XÂY DỰNG VIEW VÀ LAYOUT

.NET MVC

Nội dung

- Các khái niệm cơ bản
- HTML5, CSS3, JavaScript, thư viện Jquery
- ASP.NET Razor for C#

CÁC KHÁI NIỆM CƠ BẢN

View engine

- **View engine** là công cụ hoạt động giữa **view** và **trình duyệt (browser)** cung cấp khả năng hiển thị HTML từ **view** tới trình duyệt.
- Mặc định, **Asp.net MVC** đi kèm với hai phiên bản **Form (ASPX) và Razor View Engine**
  - **ASPX View Engine** là **view engine** mặc định cho ASP.NET MVC đi kèm với ASP.NET MVC từ MVC 1.0
  - **Razor View Engine** từ MVC3 hoặc phiên bản mới hơn.

View engine

- **ASPX view**
  - **ASPX View Engine** hay **Web Form View Engine**
- **Cú pháp** để hiển thị nội dung phía server:

“<% =%>” hoặc “<%:%> ”

Ví dụ:

```
<%
    string strVariable = "Hello World !!";
    bool boolVar = true;
    string[] arrVar = new string[] { "A", "B", "C"};
%>
```

View engine

- **Razor view engine**
  - Sử dụng ký tự @ để bắt đầu cho **inline expressions, single statement blocks, và multi-statement blocks.**

Ví dụ

```
<!-- Single statement blocks  -->
  @{ var RollNo = 7; }
  @{ var StudentName = "Raj Parihar";
  <!-- Inline expressions -->
  <p>The student RollNo is:= @RollNo </p>
  <p>Name := @StudentName</p>
  <!-- Multi-statement block -->
  @{
      var greeting = "Welcome to our site! Mr. " + @StudentName;
      var Department = "Your department is IT";
      var myMessage = greeting + " " + Department;
  }
```

View engine

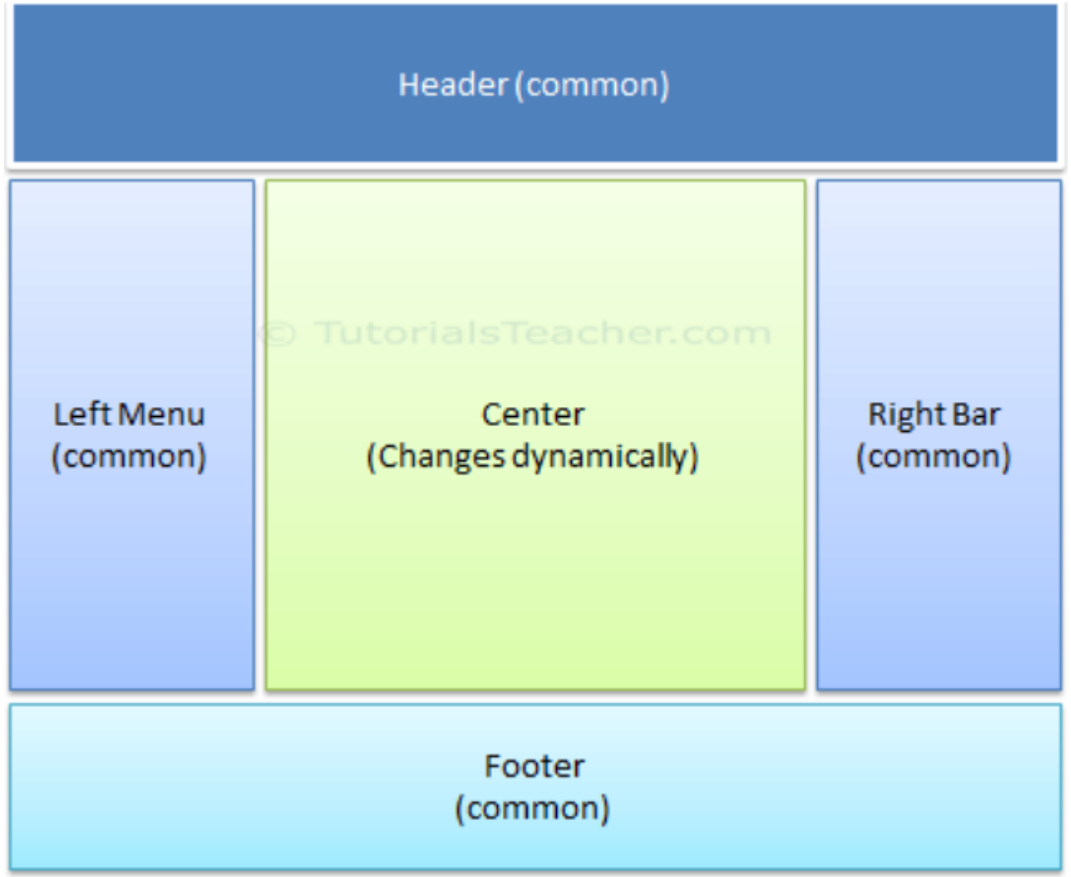
File extension	Description
.cshtml	C# Razor view. Supports C# code with html tags.
.vbhtml	Visual Basic Razor view. Supports Visual Basic code with html tags.
.aspx	ASP.Net web form
.ascx	ASP.NET web control

Layout View in ASP.NET MVC

- **Layout view**: Một ứng dụng có thể chứa một phần giao diện người dùng được giữ nguyên trong toàn bộ ứng dụng, gồm:
  - **header,**
  - **left navigation bar,**
  - **right bar,**
  - **footer section**.
- **Layout view** trong ASP.NET MVC chứa các phần **giao diện người dùng** chung này để không phải lặp lại code

Layout View in ASP.NET MVC

- **Layout view**:

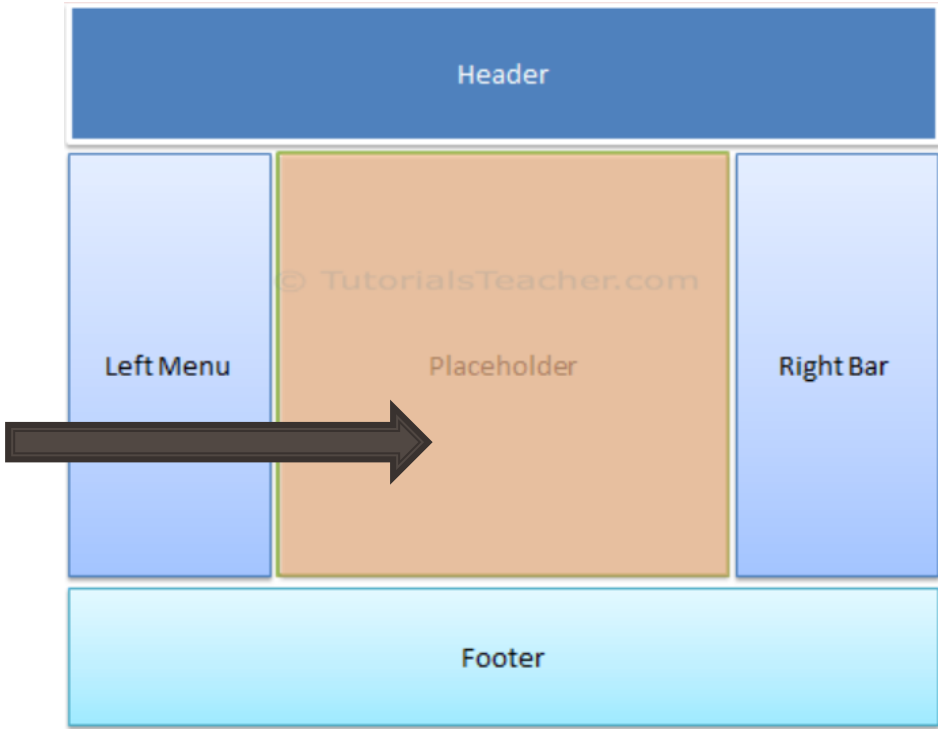


Layout View in ASP.NET MVC

- **Layout view**: xác định một template chung, template này có thể được kế thừa trong nhiều views để cung cấp giao diện nhất quán trong một ứng dụng.
- **Layout views** được lưu trữ trong thư mục **Shared**. Mặc định, một layout view **_Layout.cshtml** được tạo khi tạo ứng dụng MVC trong Visual Studio,

Layout View in ASP.NET MVC

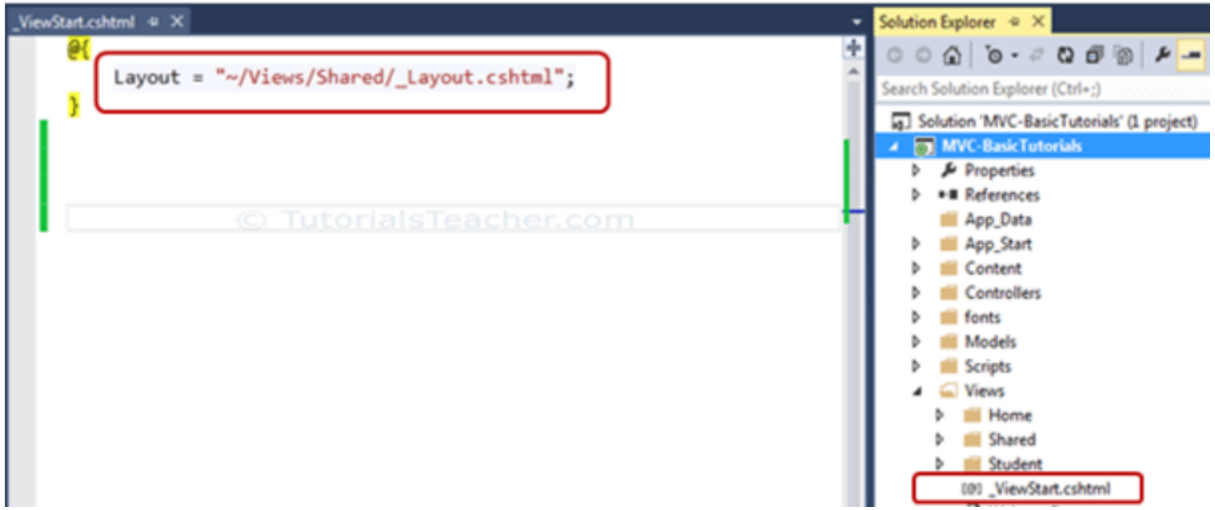
- **Sử dụng Layout View:** Các views sẽ được hiển thị trong placeholder **RenderBody().**
- Có nhiều cách để chỉ định **layout view** nào sẽ được sử dụng:
  - ❖ Chỉ định trong **_ViewStart.cshtml**
  - ❖ Hoặc trong phương thức **action**.



Layout View in ASP.NET MVC

- **ViewStart**
  - Mặc định **_ViewStart.cshtml** được tạo trong thư mục **Views,** dùng để chỉ định các thiết lập chung cho tất cả các view trong một thư mục và các thư mục con nơi nó được tạo.

Ví dụ: _ViewStart.cshtml trong thư mục Views thiết lập Layout property: **"~/Views/Shared/_Layout.cshtml"**



Layout View in ASP.NET MVC

- **Chỉ định Layout Page trong Action Method**.
  - ❖ Chỉ định tên **layout view** làm tham số thứ hai trong phương thức **View ()**. Mặc định, **layout view** sẽ được tìm kiếm trong thư mục **Share**.

Ví dụ:

```
public class HomeController : Controller {
      public ActionResult Index() {
            //set "_myLayoutView" as layout view
           return View("Index", "_myLayoutPage");
      }
}
```

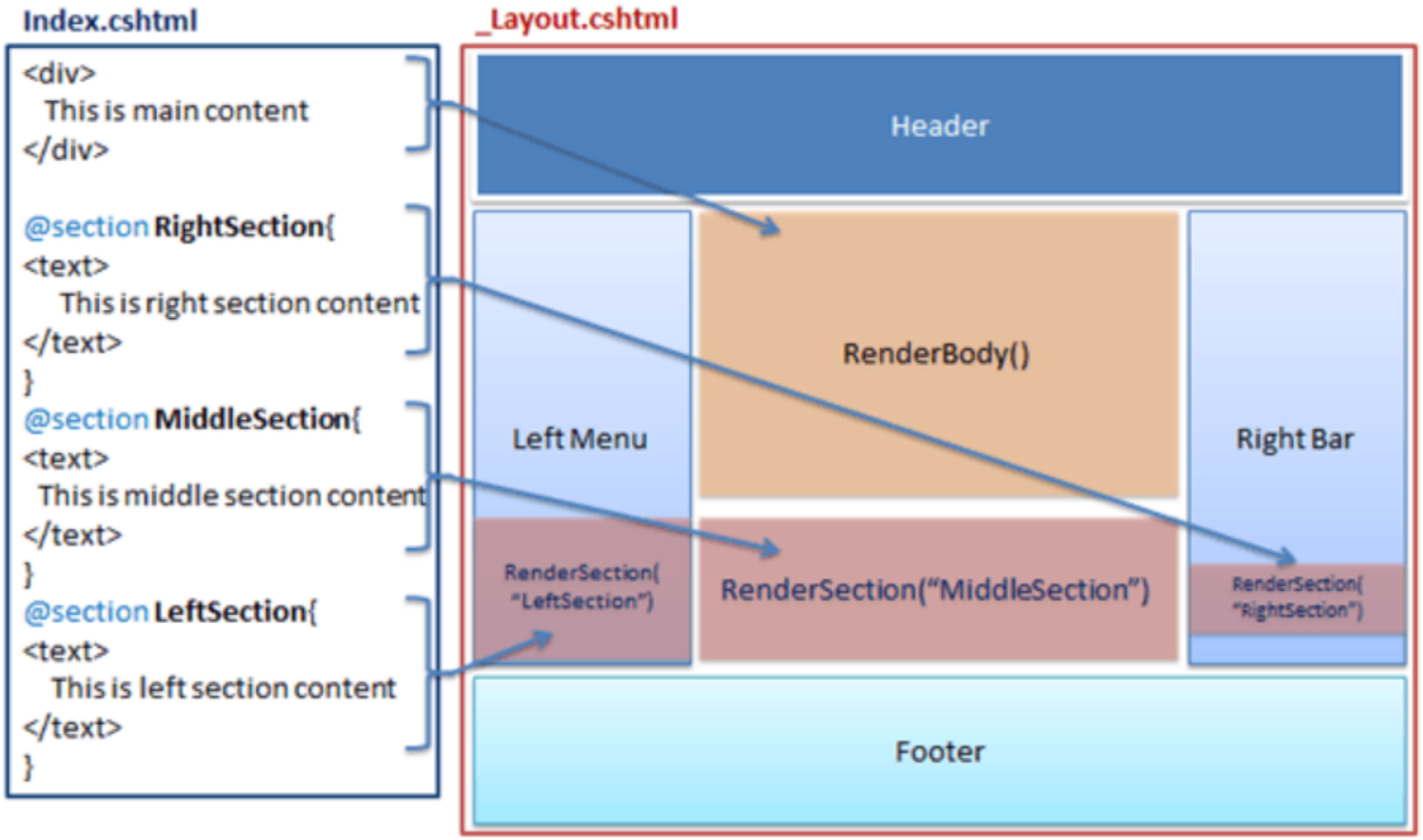
Layout View in ASP.NET MVC

- **Rendering Methods**: **ASP.NET MVC layout view** sử dụng các phương thức để hiển thị View

Method	Description
RenderBody()	Hiển thị view không chứa tên section. **Layout view** cần phải có phương thức RenderBody() .
RenderSection(string name)	Renders nội dung gồm tên section và chỉ định section được yêu cầu

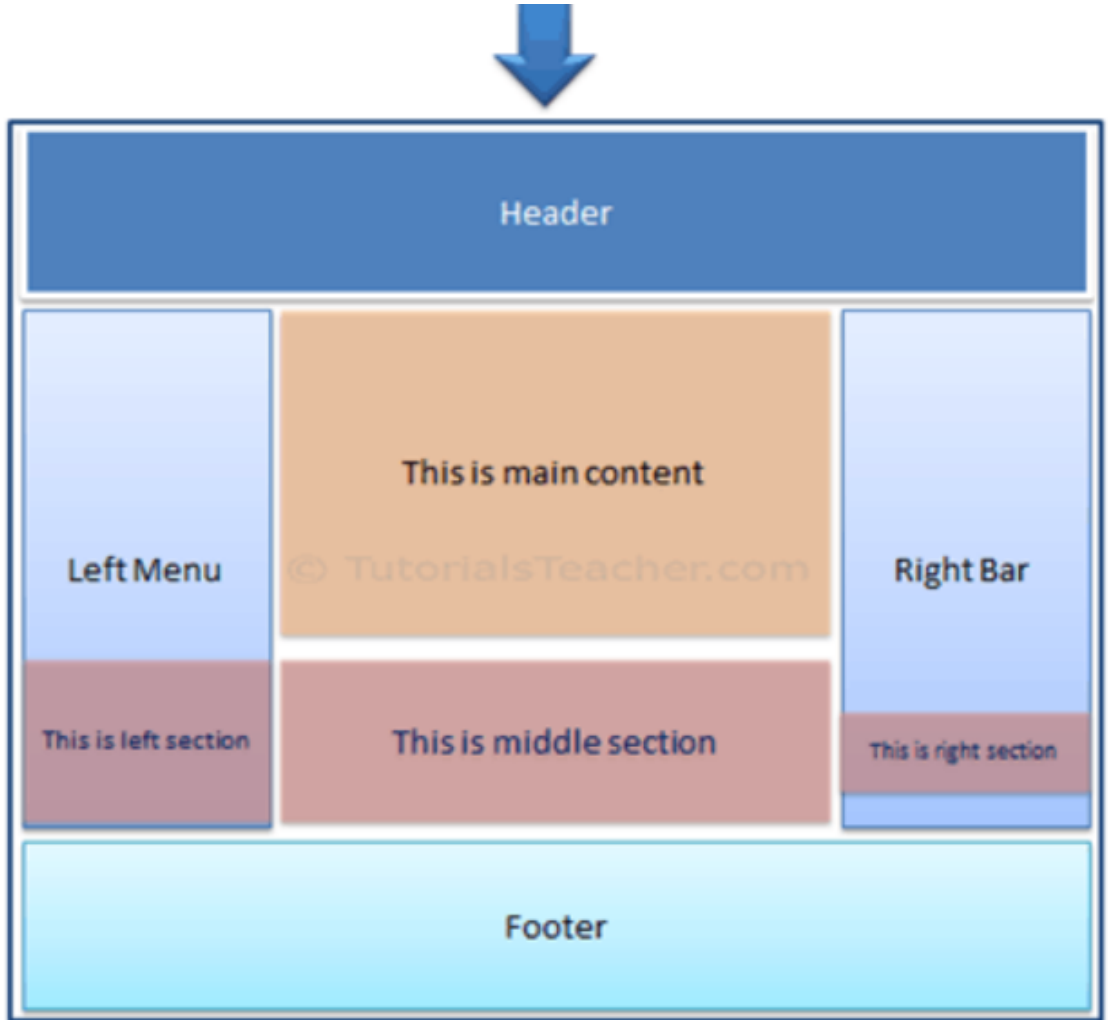
Layout View in ASP.NET MVC

- **RenderBody() và RenderSection().**



Layout View in ASP.NET MVC

- **RenderBody() và RenderSection().**



Layout View in ASP.NET MVC

- **RenderBody() và RenderSection().**
  - **Trong _Layout.cshtml** gồm **RenderBody()** và **RenderSection()**.
  - Trong **Index.cshtml** chứa tên sections sử dụng **@section, tên của mỗi** section trùng với tên được chỉ định trong **RenderSection()** của layout view **_Layout.cshtml**,
  - **@Section RightSection**. Tại thời điểm thực thi, các sections của Index.cshtml được đặt tên: **LeftSection, RightSection**, và **MiddleSection** sẽ rendered tại vị trí tương ứng khi **RenderSection()** được gọi.
  - Phần còn lại của view **Index.cshtml,** không nằm trong bất kỳ phần nào được đặt tên, sẽ được hiển thị trong **RenderBody ()**

Layout View in ASP.NET MVC

- **Tạo Layout view**: có thể tạo **Layout view** trong bất kỳ thư mục nào trong thư mục View. Tuy nhiên, nên tạo tất cả các **Layout view** trong thư mục **Share** để dễ bảo trì.

- Cách tạo **Layout view:**
  - **Trong** Visual Studio, click phải vào thư mục **Share->** chọn **Add** -> click **New Item...**
  - Trong hộp thoại **Add New Item**
    - Chọn **MVC 5 Layout Page (Razor) template**
    - Nhập tên cho **Layout view,**

    Ví dụ: _myLayoutPage.cshtml click **Add.**

Layout View in ASP.NET MVC

- **Tạo Layout view**: code **_myLayoutPage.cshtml** có dạng

```
<!DOCTYPE html>
<html>
      <head>
      <meta name="viewport" content="width=device-width"/>
      <title>@ViewBag.Title</title>
</head>
<body>
      <div>
            @RenderBody()
      </div>
</body>
</html>
```

Layout View in ASP.NET MVC

- **Tạo Layout view**:
  - Thêm thẻ **<footer>** với phương thức **RenderSection ("footer", true).** Khi đó, bất kỳ View nào sử dụng **_myLayoutPage** làm dạng Layout view đều bao gồm phần **Footer**.

```
<body>
    <div>
        @RenderBody()
    </div>
    <footer class="panel-footer">
        @RenderSection("footer", true)
    </footer>
</body>
```

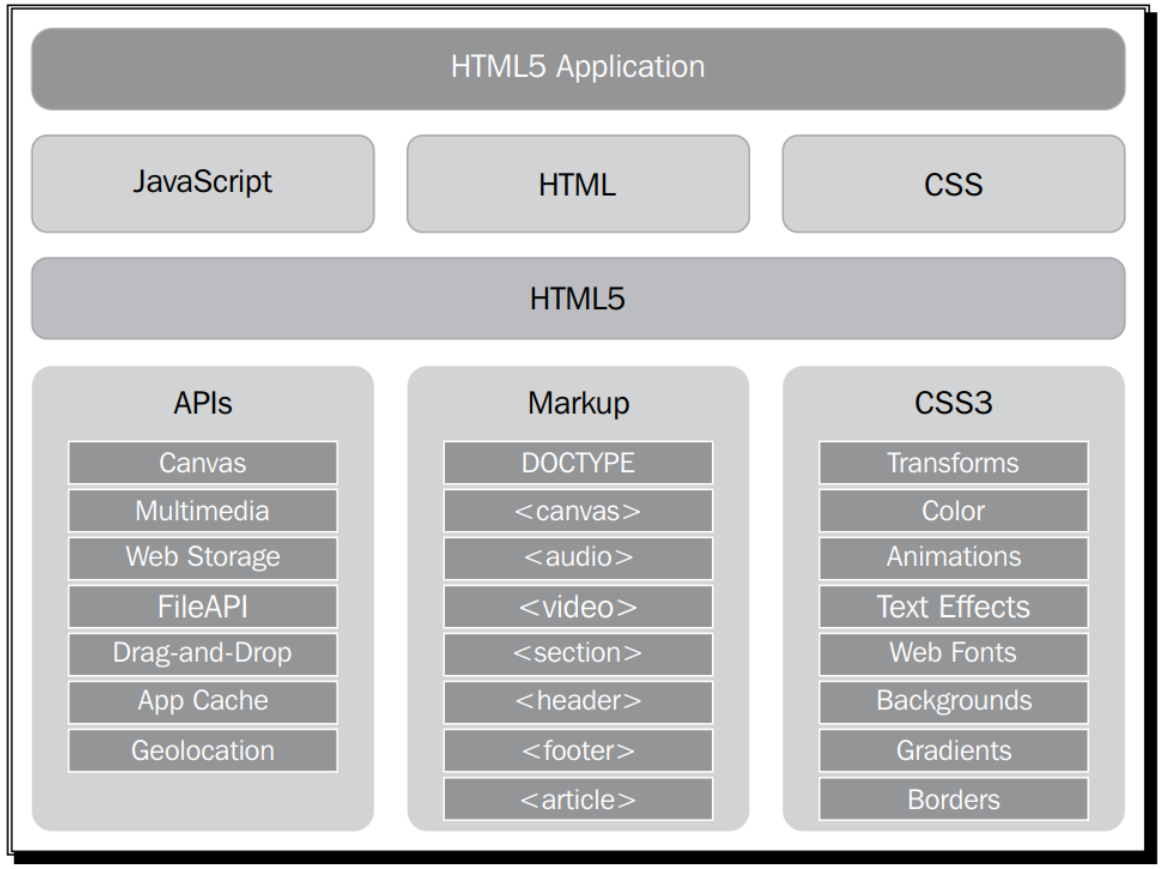
HTML5, CSS3, JAVASCRIPT, THƯ VIỆN JQUERY

GIỚI THIỆU HTML5

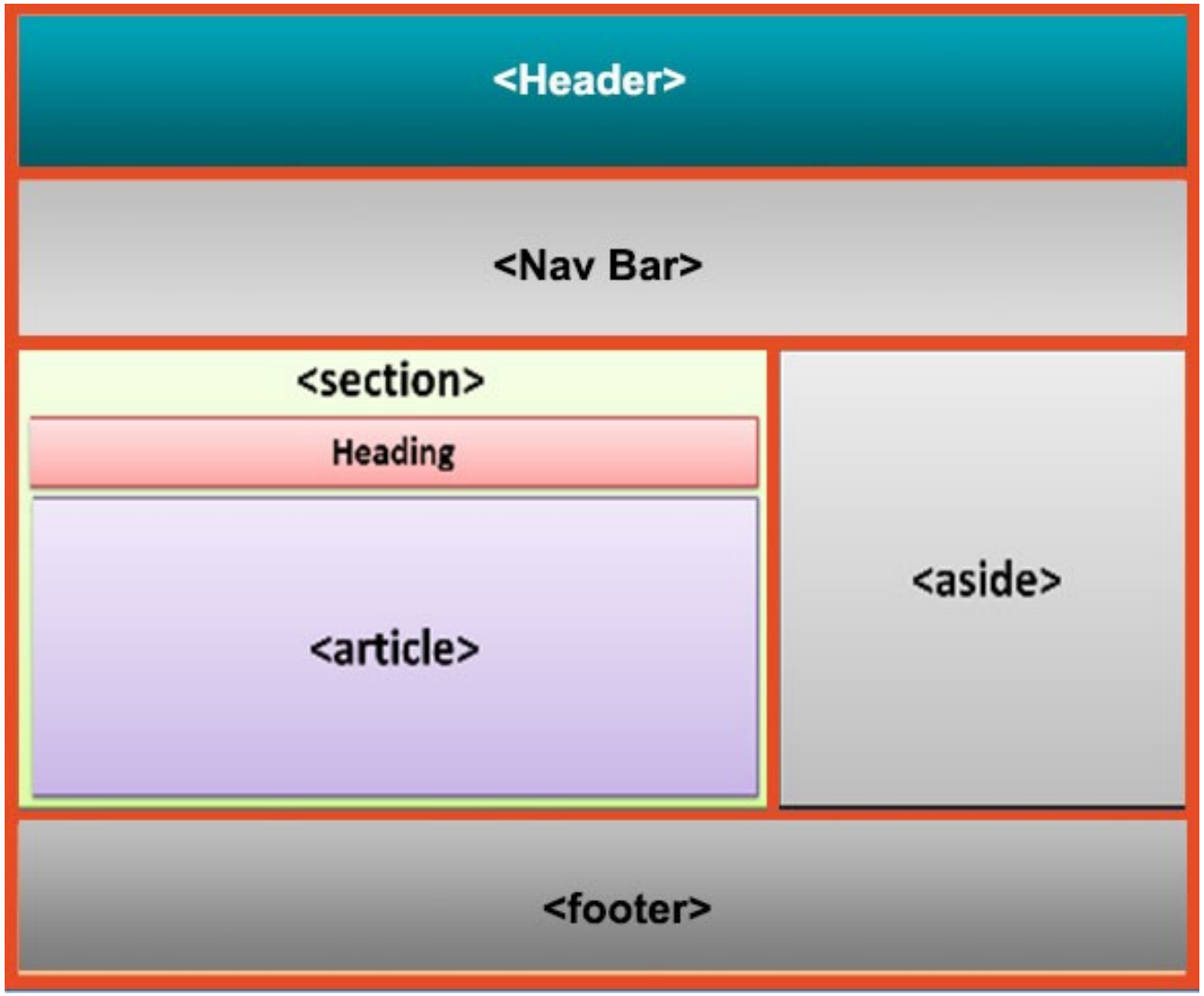
- **HTML5** là ngôn ngữ đánh dấu được sử dụng để cấu trúc và trình bày nội dung trên ***World Wide Web***. Đây là phiên bản thứ năm và hiện tại của tiêu chuẩn HTML.
- **HTML5 hỗ trợ đa phương tiện** và tạo sự nhất quán giữa máy tính và trình duyệt web.
- **Các thẻ mới trong HTML5**: <main>, <section>, <article>, <header>, <footer>, <aside>, <nav> và <figure>,
- **Khai báo ở đầu trang HTML**: **<! DOCTYPE html>**
- **Khai báo mã hóa ký tự:** `<meta charset="UTF-8">`

GIỚI THIỆU HTML5

- **Các thành phần của một ứng dụng HTML5**



Bố cục cơ bản của HTML5



Bố cục cơ bản của HTML5

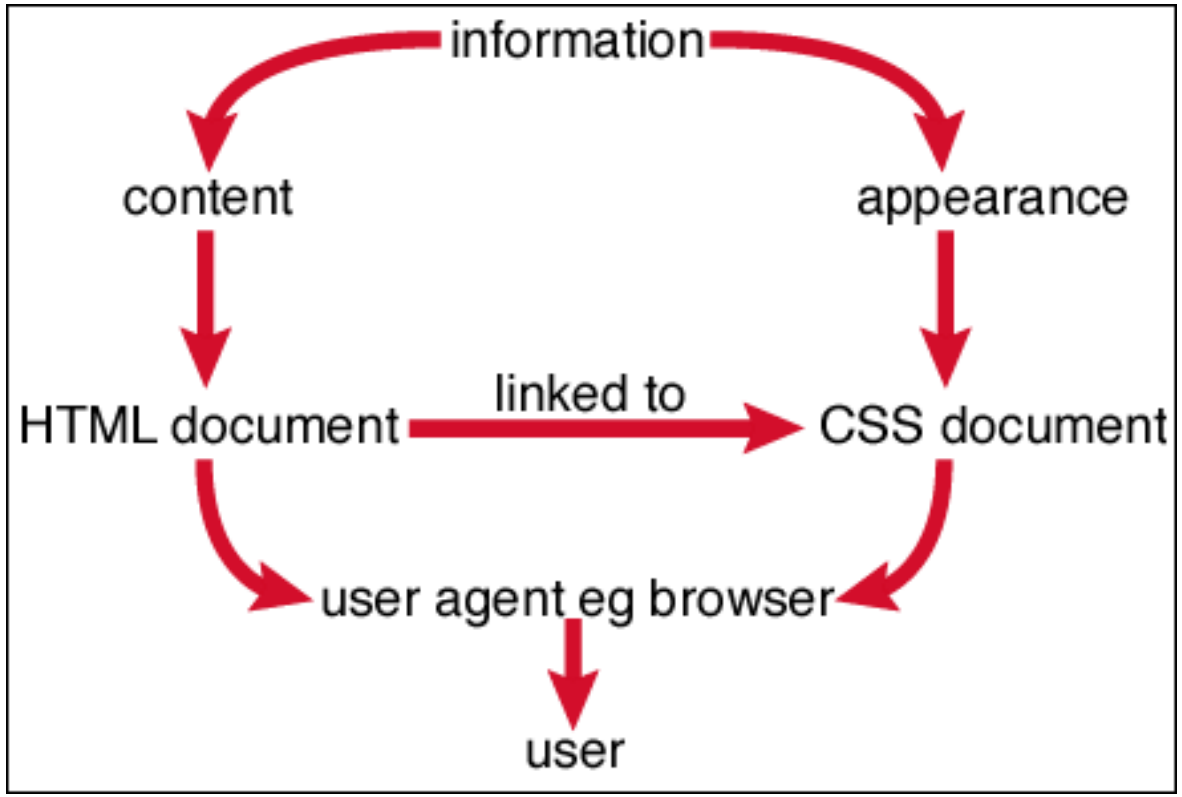
Tag	Description
<header>	Chỉ định tiêu đề của trang web hoặc một phần của trang
<nav>	xác định một tập hợp các liên kết điều hướng.
<section>	xác định một phần trong tài liệu.
<article>	Chỉ định nội dung độc lập, khép kín.
<aside>	Xác định nội dung bên ngoài từ nội dung của trang khác
<figure>	Chỉ định nội dung độc lập, như hình minh họa, sơ đồ, ảnh, danh sách mã, v.v.
<footer>	Xác định chân trang cho tài liệu hoặc phần

Cascading Style Sheets

- **Vai trò của Cascading Style Sheets**
  - **Tách** phần trình bày tài liệu khỏi nội dung tài liệu
  - **Cung cấp** các tính năng khác để định dạng giao diện
    - Có thể xác định font, size, background color, background image, margins, etc.
  - **Chia sẻ** các định dạng cho toàn bộ Web site
    - Giảm thời gian phát triển và bảo trì
  - **Chỉ định** một **định nghĩa lớp** cho một kiểu, xác định hiệu quả các phần tử HTML mới
  - **Linh hoạt**: các quy tắc được áp dụng theo cách phân cấp (quy tắc ưu tiên)

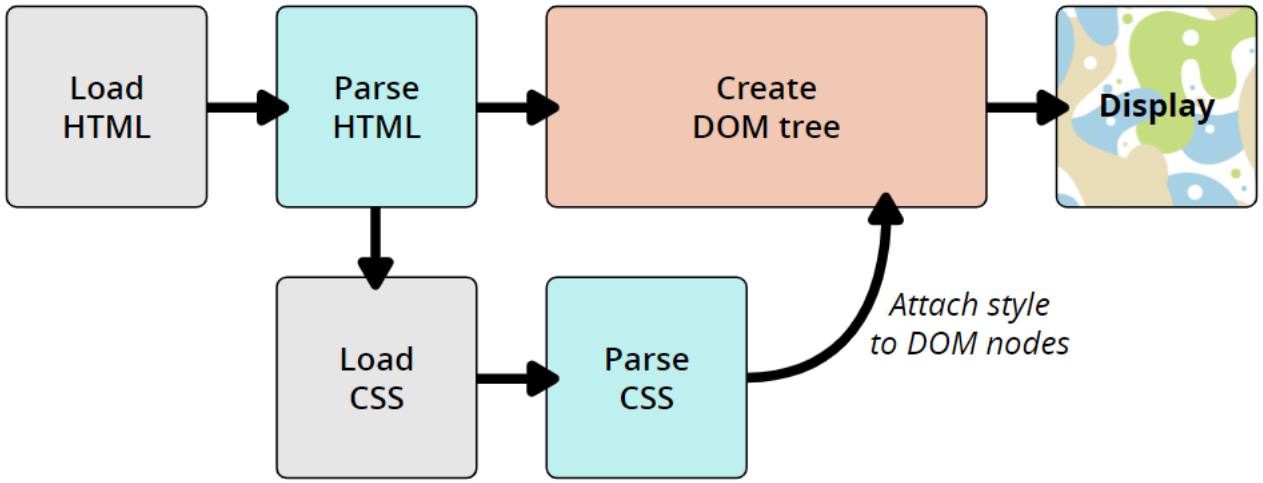
Cascading Style Sheets

- **Cách hoạt động của Style sheet**: Sử dụng **style sheets** tách biệt nội dung (content) và phần trình bày (appearance)



Cascading Style Sheets

- **Cascading Style Sheet** là một chuỗi **các lệnh (statements),** mỗi lệnh thực hiện 2 việc
  - ❖ Xác định các phần tử trong tài liệu HTML mà nó ảnh hưởng đến
  - ❖ Cho trình duyệt biết cách hiển thị các phần tử HTML
- Khi trình duyệt hiển thị trang web thì phải kết hợp nội dung của tài liệu với thông tin **Style Sheet**



CSS support styles

- **CSS Level 1 (1996)**
  - **Font properties**: such as typeface and emphasis
  - **Color of text:** backgrounds, and other elements
  - **Text attributes:** spacing between words, lines
  - **Alignment** of text, images, tables, etc.
  - **Margin**, border, padding, and positioning of most elements
  - Dimension
- **CSS Level 2 (1998)**
  - Relative and fixed positioning of most elements
  - Bidirectional texts
  - New font properties

CSS support styles

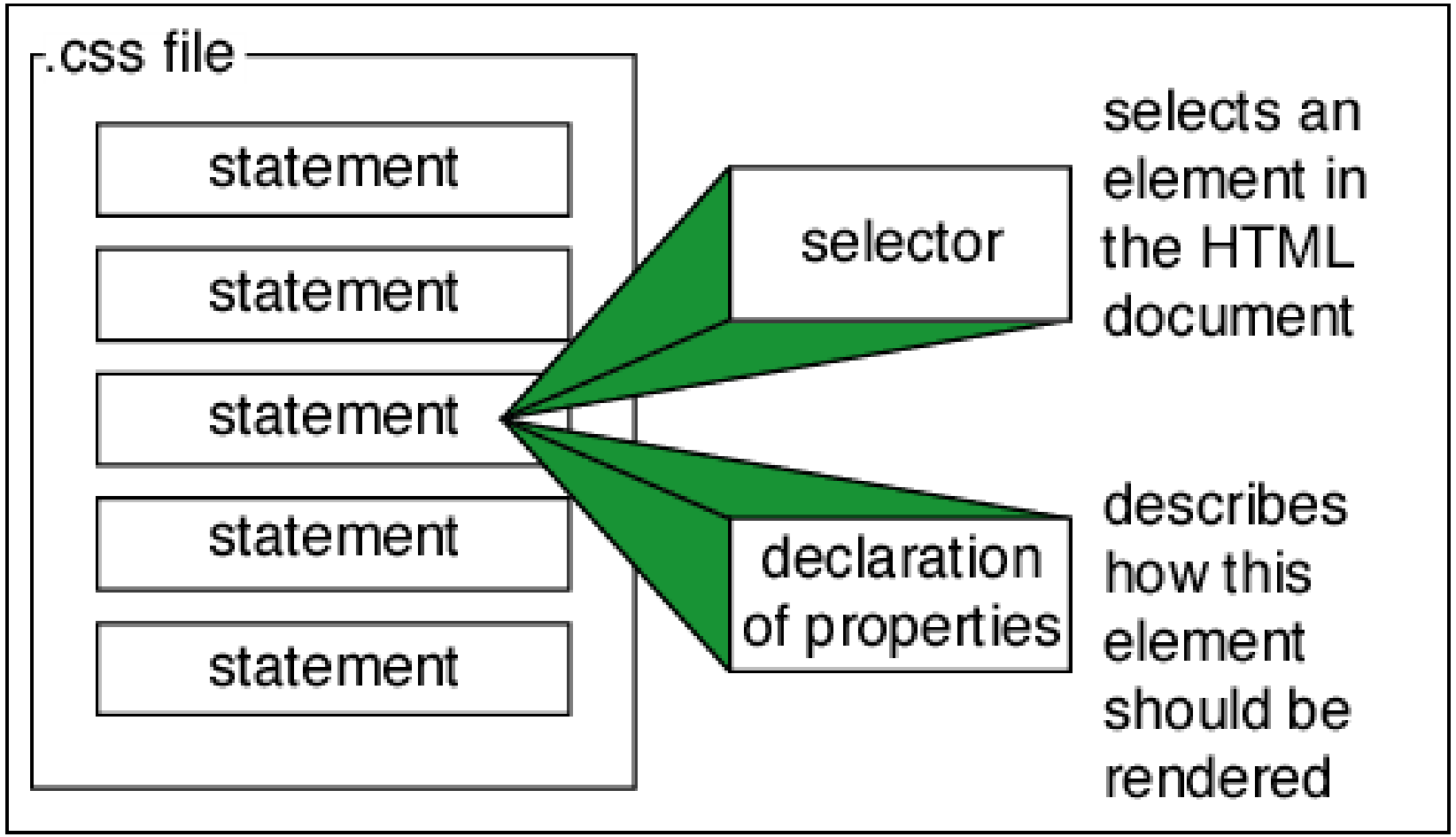
- **CSS 3** (là một tiêu chuẩn mới nhất của các phiên bản css)
  - ❖ Kết hợp và mở rộng CSS-2
  - ❖ Tập trung vào mô-đun hóa đặc tả CSS
  - ❖ Bộ chọn mới, đường viền và hình nền lạ mắt, văn bản dọc, tương tác người dùng, giọng nói ….
  - ❖ **CSS3 modules**

- **Selectors**
- **Box Model**
- **Backgrounds**
- **Image Values and Replaced Content**
- **Text Effects**
- **2D Transformations**
- **3D Transformations**
- **Animations**
- **Multiple Column Layout**
- **User Interface**

Các thành phần của một style sheet

- **Câu lệnh (statements):** là một chuỗi hướng dẫn trong CSS
- **Phần tử (elements):** là nội dung bên trong các thẻ HTML.
- **Bộ chọn (selectors):** Phần của **câu lệnh (statement)** xác định các phần tử của trang web
- **Phần khai báo (declaration)**: Phần của một câu lệnh cho trình duyệt biết cách các phần tử đã chọn hiển thị trên trình duyệt. **Một khai báo** chứa các Style sẽ được áp dụng cho phần tử đã chọn.

Các thành phần của một style sheet



Các thành phần của một style sheet

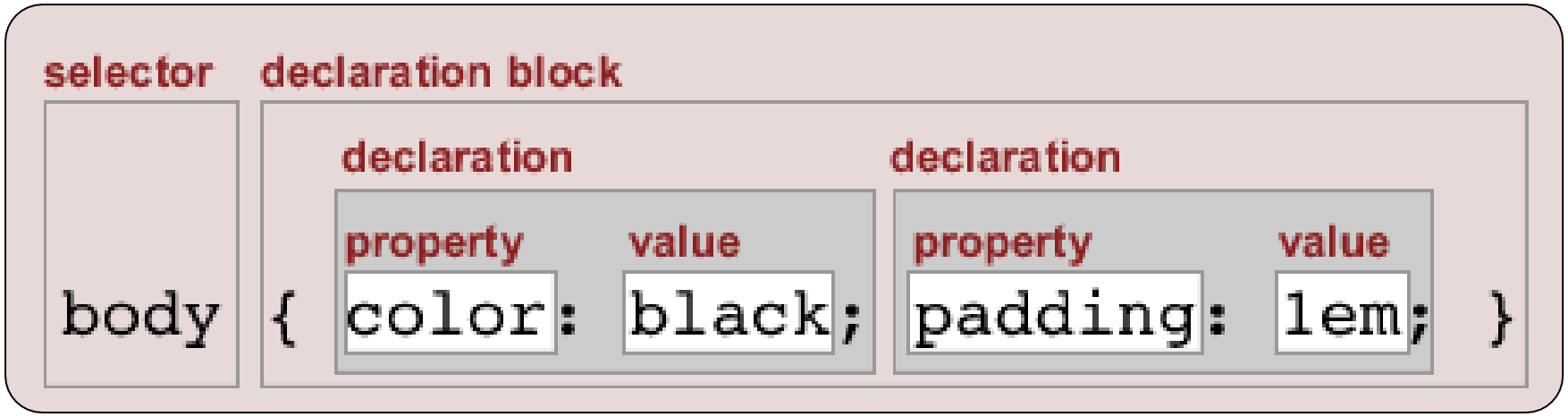
- Ví dụ: **statement**

```
body {
    font-family: Verdana,"Minion Web", Helvetica, sans-serif;
    font-size: 1em;
    text-align: justify;
    }
```

- **body**: selector
- Câu lệnh (**statement**) ảnh hưởng đến các **phần tử <body>** của những trang có liên kết đến style sheet này
- Câu lệnh (**statement**): có một khai báo (**declaration**) gồm 3 thuộc tính: **font-family, font-size, text-align**

Cú pháp của CSS

- **Một câu lệnh (Statement)** có một hoặc nhiều bộ chọn (selectors) và các khai báo (declaration)
  - **Bộ chọn (Selector):** chỉ định phần tử HTML
  - **Khai báo (Declaration):** là một hoặc nhiều thuộc tính được phân tách bằng dấu chấm phẩy "**;**".
  - **Thuộc tính (Property):** gồm tên và giá trị được phân tách bằng dấu hai chấm "**:**".



Giới thiệu về DHTML (*Dynamic HTML)*

- **HTML** xác định nội dung trang Web thông qua các thẻ ngữ nghĩa: headings, paragraphs, lists, …
- **CSS** xác định ‘rules' hoặc ‘Style' để trình bày mọi khía cạnh của tài liệu HTML:
  - ❖ Font (family, size, color, weight, etc.)
  - ❖ Background (color, image, position, repeat)
  - ❖ Position và layout: của bất kỳ đối tượng nào trên trang
- **JavaScript** xác định hành vi động tương tác với người dùng, để xử lý các sự kiện, v.v.

JavaScript

- **JavaScript** là một **ngôn ngữ kịch bản front-end** do **Netscape** phát triển cho nội dung động
  - ❖ Nhẹ, nhưng với khả năng hạn chế
  - ❖ Có thể được sử dụng như ngôn ngữ hướng đối tượng
  - ❖ Công nghệ Client-side
  - ❖ Được nhúng vào trang HTML
  - ❖ Được thông dịch bởi trình duyệt Web
  - ❖ Đơn giản và linh hoạt
  - ❖ Mạnh mẽ để thao tác DOM (Document Object Model)

JavaScript

- **Chức năng JavaScript**
  - Triển khai xác thực biểu mẫu - Form
  - Đáp trả các hành động của người dùng, ví dụ: xử lý các phím
  - Thay đổi hình ảnh khi di chuyển chuột qua nó
  - Các phần của trang xuất hiện và biến mất, nội dung tải và thay đổi động
  - Thực hiện các phép tính phức tạp
  - Các điều khiển HTML tùy chỉnh, ví dụ: bảng có thể cuộn
  - Triển khai chức năng AJAX

JavaScript

- **Hoạt động của JavaScript**
  - ❖ Xử lý các sự kiện
  - ❖ Đọc và ghi các phần tử HTML và sửa đổi cây DOM
  - ❖ Xác thực dữ liệu biểu mẫu
  - ❖ Truy cập hoặc sửa đổi cookie của trình duyệt
  - ❖ Phát hiện trình duyệt và hệ điều hành của người dùng
  - ❖ Được sử dụng như ngôn ngữ hướng đối tượng
  - ❖ Xử lý các trường hợp ngoại lệ
  - ❖ Thực hiện các cuộc gọi máy chủ không đồng bộ (AJAX)

Mã javascript trong HTML

- **Phần tử <script>:** Mã javascript đặt trong cặp thẻ **<script>**

```
<head>
        <Script language=”JavaScript”>
                Javascript Comments
        </script>
</head>
```

Mã javascript trong HTML

- **Phần tử <script>:** có thể đặt ở bất kỳ vị trí nào trong trang HTML, nhưng tốt nhất là đặt trong phần **<head>**

- **Ví dụ:**

```
<head>
    <script>
      var x = 5;
      var y = 10;
      var sum = x + y;
      document.write(sum); //in giá trị của biến
    </script>
 </head>
```

Tạo tập tin JavaScript

- **Tập tin Javascript**
  - ❖ Mã Javascript có thể đặt trong tập tin có phần mở rộng **.js**
  - ❖ Không chứa phần tử <script>
- **Cách truy cập tập tin JavaScript**
  - ❖ Trong phần <Head> của trang HTML, tạo liên kết đến tập tin Javascript

```
<script src=”fileJavascript.js” type="text/javscript" >
                JavaScript program
</script>
```

Thực thi mã Javascript

- **Mã JavaScript thực thi** trong quá trình **tải trang** hoặc khi **trình duyệt kích hoạt một sự kiện**
  - ❖ Tất cả các câu lệnh được thực thi khi tải trang
  - ❖ Một số câu lệnh chỉ xác định các hàm có thể được gọi sau này
  - ❖ Mã hoặc lệnh gọi hàm có thể được đính kèm dưới dạng "trình xử lý sự kiện" thông qua thuộc tính thẻ
  - ❖ Được thực thi khi **sự kiện** được **trình duyệt kích hoạt**

```
<img src="logo.gif" onclick="alert('clicked!')" />
```

Thực thi mã Javascript

- Ví dụ:

```
<html>
<head>
   <script type="text/javascript">
     function test (message) {
       alert(message);
     }
   </script>
</head>
<body>
  <img src="logo.gif“ onclick="test('clicked!')" />
</body>
</html>
```

Thực thi mã Javascript

- Ví dụ:

```
<html>
<head>
  <script src="sample.js type="text/javascript">
  </script>
</head>
<body>
  <button onclick="sample()" value="Call JavaScript
    function from sample.js" />
</body>
</html>
```

Tập tin js

```
function sample() {
  alert('Hello from sample.js!')
}
```

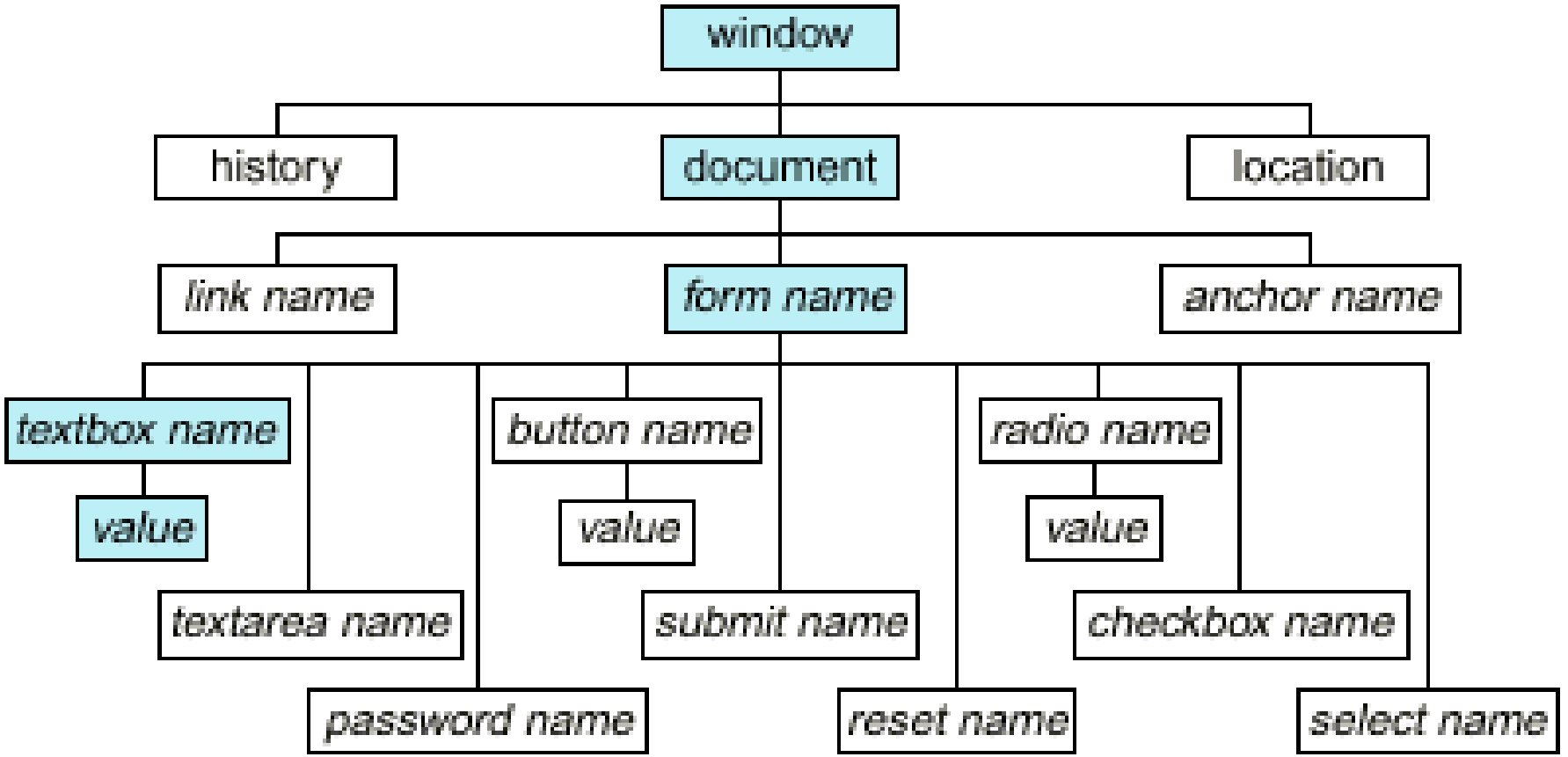
Tổng quan về DOM

- **Mỗi trang web** hiển thị trên cửa sổ trình duyệt được coi là một đối tượng.
- **Đối tượng Document:** đại diện cho **tài liệu HTML** được hiển thị trong cửa sổ trình duyệt. Đối tượng Document có nhiều thuộc tính tham chiếu đến các đối tượng khác cho phép truy cập và sửa đổi nội dung tài liệu.
- Cách mà nội dung tài liệu được truy cập và sửa đổi được gọi là **Mô hình Đối tượng Tài liệu - DOM**. Các đối tượng được tổ chức theo một hệ thống phân cấp. Cấu trúc phân cấp này áp dụng cho việc tổ chức các đối tượng trong tài liệu Web.

Tổng quan về DOM

- **Mô hình Đối tượng Tài liệu - DOM**
  - **Window object**: Trên cùng của hệ thống phân cấp, là phần tử ngoài cùng của hệ thống phân cấp đối tượng.
  - **Document object**: tài liệu HTML được tải trên cửa sổ trình duyệt là một đối tượng tài liệu. Tài liệu chứa nội dung của trang.
  - **Form object: <form> ... </form>:** chứa tất cả các phần tử được xác định cho đối tượng đó như ext fields, buttons, radio buttons, and checkboxes.

Tổng quan về DOM



Tổng quan về DOM

- **Tính chất của DOM**:
  - Mọi phần tử HTML đều có thể truy cập được qua JavaScript DOM API
  - Hầu hết các đối tượng DOM có thể được điều khiển bởi lập trình viên
  - Mô hình sự kiện cho phép một tài liệu phản ứng khi người dùng làm điều gì đó trên trang
- **Thuận lợi**
  - Tạo các trang tương tác
  - Cập nhật các đối tượng của một trang mà không cần tải lại nó

Truy cập các phần tử của DOM

- **Truy cập các phần tử thông qua thuộc tính ID**

```
var elem = document.getElementById("some_id")
```

- **Qua thuộc tính name**

```
var arr = document.getElementsByName("some_name")
```

- **Qua tên thẻ**

```
var imgTags = el.getElementsByTagName("img")
```

Thao tác với các phần tử trên DOM

- Khi truy cập vào một phần tử trong DOM, thì có thể lấy giá trị hoặc gán giá trị cho các thuộc tính của phần tử đó
- **Thuộc tính của các phần tử của DOM** đều bắt nguồn từ các thuộc tính của thẻ HTML:
  - **id, name, href, alt, title, src,** etc…
  - **thuộc tính style:** hiệu chỉnh CSS của phần tử, chính là inline Style.
  - Một số thuộc tính được nhúng hoặc bên ngoài: `style.width, style.marginTop, style.backgroundImage`

Thao tác với các phần tử trên DOM

- Ví dụ:

```
function change(state) {
  var lampImg = document.getElementById("lamp");
  lampImg.src = "lamp_" + state + ".png";
  var statusDiv = document.getElementById("statusDiv");
  statusDiv.innerHTML = "The lamp is " + state";
}
…
<img src="test_on.gif" onmouseover="change('off')"
  onmouseout="change('on')" />
```

Thao tác với các phần tử trên DOM

- **class = className:** chỉ định các phần tử HTML thuộc 1 lớp.
- **innerHTML:** giữ tất cả toàn bộ mã HTML bên trong phần tử
- **Read-only:** trạng thái của phần tử
  - `tagName`, `offsetWidth`, `offsetHeight`, `scrollHeight`, `scrollTop`, `nodeType`, etc…

Thao tác với các phần tử trên DOM

- **Truy cập các phần tử thông qua cấu trúc cây DOM:** có thể truy cập các phần tử trong DOM thông qua một số thuộc tính thao tác trên cấu trúc của mô hình DOM
  - `element.childNodes`
  - `element.parentNode`
  - `element.nextSibling`
  - `element.previousSibling`
  - `element.firstChild`
  - `element.lastChild`

Thao tác với các phần tử trên DOM

- **Truy cập các phần tử thông qua cấu trúc cây DOM:**

  Ví dụ:

```
var el = document.getElementById('div_tag');
alert (el.childNodes[0].value);
alert (el.childNodes[1].getElementsByTagName('span').id);
…
<div id="div_tag">
  <input type="text" value="test text" />
  <div>
    <span id="test">test span</span>
  </div>
</div>
```

HTML DOM Event

- **HTML DOM event** cho phép JavaScript thiết lập các trình xử lý sự kiện khác nhau trên các phần tử trong tài liệu HTML.
  - ❖ Sự kiện được trình duyệt kích hoạt và được gửi đến hàm xử lý sự kiện JavaScript được chỉ định
  - ❖ Có thể được đặt bằng các thuộc tính HTML
  - ❖ Có thể được truy cập thông qua DOM

```
<img src="test.gif" onclick="imageClicked()" />
```

```
var img = document.getElementById("myImage");
img.onclick = imageClicked;
```

HTML DOM Event

- **Tham số của các trình xử lý sự kiện**: cung cấp thông tin về sự kiện gồm
  - ❖ Loại sự kiện (nhấp chuột, nhấn phím, v.v.)
  - ❖ Dữ liệu về vị trí nơi sự kiện đã được kích hoạt (ví dụ: tọa độ chuột)
  - ❖ Giữ một tham chiếu đến người gửi sự kiện

    Ví dụ. nút đã được nhấp
  - ❖ Lưu trữ thông tin về trạng thái của các phím [Alt], [Ctrl] và [Shift]. Một số trình duyệt không gửi đối tượng này, nhưng đặt nó trong **document.event**

Các sự kiện thông dụng

- **Mouse events:**
  - `onclick`, `onmousedown`, `onmouseup`
  - `onmouseover`, `onmouseout`, `onmousemove`
- **Key events:**
  - `onkeypress`, `onkeydown`, `onkeyup`
  - Only for input fields
- **Interface events:**
  - `onblur`, `onfocus`
  - `onscroll`

Các sự kiện thông dụng

- **Form events**
  - onchange – for input fields
  - onsubmit
    - Cho phép hủy gửi biểu mẫu
    - Hữu ích cho việc xác thực biểu mẫu
- **Miscellaneous events**
  - onload, onunload
    - Chỉ được phép cho phần tử <body>
    - Kích hoạt khi tất cả nội dung trên trang đượcloaded / unloaded

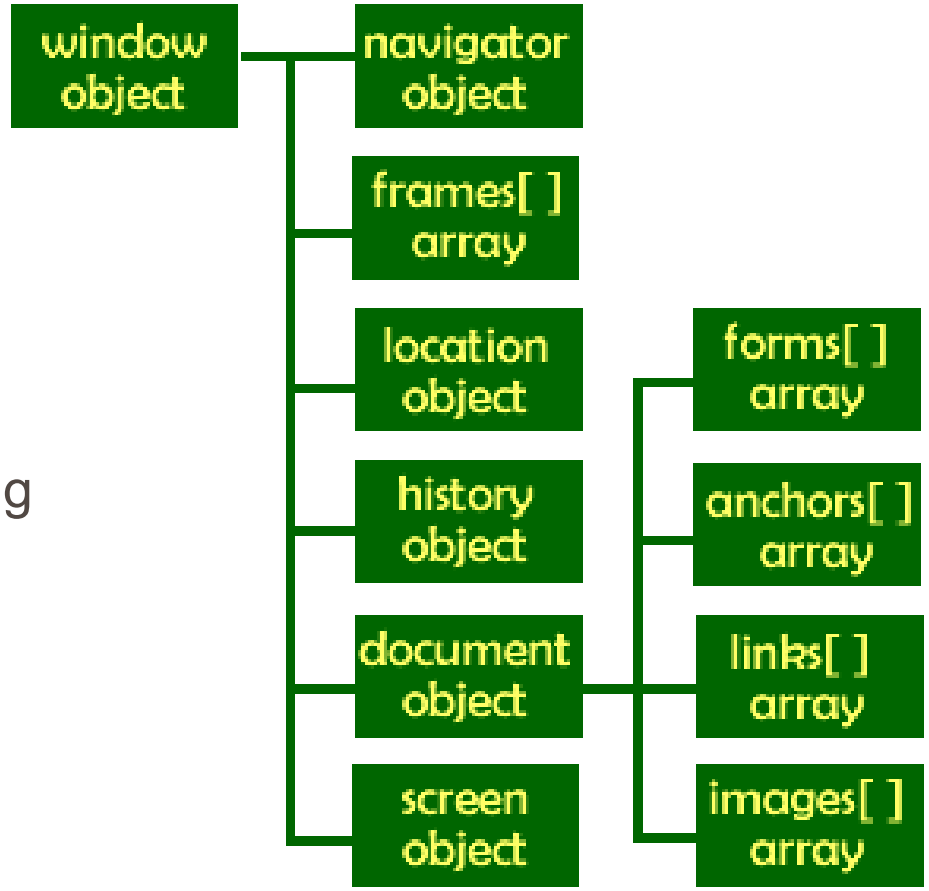
Các sự kiện thông dụng

- Ví dụ:

```
<html>

<head>
  <script type="text/javascript">
    function greet() {
      alert("Loaded.");
    }
  </script>
</head>

<body onload="greet()" >
</body>

</html>
```

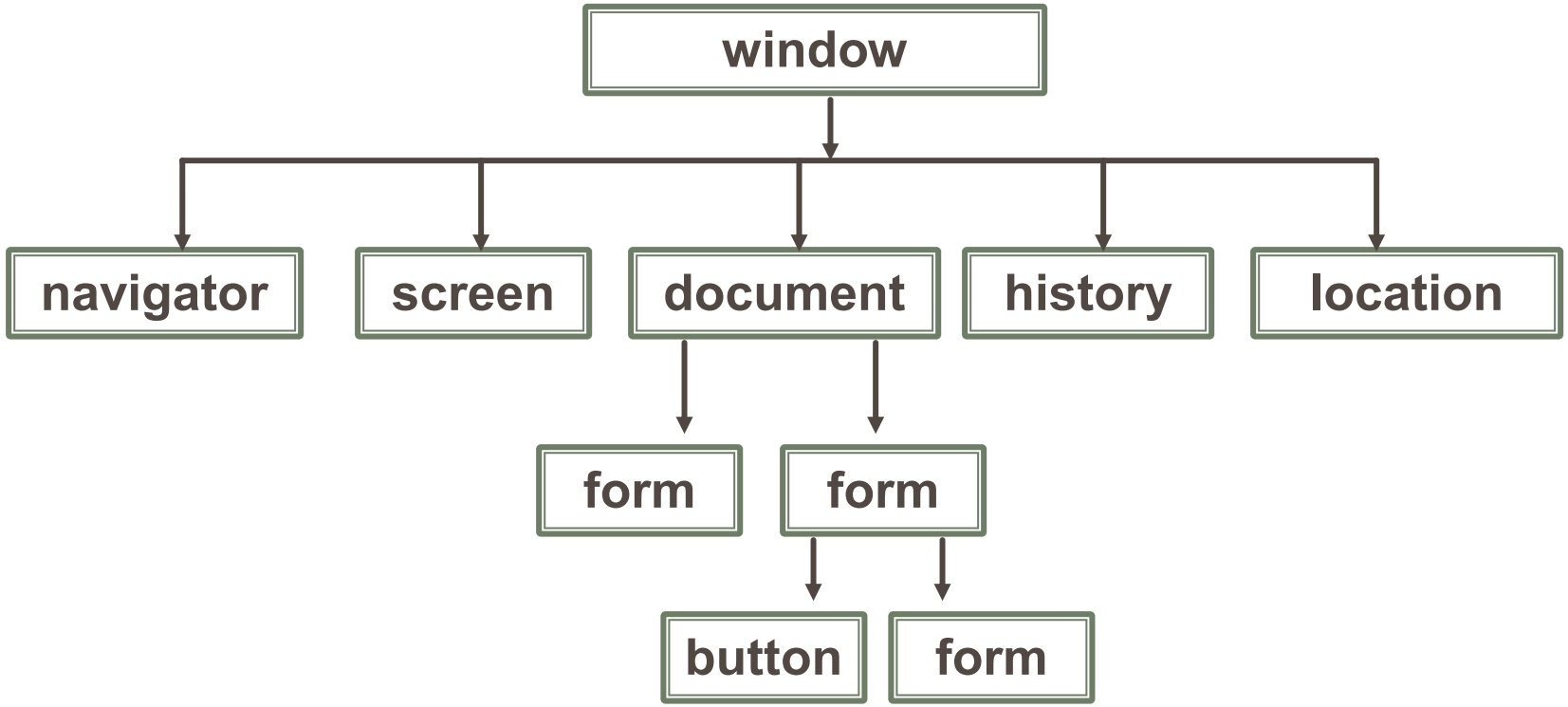
Built-in Browser Objects

- **Trình duyệt cung cấp một số dữ liệu chỉ đọc qua:**
  - **window**
    - Nút trên cùng của DOM
    - Đại diện cho cửa sổ của trình duyệt
  - **document**
    - Chứa thông tin về tài liệu được tải
  - **screen**
    - Chứa các thuộc tính hiển thị của người dùng
  - **browser**
    - Chứa thông tin của trình duyệt



Built-in Browser Objects

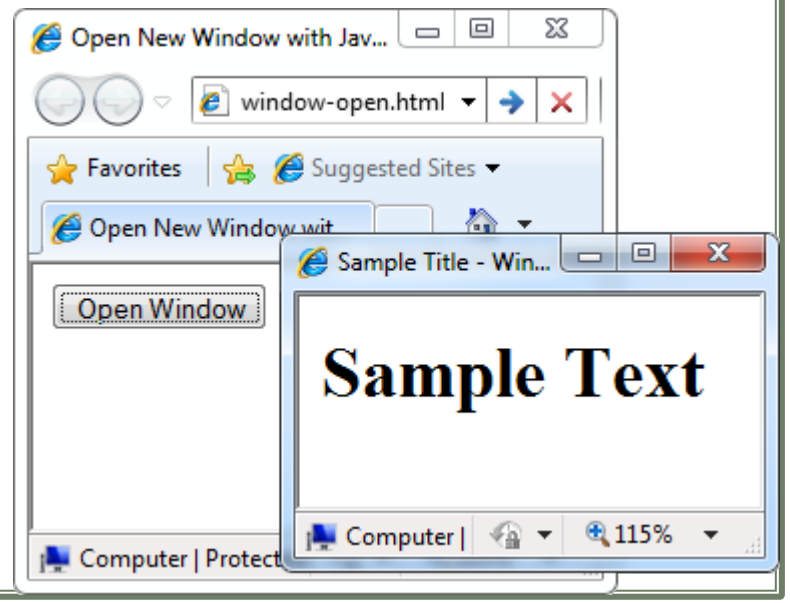
- Ví dụ: **DOM Hierarchy**



Đối tượng trình duyệt tích hợp

- Ví dụ: **Window.open()**

```
var newWindow = window.open("", "sampleWindow",
  "width=300, height=100, menubar=yes, status=yes,
resizable=yes");
newWindow.document.write(
  "<html><head><title>
  Sample Title</title>
  </head><body><h1>Sample
  Text</h1></body>");
newWindow.status =
  "Hello folks";
```

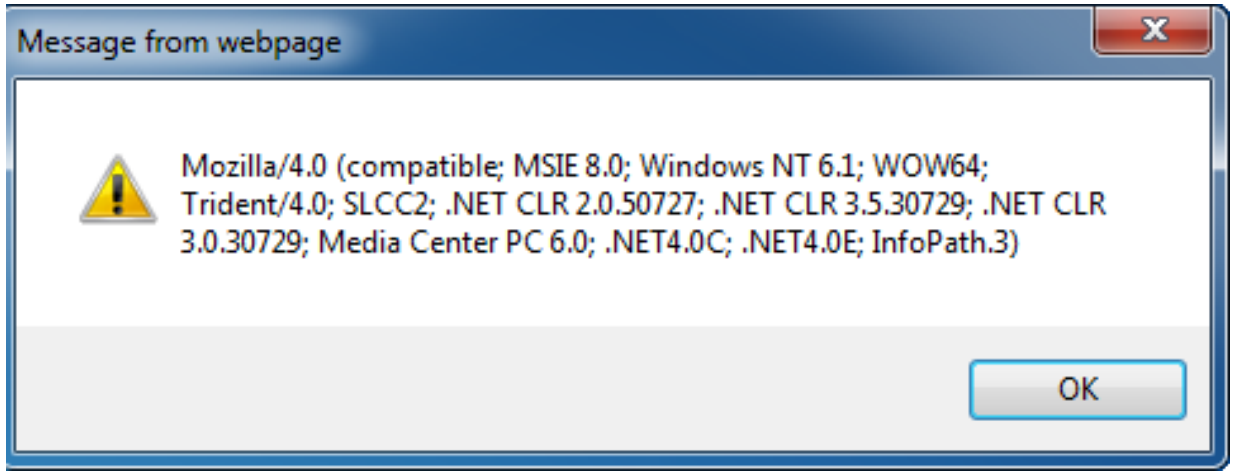


Navigator Object

- **Đối tượng navigator** chứa thông tin về trình duyệt.

```
alert(window.navigator.userAgent);
```

- The browser window
- The navigator in the browser window
- The userAgent (browser ID)



Screen Object

- **Đối tượng screen** chứa thông tin về màn hình
- Ví dụ:

```
window.moveTo(0, 0);
x = screen.availWidth;
y = screen.availHeight;
window.resizeTo(x, y);
```

Document and Location

- **Đối tượng document**
  - Cung cấp một số built-in arrays của các đối tượng cụ thể trên trang Web hiện đang được tải
  - **Ví dụ**:

```
document.links[0].href = "yahoo.com";
document.write("This is some <b>bold text</b>");
```

- **document.location**
  - Truy cập **URL** hiện đang mở hoặc chuyển hướng trình duyệt

```
document.location = "http://www.yahoo.com/";
```

Form Validation

- **Việc xác thực HTML form** được thực hiện bằng JavaScript.
  - Nếu **form field (fname)** trống, hàm sẽ cảnh báo một thông báo và trả về false để ngăn form được **submit**
- **Ví dụ**:

Form Validation

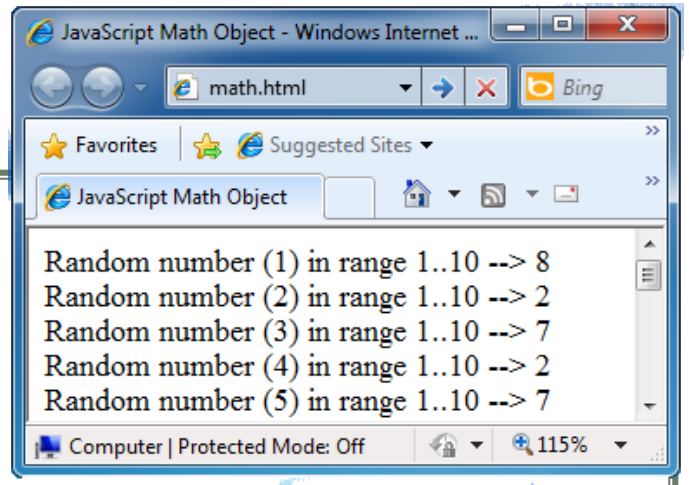
```
function checkForm()
{
  var valid = true;
  if (document.mainForm.firstName.value == "") {
    alert("Please type in your first name!");
    document.getElementById("firstNameError").
      style.display = "inline";
    valid = false;
  }
  return valid;
}
…
<form name="mainForm" onsubmit="return checkForm()">
  <input type="text" name="firstName" />
  …
</form>
```

Math Object

- **Đối tượng Math** cung cấp một số hàm toán học

Ví dụ:

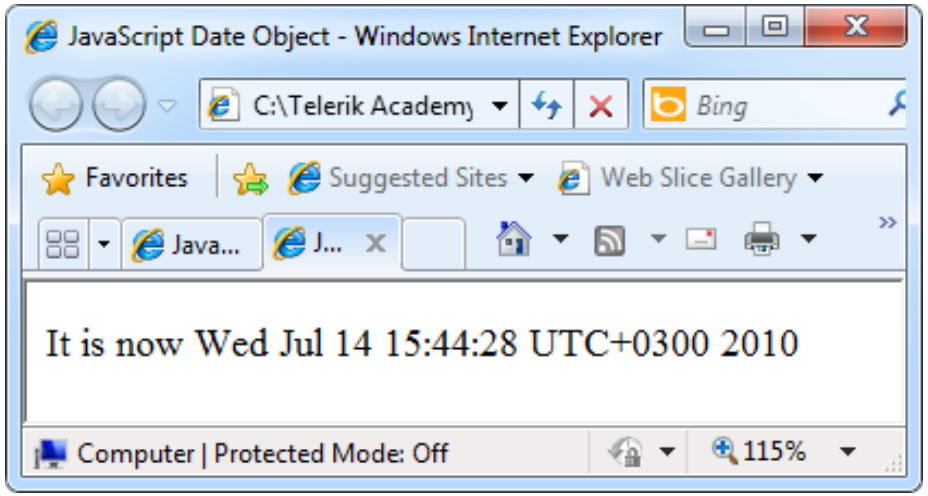
```
for (i=1; i<=20; i++) {
  var x = Math.random();
  x = 10*x + 1;
  x = Math.floor(x);
  document.write("Random number("+ i+ ") in range " +
    "1..10 --> " + x + "<br/>");
}
```



The Date Object

- **Đối tượng `Date`** cung cấp các hàm date / calendar

```
var now = new Date();
var result = "It is now " + now;
document.getElementById("timeField").innerText = result;
...
<p id="timeField"></p>
```



Timers - Timing Events

- **Đối tượng window** cho phép thực thi mã trong các khoảng thời gian xác định. Những khoảng thời gian này được gọi là **timing events**. Gồm hai phương thức:
- **setTimeout():** Thực thi hàm, sau số mili giây được chỉ định

```
window.setTimeout(function, milliseconds);
```

- ❖ ***function***: hàm được thực thi.
- ❖ ***milliseconds:*** số mili giây trước khi thực thi.

Timers - Timing Events

- **setTimeout():**

  Ví dụ:

```
var timer = setTimeout('bang()', 5000);
```

5 seconds after this statement executes, this function is called

```
clearTimeout(timer);
```

Cancels the timer

Timers - Timing Events

- **setInterval():** Tương tự như **setTimeout ()**, nhưng lặp lại việc thực thi hàm liên tục.
- Ví dụ:

```
var timer = setInterval('clock()', 1000);
```

This function is called continuously per 1 second.

```
clearInterval(timer);
```

Stop the timer.

Timers - Timing Events

- Ví dụ:

```
<script type="text/javascript">
  function timerFunc() {
    var now = new Date();
    var hour = now.getHours();
    var min = now.getMinutes();
    var sec = now.getSeconds();
    document.getElementById("clock").value =
      "" + hour + ":" + min + ":" + sec;
  }

  setInterval('timerFunc()', 1000);
</script>

<input type="text" id="clock" />
```

Giới thiệu jQuery

- **Jquery** là một **thư viện** được viết từ **Javascript** giúp xây dựng các chức năng bằng Javascript dễ dàng, nhanh và giàu tính năng hơn.

Giới thiệu jQuery

- **Jquery** có thể
  - ❖ Giúp định dạng lại và thay đổi thông tin trong trang web.
  - ❖ Tạo ra nhiều hiệu ứng trong trang như: các hiệu ứng tương tác user, các hiệu ứng đồ họav.v..
  - ❖ Tạo ra các plugin mới.
  - ❖ Jquery cũng được bootstrap sử dụng để chạy các code js của nó.
- Microsoft đã sử dụng jQuery trong Visual Studio
  - ❖ Sử dụng trong Microsoft's ASP.NET AJAX Framework and ASP.NET MVC Framework

Giới thiệu jQuery

- **Các module JQuery phổ biến**:
  - **Ajax:** xử lý Ajax
  - **Atributes:** Xử lý các thuộc tính của đối tượng HTML
  - **Effect:** xử lý hiệu ứng
  - **Event:** xử lý sự kiện
  - **Form:** xử lý sự kiện liên quan tới form
  - **DOM:** xử lý Data Object Model
  - **Selector:** xử lý luồng lách giữa các đối tượng HTML

Giới thiệu jQuery

- **Các tính năng quan trọng của jQuery**
  - **Gọn nhẹ**: jQuery là một thư viện khá gọn nhẹ, khoảng 19KB (gzipped).
  - **Tương thích đa nền tảng**: Nó tự động sửa lỗi và chạy được trên các trình duyệt phổ biến như Chrome, Firefox, Safari, MS Edge, IE, Android và iOS.
  - **Dễ dàng tạo Ajax**: Nhờ thư viện jQuery, code được viết bởi Ajax có thể dễ dàng tương tác với server và cập nhật nội dung tự động mà không cần phải tải lại trang.

Giới thiệu jQuery

- **Các tính năng quan trọng của jQuery**
  - **Xử lý nhanh thao tác DOM**: jQuery giúp lựa chọn các phần tử DOM một cách dễ dàng, và chỉnh sửa nội dung của chúng bằng cách sử dụng Selector mã nguồn mở gọi là Sizzle.
  - **Đơn giản hóa việc tạo hiệu ứng**: Giống với code snippet có hiệu ứng animation
  - **Xử lý sự kiện:** jQuery xử lý các sự kiện đa dạng

Cài đặt - tích hợp jQuery vào Website

- **Cách 1**:
  - Tải file **js** tại http://www.jquery.com, chọn bản đã nén **.min.js**,
  - Tích hợp vào trang bằng thẻ <script>.
  - **Cú pháp**: <script src=**jquery-3.6.0.min.js** "></script>

Cài đặt - tích hợp jQuery vào Website

- **Cách 2**: dùng CDN (Content Delivery Network) đưa thư viện jQuery vào trong code HTML trực tiếp từ CDN
  - Google CDN
  - Microsoft CDN
  - CDNJS CDN
  - jsDelivr CDN
- Tích hợp vào trang bằng thẻ <script>

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery:** để chọn các phần tử khác nhau trong trang HTML theo **id, tên thẻ, class**
- **Cú pháp**:

```
$(selector)
```

- Ví dụ:

```
//by tag
$("div") //document.querySelectorAll("div");

//by class
$(".menu-item")
//document.querySelectorAll(".menu-item");

//by id
$("#navigation")

//by combination of selectors
$("ul.menu li")
```

Selectors - Chọn phần tử trong jQuery

- **Selector cơ bản**

Selector	Ví dụ	Diễn giải
*	$("*")	Chọn tất cả các phần tử
#id	$("#lastname")	Chọn phần tử có id="lastname"
.class	$(".intro")	Chọn phần tử có class="intro"
.class,.class	$(".intro,.demo")	chọn phần tử có class là "intro" hoặc "demo"
element	$("p")	Chọn các phần tử thẻ <p>
el1,el2,el3	$("h1,div,p")	Chọn tất cả các phần tử thẻ <h1> và<p>

Selectors - Chọn phần tử trong jQuery

- **Selector nâng cao**

Selector	Ví dụ	Diễn giải
:first	$("p:first")	Chọn phần tử <p> đầu tiên trong luồng HTML
:last	$("p:last")	Chọn phần tử <p> cuối cùng
:even	$("tr:even")	Chọn các phần tử <tr> ở vị trí chẵn
:odd	$("tr:odd")	Chọn các phần tử <tr> ở vị trí lẻ
:first-child	$("p:first-child")	Chọn tất cả phần tử <p> là phần tử con đầu tiên trong phần tử cha chứa nó
:first-of-type	$("p:first-of-type")	Chọn các phần tử <p> là phần tử <p> đầu tiên trong các phần tử con mà phần tử cha chứa

Selectors - Chọn phần tử trong jQuery

- **Selector nâng cao**

Selector	Ví dụ	Diễn giải
:last-child	$("p:last-child")	Chọn các phần tử <p> là phần tử cuối cùng trong phần tử cha chứa nó.
:last-of-type	$("p:last-of-type")	Chọn các phần tử <p> là phần tử <p> sau cùng thấy trong phần tử cha
:nth-child(n)	$("p:nth-child(2)")	Tất cả các phần tử <p> là con thứ 2
:nth-last-child(n)	$("p:nth-last-child(2)")	Tất cả phần tử <p> là con thứ 2 đếm từ dưới lên.
:nth-of-type(n)	$("p:nth-of-type(2)")	Tất cả phần <p> là phần tử thứ 2 dạng <p> trong các phần tử con

Selectors - Chọn phần tử trong jQuery

- **Selector nâng cao**

Selector	Ví dụ	Diễn giải
:nth-last-of-type(n)	$("p:nth-last-of-type(2)")	Tất cả các phần tử <p> thứ 2 đếm từ dưới lên.
parent < child	$("div > p")	Tất cả phần tử <p> là con trực tiếp của phần tử
parent descendant	$("div p")	Tất cả phần tử <p> là con, cháu ... của
element + next	$("div + p")	Chọn phần tử <p> là phần tử tiếp theo của phần tử
element ~ siblings	$("div ~ p")	Các phần tử <p> có cấp ngang hàng với một phần tử

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery chọn phần tử danh sách**

Selector	Ví dụ	Diễn giải
:eq(index)	$("ul li:eq(3)")	Phần tử thứ 4 trong một danh sách
:gt(no)	$("ul li:gt(3)")	Các phần tử có chỉ số lớn hơn 3
:lt(no)	$("ul li:lt(3)")	Các phần tử trong danh sách có chỉ số nhỏ hơn 3
:not(selector)	$("input:not(:empty)")	Các phần tử <input> không rỗng

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery theo trạng thái**

Selector	Ví dụ	Diễn giải
:header	$(":header")	Tất cả các phần tử <h1>, <h2> ...
:animated	$(":animated")	Các phần tử động
:focus	$(":focus")	Phần tử đang giữ focus
:contains(text)	$(":contains('Hello')")	Các phần tử có chứa chữ "Hello"
:has(selector)	$("div:has(p)")	Các phần tử trong nó có chứa một phần tử <p>

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery theo trạng thái**

Selector	Ví dụ	Diễn giải
:empty	$(":empty")	Tất cả các phần tử rỗng
:parent	$(":parent")	Các phần tử là cha của 1 phần tử khác
:hidden	$("p:hidden")	Tất cả các phần tử <p> đang ẩn
:visible	$("table:visible")	Tất cả các <table> đang hiện thị

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery theo thuộc tính phần tử**

Selector	Ví dụ	Diễn giải
[attribute]	$("[href]")	Các phần tử có thuộc tính href
[attribute=value]	$("[href='default.htm']")	Các phần tử có thuộc tính href và giá trị là "default.htm"
[attribute!=value]	$("[href!='default.htm']")	Các phần tử có thuộc tính href với giá trị khác "default.htm"
[attribute$=value]	$("[href$='.jpg']")	Các phần tử có thuộc tính href với giá trị là file ".jpg"
[attribute*=value]	$("[title*='hello']")	Các phần tử có thuộc tính title và giá trị chứa "hello"

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery theo thuộc tính phần tử**

Selector	Ví dụ	Diễn giải
[attribute|=value]	$("[title|='Tomorrow']")	Các phần tử có title bằng 'Tomorrow' hoặc bắt đầu bởi 'Tomorrow'
[attribute^=value]	$("[title^='Tom']")	Các phần tử có title với giá trị bắt đầu bằng "Tom"
[attribute~=value]	$("[title~='hello']")	Các phần tử có title, và giá trị có chứa "hello"
[attribute*=value]	$("[title*='hello']")	Các phần tử có title và giá trị chứa "hello"

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery trong FORM**

Selector	Ví dụ	Diễn giải
:input	$(":input")	Tất cả các phần tử input
:text	$(":text")	Tất cả các phần tử có type="text"
:password	$(":password")	Tất cả phần tử có type="password"
:radio	$(":radio")	Tất cả phần tử có type="radio"
:checkbox	$(":checkbox")	Tất cả phần tử có type="checkbox"
:submit	$(":submit")	Tất cả phần tử có type="submit"
:reset	$(":reset")	Tất cả phần tử cótype="reset"

Selectors - Chọn phần tử trong jQuery

- **Selector jQuery trong FORM**

Selector	Ví dụ	Diễn giải
:button	$(":button")	Tất cả phần tử có type="button"
:image	$(":image")	Tất cả phần tử có type="image"
:file	$(":file")	Tất cả phần tử có type="file"
:enabled	$(":enabled")	Tất cả các phần tử input là enable
:disabled	$(":disabled")	Các phần tử input bị vô hiệu
:selected	$(":selected")	Các phần tử input là selected
:checked	$(":checked")	Các phần tử input là checked

jQuery Syntax

- **Cú pháp jQuery:** được thiết kế để chọn các phần tử HTML và thực hiện một số hành động trên các phần tử.

- **Cú pháp**: **$(selector).action()**

  Ví dụ:

  - $(this).hide() *// hides current element*.
  - $("p").hide() *// hides all paragraphs*.
  - $("p.test").hide() *// hides all paragraphs with class="test".*
  - $("#test").hide() *// hides the element with id="test"*

jQuery Syntax

- **$(document).ready()**
  - Các câu lệnh **jQuery** bắt đầu với sự kiện **document.ready.** Code bên trong **$ (document) .ready ()** sẽ chỉ thực thi khi trang DOM đã sẵn sàng.

- **Cú pháp**:

```
<script type="text/javascript">
   $(document).ready(function()
   {
      // your code
      $(selector).action();
   });
</script>
```

jQuery Syntax

- **$(document).ready()**
  - **$(document).ready(function(){}):** đại diện cho sự kiện sẵn sàng của tài liệu và thực thi mã **jQuery** sau khi tài liệu được tải đầy đủ và sẵn sàng.

Ví dụ:

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script>
        $(document).ready(function(){
            $("p").text("Hello World!");
        });
    </script>
```

DOM Traversal

- **jQuery traversing:** là "move through - di chuyển qua" được sử dụng để "tìm" hoặc chọn các phần tử HTML dựa trên mối quan hệ của chúng với các phần tử khác.
- Các phương thức **DOM traversal** giúp duyệt xung quanh cây DOM rất dễ dàng.
- Để duyệt qua DOM cần xác định mối quan hệ giữa các phần tử trong cây DOM.

DOM Traversal

- **Di chuyển trong DOM với jQuery**: Sử dụng các hàm jQuery như **parent(), children(), next(), pre(), find()** để lựa chọn phần tử trong DOM dựa trên phần tử đang chọn

Hàm	Giải thích
.parent()	lấy phần tử cha trực tiếp của phần tử.
.parents()	lấy phần tử các phần tử cha (kể cả không trực tiếp)
.children()	lấy các phần tử con
.siblings()	các phần tử ngang hàng (anh em)
.next()	phần tử ngang hàng tiếp theo
.nextAll()	tất cả các phần tử ngang hàng tiếp theo

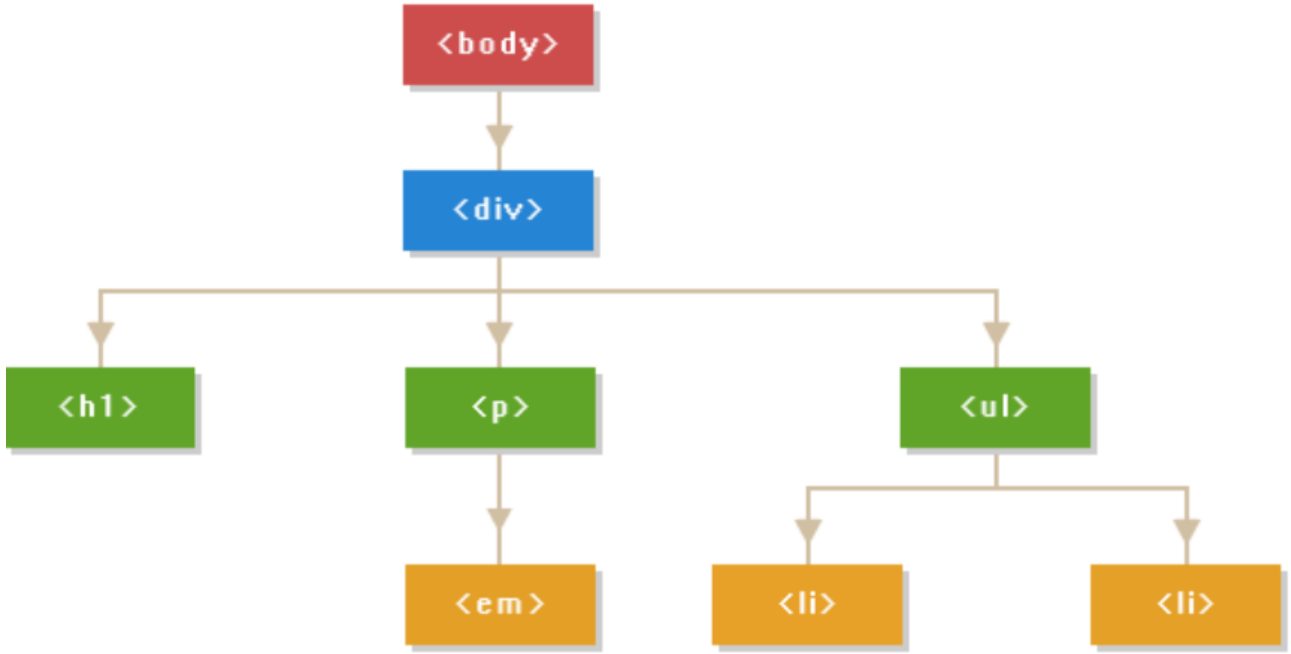
DOM Traversal

- **Di chuyển trong DOM với jQuery**:

Hàm	Giải thích
.prev()	phần tử ngang hàng trước
.prevAll()	tất cả các phần tử ngang hàng phía trước
.eq(index)	phần tử có thứ tự index trong tập hợp chọn được
.find()	tìm phần tử trong các phần tử con, cháu ...
.each()	Lặp lại các phần tử được chỉ định và thực hiện hàm gọi lại cho từng phần tử.
.first()	Lấy phần tử được chỉ định xuất hiện đầu tiên
.next()	Lấy phần tử ngay sau của phần tử được chỉ định

DOM Traversal

- **Di chuyển trong DOM với jQuery**:
- Ví dụ: cây DOM



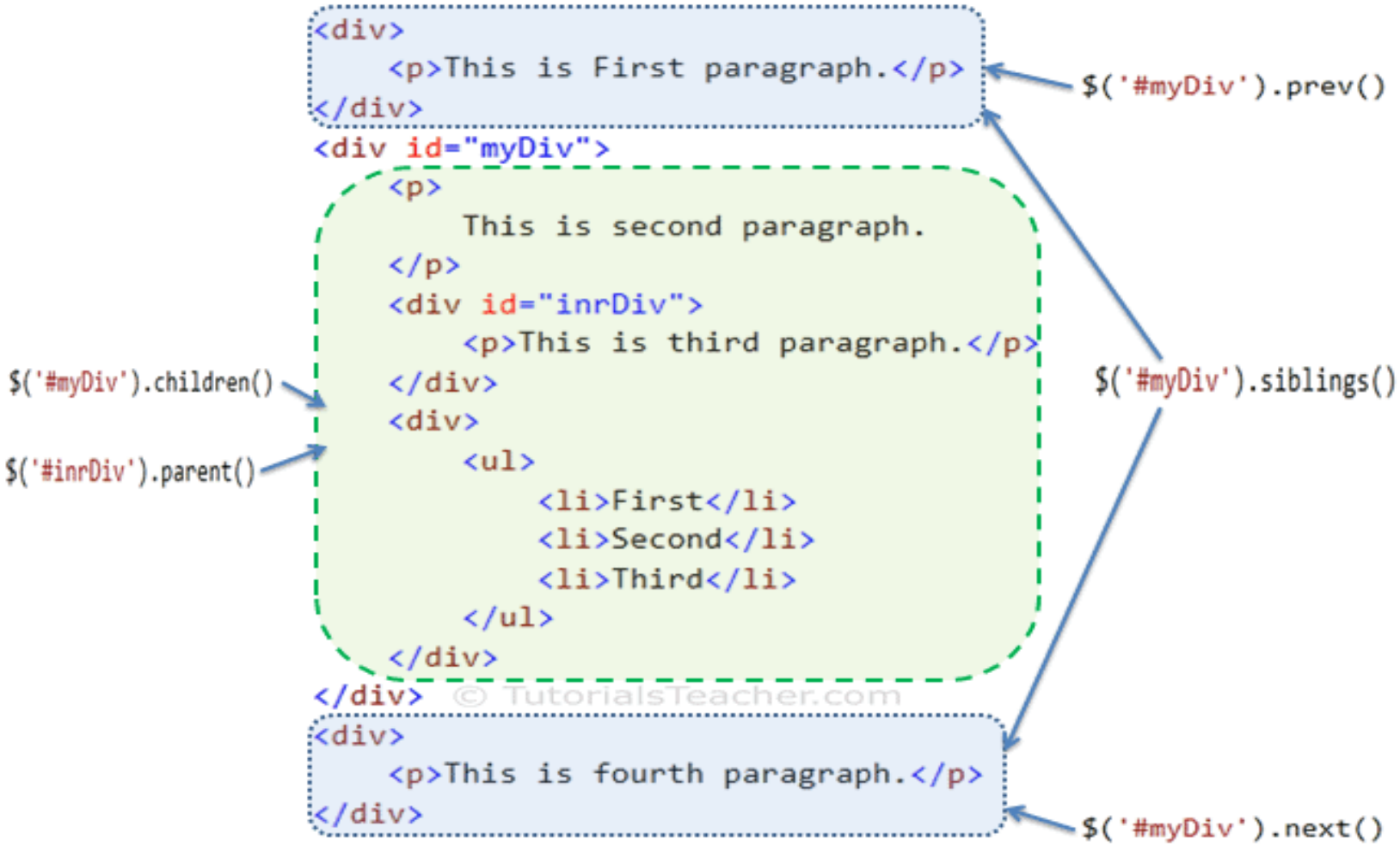
jQuery Traversing - Ancestors

- **Duyệt phần tử parent, grandparent, great-grandparent**
- **Phương thức:**
  - parent()
  - parents()
  - parentsUntil()

jQuery Traversing - Ancestors

- **jQuery parent()**

Ví dụ:



jQuery Traversing - Ancestors

- **jQuery parent()**

Ví dụ:

```
<style>
    .highlight{
        background: yellow;
    }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
      $(document).ready(function(){
            $("li").parent().addClass("highlight");
      });
</script>
```

jQuery Traversing - Ancestors

- **jQuery parent()**

  Ví dụ:

```
<body>
    <div class="container">
        <h1>Hello World</h1>
        <p>This is a <em>simple paragraph</em>.</p>
        <ul>
            <li>Item One</li>
            <li>Item Two</li>
        </ul>
    </div>
</body>
```

This is a *simple paragraph*.

- Item One
- Item Two

jQuery Traversing - Ancestors

- **jQuery parents():** lấy các phần tử cha (**ancestors)** của phần tử đã chọn

Ví dụ:

```
<style>
  *      { margin: 10px;}
  .frame{ border: 2px solid green;}
</style>
```

```
<script>
   $(document).ready(function(){
      $("li").parents().addClass("frame");});
</script>
```

jQuery Traversing - Ancestors

- **jQuery parents():** lấy các phần tử cha (**ancestors)** của phần tử đã chọn

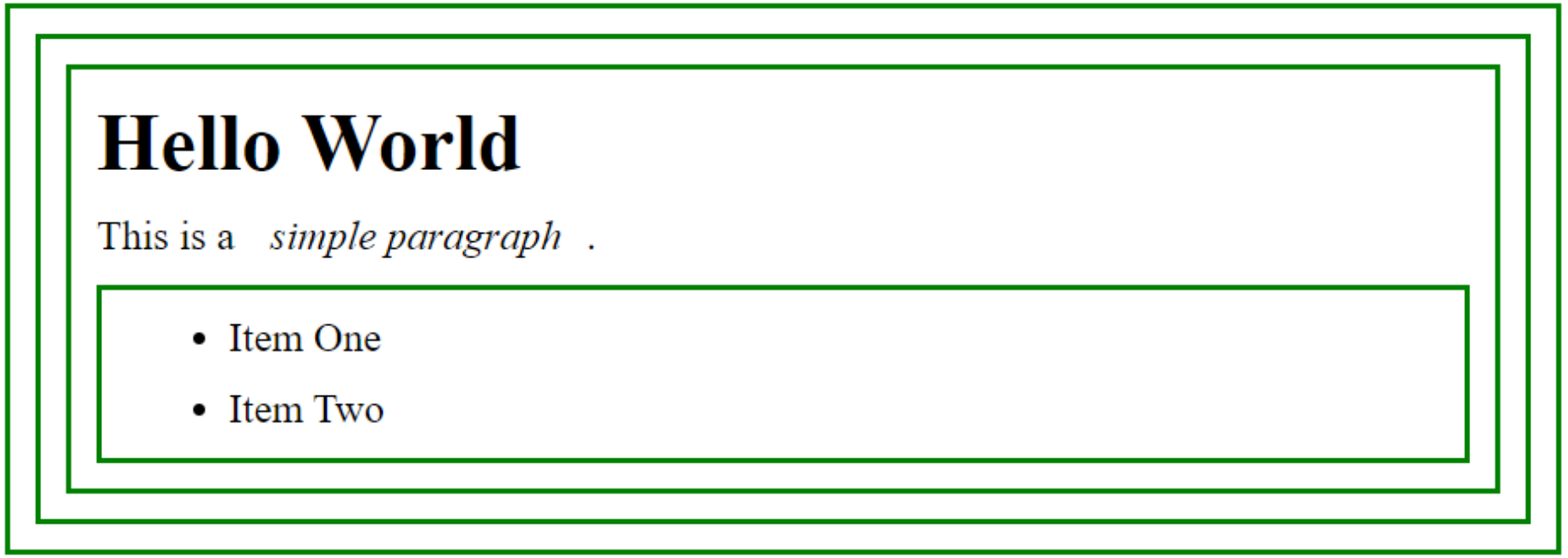
Ví dụ:

```
<div class="container">
   <h1>Hello World</h1>
   <p>This is a<em>simple paragraph</em>.</p>
   <ul>
      <li>Item One</li>
      <li>Item Two</li>
   </ul>
</div>
```

jQuery Traversing - Ancestors

- **jQuery parents():** lấy các phần tử cha (**ancestors)** của phần tử đã chọn

  Ví dụ:



jQuery Traversing - Ancestors

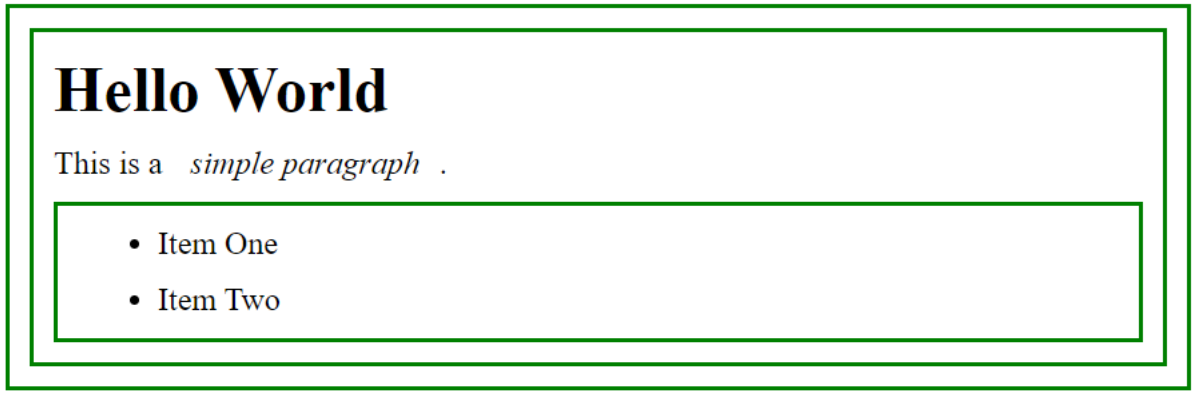
- **jQuery parentsUntil():** lấy tất cả các phần tử **ancestors** giữa hai phần tử đã cho trong một hệ thống phân cấp DOM.
- Ví dụ:

```
<script>
    $(document).ready(function(){
        $("li").parentsUntil("html").addClass("frame");
    });
</script>
```

jQuery Traversing - Ancestors

- **jQuery parentsUntil():** lấy tất cả các phần tử **ancestors** giữa hai phần tử đã cho trong một hệ thống phân cấp DOM.
- Ví dụ:

```
<script>
    $(document).ready(function(){
      $("li").parentsUntil("html").addClass("frame");
     });
</script>
```



jQuery Traversing - Descendants

- **jQuery Descendants**: duyệt qua DOM để tìm con, cháu (child, grandchild, great-grandchild) của một phần tử được chọn.
- **Các phương thức**:
  - children()
  - find()

jQuery Traversing - Descendants

- **jQuery children():** lấy các phần tử con trực tiếp của phần tử được chọn.

  Ví dụ: highlight các phần tử con trực tiếp của <ul> là <li> bằng cách thêm lớp **.highlight** trên tài liệu.

```
<style>
      .highlight{ background: yellow; }
</style>
```

```
<script>
   $(document).ready(function(){
      $("ul").children().addClass("highlight"); });
</script>
```

jQuery Traversing - Descendants

- **jQuery children():**

Ví dụ:

```
<div class="container">
  <h1>Hello World</h1>
    <p>This is a <em>simple paragraph</em>.</p>
      <ul>
            <li>Item One</li>
            <li>Item Two</li>
      </ul>
</div>
```

This is a *simple paragraph*.

- Item One
- Item Two

jQuery Traversing - Descendants

- **jQuery find():** lấy các phần tử con của phần tử được chọn.
- Phương thức **find ()** và **children ()** tương tự nhau, nhưng
  - **find ()** tìm kiếm qua nhiều cấp của DOM đến con cuối cùng
  - **children ()** tìm kiếm ở một cấp duy nhất trên DOM

Ví dụ:

```
<script>
  $(document).ready(function(){
     $("div").find("li").addClass("frame");});
</script>
```

```
<style>
      *{ margin: 10px; }
      .frame{ border: 2px solid green; }
</style>
```

jQuery Traversing - Descendants

- **jQuery find():**

Ví dụ:

```
<body>
    <div class="container">
        <h1>Hello World</h1>
        <p>This is a <em>simple paragraph</em>.</p>
        <ul>
                <li>Item One</li>
                <li>Item Two</li>
        </ul>
    </div>
</body>
```

jQuery Traversing - Descendants

- **jQuery find():**

Ví dụ:

```
<body>
    <div class="container">
        <h1>Hello World</h1>
        <p>This is a <em>simple paragraph</em>.</p>
        <ul>
            <li>Item One</li>
            <li>Item Two</li>
        </ul>
    </div>
</body>
```

ASP.NET RAZOR FOR C#

ASP.NET Razor

- **Razor** là một cú pháp đánh dấu cho phép nhúng server-based code vào các trang web bằng C # và VB.Net
- ASP.NET MVC đã triển khai một **view engine** cho phép sử dụng **Razor** bên trong ứng dụng MVC để tạo HTML.
- **Cú pháp Razor View Engine**

```
@{
      //Razor block
 }
```

ASP.NET Razor

- **Cú pháp Razor trong C #**
  - Các khối mã **Razor** được đặt trong **@ {...}**
  - Biểu thức **inline (biến và hàm)** bắt đầu bằng **@**
  - Câu lệnh mã kết thúc bằng dấu chấm **phẩy**
  - Các biến được khai báo với từ khóa **var**
  - Các chuỗi được đặt trong dấu ngoặc kép
  - Mã C # phân biệt chữ hoa chữ thường
  - Các tệp C # có phần mở rộng là **.cshtml**

ASP.NET Razor

- **Inline expression**
  - ❖ Bắt đầu với ký hiệu @

Ví dụ:

```
<h1>Razor syntax demo</h1>
<h2>@DateTime.Now.ToShortDateString()</h2>
```

Output:

Razor syntax demo

08-09-2014

ASP.NET Razor

- **Multi-statement Code block**:
  - ❖ Khối lệnh được đặt trong @{ ... }, mỗi dòng được kế thúc bằng dấu chấm phẩy

```
@{
      var date = DateTime.Now.ToShortDateString();
      var message = "Hello World";
}
<h2>Today's date is: @date </h2>
<h3>@message</h3>
```

Output:

```
Today's date is: 08-09-2014
Hello World!
```

ASP.NET Razor

- **Display Text from Code Block**
  - ❖ Sử dụng **@:** hoặc **<text>/<text>** để hiển thị văn bản bên trong khối lệnh

Ví dụ: sử dụng @

```
@{
      var date = DateTime.Now.ToShortDateString();
      string message = "Hello World!";
      @:Today's date is: @date <br />
      @message
}
```

ASP.NET Razor

- **Display Text from Code Block**
  - Sử dụng **@:** hoặc **<text>/<text>** để hiển thị văn bản bên trong khối lệnh

Ví dụ: sử dụng **<text>/<text>**

```
@{
      var date = DateTime.Now.ToShortDateString();
      string message = "Hello World!";
      <text>Today's date is:</text>
      @date <br />
      @message
}
```

```
Today's date is: 08-09-2014
Hello World!
```

ASP.NET Razor

- **Cấu trúc điều khiển**
  - ❖ if-else condition

```
@if (Điều kiện) { khối lệnh 1}
else { khối lệnh 2}
```

  - ❖ Loop“

```
@for(var i = 10; i < 21; i++)
    {khối lệnh}
```

```
@foreach (var x in list)
    {<li>@x</li>}
```

ASP.NET Razor

- **Model**: dùng **@model** để sử dụng đối tượng **model** trong **view**.
- Ví dụ:

```
@model Student
<h2>Student Detail:</h2>
<ul>
      <li>Student Id: @Model.StudentId</li>
      <li>Student Name: @Model.StudentName</li>
      <li>Age: @Model.Age</li>
</ul>
```

Razor Page ASP.NET

- **Tạo một project Razor Web ASP.NET MVC**
  - Tạo một **Model**
  - Ví dụ:

```
namespace Razor.Models
{
    public class Product
    {
        public int ProductID { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public string Category { set; get; }
    }
}
```

Razor Page ASP.NET

- **Tạo một project Razor Web ASP.NET MVC**
  - ❖ Tạo một Controller: khai báo using Razor.Models;

```
public class HomeController : Controller {
    Product myProduct = new Product {
            ProductID = 1,
            Name = "Kayak",
            Description = "A boat for one person",
            Category = "Watersports",
            Price = 275M
        };
}
 public ActionResult Index() {
     return View(myProduct);
 }
```

Razor Page ASP.NET

- **Tạo một project Razor Web ASP.NET MVC**
  - Tạo một **View**: **Index.cshtml**

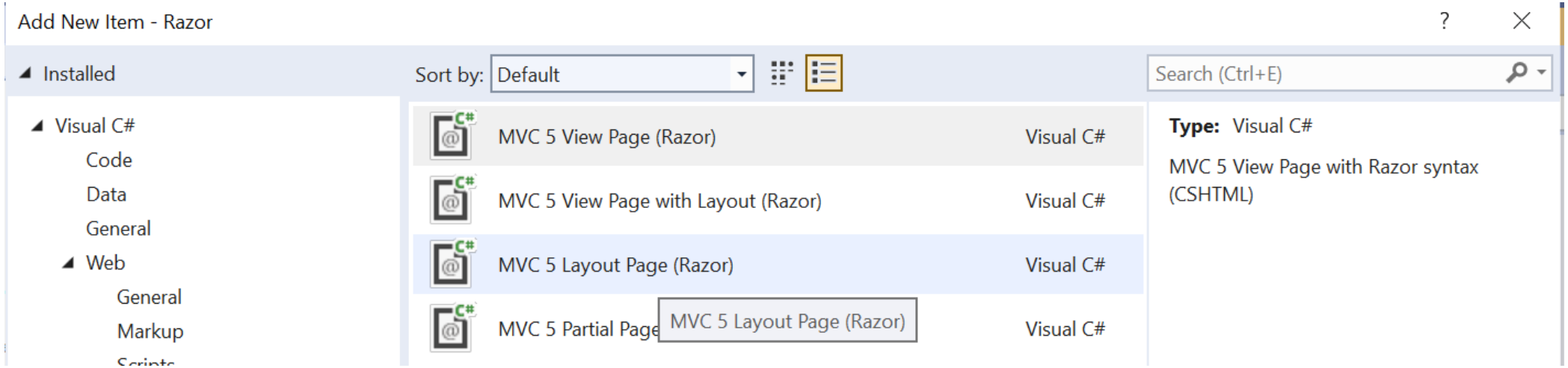
```
@model Razor.Models.Product
@{
        Layout = null;
}
```

  - **@model** khai báo loại mô hình đối tượng sẽ chuyển tới view từ phương thức **action**.

```
<body>
    <div>
        @Model.Name
    </div>
</body>
```

Razor Page ASP.NET

- **Tạo một project Razor Web ASP.NET MVC**
  - ❖ **Tạo một Layout**: right-click trên thư mục **Views** chọn **Add ➤ New Item** -> chọn **MVC 5 Layout Page (Razor)**
  - ❖ Nhập tên Layout: **_BasicLayout.cshtml**



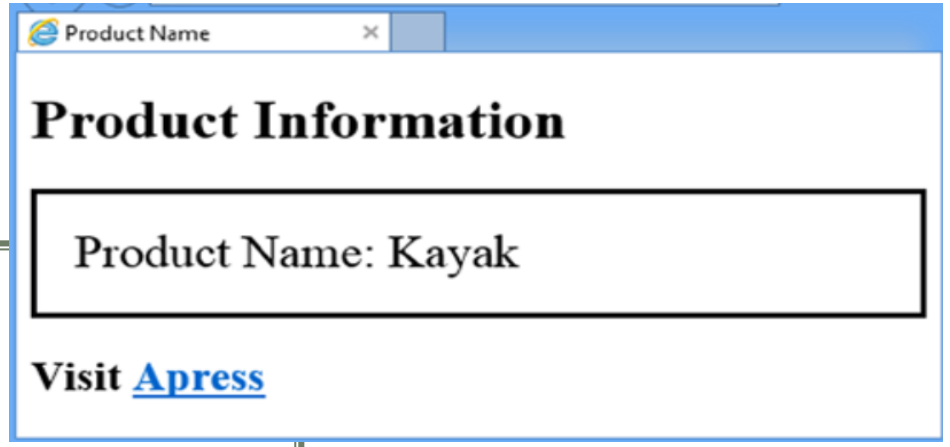
Razor Page ASP.NET

- **Tạo một project Razor Web ASP.NET MVC**
  - ❖ **Tạo một Layout:** hiệu chỉnh nội dung của **_BasicLayout.cshtml**

```
<body>
    <h1>Product Information</h1>
    <div style="border: solid medium black; font-size:20pt">
    @RenderBody()
</body>
```

Razor Page ASP.NET

- **Tạo một project Razor Web ASP.NET MVC**
  - ❖ **Áp dụng Layout _BasicLayout.cshtml vào view**
    - **Mở lại tập tin index()**




```
@ model Razor.Models.Product
@{
    ViewBag.Title = "Product name";
    Layout = "~/Views/_BasicLayout.cshtml";
}
Product Name: @Model.Name
```