tính `value` sẽ được sử dụng để gắn nhãn cho nút. Ngoài ra, thẻ `<button>` (nút bấm) có thể được sử dụng để thêm các nút vào biểu mẫu hoặc bất kỳ nơi nào khác trong trang. Không giống như các nút được tạo bằng thẻ `<input>` (đầu vào), phần tử button có thẻ đóng và nhãn của nút sẽ là nội dung bên trong của chúng:

```
<button>Submit form</button>
```

Khi ở trong một biểu mẫu, hành vi mặc định cho phần tử `button` là gửi biểu mẫu. Giống như các nút nhập liệu, thuộc tính loại của nút có thể được chuyển sang `reset`.

Hành động và Phương thức của Biểu mẫu

Bước cuối cùng trong việc viết một biểu mẫu HTML là xác định cách thức và nơi gửi dữ liệu. Những khía cạnh này sẽ phụ thuộc vào các chi tiết ở cả máy khách và máy chủ.

Về phía máy chủ, cách tiếp cận phổ biến nhất là có một tệp tệp lệnh phân tích cú pháp, xác thực và xử lý dữ liệu biểu mẫu theo mục đích của ứng dụng. Ví dụ: nhà phát triển phía máy chủ có thể viết tệp lệnh có tên `receive_form.php` để nhận dữ liệu được gửi từ biểu mẫu. Về phía máy khách, tệp lệnh sẽ được chỉ định trong thuộc tính `action` (hành động) của thẻ biểu mẫu:

```
<form action="receive_form.php">
```

Thuộc tính `action` sẽ tuân theo các quy ước giống như tất cả các địa chỉ HTTP. Nếu tệp lệnh ở cùng mức phân cấp của trang chứa biểu mẫu thì tệp lệnh có thể được viết mà không cần đường dẫn của nó. Mặt khác, đường dẫn tuyệt đối hoặc tương đối sẽ phải được cung cấp. Tệp lệnh cũng sẽ tạo phản hồi để phục vụ dưới dạng trang đích được trình duyệt tải sau khi khách truy cập gửi biểu mẫu.

HTTP sẽ cung cấp các phương thức riêng biệt để gửi dữ liệu biểu mẫu thông qua kết nối với máy chủ. Các phương thức phổ biến nhất là `get` và `post`; các phương thức này sẽ được chỉ định trong thuộc tính `method` (phương thức) của thẻ `form` (biểu mẫu):

```
<form action="receive_form.php" method="get">
```

Hoặc:

```
<form action="receive_form.php" method="post">
```

Trong phương thức `get`, dữ liệu biểu mẫu sẽ được mã hóa trực tiếp trong URL yêu cầu. Khi khách truy cập gửi biểu mẫu, trình duyệt sẽ tải URL được xác định trong thuộc tính `action` với các trường biểu mẫu được nối với nó.

Phương thức `get` được ưu tiên cho lượng dữ liệu nhỏ, chẳng hạn như biểu mẫu liên hệ đơn giản. Tuy nhiên, URL không thể vượt quá vài nghìn ký tự; vì vậy, ta nên sử dụng phương thức `post` khi biểu mẫu chứa các trường lớn hoặc không phải là văn bản ví(dụ như là hình ảnh).

Phương thức `post` sẽ khiến trình duyệt gửi dữ liệu biểu mẫu trong phần nội dung của yêu cầu HTTP. Mặc dù là cần thiết đối với dữ liệu nhị phân vượt quá giới hạn kích thước của URL nhưng phương thức `post` sẽ gia tăng tải lượng không cần thiết vào việc kết nối khi được sử dụng ở dạng các biểu mẫu văn bản đơn giản; vì vậy, phương thức `get` thường được ưa chuộng hơn trong những trường hợp như vậy.

Phương thức đã chọn sẽ không ảnh hưởng đến cách khách truy cập tương tác với biểu mẫu. Các phương thức `get` và `post` được xử lý theo những cách khác nhau bởi tệp lệnh bên phía máy chủ nhận biểu mẫu.

Khi sử dụng phương thức `post`, ta cũng có thể thay đổi loại MIME của nội dung biểu mẫu bằng thuộc tính biểu mẫu `enctype` (mã hoá). Điều này sẽ ảnh hưởng đến cách các trường và giá trị biểu mẫu được xếp chồng lên nhau trong giao tiếp HTTP với máy chủ. Giá trị mặc định cho `enctype` là `application/x-www-form-urlencoded`, tương tự như định dạng được sử dụng trong phương thức `get`. Nếu biểu mẫu chứa các trường đầu vào thuộc loại `file` thì ta nên sử dụng mã hóa `multipart/form-data` để thay thế.

Bài tập Hướng dẫn

1. Cách đúng đắn nhất để liên kết thẻ `<label>` với thẻ `<input>` là gì?

2. Loại phần tử đầu vào nào sẽ cung cấp thanh trượt điều khiển để chọn một giá trị số?

3. Mục đích của thuộc tính biểu mẫu `placeholder` là gì?

4. Làm thế nào để khiến lựa chọn thứ hai trong phần điều khiển select được chọn mặc định?

Bài tập Mở rộng

1. Làm cách nào để có thể thay đổi đầu vào tệp để khiến nó chỉ chấp nhận các tệp PDF?

2. Làm cách nào để có thể thông báo cho khách truy cập về những trường bắt buộc trong biểu mẫu?

3. Hãy tập hợp ba đoạn mã trong bài học này ở một dạng duy nhất. Hãy đảm bảo không sử dụng cùng một thuộc tính `name` hoặc `id` trong nhiều phần kiểm soát biểu mẫu.

Tóm tắt

Bài học này đã nói về cách tạo các biểu mẫu HTML đơn giản để gửi dữ liệu trở lại máy chủ. Ở phía máy khách, các biểu mẫu HTML bao gồm các phần tử HTML tiêu chuẩn được kết hợp để xây dựng các giao diện tùy chỉnh. Hơn nữa, các biểu mẫu phải được cấu hình để có thể giao tiếp đúng cách với máy chủ. Bài học đã nhắc đến các khái niệm và quy trình sau:

- Thẻ `<form>` và cấu trúc biểu mẫu cơ bản.
- Các yếu tố đầu vào cơ bản và đặc biệt.
- Vai trò của các thẻ đặc biệt như `<label>`, `<fieldset>` và `<legend>`.
- Các nút và hành động của Biểu mẫu.

Đáp án Bài tập Hướng dẫn

1. Cách đúng đắn nhất để liên kết thẻ `<label>` với thẻ `<input>` là gì?

   Thuộc tính `for` của thẻ `<label>` phải chứa id của thẻ `<input>` tương ứng.

2. Loại phần tử đầu vào nào sẽ cung cấp thanh trượt điều khiển để chọn một giá trị số?

   Loại đầu vào `range` (phạm vi).

3. Mục đích của thuộc tính biểu mẫu `placeholder` là gì?

   Thuộc tính `placeholder` chứa một ví dụ về đầu vào có thể nhìn thấy của khách truy cập khi trường đầu vào tương ứng trống.

4. Làm thế nào để khiến lựa chọn thứ hai trong phần điều khiển select được chọn mặc định?

   Phần tử `option` thứ hai phải có thuộc tính `selected`.

Đáp án Bài tập Mở rộng

1. Làm cách nào để có thể thay đổi đầu vào tệp để khiến nó chỉ chấp nhận các tệp PDF?

   Thuộc tính `accept` của phần tử đầu vào phải được đặt thành `application/pdf`.

2. Làm cách nào để có thể thông báo cho khách truy cập về những trường bắt buộc trong biểu mẫu?

   Thông thường, các trường bắt buộc sẽ được đánh dấu hoa thị (`*`) với một ghi chú ngắn gọn như "Các trường được đánh dấu * là bắt buộc" được đặt gần biểu mẫu.

3. Hãy tập hợp ba đoạn mã trong bài học này ở một dạng duy nhất. Hãy đảm bảo không sử dụng cùng một thuộc tính `name` hoặc `id` trong nhiều phần kiểm soát biểu mẫu.

```
<form action="receive_form.php" method="get">

<h2>Personal Information</h2>

<label for="fullname">Full name:</label>
<p><input type="text" name="fullname" id="fullname"></p>

<p>
  <label for="date">Date:</label>
  <input type="date" name="date" id="date">
</p>

<p>Favorite Browser:</p>

<p>
  <input type="checkbox" id="browser-firefox" name="browser" value="firefox" checked>
  <label for="browser-firefox">Mozilla Firefox</label>
</p>

<p>
  <input type="checkbox" id="browser-chrome" name="browser" value="chrome" checked>
  <label for="browser-chrome">Google Chrome</label>
</p>

<p>
  <input type="checkbox" id="browser-opera" name="browser" value="opera">
  <label for="browser-opera">Opera</label>
</p>
```

```
<p>
  <input type="checkbox" id="browser-edge" name="browser" value="edge">
  <label for="browser-edge">Microsoft Edge</label>
</p>

<p><input type="reset" value="Clear form"></p>
<p><input type="submit" value="Submit form"></p>

</form>
```

Linux Professional Institute

Chủ đề 033: Tạo kiểu Nội dung CSS

033.1 Khái niệm cơ bản về CSS

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 033.1

Khối lượng

1

Các lĩnh vực kiến thức chính

- Nhúng CSS vào tài liệu HTML
- Hiểu về cú pháp CSS
- Thêm chú thích vào CSS
- Nhận thức về các tính năng và yêu cầu trợ năng

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Thuộc tính kiểu và loại trong HTML (text/css)
- `<style>`
- `<link>`, bao gồm các thuộc tính `rel` (stylesheet), `type` (text/css) and `src`
- `;`
- `/,/`

033.1 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	033 Tạo kiểu Nội dung CSS
Mục tiêu:	033.1 Khái niệm cơ bản về CSS
Bài:	1 trên 1

Giới thiệu

Tất cả các trình duyệt web đều hiển thị các trang HTML bằng cách sử dụng các quy tắc trình bày mặc định mang tính thực tế và đơn giản nhưng lại không được bắt mắt. Bản thân HTML có cung cấp một số tính năng để viết các trang có nội dung phức tạp hơn dựa trên các kiểu trải nghiệm của người dùng hiện đại. Sau khi viết các trang HTML đơn giản, có lẽ bạn cũng đã nhận ra rằng chúng không so sánh được với các trang được thiết kế rất đẹp mắt trên Internet. Điều này là do trong HTML hiện đại, mã đánh dấu dành cho cấu trúc và chức năng của các phần tử trong tài liệu (tức *nội dung ngữ nghĩa*) được tách biệt hoàn toàn với các quy tắc xác định giao diện của các phần tử (tức *phần trình bày*). Các quy tắc trình bày được viết bằng một ngôn ngữ khác gọi là *Tệp định kiểu Xếp tầng* (Cascading Style Sheets - CSS). Nó cho phép bạn thay đổi gần như tất cả các khía cạnh trực quan của tài liệu như phông chữ, màu sắc và vị trí của các thành phần trong cả trang.

Trong hầu hết các trường hợp, các tài liệu HTML sẽ không được thiết kế để hiển thị theo một bố cục cố định như tệp PDF. Thay vào đó, các trang web dựa trên HTML có thể sẽ được hiển thị trên nhiều kích cỡ màn hình khác nhau hoặc thậm chí là được in ra. CSS cung cấp các tùy chọn có thể điều chỉnh để đảm bảo rằng trang web sẽ được hiển thị đúng như mong muốn và được điều chỉnh cho phù hợp với thiết bị hoặc ứng dụng mở trang web đó.

Áp dụng các Kiểu

Có nhiều cách để đưa CSS vào tài liệu HTML: viết CSS trực tiếp vào thẻ của phần tử trong một phần riêng của mã nguồn trang hoặc trong một tệp bên ngoài để trang HTML tham chiếu.

Thuộc tính `style`

Cách đơn giản nhất để sửa đổi kiểu của một phần tử cụ thể là viết trực tiếp vào thẻ phần tử bằng cách sử dụng thuộc tính `style` (kiểu). Tất cả các phần tử HTML hiển thị đều có thuộc tính `style` với giá trị có thể là một hoặc nhiều quy tắc CSS hay còn được gọi là *đặc tính*. Hãy bắt đầu với một ví dụ đơn giản là một phần tử đoạn văn bản:

```
<p>My stylized paragraph</p>
```

Cú pháp cơ bản của đặc tính CSS tùy chỉnh là `property: value`, trong đó `property` là một khía cạnh cụ thể mà bạn muốn thay đổi trong phần tử và `value` sẽ xác định phép thay thế cho tùy chọn mặc định do trình duyệt tạo. Một số đặc tính có thể áp dụng cho tất cả các phần tử và một số đặc tính lại chỉ áp dụng được cho một số phần tử cụ thể. Tương tự như vậy, ta sẽ có những giá trị thích hợp được sử dụng trong mỗi đặc tính.

Ví dụ: để thay đổi màu của đoạn văn bản đơn thuần, chúng ta sẽ sử dụng đặc tính `color` (màu). Đặc tính `color` là đặc tính màu *tiền cảnh*, tức màu của các chữ cái trong đoạn văn bản. Bản thân yếu tố màu sắc đã nằm trong phần giá trị của quy tắc và nó có thể được chỉ định ở nhiều định dạng khác nhau, bao gồm các tên đơn giản như `red`, `green`, `blue`, `yellow`, v.v. Do đó, để đổi màu chữ của đoạn văn thành `purple` (tím), hãy thêm đặc tính tùy chỉnh `color: purple` vào thuộc tính `style` của phần tử:

```
<p style="color: purple">My first stylized paragraph</p>
```

Các đặc tính tùy chỉnh khác có thể nằm trong cùng một đặc tính `style` nhưng chúng phải được phân tách bằng dấu chấm phẩy. Ví dụ: nếu bạn muốn làm cho kích thước của phông chữ lớn hơn, hãy thêm `font-size:larger` vào đặc tính `style`:

```
<p style="color: purple; font-size: larger">My first stylized paragraph</p>
```

NOTE Không cần thiết phải thêm khoảng trắng xung quanh dấu hai chấm và dấu chấm phẩy, nhưng chúng có thể giúp ta đọc mã CSS một cách dễ dàng hơn.

Để xem kết quả của những thay đổi này, hãy lưu HTML sau vào một tệp rồi mở tệp đó trong trình

duyệt web (kết quả sẽ được hiển thị trong [img-033-1-fig01]):

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>CSS Basics</title>
</head>
<body>

<p style="color: purple; font-size: larger">My first stylized paragraph</p>

<p style="color: green; font-size: larger">My second stylized paragraph</p>

</body>
</html>
```

Một thay đổi giao diện rất đơn giản sử dụng CSS. image::./images/fig01.png[]

Bạn có thể tưởng tượng rằng mỗi phần tử trong trang là một hình chữ nhật hoặc một chiếc hộp có các đặc tính hình học và cách trang trí mà bạn có thể thay đổi bằng CSS. Cơ chế kết xuất sử dụng hai khái niệm tiêu chuẩn cơ bản cho việc sắp xếp phần tử: sắp xếp theo *khối* và sắp xếp *nội tuyến*. Các phần tử khối chiếm toàn bộ không gian theo chiều ngang của phần tử mẹ và sẽ được đặt tuần tự từ trên xuống dưới. Chiều cao của chúng (tức *chiều dọc* của chúng, đừng nhầm lẫn với vị trí *trên cùng* của chúng trong trang) thường phụ thuộc vào lượng nội dung mà chúng có. Các phần tử nội tuyến sẽ tuân theo quy tắc từ trái sang phải tương tự như các ngôn ngữ viết của phương Tây: các phần tử được đặt theo chiều ngang, từ trái sang phải cho đến khi không còn khoảng trống ở phía bên phải, sau đó phần tử sẽ tiếp tục trên một dòng mới ngay bên dưới dòng hiện tại. Các phần tử như `p`, `div` và `section` sẽ được đặt dưới dạng khối theo mặc định, trong khi các phần tử như `span`, `em`, `a` và các chữ cái đơn sẽ được đặt ở trong dòng (nội tuyến). Các phương thức sắp xếp cơ bản này có thể được sửa đổi cơ bản theo các quy tắc CSS.

Hình chữ nhật tương ứng với phần tử `p` trong nội dung của tài liệu HTML mẫu sẽ hiển thị nếu bạn thêm đặc tính `background-color` vào quy tắc (Figure 31):

```
<p style="color: purple; font-size: larger; background-color: silver">My first stylized
paragraph</p>

<p style="color: green; font-size: larger; background-color: silver">My second stylized
paragraph</p>
```

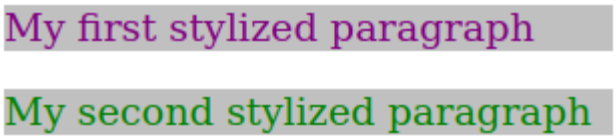

Figure 31. Hình chữ nhật tương ứng với các đoạn văn.

Khi thêm các đặc tính tùy chỉnh CSS mới vào thuộc tính `style`, ta sẽ thấy được rằng toàn bộ đoạn mã đã bắt đầu trở nên lộn xộn. Việc viết quá nhiều quy tắc CSS trong thuộc tính `style` sẽ làm suy yếu sự tách biệt giữa cấu trúc (HTML) và phần trình bày (CSS). Hơn nữa, bạn sẽ sớm nhận ra rằng có nhiều phong cách có thể được áp dụng cho các phần tử khác nhau và việc lặp lại chúng trong mọi phần tử sẽ không phải là một ý hay.

Quy tắc CSS

Thay vì tạo kiểu trực tiếp cho các phần tử trong thuộc tính `style` của chúng, việc thông báo cho trình duyệt về toàn bộ tập hợp các phần tử áp dụng kiểu tùy chỉnh sẽ thực tế hơn nhiều. Chúng ta có thể làm điều đó bằng cách thêm *bộ chọn* vào các đặc tính tùy chỉnh, khớp các phần tử theo loại, hạng, ID duy nhất, vị trí tương đối, v.v. Sự kết hợp giữa *bộ chọn* và các đặc tính tùy chỉnh tương ứng — hay còn được gọi là *đặc tính khai báo* — được gọi là *Quy tắc CSS*. Cú pháp cơ bản của quy tắc CSS là `selector { property: value }`. Như trong thuộc tính `style`, các đặc tính được phân tách bằng dấu chấm phẩy có thể được nhóm lại với nhau (ví dụ như `p { color: purple; font-size: larger }`). Quy tắc này khớp với mọi thành phần `p` trong trang và cũng áp dụng các đặc tính `color` và `font-size` tùy chỉnh.

Quy tắc CSS cho một phần tử mẹ sẽ tự động khớp với tất cả các phần tử con của nó. Điều này có nghĩa là chúng ta có thể áp dụng các đặc tính tùy chỉnh cho toàn bộ phần văn bản trong trang (bất kể văn bản đó ở bên trong hay bên ngoài thẻ `<p>`) bằng cách sử dụng bộ chọn `body` để thay thế: `body { color: purple; font-size: larger }`. Chiến lược này sẽ giải phóng chúng ta khỏi việc viết lại cùng một quy tắc cho tất cả các phần tử con của nó, nhưng có thể chúng ta sẽ phải phải viết các quy tắc bổ sung để "huỷ bỏ" hoặc sửa đổi các quy tắc kế thừa.

Thẻ `style`

Thẻ `<style>` cho phép chúng ta viết các quy tắc CSS bên trong trang HTML mà mình muốn thiết kế. Nó cho phép trình duyệt phân biệt mã CSS với mã HTML. Thẻ `<style>` sẽ nằm trong phần `head` của tài liệu:

```
<!DOCTYPE html>
<html>
<head>
```

```
  <meta charset="utf-8" />
  <title>CSS Basics</title>

  <style type="text/css">

  p { color: purple; font-size: larger }

  </style>

</head>
<body>

<p>My first stylized paragraph</p>

<p>My second stylized paragraph</p>

</body>
</html>
```

Thuộc tính `type` sẽ cho trình duyệt biết loại nội dung bên trong thẻ `<style>`, tức là *loại MIME* của nó vì mọi trình duyệt hỗ trợ CSS đều giả định rằng kiểu của thẻ `<style>` là `text/css` theo mặc định (bao gồm cả thuộc tính `type` là tùy chọn). Ngoài ra còn có thuộc tính `media` cho biết phương tiện — ví dụ như màn hình máy tính hoặc bản in — áp dụng các quy tắc CSS trong thẻ `<style>`. Theo mặc định, các quy tắc CSS sẽ áp dụng cho bất kỳ phương tiện nào nơi tài liệu được kết xuất.

Như trong mã HTML, việc ngắt dòng và lùi đầu dòng trong mã sẽ không thay đổi cách trình duyệt diễn giải các quy tắc CSS. Đoạn:

```
<style type="text/css">

p { color: purple; font-size: larger }

</style>
```

sẽ cho ra kết quả giống y hệt như đoạn:

```
<style type="text/css">

p {
  color: purple;
  font-size: larger;
}
```

```
</style>
```

Các byte bổ sung được sử dụng bởi khoảng trắng và dấu ngắt dòng sẽ không tạo ra khác biệt quá lớn trong kích thước cuối cùng của tài liệu và sẽ không có tác động đáng kể đến thời gian tải trang; vì vậy, việc lựa chọn bố cục là tuỳ theo sở thích của từng cá nhân. Hãy lưu ý dấu chấm phẩy đứng sau phần khai báo cuối cùng (`font-size: larger;`) của quy tắc CSS. Dấu chấm phẩy không phải là bắt buộc, nhưng có nó ở đó sẽ giúp ta dễ dàng thêm một phần khai báo khác vào dòng tiếp theo mà không lo thiếu dấu chấm phẩy.

Việc có các phần khai báo riêng trong từng dòng cũng rất hữu ích khi bạn cần chú giải một khai báo nào đó. Ví dụ: bất cứ khi nào bạn muốn tạm thời vô hiệu hóa một khai báo vì lý do khắc phục sự cố, bạn có thể đính kèm `/*` và `*/` vào dòng tương ứng:

```
p {
  color: purple;
  /*
  font-size: larger;
  */
}
```

hoặc:

```
p {
  color: purple;
  /* font-size: larger; */
}
```

Khi được viết như thế này, phần khai báo `font-size: larger` sẽ bị trình duyệt bỏ qua. Hãy cẩn thận với việc mở và đóng chú thích đúng cách; nếu không, trình duyệt có thể sẽ không hiểu được các quy tắc.

Chú thích cũng khá là hữu ích trong việc viết lời nhắc về các quy tắc:

```
/* Texts inside <p> are purple and larger */
p {
  color: purple;
  font-size: larger;
}
```

Lời nhắc giống như trong ví dụ có thể là không cần thiết lắm trong biểu định kiểu có chứa một số quy tắc nhỏ, nhưng chúng lại rất hữu ích trong việc điều hướng biểu định kiểu với hàng trăm quy tắc hoặc nhiều hơn.

Mặc dù thuộc tính `style` và thẻ `<style>` đều đủ để thử nghiệm các kiểu tùy chỉnh và hữu ích trong một số tình huống cụ thể nhưng chúng lại không được sử dụng phổ biến. Thay vào đó, các quy tắc CSS thường được giữ trong một tệp riêng biệt có thể được tham chiếu từ tài liệu HTML.

CSS trong Tệp bên ngoài

Một phương pháp được ưa chuộng để liên kết các định nghĩa CSS với tài liệu HTML là lưu trữ CSS trong một tệp riêng biệt. Phương pháp này có hai ưu điểm chính so với các phương pháp trước:

- Các quy tắc tạo kiểu giống nhau có thể được chia sẻ giữa các tài liệu riêng biệt.
- Tệp CSS thường được lưu trong bộ nhớ đệm của trình duyệt giúp cải thiện thời gian tải trong tương lai.

Các tệp CSS có phần mở rộng `.css` và, cũng giống như các tệp HTML, chúng có thể được chỉnh sửa bởi bất kỳ trình soạn thảo văn bản thuần túy nào. Khác với các tệp HTML, các tệp CSS không có cấu trúc cấp cao nhất được xây dựng bằng các thẻ phân cấp như `<head>` hoặc `<body>`. Thay vào đó, tệp CSS chỉ là một danh sách các quy tắc được trình duyệt xử lý theo tuần tự. Thay vào đó, các quy tắc tương tự được viết bên trong thẻ `<style>` có thể được đặt trong một tệp CSS.

Mối liên hệ giữa tài liệu HTML và các quy tắc CSS được lưu trữ trong một tệp sẽ chỉ được xác định trong tài liệu HTML. Đối với tệp CSS, việc các phần tử phù hợp với quy tắc của nó có tồn tại hay không không quan trọng; vì vậy, ta không cần liệt kê các tài liệu HTML mà nó được liên kết đến trong tệp CSS. Về phía HTML, mọi biểu định kiểu được liên kết sẽ được áp dụng cho tài liệu cũng giống như các quy tắc được viết trong thẻ `<style>`.

Thẻ HTML `<link>` sẽ xác định biểu định kiểu bên ngoài được sử dụng trong tài liệu hiện tại và sẽ nằm trong phần `head` của tài liệu HTML:

```
<head>
  <meta charset="utf-8" />
  <title>CSS Basics</title>

  <link href="style.css" rel="stylesheet">

</head>
```

Giờ đây, bạn có thể đặt quy tắc cho phần tử `p` mà chúng ta đã sử dụng trước đó vào tệp `style.css`

và kết quả mà khách truy cập vào trang web nhìn thấy sẽ giống nhau. Nếu tệp CSS không nằm trong cùng thư mục với tài liệu HTML, hãy chỉ định đường dẫn đầy đủ hoặc tương đối của nó trong thuộc tính `href`. Thẻ `<link>` có thể liên kết nhiều loại tài nguyên bên ngoài khác nhau; vì vậy, điều quan trọng là thiết lập mối quan hệ của tài nguyên bên ngoài với tài liệu hiện tại. Đối với các tệp CSS bên ngoài, mối quan hệ sẽ được xác định trong `rel="stylesheet"`.

Thuộc tính `media` có thể được sử dụng theo cách tương tự như đối với thẻ `<style>`: nó cho biết về phương tiện như màn hình máy tính hoặc bản in; các phương tiện này sẽ áp dụng các quy tắc trong tệp bên ngoài.

CSS hoàn toàn có thể thay đổi một phần tử, nhưng điều quan trọng vẫn là phải sử dụng phần tử thích hợp cho các thành phần của trang. Nếu không, các công nghệ hỗ trợ như trình đọc màn hình có thể sẽ không xác định được đúng các phần của trang.

Bài tập Hướng dẫn

1. Những phương pháp nào có thể được sử dụng để thay đổi giao diện của các phần tử HTML sử dụng CSS?

2. Tại sao không nên sử dụng thuộc tính `style` của thẻ `<p>` nếu có các đoạn văn bản giống nhau?

3. Chính sách sắp đặt mặc định để đặt phần tử `div` là gì?

4. Thuộc tính nào của thẻ `<link>` sẽ cho biết vị trí của tệp CSS bên ngoài?

5. Phần nào bên trong tài liệu HTML có thể chèn phần tử `link` vào?

Bài tập Mở rộng

1. Tại sao không nên sử dụng thẻ `<div>` để xác định một từ trong một câu thông thường?

2. Điều gì sẽ xảy ra nếu bạn bắt đầu một chú thích bằng `/*` ở giữa tệp CSS nhưng lại quên đóng nó bằng `*/`?

3. Hãy viết quy tắc CSS để gạch chân tất cả các phần tử `em` của tài liệu.

4. Làm cách nào để có thể cho biết biểu định kiểu từ thẻ `<style>` hoặc `<link>` chỉ nên được sử dụng bởi bộ tổng hợp giọng nói?

Tóm tắt

Bài học này bao gồm các khái niệm cơ bản về CSS và cách tích hợp nó với HTML. Các quy tắc được viết trong biểu định kiểu CSS là phương pháp tiêu chuẩn để thay đổi giao diện của tài liệu HTML. CSS cho phép chúng ta giữ nội dung ngữ nghĩa tách biệt với các cách trình bày. Bài học này đã đi qua các khái niệm và quy trình sau:

- Cách thay đổi đặc tính CSS bằng thuộc tính `style`.
- Cú pháp quy tắc CSS cơ bản.
- Sử dụng thẻ `<style>` để nhúng các quy tắc CSS vào tài liệu.
- Sử dụng thẻ `<link>` để kết hợp các biểu định kiểu bên ngoài vào tài liệu.

Đáp án Bài tập Hướng dẫn

1. Những phương pháp nào có thể được sử dụng để thay đổi giao diện của các phần tử HTML sử dụng CSS?

   Ba phương pháp cơ bản: Viết trực tiếp trong thẻ của phần tử, trong một phần riêng của mã nguồn của trang hoặc trong một tệp bên ngoài được trang HTML tham chiếu.

2. Tại sao không nên sử dụng thuộc tính `style` của thẻ `<p>` nếu có các đoạn văn bản giống nhau?

   Việc khai báo CSS sẽ cần phải được sao chép trong các thẻ `<p>` khác. Điều này gây tốn thời gian, tăng kích thước tệp và dễ bị lỗi.

3. Chính sách sắp đặt mặc định để đặt phần tử `div` là gì?

   Phần tử `div` được coi là phần tử khối theo mặc định. Do đó, phần tử này sẽ chiếm toàn bộ không gian theo chiều ngang của phần tử mẹ và chiều cao của phần tử này sẽ phụ thuộc vào nội dung của chính nó.

4. Thuộc tính nào của thẻ `<link>` sẽ cho biết vị trí của tệp CSS bên ngoài?

   Thuộc tính `href`.

5. Phần nào bên trong tài liệu HTML có thể chèn phần tử `link` vào?

   Phần tử `link` phải nằm trong phần `head` của tài liệu HTML.

Đáp án Bài tập Mở rộng

1. Tại sao không nên sử dụng thẻ `<div>` để xác định một từ trong một câu thông thường?

   Thẻ `<div>` cung cấp một sự phân tách về ngữ nghĩa giữa hai phần riêng biệt của tài liệu và sẽ cản trở các công cụ trợ năng khi nó được sử dụng cho các thành phần văn bản trong dòng (nội tuyến).

2. Điều gì sẽ xảy ra nếu bạn bắt đầu một chú thích bằng `/*` ở giữa tệp CSS nhưng lại quên đóng nó bằng `*/`?

   Tất cả các quy tắc đứng sau chú thích sẽ bị trình duyệt bỏ qua.

3. Hãy viết quy tắc CSS để gạch chân tất cả các phần tử `em` của tài liệu.

   ```
   em { text-decoration: underline }
   ```

   hoặc

   ```
   em { text-decoration-line: underline }
   ```

4. Làm cách nào để có thể cho biết biểu định kiểu từ thẻ `<style>` hoặc `<link>` chỉ nên được sử dụng bởi bộ tổng hợp giọng nói?

   Giá trị của thuộc tính `media` phải là `speech`.

033.2 Bộ chọn CSS và Ứng dụng kiểu

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 033.2

Khối lượng

3

Các lĩnh vực kiến thức chính

- Sử dụng bộ chọn để áp dụng quy tắc CSS cho các phần tử
- Hiểu về các hạng giả trong CSS
- Hiểu về thứ tự quy tắc và quyền ưu tiên trong CSS
- Hiểu về tính kế thừa trong CSS

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `element`; `.class`; `#id`
- `a, b`; `a.class`; `a b`;
- `:hover`, `:focus`
- `!important`

Linux Professional Institute

033.2 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	033 Tạo kiểu Nội dung CSS
Mục tiêu:	033.2 Bộ Chọn trong CSS và Ứng dụng Kiểu
Bài:	1 trên 1

Giới thiệu

Khi viết một quy tắc CSS, chúng ta phải báo cho trình duyệt biết quy tắc đó áp dụng cho những phần tử nào. Chúng ta có thể làm được điều này bằng cách chỉ định *bộ chọn* - tức là một mẫu có thể khớp với một phần tử hoặc một nhóm phần tử. Bộ chọn có nhiều dạng khác nhau, có thể dựa trên tên, thuộc tính, vị trí tương đối của phần tử trong cấu trúc tài liệu hoặc sự kết hợp giữa các đặc điểm này.

Kiểu Trang tổng thể

Một trong những ưu điểm của việc sử dụng CSS là bạn sẽ không cần phải viết các quy tắc riêng lẻ cho các phần tử có cùng kiểu. Dấu hoa thị sẽ áp dụng quy tắc cho tất cả các phần tử trong trang web như được minh họa trong ví dụ sau:

```
* {
  color: purple;
  font-size: large;
}
```

Bộ chọn `*` không phải là phương pháp duy nhất để áp dụng quy tắc kiểu cho toàn bộ phần tử trong trang. Một bộ chọn mà chỉ khớp một phần tử theo tên thẻ của nó sẽ được gọi là *bộ chọn kiểu*; do đó, bất kỳ tên thẻ HTML nào (chẳng hạn như `body`, `p`, `table`, `em`, v.v.) cũng đều có thể được sử dụng làm bộ chọn. Trong CSS, kiểu của phần tử mẹ sẽ được các phần tử con của nó *kế thừa*. Vì vậy, trong thực tế, việc sử dụng bộ chọn `*` có tác dụng tương tự như việc áp dụng một quy tắc cho phần tử `body`:

```
body {
  color: purple;
  font-size: large;
}
```

Hơn nữa, tính năng xếp tầng của CSS sẽ cho phép bạn tinh chỉnh các đặc tính kế thừa của một phần tử. Bạn có thể viết quy tắc CSS chung áp dụng cho toàn bộ các phần tử trong trang, sau đó viết quy tắc cho các phần tử hoặc các tập hợp phần tử cụ thể.

Nếu một phần tử cùng khớp với hai hoặc nhiều quy tắc xung đột, trình duyệt sẽ áp dụng quy tắc từ bộ chọn cụ thể nhất. Hãy lấy các quy tắc CSS sau đây làm ví dụ:

```
body {
  color: purple;
  font-size: large;
}

li {
  font-size: small;
}
```

Trình duyệt sẽ áp dụng kiểu `color: purple` và `font-size: large` cho tất cả các phần tử bên trong phần tử `body`. Tuy nhiên, nếu có các phần tử `li` trong trang, trình duyệt sẽ thay thế kiểu `font-size: large` bằng kiểu `font-size: small` bởi bộ chọn `li` có mối quan hệ chặt chẽ với ` li` hơn bộ chọn `body`.

CSS không giới hạn số lượng bộ chọn tương đương trong cùng một biểu định kiểu; vì vậy, ta có thể có hai hoặc nhiều quy tắc sử dụng cùng một bộ chọn:

```
li {
  font-size: small;
}

li {
```

```
  font-size: x-small;
}
```

Trong trường hợp này, cả hai quy tắc đều áp dụng cho phần tử `li`. Trình duyệt không thể áp dụng các quy tắc xung đột; do đó, trình duyệt sẽ chọn quy tắc xuất hiện sau trong tệp CSS. Để tránh nhầm lẫn, chúng ta nên nhóm tất cả các thuộc tính sử dụng cùng một bộ chọn lại với nhau.

Thứ tự các quy tắc xuất hiện trong biểu định kiểu sẽ ảnh hưởng đến cách chúng được áp dụng trong tài liệu, nhưng ta có thể ghi đè lên hành vi này bằng cách sử dụng quy tắc `important` (quan trọng). Nếu vì bất kỳ lý do gì mà bạn muốn giữ hai quy tắc `li` riêng biệt, nhưng cũng muốn bắt buộc áp dụng quy tắc đầu tiên thay vì quy tắc thứ hai, hãy đánh dấu quy tắc đầu tiên là quan trọng:

```
li {
  font-size: small !important;
}

li {
  font-size: x-small;
}
```

Bạn cần thận trọng khi sử dụng các quy tắc được đánh dấu bằng `!important` vì chúng sẽ phá vỡ tầng biểu định kiểu tự nhiên và khiến việc tìm và khắc phục sự cố trong tệp CSS trở nên khó khăn hơn.

Bộ chọn Hạn chế

Chúng ta đã thấy rằng có thể thay đổi một số đặc tính kế thừa nhất định bằng cách sử dụng bộ chọn khớp với các thẻ cụ thể. Tuy nhiên, chúng ta thường sẽ cần sử dụng các kiểu riêng biệt cho các phần tử cùng loại.

Các thuộc tính của thẻ HTML có thể được kết hợp vào bộ chọn để hạn chế tập hợp các phần tử mà chúng tham chiếu. Giả sử trang HTML mà bạn đang xử lý có hai loại danh sách không có thứ tự (`<ul>`): một loại được sử dụng ở đầu trang làm menu cho các phần của trang web và loại còn lại được sử dụng cho danh sách thông thường trong phần thân văn bản:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
```

```
  <title>CSS Basics</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/articles">Articles</a></li>
  <li><a href="/about">About</a></li>
</ul>

<div id="content">

<p>The three rocky planets of the solar system are:</p>

<ul>
  <li>Mercury</li>
  <li>Venus</li>
  <li>Earth</li>
  <li>Mars</li>
</ul>

<p>The outer giant planets made most of gas are:</p>

<ul>
  <li>Jupiter</li>
  <li>Saturn</li>
  <li>Uranus</li>
  <li>Neptune</li>
</ul>

</div><!-- #content -->


<div id="footer">

<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/articles">Articles</a></li>
  <li><a href="/about">About</a></li>
</ul>

</div><!-- #footer -->
```

```
</body>
</html>
```

Theo mặc định, mỗi danh mục sẽ có một hình tròn màu đen ở bên trái của nó. Có thể bạn sẽ muốn xóa các hình tròn này khỏi danh sách menu trên cùng trong khi để lại các hình tròn trong danh sách còn lại. Tuy nhiên, ta không thể chỉ đơn giản sử dụng bộ chọn `li` vì làm như vậy cũng sẽ loại bỏ các hình tròn trong các danh sách khác bên trong nội dung văn bản. Chúng ta sẽ cần một cách để yêu cầu trình duyệt chỉ sửa đổi các danh mục trong một danh sách nhất định chứ không phải danh sách khác.

Có một số cách để viết các bộ chọn khớp với các phần tử cụ thể trong trang. Như đã đề cập trước đó, trước tiên, chúng ta sẽ xem thử cách sử dụng các thuộc tính của phần tử để thực hiện điều này. Riêng đối với ví dụ này, chúng ta có thể sử dụng thuộc tính `id` để chỉ định duy nhất danh sách ở trên cùng.

Thuộc tính `id` sẽ gán một mã định danh duy nhất cho phần tử tương ứng mà chúng ta có thể sử dụng làm bộ chọn của quy tắc CSS. Trước khi viết quy tắc CSS, hãy chỉnh sửa tệp HTML mẫu và thêm `id="topmenu"` vào phần tử `ul` được sử dụng cho menu trên cùng:

```
<ul id="topmenu">
  <li>Home</li>
  <li>Articles</li>
  <li>About</li>
</ul>
```

Đã có một phần tử `link` trong phần `head` của tài liệu HTML trỏ đến tệp biểu định kiểu `style.css` trong cùng một thư mục. Vì vậy, chúng ta có thể thêm các quy tắc CSS sau vào đó:

```
ul#topmenu {
  list-style-type: none
}
```

Dấu thăng được sử dụng trong bộ chọn theo sau một phần tử là để chỉ định một ID (không có dấu cách ngăn cách chúng). Tên thẻ ở bên trái của dấu thăng là tùy chọn vì sẽ không có phần tử nào khác có cùng ID. Do đó, bộ chọn có thể được viết dưới dạng `#topmenu`.

Mặc dù đặc tính `list-style-type` không phải là đặc tính trực tiếp của phần tử `ul`, nhưng các thuộc tính CSS của phần tử mẹ sẽ được kế thừa bởi các phần tử con của nó. Vì vậy, kiểu được gán cho phần tử `ul` sẽ được kế thừa bởi các phần tử `li` con của nó.

Không phải tất cả các phần tử đều sẽ có ID riêng để có thể được chọn. Chỉ một vài phần tử bố cục chính trong một trang được coi là có ID. Ví dụ: lấy danh sách các hành tinh được sử dụng trong mã mẫu. Mặc dù có thể gán các ID duy nhất cho từng phần tử lặp lại riêng lẻ như thế này, nhưng phương pháp này không thực tế đối với các trang dài có nhiều nội dung. Thay vào đó, chúng ta có thể sử dụng ID của phần tử `div` gốc làm bộ chọn để thay đổi đặc tính của các phần tử bên trong nó.

Tuy nhiên, phần tử `div` lại không liên quan trực tiếp đến danh sách HTML. Vì vậy, việc thêm thuộc tính `list-style-type` vào phần tử này sẽ không ảnh hưởng đến phần tử con của nó. Do đó, để thay đổi hình tròn màu đen trong danh sách bên trong nội dung `div` thành hình tròn rỗng, chúng ta nên sử dụng bộ chọn *hậu duệ*:

```
#topmenu {
  list-style-type: none
}

#content ul {
  list-style-type: circle
}
```

Bộ chọn `#content ul` được gọi là bộ chọn hậu duệ vì nó chỉ khớp với các phần tử `ul` là phần tử con của phần tử có ID là `content`. Chúng ta có thể sử dụng nhiều cấp độ hậu duệ nếu cần. Ví dụ: sử dụng `#content ul li` sẽ chỉ khớp với các phần tử `li` là phần tử con của phần tử `ul`, `ul` cũng lại là phần tử con của phần tử có ID là `content`. Nhưng trong ví dụ này, bộ chọn `#content ul li` sẽ có tác dụng tương tự như khi sử dụng `#content ul` bởi vì các phần tử `li` sẽ kế thừa các đặc tính CSS được đặt cho phần tử mẹ của chúng là `ul`. Bộ chọn hậu duệ là một kỹ thuật thiết yếu khi bố cục trang ngày càng trở nên phức tạp hơn.

Giả sử bây giờ ta muốn thay đổi đặc tính `font-style` (kiểu chữ) của các danh mục trong danh sách `topmenu` và trong danh sách trong phần *footer div* để biến chúng thành chữ nghiêng đậm. Ta không thể chỉ viết một quy tắc CSS bằng cách sử dụng `ul` làm bộ chọn bởi vì nó cũng sẽ thay đổi các danh mục trong *content div*. Cho đến nay, chúng ta đã thay đổi các đặc tính CSS bằng cách sử dụng mỗi lần một bộ chọn. Phương pháp này cũng có thể được sử dụng cho tác vụ trước mắt:

```
#topmenu {
  font-style: oblique
}

#footer ul {
  font-style: oblique
```

```
}
```

Tuy nhiên, các bộ chọn riêng biệt không phải là cách duy nhất để làm điều đó. CSS cho phép chúng ta nhóm các bộ chọn có chung một hoặc nhiều kiểu bằng cách sử dụng danh sách các bộ chọn được phân tách bằng dấu phẩy:

```
#topmenu, #footer ul {
  font-style: oblique
}
```

Việc nhóm các bộ chọn sẽ giúp ta giảm thiểu việc viết đi viết lại các kiểu trùng lặp. Hơn nữa, chúng ta có thể sẽ muốn thay đổi đặc tính một lần nữa trong tương lai và sẽ không thể nhớ được tất cả các vị trí khác nhau của đặc tính đó.

Hạng

Nếu không muốn lo lắng quá nhiều về hệ thống phân cấp phần tử, ta chỉ cần thêm `class` (hạng) vào tập hợp các phần tử mà mình muốn tùy chỉnh. Hạng cũng tương tự như ID, nhưng thay vì chỉ xác định một phần tử duy nhất trong trang, chúng có thể xác định một nhóm phần tử có chung đặc điểm.

Lấy ví dụ như trang HTML mẫu mà chúng ta đang xử lý. Trong các trang thực tế, không chắc chúng ta có thể tìm thấy các cấu trúc đơn giản như vậy. Vì vậy, sẽ thực tế hơn nếu ta chọn một phần tử chỉ bằng cách sử dụng các hạng hoặc kết hợp các hạng và hậu duệ. Để áp dụng thuộc tính `font-style: oblique` cho các danh sách menu bằng cách sử dụng hạng, trước tiên, chúng ta cần thêm thuộc tính `class` vào các thành phần trong tệp HTML. Chúng ta sẽ làm điều này với menu trên cùng đầu tiên:

```
<ul id="topmenu" class="menu">
  <li><a href="/">Home</a></li>
  <li><a href="/articles">Articles</a></li>
  <li><a href="/about">About</a></li>
</ul>
```

Và sau đó trong menu chân trang:

```
<div id="footer">

<ul class="menu">
  <li><a href="/">Home</a></li>
```

```
  <li><a href="/articles">Articles</a></li>
  <li><a href="/about">About</a></li>
</ul>

</div><!-- #footer -->
```

Cùng với đó, chúng ta có thể thay thế nhóm bộ chọn `#topmenu, #footer ul` bằng bộ chọn dựa trên hạng `.menu`:

```
.menu {
  font-style: oblique
}
```

Như với bộ chọn dựa trên ID, việc thêm tên thẻ vào bên trái dấu chấm trong bộ chọn dựa trên hạng là tùy chọn. Tuy nhiên, không giống như ID, cùng một hạng nên được sử dụng trong nhiều phần tử và các phần tử đó thậm chí không cần phải là cùng loại với nhau. Do đó, nếu hạng `menu` được chia sẻ giữa các loại phần tử khác nhau thì việc sử dụng bộ chọn `ul.menu` sẽ chỉ khớp với các phần tử `ul` có hạng `menu`. Thay vào đó, việc sử dụng `.menu` làm bộ chọn sẽ khớp với bất kỳ phần tử nào có hạng `menu`, bất kể nó thuộc loại gì.

Hơn nữa, các phần tử có thể được liên kết với nhiều hạng khác nhau. Chúng ta có thể phân biệt giữa menu trên cùng và menu dưới cùng bằng cách thêm một hạng bổ sung cho mỗi menu:

```
<ul id="topmenu" class="menu top">
```

Và trong menu chân trang:

```
<ul class="menu footer">
```

Khi thuộc tính `class` có nhiều hơn một hạng, chúng phải được phân tách bằng dấu cách. Bây giờ, ngoài quy tắc CSS được chia sẻ giữa các phần tử của hạng `menu`, chúng ta có thể xử lý menu trên cùng và menu chân trang bằng cách sử dụng các hạng tương ứng của chúng:

```
.menu {
  font-style: oblique
}

.menu.top {
  font-size: large
```

```
}

.menu.footer {
  font-size: small
}
```

Hãy lưu ý rằng cách viết `.menu.top` sẽ khác với `.menu .top` (có dấu cách giữa các từ). Bộ chọn đầu tiên sẽ khớp với các phần tử có cả hai hạng `menu` và `top`, trong khi bộ chọn thứ hai sẽ khớp với các phần tử có hạng `top` và một phần tử mẹ có hạng `menu`.

Bộ chọn Đặc biệt

Bộ chọn CSS cũng có thể khớp với trạng thái động của các phần tử. Các bộ chọn này đặc biệt hữu ích cho các phần tử tương tác, chẳng hạn như siêu liên kết. Có thể bạn sẽ cần sự xuất hiện của các siêu liên kết khi con trỏ chuột di chuyển qua chúng để thu hút sự chú ý của khách truy cập.

Quay lại trang mẫu của chúng ta, ta có thể xóa phần gạch chân khỏi các liên kết trong menu trên cùng và chỉ hiển thị một dòng duy nhất khi con trỏ chuột di chuyển qua liên kết tương ứng. Để làm được điều này, trước tiên, chúng ta phải viết một quy tắc để xóa phần gạch chân khỏi các liên kết trong menu trên cùng:

```
.menu.top a {
  text-decoration: none
}
```

Sau đó, chúng ta sử dụng hạng giả `:hover` (di qua) trên cùng các phần tử đó để tạo quy tắc CSS sẽ chỉ áp dụng khi con trỏ chuột di chuyển đến phần tử tương ứng:

```
.menu.top a:hover {
  text-decoration: underline
}
```

Bộ chọn hạng giả `:hover` sẽ chấp nhận tất cả các đặc tính CSS của các quy tắc CSS thông thường. Các hạng giả khác bao gồm `:visited` (đã truy cập) - phù hợp với các siêu liên kết đã được truy cập và `:focus` (tập trung) - phù hợp với các phần tử biểu mẫu đã vào tiêu điểm.

Bài tập Hướng dẫn

1. Giả sử một trang HTML có một biểu định kiểu được liên kết với nó có chứa hai quy tắc sau:

```
p {
  color: green;
}

p {
  color: red;
}
```

   Trình duyệt sẽ áp dụng màu gì cho văn bản bên trong các phần tử `p`?

2. Sự khác biệt giữa việc sử dụng bộ chọn ID `div#main` và `#main` là gì?

3. Bộ chọn nào sẽ phù hợp với tất cả các phần tử `p` được sử dụng bên trong `div` với thuộc tính ID `#main`?

4. Sự khác biệt giữa việc sử dụng bộ chọn hạng `p.highlight` và `.highlight` là gì?

Bài tập Mở rộng

1. Hãy viết một quy tắc CSS duy nhất để thay đổi đặc tính `font-style` thành `oblique`. Quy tắc chỉ được khớp với các phần tử `a` bên trong `<div id="sidebar"></div>` hoặc `<ul class="links"></ul>`.

2. Giả sử bạn muốn thay đổi kiểu của các phần tử có thuộc tính `class` được đặt thành `article reference` như trong `<p class="article reference">`. Tuy nhiên, bộ chọn `.article .reference` dường như lại không làm thay đổi diện mạo của chúng. Tại sao bộ chọn lại không khớp với các phần tử như được mong đợi?

3. Hãy viết một quy tắc CSS để thay đổi đặc tính `color` của tất cả các liên kết đã truy cập trong trang thành `red`.

Tóm tắt

Bài học này đã đề cập tới cách sử dụng bộ chọn CSS và cách trình duyệt quyết định kiểu nào sẽ áp dụng cho phần tử nào. Tách biệt với phần đánh dấu HTML, CSS cung cấp nhiều bộ chọn để khớp với các phần tử riêng lẻ hoặc nhóm phần tử trong trang. Bài học đã đi qua các khái niệm và quy trình sau:

- Kiểu Trang Tổng thể và Kế thừa Kiểu.
- Tạo kiểu cho các phần tử theo loại.
- Sử dụng ID phần tử và hạng làm bộ chọn.
- Bộ chọn kết hợp.
- Sử dụng các hạng giả để thiết kế động cho các phần tử.

Đáp án Bài tập Hướng dẫn

1. Giả sử một trang HTML có một biểu định kiểu được liên kết với nó có chứa hai quy tắc sau:

```
p {
  color: green;
}

p {
  color: red;
}
```

   Trình duyệt sẽ áp dụng màu gì cho văn bản bên trong các phần tử `p`?

   Màu đỏ (`red`). Khi hai hoặc nhiều bộ chọn tương đương có thuộc tính xung đột, trình duyệt sẽ chọn bộ chọn cuối cùng.

2. Sự khác biệt giữa việc sử dụng bộ chọn ID `div#main` và `#main` là gì?

   Bộ chọn `div#main` sẽ khớp với phần tử `div` có ID `main`, trong khi bộ chọn `#main` sẽ khớp với phần tử có ID `main` bất kể nó thuộc loại gì.

3. Bộ chọn nào sẽ phù hợp với tất cả các phần tử `p` được sử dụng bên trong `div` với thuộc tính ID `#main`?

   Bộ chọn `#main p` hoặc `div#main p`.

4. Sự khác biệt giữa việc sử dụng bộ chọn hạng `p.highlight` và `.highlight` là gì?

   Bộ chọn `p.highlight` sẽ chỉ khớp với các phần tử thuộc loại `p` có hạng `highlight`, trong khi bộ chọn `.highlight` sẽ khớp với tất cả các phần tử có hạng `highlight` bất kể chúng thuộc loại gì.

Đáp án Bài tập Mở rộng

1. Hãy viết một quy tắc CSS duy nhất để thay đổi đặc tính `font-style` thành `oblique`. Quy tắc chỉ được khớp với các phần tử `a` bên trong `<div id="sidebar"></div>` hoặc `<ul class="links"></ul>`.

   ```
   #sidebar a, ul.links a {
     font-style: oblique
   }
   ```

2. Giả sử bạn muốn thay đổi kiểu của các phần tử có thuộc tính `class` được đặt thành `article reference` như trong `<p class="article reference">`. Tuy nhiên, bộ chọn `.article .reference` dường như lại không làm thay đổi diện mạo của chúng. Tại sao bộ chọn lại không khớp với các phần tử như được mong đợi?

   Bộ chọn `.article .reference` sẽ khớp với các phần tử có hạng `reference` là hậu duệ của các phần tử có hạng `article`. Để khớp các phần tử có cả hai hạng `article` và `reference`, bộ chọn phải là `.article.reference` (không có khoảng cách giữa chúng).

3. Hãy viết một quy tắc CSS để thay đổi đặc tính `color` của tất cả các liên kết đã truy cập trong trang thành `red`.

   ```
   a:visited {
     color: red;
   }
   ```

033.3 Tạo kiểu CSS

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 033.3

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu về các thuộc tính CSS cơ bản
- Hiểu về các đơn vị thường được sử dụng trong CSS

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `px`, `%`, `em`, `rem`, `vw`, `vh`
- `color`, `background`, `background-*`, `font`, `font-*`, `text-*`, `list-style`, `line-height`

033.3 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	033 Tạo kiểu Nội dung CSS
Mục tiêu:	033.3 Tạo kiểu CSS
Bài:	1 trên 1

Giới thiệu

CSS cung cấp hàng trăm các đặc tính có thể được sử dụng để sửa đổi giao diện của các phần tử HTML. Chỉ những nhà phát triển có kinh nghiệm mới nhớ hết được chúng. Tuy nhiên, việc biết về các đặc tính cơ bản nào có thể áp dụng cho bất kỳ phần tử nào, cũng như một số đặc tính dành riêng cho từng phần tử sẽ khá hữu ích. Bài học này sẽ nói về một số đặc tính phổ biến mà khả năng cao bạn sẽ phải sử dụng.

Các Đặc tính và Giá trị phổ biến trong CSS

Nhiều đặc tính CSS có cùng loại giá trị với nhau. Ví dụ: các loại màu sắc vẫn có cùng định dạng số liệu như nhau cho dù ta đang thay đổi màu phông chữ hay màu nền. Tương tự, các đơn vị có sẵn để thay đổi kích thước của phông chữ cũng được sử dụng để thay đổi độ dày của đường viền. Tuy nhiên, định dạng của giá trị không phải lúc nào cũng là duy nhất, ví dụ như màu sắc có thể được nhập ở nhiều định dạng khác nhau như chúng ta sẽ thấy tiếp theo đây.

Màu sắc

Việc thay đổi màu sắc của một phần tử có lẽ là một trong những điều đầu tiên mà các nhà phát triển sẽ học để thực hiện với CSS. Ta có thể thay đổi màu của văn bản, màu nền, màu đường viền của các phần tử, v.v.

Giá trị của một màu trong CSS có thể được viết dưới dạng *từ khóa màu* (tức tên màu) hoặc dưới dạng giá trị số liệt kê từng thành phần màu. Tất cả các tên màu phổ biến, chẳng hạn như `black`, `white`, `red`, `purple`, `green`, `yellow` và `blue` đều được chấp nhận làm từ khóa màu. Toàn bộ danh sách các từ khóa màu được CSS chấp nhận hiện có tại trang web W3C. Nhưng để kiểm soát màu sắc một cách tốt hơn, chúng ta nên sử dụng giá trị số của chúng.

Từ khóa Màu

Đầu tiên, chúng ta sẽ sử dụng từ khóa màu vì nó đơn giản hơn. Đặc tính `color` sẽ thay đổi màu của văn bản trong phần tử tương ứng. Vì vậy, để đặt tất cả văn bản trên trang thành màu tím, ta có thể viết quy tắc CSS sau:

```
* {
  color: purple;
}
```

Giá trị Màu Số

Mặc dù có lợi thế trực quan, các từ khóa màu lại không cung cấp đầy đủ các màu có thể có trong màn hình hiện đại. Các nhà thiết kế trang web thường phát triển một bảng màu sử dụng các màu tùy chỉnh bằng cách gán các giá trị cụ thể cho các thành phần màu đỏ, lục và lam.

Mỗi thành phần màu sẽ là một số nhị phân tám bit nằm trong khoảng từ 0 đến 255. Tất cả ba thành phần này phải được chỉ định trong hỗn hợp màu và thứ tự của chúng luôn là đỏ, lục, lam (RGB). Do đó, để thay đổi màu của tất cả văn bản trong trang thành màu đỏ bằng cách sử dụng ký hiệu RGB, hãy sử dụng `rgb(255,0,0)`:

```
* {
  color: rgb(255,0,0);
}
```

Một thành phần được đặt thành `0` có nghĩa là màu cơ bản tương ứng không được sử dụng trong hỗn hợp màu. Tỷ lệ phần trăm cũng có thể được sử dụng thay vì số:

```
* {
  color: rgb(100%,0%,0%);
}
```

Ký hiệu RGB hiếm khi được nhìn thấy nếu ta sử dụng ứng dụng vẽ để tạo bố cục hoặc chỉ để chọn màu. Thay vào đó, màu sắc trong CSS được biểu thị dưới dạng giá trị *thập lục phân* thường sẽ phổ biến hơn. Các thành phần màu ở dạng thập lục phân cũng nằm trong khoảng từ 0 đến 255 nhưng lại theo cách ngắn gọn hơn, bắt đầu bằng dấu thăng `#` và sử dụng độ dài hai chữ số cố định cho tất cả các thành phần. Giá trị nhỏ nhất `0` sẽ là `00` và giá trị lớn nhất `255` sẽ là `FF`. Vì vậy, màu đỏ sẽ là `#FF0000`.

TIP Nếu các chữ số trong mỗi thành phần của màu thập lục phân lặp lại, chữ số thứ hai có thể được bỏ qua. Ví dụ: màu đỏ có thể được viết bằng một chữ số cho mỗi thành phần: `#F00`.

Danh sách sau đây sẽ cho biết ký hiệu RGB và hệ thập lục phân của một số màu cơ bản:

Từ khóa Màu	Ký hiệu RGB	Giá trị Thập lục phân
`black`	`rgb(0,0,0)`	`#000000`
`white`	`rgb(255,255,255)`	`#FFFFFF`
`red`	`rgb(255,0,0)`	`#FF0000`
`purple`	`rgb(128,0,128)`	`#800080`
`green`	`rgb(0,128,0)`	`#008000`
`yellow`	`rgb(255,255,0)`	`#FFFF00`
`blue`	`rgb(0,0,255)`	`#0000FF`

Từ khóa màu và các chữ cái trong giá trị màu thập lục phân không phân biệt chữ hoa chữ thường.

Độ Mờ của Màu

Ngoài các màu đục, ta cũng có thể đổ màu cho các phần tử trang bằng các màu bán trong suốt. Độ mờ của màu có thể được đặt bằng cách sử dụng thành phần thứ tư trong giá trị màu. Không giống như các thành phần màu khác có các giá trị là các số nguyên nằm trong khoảng từ 0 đến 255, độ mờ là một phân số nằm trong khoảng từ `0` đến `1`.

Giá trị thấp nhất là `0`; giá trị này sẽ khiến cho màu trở nên hoàn toàn trong suốt, khiến ta không thể phân biệt được giữa nó và bất kỳ một màu hoàn toàn trong suốt nào khác. Giá trị cao nhất là `1`; giá trị này sẽ khiến cho màu trở nên đục hoàn toàn, giống như màu gốc nhưng không có độ trong.

Khi bạn sử dụng ký hiệu RGB, hãy chỉ định các màu có thành phần độ mờ thông qua tiền tố `rgba` thay vì `rgb`. Từ đó, để làm cho màu đỏ trở nên nửa trong suốt, hãy sử dụng `rgba(255,0,0,5)`. Ký tự `a` là viết tắt của *kênh alpha* - một thuật ngữ thường được sử dụng để chỉ định thành phần độ mờ trong biệt ngữ đồ họa kỹ thuật số.

Độ mờ cũng có thể được đặt bằng ký hiệu thập lục phân. Giống như các thành phần màu khác, độ mờ sẽ nằm trong khoảng từ `00` đến `FF`. Do đó, để làm cho màu đỏ trở nên nửa trong suốt bằng ký hiệu thập lục phân, hãy sử dụng `#FF000080`.

Nền

Màu nền của các phần tử đơn lẻ hoặc của toàn bộ trang được đặt bằng đặc tính `background-color`. Nó sẽ nhận các giá trị giống như đặc tính `color`, tức dưới dạng từ khóa hoặc sử dụng ký hiệu RGB/thập lục phân.

Tuy nhiên, nền của các phần tử HTML sẽ không bị hạn chế ở màu sắc. Với thuộc tính `background-image`, ta có thể sử dụng hình ảnh để làm nền. Các định dạng hình ảnh được chấp nhận là tất cả các định dạng thông thường được trình duyệt chấp nhận, chẳng hạn như JPEG và PNG.

Đường dẫn đến hình ảnh phải được chỉ định bằng trình chỉ định `url()`. Nếu hình ảnh bạn muốn sử dụng nằm trong cùng thư mục với tệp HTML thì chỉ cần sử dụng tên tệp của nó là đủ:

```
body {
  background-image: url("background.jpg");
}
```

Trong ví dụ này, tệp hình ảnh `background.jpg` sẽ được sử dụng làm hình nền cho toàn bộ nội dung của trang. Theo mặc định, hình nền sẽ được lặp lại nếu kích thước của nó không bao phủ toàn bộ trang bắt đầu từ góc trên cùng bên trái của khu vực tương ứng với bộ chọn của quy tắc. Hành vi này có thể được sửa đổi bằng thuộc tính `background-repeat`. Nếu bạn muốn hình nền được đặt trong khu vực của phần tử mà không bị lặp lại, hãy sử dụng giá trị `no-repeat`:

```
body {
  background-image: url("background.jpg");
  background-repeat: no-repeat;
}
```

Ta cũng có thể làm cho hình ảnh chỉ lặp lại theo chieeuf ngang (`background-repeat: repeat-x`) hoặc chỉ chiều dọc (`background-repeat: repeat-y`).

Cách đặt nền sử dụng đặc tính `background-repeat`. image::./images/background-repeat.png[width="50%"]

TIP

Hai hoặc nhiều đặc tính CSS có thể được kết hợp trong một đặc tính duy nhất được gọi là đặc tính *tốc ký* nền. Ví dụ: các đặc tính `background-image` và `background-repeat` có thể được kết hợp trong một đặc tính nền duy nhất với `background: no-repeat url("background.jpg")`.

Một hình nền cũng có thể được đặt ở một vị trí cụ thể trong khu vực của phần tử bằng cách sử dụng đặc tính `background-position`. Năm vị trí cơ bản là `top` (trên cùng), `bottom` (dưới cùng), `left (bên trái)`, `right` (bên phải) và `center` (giữa), nhưng vị trí trên cùng bên trái của hình ảnh cũng có thể được điều chỉnh bằng phần trăm:

```
body {
  background-image: url("background.jpg");
  background-repeat: no-repeat;
  background-position: 30% 10%;
}
```

Vị trí của hình nền (`background-position`) được đo từ các cạnh của phần tử chứa, trong ví dụ này là toàn bộ nội dung của trang. Vị trí có thể được chỉ định theo đơn vị chính xác (ví dụ: `1em`) hoặc theo tỷ lệ phần trăm của phần tử chứa. Do đó, trong ví dụ này, hình nền sẽ bắt đầu bằng 30% chiều rộng của nội dung trang tính từ phía bên trái và 10% chiều cao của nội dung trang tính từ phía trên cùng.

Đường viền

Thay đổi đường viền của phần tử cũng là một tùy chỉnh bố cục phổ biến được thực hiện bằng CSS. Đường viền ở đây là một đường tạo thành hình chữ nhật xung quanh phần tử và có ba đặc tính cơ bản: `color` (màu), `style` (kiểu) và `width` (độ dày).

Màu của đường viền được xác định bằng `border-color` theo cùng định dạng mà chúng ta đã thấy đối với các đặc tính màu khác.

Các đường viền có thể được vẽ theo nhiều kiểu khác ngoài một đường liền nét. Ví dụ: chúng ta sẽ sử dụng dấu gạch ngang (dash) cho đường viền với đặc tính `border-style: dashed`. Các giá trị kiểu khác là:

`dotted`

Chuỗi các chấm tròn

`double`

Hai đường thẳng

`groove`

Một đường có hình chạm khắc

`ridge`

Một đường có dạng nổi

`inset`

Một đường có dạng chìm

`outset`

Một khung dập nổi

Đặc tính `border-width` sẽ đặt độ dày của đường viền. Giá trị của nó có thể là một từ khóa (`thin` - mỏng, `medium` - trung bình, hoặc `thick` - dày) hoặc một giá trị số cụ thể. Nếu muốn sử dụng một giá trị số, ta cũng sẽ cần chỉ định đơn vị tương ứng của nó. Điều này sẽ được nói đến ngày sau đây.

Giá trị Đơn vị

Kích thước và khoảng cách trong CSS có thể được xác định theo nhiều cách khác nhau. Các đơn vị tuyệt đối được dựa trên một số lượng pixel màn hình cố định. Vì vậy, chúng sẽ không quá khác biệt so với kích thước và chiều cố định được sử dụng trong phương tiện in. Ngoài ra còn có các đơn vị tương đối được tính toán động từ một số phép đo được cung cấp bởi phương tiện nơi trang đang được hiển thị, chẳng hạn như không gian có sẵn hoặc một kích thước khác được viết bằng đơn vị tuyệt đối.

Đơn vị Tuyệt đối

Các đơn vị tuyệt đối cũng tương đương với các đối tác vật lý của chúng, như cm hoặc inch. Trên màn hình máy tính thông thường, một inch sẽ rộng 96 pixel (px). Các đơn vị tuyệt đối sau đây thường được sử dụng:

`in` (inch)

1 inch tương đương với 2,54 cm hoặc 96 px.

`cm` (xentimét)

1 cm tương đương với 96 px / 2,54.

`mm` (mi-li-mét)

1 mm tương đương với 1 cm/10.

`px` (pixel)

1 px tương đương với 1/96 inch.

`pt` (điểm)

1pt tương đương với 1/72 inch.

Hãy nhớ rằng tỷ lệ pixel trên inch có thể sẽ khác nhau. Ở màn hình có độ phân giải cao, nơi các điểm ảnh dày đặc hơn thì một inch sẽ tương ứng với hơn 96 điểm ảnh.

Đơn vị Tương đối

Các đơn vị tương đối sẽ khác nhau tùy theo phép đo hoặc theo kích thước khung nhìn. Khung nhìn là khu vực của tài liệu hiện người dùng có thể nhìn thấy trong cửa sổ của nó. Ở chế độ toàn màn hình, khung nhìn sẽ tương ứng với màn hình thiết bị. Các đơn vị tương đối sau đây thường được sử dụng:

`%`

Tỷ lệ phần trăm — nó có liên quan đến phần tử gốc.

`em`

Kích thước của phông chữ được sử dụng trong phần tử.

`rem`

Kích thước của phông chữ được sử dụng trong phần tử gốc.

`vw`

1% chiều rộng của khung nhìn.

`vh`

1% chiều cao của khung nhìn.

Ưu điểm của việc sử dụng các đơn vị tương đối là ta có thể tạo các bố cục có thể điều chỉnh được bằng cách chỉ thay đổi một vài kích thước chủ chốt. Ví dụ: chúng ta sẽ sử dụng đơn vị `pt` để đặt kích thước phông chữ trong phần tử nội dung và đơn vị `rem` cho phông chữ trong các phần tử khác. Khi thay đổi kích thước phông chữ cho nội dung, tất cả các kích thước phông chữ khác sẽ điều chỉnh tương ứng. Ngoài ra, việc sử dụng `vw` và `vh` để đặt kích thước cho các phần trang sẽ giúp chúng có thể điều chỉnh theo các màn hình có kích thước khác nhau.

Phông chữ và Đặc tính của Văn bản

Phong cách chữ (hoặc việc nghiên cứu về các loại phông chữ) là một chủ đề thiết kế rất rộng, và phong cách chữ trong CSS cũng rộng không kém. Tuy nhiên, có một vài đặc tính phông chữ cơ bản sẽ đáp ứng được nhu cầu của hầu hết người dùng học CSS.

Đặc tính `font-family` sẽ đặt tên của phông chữ được sử dụng. Không có gì đảm bảo được rằng phông chữ đã chọn sẽ có sẵn trong hệ thống nơi trang sẽ được xem. Vì vậy, đặc tính này có thể sẽ không có hiệu lực trong tài liệu. Mặc dù có thể làm cho trình duyệt tải xuống và sử dụng tệp phông chữ được chỉ định, nhưng hầu hết các nhà thiết kế web đều hài lòng với việc sử dụng các loại phông chữ cơ bản trong tài liệu của mình.

Ba họ phông chữ cơ bản phổ biến nhất là `serif`, `sans-serif` và `monospace`. Serif là họ phông chữ mặc định trong hầu hết các trình duyệt. Nếu muốn sử dụng `sans-serif` cho toàn bộ trang, hãy thêm quy tắc sau vào biểu định kiểu của bạn:

```
* {
  font-family: sans-serif;
}
```

Có thể tuỳ chọn đặt một họ phông chữ cụ thể trước tiên, theo sau là họ chung:

```
* {
  font-family: "DejaVu Sans", sans-serif;
}
```

Nếu thiết bị hiển thị trang có họ phông chữ cụ thể đó, trình duyệt sẽ sử dụng nó. Nếu không, nó sẽ sử dụng phông chữ mặc định phù hợp với họ chung. Trình duyệt tìm kiếm phông chữ theo thứ tự chúng được chỉ định trong đặc tính.

Bất kỳ ai đã từng sử dụng ứng dụng xử lý văn bản cũng sẽ quen thuộc với ba khía cạnh điều chỉnh phông chữ khác: kích thước, độ đậm nhạt và kiểu dáng. Ba điều chỉnh này có các đặc tính CSS tương ứng là `font-size` (kích thước chữ), `font-weight` (độ đậm nhạt) và `font-style` (kiểu chữ).

Đặc tính `font-size` chấp nhận các kích thước từ khóa như `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`, `xxx -large`. Những từ khóa này có liên quan đến kích thước phông chữ mặc định được sử dụng bởi trình duyệt. Các từ khóa `larger` (lớn hơn) và `smaller` (nhỏ hơn) sẽ thay đổi kích thước phông chữ tương ứng với kích thước phông chữ của phần tử gốc. Ta cũng có thể biểu thị kích thước của phông chữ bằng các giá trị số (bao gồm đơn vị sau giá trị) hoặc bằng tỷ lệ phần trăm.

Nếu không muốn thay đổi kích thước phông chữ nhưng lại muốn thay đổi khoảng cách giữa các dòng, hãy sử dụng thuộc tính `line-height` (chiều cao của dòng). `line-height` có giá trị `1` sẽ làm cho chiều cao của dòng có cùng kích thước với phông chữ của phần tử, điều này có thể làm cho các dòng văn bản hơi quá gần nhau. Do đó, giá trị lớn hơn 1 sẽ phù hợp hơn cho văn bản. Giống như thuộc tính `font-size`, các đơn vị khác cũng có thể được sử dụng cùng với giá trị số.

`font-weight` sẽ chỉ định độ đậm nhạt của phông chữ với các từ khóa quen thuộc là `normal` (bình thường) và `bold` (đậm). Các từ khóa `lighter` (nhạt hơn) và `bolder` (đậm hơn) sẽ thay đổi độ đậm của phông chữ của phần tử so với phần tử mẹ của nó.

Đặc tính `font-style` có thể được đặt thành `italic` để chọn phiên bản in nghiêng của họ phông chữ hiện tại. Giá trị `oblique` sẽ chọn phiên bản nghiêng đậm của phông chữ. Hai tùy chọn này gần như giống hệt nhau, nhưng phiên bản in nghiêng của phông chữ thường sẽ hẹp hơn một chút so với phiên bản in nghiêng đậm. Nếu trong trình duyệt không tồn tại các phiên bản in nghiêng hoặc nghiêng đậm, phông chữ sẽ bị trình duyệt tự nghiêng.

Có các đặc tính khác để thay đổi cách hiển thị văn bản trong tài liệu. Ví dụ: ta có thể thêm gạch chân vào một số phần của văn bản mà mình muốn nhấn mạnh. Đầu tiên, hãy sử dụng thẻ `<span>` để phân định văn bản:

```
<p>CSS is the <span class="under">proper mechanism</span> to style HTML documents.</p>
```

Sau đó, chúng ta sẽ sử dụng bộ chọn `.under` để thay đổi đặc tính `text-decoration`:

```
.under {
  text-decoration: underline;
}
```

Theo mặc định, tất cả các phần tử `a` (liên kết) đều sẽ được gạch dưới. Nếu muốn xóa nó, hãy sử dụng giá trị `none` cho `text-decoration` của tất cả các phần tử `a`:

```
a {
  text-decoration: none;
}
```

Khi xem xét nội dung, một số tác giả sẽ muốn gạch bỏ những phần không chính xác hoặc lỗi thời của văn bản để người đọc biết văn bản đã được cập nhật và những gì đã bị xóa. Để làm như vậy, hãy sử dụng giá trị `line-through` của đặc tính `text-decoration`:

```
.disregard {
  text-decoration: line-through;
}
```

Một lần nữa, thẻ `<span>` có thể được sử dụng để áp dụng kiểu:

```
<p>Netscape Navigator <span class="disregard">is</span> was one of the most popular Web
browsers.</p>
```

Các kiểu trang trí khác có thể là dành riêng cho từng phần tử cụ thể. Các hình tròn trong danh sách dấu đầu dòng có thể được tùy chỉnh bằng cách sử dụng đặc tính `list-style-type`. Ví dụ: để thay đổi chúng thành hình vuông (square), hãy sử dụng `list-style-type: square`. Để xóa chúng, hãy đặt giá trị của `list-style-type` thành `none`.

Bài tập Hướng dẫn

1. Làm cách nào để có thể thêm độ nửa trong suốt vào màu xanh lam (có ký hiệu RGB `rgb(0,0,255)`) để sử dụng nó trong đặc tính `color` trong CSS?

2. Màu gì sẽ tương ứng với giá trị thập lục phân `#000`?

3. Giả sử `Times New Roman` là một phông chữ `serif` và nó không khả dụng trên tất cả các thiết bị, làm cách nào để có thể viết quy tắc CSS yêu cầu `Times New Roman` trong khi đặt họ phông chữ chung `serif` làm dự phòng ?

4. Làm cách nào để có thể sử dụng từ khóa kích thước tương đối để đặt kích thước phông chữ của phần tử `<p class="disclaimer">` thành nhỏ hơn so với phần tử gốc của nó?

Bài tập Mở rộng

1. Đặc tính `background` là tốc ký để đặt nhiều đặc tính `background-*` cùng một lúc. Hãy viết lại quy tắc CSS sau dưới dạng một đặc tính tốc ký `background` duy nhất.

```
body {
  background-image: url("background.jpg");
  background-repeat: repeat-x;
}
```

2. Hãy viết một quy tắc CSS cho phần tử `<div id="header"></div>` để *chỉ* thay đổi chiều rộng đường viền dưới cùng thành `4px`.

3. Hãy viết một đặc tính `font-size` để tăng gấp đôi kích thước phông chữ được sử dụng trong phần tử gốc của trang.

4. *Cách dòng đôi* (Double-spacing) là một tính năng định dạng văn bản phổ biến trong trình xử lý văn bản. Làm cách nào để có thể đặt định dạng tương tự bằng CSS?

Tóm tắt

Bài học này đã nói về việc áp dụng các kiểu đơn giản cho các phần tử của tài liệu HTML. CSS cung cấp hàng trăm đặc tính và hầu hết các nhà thiết kế web sẽ cần đến các tài liệu tham khảo để có thể ghi nhớ. Tuy thế, một tập hợp nhỏ các đặc tính và giá trị tương đối được sử dụng khá nhiều và điều quan trọng là ta phải thuộc lòng chúng. Bài học đã đi qua các khái niệm và quy trình sau:

- Các đặc tính CSS cơ bản xử lý màu sắc, hình nền và phông chữ.
- Các đơn vị tuyệt đối và tương đối mà CSS có thể sử dụng để đặt kích thước và khoảng cách, chẳng hạn như `px`, `em`, `rem`, `vw`, `vh`, v.v.

Đáp án Bài tập Hướng dẫn

1. Làm cách nào để có thể thêm độ nửa trong suốt vào màu xanh lam (có ký hiệu RGB `rgb(0,0,255)`) để sử dụng nó trong đặc tính `color` trong CSS?

   Sử dụng tiền tố `rgba` và thêm `0,5` làm giá trị thứ tư: `rgba(0,0,0,0,5)`.

2. Màu gì sẽ tương ứng với giá trị thập lục phân `#000`?

   Màu đen. Giá trị thập lục phân `#000` là cách viết tắt của `#000000`.

3. Giả sử `Times New Roman` là một phông chữ `serif` và nó sẽ không khả dụng trên tất cả các thiết bị, làm cách nào để có thể viết quy tắc CSS yêu cầu `Times New Roman` trong khi đặt họ phông chữ chung `serif` làm dự phòng ?

   ```
   * {
     font-family: "Times New Roman", serif;
   }
   ```

4. Làm cách nào để có thể sử dụng từ khóa kích thước tương đối để đặt kích thước phông chữ của phần tử `<p class="disclaimer">` thành nhỏ hơn so với phần tử gốc của nó?

   Sử dụng từ khóa `smaller`:

   ```
   p.disclaimer {
     font-size: smaller;
   }
   ```

Đáp án Bài tập Mở rộng

1. Đặc tính `background` là tốc ký để đặt nhiều đặc tính `background-*` cùng một lúc. Hãy viết lại quy tắc CSS sau dưới dạng một đặc tính tốc ký `background` duy nhất.

```
body {
  background-image: url("background.jpg");
  background-repeat: repeat-x;
}
```

```
body {
  background: repeat-x url("background.jpg");
}
```

2. Hãy viết một quy tắc CSS cho phần tử `<div id="header"></div>` để *chỉ* thay đổi chiều rộng đường viền dưới cùng thành `4px`.

```
#header {
  border-bottom-width: 4px;
}
```

3. Hãy viết một đặc tính `font-size` để tăng gấp đôi kích thước phông chữ được sử dụng trong phần tử gốc của trang.

   Đơn vị `rem` tương ứng với kích thước phông chữ được sử dụng trong phần tử gốc. Vì vậy, đặc tính sẽ phải là `font-size: 2rem`.

4. *Cách dòng đôi* (Double-spacing) là một tính năng định dạng văn bản phổ biến trong trình xử lý văn bản. Làm cách nào để có thể đặt định dạng tương tự bằng CSS?

   Đặt đặc tính `line-height` thành giá trị `2em` vì đơn vị `em` sẽ tương ứng với kích thước phông chữ của phần tử hiện tại.

033.4 Mô hình và Bố cục Hộp trong CSS

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 033.4

Khối lượng

2

Các lĩnh vực kiến thức chính

- Xác định kích thước, vị trí và căn chỉnh của các thành phần trong bố cục CSS
- Chỉ định cách văn bản di chuyển xung quanh các phần tử khác
- Hiểu về luồng tài liệu
- Nhận thức về lưới CSS
- Nhận thức về kiểu thiết kế web đáp ứng
- Nhận thức về truy vấn phương tiện CSS

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `width`, `height`, `padding`, `padding-*`, `margin`, `margin-*`, `border`, `border-*`
- `top`, `left`, `right`, `bottom`
- `display: block | inline | flex | inline-flex | none`
- `position: static | relative | absolute | fixed | sticky`
- `float: left | right | none`
- `clear: left | right | both | none`

033.4 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	033 Tạo kiểu Nội dung CSS
Mục tiêu:	033.4 Bố cục và Mô hình Hộp trong CSS
Bài:	1 trên 1

Giới thiệu

Mỗi phần tử hiện hữu trong tài liệu HTML đều được hiển thị dưới dạng một chiếc hộp hình chữ nhật. Do đó, thuật ngữ *mô hình hộp* mô tả cách tiếp cận mà CSS sử dụng để sửa đổi các đặc tính trực quan của các phần tử. Giống như các hộp có kích thước khác nhau, các phần tử HTML cũng có thể được lồng bên trong các phần tử *chứa* — thường là phần tử `div` — để chúng có thể được phân chia thành từng phần riêng biệt.

Chúng ta có thể sử dụng CSS để sửa đổi vị trí của các hộp, từ những điều chỉnh nhỏ đến những thay đổi lớn trong cách bố trí các phần tử trên trang. Bên cạnh luồng thông thường, vị trí của mỗi hộp có thể được dựa trên các thành phần xung quanh nó, có thể là mối quan hệ của hộp với vùng chứa mẹ hoặc của hộp với *khung nhìn* - khu vực hiển thị cho người dùng của trang. Không có một cơ chế đơn lẻ nào có thể đáp ứng tất cả các yêu cầu về bố cục; do đó, ta có thể sẽ cần phải kết hợp chúng.

Luồng thông thường

Cách mặc định mà trình duyệt hiển thị cây tài liệu được gọi là *luồng thông thường*. Các hình chữ nhật tương ứng với các phần tử sẽ ít nhiều được đặt theo cùng một thứ tự như trong cây tài liệu và

tương quan với các phần tử mẹ. Tuy nhiên, tùy thuộc vào loại phần tử, hộp tương ứng có thể sẽ tuân theo các quy tắc định vị riêng biệt.

Một cách hay để hiểu được logic của luồng thông thường là làm cho các hộp có thể nhìn thấy được. Chúng ta có thể bắt đầu với một trang rất cơ bản chỉ có ba phần tử `div` riêng biệt, mỗi phần tử có một đoạn văn bản ngẫu nhiên:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CSS Box Model and Layout</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

<div id="first">
  <h2>First div</h2>
  <p><span>Sed</span> <span>eget</span> <span>velit</span>
  <span>id</span> <span>ante</span> <span>tempus</span>
  <span>porta</span> <span>pulvinar</span> <span>et</span>
  <span>ex.</span></p>
</div><!-- #first -->

<div id="second">
  <h2>Second div</h2>
  <p><span>Fusce</span> <span>vitae</span> <span>vehicula</span>
  <span>neque.</span> <span>Etiam</span> <span>maximus</span>
  <span>vulputate</span> <span>neque</span> <span>eu</span>
  <span>lobortis.</span> <span>Phasellus</span> <span>condimentum,</span>
  <span>felis</span> <span>eget</span> <span>eleifend</span>
  <span>aliquam,</span> <span>dui</span> <span>dolor</span>
  <span>bibendum</span> <span>leo.</span></p>
</div><!-- #second -->

<div id="third">
  <h2>Third div</h2>
  <p><span>Pellentesque</span> <span>ornare</span> <span>ultricies</span>
  <span>elementum.</span> <span>Morbi</span> <span>vulputate</span>
  <span>pretium</span> <span>arcu,</span> <span>sed</span>
  <span>faucibus.</span></p>
</div><!-- #third -->
```

```
</body>
</html>
```

Mỗi từ đều nằm trong phần tử `span` để chúng ta có thể tạo kiểu cho các từ và xem cách chúng cũng được xử lý dưới dạng hộp. Để hiển thị các hộp, chúng ta phải chỉnh sửa tệp biểu định kiểu `style.css` được tham chiếu bởi tài liệu HTML. Các quy tắc sau đây sẽ làm được điều này:

```
* {
  font-family: sans;
  font-size: 14pt;
}

div {
  border: 2px solid #00000044;
}

#first {
  background-color: #c4a000ff;
}

#second {
  background-color: #4e9a06ff;
}

#third {
  background-color: #5c3566da;
}

h2 {
  background-color: #ffffff66;
}

p {
  background-color: #ffffff66;
}

span {
  background-color: #ffffffaa;
}
```

Kết quả sẽ được hiển thị trong Figure 32.

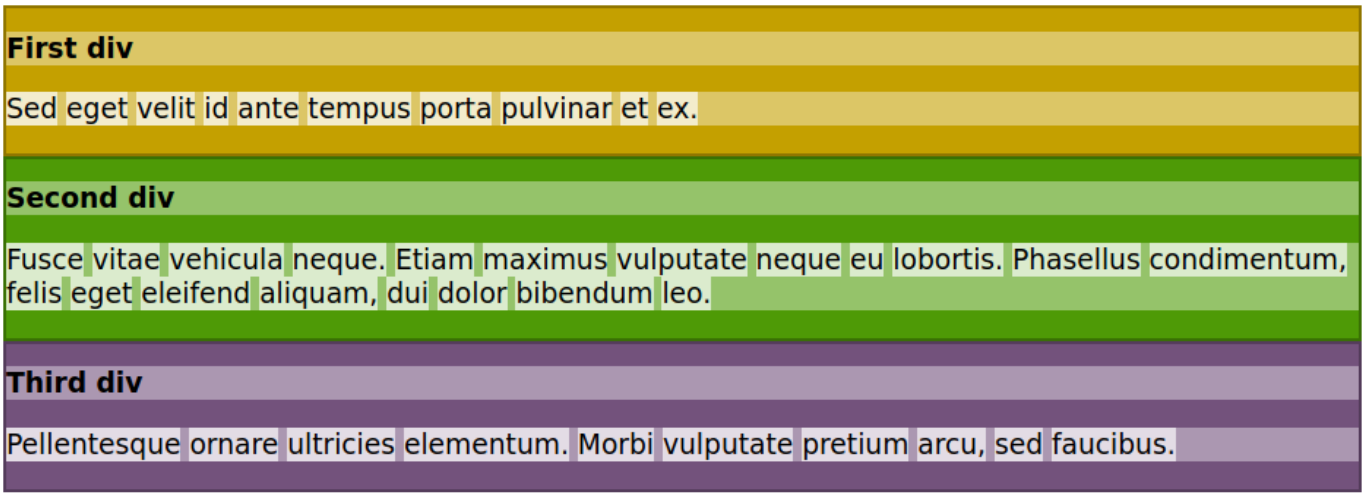

Figure 32. Luồng phần tử cơ bản là từ trên xuống dưới và từ trái sang phải.

Figure 32 cho ta thấy rằng mỗi thẻ HTML sẽ có một hộp tương ứng trong bố cục. Các phần tử `div`, `h2` và `p` sẽ mở theo chiều rộng của phần tử mẹ của chúng. Chẳng hạn như phần tử mẹ của các phần tử `div` là phần tử `body`; vì vậy, chúng sẽ mở rộng theo chiều rộng của phần thân, trong khi phần tử mẹ của mỗi phần tử `h2` và `p` là phần tử `div` tương ứng của nó. Các hộp mở rộng theo chiều rộng của phần tử mẹ được gọi là các phần tử *khối*. Một số thẻ HTML phổ biến nhất được hiển thị dưới dạng khối là `h1`, `h2`, `h3`, `p`, `ul`, `ol`, `table`, `li`, `div`, `section` , `form` và `aside`. Các phần tử khối cùng cấp — các phần tử khối chia sẻ trực tiếp cùng một phần tử mẹ — sẽ được xếp chồng lên nhau bên trong phần tử mẹ của chúng theo chiều từ trên xuống dưới.

NOTE

Một số phần tử khối không được nhắm đến việc trở thành vùng chứa cho các phần tử khối khác. Ví dụ: có thể chèn một phần tử khối bên trong phần tử `h1` hoặc `p` nhưng nó sẽ không được coi là phương pháp hay nhất. Thay vào đó, nhà phát triển nên sử dụng thẻ thích hợp để làm vùng chứa. Các thẻ vùng chứa phổ biến là `div`, `section` và `aside`.

Bên cạnh văn bản, các phần tử như `h1`, `p` và `li` chỉ cho phép các phần tử *nội tuyến* làm phần tử con của nó. Giống như hầu hết các tệp lệnh phương Tây, các thành phần nội tuyến sẽ tuân theo luồng văn bản từ trái sang phải. Khi không còn chỗ trống ở phía bên phải, luồng của các thành phần nội tuyến sẽ tiếp tục ở dòng tiếp theo giống như văn bản. Một số thẻ HTML phổ biến được coi là hộp nội tuyến là `span`, `a`, `em`, `strong`, `img`, `input` và `label`.

Trong trang HTML mẫu của chúng ta, mỗi từ bên trong đoạn văn bản đều được bao quanh bởi thẻ `span`; vì vậy, chúng có thể được đánh dấu bằng quy tắc CSS tương ứng. Như được thể hiện trong hình ảnh, mỗi phần tử `span` đều được đặt theo chiều ngang, từ trái sang phải cho đến khi không còn chỗ trong phần tử mẹ.

Chiều cao của phần tử phụ thuộc vào nội dung của nó; vì vậy, trình duyệt sẽ điều chỉnh chiều cao

của phần tử chứa để phù hợp với các phần tử khối lồng nhau hoặc các dòng của phần tử nội tuyến. Tuy nhiên, một số đặc tính CSS sẽ ảnh hưởng đến hình dạng, vị trí của hộp và vị trí của các thành phần bên trong hộp.

Các đặc tính `margin` (lề) và `padding` (đệm) đều ảnh hưởng đến tất cả các loại hộp. Nếu không thiết lập các đặc tính này một cách rõ ràng, trình duyệt sẽ tự thiết lập một số giá trị tiêu chuẩn. Như đã thấy trong Figure 32, các phần tử `h2` và `p` được hiển thị có khoảng cách giữa chúng. Những khoảng cách này là lề trên và lề dưới mà trình duyệt mặc định thêm vào các phần tử này. Chúng ta có thể xóa chúng bằng cách sửa đổi các quy tắc CSS cho bộ chọn `h2` và `p`:

```
h2 {
  background-color: #ffffff66;
  margin: 0;
}

p {
  background-color: #ffffff66;
  margin: 0;
}
```

Kết quả sẽ được hiển thị trong Figure 33.

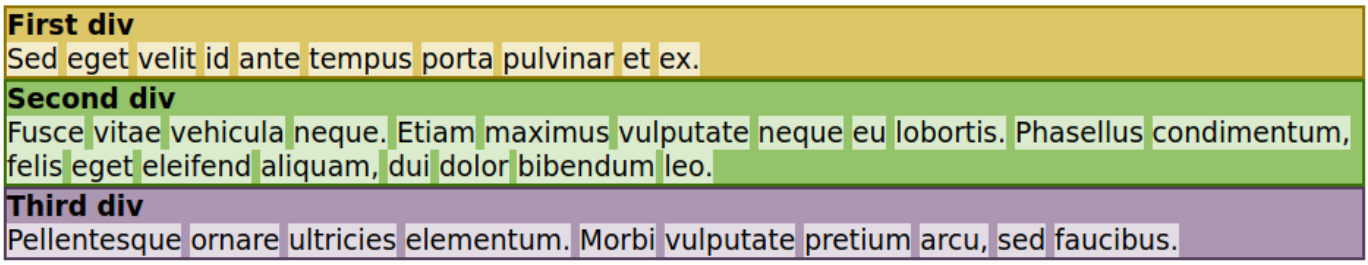

Figure 33. Đặc tính `margin` có thể thay đổi hoặc loại bỏ lề khỏi các phần tử.

Theo mặc định, phần tử `body` cũng có một lề nhỏ tạo ra một khoảng trống xung quanh. Khoảng cách này cũng có thể được loại bỏ bằng cách sử dụng đặc tính `margin`.

Trong khi đặc tính `margin` xác định khoảng cách giữa phần tử và môi trường xung quanh thì đặc tính `padding` của phần tử sẽ xác định khoảng cách bên trong giữa ranh giới của vùng chứa và các phần tử con của nó. Ví dụ: hãy xem xét các phần tử `h2` và `p` bên trong mỗi `div` trong đoạn mã mẫu. Chúng ta có thể sử dụng đặc tính lề của chúng để tạo khoảng cách với đường viền của `div` tương ứng, nhưng việc thay đổi đặc tính `padding` của vùng chứa sẽ đơn giản hơn:

```
#second {
```

```
  background-color: #4e9a06ff;
  padding: 1em;
}
```

Chỉ có quy tắc cho `div` thứ hai được sửa đổi. Do đó, kết quả (Figure 34) cho thấy sự khác biệt giữa `div` thứ hai và các vùng chứa `div` khác.

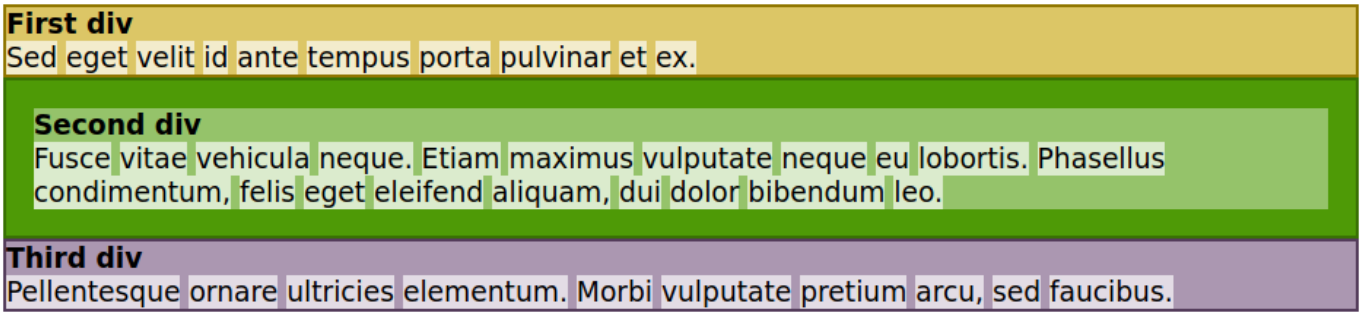

Figure 34. Các vùng chứa `div` khác nhau có thể có các phần đệm khác nhau.

Đặc tính `margin` là cách viết tắt của bốn đặc tính kiểm soát bốn cạnh của hộp: `margin-top` (lề trên), `margin-right` (lề phải), `margin-bottom` (lề dưới) và `margin-left` (lề trái). Khi `margin` được chỉ định một giá trị như trong các ví dụ, cả bốn lề của hộp đều sẽ sử dụng giá trị đó. Khi có hai giá trị được chỉ định, giá trị đầu tiên sẽ xác định lề trên và dưới, trong khi giá trị thứ hai sẽ xác định lề phải và trái. Ví dụ: ta sử dụng `margin: 1em 2em` để xác định khoảng cách 1 em cho lề trên và lề dưới và khoảng cách 2 em cho lề phải và trái. Việc đặt bốn giá trị sẽ đặt lề cho bốn cạnh theo chiều kim đồng hồ, bắt đầu từ cạnh trên cùng. Các giá trị khác nhau trong đặc tính tốc ký không bắt buộc phải sử dụng cùng một đơn vị.

Đặc tính `padding` cũng là một cách viết tắt và tuân theo các nguyên tắc giống như đặc tính `margin`.

Trong hành vi mặc định của chúng, các phần tử khối sẽ kéo dài để vừa với chiều rộng có sẵn. Nhưng điều này không phải là bắt buộc. Đặc tính `width` có thể đặt kích thước ngang cố định cho hộp:

```
#first {
  background-color: #c4a000ff;
  width: 6em;
}
```

Việc thêm `width: 6em` vào quy tắc CSS sẽ thu nhỏ `div` đầu tiên theo chiều ngang và để lại một khoảng trống ở phía bên phải của nó (Figure 35).

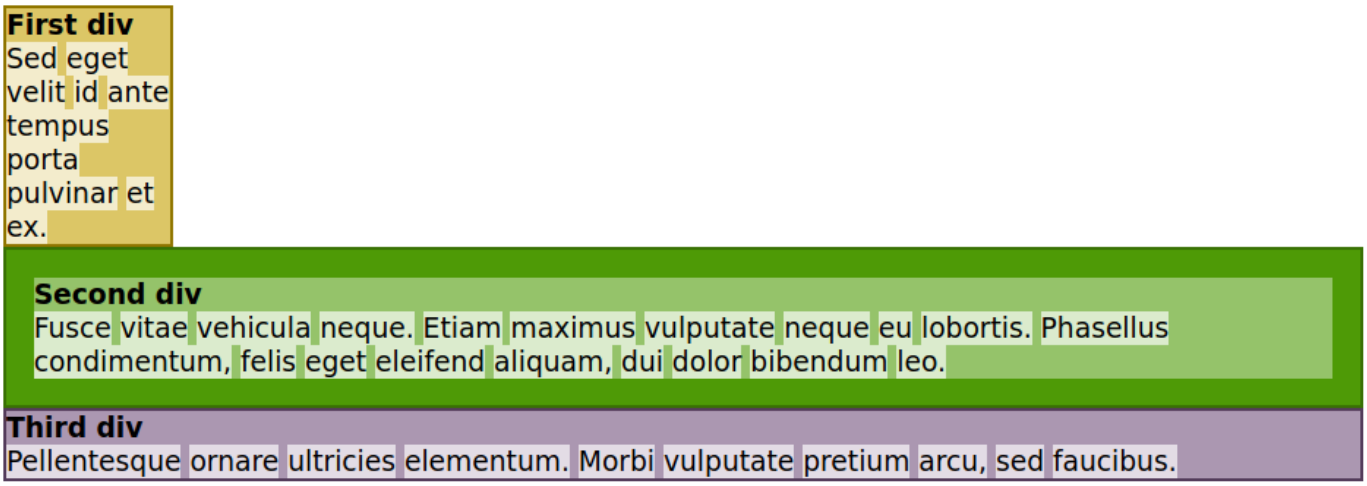

Figure 35. Đặc tính `width` sẽ thay đổi độ rộng theo chiều ngang của `div` đầu tiên.

Thay vì để `div` đầu tiên được căn sang bên trái, chúng ta có thể căn nó vào giữa. Căn giữa cho hộp cũng tương đương với việc đặt lề có cùng kích thước ở cả hai bên. Vì vậy, chúng ta cũng có thể sử dụng đặc tính lề để căn giữa hộp. Kích thước của không gian có sẵn có thể sẽ khác nhau; vì vậy, chúng ta sẽ sử dụng giá trị `auto` cho lề trái và phải:

```
#first {
  background-color: #c4a000ff;
  width: 6em;
  margin: 0 auto;
}
```

Lề trái và phải sẽ được trình duyệt tự động tính toán và hộp sẽ được căn giữa (Figure 36).

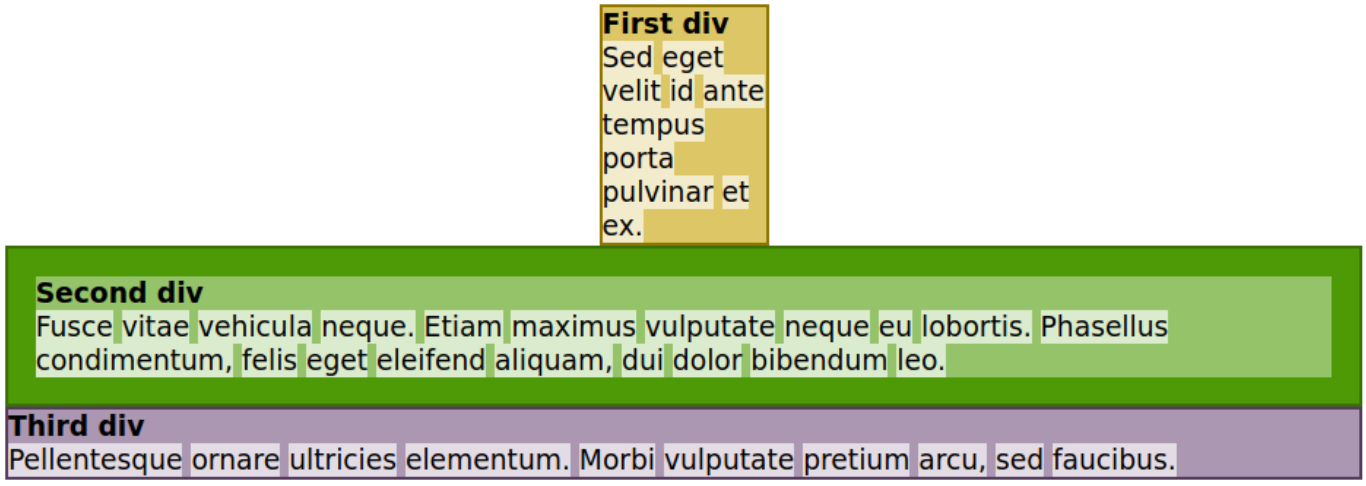

Figure 36. Đặc tính `margin` được sử dụng để căn giữa `div` đầu tiên.

Như được hiển thị, việc làm cho một phần tử khối hẹp hơn sẽ không cho phần tử tiếp theo khoảng trống còn lại. Luồng tự nhiên vẫn được giữ nguyên như thể phần tử hẹp hơn vẫn chiếm hết chiều rộng có sẵn.

Tùy chỉnh Luồng thông thường

Luồng thông thường có tính chất đơn giản và tuần tự. CSS cũng cho phép bạn phá vỡ luồng thông thường và định vị các phần tử theo những cách riêng, thậm chí có thể ghi đè lên thao tác cuộn trang nếu muốn. Chúng ta sẽ xem xét một số cách để kiểm soát vị trí của các phần tử trong phần này.

Phần tử Nổi

Ta có thể làm cho các phần tử khối cùng cấp chia sẻ cùng một không gian theo chiều ngang. Một cách để làm việc này là thông qua đặc tính `float` (nổi) và loại bỏ phần tử khỏi luồng thông thường. Như tên gọi của nó, đặc tính `float` sẽ làm cho hộp nổi lên trên các phần tử khối theo sau. Vì vậy, chúng sẽ được hiển thị như thể chúng đang nằm dưới hộp nổi. Để làm cho `div` đầu tiên nổi sang bên phải, hãy thêm `float: right` vào quy tắc CSS tương ứng:

```
#first {
  background-color: #c4a000ff;
  width: 6em;
  float: right;
}
```

Các lề tự động sẽ bị bỏ qua trong hộp nổi; do đó, thuộc tính `margin` có thể sẽ bị xóa. Figure 37 hiển thị kết quả của việc di chuyển `div` đầu tiên sang bên phải.

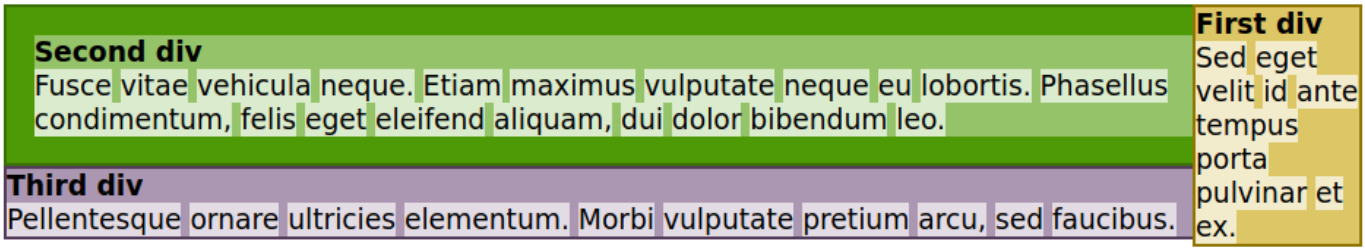

Figure 37. `div` đầu tiên đang nổi và không phải là một phần của luồng thông thường.

Theo mặc định, tất cả các phần tử khối xuất hiện sau phần tử nổi sẽ nằm bên dưới nó. Do đó, nếu có đủ chiều cao, hộp nổi sẽ bao phủ tất cả các phần tử khối còn lại.

Mặc dù một phần tử nổi sẽ nằm trên các phần tử khối khác, nội dung nội tuyến bên trong vùng

chứa của phần tử nổi vẫn sẽ bao quanh phần tử nổi. Nguồn cảm hứng cho điều này đến từ bố cục trong tạp chí và báo (ví dụ như việc sử dụng càng phần văn bản để bao quanh một hình ảnh).

Hình ảnh trước đã cho thấy cách `div` đầu tiên bao phủ `div` thứ hai và một phần của `div` thứ ba. Giả sử chúng ta muốn `div` đầu tiên nổi trên `div` thứ hai chứ không phải div thứ ba. Giải pháp ở đây sẽ là sử dụng đặc tính `clear` trong quy tắc CSS tương ứng cho `div` thứ ba:

```
#third {
  background-color: #5c3566da;
  clear: right;
}
```

Đặt đặc tính `clear` thành `right` sẽ làm cho phần tử tương ứng bỏ qua bất kỳ phần tử nào trước đó được đặt nổi sang bên phải và tiếp tục luồng thông thường (Figure 38).

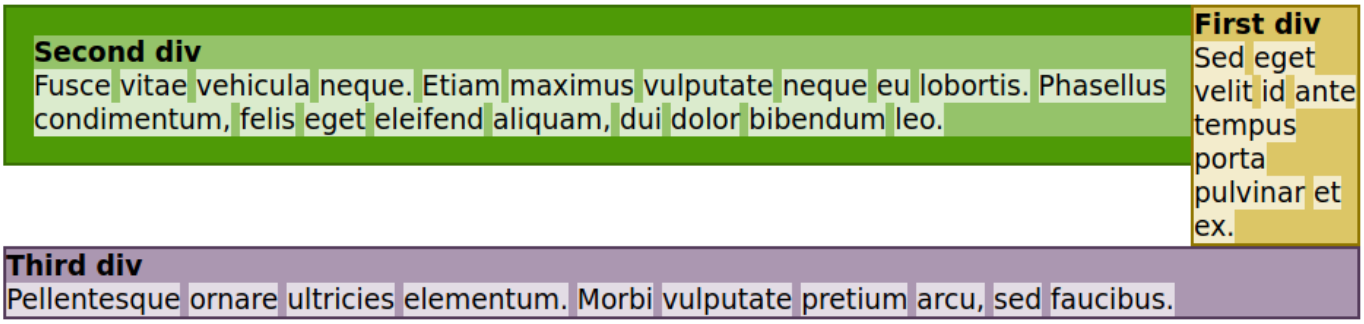

Figure 38. Đặc tính `clear` trở lại luồng thông thường.

Tương tự như vậy, nếu một phần tử trước đó nổi sang trái, chúng ta sẽ sử dụng `clear: left` để tiếp tục luồng bình thường. Khi bạn muốn bỏ qua các phần tử nổi ở cả bên trái và bên phải, hãy sử dụng `clear: both`.

Định vị Hộp

Trong luồng thông thường, mỗi hộp sẽ đi sau các hộp trước đó trong cây tài liệu. Các phần tử cùng cấp trước đó sẽ "đẩy" các phần tử theo sau chúng, di chuyển chúng sang phải và xuống dưới bên trong phần tử mẹ. Phần tử mẹ có thể có những phần tử cùng cấp cùng làm điều tương tự với nó. Nó giống như việc đặt những viên gạch cạnh nhau trên tường, bắt đầu từ trên cùng.

Phương thức định vị các hộp này được gọi là phương thức *tĩnh* và là giá trị mặc định cho đặc tính `position` (vị trí) trong CSS. Ngoài việc xác định lề và phần đệm, không có cách nào khác để định vị lại hộp tĩnh trong trang.

Giống như hình ảnh các viên gạch trên một bức tường, việc đặt hộp tĩnh không phải là bắt buộc.

Đối với gạch, ta có thể đặt các hộp ở bất cứ đâu mình muốn, thậm chí là che phủ lên các hộp khác. Để làm điều tương tự như vậy, hãy gán đặc tính `position` cho một trong các giá trị sau:

`relative` (tương đối)

Phần tử sẽ tuân theo luồng thông thường của tài liệu, nhưng nó có thể sử dụng các đặc tính `top`, `right`, `bottom` và `left` để đặt độ lệch so với vị trí tĩnh ban đầu của nó. Độ lệch cũng có thể là số âm. Các phần tử khác sẽ vẫn giữ nguyên vị trí ban đầu của chúng như thể phần tử tương đối vẫn đang ở trạng thái tĩnh.

`absolute` (tuyệt đối)

Phần tử sẽ bỏ qua luồng thông thường của các phần tử khác và tự định vị trên trang theo các đặc tính `top`, `right`, `bottom` và `left`. Các giá trị của chúng có liên quan đến phần thân của tài liệu hoặc với vùng chứa chính không tĩnh.

`fixed` (cố định)

Phần tử sẽ bỏ qua luồng thông thường của các phần tử khác và định vị chính nó theo các đặc tính `top`, `right`, `bottom` và `left`. Các giá trị của chúng liên quan đến khung nhìn (tức là vùng màn hình nơi tài liệu được hiển thị). Các phần tử cố định sẽ không di chuyển khi khách truy cập cuộn qua tài liệu mà sẽ giống như một nhãn dán cố định trên màn hình.

`sticky` (dính)

Phần tử sẽ tuân theo luồng thông thường của tài liệu. Tuy nhiên, thay vì đi biến mất khỏi khung nhìn khi tài liệu cuộn, nó sẽ dừng ở vị trí được thiết lập bởi các đặc tính `top`, `right`, `bottom` và `left`. Ví dụ: nếu giá trị `top` là `10px` thì phần tử sẽ ngừng cuộn dưới phần trên cùng của khung nhìn khi nó đạt đến 10 pixel tính từ giới hạn trên cùng của khung nhìn. Khi điều này xảy ra, phần còn lại của trang vẫn sẽ tiếp tục cuộn, nhưng phần tử dính sẽ hoạt động giống như một phần tử cố định ở vị trí đó. Nó sẽ quay trở lại vị trí ban đầu khi tài liệu cuộn trở lại vị trí của nó trong khung nhìn. Các phần tử dính ngày nay thường được sử dụng để tạo các menu đầu trang luôn cần được hiển thị.

Các vị trí không bắt buộc phải sử dụng tất cả các đặc tính `top`, `right`, `bottom` và `left`. Ví dụ: nếu bạn đặt cả hai đặc tính `top` và `height` của một phần tử tuyệt đối thì trình duyệt sẽ ngầm tính toán đặc tính `bottom` của nó (top + height = bottom).

Đặc tính `display`

Nếu thứ tự do luồng thông thường đưa ra không phải là vấn đề trong thiết kế của bạn nhưng bạn lại muốn thay đổi cách các hộp tự sắp xếp trong trang thì hãy sửa đổi đặc tính `display` (hiển thị) của phần tử. Đặc tính `display` thậm chí có thể làm cho phần tử biến mất hoàn toàn khỏi tài liệu được hiển thị bằng cách đặt `display: none`. Điều này sẽ hữu ích khi bạn muốn hiển thị phần tử sau này bằng JavaScript.

Ví dụ như đặc tính `display` cũng có thể làm cho một phần tử khối hoạt động giống như một phần tử nội tuyến (`display: inline`). Tuy nhiên, việc này không phải là một ý hay. Có các phương pháp tốt hơn để đặt các phần tử vùng chứa cạnh nhau, chẳng hạn như *mô hình hộp linh hoạt*.

Mô hình hộp linh hoạt được phát minh để khắc phục những hạn chế của các phần tử nổi và để loại bỏ việc sử dụng các bảng không phù hợp trong việc cấu trúc bố cục trang. Khi đặt đặc tính `display` của phần tử chứa thành `flex` để biến nó thành vật chứa hộp linh hoạt, các phần tử con trực tiếp của nó sẽ hoạt động giống như các ô trong một hàng của bảng.

TIP Nếu muốn kiểm soát nhiều hơn nữa đối với vị trí của các phần tử trên trang, hãy xem tính năng *lưới (grid) CSS*. Lưới là một hệ thống mạnh mẽ dựa trên các hàng và cột để tạo các bố cục phức tạp.

Để kiểm tra hiển thị linh hoạt, hãy thêm một phần tử `div` mới vào trang ví dụ và biến nó thành vật chứa cho ba phần tử `div` hiện có:

```
<div id="container">

<div id="first">
  <h2>First div</h2>
  <p><span>Sed</span> <span>eget</span> <span>velit</span>
  <span>id</span> <span>ante</span> <span>tempus</span>
  <span>porta</span> <span>pulvinar</span> <span>et</span>
  <span>ex.</span></p>
</div><!-- #first -->

<div id="second">
  <h2>Second div</h2>
  <p><span>Fusce</span> <span>vitae</span> <span>vehicula</span>
  <span>neque.</span> <span>Etiam</span> <span>maximus</span>
  <span>vulputate</span> <span>neque</span> <span>eu</span>
  <span>lobortis.</span> <span>Phasellus</span> <span>condimentum,</span>
  <span>felis</span> <span>eget</span> <span>eleifend</span>
  <span>aliquam,</span> <span>dui</span> <span>dolor</span>
  <span>bibendum</span> <span>leo.</span></p>
</div><!-- #second -->

<div id="third">
  <h2>Third div</h2>
  <p><span>Pellentesque</span> <span>ornare</span> <span>ultricies</span>
  <span>elementum.</span> <span>Morbi</span> <span>vulputate</span>
  <span>pretium</span> <span>arcu,</span> <span>sed</span>
  <span>faucibus.</span></p>
```

```
</div><!-- #third -->


</div><!-- #container -->
```

Thêm quy tắc CSS sau vào biểu định kiểu để biến vật chứa `div` thành vật chứa hộp linh hoạt:

```
#container {
  display: flex;
}
```

Kết quả là ba phần tử `div` bên trong được hiển thị cạnh nhau (Figure 39).

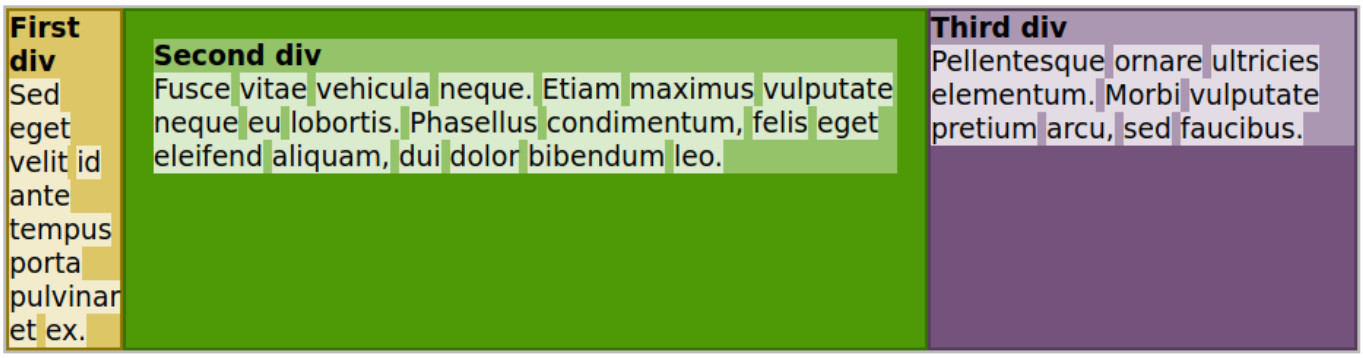

Figure 39. Mô hình hộp linh hoạt đã tạo ra lưới.

Việc sử dụng giá trị `inline-flex` thay vì `flex` về cơ bản sẽ mang lại cùng một kết quả, nhưng nó sẽ làm cho phần tử con hoạt động giống các phần tử nội tuyến hơn.

Thiết kế Đáp ứng

Chúng ta đã biết rằng CSS cung cấp các đặc tính điều chỉnh kích thước của các phần tử và phông chữ tương ứng với vùng màn hình có sẵn. Tuy nhiên, ta có thể sẽ muốn tiến xa hơn và sử dụng những thiết kế khác nhau cho các thiết bị khác nhau, ví dụ như cho một hệ thống máy tính để bàn so với thiết bị có kích thước màn hình dưới một ngưỡng nhất định. Cách tiếp cận này được gọi là *thiết kế web đáp ứng* và CSS có cung cấp các phương thức được gọi là *truy vấn phương tiện* để thực hiện điều này.

Trong ví dụ trước, chúng ta đã sửa đổi bố cục trang để đặt các phần tử `div` cạnh nhau thành các cột. Bố cục đó phù hợp với màn hình lớn, nhưng nó sẽ quá lộn xộn trên các màn hình nhỏ hơn. Để giải quyết vấn đề này, chúng ta có thể thêm một truy vấn phương tiện vào biểu định kiểu chỉ phù hợp với màn hình có chiều rộng ít nhất `600px`:

```
@media (min-width: 600px){
  #container {
    display: flex;
  }
}
```

Các quy tắc CSS bên trong lệnh `@media` (phương tiện) sẽ chỉ được sử dụng nếu tiêu chí trong ngoặc đơn được đáp ứng. Trong ví dụ này, nếu chiều rộng của khung nhìn nhỏ hơn `600px` thì quy tắc sẽ không được áp dụng cho vật chứa `div` và các phần tử con của nó sẽ được hiển thị dưới dạng các phần tử `div` thông thường. Trình duyệt sẽ đánh giá lại các truy vấn phương tiện mỗi khi kích thước khung nhìn thay đổi; do đó, bố cục có thể được thay đổi trong thời gian thực, cùng lúc thay đổi kích thước cửa sổ trình duyệt hoặc chiều xoay của điện thoại thông minh.

Bài tập Hướng dẫn

1. Nếu đặc tính `position` không được sửa đổi thì trình duyệt sẽ sử dụng phương pháp định vị nào?

2. Làm cách nào để có thể chắc chắn rằng hộp của phần tử sẽ được hiển thị sau bất kỳ phần tử nổi nào trước đó?

3. Làm cách nào để có thể sử dụng đặc tính tốc ký `margin` để đặt lề trên/dưới thành `4px` và lề phải/lề trái thành `6em`?

4. Làm cách nào để có thể căn giữa theo chiều ngang một phần tử chứa tĩnh có chiều rộng cố định trên trang?

Bài tập Mở rộng

1. Hãy viết một quy tắc CSS khớp với phần tử `<div class="picture">` để văn bản bên trong các phần tử khối tiếp theo sau sẽ nằm về phía bên phải của nó.

2. Làm thế nào để thuộc tính `top` ảnh hưởng đến một phần tử tĩnh liên quan tới phần tử mẹ của nó?

3. Việc thay đổi đặc tính `display` của một phần tử thành `flex` sẽ ảnh hưởng như thế nào đến vị trí của nó trong luồng thông thường?

4. Tính năng CSS nào cho phép bạn sử dụng một bộ quy tắc riêng tùy thuộc vào kích thước của màn hình?

Tóm tắt

Bài học này đã nói về mô hình hộp trong CSS và cách chúng ta có thể tùy chỉnh nó. Ngoài luồng thông thường của tài liệu, nhà phát triển có thể sử dụng các cơ chế định vị khác nhau để triển khai một bố cục tùy chỉnh. Bài học đã đi qua các khái niệm và quy trình sau:

- Luồng thông thường của tài liệu.
- Điều chỉnh lề và phần đệm của hộp phần tử.
- Sử dụng đặc tính float và clear.
- Cơ chế định vị: tĩnh, tương đối, tuyệt đối, cố định và dính.
- Các giá trị thay thế cho đặc tính `display`.
- Khái niệm cơ bản về thiết kế đáp ứng.

Đáp án Bài tập Hướng dẫn

1. Nếu đặc tính `position` không được sửa đổi thì trình duyệt sẽ sử dụng phương pháp định vị nào?

   Phương thức `static`.

2. Làm cách nào để có thể chắc chắn rằng hộp của phần tử sẽ được hiển thị sau bất kỳ phần tử nổi nào trước đó?

   Đặt đặc tính `clear` của phần tử thành `` both``.

3. Làm cách nào để có thể sử dụng đặc tính tốc ký `margin` để đặt lề trên/dưới thành `4px` và lề phải/lề trái thành `6em`?

   Có thể là `margin: 4px 6em` hoặc `margin: 4px 6em 4px 6em`.

4. Làm cách nào để có thể căn giữa theo chiều ngang một phần tử chứa tĩnh có chiều rộng cố định trên trang?

   Sử dụng giá trị `auto` trong đặc tính `margin-left` và `margin-right`.

Đáp án Bài tập Mở rộng

1. Hãy viết một quy tắc CSS khớp với phần tử `<div class="picture">` để văn bản bên trong các phần tử khối tiếp theo sau sẽ nằm về phía bên phải của nó.

   ```
   .picture { float: left; }
   ```

2. Làm thế nào để thuộc tính `top` ảnh hưởng đến một phần tử tĩnh liên quan tới phần tử mẹ của nó?

   Đặc tính `top` không áp dụng cho các phần tử tĩnh.

3. Việc thay đổi đặc tính `display` của một phần tử thành `flex` sẽ ảnh hưởng như thế nào đến vị trí của nó trong luồng thông thường?

   Bản thân vị trí của phần tử sẽ không thay đổi, nhưng các phần tử con trực tiếp của nó sẽ được hiển thị cạnh nhau theo chiều ngang.

4. Tính năng CSS nào cho phép bạn sử dụng một bộ quy tắc riêng tùy thuộc vào kích thước của màn hình?

   Truy vấn phương tiện sẽ cho phép trình duyệt xác minh kích thước khung nhìn trước khi áp dụng quy tắc CSS.

Linux Professional Institute

Chủ đề 034: Lập trình JavaScript

034.1 Cú pháp và Việc Thực thi trong JavaScript

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 034.1

Khối lượng

1

Các lĩnh vực kiến thức chính

- Chạy JavaScript trong tài liệu HTML
- Hiểu về cú pháp trong JavaScript
- Thêm chú thích vào mã JavaScript
- Truy cập vào bảng điều khiển JavaScript
- Ghi vào bảng điều khiển JavaScript

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `<script>`, bao gồm các thuộc tính `type` (`` `text/javascript ``) và `src`
- `;`
- `//`, `/* */`
- `console.log`

034.1 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	034 Lập trình JavaScript
Mục tiêu:	034.1 Thực thi và Cú pháp của JavaScript
Bài học:	1 trên 1

Giới thiệu

Các trang web được phát triển bằng ba công nghệ tiêu chuẩn: HTML, CSS và JavaScript. JavaScript là ngôn ngữ lập trình cho phép trình duyệt tự động cập nhật nội dung trang web. JavaScript thường được thực thi bởi cùng một trình duyệt được sử dụng để xem trang web. Điều này có nghĩa là tương tự như CSS và HTML, hành vi của bất kỳ mã nào chúng ta viết cũng có thể sẽ khác nhau giữa các trình duyệt khác nhau. Nhưng hầu hết các trình duyệt phổ biến đều tuân thủ đặc tả ECMAScript. Đây là một tiêu chuẩn thống nhất việc sử dụng JavaScript trong web và sẽ là cơ sở cho bài học này cùng với đặc tả HTML5 dùng để quy định cách JavaScript được đưa vào một trang web để trình duyệt thực thi.

Chạy JavaScript trong Trình duyệt

Để thực thi JavaScript, trình duyệt cần lấy mã trực tiếp như một phần của HTML tạo nên trang web hoặc dưới dạng URL cho biết vị trí để tệp lệnh được thực thi.

Ví dụ sau đây sẽ cho thấy cách để đưa mã trực tiếp vào tệp HTML:

```
<html>
```

```
  <head>
  </head>
  <body>
    <h1>Website Headline</h1>
    <p>Content</p>

    <script>
      console.log('test');
    </script>

  </body>
</html>
```

Mã đã được bao bọc bên trong các thẻ `<script>` và `</script>`. Mọi thứ nằm trong các thẻ này sẽ được trình duyệt thực thi trực tiếp khi tải trang.

Vị trí của phần tử `<script>` trong trang sẽ cho biết khi nào nó sẽ được thực thi. Một tài liệu HTML sẽ được phân tích cú pháp từ trên xuống dưới và trình duyệt sẽ quyết định khi nào sẽ hiển thị các phần tử trên màn hình. Trong ví dụ ở trên, các thẻ `<h1>` và `<p>` của trang web đã được phân tích cú pháp và có khả năng được hiển thị trước khi tệp lệnh chạy. Nếu mã JavaScript trong thẻ `<script>` cần nhiều thời gian để thực thi thì trang vẫn hiển thị mà không gặp bất kỳ sự cố nào. Tuy nhiên, nếu tệp lệnh đã được đặt phía trên các thẻ khác, khách truy cập trang web sẽ phải đợi cho đến khi tệp lệnh thực thi xong trước khi xem được trang. Vì lý do này, các thẻ `<script>` thường được đặt ở một trong hai vị trí:

- Ở cuối của phần thân HTML, để tệp lệnh là phần cuối cùng được thực thi. Việc đặt như vậy khi mã thêm nội dung nào đó vào trang sẽ không thật sự hữu ích nếu không có nội dung. Một ví dụ có thể là việc thêm chức năng vào một nút, nút này phải tồn tại thì chức năng mới có ý nghĩa.
- Bên trong phần tử `<head>` của HTML. Điều này đảm bảo rằng tệp lệnh sẽ được thực thi trước khi phần thân HTML được phân tích cú pháp. Nếu muốn thay đổi hành vi tải của trang hoặc có điều gì đó cần được thực thi trong khi trang vẫn chưa được tải đầy đủ, ta có thể đặt tệp lệnh tại đây. Ngoài ra, nếu có nhiều tệp lệnh phụ thuộc vào một tệp lệnh cụ thể, ta có thể đặt tệp lệnh chung đó lên đầu để đảm bảo rằng nó sẽ được thực thi trước các tệp lệnh khác.

Vì nhiều lý do, bao gồm cả khả năng quản lý, ta nên đặt mã JavaScript vào các tệp riêng biệt nằm ngoài mã HTML. Các tệp JavaScript bên ngoài sẽ được bao gồm bằng cách sử dụng thẻ `<script>` với thuộc tính `src`, như sau:

```
<html>
  <head>
    <script src="/button-interaction.js"></script>
```

```
  </head>
  <body>
  </body>
</html>
```

Thẻ `src` sẽ cho trình duyệt biết vị trí của nguồn của tệp, nghĩa là tệp chứa mã JavaScript. Vị trí ở đây có thể là một tệp trên cùng một máy chủ (như trong ví dụ trên) hoặc bất kỳ URL nào có thể truy cập trên web (chẳng hạn như https://www.lpi.org/example.js). Giá trị của thuộc tính `src` phải tuân theo quy ước tương tự như khi nhập tệp CSS hoặc tệp hình ảnh, tức là nó có thể là giá trị tương đối hoặc tuyệt đối. Khi gặp một thẻ tệp lệnh có thuộc tính `src`, trình duyệt sẽ cố gắng lấy tệp nguồn bằng cách sử dụng yêu cầu HTTP `GET`. Do đó, các tệp ngoại vi phải ở trạng thái có thể truy cập được.

Khi sử dụng thuộc tính `src`, bất kỳ mã hoặc văn bản nào được đặt giữa các thẻ `<script>...</script>` đều sẽ bị bỏ qua theo đặc tả HTML.

```
<html>
  <head>
    <script src="/button-interaction.js">
      console.log("test"); // <-- This is ignored
    </script>
  </head>
  <body>
  </body>
</html>
```

Có các thuộc tính khác mà ta có thể thêm vào thẻ `script` để chỉ định thêm cách trình duyệt nhận và xử lý tệp sau đó như thế nào. Danh sách sau đây sẽ đi vào chi tiết về các thuộc tính quan trọng:

`async`

Có thể được sử dụng trên các thẻ `script` và sẽ hướng dẫn trình duyệt tìm nạp tệp lệnh trong nền để không làm gián đoạn quá trình tải trang. Quá trình tải trang sẽ vẫn bị gián đoạn sau khi trình duyệt nhận được tệp lệnh bởi trình duyệt phải phân tích cú pháp tệp lệnh. Việc này sẽ được thực hiện ngay lập tức sau khi tệp lệnh đã được tìm nạp hoàn toàn. Thuộc tính này là Boolean; do đó, ta chỉ cần viết thẻ như sau mà không cần cung cấp giá trị: `<script async src="/script.js"></script>`.

`defer`

Tương tự như `async`, thuộc tính này sẽ hướng dẫn trình duyệt không chặn quá trình tải trang trong khi tìm nạp tệp lệnh. Thay vào đó, trình duyệt sẽ trì hoãn việc phân tích cú pháp tệp lệnh. Trình duyệt sẽ đợi cho đến khi toàn bộ tài liệu HTML được phân tích cú pháp và chỉ khi đó nó

mới phân tích cú pháp tệp lệnh trước khi thông báo rằng tài liệu đã được tải hoàn toàn. Giống như `async`, `defer` là một thuộc tính Boolean và cũng được sử dụng theo cách tương tự. Vì `defer` cũng có ý nghĩa tương tự như `async` nên ta không nên chỉ định cả hai thẻ cùng với nhau.

NOTE

Khi một trang đã được phân tích cú pháp hoàn toàn, trình duyệt sẽ cho biết rằng nó đã sẵn sàng hiển thị bằng cách kích hoạt sự kiện `DOMContentLoaded`. Khi đó, khách truy cập sẽ có thể xem tài liệu. Do đó, JavaScript được bao gồm bên trong `<head>` sẽ luôn có thể hoạt động trên trang trước khi nó được hiển thị (ngay cả khi ta chỉ định cả thuộc tính `defer`).

`type`

Biểu thị loại tệp lệnh mà trình duyệt sẽ mong đợi trong thẻ, mặc định sẽ là JavaScript (`type="application/javascript"`). Vì vậy, thuộc tính này sẽ không cần thiết khi bao gồm mã JavaScript hoặc trỏ đến tài nguyên JavaScript bằng thẻ `src`. Nói chung, tất cả các loại MIME đều có thể được chỉ định, nhưng chỉ các tệp lệnh được ký hiệu là JavaScript mới được trình duyệt thực thi. Có hai trường hợp sử dụng thực tế cho thuộc tính này: yêu cầu trình duyệt không thực thi tệp lệnh bằng cách đặt `type` thành một giá trị tùy ý như `template` hoặc `other`, hoặc yêu cầu trình duyệt biết rằng tệp lệnh là một mô-đun ES6. Chúng ta sẽ không đề cập đến các mô-đun ES6 trong bài học này.

WARNING

Khi nhiều tệp lệnh có thuộc tính `async`, chúng sẽ được thực thi theo thứ tự hoàn tất tải xuống chứ *không* phải theo thứ tự của các thẻ `script` trong tài liệu. Mặt khác, thuộc tính `defer` sẽ giữ nguyên thứ tự của các thẻ `script`.

Bảng Điều khiển Trình duyệt

Mặc dù thường được thực thi như một phần của trang web, ta cũng có một cách khác để thực thi JavaScript: thông qua bảng điều khiển trình duyệt. Tất cả các trình duyệt máy tính để bàn hiện đại đều sẽ cung cấp một menu mà qua đó ta có thể thực thi mã JavaScript trong công cụ JavaScript của trình duyệt. Điều này thường được thực hiện để kiểm tra mã mới hoặc gỡ lỗi các trang web hiện có.

Có nhiều cách để truy cập bảng điều khiển trình duyệt, tùy thuộc vào trình duyệt. Cách dễ nhất là thông qua các phím tắt. Sau đây là các phím tắt cho một số trình duyệt hiện đang được sử dụng:

Chrome

`Ctrl` + `Shift` + `J` (`Cmd` + `Option` + `J` trên Mac)

Firefox

`Ctrl` + `Shift` + `K` (`Cmd` + `Option` + `K` trên Mac)

Safari

`Ctrl` + `Shift` + `?` (`Cmd` + `Option` + `?` trên Mac)

Ta cũng có thể nhấp chuột phải vào một trang web và chọn tùy chọn “Inspect” (Kiểm tra) hoặc “Inspect Element” (Kiểm tra Phần tử) để mở trình kiểm tra - đây là một công cụ khác của trình duyệt. Khi trình kiểm tra mở ra, một bảng điều khiển mới sẽ xuất hiện. Trong bảng điều khiển này, ta có thể chọn tab “Console” để hiển thị bảng điều khiển trình duyệt.

Sau khi kéo bảng điều khiển lên, ta có thể thực thi JavaScript trên trang bằng cách nhập trực tiếp JavaScript vào trường nhập liệu. Kết quả của bất kỳ mã được thực thi nào cũng sẽ được hiển thị trên một dòng riêng biệt.

Câu lệnh JavaScript

Vì chúng ta đã biết cách thực thi một tệp lệnh, hãy cùng tìm hiểu những khái niệm cơ bản để một tệp lệnh thực sự được thực thi. Tệp lệnh JavaScript là một tập hợp các câu lệnh và khối. Câu lệnh ví dụ là `console.log('test')`. Phần hướng dẫn này sẽ yêu cầu trình duyệt xuất từ `test` ra bảng điều khiển trình duyệt.

Mọi câu lệnh trong JavaScript đều sẽ được kết thúc bằng dấu chấm phẩy (`;`). Điều này sẽ cho trình duyệt biết rằng câu lệnh đã được thực hiện và một câu lệnh mới có thể được bắt đầu. Hãy xem đoạn lệnh sau:

```
var message = "test"; console.log(message);
```

Chúng ta đã viết hai câu lệnh. Mọi câu lệnh đều được kết thúc bằng dấu chấm phẩy hoặc ở cuối tệp lệnh. Vì để cho dễ đọc, chúng ta có thể đặt các câu lệnh trên các dòng riêng biệt. Bằng cách này, đoạn lệnh cũng có thể được viết là:

```
var message = "test";
console.log(message);
```

Điều này có thể thực hiện được vì tất cả các khoảng trắng giữa các câu lệnh như khoảng trắng, dấu xuống dòng hoặc tab đều sẽ bị bỏ qua. Khoảng trắng cũng thường được đặt giữa các từ khóa riêng lẻ trong các câu lệnh, nhưng điều này sẽ được giải thích rõ hơn trong bài học sắp tới. Các câu lệnh cũng có thể được để trống hoặc chỉ bao gồm khoảng trắng.

Nếu một câu lệnh không hợp lệ vì nó chưa được kết thúc bằng dấu chấm phẩy thì ECMAScript sẽ cố gắng tự động chèn các dấu chấm phẩy thích hợp dựa trên một bộ quy tắc phức tạp. Quy tắc quan trọng nhất là nếu một câu lệnh không hợp lệ bao gồm hai câu lệnh hợp lệ được phân tách

bằng một dòng mới, hãy chèn dấu chấm phẩy vào dòng mới. Ví dụ: đoạn mã sau không tạo thành một câu lệnh hợp lệ:

```
console.log("hello")
console.log("world")
```

Nhưng một trình duyệt hiện đại sẽ tự động thực thi nó như thể nó có các dấu chấm phẩy thích hợp:

```
console.log("hello");
console.log("world");
```

Như vậy, ta có thể bỏ qua dấu chấm phẩy trong một số trường hợp nhất định. Tuy nhiên, vì các quy tắc chèn dấu chấm phẩy tự động rất phức tạp, chúng ta nên luôn luôn chấm dứt câu lệnh của mình đúng cách để tránh các lỗi không mong muốn.

Chú thích trong JavaScript

Các đoạn mã lớn sẽ có thể trở nên rất phức tạp. Có thể chúng ta sẽ muốn chú thích về những gì mình đang viết để khiến cho đoạn mã trở nên dễ đọc hơn, cho một ai đó khác hoặc cho chính bản thân mình trong tương lai. Ngoài ra, có thể ta sẽ muốn đưa thông tin meta vào tệp lệnh, chẳng hạn như thông tin bản quyền hoặc thông tin về thời điểm và lý do viết tệp lệnh.

Để có thể chèn các thông tin meta như vậy, JavaScript có hỗ trợ việc *chú thích*. Một nhà phát triển có thể chèn các ký tự đặc biệt trong tệp lệnh để biểu thị một số phần nhất định của tệp lệnh dưới dạng một chú thích, phần này sẽ bị bỏ qua khi thực thi. Sau đây là một phiên bản được chú thích tương đối cụ thể của tệp lệnh mà chúng ta đã thấy ở trên.

```
/*
 This script was written by the author of this lesson in May, 2020.
 It has exactly the same effect as the previous script, but includes comments.
*/

// First, we define a message.
var message = "test";

console.log(message); // Then, we output the message to the console.
```

Chú thích không phải là câu lệnh và không cần kết thúc bằng dấu chấm phẩy. Thay vào đó, chúng

tuân theo các quy tắc chấm dứt của riêng mình, tùy thuộc vào cách viết. Có hai cách để viết chú thích trong JavaScript:

Chú thích nhiều dòng

Sử dụng `/*` và `*/` để báo hiệu bắt đầu và kết thúc chú thích nhiều dòng. Mọi thứ sau `/*` cho đến khi `*/` xuất hiện lần đầu tiên đều sẽ bị bỏ qua. Loại chú thích này thường được sử dụng để mở rộng nhiều dòng, nhưng nó cũng có thể được sử dụng cho các dòng đơn lẻ hoặc thậm chí là trong một dòng như sau:

```
console.log(/* what we want to log: */ "hello world")
```

Vì mục tiêu của chú thích nói chung là để tăng khả năng đọc của tệp lệnh, ta nên tránh sử dụng kiểu chú thích này trong một dòng.

Chú thích một dòng

Sử dụng `//` (hai dấu gạch chéo lên phía trước) để *chú thích* một dòng. Mọi thứ sau dấu gạch chéo kép trên cùng một dòng đều sẽ bị bỏ qua. Trong ví dụ được hiển thị ở trên, mẫu này được sử dụng trước tiên để chú thích toàn bộ dòng. Sau câu lệnh `console.log(message);`, nó sẽ được dùng để viết chú thích trên phần còn lại của dòng.

Nói chung, chú thích một dòng nên được sử dụng cho một dòng và chú thích nhiều dòng nên dùng cho nhiều dòng ngay cả khi ta có thể sử dụng chúng theo những cách khác. Chúng ta nên tránh việc chú thích trong một câu lệnh.

Chú thích cũng có thể được sử dụng để tạm thời xóa các dòng mã như sau:

```
// We temporarily want to use a different message
// var message = "test";
var message = "something else";
```

Bài tập Hướng dẫn

1. Hãy tạo một biến có tên `ColorName` và gán giá trị `RED` cho nó.

2. Tệp lệnh nào sau đây là hợp lệ?

`console.log("hello")` `console.log("world");`	
`console.log("hello");` `console.log("world");`	
`// console.log("hello")` `console.log("world");`	
`console.log("hello");` `console.log("world") //;`	
`console.log("hello"); /*` `console.log("world") */`	

Bài tập Mở rộng

1. Có bao nhiêu câu lệnh JavaScript có thể được viết trên một dòng mà không sử dụng dấu chấm phẩy?

2. Hãy tạo hai biến có tên `x` và `y`, sau đó in tổng của chúng ra bảng điều khiển.

Tóm tắt

Trong bài học này, chúng ta đã học về các cách để thực thi JavaScript và sửa đổi hành vi tải tệp lệnh. Chúng ta cũng đã học về các khái niệm cơ bản của thành phần tệp lệnh và chú thích, đồng thời cũng đã học cách sử dụng lệnh `console.log()`.

HTML được sử dụng trong bài học này:

<script>

Thẻ `script` có thể được sử dụng để chèn JavaScript trực tiếp hoặc bằng cách chỉ định một tệp có thuộc tính `src`. Nó có thể sửa đổi cách tệp lệnh được tải bằng các thuộc tính `async` và `defer`.

Các khái niệm JavaScript được giới thiệu trong bài học này:

`;`

Dấu chấm phẩy được dùng để ngăn cách các câu lệnh. Dấu chấm phẩy đôi khi có thể — nhưng không nên — được bỏ qua.

`//`, `/*…*/`

Chú thích có thể được sử dụng để thêm chú giải hoặc thông tin meta vào tệp lệnh hoặc để chặn việc thực thi các câu lệnh.

`console.log("text")`

Ta có thể sử dụng lệnh `console.log()` để xuất văn bản ra bảng điều khiển của trình duyệt.

Đáp án Bài tập Hướng dẫn

1. Hãy tạo một biến có tên `ColorName` và gán giá trị `RED` cho nó.

```
var ColorName = "RED";
```

2. Tệp lệnh nào sau đây là hợp lệ?

`console.log("hello")` `console.log("world");`	Không hợp lệ: Lệnh `console.log()` đầu tiên không được kết thúc đúng cách và toàn bộ dòng đã không tạo thành một câu lệnh hợp lệ.
`console.log("hello");` `console.log("world");`	Hợp lệ: Mỗi câu lệnh đều đã được chấm dứt một cách hợp lệ.
`// console.log("hello")` `console.log("world");`	Hợp lệ: Toàn bộ mã sẽ bị bỏ qua vì nó là một chú thích.
`console.log("hello");` `console.log("world") //;`	Không hợp lệ: Câu lệnh cuối cùng bị thiếu dấu chấm phẩy. Dấu chấm phẩy ở cuối sẽ bị bỏ qua vì nó là chú thích.
`console.log("hello"); /*` `console.log("world") */`	Hợp lệ: Một câu lệnh hợp lệ được theo sau bởi mã chú thích; mã này sẽ bị bỏ qua.

Đáp án Bài tập Mở rộng

1. Có bao nhiêu câu lệnh JavaScript có thể được viết trên một dòng mà không sử dụng dấu chấm phẩy?

   Nếu chúng ta đang ở phần cuối của tệp lệnh, chúng ta có thể viết một câu lệnh và nó sẽ bị loại bỏ bởi phần kết thúc của tệp. Trái lại, bạn sẽ không thể viết một câu lệnh mà không có dấu chấm phẩy với cú pháp mà bạn đã học.

2. Hãy tạo hai biến có tên `x` và `y`, sau đó in tổng của chúng ra bảng điều khiển.

```
var x = 5;
var y = 10;
console.log(x+y);
```

034.2 Cấu trúc Dữ liệu JavaScript

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 034.2

Khối lượng

3

Các lĩnh vực kiến thức chính

- Định nghĩa và cách sử dụng biến và hằng
- Hiểu về kiểu dữ liệu
- Hiểu về chuyển đổi/cưỡng chế loại
- Hiểu về mảng và đối tượng
- Nhận thức về phạm vi biến

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `=`, `` `- ``, `` `\*` ``, `` `/` ``, `` `%` ``, `` `-- ``, `` `+ ``, `+=`, `-=`, `*=`, `/=`
- `var`, `let`, `const`
- `boolean`, `number`, `string`, `symbol`
- `array`, `object`
- `undefined`, `null`, `NaN`

034.2 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	034 Lập trình JavaScript
Mục tiêu:	034.2 Cấu trúc Dữ liệu trong JavaScript
Bài học:	1 trên 1

Giới thiệu

Giống như ngôn ngữ tự nhiên, ngôn ngữ lập trình đại diện cho hiện thực thông qua các ký hiệu được kết hợp thành các câu lệnh có ý nghĩa. Hiện thực được biểu thị bằng ngôn ngữ lập trình là tài nguyên của máy, chẳng hạn như hoạt động của bộ xử lý, thiết bị và bộ nhớ.

Trong số vô số các ngôn ngữ lập trình, mỗi ngôn ngữ đều áp dụng một mô hình để biểu diễn thông tin. JavaScript áp dụng các quy ước điển hình của ngôn ngữ *cấp cao* (trong đó hầu hết các chi tiết như việc phân bổ bộ nhớ đều được ngầm xác định) cho phép lập trình viên tập trung vào mục đích của tệp lệnh trong ngữ cảnh của ứng dụng.

Ngôn ngữ Cấp cao

Các ngôn ngữ cấp cao cung cấp các quy tắc trừu tượng giúp lập trình viên có thể viết ít mã hơn để diễn đạt một ý tưởng. JavaScript cung cấp các cách thuận tiện để sử dụng bộ nhớ máy tính thông qua các khái niệm lập trình giúp đơn giản hóa việc viết các mã lặp lại và thường là đủ cho mục đích của nhà phát triển web.

NOTE Mặc dù có thể sử dụng các cơ chế chuyên dụng để truy cập bộ nhớ một cách chi tiết

> nhưng các loại dữ liệu đơn giản hơn mà chúng ta sẽ xem xét sau đây lại được sử dụng phổ biến hơn.

Các hoạt động điển hình trong ứng dụng web bao gồm việc yêu cầu dữ liệu thông qua một số lệnh JavaScript và lưu trữ chúng để xử lý và trình bày cho người dùng. Việc lưu trữ này khá linh hoạt trong JavaScript với các định dạng lưu trữ phù hợp cho từng mục đích sử dụng.

Khai báo Hằng và Biến

Việc khai báo các hằng số và biến để chứa dữ liệu là nền tảng của tất cả các ngôn ngữ lập trình. JavaScript áp dụng quy ước của hầu hết các ngôn ngữ lập trình là gán giá trị cho hằng số hoặc biến bằng cú pháp `name = value`. Hằng số hoặc biến ở bên trái sẽ nhận giá trị ở bên phải. Tên hằng số hoặc tên biến phải bắt đầu bằng một chữ cái hoặc dấu gạch dưới.

Loại dữ liệu được lưu trữ trong biến không cần phải được chỉ định vì JavaScript là một ngôn ngữ *gõ động*. Loại của biến sẽ được suy ra từ giá trị được gán cho nó. Tuy nhiên, sẽ thuận tiện hơn khi ta chỉ định các thuộc tính cụ thể trong phần khai báo để đảm bảo được một kết quả như mong đợi.

NOTE TypeScript là một ngôn ngữ lấy cảm hứng từ JavaScript. Nó giống như ngôn ngữ cấp thấp và cho phép ta khai báo các biến cho các loại dữ liệu cụ thể.

Hằng số

Hằng số (constant) là một ký hiệu được gán ngay khi chương trình bắt đầu và sẽ không bao giờ thay đổi. Các hằng số sẽ rất hữu ích trong việc chỉ định các giá trị cố định, chẳng hạn như xác định hằng số `PI` là 3.14159265 hoặc `COMPANY_NAME` để giữ tên công ty của bạn.

Ví dụ như trong một ứng dụng web, có một máy khách nhận thông tin thời tiết từ một máy chủ từ xa. Lập trình viên có thể quyết định rằng địa chỉ máy chủ phải là hằng số bởi nó sẽ không thay đổi trong quá trình thực thi ứng dụng. Tuy nhiên, thông tin nhiệt độ có thể thay đổi với mỗi dữ liệu mới đến từ máy chủ.

Khoảng thời gian giữa (interval) các truy vấn được thực hiện cho máy chủ cũng có thể được xác định bởi một hằng số và có thể được truy vấn từ bất kỳ phần nào của chương trình:

```
const update_interval = 10;

function setup_app(){
  console.log("Update every " + update_interval + "minutes");
}
```

Khi được gọi, hàm `setup_app()` sẽ hiển thị thông báo `Update every 10 minutes` (Cập nhật 10 phút một lần) trên bảng điều khiển. Thuật ngữ `const` được đặt trước tên `update_interval` đảm bảo rằng giá trị của nó sẽ được giữ nguyên trong toàn bộ quá trình thực thi tệp lệnh. Nếu ta muốn đặt lại giá trị của một hằng số, hệ thống sẽ xuất lỗi `TypeError: Assignment to constant variable` (Gán cho biến không đổi).

Biến

Nếu không có thuật ngữ `const`, JavaScript sẽ tự động giả định rằng `update_interval` là một biến (variable) và giá trị của nó sẽ có thể được sửa đổi. Điều này tương đương với việc khai báo biến rõ ràng với `var`:

```
var update_interval;
update_interval = 10;

function setup_app(){
  console.log("Update every " + update_interval + "minutes");
}
```

Hãy lưu ý rằng mặc dù biến `update_interval` được xác định bên ngoài hàm nhưng nó lại được truy cập từ bên trong hàm. Bất kỳ hằng số hoặc biến nào được khai báo bên ngoài các hàm hoặc khối mã được xác định bởi dấu ngoặc nhọn (`{}`) đều sẽ có *phạm vi toàn cục* - nghĩa là nó có thể được truy cập từ bất kỳ phần nào của mã. Điều ngược lại thì lại không đúng: một hằng số hoặc biến được khai báo bên trong một hàm có *phạm vi cục bộ* thì nó chỉ có thể được truy cập từ bên trong chính hàm đó. Các khối mã được phân định bằng dấu ngoặc nhọn (chẳng hạn như các khối được đặt trong cấu trúc quyết định `if` hoặc các vòng lặp `for`) sẽ phân định phạm vi của các hằng số chứ không phải các biến được khai báo là `var`. Ví dụ như mã sau đây sẽ được coi là hợp lệ:

```
var success = true;
if ( success == true )
{
  var message = "Transaction succeeded";
  var retry = 0;
}
else
{
  var message = "Transaction failed";
  var retry = 1;
}
```

```
console.log(message);
```

Câu lệnh `console.log(message)` có thể truy cập vào biến `message` dù nó đã được khai báo trong khối mã của câu lệnh `if`. Điều tương tự sẽ không xảy ra nếu `message` là hằng số như trong ví dụ sau:

```
var success = true;
if ( success == true )
{
  const message = "Transaction succeeded";
  var retry = 0;
}
else
{
  const message = "Transaction failed";
  var retry = 1;
}

console.log(message);
```

Trong trường hợp này, một thông báo lỗi loại `ReferenceError: message is not defined` (Lỗi tham chiếu: thông báo không được xác định) sẽ được đưa ra và tệp lệnh sẽ bị dừng. Mặc dù trông có vẻ giống một hạn chế, việc giới hạn phạm vi của các biến và hằng số có thể giúp ta tránh nhầm lẫn giữa thông tin được xử lý trong phần nội dung của tệp lệnh và trong các khối mã khác nhau của nó. Vì lý do này, các biến được khai báo bằng `let` thay vì `var` cũng sẽ bị giới hạn phạm vi bởi các khối được phân định bằng dấu ngoặc nhọn. Có những khác biệt nhỏ khác giữa việc khai báo một biến với `var` và với `let`, nhưng điều quan trọng nhất là liên quan đến phạm vi của biến (như chúng ta đang thảo luận ở đây).

Các loại Giá trị

Trong hầu hết mọi trường hợp, lập trình viên sẽ không cần phải lo lắng về loại dữ liệu được lưu trữ trong một biến vì JavaScript sẽ tự động xác định nó là một trong các loại *nguyên thủy* trong quá trình gán giá trị cho biến lần đầu. Tuy nhiên, một số thao tác có thể sẽ chỉ dành riêng cho một loại dữ liệu này hay một loại dữ liệu khác và có thể dẫn đến lỗi khi bị sử dụng một cách tùy ý. Ngoài ra, JavaScript cũng cung cấp các loại *có cấu trúc* cho phép ta kết hợp nhiều loại nguyên thủy thành một.

Các loại Nguyên thủy

Các loại nguyên thủy tương ứng với các biến truyền thống chỉ lưu trữ một giá trị. Các loại sẽ được xác định hoàn toàn; vì vậy, toán tử `typeof` có thể được sử dụng để xác định loại giá trị nào sẽ được lưu trữ trong một biến:

```
console.log("Undefined variables are of type", typeof variable);

{
  let variable = true;
  console.log("Value `true` is of type " + typeof variable);
}

{
  let variable = 3.14159265;
  console.log("Value `3.14159265` is of type " + typeof variable);
}

{
  let variable = "Text content";
  console.log("Value `Text content` is of type " + typeof variable);
}

{
  let variable = Symbol();
  console.log("A symbol is of type " + typeof variable);
}
```

Tệp lệnh này sẽ hiển thị trên bảng điều khiển loại biến nào được sử dụng trong từng trường hợp:

```
undefined variables are of type undefined
Value `true` is of type boolean
Value `3.114159265` is of type number
Value `Text content` is of type string
A symbol is of type symbol
```

Hãy lưu ý rằng dòng đầu tiên đã cố gắng tìm một loại biến không được khai báo. Điều này đã khiến biến đã cho được coi là không xác định (`undefined`). Loại ký hiệu (`symbol`) là loại nguyên thủy ít trực quan nhất. Mục đích của nó là cung cấp một tên thuộc tính duy nhất trong một đối tượng khi không cần xác định một tên thuộc tính cụ thể. Một đối tượng là một trong những cấu trúc dữ liệu mà chúng ta sẽ xem xét tiếp theo.

Loại có Cấu trúc

Trong khi các loại nguyên thủy là đủ để viết các tiến trình đơn giản thì việc chỉ sử dụng chúng trong các ứng dụng phức tạp lại có nhiều hạn chế. Ví dụ như một ứng dụng thương mại điện tử sẽ khó viết hơn nhiều bởi lập trình viên sẽ cần tìm cách lưu trữ danh sách các mục và giá trị tương ứng chỉ sử dụng các biến loại nguyên thủy.

Loại có cấu trúc sẽ đơn giản hóa việc nhóm các thông tin có cùng bản chất thành một biến duy nhất. Ví dụ như danh sách các mặt hàng trong giỏ hàng có thể được lưu trữ trong một biến duy nhất kiểu *mảng*:

```
let cart = ['Milk', 'Bread', 'Eggs'];
```

Như được minh họa trong ví dụ, một mảng các mục sẽ được chỉ định bằng dấu ngoặc vuông. Ví dụ đã điền vào mảng ba giá trị chuỗi ký tự, do đó mà nó sử dụng dấu nháy đơn. Các biến cũng có thể được sử dụng như các mục trong một mảng; nhưng trong trường hợp đó, chúng phải được chỉ định mà không có dấu ngoặc kép. Số lượng phần tử trong một mảng có thể được truy vấn bằng thuộc tính độ dài (`length`):

```
let cart = ['Milk', 'Bread', 'Eggs'];
console.log(cart.length);
```

Số `3` sẽ được hiển thị trong đầu ra của bảng điều khiển. Các mục mới có thể được thêm vào mảng bằng phương thức `push()`:

```
cart.push('Candy');
console.log(cart.length);
```

Lần này, số được hiển thị là `4`. Mỗi mục trong danh sách có thể được truy cập thông qua chỉ số số học của nó, bắt đầu bằng `0`:

```
console.log(cart[0]);
console.log(cart[3]);
```

Đầu ra hiển thị trên bảng điều khiển sẽ là:

```
Milk
Candy
```

Giống như việc có thể sử dụng `push()` để thêm một phần tử, ta cũng có thể sử dụng `pop()` để xóa phần tử cuối cùng khỏi một mảng.

Các giá trị được lưu trữ trong một mảng không nhất thiết phải cùng một kiểu (ví dụ như ta có thể lưu trữ số lượng của từng mặt hàng ngay bên cạnh nó). Một danh sách mua sắm giống như danh sách trong ví dụ trên có thể được xây dựng như sau:

```
let cart = ['Milk', 1, 'Bread', 4, 'Eggs', 12, 'Candy', 2];

// Item indexes are even
let item = 2;

// Quantities indexes are odd
let quantity = 3;

console.log("Item: " + cart[item]);
console.log("Quantity: " + cart[quantity]);
```

Đầu ra sẽ hiển thị trên bảng điều khiển sau khi chạy mã này là:

```
Item: Bread
Quantity: 4
```

Như có thể đã nhận thấy, việc kết hợp tên của các mục với số lượng tương ứng của chúng trong một mảng có thể không phải là một ý tưởng hay vì mối quan hệ giữa chúng không rõ ràng trong cấu trúc dữ liệu và sẽ rất dễ bị lỗi (do con người). Ngay cả khi dùng một mảng cho tên và một mảng khác cho số lượng, việc duy trì tính toàn vẹn của danh sách cũng sẽ cần phải rất cẩn thận và sẽ không được hiệu quả. Trong những tình huống này, giải pháp thay thế tốt nhất là sử dụng cấu trúc dữ liệu phù hợp hơn: một *đối tượng*.

Trong JavaScript, cấu trúc dữ liệu kiểu đối tượng sẽ cho phép ta liên kết các đặc tính với một biến. Ngoài ra, không giống như một mảng, các phần tử tạo nên một đối tượng sẽ không có thứ tự cố định. Ví dụ: một mặt hàng trong danh sách mua sắm có thể là một đối tượng có các đặc tính `name` và `quantity` (số lượng):

```
let item = { name: 'Milk', quantity: 1 };
console.log("Item: " + item.name);
console.log("Quantity: " + item.quantity);
```

Ví dụ này cho thấy rằng một đối tượng có thể được xác định bằng dấu ngoặc nhọn (`{}`), trong đó

mỗi cặp đặc tính/giá trị sẽ được phân tách bằng dấu hai chấm, và các đặc tính sẽ được phân tách bằng dấu phẩy. Ta có thể truy cập đăc tính ở định dạng *biến.đặc tính* như trong `item.name` để đọc và gán giá trị mới. Đầu ra hiển thị trên bảng điều khiển sau khi chạy mã này sẽ là:

```
Item: Milk
Quantity: 1
```

Cuối cùng, mỗi đối tượng đại diện cho một mặt hàng có thể được đưa vào mảng danh sách mua sắm. Điều này có thể được thực hiện trực tiếp khi tạo danh sách:

```
let cart = [{ name: 'Milk', quantity: 1 }, { name: 'Bread', quantity: 4 }];
```

Như trước đây, một đối tượng mới đại diện cho một mục có thể được thêm vào mảng sau:

```
cart.push({ name: 'Eggs', quantity: 12 });
```

Các mục trong danh sách hiện sẽ được truy cập theo chỉ mục và tên đặc tính của chúng:

```
console.log("Third item: " + cart[2].name);
console.log(cart[2].name + " quantity: " + cart[2].quantity);
```

Đầu ra hiển thị trên bảng điều khiển sau khi chạy mã này sẽ là:

```
third item: eggs
Eggs quantity: 12
```

Cấu trúc dữ liệu sẽ cho phép lập trình viên giữ cho mã của họ có tổ chức và dễ bảo trì hơn, cho dù là bởi tác giả ban đầu hay bởi các lập trình viên khác trong nhóm. Ngoài ra, nhiều kết quả đầu ra từ các hàm JavaScript sẽ ở dưới dạng có cấu trúc và cần được lập trình viên xử lý đúng cách.

Toán tử

Cho đến nay, chúng ta mới chỉ thấy cách gán giá trị cho các biến mới được tạo. Đơn giản như vậy, bất kỳ chương trình nào cũng sẽ thực hiện một số thao tác khác trên các giá trị của biến. JavaScript cung cấp một số loại *toán tử* có thể tác động trực tiếp lên giá trị của một biến hoặc lưu trữ kết quả của phép toán trong một biến mới.

Hầu hết các toán tử đều hướng tới các phép toán số học. Ví dụ: để tăng số lượng của một mặt hàng

trong danh sách mua sắm, chỉ cần sử dụng toán tử cộng `+`:

```
item.quantity = item.quantity + 1;
```

Đoạn mã sau sẽ in giá trị của `item.quantity` trước và sau khi thực hiện phép cộng. Đừng nhầm lẫn về vai trò của các dấu cộng trong đoạn mã. Các câu lệnh `console.log` sử dụng dấu cộng để kết hợp hai chuỗi.

```
let item = { name: 'Milk', quantity: 1 };
console.log("Item: " + item.name);
console.log("Quantity: " + item.quantity);

item.quantity = item.quantity + 1;
console.log("New quantity: " + item.quantity);
```

Đầu ra hiển thị trên bảng điều khiển sau khi chạy mã này sẽ là:

```
Item: Milk
Quantity: 1
New quantity: 2
```

Hãy lưu ý rằng giá trị được lưu trữ trước đó trong `item.quantity` được sử dụng làm toán hạng của phép cộng: `item.quantity = item.quantity + 1`. Chỉ sau khi thao tác hoàn tất, giá trị trong `item.quantity` mới được cập nhật cùng với kết quả của phép tính. Loại phép toán số học liên quan đến giá trị hiện tại của biến mục tiêu này là khá phổ biến; do đó, ta có các toán tử tốc ký cho phép viết cùng một phép toán ở định dạng rút gọn:

```
item.quantity += 1;
```

Các phép toán cơ bản khác cũng có các toán tử tốc ký tương đương:

- `a = a - b` tương đương với `a -= b`.
- `a = a * b` tương đương với `a *= b`.
- `a = a / b` tương đương với `a /= b`.

Đối với phép cộng và phép trừ, ta có sẵn định dạng thứ ba khi toán hạng thứ hai chỉ có một đơn vị:

- `a = a + 1` tương đương với `a++`.

- `a = a - 1` tương đương với `a--`.

Ta có thể kết hợp nhiều toán tử trong cùng một thao tác và kết quả có thể sẽ được lưu trữ trong một biến mới. Ví dụ: câu lệnh sau tính tổng giá của một mặt hàng cộng với chi phí vận chuyển:

```
let total = item.quantity * 9.99 + 3.15;
```

Thứ tự thực hiện các phép toán sẽ tuân theo thứ tự ưu tiên truyền thống: đầu tiên thực hiện các phép toán nhân và chia, sau đó mới thực hiện các phép toán cộng và trừ. Các toán tử có cùng mức độ ưu tiên được thực thi theo thứ tự xuất hiện trong biểu thức và từ trái sang phải. Để ghi đè thứ tự ưu tiên mặc định, chúng ta sẽ sử dụng dấu ngoặc đơn như trong `a * (b + c)`.

Trong một số trường hợp, kết quả của một phép toán thậm chí không cần được lưu trữ trong một biến. Đây là trường hợp khi ta muốn đánh giá kết quả của một biểu thức trong câu lệnh `if`:

```
if ( item.quantiy % 2 == 0 )
{
  console.log("Quantity for the item is even");
}
else
{
  console.log("Quantity for the item is odd");
}
```

Toán tử `%` (modulo) sẽ trả về số dư của phép chia toán hạng thứ nhất cho toán hạng thứ hai. Trong ví dụ này, câu lệnh `if` sẽ kiểm tra xem số dư của phép chia `item.quantity` cho 2 có bằng 0 hay không, nghĩa là `item.quantity` có phải là bội số của 2 hay không.

Khi một trong các toán hạng của toán tử `+` là một chuỗi thì các toán tử khác sẽ bị *cưỡng chế* thành chuỗi và kết quả sẽ cho ra phép nối các chuỗi. Trong các ví dụ trước, loại phép tính này được sử dụng để nối các chuỗi và biến thành đối số của câu lệnh `console.log`.

Việc chuyển đổi tự động này có thể không phải là hành vi được mong đợi. Ví dụ: một giá trị do người dùng cung cấp trong trường biểu mẫu có thể được xác định là một chuỗi, nhưng thực sự thì nó lại là một giá trị số. Trong những trường hợp như thế này, trước tiên, biến phải được chuyển đổi thành một số bằng hàm `Number()`:

```
sum = Number(value1) + value2;
```

Hơn nữa, điều quan trọng ở đây là phải xác minh rằng người dùng đã cung cấp một giá trị hợp lệ

trước khi tiếp tục thao tác. Trong JavaScript, một biến không có giá trị được gán sẽ chứa giá trị `null`. Điều này cho phép lập trình viên sử dụng câu lệnh quyết định, chẳng hạn như `if ( value1 == null )`, để kiểm tra xem một biến có được gán giá trị hay không, bất kể loại giá trị được gán cho biến đó là gì.

Bài tập Hướng dẫn

1. Mảng là một cấu trúc dữ liệu có trong một số ngôn ngữ lập trình, một số trong đó chỉ cho phép các mảng có các mục cùng một loại. Trong trường hợp của JavaScript, có thể xác định được một mảng với các mục thuộc các loại khác nhau hay không?

2. Dựa trên ví dụ `let item = { name: 'Milk', quantity: 1 }` đối với một đối tượng trong danh sách mua sắm, làm cách nào để khai báo đối tượng này để bao gồm cả giá của mặt hàng?

3. Chỉ với một dòng mã, có những cách nào để cập nhật giá trị của một biến thành một nửa giá trị hiện tại của nó?

Bài tập Mở rộng

1. Trong đoạn mã sau, giá trị nào sẽ được hiển thị trong đầu ra của bảng điều khiển?

```
var value = "Global";

{
  value = "Location";
}

console.log(value);
```

2. Điều gì sẽ xảy ra khi một hoặc nhiều toán hạng tham gia vào phép toán nhân là một chuỗi?

3. Làm cách nào để có thể xóa mục `Eggs` khỏi mảng `cart` được khai báo với `let cart = ['Milk', 'Bread', 'Eggs']`?

Tóm tắt

Bài học này đã nói về cách sử dụng cơ bản của hằng số và biến trong JavaScript. JavaScript là một *ngôn ngữ gõ động*; vì vậy, lập trình viên không cần chỉ định loại biến trước khi gán giá trị cho nó. Tuy nhiên, điều quan trọng là phải biết về các loại nguyên thủy của ngôn ngữ để đảm bảo kết quả chính xác cho các phép toán cơ bản. Hơn nữa, các cấu trúc dữ liệu như mảng và đối tượng sẽ kết hợp các loại nguyên thủy và cho phép lập trình viên xây dựng các biến tổng hợp và phức tạp hơn. Bài học này đã đi qua các khái niệm và quy trình sau:

- Hằng và biến
- Phạm vi của biến
- Khai báo biến với `var` và `let`
- Các loại nguyên thủy
- Toán tử số học
- Mảng và đối tượng
- Cưỡng chế và chuyển đổi kiểu

Đáp án Bài tập Hướng dẫn

1. Mảng là một cấu trúc dữ liệu có trong một số ngôn ngữ lập trình, một số trong đó chỉ cho phép các mảng có các mục cùng một loại. Trong trường hợp của JavaScript, có thể xác định được một mảng với các mục thuộc các loại khác nhau hay không?

   Có. Trong JavaScript, ta có thể xác định mảng với các mục thuộc các loại nguyên thủy khác nhau, chẳng hạn như chuỗi và số.

2. Dựa trên ví dụ `let item = { name: 'Milk', quantity: 1 }` đối với một đối tượng trong danh sách mua sắm, làm cách nào để khai báo đối tượng này để bao gồm cả giá của mặt hàng?

   ```
   let item = { name: 'Milk', quantity: 1, price: 4.99 };
   ```

3. Chỉ với một dòng mã, có những cách nào để cập nhật giá trị của một biến thành một nửa giá trị hiện tại của nó?

   Người ta có thể sử dụng chính biến đó làm toán hạng, `value = value / 2` hoặc toán tử tốc ký `/=`: `value /= 2`.

Đáp án Bài tập Mở rộng

1. Trong đoạn mã sau, giá trị nào sẽ được hiển thị trong đầu ra của bảng điều khiển?

```
var value = "Global";

{
  value = "Location";
}

console.log(value);
```

   `Location`

2. Điều gì sẽ xảy ra khi một hoặc nhiều toán hạng tham gia vào phép toán nhân là một chuỗi?

   JavaScript sẽ gán giá trị `NaN` (Not a Number - Không phải là Số) cho kết quả để cho biết rằng phép toán không hợp lệ.

3. Làm cách nào để có thể xóa mục `Eggs` khỏi mảng `cart` được khai báo với `let cart = ['Milk', 'Bread', 'Eggs']`?

   Mảng trong Javascript có phương thức `pop()` để loại bỏ mục cuối cùng trong danh sách: `cart.pop()`.

034.3 Cấu trúc và Chức năng Điều khiển trong JavaScript

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 034.3

Khối lượng

4

Các lĩnh vực kiến thức chính

- Hiểu về các giá trị thật và giả
- Hiểu về các toán tử so sánh
- Hiểu về sự khác biệt giữa so sánh lỏng lẻo và so sánh nghiêm ngặt
- Sử dụng điều kiện
- Sử dụng vòng lặp
- Xác định các chức năng tùy chỉnh

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `if`, `else if`, `else`
- `switch`, `case`, `break`
- `for`, `while`, `break`, `continue`
- `function`, `return`
- `==`, `!=`, `<`, `<=`, `>`, `>=`
- `===`, `!==`

034.3 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	034 Lập trình JavaScript
Mục tiêu:	034.3 Cấu trúc và Hàm Điều khiển trong JavaScript
Bài học:	1 trên 2

Giới thiệu

Giống như bất kỳ mootj ngôn ngữ lập trình nào khác, JavaScript là một tập hợp các câu lệnh cho trình thông dịch lệnh biết phải làm gì theo một thứ tự nhất định. Tuy nhiên, điều này không có nghĩa là mọi câu lệnh chỉ được thực thi một lần hoặc phải được thực thi trong mọi trường hợp. Hầu hết các câu lệnh sẽ chỉ được thực thi khi các điều kiện cụ thể được đáp ứng. Ngay cả khi một tệp lệnh được kích hoạt không đồng bộ bởi các sự kiện độc lập, nó thường sẽ phải kiểm tra một số biến điều khiển để tìm ra phần mã phù hợp để chạy.

Câu lệnh If

Cấu trúc điều khiển đơn giản nhất được cung cấp bởi lệnh `if`. Lệnh này sẽ thực thi câu lệnh ngay sau nó nếu điều kiện được chỉ định là đúng. JavaScript coi các điều kiện là đúng khi giá trị được xác định là khác không (0). Bất cứ điều gì bên trong dấu ngoặc đơn sau từ `if` (khoảng trắng sẽ được bỏ qua) sẽ được hiểu là một điều kiện. Trong ví dụ sau, chữ số `1` chính là điều kiện:

```
if ( 1 ) console.log("1 is always true");
```

Tropng ví dụ này, số `1` đã được viết rõ ràng trong điều kiện; do đó, nó được coi là một giá trị không đổi (giá trị này sẽ không thay đổi trong suốt quá trình thực thi tệp lệnh) và sẽ luôn cho kết quả đúng khi được sử dụng làm biểu thức điều kiện. Từ `true` (không có dấu trích dẫn) cũng có thể được sử dụng thay cho `1` vì nó cũng được ngôn ngữ coi là giá trị đúng theo nghĩa đen. Lệnh `console.log` sẽ in các đối số của nó trong *cửa sổ giao diện điều khiển* của trình duyệt.

TIP

Bảng điều khiển trình duyệt đã hiển thị lỗi, cảnh báo và thông báo thông tin được gửi qua lệnh `console.log` trong JavaScript. Trên Chrome, tổ hợp phím `Ctrl` + `Shift` + `J` (`Cmd` + `Option` + `J` trên Mac) sẽ mở bảng điều khiển. Trên Firefox, tổ hợp phím `Ctrl` + `Shift` + `K` (`Cmd` + `Option` + `K` trên Mac) sẽ mở tab bảng điều khiển trong phần công cụ của nhà phát triển.

Mặc dù đúng về mặt cú pháp, việc sử dụng các biểu thức hằng số trong các điều kiện thường không hữu ích lắm. Trong một ứng dụng thực tế, có thể ta sẽ muốn kiểm tra tính đúng đắn của một biến:

```
let my_number = 3;
if ( my_number ) console.log("The value of my_number is", my_number, "and it yields true");
```

Giá trị được gán cho biến `my_number` (3) là khác không; vì vậy, giá trị này trả về giá trị đúng. Nhưng ví dụ này không phải là cách sử dụng phổ biến bởi ta sẽ hiếm khi cần kiểm tra xem một số có bằng 0 hay không. Việc so sánh một giá trị này với một giá trị khác và kiểm tra xem kết quả có đúng hay không thường phổ biến hơn rất nhiều:

```
let my_number = 3;
if ( my_number == 3 ) console.log("The value of my_number is", my_number, "indeed");
```

Toán tử so sánh bằng kép được sử dụng vì toán tử bằng đơn đã được định nghĩa là toán tử gán. Giá trị ở mỗi bên của toán tử được gọi là *toán hạng*. Thứ tự của toán hạng không quá quan trọng và bất kỳ biểu thức nào trả về giá trị đều có thể là toán hạng. Dưới đây là danh sách các toán tử so sánh có sẵn khác:

`value1 == value2`

Đúng nếu `value1` bằng với `value2`.

`value1 != value2`

Đúng nếu `value1` không bằng với `value2`.

`value1 < value2`

Đúng nếu `value1` nhỏ hơn `value2`.

`value1 > value2`

Đúng nếu `value1` lớn hơn `value2`.

`value1 <= value2`

Đúng nếu `value1` nhỏ hơn hoặc bằng `value2`.

`value1 >= value2`

Đúng nếu `value1` lớn hơn hoặc bằng `value2`.

Thông thường, không nhất thiết toán hạng bên trái của toán tử phải là một chuỗi và toán hạng bên phải phải là một số, miễn là JavaScript có thể chuyển đổi biểu thức thành một phép so sánh có ý nghĩa. Vì vậy, chuỗi chứa ký tự 1 sẽ được coi là số 1 khi so sánh với một biến số. Để đảm bảo rằng biểu thức chỉ cho kết quả đúng nếu cả hai toán hạng có cùng loại và giá trị, ta nên sử dụng toán tử nhận dạng nghiêm ngặt `===` thay vì `==`. Tương tự như vậy, toán tử không nhận dạng nghiêm ngặt `!==` sẽ đánh giá là đúng nếu toán hạng đầu tiên không cùng loại và cùng giá trị với toán tử thứ hai.

Cấu trúc điều khiển `if` có thể thực thi theo tuỳ chọn một câu lệnh thay thế khi biểu thức đánh giá là sai :

```
let my_number = 4;
if ( my_number == 3 ) console.log("The value of my_number is 3");
else console.log("The value of my_number is not 3");
```

Lệnh `else` phải theo ngay sau lệnh `if`. Cho đến nay, chúng ta chỉ mới thực hiện một câu lệnh khi điều kiện được đáp ứng. Để thực hiện nhiều hơn một câu lệnh, ta phải đặt chúng trong dấu ngoặc nhọn:

```
let my_number = 4;
if ( my_number == 3 )
{
  console.log("The value of my_number is 3");
  console.log("and this is the second statement in the block");
}
else
{
  console.log("The value of my_number is not 3");
  console.log("and this is the second statement in the block");
}
```

Một nhóm gồm một hoặc nhiều câu lệnh được phân tách bằng một cặp dấu ngoặc nhọn được gọi

là *câu lệnh khối*. Người ta thường sử dụng các câu lệnh khối ngay cả khi chỉ có một lệnh để thực thi, nhằm đảm bảo kiểu mã hóa nhất quán trong toàn bộ tệp lệnh. Hơn nữa, JavaScript không yêu cầu dấu ngoặc nhọn hoặc bất kỳ câu lệnh nào phải nằm trên các dòng riêng biệt, nhưng làm như vậy sẽ cải thiện khả năng đọc và giúp việc bảo trì mã trở nên dễ dàng hơn.

Các cấu trúc điều khiển có thể được lồng vào nhau, nhưng điều quan trọng là không trộn lẫn các dấu ngoặc nhọn mở và đóng của mỗi câu lệnh khối:

```
let my_number = 4;

if ( my_number > 0 )
{
  console.log("The value of my_number is positive");

  if ( my_number % 2 == 0 )
  {
    console.log("and it is an even number");
  }
  else
  {
    console.log("and it is an odd number");
  }
} // end of if ( my_number > 0 )
else
{
  console.log("The value of my_number is less than or equal to 0");
  console.log("and I decided to ignore it");
}
```

Các biểu thức được đánh giá bởi lệnh `if` có thể sẽ phức tạp hơn các phép so sánh đơn giản. Điều này đúng trong trường hợp của ví dụ trước, trong đó biểu thức số học `my_number % 2` được sử dụng bên trong dấu ngoặc đơn của lệnh `if` được lồng vào. Toán tử `%` sẽ trả về số dư sau khi chia số bên trái cho số bên phải. Các toán tử số học như `%` sẽ được ưu tiên hơn các toán tử so sánh như `==`; do đó, phép so sánh sẽ sử dụng kết quả của biểu thức số học làm toán hạng bên trái của nó.

Trong nhiều trường hợp, các cấu trúc điều kiện lồng nhau có thể được kết hợp thành một cấu trúc duy nhất bằng cách sử dụng *các toán tử logic*. Ví dụ: nếu chỉ quan tâm đến các số chẵn dương, ta có thể sử dụng một cấu trúc `if` duy nhất:

```
let my_number = 4;

if ( my_number > 0 && my_number % 2 == 0 )
```

```
{
  console.log("The value of my_number is positive");
  console.log("and it is an even number");
}
else
{
  console.log("The value of my_number either 0, negative");
  console.log("or it is a negative number");
}
```

Toán tử dấu và kép `&&` trong biểu thức được đánh giá là toán tử logic *AND*. Nó chỉ tính là đúng nếu biểu thức bên trái của nó và biểu thức bên phải của nó tính là đúng. Nếu muốn so khớp các số dương hoặc số chẵn, toán tử `||` nên được sử dụng thay thế (viết tắt của toán tử logic *OR*):

```
let my_number = -4;

if ( my_number > 0 || my_number % 2 == 0 )
{
  console.log("The value of my_number is positive");
  console.log("or it is a even negative number");
}
```

Trong ví dụ này, chỉ các số lẻ âm sẽ không khớp với tiêu chí do biểu thức tổng hợp đặt ra. Nếu muốn làm ngược lại, tức là chỉ khớp với các số lẻ âm, hãy thêm toán tử logic *NOT* `!` vào đầu biểu thức:

```
let my_number = -5;

if ( ! ( my_number > 0 || my_number % 2 == 0 ) )
{
  console.log("The value of my_number is an odd negative number");
}
```

Việc thêm dấu ngoặc đơn vào biểu thức tổng hợp sẽ buộc hệ thống tính biểu thức bên trong nó trước. Nếu không có các dấu ngoặc đơn này, toán tử NOT sẽ chỉ áp dụng cho `my_number > 0` và khi đó biểu thức OR sẽ được tính. Toán tử && và `||` được gọi là toán tử logic *nhị phân* vì chúng yêu cầu hai toán hạng. `!` được coi như một toán tử logic *một ngôi* bởi nó chỉ yêu cầu một toán hạng.

Cấu trúc Chuyển

Mặc dù cấu trúc `if` khá là linh hoạt và đủ để kiểm soát luồng của chương trình, cấu trúc điều khiển `switch` có thể sẽ phù hợp hơn khi cần tính các kết quả không phải đúng hoặc sai. Ví dụ: nếu chúng ta muốn thực hiện một hành động riêng biệt cho từng mục được chọn từ danh sách, ta sẽ cần viết cấu trúc `if` cho mỗi một đánh giá:

```
// Available languages: en (English), es (Spanish), pt (Portuguese)
let language = "pt";

// Variable to register whether the language was found in the list
let found = 0;

if ( language == "en" )
{
  found = 1;
  console.log("English");
}

if ( found == 0 && language == "es" )
{
  found = 1;
  console.log("Spanish");
}

if ( found == 0 && language == "pt" )
{
  found = 1;
  console.log("Portuguese");
}

if ( found == 0 )
{
  console.log(language, " is unknown to me");
}
```

Trong ví dụ này, một biến phụ trợ `found` sẽ được sử dụng bởi tất cả các cấu trúc `if` để tìm hiểu xem có xảy ra một kết quả khớp hay không. Trong trường hợp này, cấu trúc `switch` sẽ thực hiện nhiệm vụ tương tự nhưng theo một cách ngắn gọn hơn:

```
switch ( language )
{
```

```
  case "en":
    console.log("English");
    break;
  case "es":
    console.log("Spanish");
    break;
  case "pt":
    console.log("Portuguese");
    break;
  default:
    console.log(language, " not found");
}
```

Mỗi lệnh `case` được lồng vào sẽ được gọi là một *mệnh đề*. Khi một mệnh đề khớp với biểu thức được đánh giá, nó sẽ thực thi các câu lệnh theo sau dấu hai chấm cho đến câu lệnh `break`. Mệnh đề cuối cùng sẽ không cần câu lệnh `break` và thường được sử dụng để thiết lập hành động mặc định khi không có kết quả khớp nào khác xảy ra. Như đã thấy trong ví dụ, cấu trúc `switch` không cần biến phụ trợ.

WARNING `switch` sử dụng phép so sánh nghiêm ngặt để khớp các biểu thức với các mệnh đề `case` của nó.

Nếu có nhiều hơn một mệnh đề kích hoạt cùng một hành động, ta có thể kết hợp hai hoặc nhiều điều kiện `case`:

```
switch ( language )
{
  case "en":
  case "en_US":
  case "en_GB":
    console.log("English");
    break;
  case "es":
    console.log("Spanish");
    break;
  case "pt":
  case "pt_BR":
    console.log("Portuguese");
    break;
  default:
    console.log(language, " not found");
}
```

Vòng lặp

Trong các ví dụ trước, cấu trúc `if` và `switch` rất phù hợp cho các tác vụ chỉ cần chạy một lần sau khi vượt qua một hoặc nhiều bài kiểm tra có điều kiện. Tuy nhiên, có những tình huống khi một tác vụ phải thực thi lặp đi lặp lại — trong cái được gọi là *vòng lặp* (loop) — miễn là biểu thức điều kiện của nó vẫn là đúng. Ví dụ như nếu cần biết một số có phải là số nguyên tố hay không, ta sẽ cần kiểm tra xem liệu phép chia số này cho bất kỳ số nguyên nào lớn hơn 1 và nhỏ hơn chính nó có số dư bằng 0 hay không. Nếu đúng, số đó có thừa số nguyên và nó không phải là số nguyên tố (đây không phải là một phương pháp nghiêm ngặt hoặc hiệu quả để tìm các số nguyên tố - đây chỉ là một ví dụ đơn giản). Các cấu trúc điều khiển vòng lặp sẽ phù hợp hơn cho những trường hợp như vậy, đặc biệt là câu lệnh `while`:

```
// A naive prime number tester

// The number we want to evaluate
let candidate = 231;

// Auxiliary variable
let is_prime = true;

// The first factor to try
let factor = 2;

// Execute the block statement if factor is
// less than candidate and keep doing it
// while factor is less than candidate
while ( factor < candidate )
{

  if ( candidate % factor == 0 )
  {
    // The remainder is zero, so the candidate is not prime
    is_prime = false;
    break;
  }

  // The next factor to try. Simply
  // increment the current factor by one
  factor++;
}

// Display the result in the console window.
// If candidate has no integer factor, then
```

```
// the auxiliary variable is_prime still true
if ( is_prime )
{
  console.log(candidate, "is prime");
}
else
{
  console.log(candidate, "is not prime");
}
```

Câu lệnh khối sau lệnh `while` sẽ thực thi lặp đi lặp lại, miễn là điều kiện `factor < candidate` vẫn đúng. Nó sẽ thực thi ít nhất một lần, miễn là ta khởi tạo biến `factor` với giá trị thấp hơn `candidate` (ứng viên). Cấu trúc `if` được lồng trong cấu trúc `while` sẽ đánh giá xem số dư của `candidate` chia cho `factor` có bằng 0 hay không. Nếu đúng, số ứng viên không phải là số nguyên tố và vòng lặp có thể kết thúc. Câu lệnh `break` sẽ kết thúc vòng lặp và quy trình thực thi sẽ chuyển sang lệnh đầu tiên sau khối `while`.

Hãy lưu ý rằng kết quả của điều kiện được sử dụng bởi câu lệnh `while` phải thay đổi ở mọi vòng lặp; nếu không, câu lệnh khối sẽ lặp đi lặp lại "mãi mãi". Trong ví dụ này, chúng ta tăng biến `factor`–ước số tiếp theo mà chúng ta muốn thử–và nó sẽ đảm bảo vòng lặp sẽ kết thúc tại một thời điểm nào đó.

Phần triển khai đơn giản này của trình kiểm tra số nguyên tố đã hoạt động như mong đợi. Tuy nhiên, chúng ta biết rằng một số không chia hết cho hai sẽ không chia hết cho bất kỳ số chẵn nào khác. Do đó, chúng ta có thể bỏ qua các số chẵn bằng cách thêm một lệnh `if` khác:

```
while ( factor < candidate )
{

  // Skip even factors bigger than two
  if ( factor > 2 && factor % 2 == 0 )
  {
    factor++;
    continue;
  }

  if ( candidate % factor == 0 )
  {
    // The remainder is zero, so the candidate is not prime
    is_prime = false;
    break;
  }
```

```
  // The next number that will divide the candidate
  factor++;
}
```

Câu lệnh `continue` tương tự như câu lệnh `break`, nhưng thay vì kết thúc lần lặp này của vòng lặp, nó sẽ bỏ qua phần còn lại của khối vòng lặp và bắt đầu một lần lặp mới. Hãy lưu ý rằng biến `factor` đã được sửa đổi trước câu lệnh `continue`; nếu không, vòng lặp sẽ lại có kết quả tương tự trong lần lặp tiếp theo. Ví dụ này quá đơn giản và việc bỏ qua một phần của vòng lặp sẽ không thực sự cải thiện hiệu suất của nó; nhưng việc bỏ qua các lệnh dư thừa là rất quan trọng khi ta muốn viết các ứng dụng hiệu quả.

Các vòng lặp được sử dụng phổ biến đến mức chúng tồn tại ở nhiều biến thể khác nhau. Vòng lặp `for` đặc biệt thích hợp để lặp qua các giá trị tuần tự vì nó cho phép ta xác định quy tắc vòng lặp trong một dòng:

```
for ( let factor = 2; factor < candidate; factor++ )
{
  // Skip even factors bigger than two
  if ( factor > 2 && factor % 2 == 0 )
  {
    continue;
  }

  if ( candidate % factor == 0 )
  {
    // The remainder is zero, so the candidate is not prime
    is_prime = false;
    break;
  }
}
```

Ví dụ này sẽ tạo ra kết quả y hệt như ví dụ `while` trước đó, nhưng biểu thức trong ngoặc đơn của nó bao gồm ba phần được phân tách bằng dấu chấm phẩy: khởi tạo (`let factor = 2`), điều kiện vòng lặp (`factor < candidate`) và biểu thức cuối cùng được đánh giá ở cuối mỗi lần lặp lại vòng lặp (`factor+`). Câu lệnh `continue` và `break` cũng áp dụng cho các vòng lặp `for`. Biểu thức cuối cùng trong ngoặc đơn (`factor+`) sẽ được đánh giá sau câu lệnh `continue`; do đó, nó không được nằm trong câu lệnh khối; nếu không, nó sẽ được tăng lên hai lần trước lần lặp tiếp theo.

JavaScript có các loại vòng lặp `for` đặc biệt để xử lý các đối tượng dạng mảng. Ví dụ như chúng ta

có thể kiểm tra một mảng của các biến ứng viên thay vì chỉ một biến:

```
// A naive prime number tester

// The array of numbers we want to evaluate
let candidates = [111, 139, 293, 327];

// Evaluates every candidate in the array
for (candidate of candidates)
{
  // Auxiliary variable
  let is_prime = true;

  for ( let factor = 2; factor < candidate; factor++ )
  {
    // Skip even factors bigger than two
    if ( factor > 2 && factor % 2 == 0 )
    {
      continue;
    }

    if ( candidate % factor == 0 )
    {
      // The remainder is zero, so the candidate is not prime
      is_prime = false;
      break;
    }
  }

  // Display the result in the console window
  if ( is_prime )
  {
    console.log(candidate, "is prime");
  }
  else
  {
    console.log(candidate, "is not prime");
  }
}
```

Câu lệnh `for (candidate of candidate)` sẽ gán một phần tử của mảng `candidates` cho biến `candidate`, sử dụng nó trong câu lệnh khối và lặp lại quy trình cho mọi phần tử trong mảng. Ta không cần khai báo cho riêng `candidate` vì vòng lặp `for` sẽ xác định nó. Cuối cùng, mã từ ví dụ

trước được lồng trong câu lệnh khối mới này, nhưng lần này, nó sẽ kiểm tra từng ứng viên trong mảng.

Bài tập Hướng dẫn

1. Giá trị nào của biến `my_var` phù hợp với điều kiện `my_var > 0 && my_var < 9`?

2. Giá trị nào của biến `my_var` phù hợp với điều kiện `my_var > 0 || my_var < 9`?

3. Vòng lặp `while` sau đây thực thi câu lệnh khối của nó bao nhiêu lần?

```
let i = 0;
while ( 1 )
{
  if ( i == 10 )
  {
    continue;
  }
  i++;
}
```

Bài tập Mở rộng

1. Điều gì sẽ xảy ra nếu toán tử gán bằng `=` được sử dụng thay cho toán tử so sánh bằng `==`?

2. Hãy viết một đoạn mã bằng cách sử dụng cấu trúc điều khiển `if`, trong đó, phép so sánh đẳng thức thông thường sẽ trả về giá trị đúng nhưng phép so sánh đẳng thức nghiêm ngặt sẽ không trả về.

3. Hãy viết lại câu lệnh `for` sau bằng cách sử dụng toán tử logic NOT một ngôi trong điều kiện vòng lặp. Kết quả của điều kiện phải giống nhau.

```
for ( let factor = 2; factor < candidate; factor++ )
```

4. Dựa trên các ví dụ trong bài học này, hãy viết một cấu trúc điều khiển vòng lặp để in ra tất cả các thừa số nguyên của một số đã cho.

Tóm tắt

Bài học này đã trình bày cách sử dụng các cấu trúc điều khiển trong mã JavaScript. Cấu trúc điều kiện và vòng lặp là những yếu tố thiết yếu của bất kỳ mô hình lập trình nào và phát triển web JavaScript cũng không phải là ngoại lệ. Bài học đã đi qua các khái niệm và quy trình sau:

- Câu lệnh `if` và các toán tử so sánh.
- Cách sử dụng cấu trúc `switch` với `case`, `default` và `break`.
- Sự khác biệt giữa so sánh thông thường và so sánh nghiêm ngặt.
- Các cấu trúc điều khiển vòng lặp: `while` và `for`.

Đáp án Bài tập Hướng dẫn

1. Giá trị nào của biến `my_var` phù hợp với điều kiện `my_var > 0 && my_var < 9`?

   Chỉ các số vừa lớn hơn 0 vừa nhỏ hơn 9. Toán tử logic `&&` (AND) yêu cầu cả hai phép so sánh phải khớp với nhau.

2. Giá trị nào của biến `my_var` phù hợp với điều kiện `my_var > 0 || my_var < 9`?

   Việc sử dụng toán tử logic `||` (OR) sẽ khiến bất kỳ số nào cũng khớp vì bất kỳ số nào cũng sẽ lớn hơn 0 hoặc nhỏ hơn 9.

3. Vòng lặp `while` sau đây thực thi câu lệnh khối của nó bao nhiêu lần?

   ```
   let i = 0;
   while ( 1 )
   {
     if ( i == 10 )
     {
       continue;
     }
     i++;
   }
   ```

   Lệnh khối sẽ lặp lại vô thời hạn vì không có điều kiện dừng nào được cung cấp.

Đáp án Bài tập Mở rộng

1. Điều gì sẽ xảy ra nếu toán tử gán bằng `=` được sử dụng thay cho toán tử so sánh bằng `==`?

   Giá trị ở phía bên phải của toán tử sẽ được gán cho biến ở bên trái và kết quả sẽ được chuyển sang phép so sánh; đây có thể không phải là một hành vi được mong muốn.

2. Hãy viết một đoạn mã bằng cách sử dụng cấu trúc điều khiển `if`, trong đó, phép so sánh đẳng thức thông thường sẽ trả về giá trị đúng nhưng phép so sánh đẳng thức nghiêm ngặt sẽ không trả về.

   ```
   let a = "1";
   let b = 1;

   if ( a == b )
   {
     console.log("An ordinary comparison will match.");
   }

   if ( a === b )
   {
     console.log("A strict comparison will not match.");
   }
   ```

3. Hãy viết lại câu lệnh `for` sau bằng cách sử dụng toán tử logic NOT đơn hạng trong điều kiện vòng lặp. Kết quả của điều kiện phải giống nhau.

   ```
   for ( let factor = 2; factor < candidate; factor++ )
   ```

   Đáp án:

   ```
   for ( let factor = 2; ! (factor >= candidate); factor++ )
   ```

4. Dựa trên các ví dụ trong bài học này, hãy viết một cấu trúc điều khiển vòng lặp để in ra tất cả các thừa số nguyên của một số đã cho.

   ```
   for ( let factor = 2; factor <= my_number; factor++ )
   {
     if ( my_number % factor == 0 )
     {
   ```

```
    console.log(factor, " is an integer factor of ", my_number);
  }
}
```

034.3 Bài 2

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	034 Lập trình JavaScript
Mục tiêu:	034.3 Cấu trúc và Hàm Điều khiển trong JavaScript
Bài học:	2 trên 2

Giới thiệu

Ngoài những hàm tích hợp sẵn tiêu chuẩn do ngôn ngữ JavaScript cung cấp, nhà phát triển có thể viết thêm các hàm tùy chỉnh của riêng họ để ánh xạ đầu vào thành đầu ra phù hợp với nhu cầu của ứng dụng. Các hàm tùy chỉnh về cơ bản là một tập hợp các câu lệnh được đóng gói để sử dụng ở những nơi khác như một phần của biểu thức.

Việc sử dụng các hàm là một cách tốt để tránh việc viết mã trùng lặp, bởi chúng có thể được gọi từ các vị trí khác nhau trong suốt chương trình. Ngoài ra, việc nhóm các câu lệnh trong các hàm sẽ tạo điều kiện thuận lợi cho việc liên kết các hành động tùy chỉnh với các sự kiện - đây là khía cạnh trọng tâm trong lập trình JavaScript.

Xác định một Hàm

Khi một chương trình phát triển, việc tổ chức các chức năng của nó sẽ trở nên khó khăn hơn nếu không sử dụng hàm. Mỗi hàm có một phạm vi biến riêng; vì vậy, các biến được xác định bên trong một hàm sẽ chỉ khả dụng bên trong hàm đó. Do đó, chúng sẽ không bị trộn lẫn với các biến từ các hàm khác. Các biến toàn cục vẫn có thể truy cập được từ bên trong các hàm, nhưng cách tốt hơn

để gửi các giá trị đầu vào tới một hàm là thông qua *tham số hàm*. Ví dụ: chúng ta sẽ xây dựng trình xác thực số nguyên tố từ bài học trước:

```
// A naive prime number tester

// The number we want to evaluate
let candidate = 231;

// Auxiliary variable
let is_prime = true;

// Start with the lowest prime number after 1
let factor = 2;

// Keeps evaluating while factor is less than the candidate
while ( factor < candidate )
{

  if ( candidate % factor == 0 )
  {
    // The remainder is zero, so the candidate is not prime
    is_prime = false;
    break;
  }

  // The next number that will divide the candidate
  factor++;
}

// Display the result in the console window
if ( is_prime )
{
  console.log(candidate, "is prime");
}
else
{
  console.log(candidate, "is not prime");
}
```

Nếu sau đó ta cần kiểm tra xem một số có phải là số nguyên tố hay không, ta sẽ cần phải lặp lại mã đã được viết. Phương pháp này không được khuyến khích vì bất kỳ sửa đổi hoặc cải tiến nào đối với mã gốc sẽ cần được sao chép thủ công ở mọi nơi mà mã đã được sao chép vào. Ngoài ra, mã lặp lại sẽ tạo ra gánh nặng cho trình duyệt và mạng và có thể sẽ làm chậm quá trình hiển thị trang

web. Thay vì làm điều này, ta nên chuyển các câu lệnh thích hợp sang một hàm:

```
// A naive prime number tester function
function test_prime(candidate)
{
  // Auxiliary variable
  let is_prime = true;

  // Start with the lowest prime number after 1
  let factor = 2;

  // Keeps evaluating while factor is less than the candidate
  while ( factor < candidate )
  {

    if ( candidate % factor == 0 )
    {
      // The remainder is zero, so the candidate is not prime
      is_prime = false;
      break;
    }

    // The next number that will divide the candidate
    factor++;
  }

  // Send the answer back
  return is_prime;
}
```

Khai báo hàm sẽ bắt đầu bằng câu lệnh `function`, theo sau là tên của hàm và các tham số của hàm. Tên của hàm cũng phải tuân theo các quy tắc giống như tên của biến. Các tham số của hàm - hay còn được gọi là *đối số* hàm (arguments) - được phân tách bằng dấu phẩy và được đặt trong dấu ngoặc đơn.

TIP

Việc liệt kê các đối số trong phần khai báo hàm là không bắt buộc. Các đối số được truyền cho một hàm có thể được truy xuất từ một đối tượng `arguments` giống mảng bên trong hàm đó. Chỉ mục của các đối số bắt đầu từ 0; vì vậy, đối số đầu tiên sẽ là `arguments[0]`, đối số thứ hai sẽ là `arguments[1]`, v.v.

Trong ví dụ này, hàm `test_prime` chỉ có một đối số: đối số `candidate` - ứng viên số nguyên tố sẽ được kiểm tra. Các đối số của hàm hoạt động giống như các biến, nhưng giá trị của chúng sẽ được

gán bởi câu lệnh gọi hàm. Ví dụ: câu lệnh `test_prime(231)` sẽ gọi hàm `test_prime` và gán giá trị 231 cho đối số `candidate`; sau đó, giá trị này sẽ có sẵn bên trong phần thân của hàm giống như một biến thông thường.

Nếu câu lệnh gọi sử dụng các biến đơn giản cho các tham số của hàm, giá trị của chúng sẽ được sao chép vào các đối số của hàm. Quy trình này (việc sao chép các giá trị của tham số được sử dụng trong câu lệnh gọi sang tham số được sử dụng bên trong hàm) được gọi là *truyền đối số theo giá trị*. Bất kỳ sửa đổi nào được thực hiện đối với đối số của hàm sẽ không ảnh hưởng đến biến ban đầu được sử dụng trong câu lệnh gọi. Tuy nhiên, nếu câu lệnh gọi sử dụng các đối tượng phức tạp làm đối số (nghĩa là một đối tượng có các thuộc tính và phương thức gắn liền với nó) cho các tham số của hàm, chúng sẽ được *truyền dưới dạng tham chiếu* và hàm sẽ có thể sửa đổi đối tượng ban đầu được sử dụng trong câu lệnh gọi.

Các đối số được truyền theo giá trị cũng như các biến được khai báo bên trong hàm sẽ không hiển thị với bên ngoài, nghĩa là phạm vi của chúng sẽ bị giới hạn trong phần thân của hàm nơi chúng được khai báo. Tuy nhiên, các hàm thường được sử dụng để tạo ra một số đầu ra có thể hiển thị bên ngoài hàm. Để chia sẻ một giá trị với hàm gọi của nó, một hàm sẽ xác định câu lệnh `return`.

Chẳng hạn, hàm `test_prime` trong ví dụ trước đã trả về giá trị của biến `is_prime`. Do đó, hàm có thể thay thế biến ở bất kỳ đâu mà nó sẽ được sử dụng trong ví dụ ban đầu:

```
// The number we want to evaluate
let candidate = 231;

// Display the result in the console window
if ( test_prime(candidate) )
{
  console.log(candidate, "is prime");
}
else
{
  console.log(candidate, "is not prime");
}
```

Câu lệnh `return`, đúng như tên gọi của nó, sẽ trả về quyền điều khiển cho hàm gọi. Do đó, dù câu lệnh `return` được đặt ở đâu trong hàm thì bất cứ thứ gì theo sau nó cũng sẽ không được thực thi. Một hàm có thể chứa nhiều câu lệnh `return`. Phương pháp này có thể sẽ hữu ích nếu một số lệnh nằm trong các khối câu lệnh có điều kiện để hàm có thể hoặc không thể thực thi một câu lệnh `return` cụ thể trong mỗi lần chạy.

Một số hàm có thể sẽ không trả về giá trị; vì vậy, câu lệnh `return` không phải là bắt buộc. Ví dụ

như các câu lệnh bên trong của hàm sẽ được thực thi bất kể nó có ở đó hay không. Vì vậy, các hàm cũng có thể được sử dụng để thay đổi giá trị của các biến toàn cục hoặc nội dung của các đối tượng được truyền bởi tham chiếu. Mặc dù vậy, nếu hàm không có câu lệnh `return` thì giá trị trả về mặc định của nó sẽ được đặt thành `undefined` - một biến được đặt trước, không có giá trị và không thể ghi được.

Biểu thức Hàm

Trong JavaScript, các hàm cũng chỉ là một loại *đối tượng*. Do đó, hàm cũng có thể được sử dụng trong tệp lệnh giống như biến. Đặc điểm này trở nên rõ ràng hơn khi hàm được khai báo bằng cú pháp thay thế được gọi là *biểu thức hàm*:

```
let test_prime = function(candidate)
{
  // Auxiliary variable
  let is_prime = true;

  // Start with the lowest prime number after 1
  let factor = 2;

  // Keeps evaluating while factor is less than the candidate
  while ( factor < candidate )
  {

    if ( candidate % factor == 0 )
    {
      // The remainder is zero, so the candidate is not prime
      is_prime = false;
      break;
    }

    // The next number that will divide the candidate
    factor++;
  }

  // Send the answer back
  return is_prime;
}
```

Sự khác biệt duy nhất giữa ví dụ này và việc khai báo hàm trong ví dụ trước là ở dòng đầu tiên: `let test_prime = function(candidate)` thay vì `function test_prime(candidate)`. Trong một biểu thức hàm, tên `test_prime` sẽ được sử dụng cho đối tượng chứa hàm chứ không phải để

đặt tên cho chính hàm đó. Các hàm được định nghĩa trong các biểu thức hàm sẽ được gọi giống như các hàm được định nghĩa bằng cú pháp khai báo. Tuy nhiên, trong khi các hàm đã khai báo có thể được gọi trước hoặc sau khi khai báo, các biểu thức hàm chỉ có thể được gọi sau khi khởi tạo. Giống như với biến, việc gọi một hàm được xác định trong một biểu thức trước khi khởi tạo nó sẽ gây ra lỗi tham chiếu.

Đệ quy Hàm

Ngoài việc thực thi các câu lệnh và gọi các hàm được tích hợp sẵn, các hàm tùy chỉnh cũng có thể gọi các hàm tùy chỉnh khác, kể cả chính bản thân nó. Việc gọi một hàm từ chính nó được gọi là *đệ quy hàm*. Tùy thuộc vào loại vấn đề mà bạn đang muốn giải quyết, việc sử dụng các hàm đệ quy có thể sẽ đơn giản hơn việc sử dụng các vòng lặp lồng nhau để thực hiện các tác vụ lặp đi lặp lại.

Cho đến nay, chúng ta đã biết cách sử dụng một hàm để kiểm tra xem một số đã cho có phải là số nguyên tố hay không. Bây giờ, giả sử ta muốn tìm số nguyên tố tiếp theo sau một số đã cho. Ta có thể sử dụng vòng lặp `while` để tăng số lượng "ứng viên" và viết một vòng lặp lồng nhau để tìm các thừa số nguyên cho "ứng viên" đó:

```
// This function returns the next prime number
// after the number given as its only argument
function next_prime(from)
{
  // We are only interested in the positive primes,
  // so we will consider the number 2 as the next
  // prime after any number less than two.
  if ( from < 2 )
  {
    return 2;
  }

  // The number 2 is the only even positive prime,
  // so it will be easier to treat it separately.
  if ( from == 2 )
  {
    return 3;
  }

  // Decrement "from" if it is an even number
  if ( from % 2 == 0 )
  {
    from--;
  }
```

```
  // Start searching for primes greater then 3.

  // The prime candidate is the next odd number
  let candidate = from + 2;

  // "true" keeps the loop going until a prime is found
  while ( true )
  {
    // Auxiliary control variable
    let is_prime = true;

    // "candidate" is an odd number, so the loop will
    // try only the odd factors, starting with 3
    for ( let factor = 3; factor < candidate; factor = factor + 2 )
    {
      if ( candidate % factor == 0 )
      {
        // The remainder is zero, so the candidate is not prime.
        // Test the next candidate
        is_prime = false;
        break;
      }
    }
    // End loop and return candidate if it is prime
    if ( is_prime )
    {
      return candidate;
    }
    // If prime not found yet, try the next odd number
    candidate = candidate + 2;
  }
}

let from = 1024;
console.log("The next prime after", from, "is", next_prime(from));
```

Hãy lưu ý rằng chúng ta cần phải sử dụng một điều kiện không đổi cho vòng lặp `while` (biểu thức `true` bên trong dấu ngoặc đơn) và biến phụ trợ `is_prime` để hệ thống biết khi nào nên dừng vòng lặp. Mặc dù giải pháp này là đúng nhưng việc sử dụng các vòng lặp lồng nhau không được tinh tế bằng việc sử dụng đệ quy để thực hiện cùng một tác vụ:

```
// This function returns the next prime number
```

```
// after the number given as its only argument
function next_prime(from)
{
  // We are only interested in the positive primes,
  // so we will consider the number 2 as the next
  // prime after any number less than two.
  if ( from < 2 )
  {
    return 2;
  }

  // The number 2 is the only even positive prime,
  // so it will be easier to treat it separately.
  if ( from == 2 )
  {
    return 3;
  }

  // Decrement "from" if it is an even number
  if ( from % 2 == 0 )
  {
    from--;
  }

  // Start searching for primes greater then 3.

  // The prime candidate is the next odd number
  let candidate = from + 2;

  // "candidate" is an odd number, so the loop will
  // try only the odd factors, starting with 3
  for ( let factor = 3; factor < candidate; factor = factor + 2 )
  {
    if ( candidate % factor == 0 )
    {
      // The remainder is zero, so the candidate is not prime.
      // Call the next_prime function recursively, this time
      // using the failed candidate as the argument.
      return next_prime(candidate);
    }
  }

  // "candidate" is not divisible by any integer factor other
  // than 1 and itself, therefore it is a prime number.
```

```
  return candidate;
}

let from = 1024;
console.log("The next prime after", from, "is", next_prime(from));
```

Cả hai phiên bản của `next_prime` đều sẽ trả về số nguyên tố tiếp theo sau số được cung cấp làm đối số duy nhất của nó (`from`). Phiên bản đệ quy cũng giống như phiên bản kia, tức là nó sẽ bắt đầu bằng cách kiểm tra các trường hợp đặc biệt (nghĩa là các số nhỏ hơn hoặc bằng hai). Sau đó, nó sẽ tăng ứng viên và bắt đầu tìm kiếm bất kỳ thừa số nguyên nào bằng vòng lặp `for` (hãy lưu ý rằng vòng lặp `while` không còn ở đó nữa). Tại thời điểm đó, số nguyên tố chẵn duy nhất đã được kiểm tra; do đó, ứng viên và các thừa số có thể có của nó đã được tăng thêm hai (một số lẻ cộng với hai là số lẻ tiếp theo).

Chỉ có hai cách thoát khỏi vòng lặp `for` trong ví dụ này. Nếu tất cả các thừa số có thể được kiểm tra và không thừa số nào có số dư bằng 0 khi chia ứng viên, vòng lặp `for` sẽ hoàn tất và hàm sẽ trả về ứng viên là số nguyên tố tiếp theo sau `from`. Mặt khác, nếu `factor` là một thừa số nguyên của `candidate` (`candidate % factor == 0`), giá trị được trả về sẽ đến từ hàm `next_prime` được gọi theo cách đệ quy với `candidate` tăng lên như tham số `from` của nó. Các phiên gọi `next_prime` sẽ được xếp chồng lên nhau cho đến khi một ứng viên cuối cùng xuất hiện không có thừa số nguyên nào. Sau đó, phiên bản `next_prime` cuối cùng chứa số nguyên tố sẽ trả nó về phiên bản `next_prime` trước đó và lần lượt xuống phiên bản `next_prime` đầu tiên. Mặc dù mỗi lần gọi hàm sẽ sử dụng cùng một tên cho các biến, các phiên gọi vẫn sẽ hoàn toàn độc lập với nhau để các biến của chúng được giữ tách biệt trong bộ nhớ.

Bài tập Hướng dẫn

1. Các nhà phát triển có thể giảm thiểu loại tổn hao nào bằng việc sử dụng các hàm?

2. Sự khác biệt giữa các đối số hàm được truyền theo giá trị và các đối số hàm được truyền theo tham chiếu là gì?

3. Giá trị nào sẽ được sử dụng làm đầu ra của một hàm tùy chỉnh nếu nó không có câu lệnh trả về?

Bài tập Mở rộng

1. Nguyên nhân có thể xảy ra của *Lỗi Tham chiếu chưa được xử lý* (Uncaught Reference Error) được phát sinh khi gọi một hàm được khai báo bằng cú pháp *biểu thức* là gì?

2. Hãy viết một hàm có tên `multiples_of` nhận ba đối số: `factor`, `from` và `to`. Bên trong hàm, hãy sử dụng lệnh `console.log()` để in tất cả bội số của `factor` nằm giữa `from` và `to`.

Tóm tắt

Bài học này đã trình bày về cách viết các hàm tùy chỉnh trong mã JavaScript. Các hàm tùy chỉnh cho phép nhà phát triển chia ứng dụng thành "các khối" mã có thể tái sử dụng, giúp việc viết và duy trì các chương trình lớn trở nên dễ dàng hơn. Bài học đã đi qua các khái niệm và quy trình sau:

- Cách xác định hàm tùy chỉnh: khai báo hàm và biểu thức hàm.
- Sử dụng tham số làm đầu vào hàm.
- Sử dụng câu lệnh `return` để đặt đầu ra của hàm.
- Hàm đệ quy.

Đáp án Bài tập Hướng dẫn

1. Các nhà phát triển có thể giảm thiểu loại tổn hao nào bằng việc sử dụng các hàm?

   Các hàm cho phép chúng ta sử dụng lại mã và tạo điều kiện bảo trì mã. Tệp lệnh nhỏ hơn cũng giúp tiết kiệm bộ nhớ và thời gian tải xuống.

2. Sự khác biệt giữa các đối số hàm được truyền theo giá trị và các đối số hàm được truyền theo tham chiếu là gì?

   Khi được truyền theo giá trị, đối số sẽ được sao chép vào hàm và hàm sẽ không thể sửa đổi biến ban đầu trong câu lệnh gọi. Khi được truyền theo tham chiếu, hàm có thể thao tác với biến ban đầu được sử dụng trong câu lệnh gọi.

3. Giá trị nào sẽ được sử dụng làm đầu ra của một hàm tùy chỉnh nếu nó không có câu lệnh trả về?

   Giá trị được trả về sẽ được đặt thành `undefined`.

Đáp án Bài tập Mở rộng

1. Nguyên nhân có thể xảy ra của *Lỗi Tham chiếu chưa được xử lý* (Uncaught Reference Error) được phát sinh khi gọi một hàm được khai báo bằng cú pháp *biểu thức* là gì?

   Hàm đã được gọi từ trước khi khai báo trong tệp tệp lệnh.

2. Hãy viết một hàm có tên `multiples_of` nhận ba đối số: `factor`, `from` và `to`. Bên trong hàm, hãy sử dụng lệnh `console.log()` để in tất cả bội số của `factor` nằm giữa `from` và `to`.

```
function multiples_of(factor, from, to)
{
  for ( let number = from; number <= to; number++ )
  {
    if ( number % factor == 0 )
    {
      console.log(factor, "*", number / factor, "=", number);
    }
  }
}
```

034.4 Thao tác JavaScript đối với Nội dung và Kiểu trang Web

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 034.4

Khối lượng

4

Các lĩnh vực kiến thức chính

- Hiểu về khái niệm và cấu trúc của DOM
- Thay đổi nội dung và thuộc tính của các phần tử HTML thông qua DOM
- Thay đổi kiểu CSS của các phần tử HTML thông qua DOM
- Kích hoạt các hàm JavaScript từ các phần tử HTML

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `document.getElementById()`, `document.getElementsByClassName()`, `document.getElementsByTagName()`, `document.querySelector()`, `document.querySelectorAll()`
- `innerHTML`, `setAttribute()`, `removeAttribute()` properties and methods of DOM elements
- `classList`, `classList.add()`, `classList.remove()`, `classList.toggle()` properties and methods of DOM elements
- Thuộc tính của phần tử HTML `onClick`, `onMouseOver`, `onMouseOut`

Linux Professional Institute

034.4 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	034 Lập trình JavaScript
Mục tiêu:	034.4 Thao tác JavaScript với Nội dung và Thiết kế của Trang Web
Bài học:	1 trên 1

Giới thiệu

HTML, CSS và JavaScript là ba công nghệ riêng biệt kết hợp với nhau trên trang Web. Để tạo ra các trang web có tính động và tương tác, lập trình viên JavaScript phải kết hợp các thành phần từ HTML và CSS trong thời gian chạy - một nhiệm vụ được hỗ trợ rất nhiều thông qua *Mô hình Đối tượng Tài liệu* (DOM).

Tương tác với DOM

DOM là một cấu trúc dữ liệu hoạt động như một giao diện lập trình cho tài liệu mà trong đó, mọi khía cạnh của tài liệu sẽ được biểu diễn dưới dạng một nút trong DOM và mọi thay đổi được thực hiện đối với DOM sẽ ngay lập tức được thể hiện trên tài liệu. Để hiển thị cách sử dụng DOM trong JavaScript, hãy lưu mã HTML sau vào tệp có tên `example.html`:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
```

```
  <title>HTML Manipulation with JavaScript</title>
</head>
<body>

<div class="content" id="content_first">
<p>The dynamic content goes here</p>
</div><!-- #content_first -->

<div class="content" id="content_second" hidden>
<p>Second section</p>
</div><!-- #content_second -->

</body>
</html>
```

DOM sẽ chỉ khả dụng sau khi HTML được tải; vì vậy, hãy viết mã JavaScript sau vào cuối phần thân trang (trước thẻ kết thúc `</body>`):

```
<script>
let body = document.getElementsByTagName("body")[0];
console.log(body.innerHTML);
</script>
```

Đối tượng `document` là phần tử DOM cao nhất, tất cả các phần tử khác đều sẽ phân nhánh từ nó. Phương thức `getElementsByTagName()` sẽ liệt kê tất cả các phần tử giảm dần từ `document` có tên thẻ đã cho. Mặc dù thẻ `body` chỉ được sử dụng một lần trong tài liệu nhưng phương thức `getElementsByTagName()` sẽ luôn trả về một tập hợp giống như mảng gồm các phần tử được tìm thấy; do đó, ta sẽ sử dụng chỉ mục `[0]` để trả về phần tử đầu tiên (và duy nhất) được tìm thấy.

Nội dung HTML

Như đã trình bày trong ví dụ trước, phần tử DOM được trả về bởi `document.getElementsByTagName("body")[0]` đã được gán cho biến `body`. Sau đó, biến `body` có thể được sử dụng để thao tác với phần tử nội dung của trang bởi nó kế thừa tất cả các phương thức và thuộc tính DOM từ phần tử đó. Chẳng hạn như thuộc tính `innerHTML` sẽ chứa toàn bộ mã đánh dấu HTML được viết bên trong phần tử tương ứng; vì vậy, nó có thể được sử dụng để đọc mã đánh dấu bên trong. Lệnh gọi `console.log(body.innerHTML)` sẽ in nội dung bên trong `<body></body>` ra bảng điều khiển web. Biến cũng có thể được sử dụng để thay thế nội dung đó như trong `body.innerHTML = "<p>Content erased</p>"`.

Thay vì thay đổi toàn bộ phần đánh dấu HTML, nếu ta giữ nguyên cấu trúc tài liệu và chỉ tương

tác với các thành phần của nó thì sẽ thực tế hơn. Sau khi trình duyệt hiển thị tài liệu, tất cả các phần tử đều có thể được truy cập bằng các phương thức DOM. Ví dụ: ta có thể liệt kê và truy cập tất cả các phần tử HTML bằng cách sử dụng chuỗi đặc biệt `*` trong phương thức `getElementsByTagName()` của đối tượng `document`:

```
let elements = document.getElementsByTagName("*");
for ( element of elements )
{
  if ( element.id == "content_first" )
  {
    element.innerHTML = "<p>New content</p>";
  }
}
```

Mã này sẽ đặt tất cả các phần tử được tìm thấy trong `document` vào biến `elements`. Biến `elements` là một đối tượng giống như mảng; vì vậy, chúng ta có thể lặp qua từng mục của nó bằng một vòng lặp `for`. Nếu trang HTML nơi mã này chạy có một phần tử với thuộc tính `id` được đặt thành `content_first` (xem trang HTML mẫu được hiển thị ở đầu bài học), câu lệnh `if` sẽ khớp với phần tử đó và nội dung đánh dấu của nó sẽ được đổi thành `<p>New content</p>`. Hãy lưu ý rằng các thuộc tính của một phần tử HTML trong DOM có thể truy cập được bằng cách sử dụng ký hiệu *dấu chấm* của các đặc tính đối tượng JavaScript; do đó, `element.id` là để nói đến thuộc tính `id` của phần tử hiện tại của vòng lặp `for`. Phương thức `getAttribute()` cũng có thể được sử dụng như trong `element.getAttribute("id")`.

Không cần thiết phải lặp qua tất cả các phần tử nếu ta chỉ muốn kiểm tra một tập hợp con của chúng. Ví dụ: phương thức `document.getElementsByClassName()` sẽ giới hạn các phần tử khớp với những phần tử có một hạng cụ thể:

```
let elements = document.getElementsByClassName("content");
for ( element of elements )
{
  if ( element.id == "content_first" )
  {
    element.innerHTML = "<p>New content</p>";
  }
}
```

Tuy nhiên, việc lặp qua nhiều thành phần tài liệu bằng vòng lặp không phải là chiến lược tốt nhất khi ta cần thay đổi một thành phần cụ thể trong trang.

Chọn các Yếu tố cụ thể

JavaScript cung cấp các phương pháp được tối ưu hóa để chọn chính xác phần tử mà ta muốn thao tác. Vòng lặp trước có thể được thay thế hoàn toàn bằng phương thức `document.getElementById()`:

```
let element = document.getElementById("content_first");
element.innerHTML = "<p>New content</p>";
```

Mỗi thuộc tính `id` trong tài liệu đều phải là duy nhất; vì vậy, phương thức `document.getElementById()` sẽ chỉ trả về một đối tượng DOM duy nhất. Ngay cả việc khai báo biến `element` cũng có thể được bỏ qua vì JavaScript cho phép chúng ta xâu chuỗi các phương thức một cách trực tiếp:

```
document.getElementById("content_first").innerHTML = "<p>New content</p>";
```

Phương thức `getElementById()` là phương pháp thích hợp hơn để định vị các phần tử trong DOM vì hiệu suất của nó tốt hơn nhiều so với các phương thức lặp khi làm việc với các tài liệu phức tạp. Tuy nhiên, không phải tất cả các phần tử đều có một ID rõ ràng, và phương thức này sẽ trả về giá trị *null* nếu không có phần tử nào khớp với ID được cung cấp (điều này cũng ngăn việc sử dụng các thuộc tính hoặc hàm xâu chuỗi như `innerHTML` trong ví dụ trên). Ngoài ra, sẽ thực tế hơn nếu ta chỉ gán các thuộc tính ID cho các thành phần chính của trang và sau đó sử dụng bộ chọn CSS để định vị các phần tử con của chúng.

Bộ chọn được giới thiệu trong bài học trước về CSS chính là các mẫu khớp với các phần tử trong DOM. Phương thức `querySelector()` sẽ trả về phần tử khớp đầu tiên trong cây DOM, trong khi `querySelectorAll()` sẽ trả về tất cả các phần tử khớp với bộ chọn đã chỉ định.

Trong ví dụ trước, phương thức `getElementById()` sẽ truy xuất phần tử mang ID `content_first`. Phương thức `querySelector()` cũng có thể thực hiện tác vụ giống như vậy:

```
document.querySelector("#content_first").innerHTML = "<p>New content</p>";
```

Do phương thức `querySelector()` sử dụng cú pháp bộ chọn nên ID được cung cấp phải bắt đầu bằng một dấu thăng (#). Nếu không tìm thấy phần tử phù hợp, phương thức `querySelector()` sẽ trả về *null*.

Trong ví dụ trước, toàn bộ nội dung của div `content_first` đã được thay thế bằng chuỗi văn bản được cung cấp. Chuỗi này có mã HTML trong đó nên đây không được coi là phương pháp hay

nhất. Chúng ta cần cẩn thận khi thêm đánh dấu HTML cố định vào mã JavaScript bởi các yếu tố theo dõi có thể trở nên khó khăn trong việc thay đổi cấu trúc tài liệu tổng thể.

Bộ chọn sẽ không bị giới hạn đối với ID của phần tử. Phần tử `p` bên trong có thể được xử lý trực tiếp:

```
document.querySelector("#content_first p").innerHTML = "New content";
```

Bộ chọn `#content_first p` sẽ chỉ khớp với phần tử `p` đầu tiên bên trong div `#content_first`. Nếu chúng ta muốn thao tác với phần tử đầu tiên thì nó sẽ hoạt động bình thường. Tuy nhiên, chúng ta có thể sẽ muốn thay đổi đoạn thứ hai:

```
<div class="content" id="content_first">
<p>Don't change this paragraph.</p>
<p>The dynamic content goes here.</p>
</div><!-- #content_first -->
```

Trong trường hợp này, chúng ta có thể sử dụng hạng giả `:nth-child(2)` để khớp với phần tử `p` thứ hai:

```
document.querySelector("#content_first p:nth-child(2)").innerHTML = "New content";
```

Số `2` trong `p:nth-child(2)` cho biết đoạn thứ hai đã khớp với bộ chọn. Hãy xem bài học bộ chọn CSS để biết thêm về bộ chọn và cách sử dụng chúng.

Làm việc với Thuộc tính

Khả năng tương tác với DOM của JavaScript không chỉ giới hạn ở việc thao tác với nội dung. Trên thực tế, JavaScript trong trình duyệt được sử dụng phổ biến nhất với mục đích sửa đổi các thuộc tính của các phần tử HTML hiện có.

Giả sử trang ví dụ HTML ban đầu của chúng ta hiện có ba phần nội dung:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>HTML Manipulation with JavaScript</title>
</head>
```

```
<body>

<div class="content" id="content_first" hidden>
<p>First section.</p>
</div><!-- #content_first -->

<div class="content" id="content_second" hidden>
<p>Second section.</p>
</div><!-- #content_second -->

<div class="content" id="content_third" hidden>
<p>Third section.</p>
</div><!-- #content_third -->

</body>
</html>
```

Có thể ta sẽ chỉ muốn hiển thị một trong số chúng tại một thời điểm; do đó mà ta có thuộc tính `hidden` trong tất cả các thẻ `div`. Điều này rất hữu ích, ví dụ như trong việc chỉ hiển thị một hình ảnh từ bộ sưu tập hình ảnh. Để hiển thị một trong số chúng khi tải trang, hãy thêm mã JavaScript sau vào trang:

```
// Which content to show
let content_visible;

switch ( Math.floor(Math.random() * 3) )
{
  case 0:
    content_visible = "#content_first";
  break;
  case 1:
    content_visible = "#content_second";
  break;
  case 2:
    content_visible = "#content_third";
  break;
}

document.querySelector(content_visible).removeAttribute("hidden");
```

Biểu thức được đánh giá bởi câu lệnh `switch` sẽ trả về số 0, 1 hoặc 2 một cách ngẫu nhiên. Sau đó, bộ chọn ID tương ứng sẽ được gán cho biến `content_visible` được sử dụng bởi phương thức

`querySelector(content_visible)`. Lệnh gọi `removeAttribute("hidden")` được xâu chuỗi sẽ loại bỏ thuộc tính `hidden` khỏi phần tử.

Cách tiếp cận ngược lại cũng có thể được áp dụng: tất cả các phần có thể hiển thị ban đầu (không có thuộc tính `hidden`) và chương trình JavaScript sau đó có thể gán thuộc tính `hidden` cho mọi phần ngoại trừ phần trong `content_visible`. Để làm như vậy, ta phải lặp qua tất cả các phần tử div nội dung khác với phần tử đã chọn. Điều này có thể được thực hiện bằng cách sử dụng phương thức `querySelectorAll()`:

```
// Which content to show
let content_visible;

switch ( Math.floor(Math.random() * 3) )
{
  case 0:
    content_visible = "#content_first";
  break;
  case 1:
    content_visible = "#content_second";
  break;
  case 2:
    content_visible = "#content_third";
  break;
}

// Hide all content divs, except content_visible
for ( element of document.querySelectorAll(".content:not("+content_visible+")") )
{
  // Hidden is a boolean attribute, so any value will enable it
  element.setAttribute("hidden", "");
}
```

Nếu biến `content_visible` được đặt thành `#content_first`, bộ chọn sẽ là `.content:not(#content_first)` và nó sẽ đọc tất cả các phần tử có hạng `content` ngoại trừ những phần tử có ID `content_first`. Phương thức `setAttribute()` sẽ thêm hoặc thay đổi các thuộc tính của phần tử HTML. Tham số đầu tiên của nó là tên của thuộc tính và tham số thứ hai là giá trị của thuộc tính.

Tuy nhiên, cách thích hợp để thay đổi giao diện của các phần tử là sử dụng CSS. Trong trường hợp này, chúng ta có thể đặt thuộc tính CSS `display` thành `hidden` và sau đó thay đổi nó thành `block` bằng JavaScript:

```
<style>
div.content { display: none }
</style>

<div class="content" id="content_first">
<p>First section.</p>
</div><!-- #content_first -->

<div class="content" id="content_second">
<p>Second section.</p>
</div><!-- #content_second -->

<div class="content" id="content_third">
<p>Third section.</p>
</div><!-- #content_third -->

<script>
// Which content to show
let content_visible;

switch ( Math.floor(Math.random() * 3) )
{
  case 0:
    content_visible = "#content_first";
  break;
  case 1:
    content_visible = "#content_second";
  break;
  case 2:
    content_visible = "#content_third";
  break;
}
document.querySelector(content_visible).style.display = "block";
</script>
```

Các phương pháp hay tương tự áp dụng cho việc kết hợp các thẻ HTML với JavaScript cũng sẽ áp dụng cho CSS. Do đó, ta không nên viết các thuộc tính CSS trực tiếp bằng mã JavaScript. Thay vào đó, các quy tắc CSS nên được viết riêng khỏi JavaScript. Cách thích hợp để thay thế thiết kế trực quan là chọn một hạng CSS được xác định trước cho phần tử.

Làm việc với các Hạng

Các phần tử có thể có nhiều hơn một hạng liên kết giúp cho việc thêm hoặc bớt trong khi viết các kiểu dáng khi cần thiết trở nên dễ dàng hơn. Sẽ rất khó để thay đổi trực tiếp nhiều thuộc tính CSS trong JavaScript; vì vậy, ta có thể tạo một hạng CSS mới với các thuộc tính đó rồi thêm hạng vào phần tử. Các phần tử DOM có thuộc tính `classList` có thể được sử dụng để xem và thao tác các hạng được gán cho phần tử tương ứng.

Ví dụ: thay vì thay đổi mức độ hiển thị của phần tử, chúng ta có thể tạo một hạng CSS bổ sung để làm nổi bật div `content`:

```
div.content {
  border: 1px solid black;
  opacity: 0.25;
}
div.content.highlight {
  border: 1px solid red;
  opacity: 1;
}
```

Biểu định kiểu này sẽ thêm một đường viền đen mỏng và nửa trong suốt cho tất cả các phần tử có hạng `content`. Chỉ những phần tử cũng có hạng `highlight` mới hoàn toàn trong suốt và có đường viền mỏng màu đỏ. Sau đó, thay vì thay đổi trực tiếp các thuộc tính CSS như đã làm trước đây, ta có thể sử dụng phương thức `classList.add("highlight")` trong phần tử đã chọn:

```
// Which content to highlight
let content_highlight;

switch ( Math.floor(Math.random() * 3) )
{
  case 0:
    content_highlight = "#content_first";
  break;
  case 1:
    content_highlight = "#content_second";
  break;
  case 2:
    content_highlight = "#content_third";
  break;
}

// Highlight the selected div
```

```
document.querySelector(content_highlight).classList.add("highlight");
```

Tất cả các kỹ thuật và ví dụ mà chúng ta đã thấy cho đến nay đều được thực hiện ở cuối quá trình tải trang, nhưng chúng sẽ không bị giới hạn ở giai đoạn này. Trên thực tế, điều khiển JavaScript trở nên hữu ích đối với các nhà phát triển Web là khả năng phản ứng với các sự kiện trên trang mà chúng ta sẽ xem tiếp theo đây.

Trình Xử lý Sự kiện

Tất cả các thành phần trang để hiển thị đều dễ bị ảnh hưởng bởi các sự kiện tương tác, chẳng hạn như nhấp chuột hoặc chuyển động của chính con chuột. Chúng ta có thể liên kết các hành động tùy chỉnh với những sự kiện này, giúp mở rộng đáng kể những gì một tài liệu HTML có thể thực hiện.

Có lẽ phần tử HTML rõ ràng nhất được hưởng lợi từ một hành động được liên kết là phần tử `button`. Để biết nó hoạt động như thế nào, hãy thêm ba nút phía trên phần tử `div` đầu tiên của trang ví dụ:

```
<p>
<button>First</button>
<button>Second</button>
<button>Third</button>
</p>

<div class="content" id="content_first">
<p>First section.</p>
</div><!-- #content_first -->

<div class="content" id="content_second">
<p>Second section.</p>
</div><!-- #content_second -->

<div class="content" id="content_third">
<p>Third section.</p>
</div><!-- #content_third -->
```

Các nút sẽ không tự làm bất cứ việc gì. Nhưng giả sử ta muốn đánh dấu `div` tương ứng với nút được nhấn, chúng ta có thể sử dụng thuộc tính `onClick` để liên kết một hành động với từng nút:

```
<p>
<button onClick="document.getElementById('content_first').classList.toggle('highlight')"
```

```
>First</button>
<button onClick="document.getElementById('content_second').classList.toggle('highlight')"
>Second</button>
<button onClick="document.getElementById('content_third').classList.toggle('highlight')"
>Third</button>
</p>
```

Phương thức `classList.toggle()` sẽ thêm hạng đã chỉ định vào phần tử nếu nó chưa tồn tại và sẽ loại bỏ hạng đó nếu đã có. Nếu chạy ví dụ này, bạn sẽ thấy được rằng ta có thể làm nổi bật nhiều `div` cùng một lúc. Để chỉ đánh dấu `div` tương ứng với nút được nhấn, ta cần phải xóa hạng `highlight` khỏi các thành phần `div` khác. Tuy nhiên, nếu hành động tùy chỉnh quá dài hoặc liên quan đến nhiều dòng mã, việc viết một hàm ngoài thẻ phần tử sẽ thực tế hơn:

```
function highlight(id)
{
  // Remove the "highlight" class from all content elements
  for ( element of document.querySelectorAll(".content") )
  {
    element.classList.remove('highlight');
  }

  // Add the "highlight" class to the corresponding element
  document.getElementById(id).classList.add('highlight');
}
```

Giống như các ví dụ trước, hàm này có thể được đặt bên trong thẻ `<script>` hoặc trong tệp JavaScript bên ngoài được liên kết với tài liệu. Trước tiên, hàm `highlight` sẽ xóa hạng `highlight` khỏi tất cả các phần tử `div` được liên kết với hạng `content`, sau đó thêm hạng `highlight` vào phần tử đã chọn. Sau đó, mỗi nút sẽ gọi hàm này từ thuộc tính `onClick` của nó và sử dụng ID tương ứng làm đối số của hàm:

```
<p>
<button onClick="highlight('content_first')">First</button>
<button onClick="highlight('content_second')">Second</button>
<button onClick="highlight('content_third')">Third</button>
</p>
```

Ngoài thuộc tính `onClick`, chúng ta có thể sử dụng thuộc tính `onMouseOver` (được kích hoạt khi thiết bị trỏ được sử dụng để di chuyển con trỏ lên phần tử), thuộc tính `onMouseOut` (được kích hoạt khi thiết bị trỏ không còn được chứa bên trong phần tử), v.v. Ngoài ra, các trình xử lý sự kiện

cũng không bị hạn chế đối với các nút; vì vậy, bạn có thể chỉ định các hành động tùy chỉnh cho các trình xử lý sự kiện này đối với tất cả các phần tử HTML hiển thị.

Bài tập Hướng dẫn

1. Bằng cách sử dụng phương thức `document.getElementById()`, bạn có thể chèn cụm từ "Dynamic content" vào nội dung bên trong của phần tử có ID là `message` như thế nào?

2. Sự khác biệt giữa việc tham chiếu một phần tử bằng ID của nó bằng cách sử dụng phương thức `document.querySelector()` và thực hiện như vậy thông qua phương thức `document.getElementById()` là gì?

3. Mục đích của phương thức `classList.remove()` là gì?

4. Kết quả của việc sử dụng phương thức `myelement.classList.toggle("active")` là gì nếu `myelement` không có hạng `active` được gán cho nó?

Bài tập Mở rộng

1. Đối số nào của phương thức `document.querySelectorAll()` sẽ làm cho nó bắt chước phương thức `document.getElementsByTagName("input")`?

2. Làm cách nào để có thể sử dụng thuộc tính `classList` để liệt kê tất cả các hạng được liên kết với một phần tử đã cho?

Tóm tắt

Bài học này đã trình bày cách sử dụng JavaScript để thay đổi nội dung HTML và các thuộc tính CSS của chúng bằng cách sử dụng DOM (Mô hình Đối tượng Tài lliệu). Những thay đổi này có thể được kích hoạt bởi các sự kiện của người dùng và sẽ rất hữu ích trong việc tạo giao diện động. Bài học đã đi qua các khái niệm và quy trình sau:

- Cách kiểm tra cấu trúc của tài liệu bằng các phương thức như `document.getElementById()`, `document.getElementsByClassName()`, `document.getElementsByTagName()`, `document.querySelector()` và `document.querySelectorAll()`.
- Cách thay đổi nội dung tài liệu bằng thuộc tính `innerHTML`.
- Cách thêm và sửa đổi các thuộc tính của thành phần trang bằng các phương thức `setAttribute()` và `removeAttribute()`.
- Cách thích hợp để thao tác với các hạng của phần tử bằng cách sử dụng thuộc tính `classList` và mối quan hệ của nó với các thiết kế CSS.
- Cách liên kết các hàm với sự kiện của chuột trong các phần tử cụ thể.

Đáp án Bài tập Hướng dẫn

1. Bằng cách sử dụng phương thức `document.getElementById()`, bạn có thể chèn cụm từ "Dynamic content" vào nội dung bên trong của phần tử có ID là `message` như thế nào?

   Nó có thể được thực hiện với đặc tính `innerHTML`:

   ```
   document.getElementById("message").innerHTML = "Dynamic content"
   ```

2. Sự khác biệt giữa việc tham chiếu một phần tử bằng ID của nó bằng cách sử dụng phương thức `document.querySelector()` và thực hiện như vậy thông qua phương thức `document.getElementById()` là gì?

   ID phải đi kèm với ký tự dấu thăng trong các hàm sử dụng bộ chọn, chẳng hạn như `document.querySelector()`.

3. Mục đích của phương thức `classList.remove()` là gì?

   Nó loại bỏ hạng (có tên được cung cấp làm đối số của hàm) khỏi thuộc tính `class` của phần tử tương ứng.

4. Kết quả của việc sử dụng phương thức `myelement.classList.toggle("active")` là gì nếu `myelement` không có hạng `active` được gán cho nó?

   Phương thức này sẽ gán hạng `active` cho `myelement`.

Đáp án Bài tập Mở rộng

1. Đối số nào của phương thức `document.querySelectorAll()` sẽ làm cho nó bắt chước phương thức `document.getElementsByTagName("input")`?

   Việc sử dụng `document.querySelectorAll("input")` sẽ khớp với tất cả các phần tử `input` trong trang, giống như `document.getElementsByTagName("input")`.

2. Làm cách nào để có thể sử dụng thuộc tính `classList` để liệt kê tất cả các hạng được liên kết với một phần tử đã cho?

   Thuộc tính `classList` là một đối tượng giống như mảng; do đó, vòng lặp `for` có thể được sử dụng để lặp qua tất cả các hạng mà nó chứa.

Linux Professional Institute

Chủ đề 035: Lập trình máy chủ NodeJS

035.1 Khái niệm cơ bản về NodeJS

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 035.1

Khối lượng

1

Các lĩnh vực kiến thức chính

- Hiểu về các khái niệm về Node.js
- Chạy một ứng dụng NodeJS
- Cài đặt các gói NPM

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `node [file.js]`
- `npm init`
- `npm install [module_name]`
- `package.json`
- `node_modules`

035.1 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	035 Lập trình Máy chủ Node.js
Mục tiêu:	035.1 Khái niệm cơ bản về Node.js
Bài:	1 trên 1

Giới thiệu

Node.js là một môi trường thời gian chạy JavaScript chạy mã JavaScript trên máy chủ web – tức web *phụ trợ* (phía máy chủ) – thay vì sử dụng ngôn ngữ thứ hai như Python hoặc Ruby cho các chương trình phía máy chủ. Ngôn ngữ JavaScript đã được sử dụng trong giao diện người dùng hiện đại của các ứng dụng web; nó tương tác với HTML và CSS của giao diện mà người dùng tương tác với trình duyệt web. Việc sử dụng Node.js song song với JavaScript trong trình duyệt sẽ có thể cho phép chúng ta chỉ cần sử dụng một ngôn ngữ lập trình cho toàn bộ ứng dụng.

Lý do chính cho sự tồn tại của Node.js là cách nó xử lý đồng thời nhiều kết nối ở phía máy chủ. Một trong những cách phổ biến nhất mà máy chủ ứng dụng web xử lý các kết nối là thông qua việc thực thi đa tiến trình. Khi mở một ứng dụng trên máy tính để bàn trong máy tính của mình, một tiến trình sẽ bắt đầu và sử dụng rất nhiều tài nguyên. Hãy thử tưởng tượng nếu hàng nghìn người dùng cùng một lúc cũng làm điều tương tự trong một ứng dụng web lớn.

Node.js sẽ tránh được vấn đề này bằng cách sử dụng một thiết kế có tên là *vòng lặp sự kiện*. Đây là một vòng lặp nội bộ liên tục kiểm tra các tác vụ đến đang chờ tính toán. Nhờ việc sử dụng rộng rãi JavaScript và tính phổ biến của công nghệ web, Node.js đã được áp dụng rộng rãi trong cả các ứng dụng lớn và nhỏ. Node.js cũng có nhiều đặc điểm khác khiến nó được áp dụng rộng rãi, chẳng hạn

như việc xử lý đầu vào/đầu ra (I/O) không đồng bộ và không chặn (những đặc điểm này sẽ được giải thích trong các phần sau của bài học này).

Môi trường Node.js sử dụng một công cụ JavaScript để thông dịch và thực thi mã JavaScript ở phía máy chủ hoặc trên máy tính cá nhân. Trong những điều kiện này, mã JavaScript mà lập trình viên viết sẽ được phân tích cú pháp và biên dịch ngay lập tức để thực thi các lệnh máy do mã JavaScript gốc tạo ra.

NOTE Càng đi sâu vào bài học về Node.js này, bạn sẽ càng dễ nhận ra rằng JavaScript của Node.js không hoàn toàn giống với JavaScript chạy trên trình duyệt (tuân theo đặc tả ECMAScript), nhưng cũng lại khá tương đồng.

Bắt đầu

Phần này và các ví dụ sau đây sẽ giả định rằng Node.js đã được cài đặt trên hệ điều hành Linux và người dùng đã có các kỹ năng cơ bản về cách thực thi các lệnh trong cửa sổ dòng lệnh.

Để chạy các ví dụ sau, hãy tạo một thư mục làm việc có tên `node_examples`. Sau đó, hãy mở dòng nhắc lệnh và nhập `node`. Nếu đã cài đặt được Node.js, nó sẽ hiển thị dấu nhắc `>` nơi ta có thể tương tác kiểm tra các lệnh JavaScript. Loại môi trường này được gọi là REPL ("đọc read, đánh giá - evaluate, in - print và lặp - loop"). Hãy nhập nội dung đầu vào sau (hoặc một số câu lệnh JavaScript khác) vào dấu nhắc `>`. Nhấn Enter sau mỗi dòng và môi trường REPL sẽ trả về kết quả hành động của nó:

```
> let array = ['a', 'b', 'c', 'd'];
undefined
> array.map( (element, index) => (`Element: ${element} at index: ${index}`));
[
  'Element: a at index: 0',
  'Element: b at index: 1',
  'Element: c at index: 2',
  'Element: d at index: 3'
]
>
```

Đoạn mã được viết bằng cú pháp ES6 và cung cấp chức năng ánh xạ để lặp qua mảng và in kết quả bằng các bản mẫu chuỗi. Ta có thể viết bất kỳ lệnh nào hợp lệ. Để thoát khỏi cửa sổ dòng lệnh của Node.js, hãy nhập `.exit` và nhớ phải thêm dấu chấm đầu tiên.

Đối với các tệp lệnh và mô-đun dài hơn, sẽ thuận tiện hơn nếu ta sử dụng trình soạn thảo văn bản như VS Code, Emacs hoặc Vim. Ta có thể lưu hai dòng mã vừa hiển thị (với một chút sửa đổi) trong

tệp có tên `start.js`:

```
let array = ['a', 'b', 'c', 'd'];
array.map( (element, index) => ( console.log(`Element: ${element} at index: ${index}`)));
```

Sau đó, bạn có thể chạy tệp lệnh từ vỏ để tạo ra kết quả giống như trước:

```
$ node ./start.js
Element: a at index: 0
Element: b at index: 1
Element: c at index: 2
Element: d at index: 3
```

Trước khi đi sâu vào một số mã khác, chúng ta sẽ tìm hiểu tổng quan về cách thức hoạt động của Node.js sử dụng môi trường thực thi đơn luồng và vòng lặp sự kiện.

Vòng lặp Sự kiện và Luồng Đơn

Thật khó để biết chương trình Node.js sẽ mất bao nhiêu thời gian để xử lý một yêu cầu. Một số yêu cầu ngắn có thể chỉ lặp qua các biến trong bộ nhớ và trả chúng về, trong khi những yêu cầu khác có thể yêu cầu các hoạt động tốn thời gian hơn như mở tệp trên hệ thống hoặc truy vấn tới cơ sở dữ liệu và chờ kết quả. Node.js xử lý những vấn đề này như thế nào? Vòng lặp sự kiện chính là câu trả lời.

Hãy tưởng tượng một đầu bếp đang làm cùng lúc nhiều nhiệm vụ. Nướng bánh là một công việc đòi hỏi nhiều thời gian để lò làm chín bánh. Người đầu bếp không thể ở đó đợi bánh chín rồi mới bắt đầu đi pha cà phê. Thay vào đó, trong khi lò nướng bánh, đầu bếp sẽ phải pha cà phê và làm các công việc khác song song. Nhưng người đầu bếp cũng luôn phải kiểm tra xem đã đến lúc thích hợp để chuyển trọng tâm sang một nhiệm vụ cụ thể khác (pha cà phê) hay phải lấy bánh ra khỏi lò.

Vòng lặp sự kiện cũng giống như người đầu bếp luôn nhận thức được các hoạt động xung quanh mình. Trong Node.js, một "trình kiểm tra sự kiện" luôn kiểm tra các hoạt động đã hoàn thành hoặc đang chờ được xử lý bởi công cụ JavaScript.

Việc sử dụng một quy trình dài và không đồng bộ như thế này sẽ không chặn các thao tác nhanh khác diễn ra sau đó. Điều này là do cơ chế vòng lặp sự kiện sẽ luôn kiểm tra xem nhiệm vụ dài đó đã được thực hiện chưa, chẳng hạn như thao tác I/O. Nếu chưa, Node.js có thể sẽ tiếp tục xử lý các tác vụ khác. Khi tác vụ nền hoàn tất, kết quả sẽ được trả về và ứng dụng trên Node.js có thể sử dụng hàm kích hoạt (gọi lại) để xử lý tiếp đầu ra.

Vì Node.js tránh sử dụng nhiều luồng giống như các môi trường khác nên nó được gọi là *môi trường đơn luồng*; cũng vì vậy mà phương pháp tiếp cận không chặn là một yếu tố tối quan trọng. Đây là lý do tại sao Node.js sử dụng vòng lặp sự kiện. Tuy nhiên, đối với các tác vụ chuyên sâu về điện toán, Node.js không phải là một trong những công cụ tốt nhất. Có những ngôn ngữ và môi trường lập trình khác có thể giải quyết những vấn đề này một cách hiệu quả hơn.

Trong các phần sau, chúng ta sẽ xem xét kỹ hơn về các hàm gọi lại. Hiện tại, hãy hiểu rằng các hàm gọi lại chính là các trình kích hoạt được thực thi khi một thao tác được xác định trước được hoàn thành.

Mô-đun

Cách tốt nhất ở đây là chia nhỏ chức năng phức tạp và các đoạn mã mở rộng thành các phần nhỏ hơn. Việc thực hiện mô đun hóa này sẽ giúp ta tổ chức cơ sở mã tốt hơn, tách bạch việc triển khai và tránh các vấn đề kỹ thuật phức tạp. Để đáp ứng những nhu cầu đó, các lập trình viên sẽ đóng gói các khối mã nguồn để sử dụng cho các phần khác ở bên trong hoặc bên ngoài mã nguồn.

Hãy xét ví dụ về chương trình tính thể tích khối cầu. Hãy mở trình soạn thảo văn bản và tạo một tệp có tên `volumeCalculator.js` chứa đoạn mã JavaScript sau:

```
const sphereVol = (radius) => {
  return 4 / 3 * Math.PI * radius
}

console.log(`A sphere with radius 3 has a ${sphereVol(3)} volume.`);
console.log(`A sphere with radius 6 has a ${sphereVol(6)} volume.`);
```

Bây giờ, hãy thực thi tệp bằng Node:

```
$ node volumeCalculator.js
A sphere with radius 3 has a 113.09733552923254 volume.
A sphere with radius 6 has a 904.7786842338603 volume.
```

Ở đây, một hàm đơn giản đã được sử dụng để tính thể tích của một hình cầu dựa trên bán kính của nó. Hãy tưởng tượng rằng chúng ta cũng cần tính thể tích của hình trụ, hình nón, v.v.: chúng ta sẽ nhanh chóng nhận thấy rằng các hàm cụ thể đó phải được thêm vào tệp `volumeCalculator.js`. Tệp này có thể trở thành một bộ sưu tập khổng lồ của các hàm. Để tổ chức cấu trúc tốt hơn, chúng ta có thể sử dụng ý tưởng về các mô-đun như các gói mã riêng biệt.

Để làm điều đó, hãy tạo một tệp riêng biệt có tên `polyhedrons.js`:

```
const coneVol = (radius, height) => {
  return 1 / 3 * Math.PI * Math.pow(radius, 2) * height;
}

const cylinderVol = (radius, height) => {
  return Math.PI * Math.pow(radius, 2) * height;
}

const sphereVol = (radius) => {
  return 4 / 3 * Math.PI * Math.pow(radius, 3);
}

module.exports = {
  coneVol,
  cylinderVol,
  sphereVol
}
```

Bây giờ, trong tệp `volumeCalculator.js`, hãy xóa mã cũ và thay thế bằng đoạn mã sau:

```
const polyhedrons = require('./polyhedrons.js');

console.log(`A sphere with radius 3 has a ${polyhedrons.sphereVol(3)} volume.`);
console.log(`A cylinder with radius 3 and height 5 has a ${polyhedrons.cylinderVol(3, 5)}
volume.`);
console.log(`A cone with radius 3 and height 5 has a ${polyhedrons.coneVol(3, 5)} volume.`);
```

Và sau đó chạy tên tệp trong môi trường Node.js:

```
$ node volumeCalculator.js
A sphere with radius 3 has a 113.09733552923254 volume.
A cylinder with radius 3 and height 5 has a 141.3716694115407 volume.
A cone with radius 3 and height 5 has a 47.12388980384689 volume.
```

Trong môi trường Node.js, mọi tệp mã nguồn đều được coi là một mô-đun, nhưng từ "module" trong Node.js sẽ biểu thị một gói mã được bao bọc như trong ví dụ trước. Bằng cách sử dụng các mô-đun, chúng ta đã tách bạch các hàm tính thể tích khỏi tệp chính (`volumeCalculator.js`), từ đó giảm kích thước của tệp và giúp việc áp dụng các bài kiểm tra đơn vị trở nên dễ dàng hơn. Đây là một phương pháp hay khi phát triển ứng dụng trong thực tế.

Bây giờ, khi đã biết về cách các mô-đun được sử dụng trong Node.js, chúng ta có thể sử dụng một

trong những công cụ quan trọng nhất: *Trình quản lý Gói Nút* (NPM - Node Package Manager).

Một trong những nhiệm vụ chính của NPM là quản lý, tải xuống và cài đặt các mô-đun bên ngoài vào dự án hoặc bên trong hệ điều hành. Ta có thể khởi tạo kho lưu trữ node bằng lệnh `npm init`.

NPM sẽ hỏi các câu hỏi mặc định về tên kho lưu trữ, phiên bản, mô tả, v.v. Bạn có thể bỏ qua các bước này bằng cách sử dụng `npm init --yes` và lệnh sẽ tự động tạo tệp `package.json` mô tả các thuộc tính của dự án/mô-đun của bạn.

Hãy mở tệp `package.json` trong trình soạn thảo văn bản yêu thích và bạn sẽ thấy tệp JSON chứa các thuộc tính như từ khóa, lệnh tệp lệnh để sử dụng với NPM, tên, v.v.

Một trong những thuộc tính đó là các phần phụ thuộc được cài đặt trong kho lưu trữ cục bộ của bạn. NPM sẽ thêm tên và phiên bản của các thành phần phụ thuộc này vào `package.json` cùng với `package-lock.json` - một tệp khác được NPM sử dụng làm phương án dự phòng trong trường hợp `package.json` bị lỗi.

Hãy nhập nội dung sau vào cửa sổ dòng lệnh:

```
$ npm i dayjs
```

Cờ `i` là lối tắt cho đối số `install`. Nếu có kết nối với internet, NPM sẽ tìm kiếm mô-đun có tên `dayjs` trong kho lưu trữ từ xa của Node.js, tải mô-đun xuống và cài đặt cục bộ. NPM cũng sẽ thêm phần phụ thuộc này vào các tệp `package.json` và `package-lock.json`. Bây giờ, bạn có thể thấy rằng có một thư mục có tên `node_modules` chứa mô-đun đã cài đặt cùng với các mô-đun khác nếu cần. Thư mục `node_modules` chứa mã thực sẽ được sử dụng khi thư viện được nhập và gọi. Tuy nhiên, thư mục này không được lưu trong các hệ thống quản lý phiên bản sử dụng bằng Git vì tệp `package.json` sẽ cung cấp tất cả các phần phụ thuộc được sử dụng. Một người dùng khác có thể lấy tệp `package.json` và chỉ cần chạy `npm install` trong máy của họ và NPM sẽ tạo thư mục `node_modules` ở đó với tất cả các phần phụ thuộc trong `package.json`, từ đó tránh việc kiểm soát phiên bản cho hàng ngàn tệp có sẵn trên kho lưu trữ NPM.

Bây giờ, mô-đun `dayjs` đã được cài đặt trong thư mục cục bộ. Hãy mở bảng điều khiển Node.js và nhập các dòng sau:

```
const dayjs = require('dayjs');
dayjs().format('YYYY MM-DDTHH:mm:ss')
```

Mô-đun `dayjs` được tải bằng từ khóa `require`. Khi gọi một phương thức từ mô-đun, thư viện sẽ lấy ngày giờ hệ thống hiện tại và xuất nó theo định dạng đã được chỉ định:

```
2020 11-22T11:04:36
```

Đây là cơ chế tương tự được sử dụng trong ví dụ trước, trong đó thời gian chạy của Node.js sẽ tải hàm của bên thứ ba vào mã.

Chức năng Máy chủ

Vì Node.js kiểm soát phía máy chủ của ứng dụng web nên một trong những nhiệm vụ cốt lõi của nó là xử lý các yêu cầu HTTP.

Dưới đây là tóm tắt về cách máy chủ web xử lý các yêu cầu HTTP đến. Chức năng của máy chủ là lắng nghe các yêu cầu, xác định phản hồi nhanh nhất có thể mà mỗi yêu cầu cần và trả lại phản hồi đó cho người gửi yêu cầu. Ứng dụng này phải nhận một yêu cầu HTTP đến do người dùng kích hoạt, phân tích cú pháp yêu cầu, thực hiện phép tính, tạo phản hồi và gửi lại. Mô-đun HTTP (chẳng hạn như Node.js) được sử dụng vì nó sẽ đơn giản hóa các bước đó, cho phép lập trình viên web tập trung vào chính ứng dụng.

Hãy xem xét ví dụ sau về việc thực hiện chức năng rất cơ bản này:

```
const http = require('http');
const url = require('url');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  const queryObject = url.parse(req.url,true).query;
  let result = parseInt(queryObject.a) + parseInt(queryObject.b);

  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(`Result: ${result}\n`);
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Hãy lưu những nội dung này trong một tệp có tên `basic_server.js` và chạy nó thông qua lệnh `node`. Cửa sổ dòng lệnh đang chạy Node.js sẽ hiển thị thông báo sau:

```
Server running at http://127.0.0.1:3000/
```

Sau đó, hãy truy cập đường dẫn URL sau trên trình duyệt web: `http://127.0.0.1:3000/numbers?a=2&b=17`

Node.js đang chạy một máy chủ web trong máy tính của bạn và sử dụng hai mô-đun `http` và `url`. Mô-đun `http` sẽ thiết lập một máy chủ HTTP cơ bản, xử lý các yêu cầu web đến và chuyển chúng tới mã ứng dụng đơn giản của chúng ta. Mô-đun URL sẽ phân tích cú pháp các đối số được truyền trong URL, chuyển đổi chúng thành định dạng số nguyên và thực hiện các phép cộng. Sau đó, mô-đun `http` sẽ gửi phản hồi dưới dạng văn bản tới trình duyệt web.

Trong một ứng dụng web thực tế, Node.js thường được sử dụng để xử lý và truy xuất dữ liệu, thường là từ cơ sở dữ liệu và trả lại thông tin đã xử lý cho giao diện người dùng để hiển thị. Nhưng ứng dụng cơ bản trong bài học này cho thấy một cách chính xác cách mà Node.js sử dụng các mô-đun để xử lý các yêu cầu web như một máy chủ web.

Bài tập Hướng dẫn

1. Lý do để sử dụng các mô-đun thay vì viết các hàm đơn giản là gì?

2. Tại sao môi trường Node.js lại trở nên phổ biến như vậy? Hãy chỉ ra một đặc điểm cụ thể.

3. Mục đích của tệp `package.json` là gì?

4. Tại sao không nên lưu và chia sẻ thư mục `node_modules`?

Bài tập Mở rộng

1. Làm cách nào để có thể thực thi các ứng dụng Node.js trên máy tính của mình?

2. Làm cách nào để có thể phân định các tham số trong URL để phân tích cú pháp bên trong máy chủ?

3. Hãy chỉ ra một trường hợp mà trong đó một tác vụ cụ thể có thể gây trở ngại cho ứng dụng Node.js.

4. Bạn sẽ triển khai một tham số để nhân hoặc tính tổng hai số trong ví dụ về máy chủ như thế nào?

Tóm tắt

Bài học này đã cung cấp một cái nhìn tổng quan về môi trường Node.js, các đặc điểm của nó và cách sử dụng nó để triển khai các chương trình đơn giản. Bài học này bao gồm các khái niệm sau:

- Node.js là gì và tại sao nó được sử dụng.
- Cách chạy các chương trình Node.js bằng dòng lệnh.
- Các vòng lặp sự kiện và luồng đơn.
- Các mô-đun.
- Trình quản lý thư viện trong môi trường Node.js (NPM).
- Chức năng máy chủ.

Đáp án Bài tập Hướng dẫn

1. Lý do để sử dụng các mô-đun thay vì viết các hàm đơn giản là gì?

   Bằng cách chọn các mô-đun thay vì các hàm thông thường, lập trình viên sẽ tạo ra một cơ sở mã đơn giản hơn để đọc và bảo trì cũng như để viết các bài kiểm tra tự động.

2. Tại sao môi trường Node.js lại trở nên phổ biến như vậy? Hãy chỉ ra một đặc điểm cụ thể.

   Một lý do là tính linh hoạt của ngôn ngữ JavaScript vốn đã được sử dụng rộng rãi trong phần giao diện người dùng của các ứng dụng web. Node.js sẽ cho phép việc sử dụng chỉ một ngôn ngữ lập trình trong toàn bộ hệ thống.

3. Mục đích của tệp `package.json` là gì?

   Tệp này chứa các siêu dữ liệu cho dự án, chẳng hạn như tên, phiên bản, phần phụ thuộc (thư viện), v.v. Với một tệp `package.json`, những người dùng khác có thể tải xuống và cài đặt chính các thư viện đó cũng như chạy thử nghiệm giống như cách mà người tạo ban đầu đã làm.

4. Tại sao không nên lưu và chia sẻ thư mục `node_modules`?

   Thư mục `node_modules` chứa các phần triển khai của các thư viện có sẵn trong kho lưu trữ từ xa. Vì vậy, cách tốt hơn để chia sẻ các thư viện này là chỉ định chúng trong tệp `package.json` rồi sử dụng NPM để tải xuống các thư viện đó. Phương pháp này đơn giản hơn và ít gặp lỗi hơn vì bạn sẽ không phải theo dõi và duy trì các thư viện cục bộ.

Đáp án Bài tập Mở rộng

1. Làm cách nào để có thể thực thi các ứng dụng Node.js trên máy tính của mình?

   Bạn có thể chạy chúng bằng cách nhập `node PATH/FILE_NAME.js` tại dòng lệnh trong cửa sổ dòng lệnh của mình, thay đổi `PATH` thành đường dẫn của tệp Node.js và thay đổi `FILE_NAME.js` thành tên tệp bạn đã chọn.

2. Làm cách nào để có thể phân định các tham số trong URL để phân tích cú pháp bên trong máy chủ?

   Ký tự `&` được sử dụng để phân định các tham số đó để chúng có thể được trích xuất và phân tích cú pháp trong mã JavaScript.

3. Hãy chỉ ra một trường hợp mà trong đó một tác vụ cụ thể có thể gây trở ngại cho ứng dụng Node.js.

   Node.js không phải là môi trường tốt để chạy các quy trình sử dụng nhiều CPU vì nó sử dụng luồng đơn. Một phiên tính toán số có thể làm chậm và khóa toàn bộ ứng dụng. Nếu cần mô phỏng số, tốt hơn là nên sử dụng các công cụ khác.

4. Bạn sẽ triển khai một tham số để nhân hoặc tính tổng hai số trong ví dụ về máy chủ như thế nào?

   Sử dụng toán tử bậc ba hoặc điều kiện if-else để kiểm tra tham số bổ sung. Nếu tham số là chuỗi `mult` thì nó sẽ trả về tích của các số, nếu không thì trả về tổng. Thay thế mã cũ bằng đoạn mã bên dưới. Khởi động lại máy chủ trong dòng lệnh bằng cách nhấn `Ctrl` + `C` và chạy lại lệnh để khởi động lại máy chủ. Bây giờ, hãy kiểm tra ứng dụng mới bằng cách truy cập URL `http://127.0.0.1:3000/numbers?a=2&b=17&operation=mult` trong trình duyệt của bạn. Nếu bạn bỏ qua hoặc thay đổi tham số cuối cùng, kết quả sẽ là tổng của các số.

```
let result = queryObject.operation == 'mult' ? parseInt(queryObject.a) * parseInt
(queryObject.b) : parseInt(queryObject.a) + parseInt(queryObject.b);
```

035.2 Khái niệm cơ bản về NodeJS Express

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 035.2

Khối lượng

4

Các lĩnh vực kiến thức chính

- Xác định các tuyến đến tệp tĩnh và mẫu EJS
- Cung cấp các tệp tĩnh thông qua Express
- Cung cấp các mẫu EJS thông qua Express
- Tạo các mẫu EJS đơn giản không lồng nhau
- Sử dụng đối tượng yêu cầu để truy cập các tham số HTTP GET và POST và xử lý dữ liệu được gửi qua các biểu mẫu HTML
- Nhận thức về việc xác thực đầu vào của người dùng
- Nhận thức về Viết lệnh chéo (XSS)
- Nhận thức về giả mạo yêu cầu chéo trang (CSRF)

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `express` and `body-parser` modules
- Express app object
- `app.get()`, `app.post()`
- `res.query()`, `res.body()`
- ejs node module
- `res.render()`

- `<% ... %>`, `<%= ... %>`, `<%# ... %>`, `<%- ... %>`
- `views/`

035.2 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	035 Lập trình Máy chủ Node.js
Mục tiêu:	035.1 Khái niệm cơ bản về NodeJS Express
Bài:	1 trên 2

Giới thiệu

Express.js, hay đơn giản là Express, là một khung phổ biến chạy trên Node.js và được sử dụng để viết các máy chủ HTTP xử lý các yêu cầu từ các máy khách ứng dụng web. Express hỗ trợ nhiều cách để đọc các tham số được gửi qua HTTP.

Tệp lệnh Máy chủ Ban đầu

Để minh họa các tính năng cơ bản của Express trong việc nhận và xử lý các yêu cầu, hãy mô phỏng một ứng dụng yêu cầu một số thông tin từ máy chủ, cụ thể là máy chủ trong ví dụ:

- Cung cấp chức năng `echo` để trả về tin nhắn do khách gửi.
- Cho khách biết địa chỉ IP của nó theo yêu cầu.
- Sử dụng cookie để xác định các khách đã biết.

Bước đầu tiên sẽ là tạo tệp JavaScript hoạt động như máy chủ. Hãy sử dụng `npm` và tạo một thư mục có tên `myserver` với tệp JavaScript:

```
$ mkdir myserver
$ cd myserver/
$ npm init
```

Đối với điểm nhập, chúng ta có thể sử dụng tên tệp bất kỳ, nhưng ở đây ta sẽ sử dụng tên tệp mặc định là `index.js`. Danh sách sau đây cho thấy tệp `index.js` cơ bản sẽ được sử dụng làm điểm nhập cho máy chủ:

```
const express = require('express')
const app = express()
const host = "myserver"
const port = 8080

app.get('/', (req, res) => {
  res.send('Request received')
})

app.listen(port, host, () => {
  console.log(`Server ready at http://${host}:${port}`)
})
```

Một số hằng số quan trọng cho cấu hình máy chủ được xác định trong các dòng đầu tiên của tệp lệnh. Hai hằng số đầu tiên là `express` và `app` sẽ tương ứng với mô-đun `express` được bao gồm và một trường hợp của mô-đun này chạy ứng dụng của chúng ta. Chúng ta sẽ thêm các hành động được thực hiện bởi máy chủ vào đối tượng `app`.

Hai hằng số khác là `host` và `port` sẽ xác định máy chủ và cổng giao tiếp được liên kết với máy chủ.

Nếu có một máy chủ có thể truy cập công khai, hãy sử dụng tên của nó thay vì `myserver` làm giá trị của `host`. Nếu bạn không cung cấp tên máy chủ, Express sẽ mặc định là `localhost`, tức máy tính chạy ứng dụng. Trong trường hợp đó, sẽ không có khách bên ngoài nào có thể tiếp cận được chương trình. Việc này có thể sẽ không thành vấn đề trong trường hợp thử nghiệm nhưng sẽ mang lại rất ít giá trị trong quá trình sản xuất.

Cổng cần phải được cung cấp; nếu không, máy chủ sẽ không khởi động.

Tệp lệnh này chỉ đính kèm hai thủ tục vào đối tượng `app`: hành động `app.get()` trả lời các yêu cầu do khách thực hiện thông qua HTTP `GET` và lệnh gọi `app.listen()` để kích hoạt máy chủ và gán cho nó một máy chủ và một cổng.

Để khởi động máy chủ, ta chỉ cần chạy lệnh `node` và cung cấp tên tệp lệnh làm đối số:

```
$ node index.js
```

Ngay khi thông báo `Server ready at http://myserver:8080` xuất hiện, máy chủ đã sẵn sàng nhận yêu cầu từ máy khách HTTP. Yêu cầu có thể được thực hiện từ một trình duyệt trên cùng một máy tính nơi máy chủ đang chạy hoặc từ một máy khác có thể truy cập máy chủ.

Tất cả các chi tiết giao dịch chúng ta thấy ở đây sẽ được hiển thị trong trình duyệt nếu ta mở một cửa sổ cho bảng điều khiển của nhà phát triển. Ngoài ra, lệnh `curl` có thể được sử dụng cho giao tiếp HTTP và cho phép ta kiểm tra chi tiết kết nối một cách dễ dàng hơn. Nếu không quen thuộc với dòng lệnh vỏ, bạn có thể tạo biểu mẫu HTML để gửi yêu cầu đến máy chủ.

Ví dụ sau đây sẽ cho thấy cách sử dụng lệnh `curl` trên dòng lệnh để thực hiện yêu cầu HTTP tới máy chủ mới được triển khai:

```
$ curl http://myserver:8080 -v
* Trying 192.168.1.225:8080...
* TCP_NODELAY set
* Connected to myserver (192.168.1.225) port 8080 (#0)
> GET / HTTP/1.1
> Host: myserver:8080
> User-Agent: curl/7.68.0
>Accept: /
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: text/html; charset=utf-8
< Content-Length: 16
< ETag: W/"10-1WVvDtVyAF0vX9evlsFlfiJTT5c"
< Date: Fri, 02 Jul 2021 14:35:11 GMT
< Connection: keep-alive
<
* Connection #0 to host myserver left intact
Request received
```

Tùy chọn `-v` của lệnh `curl` sẽ hiển thị tất cả các tiêu đề yêu cầu và phản hồi cũng như các thông tin gỡ lỗi khác. Các dòng bắt đầu bằng `>` sẽ cho biết các tiêu đề yêu cầu do máy khách gửi và các dòng bắt đầu bằng `<` sẽ cho biết các tiêu đề phản hồi do máy chủ gửi. Các dòng bắt đầu bằng `*` là thông tin do chính `curl` tạo ra. Nội dung của phản hồi chỉ được hiển thị ở cuối, trong trường hợp

này là dòng `Request received`.

URL của dịch vụ, trong trường hợp này chứa tên máy chủ và cổng (`http://myserver:8080`), sẽ được cung cấp làm đối số cho lệnh `curl`. Bởi vì không có thư mục hoặc tên tệp nào được cung cấp nên chúng được đặt mặc định ở thư mục gốc `/`. Dấu gạch chéo sẽ xuất hiện dưới dạng tệp yêu cầu trong dòng `> GET / HTTP/1.1`, theo sau là tên máy chủ và cổng trong đầu ra.

Ngoài việc hiển thị các tiêu đề kết nối HTTP, lệnh `curl` cũng hỗ trợ phát triển ứng dụng bằng cách cho phép ta gửi dữ liệu đến máy chủ bằng các phương thức HTTP khác nhau và ở các định dạng khác nhau. Tính linh hoạt này giúp gỡ lỗi mọi sự cố và triển khai các tính năng mới trên máy chủ một cách dễ dàng hơn.

Tuyến

Các yêu cầu mà máy khách có thể thực hiện đối với máy chủ phụ thuộc vào việc *tuyến* nào được xác định trong tệp `index.js`. Một tuyến sẽ chỉ định một phương thức HTTP và xác định một *đường dẫn* (chính xác hơn là một URI) mà máy khách có thể yêu cầu.

Cho đến nay, máy chủ chỉ có một tuyến được cấu hình:

```
app.get('/', (req, res) => {
  res.send('Request received')
})
```

Mặc dù chỉ là một tuyến rất đơn giản và chỉ trả lại một thông báo văn bản thuần túy cho máy khách, nó vẫn đủ để xác định các thành phần quan trọng nhất được sử dụng để cấu trúc hầu hết các tuyến:

- Phương thức HTTP được tuyến sử dụng. Trong ví dụ này, phương thức HTTP `GET` được biểu thị bằng đặc tính `get` của đối tượng `app`.
- Đường dẫn được tuyến sử dụng. Khi máy khách không chỉ định đường dẫn cho yêu cầu, máy chủ sẽ sử dụng thư mục gốc - đây là thư mục cơ sở dành cho máy chủ web sử dụng. Một ví dụ ở phần sau trong chương này sử dụng đường dẫn `/echo` tương ứng với yêu cầu được gửi tới `myserver:8080/echo`.
- Hàm được thực thi khi máy chủ nhận được yêu cầu trên tuyến này thường được viết dưới dạng viết tắt là *hàm mũi tên* vì cú pháp `=>` trỏ đến định nghĩa của hàm không tên. Tham số `req` (viết tắt của "request") và tham số `res` (viết tắt của "response") cung cấp thông tin chi tiết về kết nối và được truyền vào hàm bởi chính ứng dụng.

Phương thức POST

Để mở rộng chức năng của máy chủ thử nghiệm, hãy xem cách xác định tuyến cho phương thức HTTP `POST`. Nó được khách sử dụng khi họ cần gửi thêm dữ liệu đến máy chủ ngoài những dữ liệu có trong tiêu đề yêu cầu. Tùy chọn `--data` của lệnh `curl` sẽ tự động gọi phương thức `POST` và sẽ bao gồm nội dung được gửi đến máy chủ qua `POST`. Dòng `POST / HTTP/1.1` trong đầu ra sau đây cho thấy phương thức `POST` đã được sử dụng. Tuy nhiên, máy chủ của chúng ta chỉ xác định một phương thức `GET`; do đó, sẽ xảy ra lỗi khi ta sử dụng `curl` để gửi yêu cầu qua `POST`:

```
$ curl http://myserver:8080/echo --data message="This is the POST request body"
* Trying 192.168.1.225:8080...
* TCP_NODELAY set
* Connected to myserver (192.168.1.225) port 8080 (#0)
> POST / HTTP/1.1
> Host: myserver:8080
> User-Agent: curl/7.68.0
>Accept: /
> Content-Length: 37
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 37 out of 37 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 404 Not Found
< X-Powered-By: Express
< Content-Security-Policy: default-src 'none'
< X-Content-Type-Options: nosniff
< Content-Type: text/html; charset=utf-8
< Content-Length: 140
< Date: Sat, 03 Jul 2021 02:22:45 GMT
< Connection: keep-alive
<
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot POST /</pre>
</body>
</html>
* Connection #0 to host myserver left intact
```

Trong ví dụ trước, việc chạy `curl` với tham số `--data message="This is the POST request body"` cũng tương đương với việc gửi một biểu mẫu chứa trường văn bản có tên `message` và có phần `This is the POST request body`.

Do máy chủ được định cấu hình chỉ với một tuyến cho đường dẫn `/` và tuyến đó chỉ phản hồi với phương thức HTTP `GET` nên tiêu đề phản hồi sẽ có dòng `HTTP/1.1 404 Not Found`. Ngoài ra, Express sẽ tự động tạo một phản hồi HTML ngắn với cảnh báo `Cannot POST`.

Sau khi đã xem cách tạo yêu cầu `POST` thông qua `curl`, hãy viết một chương trình Express có thể xử lý yêu cầu thành công.

Trước tiên, hãy lưu ý rằng trường `Content-Type` trong tiêu đề yêu cầu sẽ cho biết dữ liệu do khách gửi ở định dạng `application/x-www-form-urlencoded`. Express sẽ không nhận ra định dạng đó theo mặc định. Vì vậy, chúng ta cần sử dụng mô-đun `express.urlencoded`. Khi bao gồm mô-đun này, đối tượng `req`--được truyền dưới dạng tham số cho hàm xử lý-- sẽ có bộ đặc tính `req.body.message` tương ứng với trường `message` do máy khách gửi. Mô-đun được tải với `app.use` nên sẽ được đặt trước phần khai báo tuyến:

```
const express = require('express')
const app = express()
const host = "myserver"
const port = 8080

app.use(express.urlencoded({ extended: true }))
```

Khi thực hiện xong, ta chỉ cần thay đổi `app.get` thành `app.post` trong tuyến hiện có để thực hiện các yêu cầu được thực hiện qua `POST` và khôi phục nội dung yêu cầu:

```
app.post('/', (req, res) => {
  res.send(req.body.message)
})
```

Thay vì thay thế tuyến hiện tại, một lựa chọn khác là chỉ cần thêm tuyến mới này vì Express sẽ xác định phương thức HTTP trong tiêu đề yêu cầu và sử dụng tuyến thích hợp. Vì chúng ta quan tâm đến việc thêm nhiều chức năng vào máy chủ này nên sẽ thuận tiện hơn khi tách riêng từng chức năng bằng các đường dẫn riêng, chẳng hạn như `/echo` và `/ip`.

Đường dẫn và Trình Xử lý Hàm

Sau khi đã xác định phương thức HTTP nào sẽ phản hồi yêu cầu, bây giờ chúng ta cần xác định

một đường dẫn cụ thể cho tài nguyên và một hàm xử lý và tạo phản hồi cho máy khách.

Để mở rộng chức năng `echo` của máy chủ, chúng ta có thể xác định tuyến bằng cách sử dụng phương thức `POST` với đường dẫn `/echo`:

```
app.post('/echo', (req, res) => {
  res.send(req.body.message)
})
```

Tham số `req` của hàm xử lý có chứa tất cả các chi tiết yêu cầu được lưu dưới dạng thuộc tính. Nội dung của trường `message` trong nội dung yêu cầu sẽ có sẵn trong thuộc tính `req.body.message`. Ví dụ chỉ cần gửi trường này trở lại máy khách thông qua lệnh gọi `res.send(req.body.message)`.

Hãy nhớ rằng những thay đổi ta thực hiện sẽ chỉ có hiệu lực sau khi khởi động lại máy chủ. Vì đang chạy máy chủ từ cửa sổ dòng lệnh trong các ví dụ trong chương này, ta có thể tắt máy chủ bằng cách nhấn `Ctrl` + `C` trên cửa sổ dòng lệnh đó. Sau đó, hãy chạy lại máy chủ thông qua lệnh `node index.js`. Phản hồi mà máy khách nhận được đối với yêu cầu `curl` mà chúng ta đã có trước đây hiện đã thành công:

```
$ curl http://myserver:8080/echo --data message="This is the POST request body"
This is the POST request body
```

Các cách khác để truyền và trả lại thông tin trong yêu cầu GET

Việc sử dụng phương thức HTTP `POST` có thể sẽ hơi thừa nếu ta chỉ cần gửi các tin nhắn văn bản ngắn như tin nhắn được sử dụng trong ví dụ. Trong những trường hợp như vậy, dữ liệu có thể được gửi dưới dạng *chuỗi truy vấn* bắt đầu bằng dấu chấm hỏi. Do đó, chuỗi `?message=This+is+the+message` có thể được bao gồm trong đường dẫn yêu cầu của phương thức HTTP `GET`. Các trường được sử dụng trong chuỗi truy vấn sẽ có sẵn cho máy chủ trong đặc tính `req.query`. Do đó, một trường có tên `message` sẽ có sẵn trong đặc tính `req.query.message`.

Một cách khác để gửi dữ liệu qua phương thức HTTP `GET` là sử dụng các *tham số tuyến* của Express:

```
app.get('/echo/:message', (req, res) => {
  res.send(req.params.message)
})
```

Tuyến trong ví dụ này khớp với các yêu cầu được thực hiện bằng phương thức `GET` thông qua việc sử dụng đường dẫn `/echo/:message`, trong đó `:message` là trình giữ chỗ cho bất kỳ cụm từ nào được máy khách gửi cùng với nhãn đó. Các tham số này có thể truy cập được trong đặc tính `req.params`. Với tuyến mới này, hàm `echo` của máy chủ có thể được máy khách yêu cầu ngắn gọn hơn:

```
$ curl http://myserver:8080/echo/hello
hello
```

Trong các tình huống khác, thông tin mà máy chủ cần để xử lý yêu cầu sẽ không cần phải được cung cấp rõ ràng bởi máy khách. Chẳng hạn, máy chủ có một cách khác để truy xuất địa chỉ IP công khai của máy khách. Thông tin đó có trong đối tượng `req` theo mặc định trong đặc tính `req.ip`:

```
app.get('/ip', (req, res) => {
  res.send(req.ip)
})
```

Giờ đây, máy khách có thể yêu cầu đường dẫn `/ip` bằng phương thức `GET` để tìm địa chỉ IP công cộng của chính nó:

```
$ curl http://myserver:8080/ip
187.34.178.12
```

Máy khách có thể sửa đổi các đặc tính khác của đối tượng `req`, đặc biệt là các tiêu đề yêu cầu có sẵn trong `req.headers`. Ví dụ như đặc tính `req.headers.user-agent` sẽ xác định chương trình nào đang đưa ra yêu cầu. Mặc dù đây không phải là thông lệ nhưng máy khách vẫn có thể thay đổi nội dung của trường này; vì vậy, máy chủ không nên sử dụng nó để xác thực chắc chắn một máy khách cụ thể. Điều quan trọng hơn nữa là xác thực dữ liệu do máy khách cung cấp một cách rõ ràng để tránh sự không nhất quán về ranh giới và định dạng có khả năng ảnh hưởng xấu đến ứng dụng.

Điều chỉnh Phản hồi

Như chúng ta đã thấy trong các ví dụ trước, tham số `res` chịu trách nhiệm trả về phản hồi cho máy khách. Hơn nữa, đối tượng `res` có thể thay đổi các khía cạnh khác của phản hồi. Bạn có thể nhận thấy rằng, mặc dù các phản hồi mà chúng ta đã triển khai cho đến nay chỉ là các tin nhắn văn bản đơn giản ngắn gọn, tiêu đề `Content-Type` của các phản hồi lại đang sử dụng `text/html; charset=utf-8`. Mặc dù điều này không ngăn phản hồi văn bản thuần túy được chấp nhận

nhưng sẽ đúng hơn nếu chúng ta xác định lại trường này trong tiêu đề phản hồi thành `text/plain` với cài đặt `res.type('text/plain')`.

Các loại điều chỉnh phản hồi khác sẽ liên quan đến việc sử dụng *cookies* và cho phép máy chủ xác định máy khách đã đưa ra yêu cầu trước đó. Cookie rất quan trọng đối với các tính năng nâng cao, chẳng hạn như tạo các phiên riêng tư liên kết yêu cầu với một người dùng cụ thể; nhưng ở đây, chúng ta sẽ chỉ xem xét một ví dụ đơn giản về cách sử dụng cookie để xác định máy khách đã truy cập máy chủ trước đó.

Với thiết kế được mô đun hóa của Express, trình quản lý cookie phải được cài đặt bằng lệnh `npm` trước khi được sử dụng trong tệp lệnh:

```
$ npm install cookie-parser
```

Sau khi cài đặt, trình quản lý cookie phải được đưa vào tệp lệnh máy chủ. Định nghĩa sau đây nên được đưa vào gần đầu tệp:

```
const cookieParser = require('cookie-parser')
app.use(cookieParser())
```

Để minh họa việc sử dụng cookie, hãy sửa đổi chức năng xử lý của tuyến bằng đường dẫn gốc `/` đã tồn tại trong tệp lệnh. Đối tượng `req` có đặc tính `req.cookies`, trong đó, cookie được gửi trong tiêu đề yêu cầu được lưu giữ. Mặt khác, đối tượng `res` có phương thức `res.cookie()` để tạo cookie mới gửi đến máy khách. Hàm xử lý trong ví dụ sau sẽ kiểm tra xem cookie tên `known` có tồn tại trong yêu cầu hay không. Nếu một cookie như vậy không tồn tại, máy chủ sẽ giả định rằng đây là khách truy cập lần đầu và gửi cho nó một cookie có tên đó thông qua lệnh gọi `res.cookie('known', '1')`. Chúng ta sẽ tùy ý gán giá trị `1` cho cookie vì nó được cho là có một số nội dung, nhưng máy chủ sẽ không tham khảo giá trị đó. Ứng dụng này chỉ giả định rằng sự hiện diện đơn giản của cookie cho biết máy khách này trước đây đã yêu cầu tuyến này:

```
app.get('/', (req, res) => {
  res.type('text/plain')
  if ( req.cookies.known === undefined ){
    res.cookie('known', '1')
    res.send('Welcome, new visitor!')
  }
  else
    res.send('Welcome back, visitor');
})
```

Theo mặc định, `curl` không sử dụng cookie trong các giao dịch. Nhưng nó có các tùy chọn để lưu trữ (`-c cookie.txt`) và gửi các cookie đã lưu trữ (`-b cookie.txt`):

```
$ curl http://myserver:8080/ -c cookies.txt -b cookies.txt -v
* Trying 192.168.1.225:8080...
* TCP_NODELAY set
* Connected to myserver (192.168.1.225) port 8080 (#0)
> GET / HTTP/1.1
> Host: myserver:8080
> User-Agent: curl/7.68.0
>Accept: /
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: text/plain; charset=utf-8
* Added cookie known="1" for domain myserver, path /, expire 0
< Set-Cookie: known=1; Path=/
< Content-Length: 21
< ETag: W/"15-l7qrxcqicl4xv6EfA5fZFWCFrgY"
< Date: Sat, 03 Jul 2021 23:45:03 GMT
< Connection: keep-alive
<
* Connection #0 to host myserver left intact
Welcome, new visitor!
```

Vì lệnh này là lần truy cập đầu tiên kể từ khi cookie được triển khai trên máy chủ nên máy khách sẽ không có bất kỳ cookie nào để đưa vào yêu cầu. Như mong đợi, máy chủ không xác định cookie trong yêu cầu và do đó đã đưa cookie vào tiêu đề phản hồi như được chỉ ra trong dòng `Set-Cookie: known=1; Path=/` của đầu ra. Vì chúng ta đã bật cookie trong `curl`, một yêu cầu mới sẽ bao gồm cookie `known=1` trong tiêu đề yêu cầu và cho phép máy chủ xác định sự hiện diện của cookie:

```
$ curl http://myserver:8080/ -c cookies.txt -b cookies.txt -v
* Trying 192.168.1.225:8080...
* TCP_NODELAY set
* Connected to myserver (192.168.1.225) port 8080 (#0)
> GET / HTTP/1.1
> Host: myserver:8080
> User-Agent: curl/7.68.0
>Accept: /
> Cookie: known=1
```

```
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: text/plain; charset=utf-8
< Content-Length: 21
< ETag: W/"15-ATq2flQYtLMYIUpJwwpb5SjV9Ww"
< Date: Sat, 03 Jul 2021 23:45:47 GMT
< Connection: keep-alive
<
* Connection #0 to host myserver left intact
Welcome back, visitor
```

Bảo mật Cookie

Nhà phát triển cần biết về các lỗ hổng tiềm ẩn khi sử dụng cookie để xác định máy khách đưa ra yêu cầu. Những kẻ tấn công có thể sử dụng các kỹ thuật như *thực thi tệp lệnh liên trang* (XSS) và *giả mạo yêu cầu liên trang* (CSRF) để đánh cắp cookie từ máy khách và từ đó mạo danh họ khi đưa ra yêu cầu tới máy chủ. Nói chung, các kiểu tấn công này sử dụng các trường chú thích không hợp lệ hoặc các URL được xây dựng tỉ mỉ để chèn mã JavaScript độc hại vào trang. Khi được thực thi bởi một máy khách xác thực, mã này có thể sao chép các cookie hợp lệ và lưu trữ hoặc chuyển tiếp chúng đến một đích đến khác.

Do đó, đặc biệt là trong các ứng dụng chuyên nghiệp, điều quan trọng là phải cài đặt và sử dụng các tính năng Express chuyên dụng hơn được gọi là *phần mềm trung gian*. Mô-đun `express-session` hoặc `cookie-session` cung cấp khả năng kiểm soát đầy đủ và an toàn hơn đối với việc quản lý phiên và cookie. Các thành phần này cho phép kiểm soát bổ sung để ngăn cookie bị chuyển hướng khỏi nhà phát hành ban đầu của chúng.

Bài tập Hướng dẫn

1. Làm cách nào để đọc nội dung của trường `comment` được gửi trong chuỗi truy vấn của phương thức HTTP `GET` trong một hàm xử lý?

2. Hãy viết một tuyến sử dụng phương thức HTTP `GET` và đường dẫn `/agent` để gửi lại cho máy khách nội dung của tiêu đề `user-agent`.

3. Express.js có một tính năng gọi là *tham số tuyến*, trong đó một đường dẫn chẳng hạn như `/user/:name` có thể được sử dụng để nhận tham số `name` do máy khách gửi. Làm cách nào để truy cập tham số `name` trong hàm xử lý của tuyến?

Bài tập Mở rộng

1. Nếu tên máy chủ là `myserver`, tuyến Express nào sẽ nhận được nội dung gửi theo mẫu bên dưới?

```
<form action="/contact/feedback" method="post"> ... </form>
```

2. Trong quá trình phát triển máy chủ, lập trình viên không thể đọc được đặc tính `req.body` ngay cả sau khi xác minh rằng máy khách đang gửi nội dung một cách chính xác qua phương thức HTTP `POST`. Nguyên nhân chủ yếu của vấn đề này là gì?

3. Điều gì sẽ xảy ra khi máy chủ có một tuyến được đặt thành đường dẫn `/user/:name` và máy khách đưa ra yêu cầu tới `/user/`?

Tóm tắt

Bài học này đã giải thích cách viết tệp lệnh Express để nhận và xử lý các yêu cầu HTTP. Express sử dụng khái niệm *tuyến* để xác định các tài nguyên có sẵn cho máy khách, giúp bạn linh hoạt trong việc xây dựng máy chủ cho bất kỳ loại ứng dụng web nào. Bài học này đã đi qua các khái niệm và quy trình sau:

- Các tuyến sử dụng các phương thức HTTP `GET` và HTTP `POST`.
- Cách dữ liệu biểu mẫu được lưu trữ trong đối tượng `request`.
- Cách sử dụng tham số tuyến.
- Tùy chỉnh tiêu đề phản hồi.
- Quản lý cookie cơ bản.

Đáp án Bài tập Hướng dẫn

1. Làm cách nào để đọc nội dung của trường `comment` được gửi trong chuỗi truy vấn của phương thức HTTP `GET` trong một hàm xử lý?

   Trường `comment` đã có sẵn trong đặc tính `req.query.comment`.

2. Hãy viết một tuyến sử dụng phương thức HTTP `GET` và đường dẫn `/agent` để gửi lại cho máy khách nội dung của tiêu đề `user-agent`.

   ```
   app.get('/agent', (req, res) => {
     res.send(req.headers.user-agent)
   })
   ```

3. Express.js có một tính năng gọi là *tham số tuyến*, trong đó một đường dẫn chẳng hạn như `/user/:name` có thể được sử dụng để nhận tham số `name` do máy khách gửi. Làm cách nào để truy cập tham số `name` trong hàm xử lý của tuyến?

   Tham số `name` có thể truy cập được trong đặc tính `req.params.name`.

Đáp án Bài tập Mở rộng

1. Nếu tên máy chủ là `myserver`, tuyến Express nào sẽ nhận được nội dung gửi theo mẫu bên dưới?

   ```
   <form action="/contact/feedback" method="post"> ... </form>
   ```

   ```
   app.post('/contact/feedback', (req, res) => {
     ...
   })
   ```

2. Trong quá trình phát triển máy chủ, lập trình viên không thể đọc được đặc tính `req.body` ngay cả sau khi xác minh rằng máy khách đang gửi nội dung một cách chính xác qua phương thức HTTP `POST`. Nguyên nhân chủ yếu của vấn đề này là gì?

   Lập trình viên đã không bao gồm mô-đun `express.urlencoded` cho phép Express trích xuất nội dung của yêu cầu.

3. Điều gì sẽ xảy ra khi máy chủ có một tuyến được đặt thành đường dẫn `/user/:name` và máy khách đưa ra yêu cầu tới `/user/`?

   Máy chủ sẽ đưa ra phản hồi `404 Not Found` vì tuyến yêu cầu tham số `:name` do máy khách cung cấp.

035.3 Bài 2

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	035 Lập trình Máy chủ Node.js
Mục tiêu:	035.1 Khái niệm cơ bản về NodeJS Express
Bài:	2 trên 2

Giới thiệu

Máy chủ web có các cơ chế rất linh hoạt để tạo phản hồi cho các yêu cầu từ máy khách. Đối với một số yêu cầu, máy chủ web chỉ cần cung cấp một phản hồi tĩnh chưa qua xử lý cũng đã là đủ vì tài nguyên được yêu cầu sẽ giống nhau đối với bất kỳ một máy khách nào. Chẳng hạn như khi một máy khách yêu cầu một hình ảnh mà mọi người đều có thể truy cập được, máy chủ chỉ cần gửi đi tệp chứa hình ảnh đó là đủ.

Nhưng khi các phản hồi được tạo động, chúng có thể sẽ cần được cấu trúc một cách tốt hơn so với các dòng chữ đơn giản được viết trong tệp lệnh máy chủ. Trong những trường hợp như vậy, sẽ thuận tiện hơn nếu máy chủ web có thể tạo một tài liệu hoàn chỉnh mà máy khách có thể phiên dịch và hiển thị được. Trong bối cảnh phát triển ứng dụng web, các tài liệu HTML thường được tạo dưới dạng mẫu và được tách biệt khỏi tệp lệnh máy chủ. Tệp lệnh này sẽ chèn dữ liệu động vào các vị trí được xác định trước trong mẫu thích hợp, sau đó gửi phản hồi được định dạng cho máy khách.

Các ứng dụng web thường tiêu thụ cả tài nguyên tĩnh và tài nguyên động. Một tài liệu HTML ngay cả khi được tạo động vẫn có thể có tham chiếu đến các tài nguyên tĩnh như tệp CSS và các hình ảnh. Để minh hoạ cách Express hỗ trợ xử lý loại nhu cầu này, trước tiên, chúng ta sẽ thiết lập một

máy chủ mẫu cung cấp tệp tĩnh, sau đó triển khai các tuyến tạo phản hồi dựa trên mẫu có cấu trúc.

Tệp Tĩnh

Bước đầu tiên là tạo tệp JavaScript chạy dưới dạng máy chủ. Hãy làm theo mô hình tương tự như đã được trình bày trong các bài học trước để tạo một ứng dụng Express đơn giản: trước tiên, hãy tạo một thư mục có tên `server`; sau đó cài đặt các thành phần cơ sở bằng lệnh `npm`:

```
$ mkdir server
$ cd server/
$ npm init
$ npm install express
```

Tại điểm nhập, chúng ta có thể sử dụng tên tệp bất kỳ, nhưng ở đây, ta sẽ sử dụng tên tệp mặc định là `index.js`. Danh sách sau đây cho thấy tệp `index.js` cơ bản sẽ được sử dụng làm điểm nhập cho máy chủ:

```
const express = require('express')
const app = express()
const host = "myserver"
const port = 8080

app.listen(port, host, () => {
  console.log(`Server ready at http://${host}:${port}`)
})
```

Bạn không cần phải viết một mã chi tiết để có thể gửi được tệp tĩnh. Express có phần mềm trung gian cho mục đích này và được gọi là `express.static`. Nếu máy chủ cần gửi các tệp tĩnh đến máy khách, bạn chỉ cần tải phần mềm trung gian `express.static` ở đầu tệp lệnh:

```
app.use(express.static('public'))
```

Tham số `public` cho biết thư mục lưu trữ các tệp mà máy khách có thể yêu cầu. Các đường dẫn do máy khách yêu cầu không được bao gồm thư mục `public` mà chỉ có tên tệp hoặc đường dẫn đến tệp liên quan đến thư mục `public`. Ví dụ: để yêu cầu tệp `public/layout.css`, máy khách phải gửi yêu cầu tới `/layout.css`.

Đầu ra được định dạng

Trong khi việc gửi nội dung tĩnh rất đơn giản, các nội dung được tạo động lại có thể khác nhau một cách đáng kể. Việc tạo phản hồi động với các thông báo ngắn giúp ta dễ dàng kiểm tra các ứng dụng trong giai đoạn phát triển ban đầu. Ví dụ: sau đây là một tuyến thử nghiệm chỉ gửi lại cho máy khách một thông báo mà nó đã gửi bằng phương thức HTTP `POST`. Phản hồi chỉ có thể sao chép nội dung thư ở dạng văn bản thuần túy mà không có bất kỳ định dạng nào:

```
app.post('/echo', (req, res) => {
  res.send(req.body.message)
})
```

Một tuyến dạng như vậy là một ví dụ điển hình khi học về Express và cho các mục đích chẩn đoán, trong đó, một phản hồi thô chỉ cần được gửi bằng `res.send()` là đủ. Tuy nhiên, một máy chủ hữu ích phải có khả năng tạo ra các phản hồi phức tạp hơn. Bây giờ, chúng ta sẽ cùng phát triển loại tuyến tinh vi hơn này.

Thay vì chỉ gửi lại nội dung của yêu cầu hiện tại, ứng dụng mới của chúng ta sẽ duy trì một danh sách đầy đủ các thông báo được gửi trong các yêu cầu trước đó của từng máy khách và gửi lại danh sách của từng máy khách khi được yêu cầu. Một phản hồi hợp nhất tất cả các thông báo là một tùy chọn, nhưng các chế độ đầu ra được định dạng khác sẽ phù hợp hơn, đặc biệt là khi các phản hồi trở nên phức tạp hơn.

Để nhận và lưu trữ tin nhắn của máy khách được gửi trong phiên hiện tại, trước tiên, chúng ta cần bao gồm cả các mô-đun bổ sung để xử lý cookie và dữ liệu được gửi qua phương thức HTTP `POST`. Mục đích duy nhất của máy chủ trong ví dụ sau là ghi nhật ký các thông báo được gửi qua `POST` và hiển thị các thông báo đã gửi trước đó khi máy khách đưa ra yêu cầu `GET`. Vì vậy, có hai tuyến cho đường dẫn `/`. Tuyến đầu tiên đáp ứng các yêu cầu được thực hiện bằng phương thức HTTP `POST` và tuyến thứ hai đáp ứng các yêu cầu được thực hiện bằng phương thức HTTP `GET`:

```
const express = require('express')
const app = express()
const host = "myserver"
const port = 8080

app.use(express.static('public'))

const cookieParser = require('cookie-parser')
app.use(cookieParser())

const { v4: uuidv4 } = require('uuid')
```

```
app.use(express.urlencoded({ extended: true }))

// Array to store messages
let messages = []

app.post('/', (req, res) => {

  // Only JSON enabled requests
  if ( req.headers.accept != "application/json" )
  {
    res.sendStatus(404)
    return
  }

  // Locate cookie in the request
  let uuid = req.cookies.uuid

  // If there is no uuid cookie, create a new one
  if ( uuid === undefined )
    uuid = uuidv4()

  // Add message first in the messages array
  messages.unshift({uuid: uuid, message: req.body.message})

  // Collect all previous messages for uuid
  let user_entries = []
  messages.forEach( (entry) => {
    if ( entry.uuid == req.cookies.uuid )
      user_entries.push(entry.message)
  })

  // Update cookie expiration date
  let expires = new Date(Date.now());
  expires.setDate(expires.getDate() + 30);
  res.cookie('uuid', uuid, { expires: expires })

  // Send back JSON response
  res.json(user_entries)

})

app.get('/', (req, res) => {
```

```
  // Only JSON enabled requests
  if ( req.headers.accept != "application/json" )
  {
    res.sendStatus(404)
    return
  }

  // Locate cookie in the request
  let uuid = req.cookies.uuid

  // Client's own messages
  let user_entries = []

  // If there is no uuid cookie, create a new one
  if ( uuid === undefined ){
    uuid = uuidv4()
  }
  else {
    // Collect messages for uuid
    messages.forEach( (entry) => {
      if ( entry.uuid == req.cookies.uuid )
        user_entries.push(entry.message)
    })
  }

  // Update cookie expiration date
  let expires = new Date(Date.now());
  expires.setDate(expires.getDate() + 30);
  res.cookie('uuid', uuid, { expires: expires })

  // Send back JSON response
  res.json(user_entries)

})

app.listen(port, host, () => {
  console.log(`Server ready at http://${host}:${port}`)
})
```

Chúng ta sẽ giữ cấu hình tệp tĩnh ở trên cùng bởi việc cung cấp các tệp tĩnh (chẳng hạn như `layout.css`) sẽ sớm trở nên hữu ích. Ngoài phần mềm trung gian `cookie-parser` được giới thiệu trong chương trước, ví dụ này cũng bao gồm phần mềm trung gian `uuid` được sử dụng để tạo một số nhận dạng duy nhất được truyền dưới dạng cookie cho mỗi máy khách gửi đi thông

báo. Nếu chưa được cài đặt trong thư mục máy chủ trong ví dụ, các mô-đun này có thể được cài đặt bằng lệnh `npm install cookie-parser uuid`.

Mảng toàn cục được gọi là `messages` sẽ lưu trữ các thông báo được gửi bởi tất cả các máy khách. Mỗi mục trong mảng này đều bao gồm một đối tượng có đặc tính `uuid` và `message`.

Một điều thực sự mới mẻ trong tệp lệnh này là phương thức `res.json()` được sử dụng ở cuối hai tuyến để tạo phản hồi ở định dạng JSON với mảng chứa các thông báo được gửi bởi máy khách:

```
// Send back JSON response
res.json(user_entries)
```

JSON là một định dạng văn bản thuần túy cho phép ta nhóm một tập hợp dữ liệu thành một cấu trúc duy nhất có tính liên kết - tức nội dung được thể hiện dưới dạng khóa và giá trị. JSON đặc biệt hữu ích khi các phản hồi được xử lý bởi máy khách. Khi sử dụng định dạng này, một đối tượng hoặc một mảng JavaScript có thể dễ dàng được xây dựng lại ở phía máy khách với tất cả các đặc tính và chỉ mục của đối tượng ban đầu trên máy chủ.

Vì đang cấu trúc từng thông báo trong JSON nên chúng ta sẽ từ chối các yêu cầu không chứa `application/json` trong tiêu đề `accept` của chúng:

```
// Only JSON enabled requests
if ( req.headers.accept != "application/json" )
{
  res.sendStatus(404)
  return
}
```

Yêu cầu được thực hiện bằng lệnh `curl` đơn giản để chèn thông báo mới sẽ không được chấp nhận vì theo mặc định, `curl` không chỉ định `application/json` trong tiêu đề `accept`:

```
$ curl http://myserver:8080/ --data message="My first message" -c cookies.txt -b cookies.txt
Not Found
```

Tùy chọn `-H "accept: application/json"` sẽ thay đổi tiêu đề của yêu cầu để chỉ định định dạng của phản hồi - lần này sẽ được chấp nhận và trả lời ở định dạng đã chỉ định:

```
$ curl http://myserver:8080/ --data message="My first message" -c cookies.txt -b cookies.txt
-H "accept: application/json"
```

```
["My first message"]
```

Việc nhận thông báo bằng cách sử dụng tuyến khác cũng được thực hiện theo cách tương tự, nhưng lần này sẽ sử dụng phương thức HTTP `GET`:

```
$ curl http://myserver:8080/ -c cookies.txt -b cookies.txt -H "accept: application/json"
["Another message","My first message"]
```

Mẫu

Phản hồi ở các định dạng như JSON sẽ thuận tiện cho việc giao tiếp giữa các chương trình. Tuy nhiên, mục đích chính của hầu hết các máy chủ ứng dụng web là tạo ra nội dung HTML cho mục đích sử dụng của con người. Việc nhúng mã HTML vào mã JavaScript không phải là một ý tưởng hay bởi việc trộn lẫn các ngôn ngữ trong cùng một tệp khiến chương trình dễ bị lỗi hơn và sẽ gây hại tới công tác bảo trì mã.

Express có thể làm việc với các *công cụ mẫu* khác nhau để tách riêng HTML cho nội dung động; danh sách đầy đủ có thể được tìm thấy tại Trang web công cụ mẫu Express. Một trong những công cụ mẫu phổ biến nhất là *JavaScript nhúng* (EJS) cho phép bạn tạo các tệp HTML với các thẻ cụ thể để chèn nội dung động.

Giống như các thành phần Express khác, EJS cần được cài đặt trong thư mục mà máy chủ đang chạy:

```
$ npm install ejs
```

Tiếp theo, công cụ EJS phải được đặt làm trình kết xuất mặc định trong tệp lệnh máy chủ (gần đầu tệp `index.js`, ttrước các dòng xác định tuyến):

```
app.set('view engine', 'ejs')
```

Phản hồi được tạo cùng với mẫu sẽ được gửi đến máy khách bằng hàm `res.render()`. Hàm này sẽ nhận dưới dạng tham số là tên tệp mẫu và một đối tượng chứa các giá trị có thể truy cập được từ bên trong mẫu. Các tuyến được sử dụng trong ví dụ trước có thể được viết lại để tạo các phản hồi HTML cũng như JSON:

```
app.post('/', (req, res) => {
```

```
  let uuid = req.cookies.uuid

  if ( uuid === undefined )
    uuid = uuidv4()

  messages.unshift({uuid: uuid, message: req.body.message})

  let user_entries = []
  messages.forEach( (entry) => {
    if ( entry.uuid == req.cookies.uuid )
      user_entries.push(entry.message)
  })

  let expires = new Date(Date.now());
  expires.setDate(expires.getDate() + 30);
  res.cookie('uuid', uuid, { expires: expires })

  if ( req.headers.accept == "application/json" )
    res.json(user_entries)
  else
    res.render('index', {title: "My messages", messages: user_entries})

})

app.get('/', (req, res) => {

  let uuid = req.cookies.uuid

  let user_entries = []

  if ( uuid === undefined ){
    uuid = uuidv4()
  }
  else {
    messages.forEach( (entry) => {
      if ( entry.uuid == req.cookies.uuid )
        user_entries.push(entry.message)
    })
  }

  let expires = new Date(Date.now());
  expires.setDate(expires.getDate() + 30);
  res.cookie('uuid', uuid, { expires: expires })
```

```
  if ( req.headers.accept == "application/json" )
    res.json(user_entries)
  else
    res.render('index', {title: "My messages", messages: user_entries})

})
```

Hãy lưu ý rằng định dạng của phản hồi sẽ phụ thuộc vào tiêu đề `accept` có trong yêu cầu:

```
if ( req.headers.accept == "application/json" )
  res.json(user_entries)
else
  res.render('index', {title: "My messages", messages: user_entries})
```

Một phản hồi ở định dạng JSON sẽ chỉ được gửi nếu máy khách yêu cầu nó một cách rõ ràng. Nếu không, phản hồi sẽ được tạo từ mẫu `index`. Cùng một mảng `user_entries` sẽ cung cấp cả đầu ra JSON và mẫu, nhưng đối tượng được sử dụng làm tham số cho mẫu sau cũng có đặc tính `title: "My messages"` được sử dụng để làm tiêu đề bên trong mẫu.

Mẫu HTML

Giống như các tệp tĩnh, các tệp chứa mẫu HTML nằm trong thư mục riêng của chúng. Theo mặc định, EJS sẽ giả định các tệp mẫu nằm trong thư mục `views/`. Trong ví dụ này, một mẫu có tên `index` đã được sử dụng; vì vậy, EJS sẽ tìm kiếm tệp `views/index.ejs`. Danh sách sau đây là nội dung của mẫu `views/index.ejs` đơn giản có thể được sử dụng với mã ví dụ:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title><%= title %></title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="/layout.css">
</head>
<body>

<div id="interface">

<form action="/" method="post">
<p>
  <input type="text" name="message">
```

```
  <input type="submit" value="Submit">
</p>
</form>

<ul>
<% messages.forEach( (message) => { %>
<li><%= message %></li>
<% }) %>
</ul>

</div>

</body>
</html>
```

Thẻ EJS đặc biệt đầu tiên là phần tử `<title>` trong phần `<head>`:

```
<%= title %>
```

Trong quá trình kết xuất, thẻ đặc biệt này sẽ được thay thế bằng giá trị của đặc tính `title` của đối tượng được truyền dưới dạng tham số cho hàm `res.render()`.

Hầu hết các mẫu đều được tạo thành từ mã HTML thông thường; vì vậy, mẫu sẽ chứa biểu mẫu HTML để gửi thông báo mới. Máy chủ thử nghiệm sẽ phản hồi các phương thức HTTP `GET` và `POST` cho cùng một đường dẫn `/`; do đó, ta có các thuộc tính `action="/"` và `method="post"` trong thẻ biểu mẫu.

Các phần khác của mẫu là sự kết hợp giữa mã HTML và các thẻ EJS. EJS có các thẻ dành cho các mục đích cụ thể trong mẫu:

`<% … %>`

Chèn kiểm soát luồng. Không có nội dung nào được chèn trực tiếp bởi thẻ này nhưng nó có thể được sử dụng với các cấu trúc JavaScript để chọn, lặp lại hoặc chặn các phần của HTML. Ví dụ: để bắt đầu một vòng lặp: `<% messages.forEach( (message) => { %>`

`<%# … %>`

Xác định một chú thích có nội dung bị trình phân tích cú pháp bỏ qua. Không giống như các chú thích được viết bằng HTML, những chú thích này sẽ không được hiển thị với máy khách.

`<%= … %>`

Chèn nội dung thoát của biến. Việc thoát khỏi nội dung không xác định để tránh thực thi mã

JavaScript là một yếu tố quan trọng; điều này có thể tạo kẽ hở cho các cuộc tấn công tệp lệnh liên trang (XSS). Ví dụ: `<%= title %>`

`<%- … %>`

Chèn nội dung của biến mà không thoát.

Sự kết hợp giữa mã HTML và các thẻ EJS được thể hiện rõ trong đoạn mã nơi các thông báo của khách hàng được hiển thị dưới dạng danh sách HTML:

```
<ul>
<% messages.forEach( (message) => { %>
<li><%= message %></li>
<% }) %>
</ul>
```

Trong đoạn mã này, thẻ `<% … %>` đầu tiên sẽ bắt đầu câu lệnh `forEach` lặp qua tất cả các phần tử của mảng `message`. Dấu phân cách `<%` và `%>` cho phép ta kiểm soát các đoạn mã HTML. Một danh mục HTML mới là `<li><%= message %></li>` sẽ được tạo cho từng phần tử của `message`. Với những thay đổi này, máy chủ sẽ gửi phản hồi bằng HTML khi nhận được yêu cầu như sau:

```
$ curl http://myserver:8080/ --data message="This time" -c cookies.txt -b cookies.txt
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>My messages</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="/layout.css">
</head>
<body>

<div id="interface">

<form action="/" method="post">
<p>
  <input type="text" name="message">
  <input type="submit" value="Submit">
</p>
</form>

<ul>
```

```
<li>This time</li>

<li>in HTML</li>

</ul>

</div>

</body>
</html>
```

Sự tách biệt giữa mã để xử lý yêu cầu và mã để trình bày phản hồi sẽ làm cho mã sạch hơn và cho phép nhóm công tác phân chia việc phát triển ứng dụng giữa những thành viên với chuyên môn riêng biệt. Ví dụ: một nhà thiết kế web có thể tập trung vào các tệp mẫu trong `views/` và các biểu định kiểu liên quan được cung cấp dưới dạng các tệp tĩnh được lưu trữ trong thư mục `public/` trên máy chủ mẫu.

Bài tập Hướng dẫn

1. `express.static` nên được định cấu hình như thế nào để máy khách có thể yêu cầu các tệp trong thư mục `assets`?

2. Làm cách nào để xác định loại phản hồi được chỉ định trong tiêu đề của yêu cầu trong một tuyến Express?

3. Phương thức nào của tham số tuyến `res` (phản hồi) sẽ tạo phản hồi ở định dạng JSON từ một mảng JavaScript có tên là `content`?

Bài tập Mở rộng

1. Theo mặc định, các tệp mẫu Express sẽ nằm trong thư mục `views`. Làm cách nào để sửa đổi cài đặt này để các tệp mẫu được lưu trữ trong `templates`?

2. Giả sử một máy khách nhận được phản hồi HTML không có tiêu đề (tức `<title></title>`). Sau khi xác minh mẫu EJS, nhà phát triển sẽ tìm thấy thẻ `<title><% title %></title>` trong phần `head` của tệp. Nguyên nhân của vấn đề này có thể là gì?

3. Hãy sử dụng thẻ mẫu EJS để viết thẻ HTML `<h2></h2>` với nội dung của biến JavaScript `h2`. Thẻ này chỉ được hiển thị nếu biến `h2` không trống.

Tóm tắt

Bài học này bao gồm các phương thức cơ bản mà Express.js cung cấp để tạo các phản hồi tĩnh và phản hồi động được định dạng. Việc thiết lập máy chủ HTTP cho các tệp tĩnh khá là đơn giản và hệ thống tạo khuôn mẫu EJS sẽ cung cấp một cách dễ dàng để tạo nội dung động từ các tệp HTML. Bài học này đã đi qua các khái niệm và quy trình sau:

- Sử dụng `express.static` cho phản hồi tệp tĩnh.
- Cách tạo phản hồi để khớp với trường loại nội dung trong tiêu đề yêu cầu.
- Phản hồi có cấu trúc JSON.
- Sử dụng các thẻ EJS trong các mẫu dựa trên HTML.

Đáp án Bài tập Hướng dẫn

1. `express.static` nên được định cấu hình như thế nào để máy khách có thể yêu cầu các tệp trong thư mục `assets`?

   Một lệnh gọi `app.use(express.static('assets'))` nên được thêm vào tệp lệnh máy chủ.

2. Làm cách nào để xác định loại phản hồi được chỉ định trong tiêu đề của yêu cầu trong một tuyến Express?

   Máy khách nên đặt các loại được chấp nhận trong trường tiêu đề `accept` được ánh xạ tới đặc tính `req.headers.accept`.

3. Phương thức nào của tham số tuyến `res` (phản hồi) sẽ tạo phản hồi ở định dạng JSON từ một mảng JavaScript có tên là `content`?

   Phương thức `res.json()`: `res.json(content)`.

Đáp án Bài tập Mở rộng

1. Theo mặc định, các tệp mẫu Express sẽ nằm trong thư mục `views`. Làm cách nào để sửa đổi cài đặt này để các tệp mẫu được lưu trữ trong `templates`?

   Thư mục có thể được xác định trong thiết lập ban đầu của tệp lệnh với `app.set('views', './templates')`.

2. Giả sử một máy khách nhận được phản hồi HTML không có tiêu đề (tức `<title></title>`). Sau khi xác minh mẫu EJS, nhà phát triển sẽ tìm thấy thẻ `<title><% title %></title>` trong phần `head` của tệp. Nguyên nhân của vấn đề này có thể là gì?

   Thẻ `<%= %>` nên được sử dụng để chứa nội dung của một biến như trong `<%= title %>`.

3. Hãy sử dụng thẻ mẫu EJS để viết thẻ HTML `<h2></h2>` với nội dung của biến JavaScript `h2`. Thẻ này chỉ được hiển thị nếu biến `h2` không trống.

```
<% if ( h2 != "" ) { %>
<h2><%= h2 %></h2>
<% } %>
```

035.3 Khái niệm cơ bản về SQL

Tham khảo các mục tiêu LPI

Web Development Essentials version 1.0, Exam 030, Objective 035.3

Khối lượng

3

Các lĩnh vực kiến thức chính

- Thiết lập kết nối cơ sở dữ liệu từ NodeJS
- Lấy dữ liệu từ cơ sở dữ liệu trong NodeJS
- Thực thi các truy vấn SQL từ NodeJS
- Tạo các truy vấn SQL đơn giản không bao gồm các phép nối
- Hiểu về khóa chính
- Các biến thoát được sử dụng trong các truy vấn SQL
- Nhận thức về SQL Injection

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- `sqlite3` NPM module
- `Database.run()`, `Database.close()`, `Database.all()`, `Database.get()`, `Database.each()`
- `CREATE TABLE`
- `INSERT`, `SELECT`, `DELETE`, `UPDATE`

035.3 Bài 1

Chứng chỉ:	Web Development Essentials
Phiên bản:	1.0
Chủ đề:	035 Lập trình Máy chủ Node.js
Mục tiêu:	035.3 Khái niệm cơ bản về SQL
Bài:	1 trên 1

Giới thiệu

Mặc dù có thể viết các hàm riêng để thực hiện lưu trữ liên tục, có thể sẽ thuận tiện hơn khi chúng ta sử dụng hệ thống quản lý cơ sở dữ liệu để tăng tốc độ phát triển và đảm bảo tính bảo mật cũng như ổn định cho dữ liệu dạng bảng. Chiến lược phổ biến nhất để lưu trữ dữ liệu được tổ chức trong các bảng có mối liên hệ với nhau (đặc biệt là khi các bảng đó được truy vấn và cập nhật liên tục) là cài đặt cơ sở dữ liệu quan hệ có hỗ trợ *Ngôn ngữ Truy vấn có Cấu trúc* (SQL) - một ngôn ngữ hướng đến cơ sở dữ liệu quan hệ. Node.js có hỗ trợ nhiều hệ thống quản lý cơ sở dữ liệu SQL khác nhau. Theo các nguyên tắc về tính di động và thực thi không gian người dùng được Node.js Express chấp nhận, SQLite là một lựa chọn thích hợp để lưu trữ dữ liệu liên tục được sử dụng bởi loại máy chủ HTTP này.

SQL

Ngôn ngữ Truy vấn có Cấu trúc được dành riêng cho cơ sở dữ liệu. Các thao tác viết và đọc được thể hiện trong các câu gọi là *câu lệnh* và *truy vấn*. Cả câu lệnh và truy vấn đều được tạo thành từ các *mệnh đề* xác định các điều kiện để thực hiện thao tác.

Ví dụ: tên và địa chỉ email có thể được lưu trữ trong bảng cơ sở dữ liệu chứa các trường `name` và

`email`. Một cơ sở dữ liệu có thể chứa nhiều bảng; vì vậy, mỗi bảng phải có một cái tên độc nhất. Nếu chúng ta sử dụng tên `contacts` (liên hệ) cho bảng tên và email, một bản ghi mới có thể được chèn với *câu lệnh* sau:

```
INSERT INTO contacts (name, email) VALUES ("Carol", "carol@example.com");
```

Câu lệnh chèn này bao gồm mệnh đề `INSERT INTO`; mệnh đề này xác định bảng và các trường nơi dữ liệu sẽ được chèn vào. Mệnh đề `VALUES` sẽ thiết lập các giá trị sẽ được chèn. Tuy không cần phải viết hoa các mệnh đề nhưng đây là một thông lệ phổ biến để có thể nhận dạng các từ khóa SQL trong một câu lệnh hoặc truy vấn một cách hiệu quả hơn.

Một truy vấn trên bảng liên hệ cũng được thực hiện theo cách tương tự nhưng sẽ sử dụng mệnh đề `SELECT`:

```
SELECT email FROM contacts;
dave@example.com
carol@example.com
```

Trong trường hợp này, mệnh đề `SELECT email` sẽ chọn một trường từ các mục trong bảng `contacts`. Mệnh đề `WHERE` sẽ hạn chế truy vấn đối với các hàng cụ thể:

```
SELECT email FROM contacts WHERE name = "Dave";
dave@example.com
```

SQL có nhiều mệnh đề khác nhau và chúng ta sẽ xem xét một số mệnh đề đó trong các phần sau. Trước tiên, ta cần phải hiểu về cách tích hợp cơ sở dữ liệu SQL với Node.js.

SQLite

SQLite có lẽ là giải pháp đơn giản nhất để kết hợp các tính năng của cơ sở dữ liệu SQL vào một ứng dụng. Không giống như các hệ thống quản lý cơ sở dữ liệu phổ biến khác, SQLite không phải là máy chủ cơ sở dữ liệu mà máy khách kết nối tới. Thay vào đó, SQLite sẽ cung cấp một tập hợp các hàm cho phép nhà phát triển tạo cơ sở dữ liệu giống như một tệp thông thường. Trong trường hợp máy chủ HTTP được triển khai bằng Node.js Express, tệp này thường sẽ nằm trong cùng một thư mục với tệp lệnh máy chủ.

Trước khi sử dụng SQLite trong Node.js, ta cần cài đặt mô-đun `sqlite3`. Hãy chạy lệnh sau trong thư mục cài đặt của máy chủ - tức là thư mục chứa tệp lệnh Node.js mà ta sẽ chạy.

```
$ npm install sqlite3
```

Hãy lưu ý rằng có một số mô-đun có hỗ trợ SQLite, chẳng hạn như `better-sqlite3` với cách sử dụng hơi khác so với `sqlite3`. Các ví dụ trong bài học này đều dành cho mô-đun `sqlite3`; vì vậy, chúng có thể sẽ không hoạt động được như mong đợi nếu bạn chọn một mô-đun khác.

Mở Cơ sở Dữ liệu

Để minh họa cách máy chủ Node.js Express làm việc với cơ sở dữ liệu SQL, hãy viết một tệp lệnh lưu trữ và hiển thị các thông báo được gửi từ một máy khách được xác định bằng cookie. Thông báo được gửi bởi máy khách thông qua phương thức HTTP POST và phản hồi của máy chủ có thể được định dạng dưới dạng JSON hoặc HTML (từ mẫu) tùy thuộc vào định dạng mà máy khách yêu cầu. Bài học này sẽ không đi sâu vào chi tiết về cách sử dụng các phương thức, cookie và mẫu HTTP. Các đoạn mã ở đây sẽ giả định rằng bạn đã có máy chủ Node.js Express nơi các tính năng này đã được định cấu hình và khả dụng.

Cách đơn giản nhất để lưu trữ các thông báo do máy khách gửi là lưu trữ chúng trong một mảng toàn cầu mà trong đó, mỗi thông báo đã gửi trước đó sẽ được liên kết với một khóa nhận dạng duy nhất cho mỗi máy khách. Khóa này có thể được gửi đến máy khách dưới dạng cookie; cookie này sẽ được trình cho máy chủ khi có yêu cầu truy xuất các tin nhắn trước đó trong tương lai.

Tuy nhiên, phương pháp này có một điểm yếu: bởi các thông báo chỉ được lưu trữ trong một mảng toàn cục, tất cả các thông báo sẽ bị mất khi phiên máy chủ hiện tại kết thúc. Đây là một trong những lợi thế khi làm việc với cơ sở dữ liệu vì dữ liệu sẽ được lưu trữ liên tục và không bị mất nếu máy chủ được khởi động lại.

Bằng cách sử dụng tệp `index.js` làm tệp lệnh máy chủ chính, chúng ta có thể kết hợp mô-đun `sqlite3` và chỉ ra tệp đóng vai trò là cơ sở dữ liệu như sau:

```
const sqlite3 = require('sqlite3')
const db = new sqlite3.Database('messages.sqlite3');
```

Nếu nó chưa tồn tại, tệp `messages.sqlite3` sẽ được tạo trong cùng một thư mục với tệp `index.js`. Bên trong tệp duy nhất này, tất cả các cấu trúc và dữ liệu tương ứng sẽ được lưu trữ. Tất cả các hoạt động cơ sở dữ liệu được thực hiện trong tệp lệnh sẽ được trung gian bởi hằng số `db`, tức là tên được đặt cho đối tượng `sqlite3` mới sẽ mở tệp `messages.sqlite3`.

Cấu trúc của một Bảng

Không có dữ liệu nào có thể được chèn vào cơ sở dữ liệu cho đến khi ít nhất một bảng được tạo. Các bảng có thể được tạo bằng câu lệnh `CREATE TABLE`:

```
db.run('CREATE TABLE IF NOT EXISTS messages (id INTEGER PRIMARY KEY AUTOINCREMENT, uuid
CHAR(36), message TEXT)')
```

Phương thức `db.run()` được sử dụng để thực thi các câu lệnh SQL trong cơ sở dữ liệu. Bản thân câu lệnh sẽ được viết dưới dạng tham số cho phương thức. Mặc dù các câu lệnh SQL phải kết thúc bằng dấu chấm phẩy khi được nhập vào bộ xử lý dòng lệnh nhưng nó không bắt buộc trong các câu lệnh được truyền dưới dạng tham số trong một chương trình.

Vì phương thức `run` sẽ được thực hiện mỗi khi tệp lệnh được thực thi với `node index.js` nên câu lệnh SQL sẽ bao gồm mệnh đề điều kiện `IF NOT EXISTS` (NẾU KHÔNG TỒN TẠI) để tránh lỗi trong các lần thực thi sau này khi bảng `messages` đã tồn tại.

Các trường tạo nên bảng `message` là `id`, `uuid` và `message`. Trường `id` là một số nguyên duy nhất được sử dụng để xác định từng mục nhập trong bảng; do đó, trường này sẽ được tạo dưới dạng `PRIMARY KEY` (KHOÁ CHÍNH). Khóa chính không được để trống và không thể có hai khóa chính giống hệt nhau trong cùng một bảng. Do đó, hầu hết mọi bảng SQL đều có một khóa chính để theo dõi nội dung của bảng. Mặc dù có thể chọn giá trị cho khóa chính của bản ghi mới một cách rõ ràng (với điều kiện là nó chưa tồn tại trong bảng), sẽ thuận tiện hơn cho chúng ta nếu khóa được tạo tự động. Cờ `AUTOINCREMENT` trong trường `id` được sử dụng cho mục đích này.

NOTE

Việc cài đặt một cách rõ ràng khóa chính trong SQLite là tùy chọn vì bản thân SQLite sẽ tự động tạo khóa chính. Như đã nêu trong tài liệu SQLite: “Trong SQLite, các hàng của bảng thường có một số nguyên có dấu 64 bit `ROWID`. Số nguyên này sẽ là duy nhất trong số tất cả các hàng trong cùng một bảng. Nếu một bảng có chứa một cột thuộc loại `INTEGER PRIMARY KEY` (KHOÁ CHÍNH SỐ NGUYÊN) thì cột đó sẽ trở thành bí danh cho `ROWID`. Sau đó, bạn có thể truy cập `ROWID` bằng cách sử dụng bất kỳ tên nào trong số bốn tên khác nhau: ba tên ban đầu được mô tả ở trên hoặc tên được đặt cho cột `INTEGER PRIMARY KEY`. Tất cả các tên này đều là bí danh của nhau và có thể hoạt động hiệu quả như nhau trong mọi ngữ cảnh.”

Các trường `uuid` và `message` lần lượt lưu trữ nhận dạng khách hàng và nội dung thông báo. Trường loại `CHAR(36)` lưu trữ một lượng cố định 36 ký tự và trường loại `TEXT` lưu trữ văn bản có độ dài tùy ý.

Nhập Liệu

Chức năng chính của máy chủ trong ví dụ là lưu trữ các thông báo được liên kết với máy khách đã gửi chúng. Máy khách gửi thông báo trong trường `message` trong phần nội dung của yêu cầu được gửi bằng phương thức HTTP POST. Thông tin nhận dạng của khách hàng nằm trong cookie có tên `uuid`. Với thông tin này, chúng ta có thể viết tuyến Express để chèn thông báo mới vào cơ sở dữ liệu:

```
app.post('/', (req, res) => {

  let uuid = req.cookies.uuid

  if ( uuid === undefined )
    uuid = uuidv4()

  // Insert new message into the database
  db.run('INSERT INTO messages (uuid, message) VALUES (?, ?)', uuid, req.body.message)

  // If an error occurrs, err object contains the error message.
  db.all('SELECT id, message FROM messages WHERE uuid = ?', uuid, (err, rows) => {

    let expires = new Date(Date.now());
    expires.setDate(expires.getDate() + 30);
    res.cookie('uuid', uuid, { expires: expires })

    if ( req.headers.accept == "application/json" )
      res.json(rows)
    else
      res.render('index', {title: "My messages", rows: rows})

  })

})
```

Lần này, phương thức `db.run()` sẽ thực thi một câu lệnh chèn. Hãy lưu ý rằng `uuid` và `req.body.message` sẽ không được ghi trực tiếp vào dòng câu lệnh. Thay vào đó, các dấu chấm hỏi đã được thay thế cho các giá trị. Mỗi dấu chấm hỏi sẽ tương ứng với một tham số theo sau câu lệnh SQL trong phương thức `db.run()`.

Sử dụng dấu chấm hỏi làm trình giữ chỗ trong câu lệnh được thực thi trong cơ sở dữ liệu sẽ giúp SQLite dễ dàng phân biệt giữa các phần tử tĩnh của câu lệnh và dữ liệu biến của nó. Chiến lược này cho phép SQLite *thoát* hoặc *làm sạch* nội dung biến (là một phần của câu lệnh) và ngăn chặn

một vi phạm bảo mật phổ biến được gọi là *Lỗ hổng truy vấn SQL* (SQL injection). Trong vi phạm này, những người dùng độc hại sẽ chèn các câu lệnh SQL vào dữ liệu biến với hy vọng rằng các câu lệnh sẽ được thực thi một cách vô tình. Việc làm sạch sẽ ngăn chặn cuộc tấn công bằng cách vô hiệu hóa các ký tự nguy hiểm trong dữ liệu.

Truy vấn

Như được minh hoạ trong mã mẫu, mục đích của chúng ta là sử dụng cùng một tuyến để chèn thông báo mới vào cơ sở dữ liệu và để tạo danh sách các thông báo đã gửi trước đó. Phương thức `db.all()` sẽ trả về tập hợp tất cả các mục trong bảng khớp với tiêu chí được xác định trong truy vấn.

Không giống như các câu lệnh do `db.run()` thực hiện, `db.all()` sẽ tạo ra một danh sách các bản ghi được xử lý bởi hàm mũi tên được chỉ định trong tham số cuối cùng:

```
(err, rows) => {}
```

Hàm này cũng sẽ nhận hai tham số: `err` và `rows`. Tham số `err` sẽ được sử dụng nếu xảy ra lỗi ngăn cản việc thực hiện truy vấn. Khi thành công, tất cả các bản ghi đều có sẵn trong mảng `row` (trong đó, mỗi phần tử đều là một đối tượng tương ứng với một bản ghi từ bảng). Các đặc tính của đối tượng này sẽ tương ứng với các tên trường được chỉ ra trong truy vấn `uuid` và `message`.

Mảng `row` là một cấu trúc dữ liệu JavaScript. Do đó, nó có thể được sử dụng để tạo phản hồi bằng các phương thức do Express cung cấp, chẳng hạn như `res.json()` và `res.render()`. Khi được hiển thị bên trong mẫu EJS, một vòng lặp thông thường có thể liệt kê tất cả các bản ghi:

```
<ul>
<% rows.forEach( (row) => { %>
<li><strong><%= row.id %></strong>: <%= row.message %></li>
<% }) %>
</ul>
```

Thay vì lấp đầy mảng `rows` bằng tất cả các bản ghi do truy vấn trả về, trong một số trường hợp, việc xử lý từng bản ghi riêng lẻ bằng phương thức `db.each()` có lẽ sẽ thuận tiện hơn. Cú pháp phương thức `db.each()` cũng tương tự như phương thức `db.all()`, nhưng tham số `row` trong `(err, row) ⇒ {}` sẽ chỉ khớp với một bản ghi tại mỗi một thời điểm.

Thay đổi Nội dung của Cơ sở Dữ liệu

Cho đến nay, máy khách của chúng ta chỉ có thể thêm và truy vấn thông báo trên máy chủ. Vì máy khách hiện đã biết `id` của các thông báo đã gửi trước đó nên chúng ta có thể triển khai một chức năng để sửa đổi một bản ghi cụ thể. Thông báo đã sửa đổi cũng có thể được gửi đến một tuyến phương thức HTTP POST, nhưng lần này là với một tham số tuyến để nhận ra `id` do máy khách cung cấp trong đường dẫn yêu cầu:

```
app.post('/:id', (req, res) => {
  let uuid = req.cookies.uuid

  if ( uuid === undefined ){
    uuid = uuidv4()
    // 401 Unauthorized
    res.sendStatus(401)
  }
  else {

    // Update the stored message
    // using named parameters
    let param = {
      $message: req.body.message,
      $id: req.params.id,
      $uuid: uuid
    }

    db.run('UPDATE messages SET message = $message WHERE id = $id AND uuid = $uuid', param,
function(err){

      if ( this.changes > 0 )
      {
        // A 204 (No Content) status code means the action has
        // been enacted and no further information is to be supplied.
        res.sendStatus(204)
      }
      else
        res.sendStatus(404)

    })
  }
})
```

Tuyến này trình bày cách sử dụng các mệnh đề `UPDATE` và `WHERE` để sửa đổi một bản ghi hiện có. Một điểm khác biệt quan trọng so với các ví dụ trước đó là việc sử dụng *tham số đã được đặt tên*, trong đó, các giá trị được gộp vào một đối tượng duy nhất (`param`) và được truyền đến phương thức `db.run()` thay vì chỉ định riêng từng giá trị. Trong trường hợp này, tên trường (đứng trước `$`) sẽ là đặc tính của đối tượng. Tham số đã được đặt tên sẽ cho phép sử dụng tên trường (đứng trước `$`) làm trình giữ chỗ thay vì dấu chấm hỏi.

Một câu lệnh giống như câu lệnh trong ví dụ sẽ không gây ra bất kỳ thay đổi nào đối với cơ sở dữ liệu nếu điều kiện do mệnh đề `WHERE` áp đặt không khớp với một số bản ghi trong bảng. Để đánh giá xem có bất kỳ bản ghi nào đã bị sửa đổi bởi câu lệnh hay không, hàm gọi lại có thể được sử dụng làm tham số cuối cùng của phương thức `db.run()`. Bên trong hàm, ta có thể truy vấn số lượng bản ghi đã thay đổi từ `this.changes`. Hãy lưu ý rằng các hàm mũi tên không thể được sử dụng trong trường hợp này bởi chỉ các hàm thông thường có dạng `function(){}` mới xác định được đối tượng `this`.

Việc xóa một bản ghi rất giống với việc sửa đổi nó. Ví dụ: chúng ta có thể tiếp tục sử dụng tham số tuyến `:id` để xác định thông báo cần bị xóa, nhưng lần này là trong một tuyến được gọi bởi phương thức HTTP DELETE của máy khách:

```
app.delete('/:id', (req, res) => {
  let uuid = req.cookies.uuid

  if ( uuid === undefined ){
    uuid = uuidv4()
    res.sendStatus(401)
  }
  else {
    // Named parameters
    let param = {
    $id: req.params.id,
    $uuid: uuid
    }

    db.run('DELETE FROM messages WHERE id = $id AND uuid = $uuid', param, function(err){
      if ( this.changes > 0 )
        res.sendStatus(204)
      else
        res.sendStatus(404)
    })
  }
})
```

Bản ghi sẽ bị xóa khỏi bảng với mệnh đề `DELETE FROM`. Chúng ta sẽ lại sử dụng chức năng gọi lại để xác định xem có bao nhiêu mục nhập đã bị xóa khỏi bảng.

Đóng Cơ sở Dữ liệu

Sau khi được xác định, đối tượng `db` có thể được tham chiếu bất kỳ lúc nào trong quá trình thực thi tệp lệnh vì tệp cơ sở dữ liệu vẫn mở trong suốt phiên hiện tại. Việc đóng cơ sở dữ liệu trong khi tệp lệnh đang chạy thường không quá phổ biến.

Tuy nhiên, chức năng đóng cơ sở dữ liệu lại rất hữu ích trong việc tránh đóng cơ sở dữ liệu đột ngột khi quá trình máy chủ kết thúc. Mặc dù hiếm khi xảy ra nhưng việc tắt cơ sở dữ liệu đột ngột có thể dẫn đến sự không nhất quán nếu dữ liệu trong bộ nhớ chưa được chuyển giao cho tệp. Chẳng hạn, việc tắt cơ sở dữ liệu đột ngột dẫn đến mất dữ liệu có thể xảy ra nếu tệp lệnh bị người dùng kết thúc bằng cách nhấn phím tắt `Ctrl` + `C`.

Trong trường hợp sử dụng `Ctrl` + `C` vừa được mô tả, phương thức `process.on()` có thể chặn các tín hiệu do hệ điều hành gửi và thực hiện việc tắt có trật tự cả cơ sở dữ liệu và máy chủ:

```
process.on('SIGINT', () => {
  db.close()
  server.close()
  console.log('HTTP server closed')
})
```

Phím tắt `Ctrl` + `C` sẽ gọi tín hiệu hệ điều hành `SIGINT`; tín hiệu này sẽ chấm dứt chương trình tiền cảnh trong cửa sổ dòng lệnh. Trước khi kết thúc quá trình khi nhận được tín hiệu `SIGINT`, hệ thống sẽ gọi hàm gọi lại (tham số cuối cùng trong phương thức `process.on()`). Bên trong chức năng gọi lại, ta có thể đặt bất kỳ mã làm sạch nào, cụ thể là phương thức `db.close()` để đóng cơ sở dữ liệu và `server.close()` để tự đóng phiên bản Express một cách gọn gàng.

Bài tập Hướng dẫn

1. Mục đích của khóa chính trong bảng cơ sở dữ liệu SQL là gì?

2. Sự khác biệt giữa truy vấn bằng cách sử dụng `db.all()` và `db.each()` là gì?

3. Tại sao việc sử dụng trình giữ chỗ và không bao gồm dữ liệu do máy khách gửi trực tiếp trong câu lệnh hoặc truy vấn SQL lại quan trọng?

Bài tập Mở rộng

1. Phương pháp nào trong mô-đun `sqlite3` có thể được sử dụng để chỉ trả về một mục nhập trong bảng ngay cả khi truy vấn khớp với nhiều mục nhập?

2. Giả sử mảng `rows` được truyền dưới dạng tham số cho hàm gọi lại và chứa kết quả của một truy vấn được thực hiện với `db.all()`. Làm cách nào để một trường có tên `price` hiện ở vị trí đầu tiên của `row` có thể được tham chiếu bên trong hàm gọi lại?

3. Phương thức `db.run()` thực thi các câu lệnh sửa đổi cơ sở dữ liệu, chẳng hạn như `INSERT INTO` (CHÈN VÀO). Sau khi chèn một bản ghi mới vào bảng, làm thế nào để có thể truy xuất khóa chính của bản ghi mới được chèn?

Tóm tắt

Bài học này đã nói về cách sử dụng cơ bản của cơ sở dữ liệu SQL trong các ứng dụng Node.js Express. Mô-đun `sqlite3` cung cấp một cách đơn giản để lưu trữ dữ liệu liên tục trong cơ sở dữ liệu SQLite, trong đó một tệp duy nhất sẽ chứa toàn bộ cơ sở dữ liệu và không yêu cầu một máy chủ cơ sở dữ liệu chuyên dụng. Bài học này đã đi qua các khái niệm và quy trình sau:

- Cách thiết lập kết nối cơ sở dữ liệu từ Node.js.
- Cách tạo một bảng đơn giản và vai trò của các khóa chính.
- Sử dụng câu lệnh SQL `INSERT INTO` để thêm dữ liệu mới từ bên trong tệp lệnh.
- Truy vấn SQL sử dụng các phương thức SQLite tiêu chuẩn và các hàm gọi lại.
- Thay đổi dữ liệu trong cơ sở dữ liệu bằng cách sử dụng câu lệnh SQL `UPDATE` và `DELETE`.

Đáp án Bài tập Hướng dẫn

1. Mục đích của khóa chính trong bảng cơ sở dữ liệu SQL là gì?

   Khóa chính là trường nhận dạng độc nhất của mỗi bản ghi trong bảng cơ sở dữ liệu.

2. Sự khác biệt giữa truy vấn bằng cách sử dụng `db.all()` và `db.each()` là gì?

   Phương thức `db.all()` sẽ gọi hàm gọi lại với một mảng duy nhất chứa tất cả các mục nhập tương ứng với truy vấn. Phương thức `db.each()` sẽ gọi hàm gọi lại cho mỗi hàng kết quả.

3. Tại sao việc sử dụng trình giữ chỗ và không bao gồm dữ liệu do máy khách gửi trực tiếp trong câu lệnh hoặc truy vấn SQL lại quan trọng?

   Với trình giữ chỗ, dữ liệu do người dùng gửi sẽ được thoát trước khi được đưa vào truy vấn hoặc câu lệnh. Điều này sẽ cản trở các cuộc tấn công vào lỗ hổng cơ sở dữ liệu SQL mà trong đó, các câu lệnh SQL được đặt bên trong dữ liệu biến nhằm thực hiện các thao tác tùy ý trên cơ sở dữ liệu.

Đáp án Bài tập Mở rộng

1. Phương pháp nào trong mô-đun `sqlite3` có thể được sử dụng để chỉ trả về một mục nhập trong bảng ngay cả khi truy vấn khớp với nhiều mục nhập?

   Phương thức `db.get()` có cùng cú pháp như `db.all()` nhưng chỉ trả về mục nhập đầu tiên tương ứng với truy vấn.

2. Giả sử mảng `rows` được truyền dưới dạng tham số cho hàm gọi lại và chứa kết quả của một truy vấn được thực hiện với `db.all()`. Làm cách nào để một trường có tên `price` hiện ở vị trí đầu tiên của `row` có thể được tham chiếu bên trong hàm gọi lại?

   Mỗi mục trong `row` là một đối tượng có đăhc tính tương ứng với tên trường cơ sở dữ liệu. Vì vậy, giá trị của trường `price` trong kết quả đầu tiên sẽ là ở `rows[0].price`.

3. Phương thức `db.run()` thực thi các câu lệnh sửa đổi cơ sở dữ liệu, chẳng hạn như `INSERT INTO` (CHÈN VÀO). Sau khi chèn một bản ghi mới vào bảng, làm thế nào để có thể truy xuất khóa chính của bản ghi mới được chèn?

   Một hàm thông thường có dạng `function(){}` có thể được sử dụng làm hàm gọi lại của phương thức `db.run()`. Bên trong nó, đặc tính `this.lastID` sẽ chứa giá trị khóa chính của bản ghi được chèn cuối cùng.

Ấn bản

PDF được tạo vào: 2023-08-02

Tài liệu Học tập LPI là một sáng kiến của Linux Professional Institute (https://lpi.org). Tài liệu Học tập và các Bản dịch của chúng có thể được tìm thấy tại https://learning.lpi.org.

Đối với các câu hỏi và nhận xét về ấn bản này cũng như về toàn bộ dự án, hãy gửi email tới: learning@lpi.org.