Contents

[CHAPTER I. HTML 4](#_Toc66693405)

[**1.** **HTML là gì? Bố cục HTML của một trang web** 4](#_Toc66693406)

[**2.** **HTML Elements và Attributes** 7](#_Toc66693407)

[**3.** **Các thẻ HTML thông dụng** 10](#_Toc66693408)

[a. Nhóm thẻ Block: 10](#_Toc66693409)

[b. Nhóm thẻ inline 14](#_Toc66693410)

[**4.** **Các thẻ HTML định dạng Table** 16](#_Toc66693411)

[**5.** **Form trong HTML** 19](#_Toc66693412)

[**6.** **Bài tập** 27](#_Toc66693413)

[CHAPTER II. CSS 29](#_Toc66693414)

[**1.** **Giới thiệu về CSS** 29](#_Toc66693415)

[**2.** **Cú pháp và Selector CSS** 29](#_Toc66693416)

[**3.** **Cách nhúng CSS vào HTML** 33](#_Toc66693417)

[**4.** **Một số thuộc tính CSS thường dùng** 35](#_Toc66693418)

[**5.** **CSS Table – Bảng trong CSS** 36](#_Toc66693419)

[**6.** **CSS Form – Form trong CSS** 42](#_Toc66693420)

[**7.** **CSS Layout – Bố cục website trong CSS** 43](#_Toc66693421)

[**8.** **Bài tập** 46](#_Toc66693422)

[CHAPTER III. JavaScript 48](#_Toc66693423)

[**1.** **JavaScript là gì? Viết JavaScript ở đâu?** 48](#_Toc66693424)

[**2.** **JavaScript căn bản** 49](#_Toc66693425)

[a. Biến trong JavaScript 49](#_Toc66693426)

[b. Toán tử trong JavaScript: 51](#_Toc66693427)

[c. Lệnh kiểm tra điều kiện if… else 54](#_Toc66693428)

[d. Lệnh kiểm tra điều kiện switch… case 55](#_Toc66693429)

[e. Vòng lặp for 56](#_Toc66693430)

[f. Vòng lặp while; do… while 57](#_Toc66693431)

[g. Điều khiển vòng lặp (lệnh break; continue;) 58](#_Toc66693432)

[h. Function – hàm trong JavaScript 59](#_Toc66693433)

[i. Biến toàn cục và biến cục bộ 60](#_Toc66693434)

[**3.** **Kiểu dữ liệu và phương thức xử lý dữ liệu** 61](#_Toc66693435)

[a. JS Object – Đối tượng trong JavaScript 61](#_Toc66693436)

[b. JS String – Chuỗi trong JavaScript 63](#_Toc66693437)

[c. JS Numbers – Số trong JavaScript 67](#_Toc66693438)

[d. JS Arrays – Mảng trong JavaScript 71](#_Toc66693439)

[e. JS Dates – Thời gian trong JavaScript 74](#_Toc66693440)

[f. JS Math – Đối tượng toán học trong JavaScript 77](#_Toc66693441)

[**4.** **JavaScript DOM** 79](#_Toc66693442)

[**5.** **JavaScript BOM** 84](#_Toc66693443)

[**6.** **JavaScript Even** 86](#_Toc66693444)

[**7.** **Regular Expression và Form Validation trong JavaScript** 88](#_Toc66693445)

[a. Regular Expression – Biểu thức chính quy 88](#_Toc66693446)

[b. JavaScript Form Validation 93](#_Toc66693447)

[**8.** **Bài tập** 95](#_Toc66693448)

[CHAPTER IV. GIỚI THIỆU VỀ ASP.NET WEBFORM 98](#_Toc66693449)

[**1.** **Giới thiệu về Ứng dụng Web – WebApp** 98](#_Toc66693450)

[**2.** **Giới thiệu về ASP.NET và ASP.NET WebForm** 98](#_Toc66693451)

[**3.** **Tạo Project ASP.NET và cấu trúc của một project ASP.NET** 99](#_Toc66693452)

[CHAPTER V. Điều khiển Controls 104](#_Toc66693453)

[**1.** **Cấu trúc một trang ASP.NET** 104](#_Toc66693454)

[a. Tạo một trang Asp.Net 104](#_Toc66693455)

[b. Cấu trúc của trang Asp.Net 107](#_Toc66693456)

[**2.** **HTML Server Controls** 108](#_Toc66693457)

[**3.** **Web Server Controls** 110](#_Toc66693458)

[**4.** **Validation Controls** 117](#_Toc66693459)

[**5.** **Web User Control** 119](#_Toc66693460)

[**6.** **Bài tập** 120](#_Toc66693461)

[CHAPTER VI. Master Page 123](#_Toc66693462)

[**1.** **Master Page** 123](#_Toc66693463)

[**2.** **Đối tượng Request và Response** 125](#_Toc66693464)

[**3.** **Đối tượng Cookies** 126](#_Toc66693465)

[**4.** **Đối tượng Session** 130](#_Toc66693466)

[**5.** **Bài tập** 132](#_Toc66693467)

[CHAPTER VII. Mô hình 3 lớp 136](#_Toc66693468)

[**1.** **ADO.NET** 136](#_Toc66693469)

[**2.** **ODP.NET** 137](#_Toc66693470)

[**3.** **Giới thiệu và cấu trúc chung về mô hình 3 lớp** 142](#_Toc66693471)

[**4.** **Cấu trúc mô hình 3 lớp trong các ứng dụng internal của FPTS** 144](#_Toc66693472)

[**5.** **Bài tập** 145](#_Toc66693473)

# CHAPTER I. HTML

1. **HTML là gì? Bố cục HTML của một trang web**

**HTML là gì?**

HTML – HyperTextMarkupLanguage là ngôn ngữ dùng để tạo và cấu trúc các thành phần trong trang web hoặc ứng dụng, phân chia các đoạn văn bản hoặc khai báo các tập tin kỹ thuật số như hình ảnh, video, nhạc, …

HTML không phải là ngôn ngữ lập trình, đồng nghĩa với việc nó không thể tạo ra các chức năng “động” được. Nó chỉ giống như Microsoft Word, dùng để bố cục và định dạng trang web.

**Tạo một tập tin HTML**

Trong Windows, có thể dùng Notepad (Notepad++) để tạo một tập tin HTML, lưu tập tin này thành đuôi .html và sử dụng trình duyệt web (Google Chrome, Mozilla Firefox, Safari, …) để đọc nó.

**Chạy một tập tin HTML**

Để chạy một tập tin HTML đơn giản là chỉ cần click đúp vào tập tin (chọn open) để mở với trình duyệt mặc định hoặc chọn open with để chọn và mở tập tin với trình duyệt khác.

**Cấu trúc một đoạn HTML**

Một tài liệu HTML được hình thành bởi các phần tử HTML (HTML Elements) được quy định bằng các cặp thẻ (tag), các cặp thẻ này được bao bọc bởi một dấu ngoặc nhọn và thường được khai báo thành một cặp, bao gồm thẻ mở và thẻ đóng, cấu trúc như sau:

<tagname></tagname>

Ví dụ đoạn HTML khai báo một đoạn văn bản:



Ngoài ra mỗi thẻ HTML còn có một số thuộc tính riêng (HTML Attributes) và danh sách các thuộc tính sẽ nằm bên trong thẻ mở như sau:

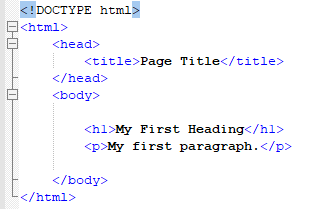
<tagname pro1="value1" prop2="value2"></tagname>

Ví dụ, đưa vào thẻ <p> thuộc tính *align* với giá trị "center" nhằm định dạng đoạn văn căn giữa trang:



**Bố cục HTML của một trang web**

Thông thường bố cục HTML của một website sẽ có dạng như sau:



Trong đó:

* <!DOCTYPE html> là phần khai báo kiểu dữ liệu hiển thị là html để trình duyệt (*Browser*) biết.
* <html> và </html> là cặp thẻ nằm ngoài cùng và nó có nhiệm vụ là bao hết nội dung của trang web lại. Thẻ này là bắt buộc.
* <head> và </head> là phần khai báo thông tin của trang web như tiêu đề trang, tập hợp ký tự, định dạng, liên kết, …
* <title> và </title> nằm bên trong thẻ <head> và dùng để khai báo tiêu đề cho trang web.
* <body> và </body> là thành phần quan trọng nhất, nó chứa những đoạn mã HTML dùng để hiển thị trên website.
* Các thẻ còn lại nằm trong thẻ <body> chính là các thẻ định dạng dữ liệu.

Như vậy trong một tập tin HTML có thể chia làm 2 phần chính:

* Phần 1: Là phần được đặt trong cặp thẻ **<head></head>**, dùng để chứa những mô tả, nội dung hoặc tham chiếu cần thiết cho trang web.
* Phần 2: Là phần được đặt trong cặp thẻ <**body></body>**, dùng để chứa những nội dung mà ta muốn hiển thị trên trang web.

1. **HTML Elements và Attributes**

**Phần tử HTML - HTML Elements**

Một phần tử HTML thường được xác định dựa trên 3 thành phần:

* Thẻ mở
* Nội dung nằm bên trong cặp thẻ (hay còn được gọi là nội dung của phần tử)
* Thẻ đóng

Tuy nhiên, có một số trường hợp đặc biệt mà phần tử chỉ có một thành phần duy nhất là thẻ mở, những phần tử này được gọi là phần tử rỗng.

Ví dụ, một đoạn mã HTML:



 Trong đó:

<p>Tài liệu hướng dẫn học Lập Trình Web</p> là một phần tử, nó có:

* Thẻ mở là: <p>
* Nội dung là: Tài liệu hướng dẫn học Lập Trình Web
* Thẻ đóng là: </p>

<u>Từ cơ bản đến nâng cao</u> là một phần tử, nó có:

* Thẻ mở là: <u>
* Nội dung là: Từ cơ bản đến nâng cao
* Thẻ đóng là: </u>

<br> là một phần tử rỗng

<hr> là một phần tử rỗng

- Lưu ý: Một phần tử thường được gọi dựa theo tên thẻ của nó, ví dụ như bốn phần tử ở phía trên sẽ được gọi lần lượt là: phần tử <p>, phần tử <u>, phần tử <br>, phần tử <hr>

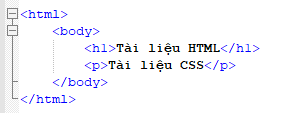
**Phần tử lồng nhau**

Thông thường, một phần tử có thể được lồng vào bên trong "phần nội dung" của một phần tử khác.

Trong đó:

* Phần tử mà nội dung của nó có chứa phần tử khác thì được gọi là phần tử cha.
* Phần tử được lồng vào bên trong nội dung của phần tử khác thì được gọi là phần tử con

Ví dụ:



Dưới đây là bảng mô tả phần tử cha & phần tử con của bốn phần tử trên:

|  |  |  |
| --- | --- | --- |
| **Phần tử HTML** | **Phần tử cha** | **Phần tử con** |
| <html> <body> <h1>Tài liệu HTML</h1> <p>Tài liệu CSS</p> </body> </html> | không có | <body> |
| <body> <h1>Tài liệu HTML</h1> <p>Tài liệu CSS</p> </body> | <html> | <h1> |
| <p> |
| <h1>Tài liệu HTML</h1> | <body> | không có |
| <p>Tài liệu CSS</p> | <body> | không có |

**Một số vấn đề cần lưu ý**

Trong một tập tin HTML, nếu chúng ta soạn thảo mã sai quy tắc thì khi chạy tập tin, các phần tử sẽ hiển thị không được chính xác như mong đợi. Cho nên cần phải hết sức lưu ý đến vấn đề cú pháp để tránh mắc phải những lỗi không đáng có.

* Không quên thẻ đóng: ngoại trừ phần tử rỗng, những phần tử còn lại tuyệt đối không được thiếu thẻ đóng, việc thiếu thẻ đóng sẽ gây ảnh hưởng nghiêm trọng đến các phần tử khác, nó sẽ khiến những phần tử khác bị xem là nội dung của phần tử thiếu thẻ đóng.
* Không được xáo trộn thứ tự giữa các thẻ: việc xáo trộn thứ tự giữa các thẻ đóng cũng gây ảnh hưởng gần giống như việc thiếu thẻ đóng.
* Trình bày mã lệnh: trong HTML, các khoảng trắng, ngắt xuống dòng giữa các thẻ không ảnh hưởng đến nội dung hiển thị. Tuy nhiên nên trình bày mã lệnh theo một tiêu chuẩn chung để việc viết code, sửa lỗi được dễ dàng hơn và tránh các lỗi. Nên trình bày mã lệnh như sau: phần tử con cần phải được ngắt xuống dòng và thụt vào bốn dấu khoảng trắng so với phần tử cha của nó.

**Thuộc tính của thẻ HTML – HTML Attributes**

Mỗi thẻ HTML có thể có một hoặc nhiều thuộc tính đi kèm, nhằm thiết lập thêm thông tin cho nó. Các thuộc tính luôn nằm trong thẻ mở bao gồm tên thuộc tính và giá trị thuộc tính, các thuộc tính cách nhau bởi khoảng trắng, theo cấu trúc name=”value” hoặc name=’value’.

Ví dụ 1: thẻ <input> tạo ra một ô nhập dữ liệu, với các thuộc tính type, class, name, value



Ví dụ 2: Sử dụng thuộc tính style để thêm kiểu cho một phần tử, chẳng hạn như màu sắc, phông chữ, kích thước, …

Cú pháp:

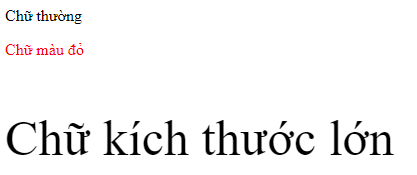
<tagname style="property:value;">

*\* Property là CSS property, value là giá trị CSS. Các phần này sẽ được học trong chapter II về CSS*

Ví dụ:



Kết quả:



*\* Còn có nhiều thuộc tính khác sẽ được giới thiệu dần bên dưới theo từng loại thẻ.*

1. **Các thẻ HTML thông dụng**

Có hai nhóm thẻ trong HTML:

* Block: định nghĩa bố cục cho trang web (phân đoạn, tiêu đề, danh sách, phân nhóm…). Một phần tử Block luôn bắt đầu trên một dòng mới và chiếm toàn bộ chiều rộng có sẵn (trải dài sang trái và phải hết mức có thể). Đặc điểm của phần tử này được thấy rõ khi hiển thị là các phần tử sau nó (hiển thị chữ, hình ảnh…) sẽ rơi xuống dòng khác.
* Inline: định nghĩa các phần tử chi tiết của trang web (hiển thị hình ảnh, ngắt dòng văn bản, tạo hyperlink…). Một phần tử Inline không bắt đầu trên một dòng mới và nó chỉ chiếm nhiều chiều rộng khi cần thiết. Đặc điểm của phần tử này được thấy rõ khi hiển thị là các phần tử inline sẽ chung một dòng, như sau thẻ <a> thì các chữ khác vẫn liền kề sau đó.

### a. Nhóm thẻ Block:

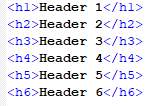
Gồm các thẻ cơ bản:

* <h1>, …, <h6>
* <p>
* <div>
* <dl>, <dt>, <đ>
* <ul>, <ol>, <li>

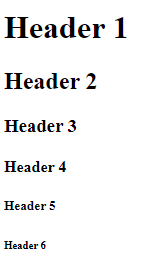
**Định dạng phân mục:** <h1>, …, <h6>

Trong HTML có 6 thẻ dành cho việc phân mục từ lớn tới nhỏ, từ <h1> đến <h6> gần tương tự các mục (A, I, 1, a…)

Ví dụ:



Kết quả:



**Định đạng đoạn văn bản:** <p> </p>

Phẩn tử <p> (paragraph) xác định một đoạn văn bản. Đoạn văn bản này sẽ bắt đầu trên một dòng mới và trình duyệt sẽ tự động thêm một số khoảng trắng (lề) trước và sau nó.

Trong phần tử <p> có thể chứa những chuỗi ký tự, văn bản hay các thẻ thuộc nhóm inline – gồm image, link, button…

*\* Phần tử <p> không chứa các thẻ thuộc nhóm Block.*

Mỗi phần tử <p> chỉ được chứa trong nó một nội dung văn bản (tức một đoạn văn / một hình ảnh / một link…), nếu có 2 đoạn văn bản, nên đặt chúng trong 2 phần tử <p> riêng biệt.

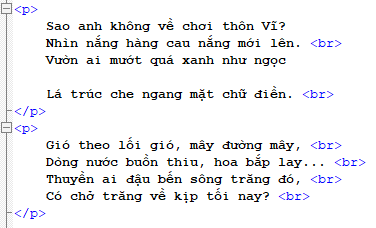
Ví dụ:

<p>Đoạn văn thứ nhất.</p>

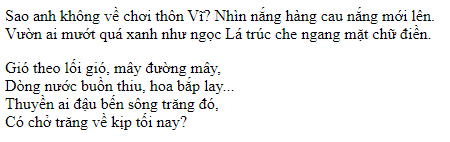
<p>Đoạn văn thứ hai.</p>

Đoạn văn bản trong HTML không nhận ký tự Enter như một ký tự xuống dòng, trường hợp đoạn văn quá dài, ta có thể đặt thẻ <br /> tại nơi muốn xuống dòng đoạn văn đó. Thẻ này thuộc nhóm inline và không được sử dụng để thay thế cho <p> trong trường hợp 2 đoạn văn bản.

Ví dụ:



Kết quả:



**Phân nhóm các đối tượng trên trang:** <div> </div>

<div> (division) là thẻ phân nhóm các đối tượng. Những đối tượng nằm trong cùng một thẻ <div> sẽ có vị trí tương quan cố định với nhau. Thẻ này hỗ trợ để trình duyệt hiểu đúng bố cục trang web, khá giống với chức năng "group" các "Shape" trong Word. Khi được sử dụng cùng với CSS, phần tử <div> có thể được sử dụng để tạo kiểu cho các khối nội dung khác nhau.

Tuy nhiên không nên lạm dụng <div> cho những chi tiết quá nhỏ như những đoạn văn bản riêng lẻ, một hình ảnh, một đường link, ...

Ví dụ:

<p>Đoạn văn thứ nhất.</p>

<div>

<p>1. Đoạn văn trong thẻ p trong div 1.</p>

<p>2. Đoạn văn trong thẻ p trong div 2.</p>

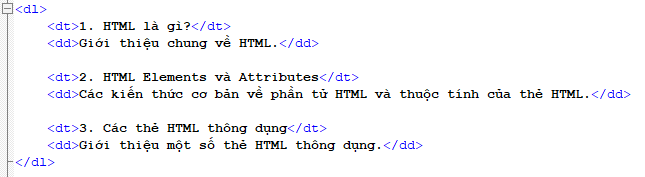
<p>3. Đoạn văn trong thẻ p trong div 3.</p>

</div>

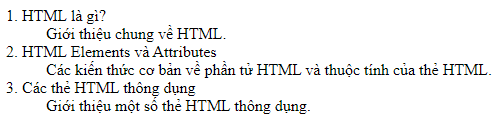
**Bộ thẻ định nghĩa danh sách:** <dl> </dl>, <dt> </dt>, <dd> </dd>

* <dl> (define list) định nghĩa những phần tử trong nó sẽ cấu thành nên một danh sách (list).
* <dt> (define term) định nghĩa phần tử trong nó là một phần tử (item) của danh sách.
* <dd> (define description) định nghĩa phần tử trong nó là một lời giải thích cho item phía trên.

Ví dụ:



Kết quả

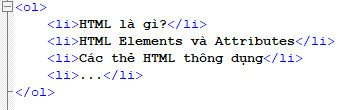


**Bộ thẻ định nghĩa danh sách (có thứ tự):** <ol> </ol>, <li> </li>

<ol> (ordered list) định nghĩa những đối tượng trong nó là những phần tử của một danh sách có thứ tự. Những kí tự đứng trước mỗi đối tượng có thể là một số hoặc một ký tự, một chữ cái.

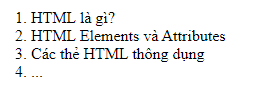
<li> định nghĩa phần tử trong nó là một đối tượng của danh sách.

Ví dụ



*\* Thuộc tính type được dùng trong thẻ <ol> để định nghĩa kiểu kí tự liệt kê của danh sách, type có các giá trị: 1, a, A, i, I.*

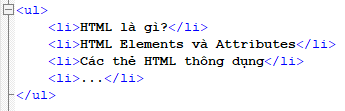
Kết quả



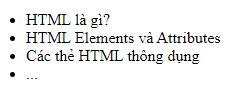
**Bộ thẻ định nghĩa danh sách (không có thứ tự):** <ul> </ul>, <li> </li>

<ul> (unordered list) định nghĩa các phần tử trong nó là những phần tử của một danh sách liệt kê nhưng không có thứ tự. Bộ thẻ này khá tương tự như <dl> nhưng không có chú thích, và có dấu ký hiệu đầu cho mỗi phần tử. Bộ thẻ này sử dụng <li> tương tự như <ol>.

Ví dụ



Kết quả



### b. Nhóm thẻ inline

Gồm các thẻ cơ bản:

* <a> </a>
* <img />
* <br />
* <strong> </strong> / <b> </b>
* <em> </em> / <i> </i>
* <u> </u>

**Định nghĩa một đường dẫn:** <a> </a>

<a> là thẻ tạo 1 liên kết đến một trang web / một email / một file, ...

Thuộc tính *href* xác định địa chỉ cụ thể mà liên kết sẽ dẫn tới.

Thuộc tính mục tiêu *target* quy định cách mở tài liệu được liên kết, gồm các giá trị:

* \_*self*- Mặc định. Mở tài liệu trong cùng một cửa sổ / tab khi nó được nhấp vào
* \_*blank* - Mở tài liệu trong cửa sổ hoặc tab mới
* \_*parent* - Mở tài liệu trong khung chính
* \_*top* - Mở tài liệu trong toàn bộ phần thân của cửa sổ

Ví dụ:



Kết quả:



**Thể hiện hình ảnh:** <img />

<img /> (image) là thẻ chèn hình ảnh vào trang web. Cấu trúc của <img /> không sử dụng thẻ đóng như những thẻ khác mà sử dụng ký tự /.

Thuộc tính *src* quy định cụ thể đường dẫn đến hình ảnh được hiển thị.

Thuộc tính alt định nghĩa một chuỗi ký tự sẽ thay thế cho hình ảnh nếu hình ảnh không thể hiển thị được lên trang web.

Ví dụ: file logo.gif tồn tại và file logo1.gif không tồn tại



Kết quả



**Định dạng ký tự cơ bản cho chuỗi ký tự:** <b> </b>; <i> </i>; <u> </u>

<b> (Bold) là thẻ cho phép ta in đậm chuỗi ký tự đặt giữa nó. Tương tự, <i> (Italic) là in nghiêng và <u> (Underline) là gạch chân chuỗi ký tự.

Ví dụ:



Kết quả:



1. **Các thẻ HTML định dạng Table**

**Các thẻ, thuộc tính cơ bản sử dụng để tạo bảng**

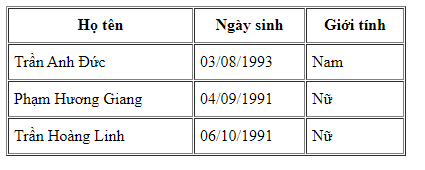
Thông thường, để tạo bảng cần dùng bốn loại thẻ:

* Thẻ <table> dùng để xác định một bảng
* Thẻ <tr> dùng để xác định một hàng bên trong bảng
* Thẻ <th> dùng để xác định một ô (tiêu đề) bên trong hàng
* Thẻ <td> dùng để xác định một ô (bình thường) bên trong hàng

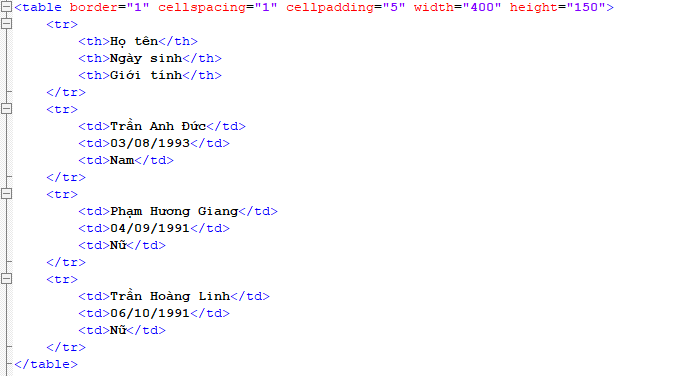
Một số thuộc tính để định dạng bảng:

|  |  |
| --- | --- |
| border | - Thiết lập độ dày của cái đường viền bao xung quanh bảng và các ô. |
| cellspacing | - Thiết lập khoảng cách nằm giữa mỗi hai đường viền lân cận.  - Nếu không thiết lập thì khoảng cách nằm giữa mỗi hai đường viền lân cận sẽ mặc định là 2 pixel. |
| cellpadding | - Thiết lập khoảng cách vùng đệm bên trong các ô.  - Nếu không thiết lập thì khoảng cách vùng đệm bên trong các ô sẽ mặc định là 1 pixel. |
| width | - Thiết lập chiều rộng cho bảng hoặc các ô.  - Nếu muốn thiết lập chiều rộng cho bảng thì đặt thuộc tính width nằm bên trong thẻ <table>. Nếu muốn thiết lập chiều rộng cho một ô thì đặt thuộc tính width nằm bên trong thẻ <th> hoặc <td>  - Khi thiết lập chiều rộng cho một ô thì mặc định những ô nằm chung cột với ô đó sẽ có cùng một chiều rộng. |
| height | - Thiết lập chiều cao cho bảng hoặc các ô. (tương tự thuộc tính width) |
| align | - Canh lề cho nội dung bên trong ô (theo chiều ngang), có 4 giá trị: left, center, right, justify |
| valign | - Canh lề cho nội dung bên trong ô (theo chiều dọc), có 4 giá trị: top, middle, bottom |

Ví dụ, tạo một bảng như sau:



Code HTML:

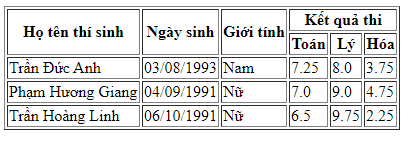


**Thuộc tính colspan và rowspan trong Table**

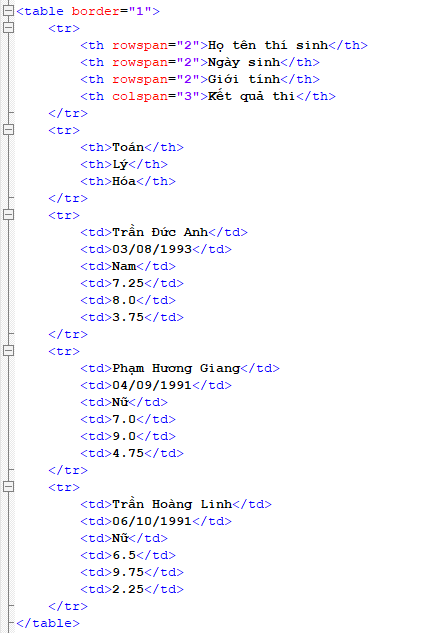
**Colspan**: Thuộc tính colspan dùng để nối hai hoặc nhiều ô với nhau theo chiều ngang tính từ ô đặt thẻ này đi sang bên phải.

**Rowspan**: Thuộc tính rowspan dùng để nối hai hoặc nhiêu ô với nhau theo chiều dọc tính từ ô đặt thẻ đi xuống phía dưới.

Ví dụ, tạo một bảng như sau:



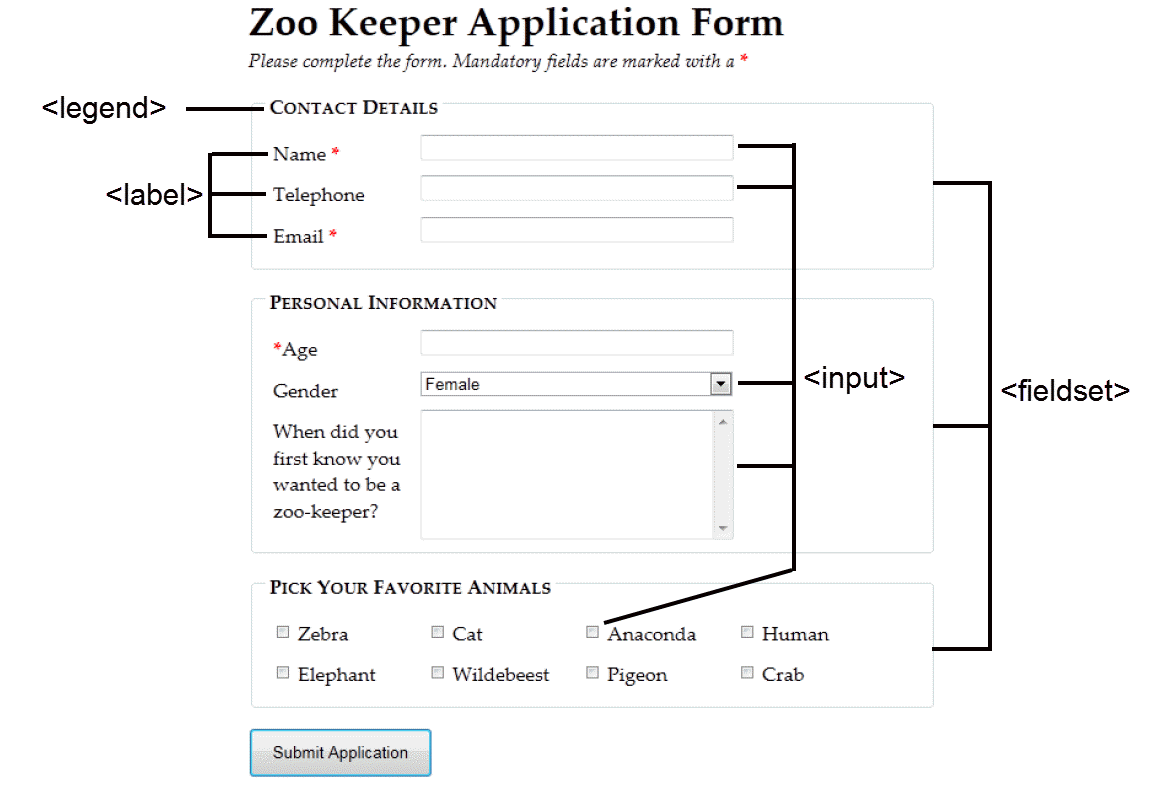
Code HTML



1. **Form trong HTML**

HTML Form là một phương tiện để thu nhập thông tin từ người dùng và gửi về máy chủ để xử lý. Ví dụ, HTML Form được sử dụng trong các phần phản hồi, bình luận bài viết, thanh toán đơn hàng trực tuyến, form đăng nhập, đăng ký…

**Các thành phần cơ bản của form nhập liệu**



Một form HTML cơ bản như hình trên có các thành phần như sau:

* Tiêu đề của form nhập liệu, sử dụng các thẻ <h1>, …, <h6> tùy thuộc vào độ quan trọng của form
* Mô tả form nhập liệu, sử dụng các thẻ <h2>, …, <h6>, <p>
* <fieldset>: Nhóm các ô nhập liệu
* <legend>: Tiêu đề của nhóm các ô nhập liệu
* <label>: tiêu đề của ô nhập liệu
* <input>: ô nhập liệu

Với form nhập liệu, chúng ta có thể có 3 hành động như sau:

* Nhập thông tin vào ô nhập liệu
* Chọn thông tin phù hợp
* Nhấp chuột vào nút gửi dữ liệu

**Phần tử <form> – Tạo khung form nhập liệu**

Phần tử <form> định nghĩa một Form trong HTML



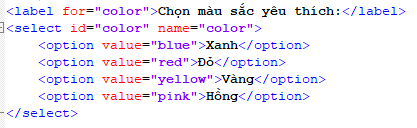
Trong đó có một số thuộc tính:

* action: Định nghĩa hành động sẽ được thực hiện khi nhấn nút gửi dữ liệu, hành động này có thể là gửi dữ liệu lên web server, gửi dữ liệu đến một mail server...
* method: Xác định phương thức HTTP sẽ sử dụng để gửi dữ liệu, có hai loại phương thức thường dùng là GET và POST.
* target: giống như thuộc tính target của thẻ <a> thiết lập sau khi ấn nút submit sẽ vẫn ở nguyên màn hình trình duyệt (giá trị \_self: mặc định) hay sẽ mở một cửa sổ hoặc một tab mới (giá trị \_blank)

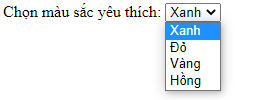
**Phần tử <select> –** xác định một danh sách đổ xuống.

Các phần tử con <option> xác định một tùy chọn trong danh sách.

Ví dụ:



Kết quả:



*\* Theo mặc định, mục đầu tiên trong danh sách thả xuống được chọn.*

*Để xác định một tùy chọn đã chọn trước, hãy thêm thuộc tính selected vào tùy chọn*

Ví dụ:



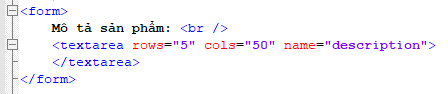
**Thẻ <textarea>**

Định nghĩa: Thẻ <textares> cho phép nhập liệu được trên nhiều dòng, thường dùng cho các nội dung có lượng dữ liệu lớn của form như nội dung mô tả, các nội dung cần nhập thông tin chi tiết, …

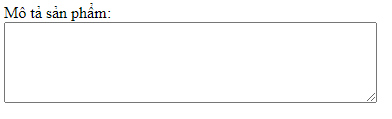
Thuộc tính: có 2 thuộc tính là rows và cols để xác định khung nhập liệu sẽ cao bao nhiêu dòng và rộng bao nhiêu cột.

Cấu trúc: <texrarea rows=”” cols=”” ></textarea>

Ví dụ:



Kết quả:



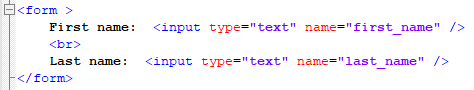
**Thẻ <input> và các dạng thẻ <input>**

Thẻ <input> tạo ra các thành phần nhập liệu như ô nhập liệu, nút lựa chọn dạng checkbox, radio…

**Text**: Thẻ <input> dạng nhập văn bản, đây là dạng phổ biến nhất dùng để nhập văn bản với các thông tin cơ bản như họ tên, mô tả…

Cấu trúc: <input type=”text”>

Ví dụ:

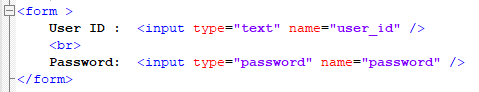


Kết quả:



**Password**: Thẻ <input> dạng nhập mật khẩu, các ký tự khi được nhập vào sẽ được hiển thị ở dạng dấu \* để che dấu các ký tự nhập vào.

Ví dụ:



Kết quả: khi nhập vào ô User ID có type=”text” và ô Password có type=”password”

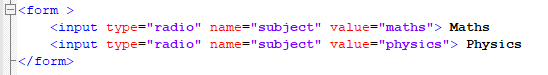


**Radio**: Thẻ <input> dạng nút radio, sử dụng để người dùng có thể lựa chọn duy nhất một giá trị trong danh sách các giá trị.

Cấu trúc: <input type=”radio”>

Lưu ý: Các lựa chọn trong cùng một danh sách phải có cùng “name”

Ví dụ:



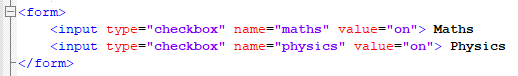
Kết quả:



**Checkbox**: Thẻ <input> dạng nút checkbox, sử dụng để người dùng có thể lựa chọn nhiều giá trị trong một danh sách các giá trị.

Cấu trúc: <input type=”checkbox”>

Ví dụ:



Kết quả:



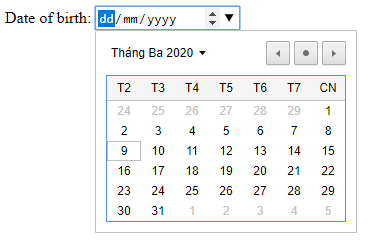
**Date**: Thẻ <input> dạng date dùng để chọn ngày tháng năm.

Cấu trúc: <input type=”date”>

Ví dụ:



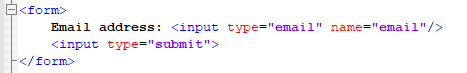
Kết quả:



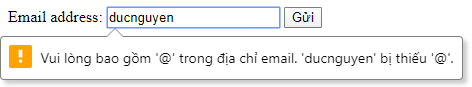
**Email**: Thẻ <input> dạng email sử dụng để nhập liệu địa chỉ email.

Cấu trúc: <input type=”email”>

Ví dụ:



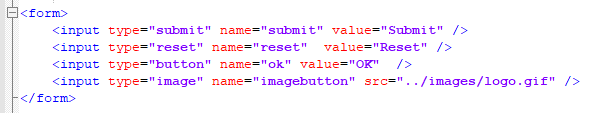
Kết quả: Khi nhập và ấn submit



**Button**: Thẻ <input> dạng nút thông thường, sử dụng tạo ra một nút để khi người dùng nhấn vào thì thực hiện một số hoạt động. Thuộc tính type có thể nhận các giá trị sau:

|  |  |
| --- | --- |
| **Kiểu** | **Miêu tả** |
| submit | Tạo một nút mà tự động chấp nhận một form, dùng để gửi dữ liêu lên máy chủ. |
| reset | Tạo một nút mà tự động reset một form về giá trị ban đầu. |
| button | Tạo một nút được sử dụng để kích hoạt một Client-side khi người sử dụng nhấn vào nút đó. |
| image | Tạo một nút có thể nhấn nhưng chúng ta sử dụng một hình ảnh để làm nền cho nút này. |

Ví dụ:



Kết quả:



Ngoài ra còn nhiều dạng thẻ <input> khác với type như: color, week, month, time, range, search, number, …

**Một số thuộc tính cơ bản của phần tử <input>**

**autocomplete**: gồm 2 giá trị là on/off, nó xác định xem trình duyệt có tự động gợi ý text đã nhập trước đây không. Thuộc tính này chỉ làm việc với các dạng ô nhập liệu như: text, email, password, datepickers…

**value**: thuộc tính value chỉ định một giá trị khởi tạo cho phần tử <input>

**readonly**: chỉ định phần tử chỉ có thể đọc (không thể thay đổi dữ liệu trên phần tử đó)

**disabled:** chỉ định trường input sẽ bị ẩn. Phần tử **disabled**sẽ không được sử dụng và không kích được trên nó. Các phần tử **disabled** sẽ không được gửi đi (submit).

**size** chỉ định kích thước của trường input (số ký tự)

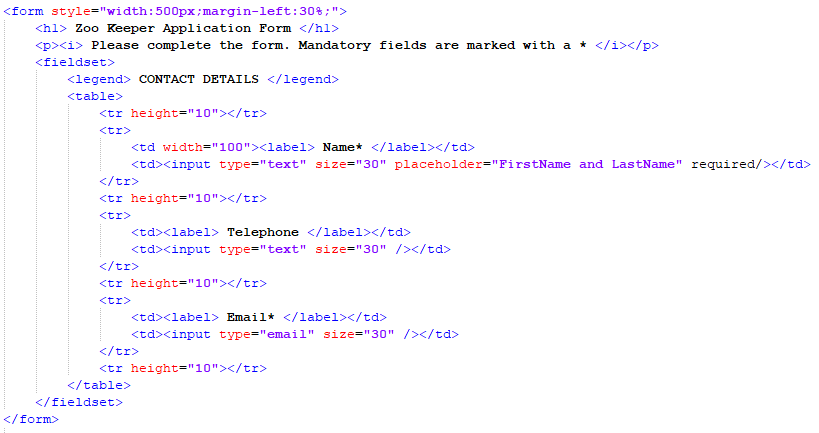
**maxlength:** chỉ định độ dài tối đa cho phép của trường input. Với thuộc tính *maxlength*, control input sẽ không chấp nhận nhiều hơn số ký tự được cho phép. Thuộc tính này không cung cấp bất kỳ phản hồi nào khi nhập quá số ký tự cho phép. Nếu bạn muốn thông báo nhận thông báo, bạn phải viết code JavaScript.

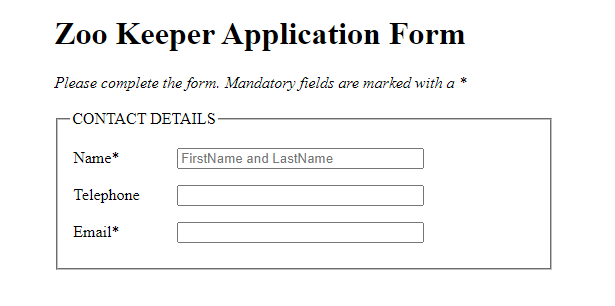
**autofocus**: Khi thẻ <input> nào được thiết lập thuộc tính này thì khi trang web tải, con trỏ nhập liệu sẽ ngay lập tức ở ô nhập liệu này. Thuộc tính này không có giá trị, chỉ cần đưa thuộc tính vào thẻ

**placeholder**: Thuộc tính placeholder để hiển thị gợi ý nhập liệu trong chính thành phần HTML, sử dụng cho các dạng nhập liệu text, search, url, tel, email và password.

**required**: Thẻ required xác định thẻ <input> có phải nhập liệu hay không trước khi gửi dữ liệu form.

**Form HTML: Ví dụ** về một đoạn code tạo một form rộng 500px và cách lề trái 30% như sau

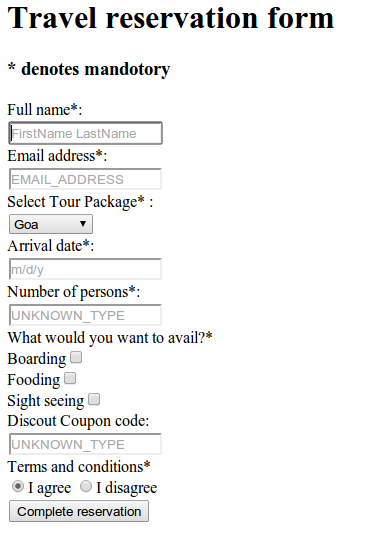


Kết quả:

1. **Bài tập**

Dựa vào những điều đã học được, hãy thiết kế các trang HTML hiển thị giống như các bức ảnh dưới đây.

Ex1:



Ex2: Nội dung form rộng 500px, cách lề trái 30%



# CHAPTER II. CSS

1. **Giới thiệu về CSS**

**CSS là gì?**

Mặc dù HTML cũng có những thuộc tính giúp chúng ta định dạng màu sắc, kích thước của một số thành phần trên web nhưng khi chúng ta muốn tùy chỉnh vị trí của một phần tử hoặc tạo hiệu ứng cho phần tử thì HTML không thể làm được. Lúc này chúng ta sẽ cần sử dụng đến CSS.

CSS là viết tắt của từ viết tắt của từ Cascading Style Sheets. Nó được dùng trong thiết kế giao diện website để tùy chỉnh vị trí các phần tử, màu sắc, màu nền, font chữ, thứ tự sắp xếp của các phần tử, hiệu ứng (đổ bóng, bo góc, xoay...) những điều mà HTML gần như không thể làm được.

**Tác dụng của CSS**

Mối tương quan giữa HTML và CSS rất mật thiết. HTML là ngôn ngữ markup (nền tảng của site) và CSS định hình phong cách (tất cả những gì tạo nên giao diện website), chúng là không thể tách rời.

CSS về lý thuyết không có cũng được, nhưng khi đó website sẽ chỉ là một trang chứa văn bản mà không có gì khác.

Dùng CSS sẽ hạn chế tối thiểu việc làm rối mã HTML của trang Web bằng các thẻ quy định kiểu dáng (chữ đậm, chữ in nghiêng, chữ có gạch chân, chữ màu), khiến mã nguồn của trang Web được gọn gàng hơn, tách nội dung của trang Web và định dạng hiển thị, dễ dàng cho việc cập nhật nội dung.

1. **Cú pháp và Selector CSS**

**Cú pháp CSS**

Một đoạn mã CSS gồm hai phần:

* Selector: chỉ định đối tượng, phần tử HTML muốn điều chỉnh, tạo kiểu (style).
* Declaration: khai báo các thuộc tính CSS dùng để điều chỉnh, tạo kiểu cho selector (định nghĩa Style Rule của selector).
  + Gồm một hoặc nhiều khối khai báo được phân tách với nhau bằng dấu chấm phẩy.
  + Mỗi khối này gồm tên thuộc tính CSS (CSS property) và giá trị (value) được gán cho property đó.
  + Mỗi khối khai báo luôn kết thúc bằng dấu chấm phẩy và toàn bộ phần khai báo declaration được bao quanh bới cặp dấu ngoặc nhọn.

Cấu trúc:

Css\_selector\_1 {

Property\_1: value\_1;

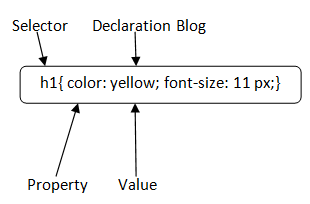
Property\_2: value\_2;

…

Property\_n: value\_n;

}

Ví dụ: đoạn mã CSS chỉ định tất cả các phần tử <h1> trên trang đều có chữ màu vàng và kích cỡ 11px được phân tích như sau:

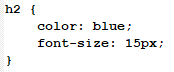


**Các loại Selector trong CSS**

**Element Selector:** chỉ định đối tượng dựa vào tên phần tử HTML, ví dụ thẻ div, thẻ p, thẻ a…

Ví dụ: Chỉ định tất cả các phần tử <h2> trên trang đều có màu xanh và kích cỡ 15px

CSS:

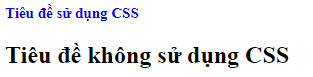


HTML:



*\* Không thêm khoảng trắng giữa giá trị thuộc tính và đơn vị (như font-size: 15 px;). Cách đúng là: font-size: 15px;*

Kết quả: phần tử <h2> khi dùng CSS và không dùng CSS



Ngoài ra còn nhiều cách chọn phần tử HTML như sau:

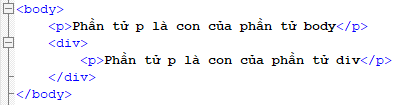
|  |  |  |
| --- | --- | --- |
| **Trường hợp** | **Ví dụ** | **Diễn giải** |
| element, element | div, p | Chọn tất cả phần tử div và tất cả phần tử p |
| element element | div p | Chọn tất cả phần tử p nằm bên trong phần tử div |
| element>element | div>p | Chọn tất cả phần tử p có cha là một phần tử div. Hay nói cách khác phần tử p phải là con trực tiếp của phần tử div. |
| element+element | div+p | Chọn phần tử p được đặt ngay sau phần tử div. |
| element~element | div~p | Chọn tất cả các phần tử p đặt sau phần tử div |

Ví dụ: Chỉ định màu chữ của tất cả các phần tử <p> là con của phần tử <body> có màu đỏ. Các phần tử <p> khác nằm bên trong các phần tử như div, td… sẽ không bị tác động.

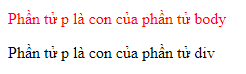
CSS:



HTML:



Kết quả:



**Id Selector:** chỉ định đối tượng bằng việc sử dụng thuộc tính *id* của một phần tử HTML. Trong một webpage, mỗi phần tử nên có một id duy nhất, vì thế Id Selector được dùng để lựa chọn một phần tử duy nhất.

Cú pháp: dùng ký tự # phía trước, sau đó là id của phần tử.

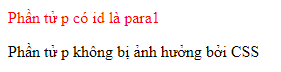
Ví dụ: chỉ định phần tử HTML có id=”para1” sẽ có màu đỏ



HTML:



Kết quả:



*\* Lưu ý: tên id không được bắt đầu bằng số*

**Class Selector:** tương tự như Id Selector, có thể chỉ định đối tượng dựa trên thuộc tính *class* của phần tử đó.

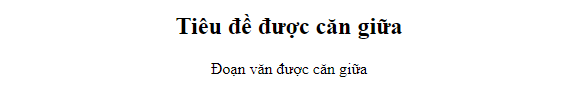
Cú pháp: dùng dấu chấm (.) phía trước, sau đó là tên class của phần tử.

Ví dụ: chỉ định tất cả các phần tử HTML có class=”center” được căn giữa





Kết quả:

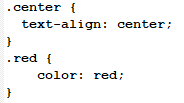


Để chọn phần tử cụ thể trong nhiều phần tử cùng một class: viết tên phần tử đó trước cú pháp trên, như sau:

Ví dụ chỉ định các phần tử <p> có class=”center” mới được căn giữa:



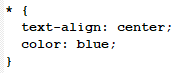
Các phần tử HTML có thể tham chiếu tới nhiều class. Do đó, có thể áp dụng nhiều Style Rule cho một phần tử với các class = "center" (như trên) và class = "red" (để xác định màu đỏ chẳng hạn), như sau:





**Universal Selector:** Là cách chỉ định tất cả các phần tử HTML trên trang đều có chung Style Rule. Cú pháp sử dụng dấu sao (\*) trước khối Declaration.

Ví dụ: chỉ định mọi phần tử trong trang đều có màu đỏ và căn giữa



1. **Cách nhúng CSS vào HTML**

Có 3 cách nhúng CSS vào tài liệu HTML là Inline CSS, Internal CSS và External CSS.

**Inline CSS – Nhúng CSS trong HTML sử dụng thuộc tính style**

Sử dụng thuộc tính *style* để áp dụng Style Rule cho một phần tử nhất định. Giá trị của thuộc tính *style* là tổ hợp của các Declaration và chúng được tách biệt nhau bằng dấu chấm phẩy (;)

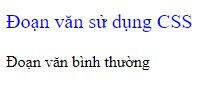
Cú pháp:

<tagname style="Property\_1:value\_1; Property\_2:value\_2;"></tagname>

Ví dụ: chỉ định 1 phần tử <p> có kích cỡ 20px, màu chữ xanh



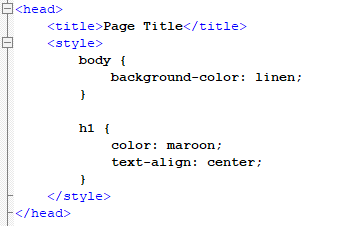
Kết quả:



**Internal CSS – Sử dụng phần tử <style>**

Phần tử <style> được đặt bên trong phần tử <head> định nghĩa Style Rule cho các phần tử trong nội bộ trang HTML đó.

Ví dụ:



**External CSS** – Sử dụng phần tử <link>

Đặt các thuộc tính CSS vào một tệp tin riêng biệt, không chứa phần tử HTML và lưu với phần mở rộng là .css. Sử dụng phần tử <link> để nhúng file css vào bất kỳ trang HTML nào (phần tử <link> được đặt trong nội dung của phần tử <head>)

Cú pháp:

<head>

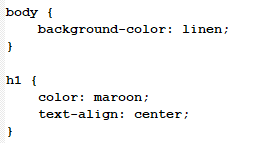
<link rel = "stylesheet" href = "..." />

</head>

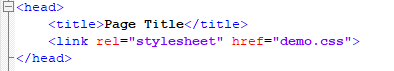
Trong đó, thuộc tính *rel* là khai báo loại tập tin nhúng và *href* là đường dẫn khai báo tên tập tin .css cần nhúng vào.

Ví dụ:

Một file CSS có tên demo.css với nội dung như sau:



Nhúng file CSS trên vào trang HTML như sau:



**Lưu ý:**

* Với những Website lớn và có nhiều page thì nên sử dụng cách External CSS để nhúng file CSS vào trong tài liệu HTML để tránh làm rối mã HTML và dễ quản lý, sửa đổi CSS cho trang Web.
* Độ ưu tiên khi sử dụng các cách trên để nhúng CSS: Inline có quyền cao nhất, tiếp đó là Internal và cuối cùng là External. Nghĩa là khi dùng các cách khác nhau để cùng định nghĩa Style Rule cho cùng một thuộc tính của một phần tử thì Inline CSS có quyền cao nhất, nó sẽ ghi đè các Style Rule được xác định trong phần tử <style>; Style Rule được định nghĩa trong phần tử <style> sẽ ghi đè Rule được định nghĩa trong External CSS file.

1. **Một số thuộc tính CSS thường dùng**

Thuộc tính Background và Color: chỉ định màu nền và màu văn bản của phần tử.

\* Giá trị của màu sắc có thể là tên màu xác định trước hoặc các giá trị RGB, HEX, HSL, RGBA, HSLA.

Thuộc tính Border: chỉ định đường viền của phần tử

Thuộc tính Margin: kiểm soát các lề, để tạo không gian xung quanh các phần tử

Thuộc tính Padding: kiểm soát phần đệm, dể tạo không gian xung quanh nội dung của phần tử

Thuộc tính Width, Height: thiết lập chiều rộng, cao của một phần tử

Thuộc tính Display: quy định phần tử có được hiển thị hay không

Thuộc tính Position: quy định các phương pháp định vị sử dụng cho một phần tử (static, relative, fixed, absolute hay sticky tương ứng với tĩnh, tương đối, cố định, tuyệt đối hay dính)

Thuộc tính Float và Clear: định vị và định dạng phần tử

*\* Có rất nhiều thuộc tính CSS khác để định dạng, căn chỉnh các phần tử. Có thể tìm hiểu thêm trên internet với key word* ***css property***

1. **CSS Table – Bảng trong CSS**

Trong bài về table trong HTML đã hướng dẫn tạo một bảng. Tuy nhiên HTML chỉ hỗ trợ một vài thuộc tính cơ bản (border, cellspacing, cellpadding, …) dùng để định dạng cho bảng. Để tạo được những bảng đẹp và nhìn chuyên nghiệp hơn thì cần vận dụng CSS để định dạng cho chúng.

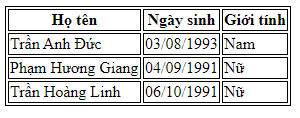
**Thuộc tính *border* – Tạo đường viền cho bảng & các ô bên trong bảng:**

* Thuộc tính border dùng để tạo đường viền cho bảng, ô tiêu đề, ô bình thường (lần lượt là các phần tử <table>, <th>, <td>)
* Đường viền của các phần tử <table>, <th>, <td> là riêng biệt, vì vậy khi khai báo tất cả các đường viền thì sẽ tạo ra bảng có đường viền kép.

Ví dụ:

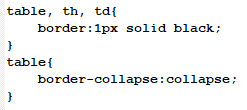


Kết quả:

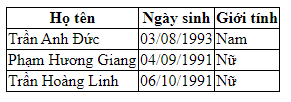


**Gộp các đường viền lân cận lại với nhau:** Có thể dùng thuộc tính *border-collapse* với giá trị collapse cho phần tử <table> để gộp các đường viền lân cận bên trong bảng lại với nhau (hoặc giá trị separate để giữ nguyên đường viền của các phần tử).

Ví dụ:



Kết quả:

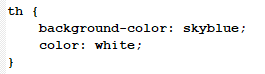


**Thuộc tính background-color và color:**

Thuộc tính *background* dùng để chỉnh màu nền cho bảng hoặc các ô bên trong bảng: thiết lập thuộc tính cho phần tử <table> để chỉnh màu nền cho bảng, phần tử <tr> cho các ô trên một hàng và phần tử <th> (<td>) cho một ô.

Thuộc tính *color* dùng để chỉnh màu văn bản (cách thiết lập tương tự như thuộc tính *background*).

Ví dụ: chỉ định màu nền và màu văn bản của các phần tử <th> như sau:



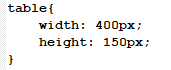
Thử và xem kết quả.

**Thuộc tính Width và Height:**

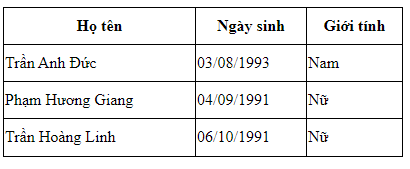
Thay vì dùng thuộc tính *width* và *height* của HTML thì có thể sử dụng thuộc tính *width* và *height* của CSS cũng có kết quả tương tự.

* Để chỉnh chiều rộng (chiều cao) cho bảng thì thiết lập thuộc tính *width* (*height*) cho phần tử <table>
* Để chỉnh chiều rộng (chiều cao) cho ô thì thiết lập thuộc tính *width* (*height*) cho phần tử <th> || <td>

Ví dụ: Thiết lập chiều rộng, chiều cao của bảng lần lượt là 400px, 150px



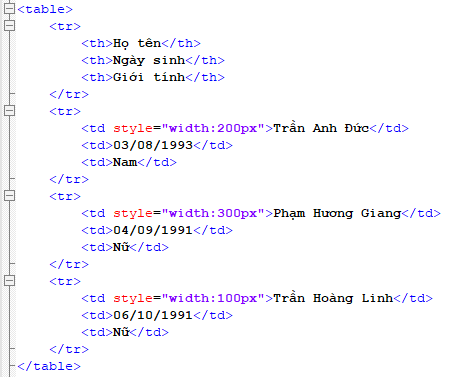
Kết quả:



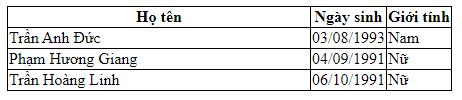
- Lưu ý:

* Đối với các ô được xếp chung một cột, chiều rộng của các ô sẽ được tính dựa theo chiều rộng của ô được thiết lập thuộc tính width với giá trị lớn nhất.
* Đối với các ô được xếp chung một hàng, chiều cao của các ô sẽ được tính dựa theo chiều cao của ô được thiết lập thuộc tính height với giá trị lớn nhất.

Ví dụ: Thiết lập chiều rộng của các ô trong cột đầu tiên với các giá trị khác nhau là 100px, 200px, 300px như sau



Kết quả: Chiều rộng của các ô trong cột thứ nhất là 300px (vì nó là giá trị lớn nhất)



**Thuộc tính padding:**

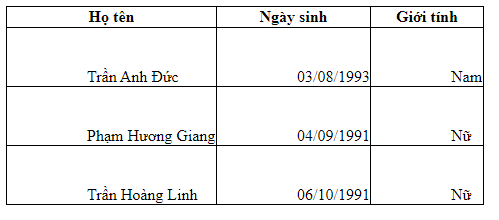
Thuộc tính *padding* dùng dể thiết lập vùng đệm (không gian giữa đường viền và nội dung bảng) bên trong một ô.

Cách sử dụng thuộc tính *padding* để thiết lập vùng đệm cho một ô cũng giống như cách sử dụng thuộc tính *padding* để thiết lập vùng đệm cho một phần tử thông thường.

Ví dụ: Thiết lập thuộc tính padding cho ô dữ liệu thông thường như sau



Kết quả:



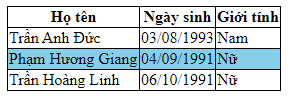
**Bộ chọn :hover**

Sử dụng bộ chọn :hover vào phần tử <tr> để tạo màu cho các hàng của bảng khi di chuột qua nó.

Ví dụ:



Kết quả:



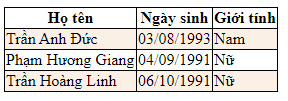
**Bộ chọn nth-child()**

Để tạo 1 bảng kiểu sọc ngựa vằn, sử dụng bộ chọn nth-child() và thuộc tính background-color vào tất cả các hàng của bảng chẵn-even (hoặc lẻ-odd).

Ví dụ: thiết lập màu cho các hàng chẵn



Kết quả:



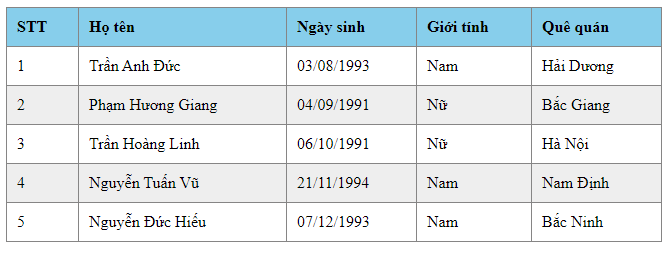
**Một số thuộc tính khác được dùng để định dạng cho bảng:**

Dưới đây là một số thuộc tính khác thường được áp dụng vào việc định dạng cho bảng.

|  |  |
| --- | --- |
| border-spacing | - Thuộc tính này dùng để thiết lập khoảng cách nằm giữa mỗi hai ô liền kề. |
| caption-side | - Thuộc tính này dùng để thiết lập vị trí tiêu đề của cái bảng. |
| empty-cells | - Thuộc tính này dùng để tùy chỉnh việc hiển thị hay ẩn một ô có nội dung rỗng. |
| text-align | - Thuộc tính này dùng để canh lề cho văn bản bên trong ô (theo chiều ngang) |
| vertical-align | - Thuộc tính này dùng để canh lề cho văn bản bên trong ô (theo chiều dọc) |

*\* Tự thử và xem kết quả với từng thuộc tính trên.*

**Kiểm tra:** dùng những kiến thức đã học tạo bảng hiển thị như sau (chiều rộng, chiều cao của bảng lần lượt là 600px, 150px; màu viền, màu nền ô gần giống nhất với hình; khi di chuyển chuột vào các ô thì có màu đậm hơn màu bình thường)



1. **CSS Form – Form trong CSS**

Thêm các thuộc tính CSS định dạng cho form được tạo bằng HTML để tạo form có phần nhìn đẹp và chuyên nghiệp hơn.

Để định dạng form, thường dùng các thuộc tính sau:

**Thuộc tính *width*, *height:*** xác định chiều rộng, chiều cao của các trường nhập vào.

Lưu ý: Muốn xác định style rule cho một kiểu input cụ thể thì có thể sử dụng [*Attribute Selector*](https://quantrimang.com/attribute-selector-trong-css-163136)*:*

* *input[type=text]:* định kiểu cho tất cả các trường có thuộc tính type, giá trị text.
* *input[type=password]:* định kiểu cho tất cả các trường có thuộc tính type, giá trị password.
* *input[type=number]:* định kiểu cho tất cả các trường có thuộc tính type, giá trị number.

**Thuộc tính padding, margin:** xác định khoảng cách giữa các trường nhập vào.

Lưu ý: Bổ sung thêm thuộc tính *box-sizing: border-box* để cho phép phần padding và đường viền của box nằm trọn trong kích thước phần tử đã được định sẵn.

**Thuộc tính border:** xác định kích cỡ, màu đường viền, kết hợp với thuộc tính *border-radius* để định dạng bo tròn góc cạnh.

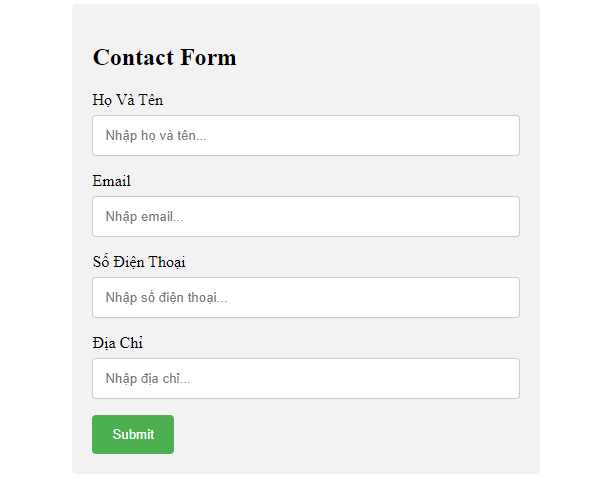
**Thuộc tính background-color và *color*:** xác định màu nền, màu văn bản cho trường nhập vào.

**Thuộc tính *outline*, bộ chọn :focus:** tạo ấn tượng cho các trường nhập vào.

* Một số trình duyệt mặc định sẽ thêm đường viền xung quanh trường nhập khi người dùng click vào, để loại bỏ điều này thì sử dụng thuộc tính outline với giá trị none. (outline: none;)
* Sử dụng selector :focus để tạo ấn tượng cho trường input, gây sự chú ý đến người dùng khi họ nhấp chuột vào ô (có thể nhấn mạnh bằng màu nền với thuộc tính *background* hoặc nhấn mạnh bằng đường viền với thuộc tính *border*).

*\* Còn nhiều thuộc tính khác để định dạng form, có thể tự tìm hiểu thêm trên internet.*

Kiểm tra: sử dụng các phần đã học được tạo form như hình sau (nút submit: khi di chuyển qua sẽ có màu đậm hơn, không có đường viền xung quanh khi nhấn vào; ô nhập liệu có được nhấn mạnh bằng đường viền khi click vào sẽ có màu giống màu nút submit; form được căn giữa; các góc form, ô nhập liệu, submit đều được bo tròn)



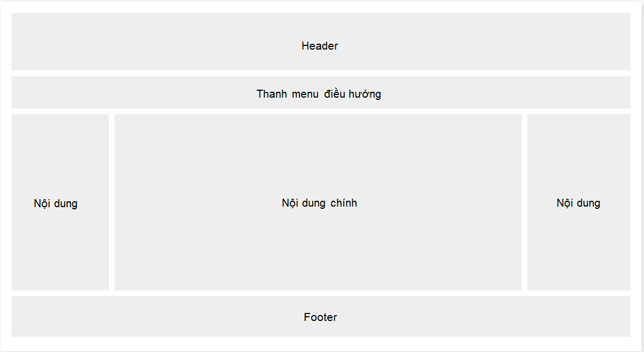
1. **CSS Layout – Bố cục website trong CSS**

**Layout - Bố cục website**

Layout có thể hiểu là cách mà chúng ta bố trí các thành phần chính trên một trang web.

Trong việc thiết kế Layout, thẻ <div> thường được dùng để phân chia các thành phần chính của trang web, kết hợp với thuộc tính định dạng CSS để thiết kế được Layout như ý.

Một website thường được chia thành các phần bao gồm header, thanh menu, nội dung và footer, cơ bản như sau:



**Header:** thường nằm ở đầu trang web (hoặc ngay bên dưới menu điều hướng trên cùng). Phần này thường chứa logo hoặc tên website hay một vài khẩu hiệu của trang web.

**Thanh điều hướng:** hay còn gọi là thanh menu, được dùng để điều hướng các mục chính trên website.

**Nội dung:** Chứa các phần nội dung của trang web**.**

**Footer:** được đặt ở cuối trang, thường chứa thông tin như bản quyền, thông tin liên lạc...

Các bước xây dựng layout cơ bản:

* Định hình website: xác định cấu trúc website cần tạo, vị trí, bố cục các phần, độ dài rộng của các phần, …
* Tạo “xương sống” cho trang web: dùng HTML tạo từng phần phân biệt (header, footer, nội dung, …) của trang web.
* Sử dụng các thuộc tính CSS (width, height, margin, padding, background, :hover, …) làm đẹp và hoàn thiện trang web.

Có một số cách thiết kế website bằng CSS (như HTML Table, HTML Frameset), tuy nhiên kỹ thuật đang được sử dụng phổ biến và đạt hiệu quả khá cao hiện nay là kết hợp HTML Div với CSS.

Xây dựng layout với div và CSS cũng có nhiều cách khác nhau:

* Sử dụng float và clear
* Sử dụng Flex-box
* Sử dụng Grid
* Sử dụng Framework (Bootstrap)

**Sử dụng float và clear:**

Việc chia cột trong CSS là việc thiết lập những phần tử con trong một phần tử mẹ nằm trên cùng một hàng.

Ví dụ, muốn phần nội dung website của có hai cột thì sẽ phải tạo ra 3 phần tử: một phẩn tử <div> gọi nó là container hoặc phần tử mẹ, hai phần tử <div> còn lại có thể gọi là column (cột). Chúng ta cần làm để các phần tử con có thể được chia thành 2 cột.

Các bước chia cột:

1. Các cột phải luôn có một container, tức là phần tử mẹ bao bọc nó.
2. Thiết lập chiều rộng cho container.
3. Thiết lập chiều rộng cho hai cột, tổng chiều rộng trong hai cột phải luôn bằng hoặc ít hơn chiều rộng của container.
4. Nên sử dụng *box-sizing: border-box* để tính toán kích thước chính xác.
5. Sử dụng thuộc tính *float* với giá trị left và right để đẩy phần tử về sang trái hoặc phải.
6. Tiến hành clear float.

**Nguyên lý hoạt động của thuộc tính *float*:**

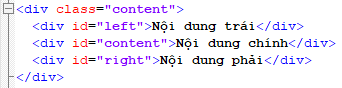
1. Khi một phần tử được thiết lập thuộc tính *float*:
   * Nó sẽ được bắt đầu ở hàng phía trên, nếu hàng phía trên còn đủ chỗ trống để chứa nó.
   * Nó sẽ được bắt đầu ở hàng mới, nếu hàng phía trên không đủ chỗ trống để chứa nó.

Lưu ý: Nếu một phần tử được thiết lập thuộc tính *float* mà trong khi phần tử đứng trước nó không được thiết lập thuộc tính *float*, thì mặc định nó được bắt đầu ở hàng mới.

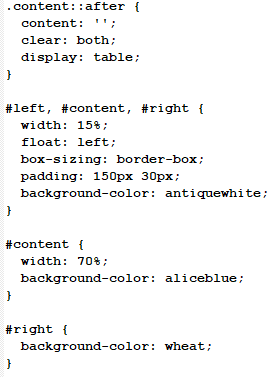
1. Khi hàng không đủ chỗ chứa phần tử thì phần tử phải bắt đầu ở hàng mới. Tuy nhiên, cách bắt đầu như thế nào mới là quan trọng. Khi trên một hàng có nhiều phần tử được thiết lập thuộc tính *float* và mỗi phần tử có chiều cao khác nhau, nếu hàng không đủ chỗ chứa phần tử thì phần tử sẽ bắt đầu bên cạnh phần tử có chiều cao thấp nhất và còn đủ khoảng trống để chứa nó.

Ví dụ: thiết lập phần nội dung gồm 3 phần như sau:

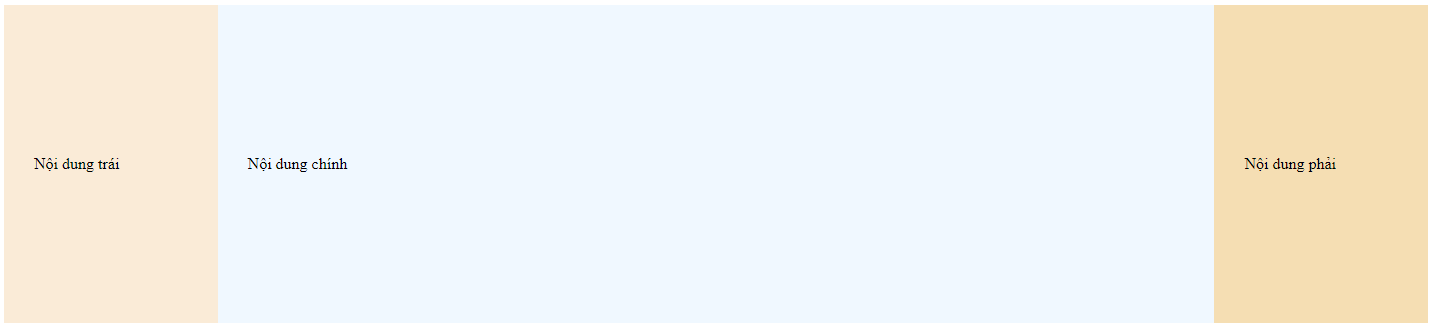
HTML:



CSS:



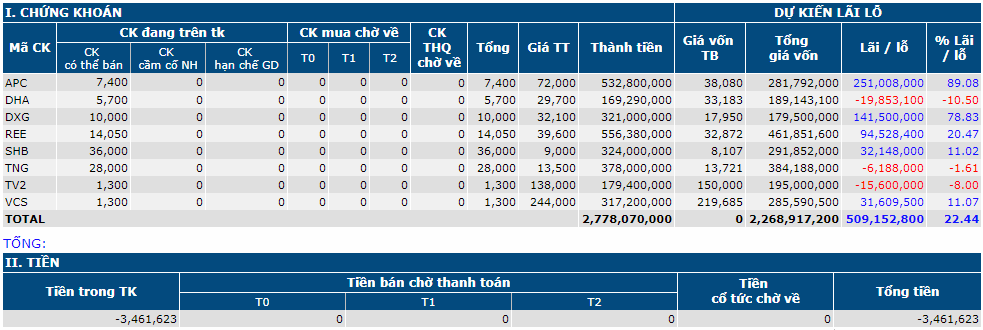
Kết quả:



*\* Tự tìm hiểu và thử các cách xây dựng layout với div và CSS còn lại.*

1. **Bài tập**

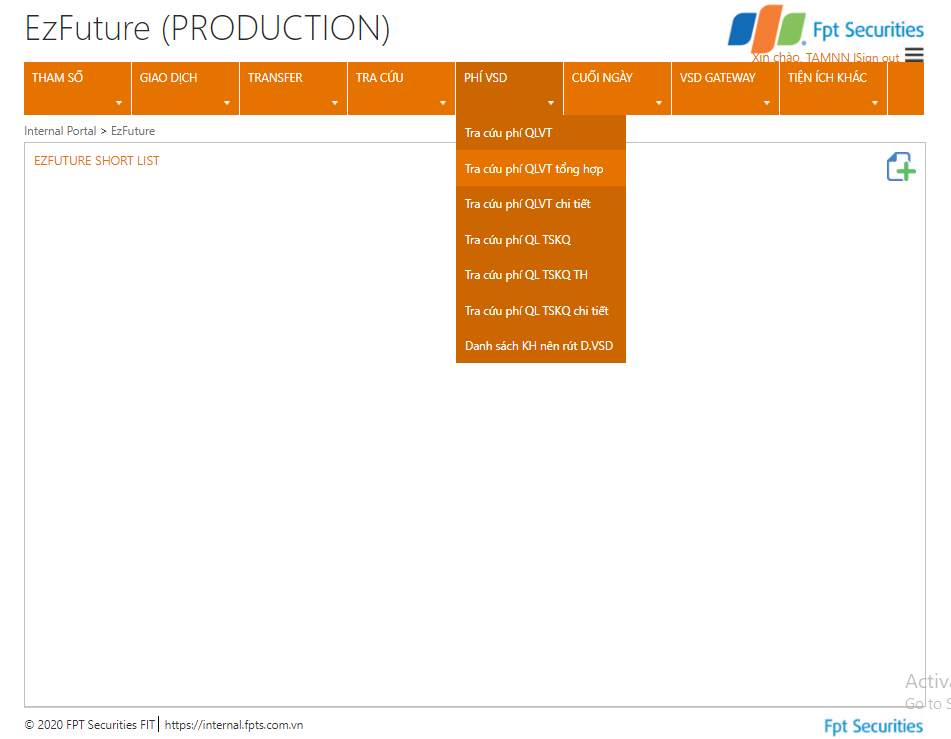
**Ex1:** Sử dụng kiến thức về CSS và CSS Table đã học tạo bảng như hình dưới đây (tham khảo bảng màu CSS để được màu nền, màu chữ của các cột, các ô giống nhất với hình)



**Ex2:** Sử dụng kiến thức về CSS và CSS Layout đã học để tạo trang web sau: (các hình ảnh như logo cần sử dụng được gửi kèm file word này)

Yêu cầu:

* Phần nội dung web rộng 1000px, cách đều hai lề trái, phải
* Menu phía trên đổ xuống các menu con khi click chuột vào
* Khi di chuyển chuột qua và khi click vào menu lớn thì sẽ hiển thị màu đậm hơn, di chuyển chuột qua các menu con sẽ có màu sáng hơn.
* Tham khảo bảng màu CSS để được màu giống nhất với hình



# CHAPTER III. JavaScript

1. **JavaScript là gì? Viết JavaScript ở đâu?**

**JavaScript là gì:**

JavaScript (viết tắt JS) là một ngôn ngữ lập trình thông dịch, được dùng rộng rãi cho các trang web để mang đến sự sinh động, tăng tính tương tác cho trang web. Trong lập trình web, HTML để tạo khung và nội dung, CSS để định dạng thiết kế, trang trí còn JS dùng để cải thiện cách hoạt động của trang web.

***\**** *Ngôn ngữ thông dịch (tiếng Anh: interpreted language) là một loại*[*ngôn ngữ lập trình*](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_l%E1%BA%ADp_tr%C3%ACnh)*mà hầu hết các hiện thực của nó thực thi lệnh trực tiếp và tự do mà không cần*[*biên dịch*](https://vi.wikipedia.org/wiki/Tr%C3%ACnh_bi%C3%AAn_d%E1%BB%8Bch)*sang lệnh*[*ngôn ngữ máy*](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_m%C3%A1y)*.*

JavaScript là ngôn ngữ lập trình hướng đối tượng, có cú pháp tương tự C và được hỗ trợ trên nhiều trình duyệt Firefox, Chrome, … Có thể sử dụng JS để phát triển front end, xây dựng các ứng dụng trên trình duyệt web (lập trình phía client – Client side) hoặc xây dựng ứng dụng phía back-end (lập trình phía server- Server side) với Nodejs.

**Lợi thế của Client side JavaScript:**

* JS được tải về máy của khách truy cập và xử lý tại đó thay vì phía server. Vì vậy dùng JS để xử lý trước khi gửi tới server sẽ giúp tiết kiệm lưu lượng tải ở Server, giúp server xử lý ít hơn, các phản hồi tới khách truy cập nhanh hơn, tăng khả năng tương tác với người dùng.
* JS xử lý nhanh hơn và nhẹ hơn các ngôn ngữ lập trình khác
* Dễ phát hiện và sửa lỗi.

**Hạn chế của Client side JavaScript:**

* Dễ bị khai thác dẫn tới bảo mật kém.
* Không cho phép đọc và ghi các file.
* Có thể bị triển khai khác nhau tùy từng thiết bị dẫn đến việc không đồng nhất.
* Không có khả năng đa luồng hoặc đa xử lý.

**IDE/Text Editor cho JavaScript:**

Để viết code JS có thể sử dụng các trình soạn thảo văn bản (Text Editor) thông thường như Notepad, NotePad++, Sublime text, Atom… hoặc dùng các IDE cho JS như Visual code, WebStorm, NetBeans…

Text Editor nhẹ, không cần cài đặt phức tạp hay cấu hình máy tính cao còn các IDE hỗ trợ lập trình nhiều hơn (gợi ý, hoàn thành cú pháp, soát lỗi, debug, …).

Với những người mới làm quen với JS thì không nên lạm dụng IDE để có thể nhớ được cú pháp, câu lệnh. Sau khi đã quen thì có thể dùng IDE để có thể tập trung vào logic của ứng dụng cần xây dựng và viết code nhanh hơn.

**Vị trí JavaScript trong file HTML:**

Các đoạn mã JavaScript đều được đặt trong cặp thẻ <script></script>.

Cặp thẻ <script> này có thể đặt ở các khu vực:

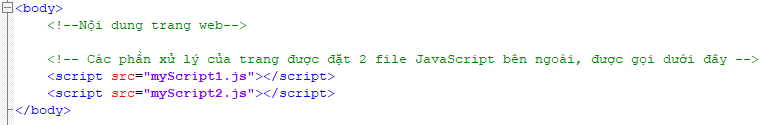
* Nội dung của phần tử <head>
* Nội dung của phần tử <body>
* Đặt trong cả phẩn tử <head> và phần tử <body>

*\* Việc đặt scripts ở cuối phần tử <body> giúp cải thiện tốc độ hiển thị, vì việc diễn giải scripts làm chậm hiển thị.*

Mã JS có thể đặt trong file riêng và sau đó được gọi đến trong file HTML (giống CSS).

* Một file HTML có thể gọi đến nhiều file JavaScript. Đường dẫn của file JavaScript được đặt trong thuộc tính *src* của thẻ <script>
* Nội dung file JavaScript bên ngoài không được chứa thẻ <script>
* Các file JavaScript có phần mở rộng là **.js**

Ví dụ: để thêm 2 file JavaScript có tên là myScript1 và myScript2 đặt trong cùng một thư mục với file HTML như sau:



1. **JavaScript căn bản**

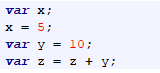
### a. Biến trong JavaScript

**Khai báo biến:**

Biến là vùng chứa để lưu trữ các giá trị dữ liệu. Biến trong JS được khai báo với từ khóa var tenbien

Có thể khai báo biến sau đó gán giá trị cho biến, hoặc khai báo biến đồng thời với gán giá trị cho biến.

Ví dụ: khai báo các biến x, y, z

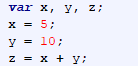


Kết quả: x, y, z lưu giữ giá trị lần lượt là 5, 10, 15

*\* Trong JS, dấu bằng (=) không phải toán tử so sánh mà là một toán tử gán.*

Cũng có thể khai báo nhiều biến với cùng một từ khóa var, các biến phân biệt nhau bằng dấu phẩy (,)

Ví dụ: khai báo các biến x, y, z như sau



Hoặc



Kết quả: tương tự như kết quả của ví dụ trên

**Tên biến:** Các quy tắc cần phải nhớ khi đặt tên biến như sau:

* Biến trong JS được xác định bằng tên duy nhất
* Tên biến có thể chứa chữ cái, chữ số, dấu gạch dưới (\_) và dấu đô la ($)
* Tên biến không được bắt đầu bằng một chữ số
* Tên biến phân biệt chữ hoa chữ thường
* Không thể sử dụng các từ dành riêng (như từ khóa JavaScript) làm tên

**Kiểu dữ liệu JavaScript:**

Có các kiểu dữ liệu cơ bản trong JS như sau:

* Kiểu số Number, ví dụ: 1, 10, 115.5, …
* Kiểu chuỗi văn bản String, giá trị kiểu chuỗi được đặt trong cặp dấu nháy đơn ‘’ hoặc nháy đôi “”, ví dụ: “đây là một giá trị kiểu chuỗi”, …
* Kiểu Boolean, gồm hai giá trị: true và false
* Kiểu Null
* Kiểu Underfined
* Kiểu đối tượng Object
* Kiểu mảng Array (mảng cũng là một kiểu Object), cho phép lưu chữ một dãy các giá trị

Lưu ý: Cả Null và Undefine đều là 2 kiểu dữ liệu biểu thị rằng biến/đối tượng không mang một giá trị nào cả. Nhưng cách dùng và so sánh lại không giống nhau. Trong đó:

* Undefined là giá trị mặc đinh của tất cả các biến/thuộc tính của đối tượng trong JavaScript. Nếu không gán giá trị cho biến thì nó sẽ mặc định là undefined.
* Null có thể coi là một kiểu đối tượng. Một biến/một thuộc tính của đối tượng chỉ có thể có giá trị bằng null khi nó được gán = null.

*\* Các kiểu dữ liệu sẽ được giới thiệu chi tiết hơn trong phần sau.*

### b. Toán tử trong JavaScript:

Trong JS hỗ trợ rất nhiều các toán tử khác nhau ngoài cộng, trừ, nhân, chia đơn thuần và thường được chia thành vài kiểu toán tử cơ bản như sau:

* Toán tử số học
* Toán tử gán
* Toán tử so sánh
* Toán tử logic
* …

**Toán tử số học:** Toán tử số học nhận giá trị kiểu số (lẫn biến kiểu số) làm toán hạng của nó và trả về một giá trị kiểu số.

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| + | Phép cộng  *\* Riêng phép cộng có thể dùng cho chuỗi. Khi đó nó sẽ thực hiện thao tác nối chuỗi* | 10+20 = 30;  “a”+10 = “a10” |
| - | Phép trừ | 20-10 = 10 |
| \* | Phép nhân | 10\*20 = 200 |
| / | Phép chia | 20/10 = 2 |
| % | Phép chia lấy phần dư (kết quả trả về là phần dư khi thực hiện phép chia) | 20%10 = 0;  22%10 = 2 |
| ++ | Phép tăng giá trị hiện tại lên 1 đơn vị | var a=10; a++;  Lúc này a = 11 |
| -- | Phép giảm giá trị hiện tại xuống 1 đơn vị | var a=10; a--;  Lúc này a = 9 |

**Toán tử gán:** Toán tử gán thường sử dụng nhiều nhất trong ngôn ngữ lập trình, dùng để gán giá trị của biểu thức bên phải cho biểu thức bên trái. Trong JS thường hay dùng các toán tử gán sau:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| = | Phép gán | c = a + b;  Lúc này c sẽ có giá trị bằng giá trị của a + b |
| += | Phép cộng và gán:  Cộng giá trị toán hạng phải với toán hạng trái và gán kết quả cho toán hạng trái | var a=10; a+=20;  Tương đương với a = a+20  Lúc này a = 30 |
| -= | Phép trừ và gán:  x -= y tương đương với x = x - y | var a=20; a-=10;  Lúc này a = 10 |
| \*= | Phép nhân và gán:  x \*= y tương đương với x = x \* y | var a=10; a\*=20;  Lúc này a = 200 |
| /= | Phép chia và gán  x /= y tương đương với x = x / y | var a=10; a/=2;  Lúc này a = 5 |
| %= | Phép chia lấy phần dư và gán  x %= y tương đương với x = x % y | var a=10; a%=2;  Lúc này a = 0 |

**Toán tử so sánh:** toán tử so sánh dùng để so sánh giữa hai giá trị (của biến) nào đó. Kết quả của toán tử so sánh trả về một trong hai giá trị True hoặc False

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| == | Phép bằng:  Kiểm tra giá trị của hai toán hạng có bằng nhau không, có thì trả ra giá trị True | 10==20  Trả ra: False |
| === | Phép bằng (cả về giá trị và kiểu dữ liệu)  Kiểm tra giá trị và kiểu dữ liệu của hai toán hạng có bằng nhau không, có thì trả ra True | 10===10; Trả ra True  10===”10”; Trả ra False |
| != | Không cân bằng:  Giá trị của hai toán hạng không cân bằng thì trả ra True | 10 != 20  Trả ra True |
| !== | Không cân bằng (cả về giá trị và kiểu dữ liệu)  Giá trị hoặc kiểu dữ liệu của hai toán hạng không cân bằng thì trả ra True | 20!==20; Trả ra False  20!==”20”; Trả ra True  20!==10; Trả ra True |
| > | Lớn hơn:  Giá trị toán hạng trái lớn hơn toán hạng phải thì trả ra True | 20>10;  Trả ra True |
| >= | Lớn hơn hoặc bằng:  Giá trị toán hạng trái lớn hơn hoặc bằng toán hạng phải thì trả ra True | 20>=20;  Trả ra True |
| < | Nhỏ hơn:  Giá trị toán hạng trái nhỏ hơn giá trị toán hạng phải thì trả ra True | 20<20;  Trả ra False |
| <= | Nhỏ hơn hoặc bằng:  Giá trị toán hạng trái nhỏ hơn hoặc bằng giá trị toán hạng phải thì trả ra True | 20<=20;  Trả ra True |

**Toán tử logic:** toán tử logic thường dùng để kết hợp hoặc phủ định… các biểu thức có chứa toán tử so sánh. Kết quả của toán tử logic cũng trả về một trong hai giá trị True hoặc False

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| && | Logical AND:  Trả về kết quả là True khi cả hai toán hạng đều true | (10==10 && 20==33);  Trả ra False |
| || | Logical OR:  Trả về kết quả là True khi một trong hai (hoặc cả hai) toán hạng là True | (10==10 || 20==33);  Trả ra True |
| ! | Logical Not:  Chuyển đổi giá trị của toán hạng từ True sang False và ngược lại | !(10==20);  Trả ra True |

*\* Còn rất nhiều các toán tử khác, có thể tự tìm hiểu hoặc được giới thiệu trong các phần sau.*

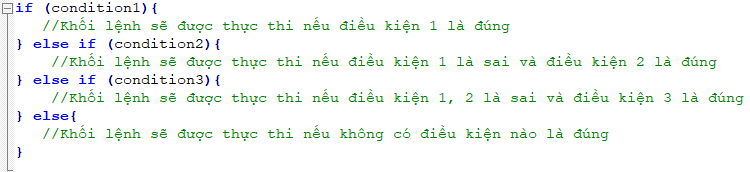
### c. Lệnh kiểm tra điều kiện if… else

Câu lệnh điều kiện được sử dụng để thực hiện các hành động khác nhau dựa trên các điều kiện khác nhau.

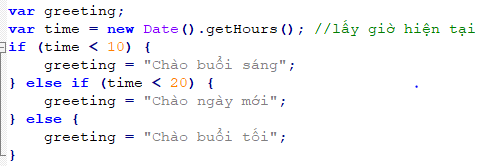
Trong JavaScript, chúng ta có các câu lệnh điều kiện sau:

* Sử dụng ***if*** để chỉ định một khối mã sẽ được thực thi, nếu một điều kiện được chỉ định là đúng
* Sử dụng ***else*** để chỉ định một khối mã sẽ được thực thi, nếu điều kiện tương tự là sai
* Sử dụng ***else if*** để chỉ định một điều kiện mới để kiểm tra, nếu điều kiện đầu tiên là sai

Cú pháp chung:



Ví dụ: Nếu thời gian nhỏ hơn 10:00h, hãy tạo lời chào "Chào buổi sáng", nếu không, những thời gian nhỏ hơn 20:00h, hãy tạo lời chào "Chào ngày mới", còn lại sẽ là "Chào buổi tối"



Kết quả:

Nếu hiện tại là 9h, lời chào sẽ là “Chào buổi sáng”

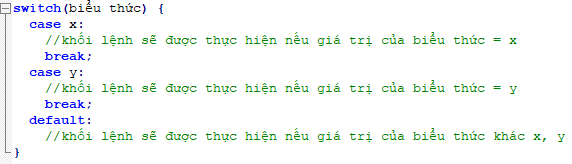
Nếu hiện tại là 21h, lời chào sẽ là “Chào buổi tối”

*\* Các chữ if, else trong câu lệnh điều kiện là chữ thường. Các chữ hoa (If hoặc IF) sẽ tạo ra lỗi JavaScript.*

### d. Lệnh kiểm tra điều kiện switch… case

Câu lệnh *switch case* cũng có công dụng như lệnh *if else*, dùng để thực hiện các hành động khác nhau dựa trên các điều kiện khác nhau.

Cú pháp:



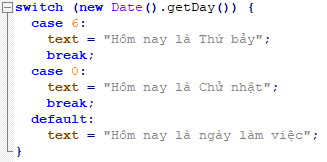
Đây là cách nó hoạt động:

* Biểu thức trong *switch* được đánh giá một lần.
* Giá trị của biểu thức được so sánh với giá trị của từng trường hợp.
* Nếu có sự trùng khớp, khối lệnh ngay bên dưới (trước lệnh *break;*) sẽ được thực thi.
* Nếu không khớp, khối lệnh dưới *default* mặc định được thực thi.

*\* Lệnh break là bắt buộc ở dưới cùng mỗi khối lệnh của một điều kiện (case). Nếu quên, các lệnh bên dưới lệnh break sẽ tiếp tục được thực hiện ngay cả khi đánh giá không khớp với trường hợp đó.*

*\* Cụ thể về lệnh break sẽ được học trong phần Điều khiển vòng lặp.*

Ví dụ: kiểm tra ngày hiện tại, nếu là thứ 7 hoặc chủ nhật thì trả ra tên ngày, nếu không thì mặc định trả ra “Hôm nay là ngày làm việc”. Sử dụng phương thức getDay() trả về các ngày trong tuần là một số từ 0 đến 6 (0 là Chủ nhật, 1 là Thứ hai, …)



Kết quả:

Nếu hiện tại là Thứ ba, sẽ trả ra “Hôm nay là ngày làm việc”

Nếu hiện tại là Thứ bảy sẽ trả ra “Hôm nay là Thứ bảy”

### e. Vòng lặp for

Các vòng lặp được dùng khi cần thực thi một khối lệnh nhiều lần. Trong JS hỗ trợ nhiều loại vòng lặp như for, for/in, for/of, while, do/while. Ở đây chỉ tìm hiểu hai loại vòng lặp cơ bản là for và while, do/while.

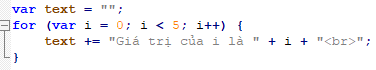
Cú pháp của vòng lặp for như sau:



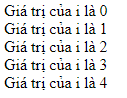
Trong đó:

* Statement 1 – khởi tạo vòng lặp: được thực hiện một lần trước khi khối mã thực thi, dùng để khởi tạo biến được dùng trong vòng lặp (có thể khởi tạo và gán luôn giá trị ban đầu cho biến)
* Statement 2 – kiểm tra lệnh: thường được sử dụng để đánh giá điều kiện của biến ban đầu. Nếu điều kiện trả về True, các khối lệnh sẽ được thực hiện, vòng lặp sẽ tiếp tục. Nếu trả về False, vòng lặp sẽ kết thúc
* Statement 3 – lệnh lặp lại: được thực hiện mọi lúc sau khi khối lệnh đã được thực thi, thường được dùng để tăng (giảm) giá trị của biến ban đầu.

Ví dụ: khởi tạo biến i có giá trị ban đầu là 0, điều kiện của vòng lặp là i<5, tăng giá trị của i lên 1 sau mỗi lần thực hiện khối lệnh



Kết quả: giá trị của biến text trả ra như sau



*\* Cần lưu ý điều kiện của vòng lặp để chương trình có thể kết thúc sau một số hữu hạn lần lặp, tránh vòng lặp lặp lại vô hạn dẫn đến làm hỏng chương trình (trình duyệt).*

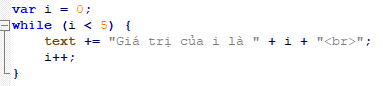
### f. Vòng lặp while; do… while

**Vòng lặp while:** vòng lặp while sẽ thực hiện khối lệnh miễn là một điều kiện nhất định là đúng.

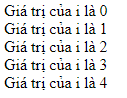
Cú pháp:



Ví dụ: khối lệnh sẽ chạy đi chạy lại miễn là biến (i) nhỏ hơn 5



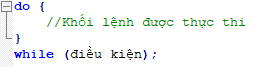
Kết quả:



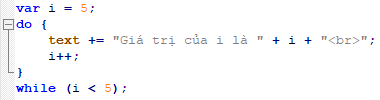
*\* Cần lưu ý tăng biến được sử dụng trong điều kiện. Nếu quên vòng lặp sẽ không bao giờ kết thúc.*

**Vòng lặp do/while:** vòng lặp do/while là một biến thể của vòng lặp while. Vòng lặp này sẽ thực hiện khối mã một lần, trước khi kiểm tra xem điều kiện có đúng không, sau đó nó sẽ lặp lại vòng lặp miễn là điều kiện là đúng.

Cú pháp:



Ví dụ:



Kết quả:



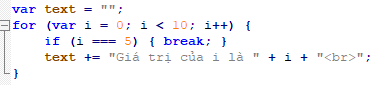
### g. Điều khiển vòng lặp (lệnh break; continue;)

**Lệnh break:**

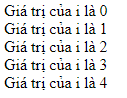
Lệnh break đã được giới thiệu trong lệnh kiển tra điều kiên switch case, nó cũng được dùng trong vòng lặp, để dừng vòng lặp cho dù điều kiện của vòng lặp vẫn đang đúng.

Lệnh break có thể dùng cho mọi vòng lặp như for, while, while/do…

Ví dụ: vòng lặp for lặp từ 1 đến 10 và bị dừng lại ở vòng lặp thứ 5



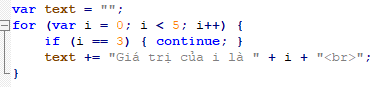
Kết quả: giá trị của biến text sau khi chạy vòng lặp là



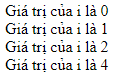
Lệnh continue:

Câu lệnh continue ngắt một lần lặp (trong vòng lặp), nếu một điều kiện xác định xảy ra và tiếp tục với lần lặp tiếp theo trong vòng lặp.

Ví dụ: vòng lặp for bỏ qua đoạn code thêm giá trị vào biến text khi giá trị biến i bằng 3



Kết quả:



### h. Function – hàm trong JavaScript

**Function:** Hàm là một khối lệnh được thiết kế để thực hiện một tác vụ cụ thể. Các lệnh trong hàm được thực thi khi ta gọi đến hàm đó trong chương trình. Điều này giúp ta không cần phải viết cùng một code nhiều lần, giúp dễ quản lý và code trông gọn hơn.

Cú pháp:



Trong đó:

* Một hàm bắt đầu bởi từ khóa **function**, theo sau là tên hàm, tiếp đến là cặp dấu ngoặc đơn () gồm danh sách tham số, cuối cùng là cặp dấu ngoặc nhọn {} gồm khối lệnh được thực thi bởi hàm
* Tên hàm có quy tắc giống quy tắc đặt tên biến
* Có thể không có, có một hoặc nhiều tham số trong cặp dấu (), các tham số cách nhau bởi dấu phẩy (,)

**Lệnh return:**

Lệnh return dùng để trả về một giá trị từ hàm. Hàm sẽ ngừng thực thi khi đến một lệnh return. Vì vậy lệnh này nên là lệnh cuối của một hàm.

Trong một hàm có thể có lệnh return hoặc không.

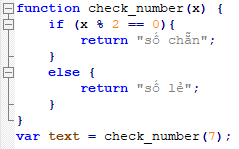
**Gọi một hàm:**

Dùng cú pháp function\_name(parameter1, parameter 2, parameter3) để gọi một hàm với:

* function\_name là tên hàm, tiếp đến là cặp dấu ngoặc đơn (), trong cặp dấu ngoặc là các tham số đầu vào tương ứng
* tham số truyền vào phải là giá trị cụ thể hoặc biến có giá trị cụ thể.
* những tham số không được truyền cho hàm sẽ mang giá trị undefined.

*\* Truy cập một hàm không có phần () sẽ trả về đối tượng hàm thay vì kết quả của hàm.*

**Ví dụ về hàm:** một hàm dùng để kiểm tra tính chẵn lẻ của số x và trả ra “số chẵn” hoặc “số lẻ”



Kết quả: Giá trị của biến text là “số lẻ”

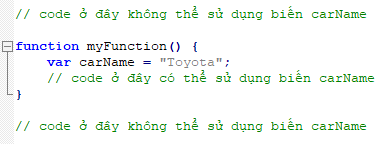
### i. Biến toàn cục và biến cục bộ

**Biến cục bộ:**

Các biến được khai báo trong một hàm JS được gọi là biến cục bộ của hàm đó. Các biến cục bộ chỉ sử dụng được trong nội bộ hàm đó.

Các biến cục bộ được tạo khi một hàm khởi động và bị xóa khi hoàn thành hàm.

Ví dụ:



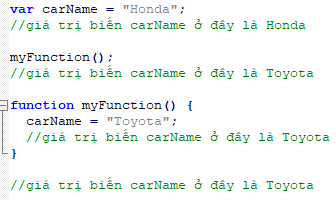
Vì các biến cục bộ chỉ được nhận dạng bên trong các hàm của chúng, các biến có cùng tên có thể được sử dụng trong các hàm khác nhau.

Biến toàn cục:

Biến toàn cục là biến được khai báo ở bên ngoài, không nằm bên trong một hàm cụ thể nào cả.

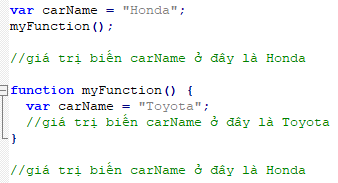
Biến toàn cục có thể được dùng trong toàn bộ cả bên trong và ngoài hàm.

Ví dụ: biến carName được khai báo bên ngoài, là biến cục bộ nên bên trong hàm cũng có thể sử dụng, các thay đổi trong hàm sẽ làm thay đổi giá trị biến toàn cục



**Lưu ý:** nếu trong hàm có sử dụng từ khóa var để tạo một biến và tên biến đó trùng với tên biến toàn cục thì bên trong hàm sẽ hiểu là đang sử dụng biến cục bộ. Mọi thay đổi trong hàm không ảnh hưởng đến giá trị của biến toàn cục.

Ví dụ: cùng tên biến carName nhưng function không làm giá trị biến carName toàn cục thay đổi



1. **Kiểu dữ liệu và phương thức xử lý dữ liệu**

### a. JS Object – Đối tượng trong JavaScript

**Khái niệm Đối tượng:**

Đối tượng là một thực thể trong thế giới thực, ví dụ như cái bàn, quyển sách, điện thoại…

Một đối tượng, ví dụ như một người có các thuộc tính như tên, tuổi, quê quán… phương thức như nói chuyện, học tập…

Mọi người đều có các thuộc tính giống nhau, nhưng giá trị của các thuộc tính khác nhau.

Mọi người đều có các phương thức giống nhau nhưng được thực hiện ở những thời điểm khác nhau.

**Đối tượng trong JavaScript:**

Trong JS, biến dùng để chứa một giá trị dữ liệu, biến cũng có thể là một đối tượng gồm nhiều giá trị.

Khai báo một biến là một đối tượng với căp ngoặc nhọn {}, các giá trị được viết dưới dạng cặp tên và giá trị name: value, phân tách nhau bằng dấu hai chấm; cách cặp name:value phân tách bởi dấu phẩy.

Ví dụ:



**Thuộc tính của đối tượng:**

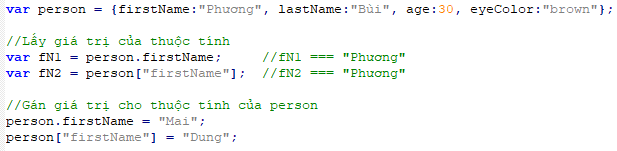
Các cặp name: value trong JS Object được gọi là thuộc tính: giá trị của thuộc tính.

Ví dụ: với đối tượng person ở trên ta có các thuộc tính là firstName, lastName, age, eyeColor; tương ứng với các giá trị của thuộc tính là Phương, Bùi, 30, brown

Các thao tác với thuộc tính của đối tượng như sau:

* Lấy giá trị của thuộc tính (cú pháp objectName.property hoặc objectName[“property”]
* Gán giá trị cho thuộc tính (dùng toán tử gán =)

Ví dụ:

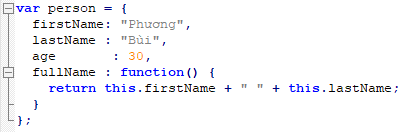


**Phương thức của đối tượng:**

Phương thức là các hành động có thể thực hiên trên các đối tượng.

Có thể tạo các phương thức cho một đối tượng bằng cách định nghĩa một hàm cho đối tượng đó.

Ví dụ:



*\* Từ khóa “this”: Trong một định nghĩa hàm, this đề cập đến "chủ sở hữu" của hàm. Tự tìm hiểu thêm về từ khóa “this” trong JavaScript.*

Với đối tượng person ở trên ta có:

* Các thuộc tính là firstName, lastName, age; tương ứng với các giá trị của thuộc tính là Phương, Bùi, 30.
* Phương thức là fullName với giá trị là function () {return this.firstName + " " + this.lastName; }
* Ở đây, *this* là đối tượng person "sở hữu" hàm fullName. (nói cách khác, this.firstName có nghĩa là thuộc tính firstName của đối tượng này)

Để truy cập vào một phương thức của đối tượng, dùng cú pháp: objectName.methodName()

Ví dụ: gọi phương thức fullName của đối tượng person ở trên



*\* Nếu truy cập một phương thức không có dấu ngoặc đơn (), nó sẽ trả về định nghĩa hàm (ví như name.fullName sẽ trả ra function() { return this.firstName + " " + this.lastName; } )*

### b. JS String – Chuỗi trong JavaScript

**Khái niệm:**

Chuỗi trong JavaScript được sử dụng để lưu trữ và thao tác văn bản. Một chuỗi gồm không hoặc nhiều ký tự được đặt bên trong dấu nháy đơn ‘’, hoặc nháy kép “”

Ví dụ:



**Dấu nháy trong chuỗi:**

Nếu trong chuỗi có một số kí tự đặc biệt như nháy đơn, nháy đôi… sẽ làm lỗi, hoặc bị hiểu sai thành kết thúc một chuỗi.

Ví dụ:



Kết quả: Giá trị của biên answer1 là “Cô ấy tên là Phương. “

Để dùng dấu nháy bên trong một chuỗi, ta có thể dùng dấu nháy bao quanh chuỗi khác với dấu nháy trong chuỗi.

Ví dụ: Sử dụng dấu nháy đơn bên ngoài chuỗi và nháy kép trong chuỗi hoặc ngược lại

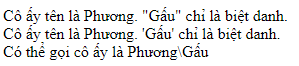


Sử dụng ký tự gạch chéo ngược (\): có thể sử dụng dấu \ để biến các ký tự đặc biệt thành ký tự chuỗi.

Ví dụ:



Kết quả: giá trị của các biến answer1, answer2, answer3 lần lượt là



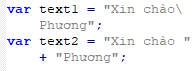
**Ngắt dòng các mã dài:**

Để dễ đọc nhất, các lập trình viên thường muốn tránh các dòng mã dài hơn 80 ký tự.

Để chia nhỏ một dòng mã trong một chuỗi văn bản, có thể dùng 2 cách sau:

* Sử dụng dấu gạch chéo ngược \
* Sử dụng nối chuỗi (toán tử +)

Ví dụ: hai cách sau đây đều cho kết quả giá trị của biến là “Xin chào Phương”



Lưu ý:

* Cần thiết phải sử dụng một trong hai cách nếu muốn ngắt dòng chuỗi. Nếu chỉ Enter xuống hàng mà không dùng sẽ dẫn đến lỗi cú pháp.
* Trong hai cách thì nên dùng phương pháp nối chuỗi. Các phương pháp dùng \ có thể không được hỗ trợ ở một số trình duyệt.

**Chiều dài chuỗi:** length là một property trả về chiều dài của một chuỗi.

Ví dụ:



Kết quả: giá trị của biến sln là 26

**Tìm kiếm chuỗi con:**

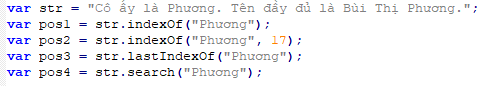
JS có 3 phương thức thường dùng để tìm kiếm chuỗi con:

* indexOf() – String.indexOf(str) là tìm chuỗi con str trong chuỗi cha String. Phương thức trả về vị trí xuất hiện đầu tiên của chuỗi con (bắt đầu là vị trí 0), nếu không tìm thấy chuỗi con sẽ trả về -1.
* lastIndexOf() – String.lastIndexOf(str) là tìm chuỗi con str trong chuỗi cha String. Phương thức trả về vị trí xuất hiện của chuỗi con cuối cùng được tìm thấy (trong trường hợp chuỗi con xuất hiện nhiều lần), nếu không tìm thấy chuỗi con sẽ trả về -1.
* search() – String.search(str) để tìm kiếm chuỗi con str trong chuỗi cha String. Phương thức trả về vị trí xuất hiện đầu tiên của chuỗi con (bắt đầu là vị trí 0), nếu không tìm thấy chuỗi con sẽ trả về -1.

Lưu ý:

* Hai phương thức indexOf() và lastIndexOf() cho phép bắt đầu tìm kiếm chuỗi từ một vị trí bất kì nào đó trở đi. Với x là một số chỉ vị trí, ta có cú pháp String.indexOf(str,x) để tìm kiếm chuỗi con str trong chuỗi cha String bắt đầu từ vị trí x. Tương tự với phương thức lastIndexOf()
* Phương thức search() thì không hỗ trợ việc tìm kiếm bắt đầu từ một vị trí nào đó này. Nhưng việc tìm kiếm dùng search() sẽ cho kết quả nhanh hơn.

Ví dụ:



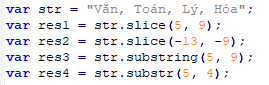
Kết quả: giá trị của các biến pos1, pos2, pos3, pos4 lần lượt là 9, 39, 39, 9

Cắt chuỗi con:

JS có 3 phương thức thường dùng để cắt chuỗi con như sau:

* slice() – String.slice(start, end) dùng để cắt lấy và trả ra một chuỗi con từ chuỗi cha String bắt đầu từ vị trí start đến vị trí kết thúc end. Nếu tham số truyền vào là số âm thì nó sẽ tính ngược lại, nghĩa là sẽ đếm từ cuối lên. Nếu chỉ truyền một tham số đầu tiên start thì nó sẽ tự hiểu vị trí end là vị trí cuối cùng.
* substring() – String.substring(start, end) dùng để cắt lấy một chuỗi con từ chuỗi cha String bắt đầu từ vị trí start đến vị trí kết thúc end. Tham số start, end truyền vào bắt buộc phải lớn hơn 0.
* substr() – String.substr(start, length) dùng để cắt lấy một chuỗi con gồm length ký tự từ chuỗi cha String bắt đầu từ vị trí start. Tham số length luôn dương, tham số start nhỏ hơn không thì nó sẽ đếm ngược từ cuối trở lên.

Ví dụ:



Kết quả: giá trị của tất cả các biến res1, res2, res3, res4 đều là “Toán”

*\* Tất cả các vị trí của chuỗi đều bắt đầu từ 0, vì vậy khi tính toán vị trí cần lưu ý để tránh bị nhầm lẫn.*

**Thay thế nội dung chuỗi:**

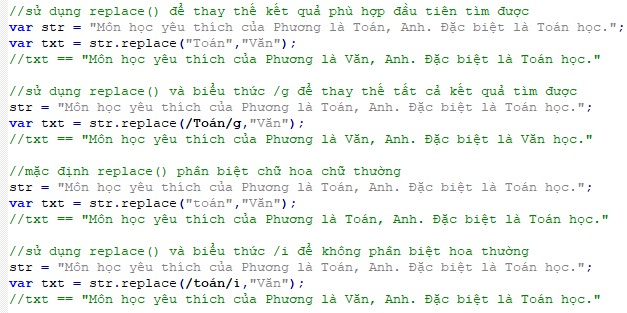
Dùng phương thức replace() trả về một chuỗi có sự thay thế một giá trị nhất định với một giá trị trong chuỗi ban đầu.

Cú pháp: string.replace(str1, str2) – thay thế chuỗi ký tự str1 bằng chuỗi str2 trong chuỗi ban đầu str.

Lưu ý:

* Theo mặc định, phương thức replace() chỉ thay thế kết quả phù hợp đầu tiên. Để thay thế tất cả các kết quả phù hợp, hãy sử dụng biểu thức chính quy /g
* Theo mặc định, phương thức replace() có phân biệt chữ hoa chữ thường. Để tìm kiếm kết quả mà không phân biệt hoa thường, hãy sử dụng biểu thức chính quy /i

Ví dụ: xem kỹ các ví dụ sau



*\* Sử dụng biểu thức chính quy thì không có dấu nháy kép bao bên ngoài chuỗi.*

*\* Các phương thức replace() không làm thay đổi chuỗi được gọi. Nó trả về một chuỗi mới.*

**Chuyển đổi sang chữ hoa và chữ thường:**

Để chuyển chuỗi thành chữ hoa ta dùng hàm toUpperCase() và chuyển thành chữ thường ta dùng hàm toLowerCase().

Ví dụ:



**Phương thức trim():**

Phương thức trim() dùng để loại bỏ khoảng trắng từ cả hai phía của một chuỗi.

Ví dụ:



*\* JS còn hỗ trợ rất nhiều phương thức để xử lý chuỗi, có thể tự tìm hiểu trên internet với từ khóa “string method in javascript” hoặc từ khóa thể hiện công dụng của phương thức “concatenate string javascript”, …*

### c. JS Numbers – Số trong JavaScript

**Kiểu number:**

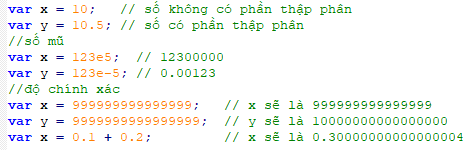
Trong JS chỉ hỗ trợ một kiểu số duy nhất là number, các số luôn luôn là 64 bit và lưu dưới dạng số dấu phẩy động kiểu float.

*\* JS không định nghĩa các loại số khác nhau như integers, short, long, … như các ngôn ngữ lập trình khác.*

Các đặc điểm về number trong JS:

* Số trong JS có thể được viết với có hoặc không có dấu chấm động (phần thập phân).
* Dùng số mũ để biểu diễn số quá lớn hoặc quá nhỏ.
* Số nguyên (không có dấu chấm hoặc ký hiệu mũ) độ chính xác đến 15 chữ số.
* Số thập phân tối đa là 17 chữ số nhưng độ chính xác của phần thập phân không phải lúc nào cũng được 100%

Ví dụ:



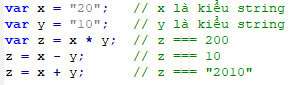
**Số và chuỗi:**

Trong các phép toán số, JS sẽ luôn cố gắng chuyển đổi chuỗi thành số để thực hiện phép toán đó.

Tuy nhiên với toán tử + JS dùng cho cả phép cộng và phép nối. Trình thông dịch JS hoạt động từ trái sang phải. Vì vậy cần đặc biệt lưu ý khi sử dụng toán tử + đối với các toán hạng có lẫn kiểu chuỗi và kiểu số. Trình tự các biến này có thể dẫn đến kết quả khác nhau với toán tử +

Ví dụ:

Với hai biến kiểu chuỗi, các toán tử: \*, /, - sẽ cho kết quả phép tính nhân, chia, trừ; với toán tử + lại cho kết quả của phép nối chuỗi



Với toán hạng có cả giá trị kiểu chuỗi và kiểu số



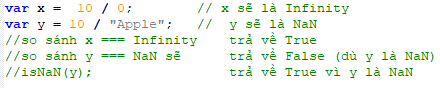
**Infinity và NaN:**

Infinity: là một số kiểu dữ liệu number, với giá trị của nó vượt mức lưu chữ cho phép của JS. (phép chia cho 0 cũng tạo ra Infinity)

NaN – Not a Number: cũng là một số kiểu dữ liệu number, nó chỉ ra rằng đây là một số không hợp lệ (thường là kết quả khi thực hiện phép toán số học với một chuỗi không phải số).

Tuy nhiên, Infinity có thể so sánh như bình thường còn NaN thì không. Để kiểm tra một biến nào đó có phải là NaN hay không thì chúng ta phải sử dụng hàm isNaN().

Ví dụ:



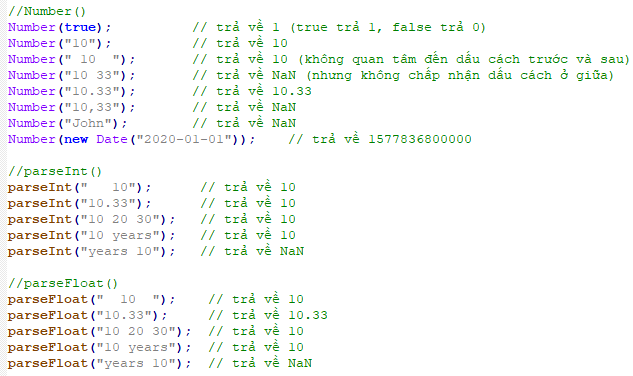
**Các phương thức với số trong JavaScipt:**

Một số phương thức thường dùng để xử lý với số trong JS như trong bảng sau:

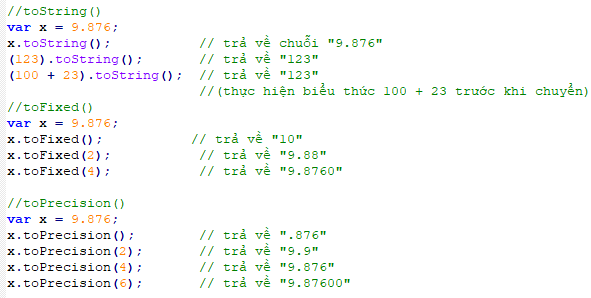
|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| Number() | Dùng để chuyển đổi biến hoặc giá trị nào đó (kiểu number, string, date, Boolean...) thành kiểu số.  Nếu không chuyển được sẽ trả về NaN |
| parseInt() | Phân tích cú pháp một chuỗi và trả về một số nguyên. Phương thức này có thể trả về số nguyên nếu chuỗi có các ký tự đầu tiên là các con số và chỉ trả về số đầu tiên tìm được.  Nếu không chuyển được sẽ trả về NaN |
| parseFloat() | Phân tích cú pháp một chuỗi và trả về một số định dạng float. Tương tự parseInt() nó chỉ trả ra số đầu tiên nếu trong chuỗi có nhiều số.  Nếu không chuyển được sẽ trả về NaN |
| toString() | Chuyển đổi kiểu number sang string.  Phương thức này có thể sử dụng trên cho số, chữ, biến, biểu thức… |
| toFixed(n) | Chuyển một số sang một chuỗi có n chữ số ở phần thập phân (làm tròn phần thập phân đến số thứ n trước khi chuyển đổi sang chuỗi)  Khi không khai báo n, mặc định n=0 |
| toPrecision(n) | Chuyển một số sang một chuỗi có n số (làm tròn phần thập phân trước khi chuyển đổi sang chuỗi)  Khi không khai báo n, mặc định là chiều dài ban đầu của số |

Ví dụ:

Về Number(), parseInt() và parseFloat()



Về toString(), toFixed() và toPrecision()



### d. JS Arrays – Mảng trong JavaScript

Mảng được sử dụng để lưu trữ nhiều giá trị trong một biến duy nhất. Mỗi giá trị là một phần tử của mảng, mỗi phần tử sẽ được đánh dấu một vị trí nhất định. Ví dụ, trong JS nếu mảng có 10 phần tử thì các phần tử sẽ được đánh dấu từ 0 - 9.

Phần tử của mảng có thể là number, string, object, hoặc các hàm, các mảng…

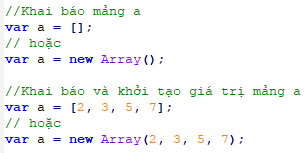
Trong cùng một mảng, các phần tử cũng có thể có kiểu khác nhau.

**Khai báo một mảng:**

Cách đơn giản và để tốc độ thực thi nhanh nhất, sử dụng cặp dấu ngoặc vuông để khai báo mảng.

Cũng có thể sử dụng từ khóa new để khai báo một mảng.

Ví dụ:



*\* Để đơn giản, dễ đọc và tốc độ thực thi, hãy sử dụng phương thức khai báo mảng bằng cặp dấu []*

**Truy cập các phần tử của một mảng:**

Để truy xuất đến phần tử của một mảng chúng ta dùng cú pháp **tenmang[vitri]**.

Ví dụ:



**Thay đổi giá trị của một phần tử:**

Sử dụng phép gán (toán tử =) để thay đổi giá trị của một phần tử

Ví dụ: gán giá trị của phần tử thứ 2 thành chuỗi “11”



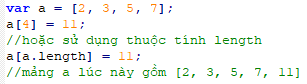
**Chiều dài của một mảng – length:**

Thuộc tính length trả về chiều dài của một mảng (số phần tử của mảng)



**Thêm phần tử vào một mảng:**

Có thể thêm phần tử mới bằng gán phần tử có vị trí (chỉ số) cao hơn chỉ số sẵn có trong mảng:



*\* Việc thêm các phần tử có chỉ số cao có thể tạo ra các lỗ hổng trong mảng. Các phần tử không được khai báo sẽ có giá trị undefined*

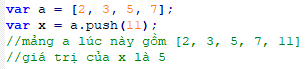
Ví dụ:



Sử dụng phương thức push() để thêm một phần tử vào mảng. Mặc định phần tử mới sẽ thêm vào cuối mảng.

Giá trị của phương thức push() trả về độ dài của mảng mới.

Ví dụ:

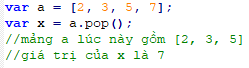


**Loại bỏ một phần tử của mảng:**

Sử dụng phương thức pop() dể loại bỏ phần tử cuối cùng của một mảng.

Giá trị của phương thức pop() trả về giá trị phần tử đã bị loại bỏ.

Ví dụ:



**Một số phương thức khác dùng cho mảng:**

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| shift() | Array.shift(): Dùng để loại bỏ phần tử đầu tiên của mảng Array. Chỉ số của các phần tử phía sau sẽ tương ứng giảm đi một bậc.  Giá trị của phương thức shift() trả về giá trị phần tử đã bị loại bỏ. |
| unshift() | Array.unshift(): Dùng để thêm một phần tử vào vị trí đầu tiên của mảng Array. Các phần tử phía sau sẽ tăng chỉ số tương ứng lên một bậc.  Giá trị của phương thức unshift () trả về độ dài của mảng mới. |
| splice() | Array.splice(x, n, value1, value2, ...): từ vị trí x, xóa n phần tử và thêm danh sách các phần tử với giá trị value1, value2, … vào mảng Array  Danh sách các phần tử thêm vào là không bắt buộc. Lúc này phương thức sẽ trở thành Array.splice(x, n) xóa n phần tử của mảng Array từ vị trí x. |
| concat() | New\_array = arr1.concat(arr2, arr3, …): Dùng để nối các mảng hiện có (arr1, arr2, arr3, …) và trả ra một mảng mới New\_array. |
| slice() | New\_array = Array.slice(x, y): Dùng để lấy các phần tử từ mảng Array bắt đầu từ vị trí x đến vị trí y và tạo ra mảng mới New\_array.  Nếu thiếu vị trí y, mặc định là lấy từ vị trí x đến cuối mảng Array. |
| sort() | Array.sort(): Dùng để sắp xếp các phần tử trong mảng Array theo thứ tự chữ cái alpha |
| reverse() | Array.reverse(): Dùng để đảo ngược các phần tử trong mảng Array. Vị trí đầu tiên trong mảng sẽ được chuyển xuống cuối, lần lượt đến phần tử ở vị trí cuối sẽ chuyển lên đầu mảng. |

*\* JS còn hỗ trợ rất nhiều phương thức để xử lý mảng, có thể tự tìm hiểu trên internet với từ khóa “array method in javascript”*

### e. JS Dates – Thời gian trong JavaScript

Date object – đối tượng thời gian trong JS chuyên dùng để xử lý ngày giờ trên Client. Mỗi Client sẽ có thời gian khác nhau phụ thuộc vào múi giờ của trình duyệt.

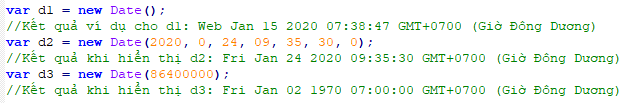
**Khai báo một đối tượng thời gian:**

Sử dụng hàm new Date() để tạo một đối tượng thời gian.

Có 4 cách để tạo:

* new Date() – tạo một đối tượng thời gian với ngày và giờ hiện tại
* new Date(year, month, day, hours, minutes, seconds, milliseconds) – tạo một đối tượng thời gian được chỉ định theo thứ tự là năm, tháng, ngày, giờ, phút, giây, mili giây. (JS tính tháng từ 0 đến 11; tháng 1 là 0; tháng 12 là 11)
* new Date(milliseconds) – tạo một đối tượng thời gian được chỉ định dưới dạng thời gian được tính bằng mili giây. Theo quy ước 07:00:00, 01/01/1970 là 0 mili giây; một ngày là 86 400 000 mili giây; từ đó cộng và trừ để ra được giá trị mili giây biểu thị cho ngày giờ cụ thể nào đó.
* new Date(date string) – tạo một đối tượng thời gian từ một chuỗi ngày tháng.

Ví dụ:



**Định dạng ngày trong JavaScript:**

Khi khởi tạo (gán) một đối tượng thời gian từ một chuỗi ngày tháng, cần có tuân theo quy tắc về định dạng chuỗi ngày tháng nhập vào.

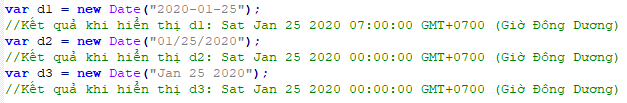
Có 3 loại định dạng đầu vào của Date JS như sau:

* ISO Date: Định dạng chuẩn của ISO 8601 là (YYYY-MM-DD) hoặc (YYYY-MM) hoặc (YYYY). (ví dụ “2020-01-25”)
* Short Date: dạng MM/DD/YYYY (ví dụ “01/25/2020”)
* Long Date: dạng MMM DD YYYY hoặc DD MMM YYYY (ví dụ “Jan 25 2020”)

Đầu ra của Date JS:

Không phụ thuộc vào định dạng đầu vào, JS mặc định sẽ xuất ngày tháng ở dạng chuỗi văn bản đầy đủ.

Ví dụ:



*\* Tùy thuộc vào múi giờ của trình duyệt có thể cho các kết quả khác nhau với cùng chuỗi thời gian đầu vào.*

**Phương thức lấy ngày trong JS:**

Có một số phương pháp có thể dùng để lấy thông tin từ một đối tượng thời gian như sau: (với biến date là một Date Object)

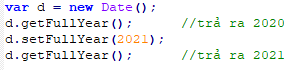
|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| date.getDay() | Phương thức trả về các ngày trong tuần của date là một số từ 0-6 bắt đầu từ Chủ Nhật (nghĩa là 0 là Chủ Nhật; 1 là Thứ Hai) |
| date.getDate() | Trả về ngày của date là một số từ 1-31 |
| date.getMonth() | Trả về tháng của date là một số từ 0-11 với 0 là Tháng 1 |
| date.getFullYear() | Trả về năm của date là một số có bốn chữ số (yyyy) |
| date.getHours() | Trả về giờ của date là một số từ 0-23 |
| date.getMinutes() | Trả về phút của date là một số từ 0-59 |
| date.getSeconds() | Trả về giây của date là một số từ 0-59 |
| date.getMilliseconds() | Trả về mili giây của date là một số từ 0-999 |
| date.getTime() | Trả về số mili giây của date kể từ 07:00:00 01/01/1970 |

**Phương thức thiết lập ngày JS:**

Phương thức thiết lập ngày cho phép đặt các giá trị năm, tháng, ngày, giờ, phút, giây, mili giây cho một đối tượng thời gian. (ví dụ với date là một Date Object) Ta có các phương thức thiết lập cho nó như sau:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| date.setDate(n) | Thiết lập ngày cho date với n có giá trị từ 1-31 |
| date.setMonth(n) | Thiết lập tháng cho date với n từ 0-11 |
| date.setFullYear(YYYY) | Thiết lập năm cho date là một số gồm 4 chữ số |
| date.setHours(n) | Thiết lập giờ cho date với n từ 0-23 |
| date.setMinutes(n) | Thiết lập phút cho date với n từ 0-59 |
| date.setSeconds(n) | Thiết lập giây cho date với n từ 0-59 |
| date.setMilliseconds(n) | Thiết lập mili giây cho date với n từ 0-999 |
| date.setTime(milliseconds) | Thiết lập thời gian cho date với milliseconds là ngày đã được chuyển thành số mili giây. |

Ví dụ: sử dụng phương thức lấy và thiết lập năm của một đối tượng thời gian



### f. JS Math – Đối tượng toán học trong JavaScript

Math Object – đối tượng toán học trong JS cung cấp các thuộc tính và phương thức cho phép thực hiện các nhiệm vụ toán học trên các con số.

Đối tượng Math là có sẵn trong JS, ta có thể sử dụng các thuộc tính và phương thức của nó mà không phải khởi tạo nó.

**Thuộc tính của Math:**

Một số thuộc tính thường dùng của Math được liệt kê trong bảng sau:

|  |  |
| --- | --- |
| **Thuộc tính** | **Miêu tả** |
| E | Hằng số E, và cơ số của thuật toán tự nhiên (ký hiệu: ln), xấp xỉ 2,718 |
| LN2 | Logarit tự nhiên của 2, xấp xỉ 0,693. |
| LN10 | Logarit tự nhiên của 10, xấp xỉ 2,302. |
| LOG2E | Logarit cơ số 2 của E, xấp xỉ 1,442. |
| LOG10E | Logarit cơ số 10 của E, xấp xỉ 0,434. |
| PI | Hệ số Pi, xấp xỉ 3,14159. |
| SQRT1\_2 | Căn bậc hai của 1/2; xấp xỉ 0,707. |
| SQRT2 | Căn bậc hai của 2, xấp xỉ 1,414. |

Ví dụ:



**Phương thức của Math:**

Một số phương thức thường dùng của Math được liệt kê trong bảng sau:

|  |  |
| --- | --- |
| **Phương thức** | **Miêu tả** |
| Math.abs(x) | Trả về trị tuyệt đối của x |
| Math.ceil(x) | Trả về giá trị của x được làm tròn lên số nguyên gần nhất của nó |
| Math.floor(x) | Trả về giá trị của x được làm tròn xuống số nguyên gần nhất của nó |
| Math.max(x1, x2…) | Trả về số lớn nhất trong danh sách số x1, x2, … |
| Math.min(x1, x2…) | Trả về số nhỏ nhất trong danh sách số x1, x2, … |
| Math.pow(a, n) | Trả về giá trị lũy thừa bậc n của cơ số a |
| Math.random() | Trả về một số ngẫu nhiên giữa hai số 0 (bao gồm) và 1 (không bao gồm). |
| Math.round(x) | Làm tròn số. Trả về giá trị của x đã làm tròn thành số integer gần nhất. |
| Math sqrt(x) | Trả về căn bậc hai của x |

Ví dụ:



*\* Còn nhiều phương thức khác của đối tượng Math. Có thể tìm hiểu trên internet với từ khóa “math object javascript”*

1. **JavaScript DOM**

**HTML DOM– HTML Document Object Model**

Mỗi trang web bên trong một cửa sổ trình duyệt có thể xem như là một đối tượng.

Khi một trang web được tải, trình duyệt sẽ tạo ra một mô hình các đối tượng tài liệu - DOM của trang đó.

HTML DOM là một mô hình các đối tượng tiêu chuẩn cho các tài liệu HTML. Nó định nghĩa:

* Các phần tử HTML dưới dạng đối tượng
* Các thuộc tính của tất cả các phần tử HTML
* Các phương pháp để truy cập tất cả các phần tử HTML
* Các sự kiện cho tất cả các phần tử HTML

Sử dụng HTML DOM, JavaScript có thể lấy, thay đổi, thêm hoặc xóa các phần tử, nội dung phần tử, giá trị thuộc tính các phần tử hoặc sự kiện cho các phần tử HTML.

**DOM Document:**

Đối tượng Document đại diện cho trang web, là chủ sở hữu của tất cả các đối tượng khác trong trang web đó.

Khi cần truy cập bất cứ phần tử nào trong trang HTML, trước hết phải bắt đầu bằng việc truy cập vào đối tượng Document.

Sử dụng các phương thức của đối tượng Document để lấy, thêm, sửa, xóa các phần tử trong trang HTML.

Ví dụ: tìm một phần tử HTML theo id, sử dụng cú pháp: document.getElementById(*id*)

**Tìm một phần tử HTML**

Để tìm một phần tử HTML trong một trang HTML, ta có một số cách thông thường như sau:

* Tìm các phần tử HTML theo id
* Tìm các phần tử HTML theo tên thẻ
* Tìm các phần tử HTML theo class

*\* Có các cách khác để tìm một phần tử như tìm theo bộ chọn CSS (Selector CSS) hoặc bằng các bộ sưu tập đối tượng HTML.*

Cú pháp tìm một phần tử là dùng các phương thức của đối tượng Document, như bảng sau:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| document.getElementById(id) | Tìm các phần tử HTML theo id |
| document.getElementsByTagName(tagName) | Tìm các phần tử HTML theo tên thẻ |
| document.getElementsByClassName(className) | Tìm các phần tử HTML theo tên class |

**Thay đổi phần tử HTML**

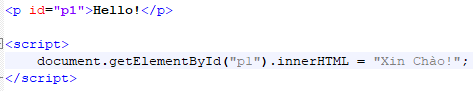
HTML DOM cho phép JS thay đổi nội dung, giá trị các thuộc tính của một phần tử HTML.

Lấy và thay đổi nội dung của một phần tử HTML bằng cách sử dụng thuộc tính innerHTML.

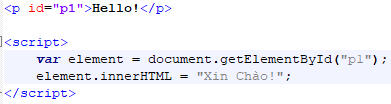
Cú pháp: document.getElementById(*id*).innerHTML = *new value*

Ví dụ:

Thay đổi nội dung của một phần tử có id là p1



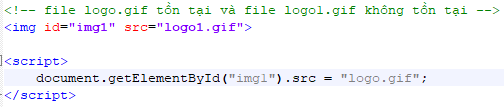
Hoặc lấy rồi thay đổi nội dung của p1



Kết quả: trang HTML hiển thị Xin Chào! thay vì Hello! vì nội dung phần tử <p> đã bị thay đổi.

Thay đổi giá trị của một thuộc tính: sử dụng cú pháp document.getElementById(id).attribute = new value

Ví dụ: thay đổi giá trị thuộc tính src của một phần tử <img> có id là img1



Kết quả: hình ảnh logo được hiển thị trên trang web

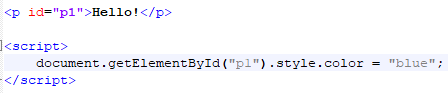


**Thay đổi CSS**

HTML DOM cho phép JS thay đổi kiểu của các phần tử HTML.

Để thay đổi, ta sử dụng cú pháp: document.getElementById(*id*).style.*property* = *new style*

Ví dụ: thay đổi màu chữ của phần tử <p> có id p1 thành màu xanh



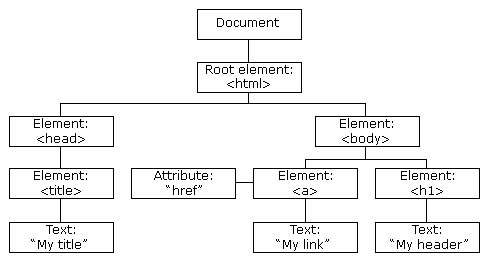
Kết quả:



**DOM Nodes**

Theo tiêu chuẩn W3C HTML DOM, mọi thứ trong tài liệu HTML là một node:

* Toàn bộ tài liệu là một nút tài liệu (document node)
* Mỗi phần tử HTML là một nút phần tử (element node)
* Văn bản bên trong các phần tử HTML là các nút văn bản (text node)
* Mỗi thuộc tính HTML là một nút thuộc tính (attribute node)
* Tất cả các comment là nút bình luận (comment node)



Với HTML DOM, tất cả các node trong cây node đều có thể truy cập bằng JavaScript, từ đó có thể thêm, sửa hoặc xóa các node.

**Mối quan hệ giữa các node:** sử dụng thuật ngữ cha, con, anh chị em để biểu thị mối quan hệ thứ bậc giữa các node (tương tự mối quan hệ giữa các phần tử HTML)

* Trong cây node, node trên cùng được gọi là gốc (roof node)
* Mọi node đều có chính xác một node cha, ngoại trừ node gốc (không có node cha)
* Một node có thể có một số node con
* Anh chị em là các node có cùng cha mẹ

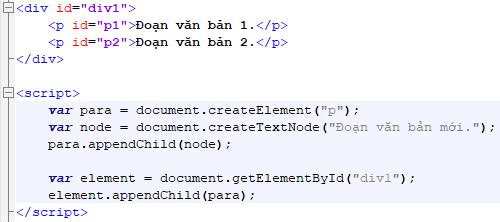
**Tạo phần tử HTML mới**

Để thêm một phần tử mới vào HTML DOM, trước tiên phải tạo phần tử (tạo một node), sau đó nối phần tử đó vào một phần tử hiện có.

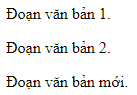
Thông thường, các bước tạo một phần tử mới cơ bản như sau:

* Dùng phương thức createElement () để tạo một phần tử mới
* Dùng phương thức createTextNode() để tạo một node văn bản và nối node văn bản vào phần tử
* Dùng phương thức appendChild() để thêm phần tử mới thành phần tử con cuối cùng của phần tử cha nào đó hoặc phương thức insertBefore() để thêm phần tử mới làm con một phần tử và ví trí đứng trước phần tử nào đó.

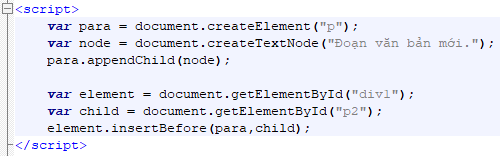
Ví dụ: tạo phần tử <p> bằng JS với nội dung “Đoạn văn bản mới.” và xác định nó là phần tử con phần tử <div>



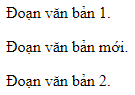
Kết quả: phần tử <p> mới sẽ được thêm vào cuối cùng trong danh sách các phần tử con của phần tử cha <div>



Để phần tử <p> mới đứng trước phần tử <p> có id là p2:



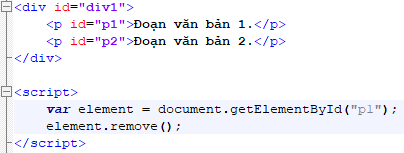
Kết quả:



**Xóa các phần tử HTML hiện có**

Dùng phương thức remove() để xóa một phần tử HTML hiện có.

Ví dụ: xóa phần tử <p> có id là p1



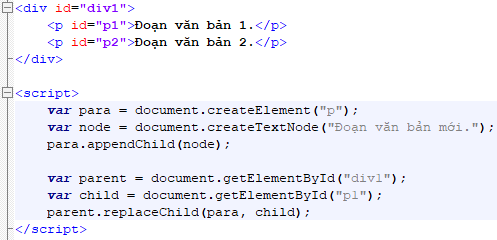
Kết quả:



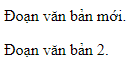
**Thay thế các phần tử HTML**

Dùng phương thức replaceChild() để thay thế một phần tử HTML là con của phần tử nào đó.

Ví dụ: tạo một phần tử <p> mới và thay thế phần tử <p> có id là p1



Kết quả:



1. **JavaScript BOM**

**BOM – Browser Object Model**

BOM là chữ viết tắt của Browser Object Model, hay còn gọi là các đối tượng liên quan đến trình duyệt browser. Mỗi browser sẽ có những đối tượng khác nhau nên nó không có một chuẩn chung nào cả, tuy nhiên các trình duyệt hiện tại (gần như) đã triển khác các phương thức và thuộc tính giống nhau cho phép JS tương tác với nó, nên nó thường được gọi là phương thức và thuộc tính của BOM.

**Đối tượng Window**

Đối tượng Window được hỗ trợ hầu hết các trình duyệt và mỗi tab trình duyệt sẽ có một đối tượng Window riêng và không liên quan tới nhau.

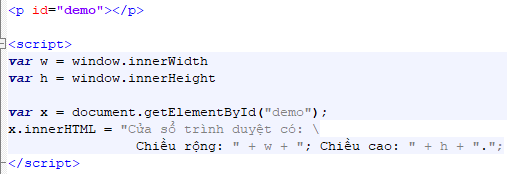
Tất cả các đối tượng, hàm và biến JS toàn cục đều là thuộc tính của đối tượng Window. Kể các đối tượng Document cũng là một thuộc tính của đối tượng Window.

**Kích thước của cửa sổ trình duyệt**

Đối tượng Window có hai thuộc tính chiều rộng, chiều cao có thể được sử dụng để xác định kích thước cửa sổ của trình duyệt.

* window.innerHeight - chiều cao bên trong của cửa sổ trình duyệt (tính bằng pixel)
* window.innerWidth - chiều rộng bên trong của cửa sổ trình duyệt (tính bằng pixel)

Ví dụ:



Kết quả:



*\* Cửa sổ trình duyệt (khung nhìn trình duyệt) không bao gồm thanh công cụ và thanh cuộn.*

*\* Một số trình duyệt cũ có thể không hỗ trợ thuộc tính innerHeight, innerWidth của đối tượng Window.*

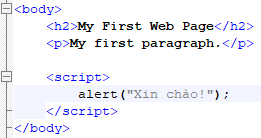
**Phương thức của đối tượng Window**

Một số phương thức thường dùng của đối tượng Window được liệt kê trong bảng sau:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| Window.open(url, name, options) | Dùng để mở một cửa sổ mới  Trong đó url là đường dẫn website muốn mở, name là tên của cửa sổ (có hoặc không), options là một chuỗi các thông số cách nhau bởi dấu phẩy như width, height… tương ứng cho chiều rộng, chiều cao… của cửa sổ mở ra. |
| windowObj.close() | Dùng để đóng cửa sổ windowObj được tạo ra bởi lệnh window.open() |
| Window.alert() | Dùng để hiển thị một hộp cảnh báo có chứa thông báo với nút ok. |
| Window.confirm() | Dùng để hiển thị một hộp thoại xác nhận có chứa thông báo với nút ok và hủy. |
| Window.prompt() | Dùng để hiển thị một hộp thoại để nhận đầu vào từ người dùng. |

*\* Với các phương thức alert(), confirm(), prompt() từ khóa window có thể bỏ qua. (dùng window.alert() cũng tương đương với chỉ dùng cú pháp alert())*

Ví dụ:



Tự thử và xem kết quả.

1. **JavaScript Even**

**Sự kiện trong JavaScript**

Sự kiện HTML là một cái gì đó xảy ra với các phần tử HTML.

Sự kiện HTML có thể là thứ trình duyệt làm hoặc người dùng làm, như tải một trang HTML, nhập dữ liệu vào ô input, click vào một button…

Khi JS được sử dụng trong trang HTML, nó có thể bắt được các sự kiện này và phản ứng lại bằng việc thực hiện những hành động nào đó.

**Các sự kiện HTML thông dụng**

| **STT** | **Sự kiện** | **Mô tả** |
| --- | --- | --- |
| 1 | onclick | Xảy ra khi click vào phần tử HTML |
| 2 | ondbclick | Xảy ra khi double click vào phần tử HTML |
| 3 | onchange | Xảy ra khi giá trị (value) của phần tử HTML đổi. Thường dùng trong các đối phần tử input |
| 4 | onmouseover | Xảy ra khi con trỏ chuột bắt đầu đi vào phần tử HTML |
| 5 | onmouseout | Xảy ra khi con trỏ chuột bắt đầu rời khỏi phần tử HTML |
| 6 | onmousemove | Xảy ra khi con chuột di chuyển bên trong phần tử HTML |
| 7 | onkeydown | Xảy ra khi gõ một phím bất kì vào ô input |
| 8 | onload | Sảy ra khi phần tử HTML được tải xong |
| 9 | onkeyup | Xảy ra khi nhả phím ra sẽ được kích hoạt |
| 10 | onkeypress | Xảy ra khi nhấn một phím vào ô input |
| 11 | onblur | Xảy ra khi con trỏ chuột rời khỏi ô input |

**Bắt sự kiện bằng JavaScipt**

Trước khi bắt sự kiện thì ta cần phải xác định rõ:

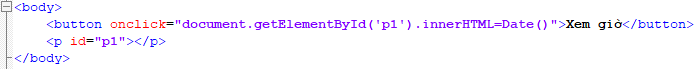
* Phần tử dùng để xảy ra sự kiện
* Sự kiện sẽ xảy ra
* Đoạn mã sẽ được thực thi khi sự kiện xảy ra

Bắt sự kiện bằng cách thêm vào mã HTML: HTML có các thuộc tính xử lý sự kiện với giá trị là các mã hoặc hàm JavaScript.

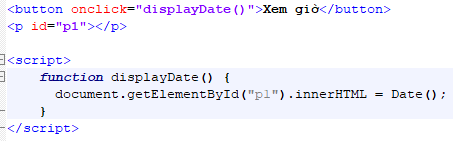
Cú pháp: <element event="some JavaScript">

Ví dụ: thay đổi nội dung của phần tử <p> có id là “p1” thành ngày giờ hiện tại khi click vào button

Viết mã JS vào sau thuộc tính sự kiện onclick



Hoặc viết hàm và gọi hàm

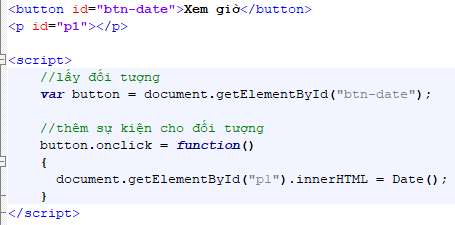


Kết quả: thử và xem kết quả khi click vào nút “Xem giờ”

Dùng DOM để bắt sự kiện trong mã JS. Cú pháp:



Ví dụ: như ví dụ trên có thể viết thành như sau



1. **Regular Expression và Form Validation trong JavaScript**

### a. Regular Expression – Biểu thức chính quy

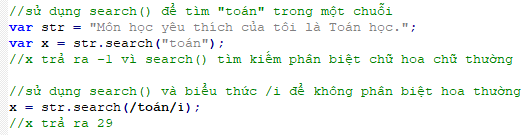
Regular Expression hay gọi cách khác là biểu thức chính quy, là một chuỗi các ký tự tạo thành một mẫu tìm kiếm. Khi tìm kiếm dữ liệu trong một văn bản, ta có thể sử dụng mẫu tìm kiếm này để mô tả những gì mình đang tìm kiếm.

Trong JS, biểu thức chính quy thường được sử dụng với các phương thức tìm kiếm và thay thế chuỗi, như search(), match() và replace()

Ở bài học trong phần JS String, phương thức replace(), ta cũng đã dùng biểu thức chính quy để có thể thay thế tất cả các kết quả phù hợp thay vì dừng lại sau kết quả phù hợp đầu tiên và để tìm kiếm, thay thế mà không phân biệt chữ hoa chữ thường.

Đối với các phương thức tìm kiếm khác cũng tương tự.

Ví dụ:



Trong đó /toán/i là một biểu thức chính quy với /toán/ là mẫu và i là cờ.

**Đối tượng RegExp**

Trong JS thì biểu thức chính quy được coi như một đối tượng, khi khai báo một biểu thức chính quy chính là tạo một đối tượng tương ứng. Đối tượng này gọi là đối tượng RegExp, nó có các thuộc tính và phương thức được xác định trước.

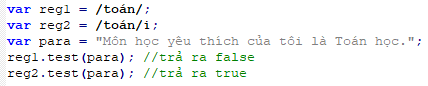
Khai báo một đối tượng RegExp: khai báo một biến là một đối tượng RegExp như bình thường:



**Phương thức test()**

Phương thức test() là một phương thức của đối tượng RegExp, nó tìm kiếm một chuỗi theo một mẫu biểu thức chính quy và trả về True hoặc False, tùy thuộc vào kết quả.

Ví dụ: tìm kiếm một chuỗi ký tự “toán” trong một chuỗi có sẵn



**Mẫu biểu thức chính quy**

Một mẫu biểu thức chính quy là một tập các kí tự thường, như /abc/ hay một tập hợp các kí tự thường và kí tự đặc biệt như /ab\*c/ hoặc /^[0-9]+$/

Khi tìm kiếm, ta thường không biết được chính xác tất cả các ký tự mà chỉ xác định được định dạng, quy tắc của chuỗi cần tìm kiếm, như tìm kiếm các chuỗi có định dạng email, chuỗi có định dạng ngày tháng năm, chuỗi là các URL đầy đủ …

Sử dụng kết hợp các kí tự thường, kí tự đặc biệt và cờ trong biểu thức chính quy ta có thể tạo được các mẫu tìm kiếm cho những mục đích tìm kiếm nâng cao như ở trên.

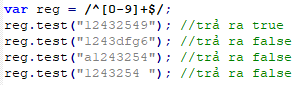
Công cụ sửa đổi biểu thức chính quy: (hay còn gọi là cờ) được sử dụng để thực hiện các tìm kiếm toàn cục không phân biệt chữ hoa chữ thường.

|  |  |
| --- | --- |
| **Cờ** | **Mô tả** |
| g | Tìm kiếm toàn cục (tìm tất cả các kết quả phù hợp thay vì dừng lại sau kết quả phù hợp đầu tiên) |
| i | Tìm kiếm không phân biệt hoa thường. |
| m | Tìm kiếm đa dòng. |

Ký tự đặc biệt: Một số các kí tự đặc biệt trong biểu thức chính quy và ý nghĩa của chúng được liệt kê trong bảng dưới đây.

|  |  |  |
| --- | --- | --- |
| Ký tự (ký hiệu) | Ý nghĩa | Ví dụ |
| ^ | So khớp các ký tự liền kề ở đầu tiên của một chuỗi. | /^ab/.test("abc"); //trả ra true  /^ab/.test(" ab"); //trả ra false  /^ab/.test("cab "); //trả ra false |
| $ | So khớp các ký tự liền kề ở cuối cùng của một chuỗi. | /ab$/.test("cab"); //trả ra true  /ab$/.test("abc"); //trả ra false  /ab$/.test("cab "); //trả ra false |
| \* | So khớp cho phép kí tự liền trước nó xuất hiện lặp lại 0 lần hoặc nhiều lần. | /ab\*/.test("abbbcc"); //trả ra true  /ab\*/.test("accccc"); //trả ra true  /ab\*/.test("bbbccc"); //trả ra false |
| + | Cho phép kí tự liền trước nó lặp lại 1 lần hoặc nhiều lần. | /ab+/.test("abbbcc"); //trả ra true  /ab+/.test("accccc"); //trả ra false  /ab+/.test("bbbccc"); //trả ra false |
| . | Khớp với mọi ký tự đơn (chữ cái, số hoặc ký hiệu) | /.a/.test("a"); //trả ra false  /.a/.test("xa"); //trả ra true  /.a/.test("?a"); //trả ra true |
| | | Tạo kiểu khớp OR (hoặc)  Không dùng ở cuối mỗi biểu thức | /ab|cd/.test("aaabbb"); //trả ra true  /ab|cd/.test("acdaaa"); //trả ra true  /ab|cd/.test("aaaddd"); //trả ra false |
| {n} | Khớp khi ký tự đứng trước nó xuất hiện ít nhất n lần  n là một số nguyên dương | /a{2}/.test("aaacbbb"); //trả ra true  /a{2}/.test("aaccbbb"); //trả ra true  /a{2}/.test("acccbbb"); //trả ra false |
| [xyz] | Khớp khi một trong các ký tự nằm trong dấu ngoặc vuông có trong chuỗi tìm kiếm ở bất kỳ vị trí nào. | /[12]/.test("1abcd"); //trả ra true  /[12]/.test("2abcd"); //trả ra true  /[12]/.test("12abc"); //trả ra true  /[12]/.test("3abcd"); //trả ra false |
| [a-z] | Khớp khi một ký tự nằm trong khoảng từ a đến z. | /[0-9]/.test("1234abc"); //trả ra true  /[0-9]/.test("abcdabc"); //trả ra false |
| [^xyz] | Khi ký tự ^ đứng đầu tiên trong dấu ngoặc vuông thì nó mang nghĩa phủ định mẫu này. | /[^0-9]/.test("1234abc"); //trả ra true  /[^0-9]/.test("abcdabc"); //trả ra true  /[^0-9]/.test("1234567"); //trả ra true |
| \b | Khớp với toàn bộ ký tự đứng trước nó (Khớp khi ký tự trong chuỗi không được theo sau (đứng trước) bởi một ký tự khác)  **Chú ý:** trong Javascript định nghĩa một [lớp kí tự](http://www.ecma-international.org/ecma-262/5.1/#sec-15.10.2.6) là những kí tự thường. Bất kỳ kí tự nào không thuộc lớp kí tự bị xem như một kí tự ngắt. Lớp kí tự bao gồm bộ kí tự La-tinh cả hoa và thường, số thập phân và kí tự gạch dưới. Kí tự có dấu, như "é" hay "ũ", bị đối xử như một kí tự ngắt. | /ello\b/.test("Hello World"); //trả ra true  /ello\b/.test("HelloWorld"); //trả ra false |
| \B | Khớp khi ký tự trước và sau của ký tự tìm kiếm là cùng kiểu: hoặc cả hai là kí tự hoặc cả hai không phải là ký tự | /ello\B/.test("HelloWorld"); //trả ra true  /ello\B/.test("H ello World"); //trả ra true  /ello\B/.test("Hello World"); //trả ra false |
| \d | Khớp với một kí tự số. (tương đương với mẫu [0-9] | /\d/.test("1234abc"); //trả ra true  /\d/.test("abcdabc"); //trả ra false |
| \D | Khớp với một kí tự không phải là kí tự số  (tương đương với mẫu [^0-9]) | /\D/.test("1234abc"); //trả ra true  /\D/.test("abcdabc"); //trả ra true  /\D/.test("1234567"); //trả ra true |
| \w | Khớp với tất cả các ký tự là chữ, số và dấu gạch dưới \_  (tương đương với [a-zA-Z0-9\_]) | /\w/.test("1234abc\_"); //trả ra true  /\w/.test("@$%^&\*()"); //trả ra false |
| \W | Khớp với tất cả kí tự không phải là chữ.  (tương đương với [^A-Za-z0-9\_]) | /\W/.test("1234abc\_"); //trả ra false  /\W/.test("@$%^&\*()"); //trả ra true |
| \s | Khớp với một kí tự khoảng trắng (bao gồm trắng tạo ra từ phím cách, tab, khoảng trắng khi phân trang, phân dòng) | /\s/.test("Hello World"); //trả ra true  /\s/.test("HelloWorld"); //trả ra false |
| \S | Khớp với một kí tự không phải khoảng trắng | /\S/.test("Hello World"); //trả ra true  /\S/.test("HelloWorld"); //trả ra true  /\S/.test(" "); //trả ra false |

Ví dụ: mẫu biểu thức chính quy kiểm tra một chuỗi có là số không



### b. JavaScript Form Validation

**Xác thực dữ liệu - Data Validate**

Xác thực dữ liệu là quá trình đảm bảo rằng thông tin đầu vào của người dùng là sạch sẽ, chính xác và hữu ích.

Thông thường, mục đích của xác thực dữ liệu là để đảm bảo người dùng nhập chính xác.

Các nhiệm vụ xác thực điển hình là:

* người dùng đã điền vào tất cả các trường bắt buộc chưa?
* giới hạn về độ dài của dữ liệu
* người dùng đã nhập dữ liệu hợp lệ chưa (đúng định dạng ngày tháng, email, url, số điện thoại…)?
* …

Xác thực dữ liệu rất quan trọng vì đầu vào không hợp lệ có thể phá vỡ logic ứng dụng, khiến trang web bị lỗi hoặc tiết lộ dữ liệu được bảo vệ.

Dữ liệu có thể được xác định bằng nhiều phương pháp khác nhau và được triển khai theo nhiều cách khác nhau.

* Server-side validation: Xác thực phía server được thực hiện bởi máy chủ web, sau khi đầu vào đã được gửi đến server.
* Client-side validation: Xác thực phía client được thực hiện bởi trình duyệt web, trước khi đầu vào được gửi đến web server.

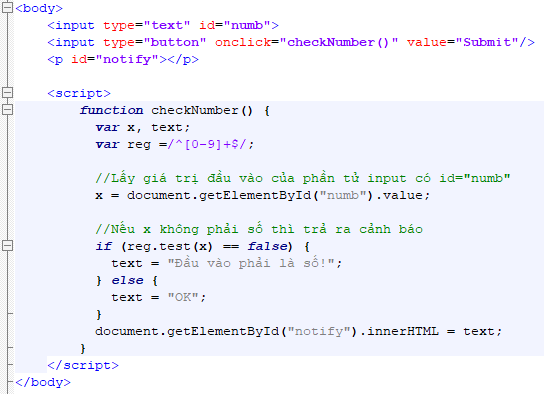
Xác thực bằng JavaScript thường là xác thực phía client, việc này giúp xác thực được nhanh chóng, các phản hồi tới khách truy cập nhanh hơn, làm tăng khả năng tương tác với người dùng (người dùng sẽ không cảm thấy khó chịu vì khi họ click submit thì sẽ được thông báo ngay chứ không cần phải refresh lại trang).

**Validate dữ liệu bằng JavaScript**

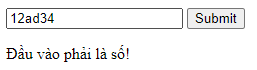
Sử dụng hàm khi bắt sự kiện trong from để validate dữ liệu, cảnh báo một thông báo và return false để ngăn dữ liệu gửi về server khi dữ liệu không đúng chuẩn hoặc return true để dữ liệu được gửi.

Sử dụng biểu thức chính quy để kiểm tra định dạng dữ liệu.

Ví dụ: sử dụng JS để xác thực dữ liệu nhập vào của ô input chỉ được chứa các chữ số, nếu không phải số thì trả ra cảnh báo, nếu là số thì trả ra OK.



Kết quả: sau khi nhập dữ liệu và ấn Submit sẽ xem được cảnh báo “Đầu vào phải là số!” hoặc “OK”



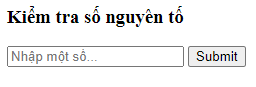
1. **Bài tập**

**Ex1:** Viết form kiểm tra số nhập vào có phải số nguyên tố hay không.

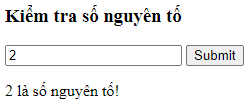
Yêu cầu:

* Validate đầu vào phải là một số tự nhiên
* Nếu số n nhập vào là số nguyên tố thì trả ra thông báo “n là số nguyên tố!”, nếu không thì trả ra “n không phải số nguyên tố!”.

Form ban đầu:



Form khi nhập số và ấn Submit:

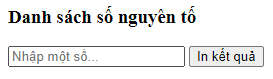


**Ex2:** Viết form nhập vào một số và in ra tất cả các số nguyên tố nhỏ hơn giá trị của số nhập vào.

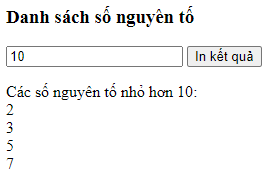
Yêu cầu:

* Validate đầu vào phải là một số tự nhiên
* Nếu không có số nguyên tố nào nhỏ hơn số n đã nhập thì trả ra thông báo “Không có số nguyên tố nào nhỏ hơn n.”

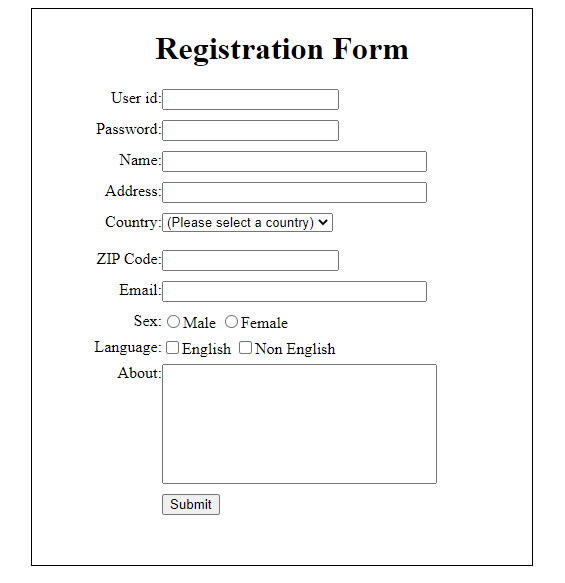
Form ban đầu



Form khi nhập số và ấn nút In kết quả



**Ex3:** Tạo form đăng ký như hình dưới đây

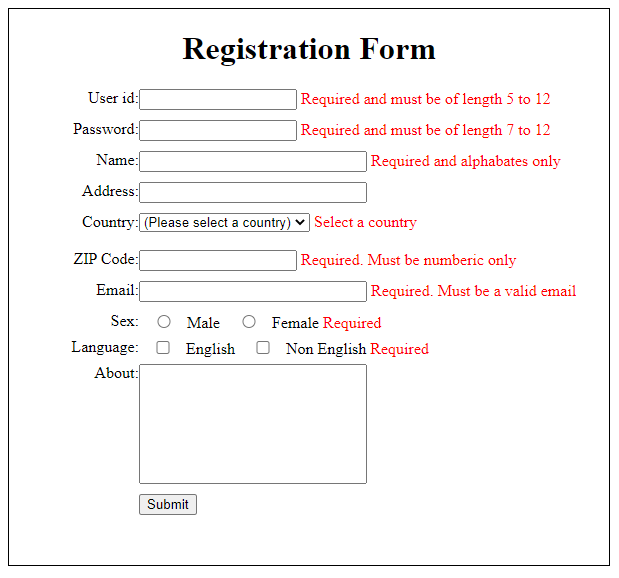


Yêu cầu: Sử dụng Javascript để Validate thông tin nhập vào như sau:

* Các trường bắt buộc nhập là: user id, password, name, zip code, email; các trường: contry, sex, language bắt buộc phải chọn
* Trường user id: độ dài chỉ được nằm trong khoảng từ 5 đến 12 ký tự
* Trường password: độ dài chỉ được nằm trong khoảng từ 7 đến 12 ký tự
* Trường name: chỉ được nhập vào kí tự là chữ trong bảng chữ cái
* Trường zip code: chỉ được nhập vào kí tự là chữ số
* Trường email: nội dung được nhập vào phải có định dạng email

Khi nhấn Submit sẽ trả ra thông báo lỗi tương ứng, thông báo lỗi hiện thị bên dưới hoặc bên cạnh trường input bị lỗi và có màu đỏ.

Ví dụ: thông báo lỗi đầy đủ như sau



# CHAPTER IV. GIỚI THIỆU VỀ ASP.NET WEBFORM

1. **Giới thiệu về Ứng dụng Web – WebApp**

Phần mềm ứng dụng (Application software, viết tắt là app) là một loại chương trình có khả năng làm cho máy tính thực hiện trực tiếp một công việc nào đó của người dùng.

Ứng dụng web (webapp) là một phần mềm ứng dụng mà người dùng có thể tiếp cận thông qua mạng như Internet hay Intranet.

Nói dễ hiểu, webapp là những ứng dụng chạy trên web. Ứng dụng web phổ biến nhờ vào sự có mặt bất cứ nơi đâu của một chương trình mà không cần cài đặt phần mềm trên tất cả các máy của người dùng. Các ứng dụng web thường được dùng như webmail (Gmail, Yahoo…), webapp quản lý bán hàng trực tuyến, webapp quản lý nhân lực, webapp quản lý siêu thị …

Cách mà một webapp hoạt động:

* Người dùng kích hoạt yêu cầu tới web server qua Internet thông qua trình duyệt web hoặc giao diện người dùng của ứng dụng.
* Web server chuyển tiếp yêu cầu này đến web application server (máy chủ ứng dụng web).
* Web application server thực hiện nhiệm vụ được yêu cầu (truy cập cơ sở dữ liệu, xử lý dữ liệu) sau đó gửi kết quả lại cho web server.
* Web server phản hồi lại cho người dùng, hiển thị trên màn hình ứng dụng của người dùng.

1. **Giới thiệu về ASP.NET và ASP.NET WebForm**

ASP.NET là một nền tảng ứng dụng web (web application framework) được phát triển và cung cấp bởi Microsoft, cho phép những người lập trình tạo ra những trang web động, những ứng dụng web.

ASP.NET là kỹ thuật lập trình và phát triển ứng dụng web ở phía máy chủ (lập trình phía server), các mã lệnh trong trang ASP sẽ được biên dịch và thi hành tại server và trả ra kết quả, kết quả sẽ được tự động chuyển sang HTML/CSS để trả về cho Client.

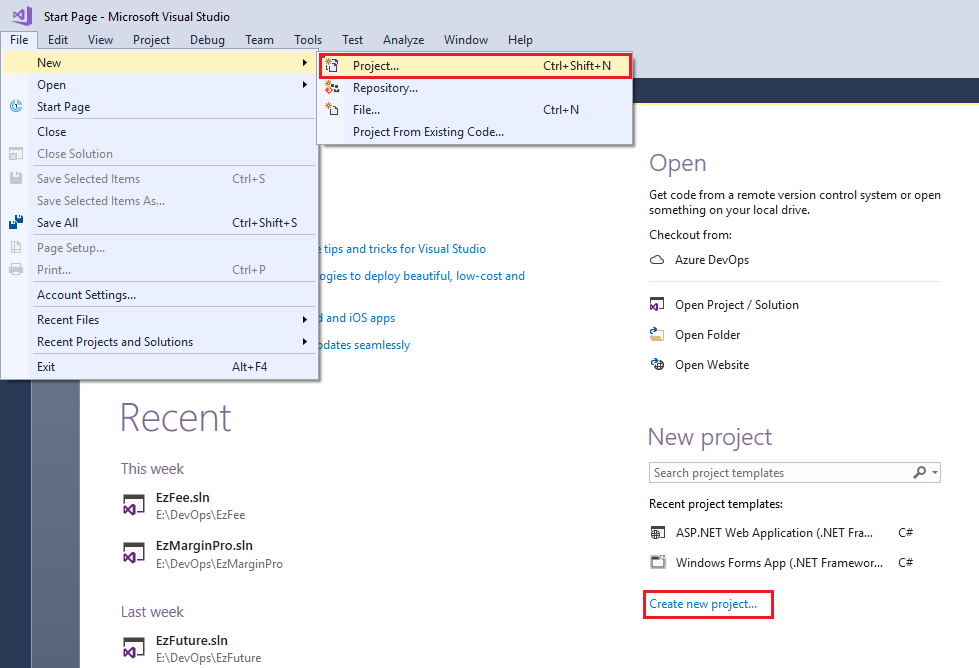
ASP.Net cung cấp các mô hình lập trình giúp cho việc xây dựng ứng dụng web như Asp.net Web Form, Asp.net MVC, Asp.net Web Pages… Mỗi mô hình đều có cấu trúc ứng dụng riêng biệt và có ưu nhược điểm riêng. Tùy thuộc vào tình huống, quy mô và nhu cầu của dự án mà lập trình viên có thể chọn và sử dụng mô hình phù hợp.

ASP.Net Web Form là mô hình lập trình hướng sự kiện và tăng hiệu suất bằng cách sử dụng các controls. ASP.Net cung cấp nhiều các server controls có đóng gói HTML, JavaScript và CSS bên trong giúp thiết kế ứng dụng dễ dàng và nhanh chóng bằng cách kéo thả các control này. ASP.Net WebForm còn cung cấp tính năng data binding nên có thể dễ dàng phát triển các ứng dụng nặng về việc truy xuất và hiển thị dữ liệu. Vì vậy, tuy Web Form là hình thức khá cổ điển và có lịch sử lâu đời, nó vẫn được nhiều lập trình viên áp dụng.

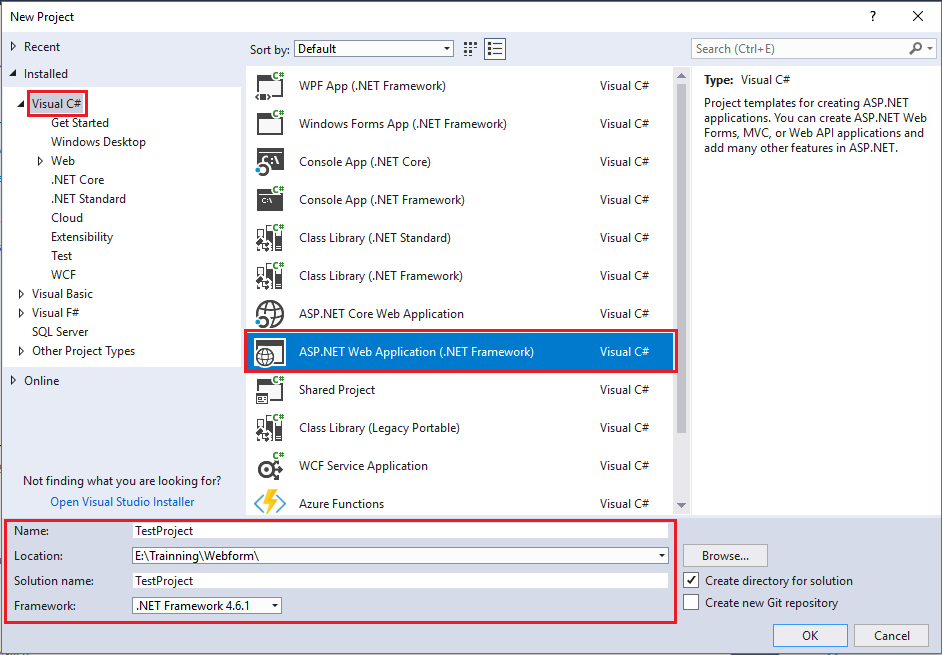
1. **Tạo Project ASP.NET và cấu trúc của một project ASP.NET**

**Tạo một dự án: theo các bước sau**

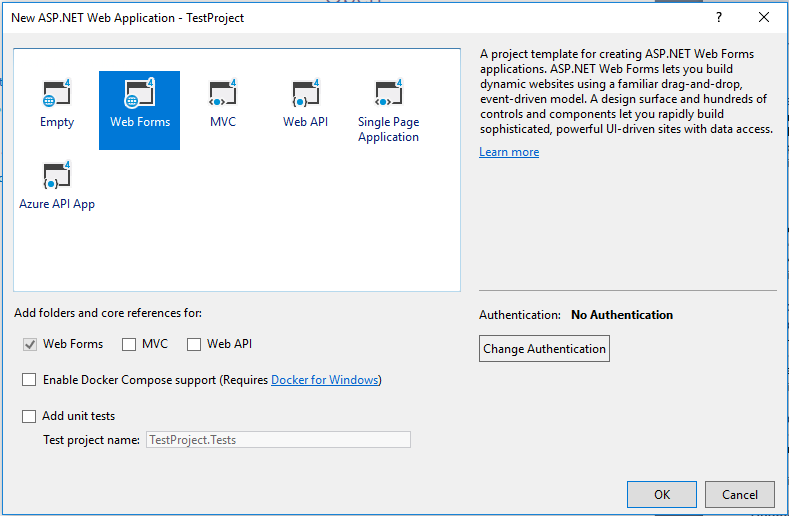
Bước 1: Mở chương trình Visual Studio (trong ví dụ là phiên bản 2017), chọn File -> New -> Project hoặc click vào Create new project… như hình dưới



Bước 2: Trong cửa sổ New Project, chọn Installed -> Visual C# -> ASP.NET Web Application, đặt tên dự án và chọn nơi lưu trữ, sau đó nhấn OK.

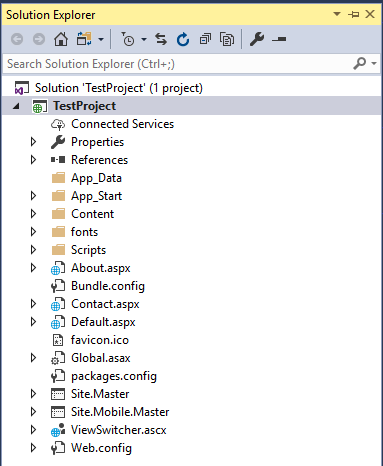


Bước 3: Tiếp đến là lựa chọn Template cho dự án Web, ở bước này chọn Web Forms, sau đó nhấn OK là dự án được khởi tạo.



**Cấu trúc thư mục thông thường của một dự án ASP.NET**

Ở cột bên phải màn hình, click để mở cửa sổ Solution Explorer (hoặc từ menu View -> chọn Solution Explorer hoặc nhấn tổ hợp phím Ctrl + Alt + L). Đây là cửa sổ quản lý các tài nguyên có trong ứng dụng.



Khi khởi tạo một dự án, ta sẽ có được các thư mục và tập tin được tạo sẵn như trên. Trong đó:

* References: mục chứa các thư viện DLL của dự án
* App\_Data: chứa cơ sở dữ liệu dạng tập tin (.mdf). Nếu sử dụng cơ sở dữ liệu SQL Server hoặc Oracle thì không cần lưu cơ sở dữ liệu ở đây.
* App\_Start: Lưu trữ cấu hình khi dự án được khởi chạy, gồm các tập tin như BundleConfig.cs (nhúng các thư viện JavaScript khi dự án Web chạy), RouteConfig.cs (chứa cấu hình đường dẫn) và StartUp.Auth.cs (chứng thực cho Web như Login …)
* Content: nội dung phong cách Web như bootstrap.css, …
* fonts: chứa kiểu chữ
* Scripts: chứa các tập tin JavaScript
* Global.asax: dùng để khai báo và khởi tạo giá trị cho các biến Session, Application. Rất quan trọng đối với các Website sử dụng Session như đăng nhập, đăng ký, phân quyền, chứng thực, đếm số người online, …
* Web.config: tập tin chứa toàn bộ cấu hình website, như chuỗi kết nối tới cơ sở dữ liệu… Đây là một tập tin rất quan trọng và bắt buộc phải có.
* Site.Master và Site.Mobile.Master: trang Master (định nghĩa bố cục) dùng làm vỏ giao diện cho các trang web khác.
* Các trang Web khi chạy sẽ có đuôi là .aspx, ví dụ About.aspx, Contact.aspx, Default.aspx (trang chủ). Mỗi trang Web của ứng dụng được gọi là một Web Form, thường gồm 2 file đi kèm với nhau: file .aspx chứa mã code HTML, CSS định hình giao diện trang web và file .aspx.cs chứa mã code behind (C#, VB, …) viết các phương thức xử lý sự kiện của trang.

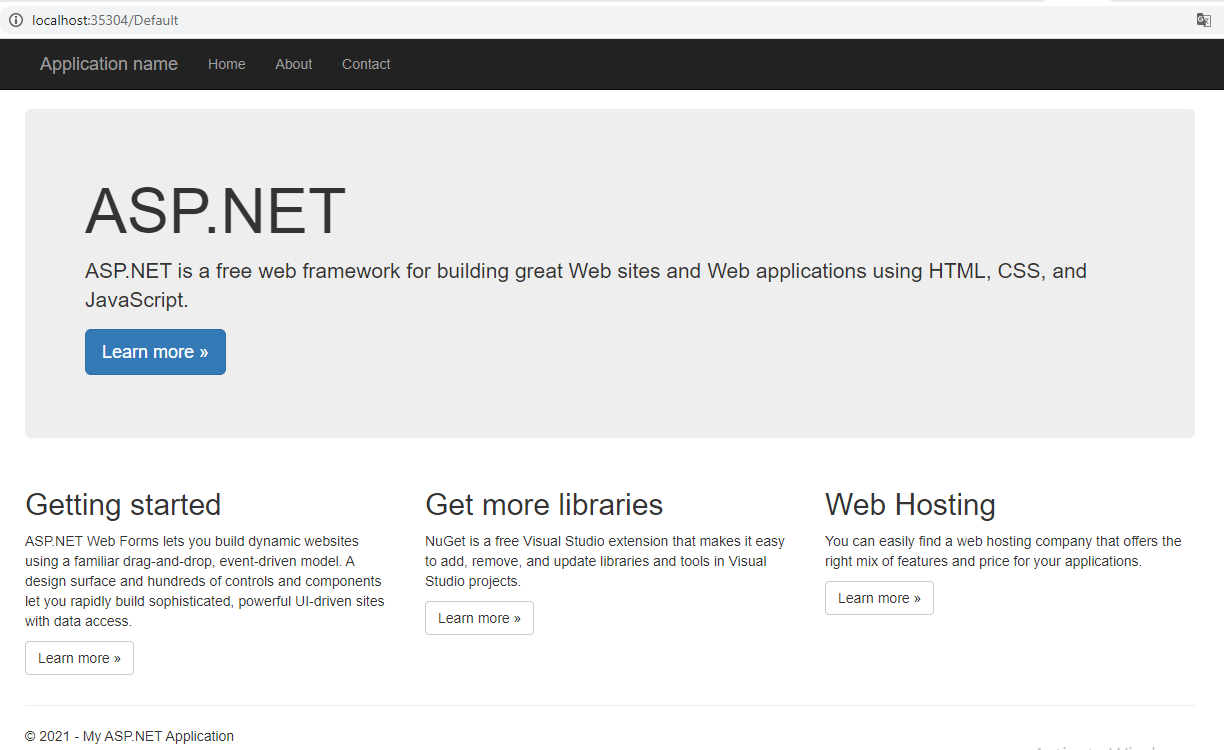
Những phần trên là do Visual Studio khởi tạo, chúng ta không cần nhất thiết phải bắt chước y chang cấu trúc, tập tin như vậy. Tại cửa sổ Solution Explorer này ta có thể thêm, xóa, sửa các thư mục, tệp tin để phù hợp với yêu cầu và cấu trúc dự án. Ta cũng có thể tạo dự án Web Empty (Web trống) nếu không muốn có cấu trúc thư mục, tập tin như mặc định và sau đó tự thêm các tập tin, thư mục theo ý muốn.

**Chạy thử dự án**

Để chạy thử dự án trên máy local, nhấn F5 (Debug -> Start Debugging) để chạy ở chế độ Debug để dò bắt lỗi hoặc Ctrl + F5 (Debug -> Start without debugging) nếu không cần dò lỗi.

Khi dự án Web chạy, nếu không định rõ trang web nào sẽ chạy đầu tiên thì mặc định sẽ chạy trang Default.aspx (trang chủ) trước.

Ví dụ, khi chạy thử dự án vừa tạo



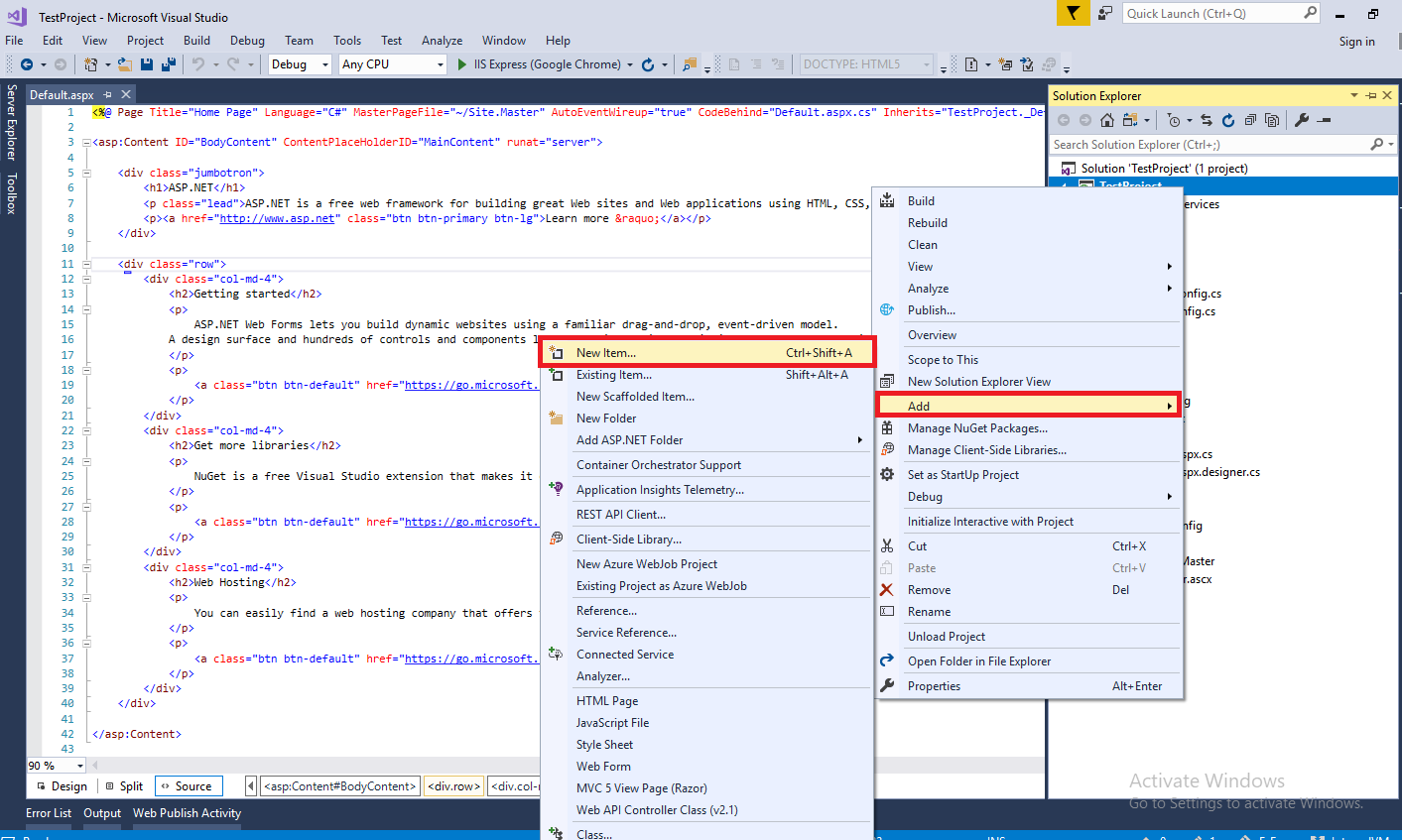
# CHAPTER V. Điều khiển Controls

1. **Cấu trúc một trang ASP.NET**

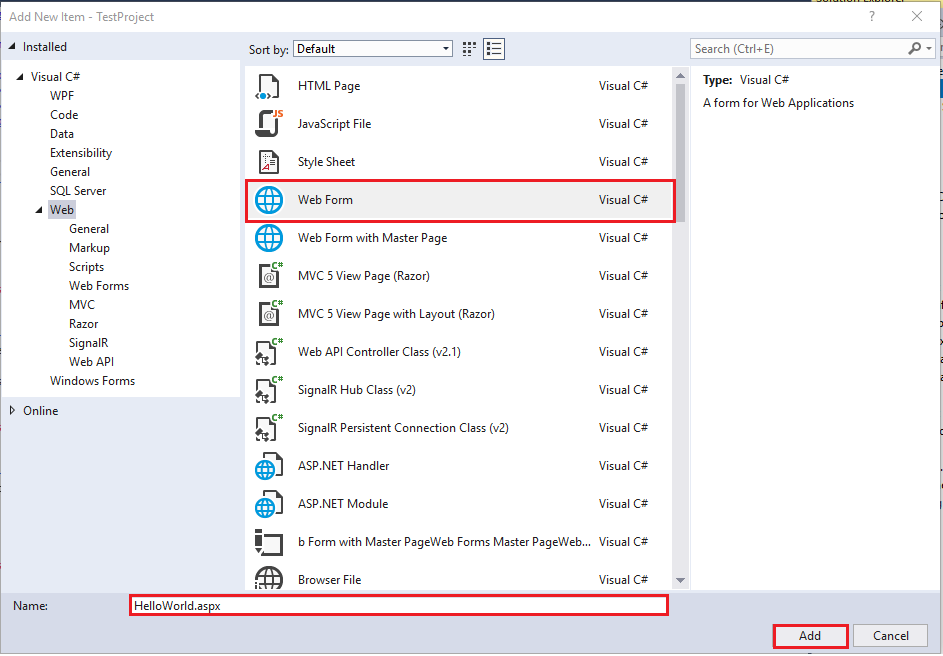
### a. Tạo một trang Asp.Net

Tạo một trang Asp.net HelloWorld đơn giản như sau:

- Bước 1: tại cửa sổ Solution Explorer, click chuột phải vào dự án, chọn Add -> New Item (hoặc nhấn tổ hợp phí Ctrl + Shift + A)



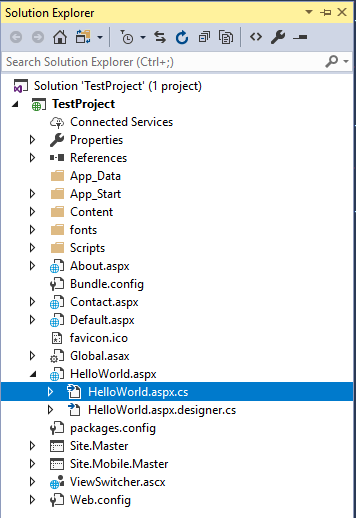
- Bước 2: Chọn đề mục WebForm, đặt tên cho trang và click Add để tạo trang

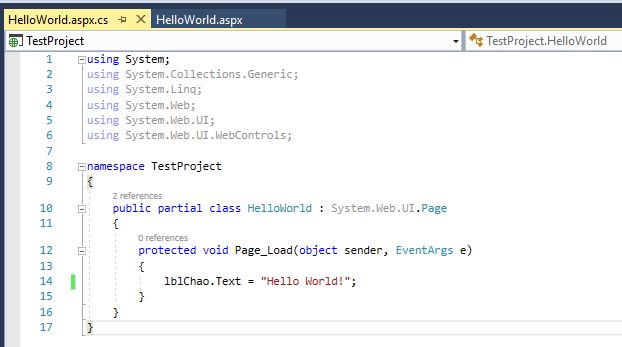


- Bước 3: tại trang HelloWorld.aspx vừa tạo, thêm tiêu đề cho trang và khai báo một control kiểu lable như sau:

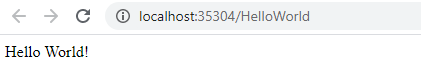


- Bước 4: để thêm nội dung cho thẻ lable, nhấn F7 để tới trang HelloWorld.aspx.cs (hoặc click vào file đó trên cửa sổ Solution Explorer. Tại trang code behind này, dùng id để gán giá trị cho cho thẻ lable vừa tạo.





- Bước 5: Chạy Ctrl F5 để xem kết quả



### b. Cấu trúc của trang Asp.Net

Như ta đã biết, mỗi trang Asp.net được gọi là một Web Form, thường gồm 2 file đi kèm với nhau: file .aspx chứa mã code HTML, CSS định hình giao diện trang web và file .aspx.cs chứa mã code behind (C#, VB, …) viết các phương thức xử lý sự kiện của trang.

**Cấu trúc của file .aspx gồm 2 phần cơ bản như sau :**

Phần 1: Được gọi là Page Directives, phần này cung cấp cho Asp.Net những thông tin đặc biệt để trình biên dịch biết được cách thực thi trang Asp.Net cũng như thông tin trong tiến trình biên dịch, gồm các thông tin sau :

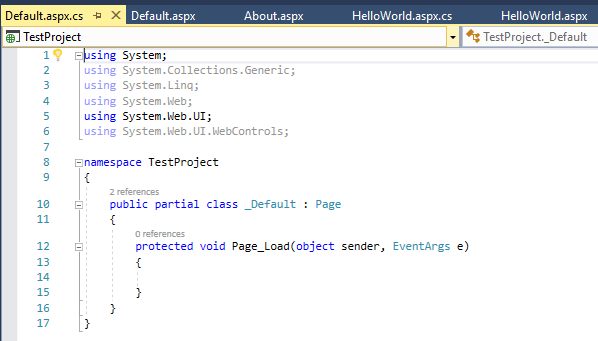


* <%@ Page %> : Khai báo các biên dịch trang.
* Language : khai báo ngôn ngữ được sử dụng để viết mã cho trang (C#, VB.NET…)
* AutoEventWireup : nếu giá trị là true thì các sự kiện của trang được tự động gọi đúng tên như Page\_Load mà không cần khởi tạo sự kiện chỉ đến phương thức Page\_Load.
* CodeBehind : chỉ ra tên tập tin code behind có phần mở rộng .aspx.cs được liên kết với trang .aspx này.
* Inherits : cho biết trang thừa kế từ lớp (class) nào.
* MasterPageFile : cho biết tên trang Master (trang định nghĩa bố cục) mà trang này sử dụng.

Phần 2: đoạn <html> <html>, là phần chứa mã HTML để khai báo bố cục, giao diện của trang Asp.net. Bố cục phần này tương tự như bố cục một trang HTML.

Cấu trúc của file .aspx.cs gồm:

* Phần using: khai báo các namespace cần thiết
* Phần public partial class: khai báo tên lớp thừa kế là lớp Page (tên lớp này được khai báo trong Inherits ở dòng chỉ dẫn @page của file aspx). Bên trong khai báo các sự kiện của trang (như Page\_Load)



1. **HTML Server Controls**

**Server Controls:** Để việc phát triển các ứng dụng web nhanh chóng và thuận tiện, Asp.Net cung cấp cho người dùng một tập hợp các điều khiển sẵn có để thực hiện hầu hết các công việc phổ biến trong lập trình như nhập, kiểm tra, hiển thị dữ liệu… gọi chung là Server Control.

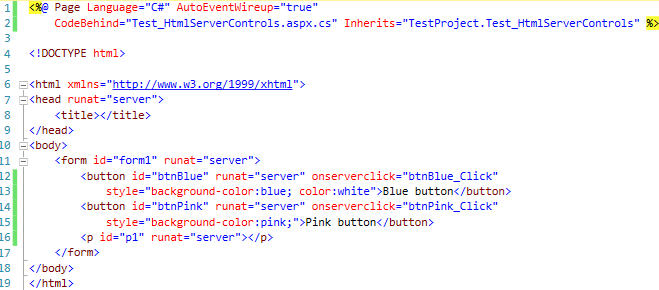
Các điều khiển này có thể chia thành những loại như: HTML Server Control, Web Server Control, Validation Server Controls, ngoài ra người dùng cũng có thể tự thiết kế các control của ứng dụng, gọi là User Control.

**HTML Server Control:** tiền thân là phần tử HTML trong trang HTML, khi muốn sử dụng chúng để lập trình phía server, ta gán thuộc tính runat="Server" cho các phần tử HTML đó. Những phần tử HTML có thuộc tính runat="Server" này được gọi là HTML Server Control.

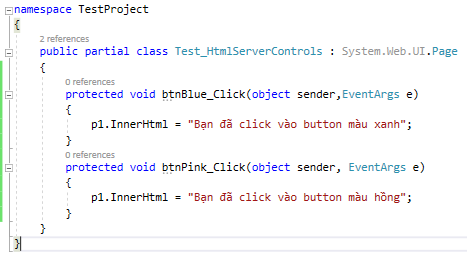
Các đối tượng HTML server control khai báo trong namespace System.Web.UI.HtmlControls được lấy từ lớp cơ sở HtmlControl.

**Ví dụ:**

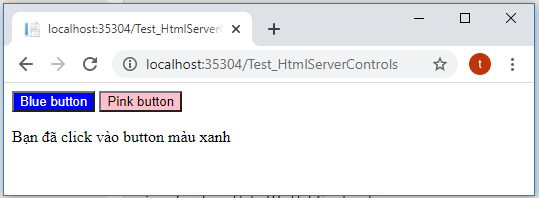
Có file aspx như sau:



Code phía server như sau:



Giao diện khi chạy có hai button một màu xanh, một màu hồng. Khi click vào button thì hàm xử lý sự kiện tương ứng là gán nội dung cho phần tử p và hiển thị như hình dưới:



*\* HTML Server Control tuy xử lí nhanh nhưng không có nhiều chức năng nên thông thường khi lập trình sẽ dùng các điều khiển ASP.Net Web Server Controls.*

1. **Web Server Controls**

**ASP.Net Web Server Controls:** Là các điều khiển chính được sử dụng trong ASP.Net. Cũng giống như HTML server control, các Web server control cũng được tạo trên server và có sẵn thuộc tính runat="server" để thực thi tạo server. Tuy nhiên, Web server control không nhất thiết phải ảnh xạ tới bất kỳ thẻ HTML nào đã tồn tại và chúng được hỗ trợ để thực hiện nhiều tính năng phức tạp hơn.

Các đối tượng Web Server Control khai báo trong namespace System.Web.UI.Web.Control được lấy từ lớp cơ sở System.Object.

Cú pháp chung để tạo một Web Server Control là:

<asp:TypeOfControl id="control\_name" runat="server"/>

Với ID là định danh cho control và là duy nhất; runat="server" là khai báo bắt buộc, để xác định control tồn tại trên server.

Các Web Server Controls có các thuộc tính chung và riêng cho từng control. Các thuộc tính chung như:

* Thuộc tính Attributes: Tập hợp các thuộc tính của điều khiển HTML
* Các thuộc tính quy định màu nền, màu đường viền, kiểu đường viền, màu chữ… như BackColor, BorderStyle, ForeColor…
* Các thuộc tính quy định chiều cao, độ rộng của điều khiển như Height, Width
* Thuộc tính CssClass: Quy định hình thức hiển thị của điều khiển qua tên CSS.
* Thuộc tính Enabled: Quy định điều khiển có được hiển thị hay không. Giá trị mặc định của thuộc tính này là True – được phép hiển thị.

Một số Web Server Controls thường dùng như: Lable, TextBox, HyperLink, Image, Button, LinkButton, ImageButton, CheckBox, RadioButton, DropDownList, DataGrid, Panel…

**a. Label, HyperLink, Image**

**Label** là điều khiển thường được sử dụng để hiển thị và trình bày văn bản trên trang web. Nội dung được hiển thị trong label được xác định thông qua thuộc tính Text.

Cú pháp:

<asp:Label ID="Label1" runat="server" Text="Label" ></asp:Label>

**HyperLink** là điều khiển sử dụng để tạo các liên kết siêu văn bản, giống với phần tử <a> trong HTML. Thuộc tính NavigateUrl chỉ định đường dẫn cần liên kết.

**Image** là điều khiển dùng để hiển thị hình ảnh, giống với phần tử <img> trong HTML. Thuộc tính ImageURL chỉ định đường dẫn đến tập tin hình ảnh cần hiển thị.

**b. Textbox**

TextBox là điều khiển được dùng để nhập và hiển thị dữ liệu

Cú pháp:

<asp:TextBox ID="TextBox1" runat="server" ></asp:TextBox>

Thuộc tính: bao gồm các thuộc tính chung của control và một số thuộc tính riêng như:

* Text: lưu nội dung của control
* MaxLength: quy định độ dài của dữ liệu mà người dùng có thể nhập trên TextBox
* ReadOnly: quy định điều khiển chỉ để hiển thị dữ liệu mà không được nhập dữ liệu vào. Mặc định giá trị là false – có thể nhập và hiển thị dữ liệu.
* …

Phương thức và sự kiện: TextBox được hỗ trợ một số phương thức và sự kiện như

* Focus: thiết lập form khi khởi tạo ưu tiên trỏ tới TextBox
* TextChanged: xảy ra trên Server khi nội dung TextBox thay đổi. Để sự kiên này xảy ra bạn cần thiết đặt thuộc tính AutoPostback là true.

**c. Button, LinkButton, ImageButton**

**Button control** được sử dụng để hiển thị nút nhấn trên trang web. Mặc định là các nút Submit Button, mỗi khi click vào nó sẽ postback về server.

Cú pháp:

<asp:Button ID="Button1" runat="server" Text="Button" />

Thuộc tính: bao gồm các thuộc tính chung của control và một số thuộc tính riêng như:

* OnClientClick: chỉ định đến một hàm phía client sẽ được thực hiện khi nhấn vào Button.
* Text: Nội dung text hiển thị trên điều khiển Button.
* …

Phương thức và sự kiện: Button được hỗ trợ một số phương thức và sự kiện như

* Focus: thiết lập khi khởi tạo Form ưu tiên điều khiển Button.
* Click: Xảy ra khi điều khiển Button được nhấn.
* Command: Xảy ra khi điều khiển Button được nhấn. CommandName và CommandArgument được truyền qua sự kiện.

**LinkButton** là điều khiển tạo nút button hiển thị dưới dạng text giống như thẻ <a> của HTML. LinkButton control cũng mặc định là submit button và có các phương thức, thuộc tính giống với điều khiển button.

**ImageButton** là điều khiển tạo nút button hiển thị dưới dạng hình ảnh. Thuộc tính ImageUrl chỉ định đường dẫn đến tập tin hình ảnh cần hiển thị. ImageButton control cũng mặc định là submit button và có các phương thức, thuộc tính giống với điều khiển button.

**d. CheckBox**

**Checkbox** là điều khiển nhận giá trị của người dùng từ các lựa chọn, Checkbox cho phép chọn một hoặc nhiều trong một danh sách các giá trị.

Cú pháp:

<asp:CheckBox ID="CheckBox2" runat="server" Text="" />

Thuộc tính: bao gồm các thuộc tính chung của control và một số thuộc tính riêng như:

* Text: Nội dung text hiển thị của điều khiển Checkbox.
* Checked : Cho phép gán hoặc thiết lập trạng thái chọn hay không chọn của CheckBox.
* AutoPostBack: Xác định form có tự động post back về server sau khi thuộc tính Checked được thay đổi hay không. Mặc định là không.
* …

Phương thức và sự kiện: Checkbox được hỗ trợ một số phương thức và sự kiện như

* Focus: thiết lập khi khởi tạo Form ưu tiên điều khiển này.
* CheckedChanged : xảy ra khi người dùng thay đổi lựa chọn của Checkbox (điều kiện: AutoPostBack=True)
* ...

**e. RadioButton**

**RadioButton** là điều khiển cho phép người dùng lựa chọn một và chỉ một phương án trong danh sách các lựa chọn.

Cú pháp:

< asp:RadioButtonID="RadioButton1" runat="server" Text=" " GroupName=" "/>

Thuộc tính: bao gồm các thuộc tính chung của control và một số thuộc tính riêng như:

* Text: Nội dung text hiển thị trên điều khiển RadioButton.
* GroupName : chỉ định tên nhóm của danh sách các lựa chọn này. Các lựa chọn trong cùng danh sách phải có cùng một GroupName.
* Checked : Cho phép gán hoặc thiết lập trạng thái chọn hay không chọn của RadioButton.
* AutoPostBack: Xác định form có tự động post back về server sau khi thuộc tính Checked được thay đổi hay không. Mặc định là không.
* …

Phương thức và sự kiện: RadioButton được hỗ trợ một số phương thức và sự kiện như

* Focus: thiết lập khi khởi tạo Form ưu tiên điều khiển này.
* CheckedChanged : xảy ra khi người dùng thay đổi lựa chọn của Checkbox
* ...

**f. DropDownList**

**DropDownList** là điều khiển hiển thị danh sách lựa chọn theo kiểu đổ xuống. Mỗi item trong DropDownList control được định nghĩa bởi một LisItem.

Cú pháp:

<asp:DropDownList ID="" runat="server">

<asp:ListItem Text="" Value="" Selected="True"></asp:ListItem>

<asp:ListItem Text="" Value=""></asp:ListItem>

<asp:ListItem Text="" Value=""></asp:ListItem>

<asp:ListItem Text="" Value=""></asp:ListItem>

</asp:DropDownList>

Thuộc tính: bao gồm các thuộc tính chung của control và một số thuộc tính riêng như:

* Text: Nội dung text hiển thị của một Item.
* Value: Giá trị của một Item.
* Selected: Chỉ định Item đang được chọn hay không. Mặc định là False.
* AutoPostBack: Xác định form có tự động post back về server sau khi thuộc tính Selected được thay đổi hay không. Mặc định là không.
* DataSource: nguồn dữ liệu của DropDownList control dưới dạng bảng với DataValueField: trường hiển thị giá trị Value, DataTextField: trường hiển thị Text của các Item trong DropDownList.
* Items: tập hợp các ListItem trong DropdownList

+ Thêm Item: Items.Add(ListItem)

+ Chèn Item: Items.Insert(index, ListItem)

+ Xóa Item: Items.Remove(ListItem); Items.RemoveAt(index); Items.Clear()

* SelectedIndex: chỉ số của mục được chọn
* SelectedItem: đối tượng ListItem được chọn
* SelectedValue: giá trị của mục được chọn
* SelectedText: văn bản hiển thị trên mục được chọn
* …

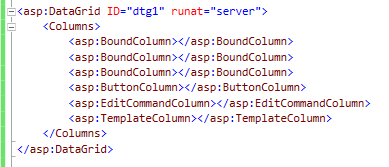
Phương thức và sự kiện: DropDownList được hỗ trợ một số phương thức và sự kiện như

* SelectedIndexChanged: xảy ra khi index của item được chọn bị thay đổi
* DataBind(): phương thức dùng để mapping các thành phần của data source vào DataSource của DropDownList.
* …

**g. DataGrid**

**DataGrid** là điều khiển dùng để hiển thị, định dạng và thao tác với dữ liệu. Bên cạnh đó, DataGrid control được ASP.Net hỗ trợ nhiều tính năng khác như sắp xếp dữ liệu, phân trang…

Cấu trúc:



Trong đó Columns là các cột sẽ hiển thị trên lưới, bao gồm:

* BoundColumn: Cột có liên kết với nguồn dữ liệu.
* ButtonColumn: Cột dạng nút lệnh đã được thiết kế sẵn. Điều khiển DataGrid cung cấp cho chúng ta 3 loại cột dạng này:

+ Select: Nút lệnh chọn dòng dữ liệu

+ Edit, Cancel, Update: Các nút lệnh hỗ trợ chức năng cập nhật dữ liệu trực tiếp trên lưới.

+ Delete: Nút lệnh xóa dòng dữ liệu

* HyperLinkColumn: Cột có liên kết dữ liệu dạng liên kết.
* EditCommandColumn: Cột chứa các nút Edit để chỉnh sửa các mục dữ liệu trong mỗi hàng.
* TemplateColumn: Cột do người dùng tự thiết kế. Đây là loại cột có khả năng làm việc khá linh hoạt.

DataGrid control ngoài những thuộc tính chung của control còn có rất nhiều thuộc tính và phương thức riêng. Một số thuộc tính, phương thức và sự kiện thường dùng như:

Thuộc tính về định dạng:

* ShowHeader: Quy định dòng tiêu đề trên có được phép hiển thị hay không. (mặc định là có hiển thị dòng tiêu đề)
* ShowFooter: Quy định dòng tiêu đề dưới có được phép hiển thị hay không. (mặc định là không hiển thị dòng tiêu đề dưới)
* CellPadding: Quy định khoảng cách giữa nội dung trong ô với các đường viền của ô.
* CellSpacing: Quy định khoảng cách giữa các ô AllowSorting: Có cho phép sắp xếp dữ liệu hay không (mặc định là không cho phép sắp xếp)
* AllowSorting: Có cho phép sắp xếp dữ liệu hay không (mặc định là không cho phép sắp xếp)

Thuộc tính của BoundColumns: Quy định thông tin chi tiết cho các cột

* HeaderText, Footer Text: Thông tin tiêu đề trên/dưới của cột
* Header Image: Hình hiển thị trên tiêu đề cột (thay thế thông tin tiêu đề cột - Header Text).
* Sort Expression: Biểu thức sắp xếp của cột.
* Visible: Quy định cột có được hiển thị hay không.
* DataField: Quy định tên field hay tên thuộc tính của đối tượng dữ liệu cần hiển thị.
* Data formatting expression: Biểu thức định dạng dữ liệu. Mẫu định dạng: {0:<chuỗi định dạng>}. Ví dụ:

+ Định dạng số: {0:000.00}, {0:0.##}

+ Định dạng ngày giờ: {0:dd/MM/yyyy}, {0:hh/mm/ss tt}

* Read Only: Chọn giá trị này để cột chỉ được phép đọc, không cho phép cập nhật dữ liệu.

Thuộc tính về phân trang:

* AllowPaging: Có cho phép phần trang hay không.
* PageSize: Quy định số dòng của mỗi trang.
* PagerStyle-Position: Quy định vị trí hiển thị của bộ nút di chuyển. Ở phía trên thanh tiêu đề, ở phía dưới hay cả hai.
* PagerStyle-Mode: Quy định hình thức hiển thị của bộ nút di chuyển. Hiển thị dạng số trang hay là các chuỗi ký tự đại diện (Next page/Previous page button text). Trong trường hợp hiển thị dạng số, Numeric buttons quy định số nút lệnh được hiển thị tối đa.

Phương thức và sự kiện:

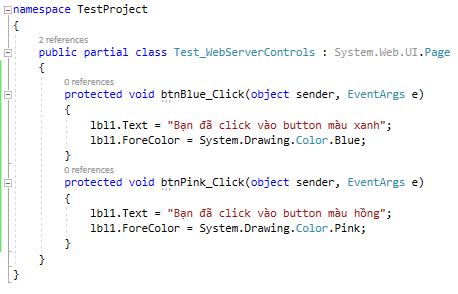
* DataBind(): liên kết control và tất cả các con của nó với nguồn dữ liệu được chỉ định.
* SortCommand: xảy ra khi sắp xếp dữ liệu của cột (AllowSorting="True")
* PageIndexChanged: xảy ra khi một trong những phần tử chọn trang được nhấp vào (khi phân trang AllowPaging="true")
* EditCommand: xảy ra khi nút Sửa Xóa được click cho một mục trong điều khiển DataGrid
* DeleteCommand: xảy ra khi bút Xóa được click cho một mục trong điều khiển DataGrid
* …

**h. Ví dụ:** viết lại ví dụ trong phần HTML Server Control bằng Web Server Control

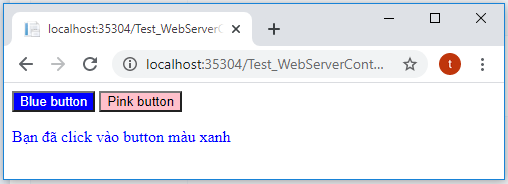
Có file aspx như sau:



Code phía server như sau:



Giao diện khi chạy có hai button một màu xanh, một màu hồng. Khi click vào button thì hàm xử lý sự kiện tương ứng và trả ra nội dung hiển thị như hình dưới:



1. **Validation Controls**

**Data Validation – Xác thực dữ liệu**

Ở bài về JavaScript ta đã biết dữ liệu có thể được xác thực từ phía client hoặc xác thực phía server. Việc xác thực phía client sẽ được thực hiện nhanh hơn và tiết kiệm tài nguyên server, tuy nhiên nó không an toàn vì có rất nhiều ngoại lệ hoặc người dùng xấu có thể vượt qua được kiểm tra phía client khiến dữ liệu không hợp lệ được đưa vào hệ thống. Vì vậy nhiều trường hợp để đảm bảo tính an toàn, ta nên hoặc phải sử dụng xác thực phía server.

Nhìn chung, ta nên kết hợp cả 2 loại validation: client-side validation để tạo trải nghiệp người dùng tốt cũng như hỗ trợ một phần cho server-side; server-side validation để đảm bảo sự chính xác và an toàn của dữ liệu trước khi xử lý.

**Validation Controls**

ASP.Net cung cấp một bộ điều khiển xác thực dữ liệu được tích hợp sẵn để việc lập trình nhanh chóng, tiện lợi và chuyên nghiệp hơn.

Cơ chế hoạt động: mỗi Validator kiểm tra cho một Server Control, mỗi Server Control có thể được kiểm tra bởi nhiều Validator khác nhau. Kết quả kiểm tra sẽ trả vào thuộc tính IsValid của Validation đó (true/false). Việc thay đổi IsValid của Validator cũng làm thay đổi IsValid của đối tượng Page.

Một số thuộc tính chung:

* ControlToValidate: khai báo control cần được xác thực.
* Text: Chuỗi thông báo xuất hiện khi có lỗi.
* ErrorMessage: Chuỗi thông báo xuất hiện trong điều khiển Validation Summary. Giá trị này sẽ được hiển thị tại vị trí của điều khiển nếu thuộc tính Text không được gán giá tị.
* EnableClientScript: Có cho phép thực hiện kiểm tra ở phía Client hay không. Mặc định là true – có kiểm tra.
* Display: thiết lập bố cục của thông báo lỗi:

+ None: không hiển thị thông báo lỗi (vẫn có kiểm tra dữ liệu)

+ Static: trong trường hợp không có lỗi thì thông báo vẫn chiếm phần diện tích báo lỗi.

+ Dynamic: nếu có thông báo lỗi thì hiển thị, nếu không hiển thị sẽ không chiếm phần diện tích này.

Một số Validation Controls thường dùng như sau:

**RequiredFieldValidator**: Điều khiển này yêu cầu người dùng phải nhập giá trị vào một trường chỉ định trên form. Nếu giá trị rỗng thì không submit được.

**CompareValidator**: Điều khiển này cho phép so sánh dữ liệu đầu vào từ một điều khiển với dữ liệu trong một điều khiển khác hoặc với một giá trị không đổi. Ví dụ: Khi cần nhập mật khẩu hai lần.

Ngoài các thuộc tính chung, Validator này có một số thuộc tính riêng như:

* Operator: quy định kiểu so sánh: DataTypeCheck(kiểm tra kiểu dữ liệu), Equal(=), GreaterThan(>), GreaterThanEqual(>=), NotEqual(khác nhau), …
* ControlToCompare: khai báo điều khiển cần so sánh giá trị, nếu chọn Operator là DataTypeCheck thì sẽ không cần xác định giá trị cho thuộc tính này.
* Type: quy định kiểu dữ liệu: int, string, double…
* ValueToCompare: khai báo giá trị cần so sánh. Khi khai báo giá trị cả 2 thuộc tính ControlToCompare và ValueToCompare thì ưu tiên giá trị khai báo trong thuộc tính ControlToCompare.

**RangeValidator**: Kiểm tra xem đầu vào của control được chỉ định có nằm trong phạm vi giá trị nhất định không.

Điều khiển này có một số thuộc tính riêng như:

* MinimumValue: Giá trị nhỏ nhất.
* MaximumValue: Giá trị lớn nhất.
* Type: quy định kiểu dữ liệu: int, string, double…

**RegularExpressionValidator**: Điều khiển này sử dụng biểu thức chính quy Regular Expression để kiểm tra đầu vào của control được chỉ định.

Thuộc tính ValidationExpress: khai báo biểu thức kiểm tra viết theo chuẩn Regular Expression (đã học trong bài JavaScript) hoặc có thể sử dụng biểu thức sẵn có trong RegularExpressionEditor của Visual Studio.

**CustomValidator**: Là điều khiển mà lập trình viên có thể viết các logic xác thực dữ liệu riêng theo nhu cầu của mình. Validator này cho phép tự xây dựng hàm kiểm tra cả ở phía client và server.

Thuộc tính:

* ValidateEmptyText: có kiểm tra khi dữ liệu của điều khiển rỗng hay không.
* ClientValidationFunction: khai báo tên hàm kiểm tra phía client. Hàm này có dạng:

<script type="text/javascript">

function theFunction(oSrc,args){

args.IsValid = kiểm tra(args.Value)

}

</script>

Sự kiện ServerValidate: đặt các xử lý kiểm tra dữ liệu hoặc khai báo tên hàm kiểm tra dữ liệu phía server.

**ValidationSummary**: Điều khiển này cho phép liệt kê tất cả các lỗi kiểm tra trên trang từ những điều khiển Validator vào một vị trí.

Thuộc tính:

* HeaderText: Dòng tiêu đề của thông báo lỗi.
* ShowMessageBox: Quy định bảng thông báo lỗi có được phép hiển thị như cửa sổ MessageBox hay không. Mặc định là false – không hiển thị.
* ShowSummary: Quy định bảng thông báo lỗi có được phép hiển thị hay không. Mặc định là True – được phép hiển thị.

1. **Web User Control**

**Web User Controls:** Là các controls do người dùng định nghĩa, có thể được sử dụng lại trong các trang của một dự án ASP.Net.

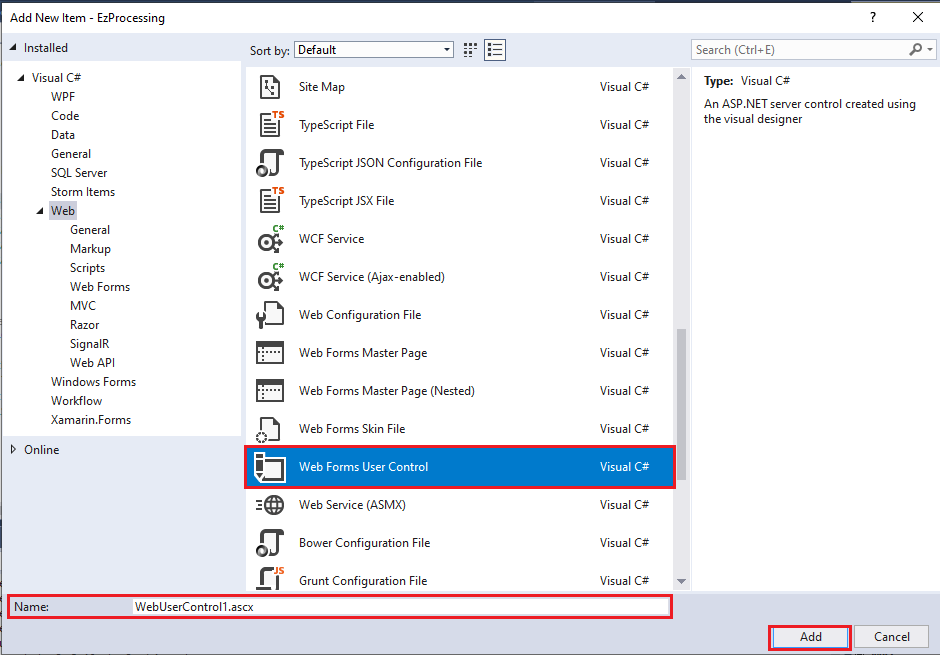
Một User Control bắt nguồn từ TemplateControl kế thừa từ lớp System.Web.UI.UserControl và có một số đặc điểm như sau:

* Tương tự như một trang aspx nhưng có phần mở rộng là .ascx
* Có mã HTML, các server controls, mã JavaScript… và các mã code phía server
* Có thể có các thuộc tính và sự kiện
* Cho phép lập trình viên tạo các điều khiển riêng với giao diện và hành vi tùy chỉnh theo nhu cầu của mình.
* Không thể yêu cầu trực tiếp từ trình duyệt web mà được sử dụng bằng cách thêm nó vào một trang khác (sử dụng như một server control)

**Tạo và sử dụng một User Control**

Các bước tạo một User Control:

* Chuột phải vào Solution Explorer -> Add -> New Item… hoặc nhần tổ hợp phím Ctrl + Shift + A.
* Trong form Add New Item, chọn Web ở khung bên trái, trong khung bên phải chọn Web User Control. Đặt tên cho control và nhấn Add để thêm.



Trong trang ascx:

* Bao gồm mã HTML, JavaScript và các controls…
* Không chứ các thẻ HTML <head>, <body> và <form>
* Sử dụng @Control thay vì @Page

Sử dụng một User Control: Có thể kéo trực tiếp user control vào trang aspx hoặc khai báo như sau:

* Trong phần Page Directives đăng ký để có thể sử dụng User Control đã tạo. Dùng @Register với TagName: xác định tên User Control, Src: đường dẫn đến User Control, TagPrefix: tiền tố của TagName trong khai báo User Control

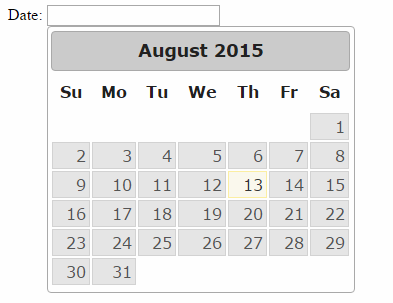
<%@ Register Src="~/TestControl.ascx" TagPrefix="uc1" TagName="TestControl" %>

* Trong phần code HTML, để gọi đến User Control đã khai báo trên phần Page Directives, dùng cú pháp <TagPrefix: TagName />

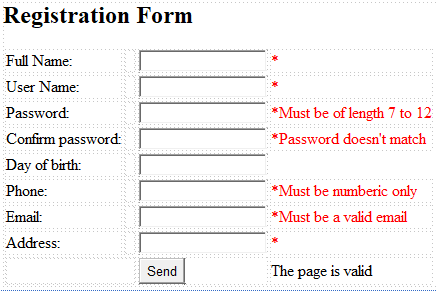
<uc1:TestControl runat="server" id="TestControl" />

1. **Bài tập**

Ex1: Tạo một web user control đặt tên là datepicker như sau: khi click vào ô textbox thì hiện bảng chọn ngày, chọn một ngày dưới bảng thì trên textbox tự động lấy ngày đó lên. Định dạng ngày hiển thị trên textbox theo dạng dd/mm/yyyy



Ex2: Sử dụng các phần đã học tạo trang như sau:



Yêu cầu:

* Sử dụng Web User Control để tạo các ô textbox
* Sử dụng Validator Control để xác thực dữ liệu
* Trường Day of birh sử dụng datepicker control đã tạo ở bài 1
* Các trường Full Name, User Name, Password, Confirm password, Phone, Email, Address không được để trống, nếu để trống thì khi Submit sẽ hiển thị dấu \* màu đỏ bên cạnh.
* Trường Password: chỉ được nhập chuỗi có độ dài từ 7 – 12 ký tự
* Trường Confirm password: dữ liệu nhập vào phải giống dữ liệu nhập vào trường Password
* Trường Phone: chỉ được nhập vào dữ liệu là số
* Trường Email: dữ liệu nhập vào phải đúng định dạng email
* Các trường nhập sai khi Submit sẽ hiển thị thông báo màu đỏ bên cạnh. Nếu tất cả các trường đều phù hợp yêu cầu thì hiển thị dòng text "The page is valid"

# CHAPTER VI. Master Page

1. **Master Page**

Trong một hệ thống website thường sẽ có nhiều trang con khác nhau, mỗi trang sẽ phụ trách một mảng nội dung nào đó, có thể kết cấu trang cũng có những điểm khác biệt nhất định nhưng tất cả sẽ được thiết kế theo một khuôn mẫu nào đó, và áp dụng chung cho tất cả các trang. Khuôn mẫu này thường chúng ta sẽ gọi là Theme hay Template, còn trong ASP.Net có khái niệm MasterPage.

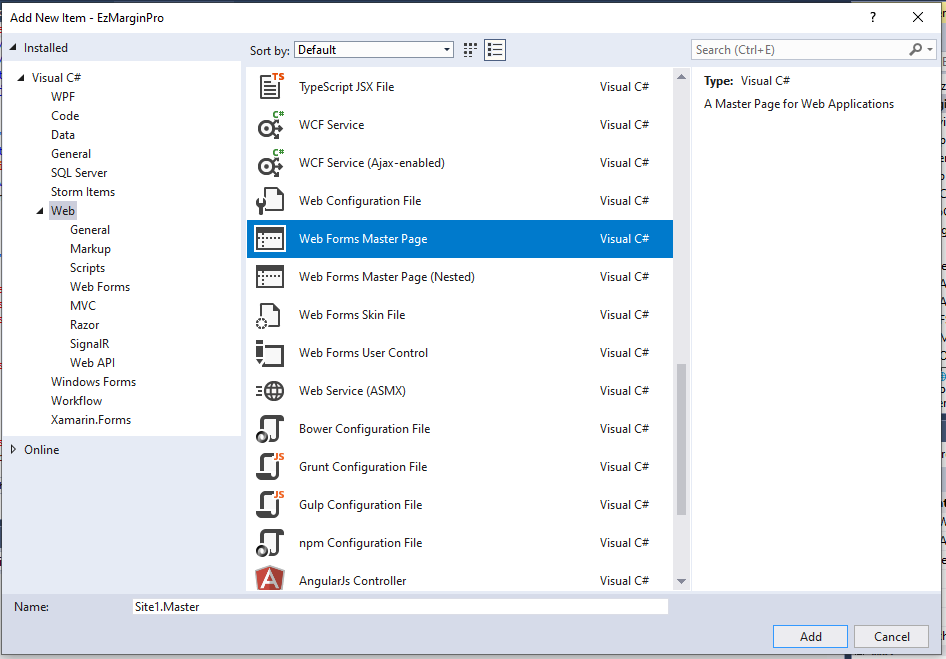
MasterPage là một trang dùng để tạo ra cấu trúc giao diện chung cho các trang web trong hệ thống, những trang thừa kế từ MasterPage sẽ áp dụng lại giao diện mẫu của nó. Trong một website của thể có 1 hoặc nhiều MasterPage.

Trong MasterPage được chia thành 2 phần:

* Phần chung: Đây là giao diện chung áp dụng cho những trang con kế thừa từ nó
* Phần riêng: Đây là nơi chứa nội dung riêng của từng trang.

**Tạo một Master Page**

* Mở cửa sổ Add -> New Item (hoặc nhấn Ctrl + Shift + A)
* Chọn Web Forms Master Page, đặt tên và nhấn Add để thêm



**Cấu trúc một trang Master Page**

Về cơ bản MasterPage cũng là 1 trang thông thường nên cũng sẽ có cấu trúc như 1 trang html bình thường, việc sắp xếp, bố trí giao diện vẫn dùng css, js như thông thường. Một trang .Master có cấu trúc tương tự như một trang .aspx

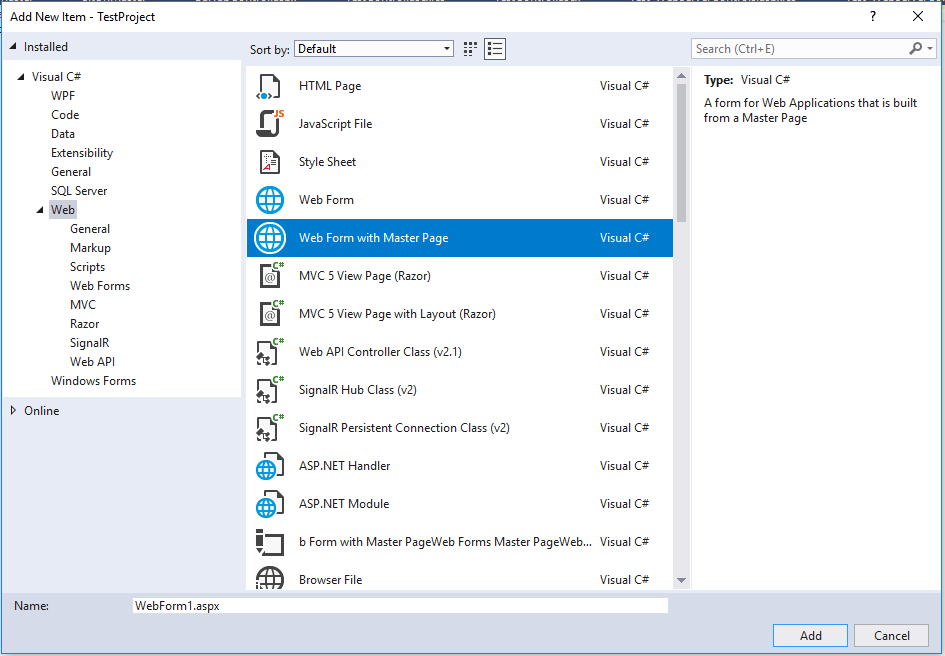
* Phần chỉ dẫn bắt đầu bằng @Masterdirective (Chủ yếu là các thuộc tính giống với @Pagedirective)
* Phần code HTML

+ Phần nội dung cho trang nằm trong cặp thẻ <html> gồm các phần tử HTML, các server control…

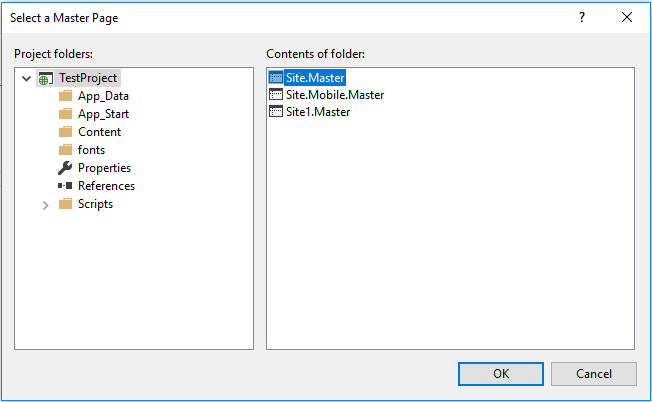
+ Đối tượng ContentPlaceHolder: <asp: ContentPlaceHolder> là điều khiển khai báo phần dành riêng cho các trang con thừa kế từ Master Page, các phần còn lại sẽ là phần chung.

**Tạo trang kế thừa từ MasterPage – Content Page**

* Nhấn nhấn Ctrl + Shift + A để mở cửa sổ Add New Item
* Chọn Web Form with Master Page



* Đặt tên cho trang và nhấn Add, cửa sổ Select a Master Page sẽ hiện lên, tại đây ta chọn trang Master Page muốn dùng và nhấn Ok



**Cấu trúc trang Content Page**

Các trang Content sẽ thừa kế layout của trang Master và có những phần riêng tại các vị trí đã khai báo trong đối tượng ContentPlaceHolder. Tại trang Content ta sẽ thêm nội dung riêng của chúng. Cấu trúc như sau:

* Sử dụng chỉ thị @Page với thuộc tính MasterPageFile để trỏ đến trang Master
* Thêm phần nội dung riêng của trang bên trong điều khiển <asp:Content> (đây là phần tương ứng với điều khiển ContentPlaceHolder từ trang Master). Điều khiển Content phải khai báo Thuộc tính ContentPlaceHolderID trỏ đến ID của điều khiển ContentPlaceHolder của trang Master mà chúng ta muốn thay thế nội dung.

1. **Đối tượng Request và Response**

ASP.Net cung cấp một số đối tượng đã được xây dựng sẵn mà không cần khởi tạo như: Request, Response, Server, Session, Application…

Request và Response là 2 đối tượng được dùng rất nhiều trong lập trình ASP, dùng để chuyển hướng, định hướng hay thực hiện việc giao tiếp giữa client với server hoặc giữa các trang web của một ứng dụng.

**Đối tượng Request**

Đối tượng Request trong ASP.Net được dùng để nhận thông tin từ trình duyệt của người dùng gửi về Web Server để xử lý thông tin.

Đối tượng này có một số thuộc tính như:

* ApplicationPath: trả về đường dẫn ảo của ứng dụng
* Url: trả về đường dẫn của trang
* Brower: trả về đối tượng HttpBrowserCapabilities trình bày thông tin của trình duyệt
* Cookie: trả về đối tượng HttpCookiesCollection
* QueryString: trả về giá trị trong phần QueryString của Url

**Đối tượng Response**

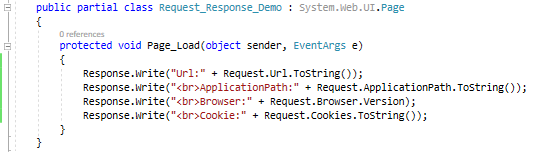
Khi đối tượng Request bao gồm những thông tin gửi đến Web server từ trình duyệt thì đối tượng Response nắm giữ những gì mà Web server phải gửi trả lại cho trình duyệt.

Đối tượng này gồm hai phương thức chính là Write và Redirect:

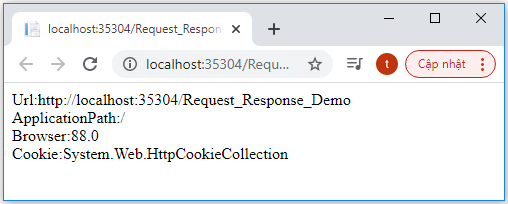
* Response.Write: dùng để in ra một chuỗi trên trang web.
* Response.Redirect: dùng để chuyển hướng sang một trang Asp.Net khác (có thể dùng địa chỉ tuyệt đối hoặc địa chỉ tương đối)

**Ví dụ:** dùng Response trả ra các thông tin như Cookie, version của Browse, địa chỉ Url…

Code phía server



Kết quả khi chạy



1. **Đối tượng Cookies**

ASP.NET Cookie là một đoạn văn bản nhỏ được sử dụng để lưu trữ thông tin người dùng cụ thể. Ứng dụng web có thể đọc thông tin này bất cứ khi nào người dùng truy cập trang web.

Khi người dùng yêu cầu một trang web, máy chủ web không chỉ gửi một trang mà còn gửi một cookie chứa ngày và giờ. Cookie này lưu trữ trong một thư mục trên đĩa cứng của người dùng.

Khi người dùng yêu cầu lại trang web, trình duyệt sẽ tìm cookie được liên kết với trang trên ổ cứng. Trình duyệt lưu trữ cookie riêng biệt cho từng trang web khác nhau mà người dùng đã truy cập.

**Class HttpCookie**

Trong ASP.NET các phương thức, thuộc tính liên quan đến việc xử lý Cookie được lưu trữ trong class HttpCookie. Các thuộc tính thường dùng trong Cookie:

* Name: Tên Cookie.
* Domain: Tên miền của Cookie lưu trữ.
* Expires: Xác định thời gian hiệu lực của Cookie.
* Value: Giá trị của Cookie.
* HasKeys: Cho biết Cookie có tập giá trị con hay không.
* Values: Tập các giá trị của Cookie.

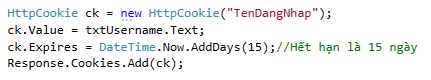
**Tạo/Viết cookie**

Có rất nhiều cách để tạo hoặc viết cookie. Cách cơ bản nhất là sử dụng lớp HttpCookies

Cú pháp:

Response.Cookies.Add(<HttpCookie>)

Ví dụ: tạo một cookie có tên là TenDangNhap lưu trữ tên đăng nhập của người dùng. Thông tin này sẽ được lưu trữ trên Cookies 15 ngày kể từ ngày hiện hành trên Web Server.



**Đọc/lấy giá trị cookie**

Cú pháp:

Request.Cookies[<Tên cookie>].Value;

Ví dụ: để lấy giá trị của cookie TenDangNhap ở ví dụ trên



*Lưu ý: Trong trường hợp Cookies chưa được lưu hoặc đã hết thời hạn duy trì tại Client, giá trị nhận được là null.*

**Hạn chế của cookie**

Cookie có những hạn chế sau đây:

* Cookie được giới hạn ở kích thước nhỏ và chỉ có thể được sử dụng để lưu trữ văn bản 4 KB (4096 Byte).
* Trên một phiên làm việc, chỉ có 20 cookie được hoạt động, nếu có cookie mới, thì cookie cũ nhất sẽ bị delete
* Người dùng cuối có thể dừng việc chấp nhận sử dụng cookie trên trình duyệt, do đó khi thao tác cần nhắc nhở user kích hoạt việc dùng cookie

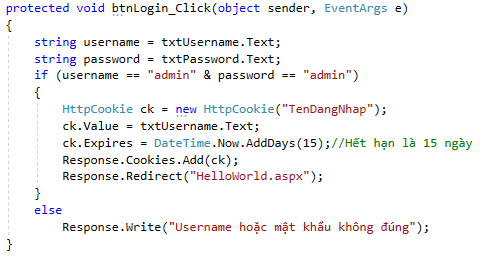
**Ví dụ về sử dụng Cookie:** sau khi đăng nhập thành công thì chuyển đến trang HelloWorld và gửi lời xin chào + tên đăng nhập.

Tạo 2 trang web: trang đăng nhập CookiesDemo.aspx và trang HelloWorld.aspx

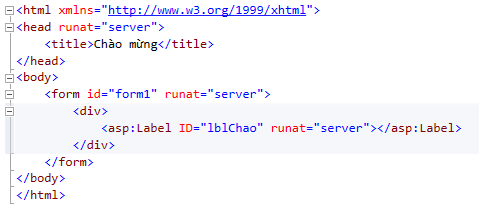
Code trang CookiesDemo.aspx



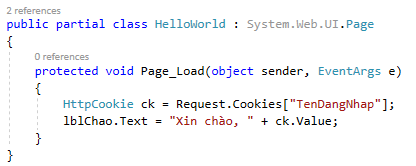
Code behind trang CookiesDemo.aspx: code sự kiện btnLogin\_Click() kiểm tra username và password (ở ví dụ này mặc định chỉ có user admin với mật khẩu admin)



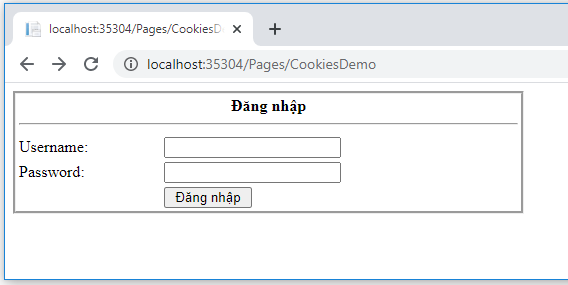
Code trang HelloWorld.aspx: gồm 1 label để chứa câu chào



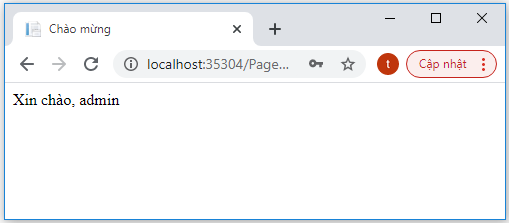
Code behind trang HelloWorld.aspx: lấy giá trị cookie đã tạo để gán cho label



Kết quả khi chạy trang đăng nhập CookiesDemo.aspx



Nhập Username (admin) và mật khẩu (admin) và nhấn Đăng nhập thì được chuyển qua trang HelloWorld như hình dưới:



1. **Đối tượng Session**

Session là một phiên làm việc giữa từng người dùng và web server, nó bắt đầu khi người đó lần đầu tiên truy cập tới 1 trang web trong website và kết thúc khi người đó rời khỏi website hoặc không tương tác với website trong một khoảng thời gian nhất định (time out).

Như vậy tại một thời điểm một website có bao nhiêu người truy cập thì có bấy nhiêu phiên ứng với mỗi người, các phiên này độc lập nhau.

Trong ASP.Net, dùng đối tượng Session để lưu những thông tin tác dụng trong 1 phiên. Khi Session được tạo, đồng thời phương thức Session\_OnStart(); sẽ được thực thi và tương tự khi kết thúc Session, phương thức Session\_OnEnd() sẽ được thực thi. Các sự kiện này được đặt trong file Global.asax.

**Giá trị của biến kiểu session**: có phạm vi trong tất cả các trang ASP.NET của ứng dụng, nhưng không có tác dụng đối với phiên làm việc khác.

**Thuộc tính và phương thức**

Thuộc tính Timeout:

Quy định khoảng thời gian (tính bằng phút) mà Web Server duy trì đối tượng Session nếu người dùng không gửi yêu cầu nào về lại Server. Mặc định là 20 (phút).

Nếu không có yêu cầu nào kể từ lần yêu cầu cuối cùng một khoảng thời gian là <Timeout> phút, đối tượng Session mà Web server cấp cho lần làm việc đó sẽ tự động được giải phóng. Những yêu cầu sau đó được Web server coi như là một người dùng mới, và sẽ được cấp một đối tượng Session mới.

Phương thức Abandon:

Trong khoảng thời gian <Timeout> phút kể từ lần yêu cầu sau cùng của Client, đối tượng Session vẫn được duy trì dù cho dù không có sự tương tác nào của Client. Điều này đồng nghĩa với việc Web server phải sử dụng một vùng nhớ để duy trì đối tượng Session trong một khoảng thời gian tương ứng.

Phương thức Abandon của đối tượng Session sẽ giải phóng vùng nhớ được dùng để duy trì đối tượng Session trên Web Server ngay khi được gọi thực hiện. Những yêu cầu sau đó được Web server coi như là một người dùng mới.

*Lưu ý: nên dùng đối tượng Session một cách hợp lý vì khác với Cookis, Session được lưu trữ trên Web Server và khi không có tác động từ Client thì biến Session vẫn tồn tại cho đến khi hết phiên làm việc (kể cả khi tắt trình duyệt thì Session này vẫn tồn tại trong khoảng thời gian quy định) nên rất tốn bộ nhớ.*

**Sử dụng biến toàn cục với Session**

Tạo biến Session với cú pháp:

Session.Add("Tên\_Biến","Giá trị khởi tạo");

Tên biến phải đặt trong cặp dấu nháy kép. <Giá trị> có thể là xâu ký tự hoặc số...

Hoặc

Session["Tên Biến"] = Giá trị;

Đọc giá trị của một biến sesstion với cú pháp:

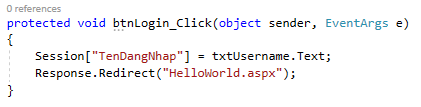
Session.Contents["Tên\_Biến"]

Hoặc

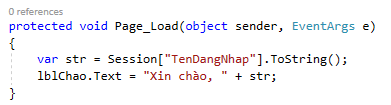
<Biến> = Session["Tên Biến"];

Ví dụ về Session: có thể dùng Session để lưu username thay cho cách dùng Cookie ở ví dụ trong phần đối tượng Cookie như sau:

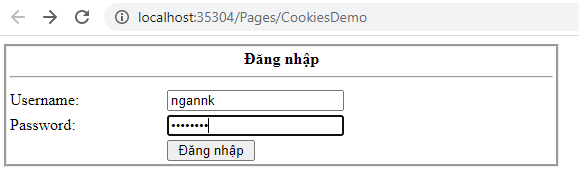
Code behind của trang đăng nhập CookiesDemo.aspx (tạm bỏ phần check username, password)



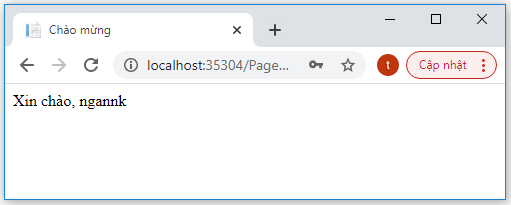
Code behind của trang HelloWorld.aspx



Chạy thử trang CookiesDemo.aspx, nhập username và password sau đó click nút Đăng nhập

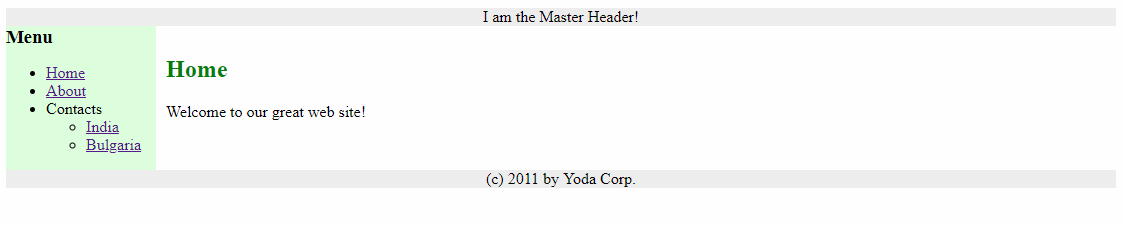


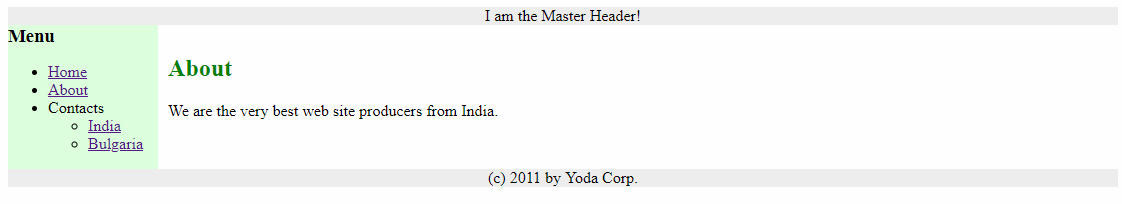
Kết quả khi click nút Đăng nhập

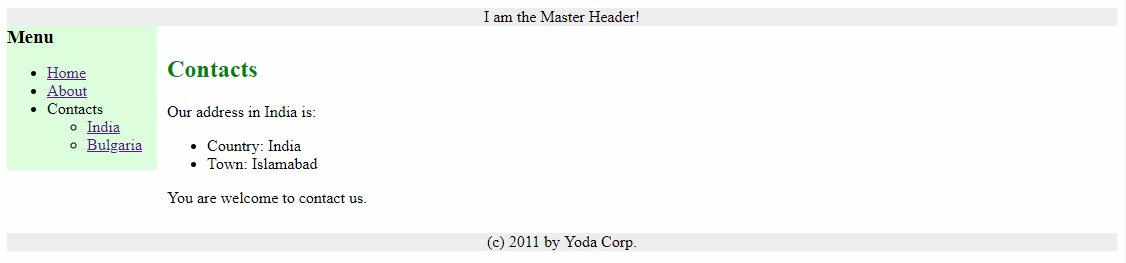


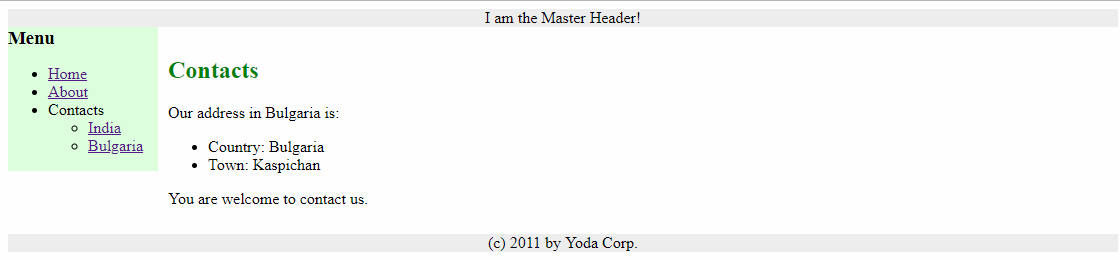
1. **Bài tập**

Ex 1: Tạo trang master pages và trang nội dung như hình:



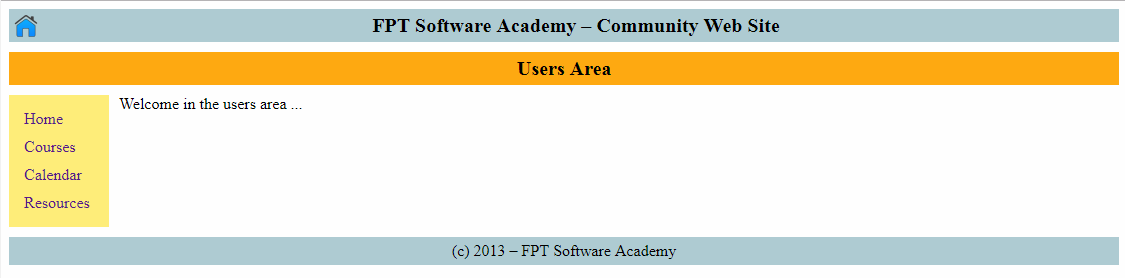


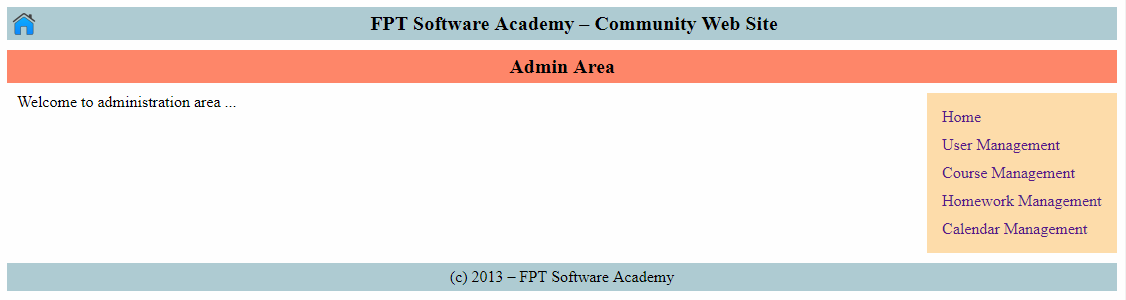




Ex2: Tạo trang master page như hình dưới (sử dụng nested master page)



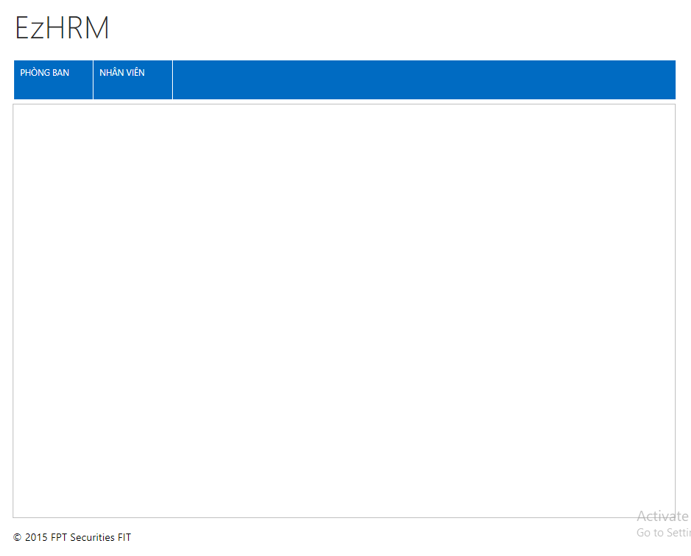


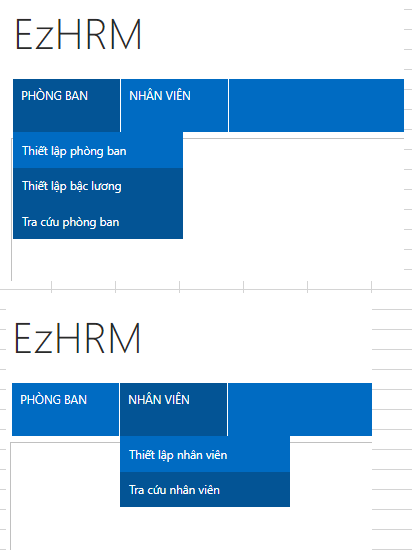


Ex3: Tạo trang Master Page như hình dưới:

- Khi Click vào menu cha sẽ hiện ra list menu con

- Click vào menu con sẽ đến các trang nội dung (các trang nội dung này có thể chứa nội dung bất kỳ)





# CHAPTER VII. Mô hình 3 lớp

1. **ADO.NET**

**Giới thiệu về ADO.Net**

ADO.NET là một bộ thư viện hướng đối tượng (OOP) cho phép tương tác với dữ liệu nguồn. Thông thường thì dữ liệu nguồn là một cơ sở dữ liệu (Database), nhưng cũng có thể là file text, exel hoặc XML. Ở đây ta chỉ xét ADO.NET làm việc với database.

ADO.NET có 2 thành phần chính: Data Provider và DataSets:

**Data Provider**

Data Provider là các thư viện lớp cung cấp chức năng tạo kết nối đến nguồn dữ liệu. Với mỗi nguồn dữ liệu lại cần phải sử dụng một Data Provider khác nhau.

Ví dụ như với SQL Server: loại Data Provider là SqlClient ở namespace System.Data.SqlClient cung cấp khả năng kết nối đến SQL Server.

MySQL: Nếu muốn có loại Data Provider truy cập đến MySQL thì cài đặt package MySql.Data, sẽ có Data Provider MySql.Data.MySqlClient

Oracle: loại Data Provider là Oracle Data Provider ở namespace System.Data.OracleClient

**DataSet**

DataSet là các thư viện lớp (độc lập với Data Provider) tạo ra các đối tượng để quản lý dữ liệu không phụ thuộc nguồn dữ liệu đến từ đâu, đã ở trong ứng dụng (local) hay từ nguồn XML.

DataSet thường gồm nhiều DataTable, trong DataTable lại gồm DataColumn, các ràng buộc, các khóa chính ... Vậy DataSet là sự trừu tượng hóa một CSDL thực.

**Các đối tượng của ADO.Net**

ADO.Net bao gồm nhiều đối tượng để dùng với dữ liệu. Trong đó có một số đối tượng chính như:

**Connection:**

Là một kết nối tới database. Kết nối giúp xác định database server, database name, user name, password, và các tham số cần thiết để kết nối tới database. Một đối tượng Connection được dùng bởi đối tượng Command.

Đối tượng Connection có một số thuộc tính và phương thức như:

* Thuộc tính ConnectionString: truyền vào chuỗi kết nối tới nguồn dữ liệu
* Phương thức Open: mở kết nối
* Phương thức Close: đóng kết nối

**Command:**

Đối tượng Connection là để kết nối đến database. Khi đã kết nối với database rồi ta sẽ thực hiện một số hành động tương tác với database (thêm, sửa, xóa, lấy dữ liệu…). Điều này được thực hiện bởi đối tượng command. Nói ngắn gọn, đối tượng Command dùng để thực thi câu lệnh SQL SQL queries hoặc gọi đến các thủ tục Store procedure.

Một đối tượng command dùng một đối tượng connection để xác định database nào sẽ được truy xuất.

Ta có thể dùng một đối tượng command riêng lẻ để thực thi lệnh trực tiếp, hoặc để gắn một tham chiếu của đối tượng command cho một DataAdapter.

**DataAdapter:**

Là cầu nối trung gian giữa DataSet và Data Source.

Khi dữ liệu mà ta làm việc là read-only, khi ít khi cần thay đổi dữ liệu nguồn hoặc cần lưu trữ dữ liệu tạm thời trong bộ nhớ để hạn chế truy xuất đến Database thì ta có thể dùng đối tượng DataAdapter.

DataAdapter giúp quản lý dữ liệu trong chế độ ngắt kết nối. Data adapter sẽ đổ vào DataSet khi đọc dữ liệu và thực hiện thay đổi dữ liệu một lượt vào database.

**DataReader:**

Đối tượng DataReader trả về dữ liệu là một luồng dữ liệu fast forward-only. Có nghĩa là chỉ có thể lấy dữ liệu từ luồng theo một thứ tự nhất định. Điều này làm tăng hiệu suất thực hiện nhưng không dùng khi cần phải thao tác dữ liệu. Khi cần thao tác dữ liệu thì một DataSet sẽ là đối tượng tốt hơn để làm việc.

**Lưu ý:**

Với mỗi Data Provider khác nhau thì các đối tượng trong ADO.Net cũng có tên khác nhau. Ví dụ nếu dùng Sql Data Provider thì các đối tượng connection, command, DataAdapter, DataReader lần lượt sẽ là SqlConnection, SqlCommand, SqlDataAdapter, SqlDataReader…

DataSet là một đối tượng không phụ thuộc vào Data Provider và dùng được bởi tất cả Data Provider nên nó không cần có tiền tố của Data Provider đứng trước như các đối tượng khác ở trên.

1. **ODP.NET**

ODP.Net là gọi tắt của Oracle Data Provider cho .Net, nó là một Data Provider cung cấp các tính năng để truy cập dữ liệu từ cơ sở dữ liệu Oracle.

Các lớp hỗ trợ của ODP.Net thuộc namespace Oracle.DataAccess.Client được lưu trữ trong Oracle.DataAccess.dll

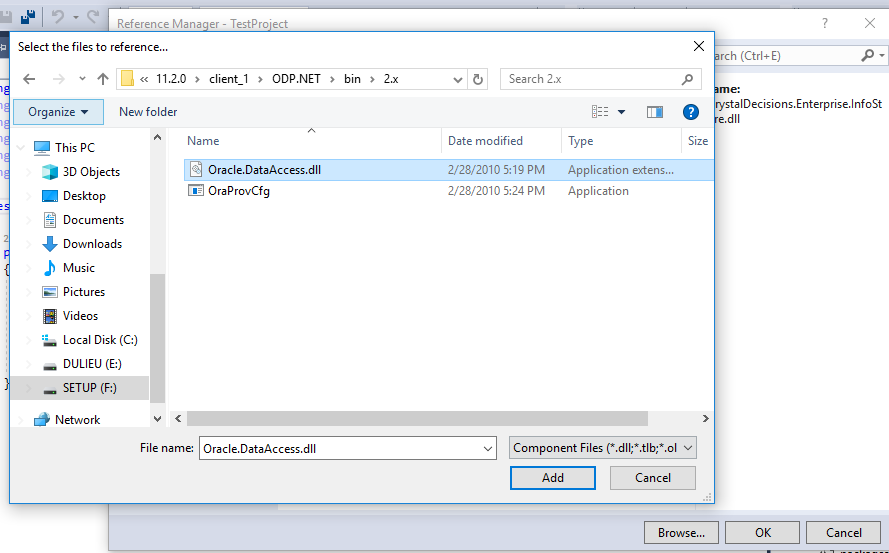
Các đối tượng connection, command, DataAdapter, DataReader tương ứng trong ODP.Net lần lượt là OracleConnection, OracleCommand, OracleDataAdapter, OracleDataReader.

**Các bước kết nối đến cơ sở dữ liệu Oracle bằng ODP.NET**

**Bước 1:** Thêm tham chiếu đến Oracle.DataAccessClient.dll

Chọn Project -> Add Reference (hoặc tại cửa sổ Solution Explorer, click chuột phải vào Reference -> Add Reference

Trong cửa sổ Reference Manage, chọn Browse để đi đến file Oracle.DataAccessClient.dll (trong thư mục ORACLE\_HOME \ ODP.NET \ bin) của Oracle Client đã cài đặt. Add file rồi nhấn Ok.



**Bước 2:** Khai báo một số namespace

Thêm khai báo namespace sau:

using Oracle.DataAccess.Client;  
using Oracle.DataAccess.Types;

**Bước 3:** Tạo kết nối: tạo chuỗi kết nối Connection String và đối tượng kết nối Connection

**Connection String:**

Chuỗi kết nối là 1 tập các thông tin cung cấp cho một thiết lập kết nối (tham số) và có hình thức các cặp key-value cách nhau bằng dấu ";"

Các key cơ bản như: Data Source (tên server); Database (tên Database); User Id; Password…

Chuỗi kết nối có dạng như sau:

"Data Source=ServerAddress; Database=DataBaseName; User Id=Username; Password=myPassword;"

Đối với kết nối với Cơ sở dữ liệu Oracle bằng ODP.Net, ta nên thêm danh sách các Data Source bằng cách chỉnh sửa file tnsnames.ora (trong thư mục ORACLE\_HOME \ network \ admin). Một Data Source sẽ được khai báo địa chỉ server (host, port), server, tên server…

Ví dụ cấu hình tệp tnsnames.ora như được hiển thị bên dưới để kết nối cơ sở dữ liệu DBSV

DBSV =

(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST = 10.26.7.23)(PORT = 1521))

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SERVICE\_NAME = orcl)

)

)

Lúc này chuỗi kết nối chỉ cần khai báo:

"Data Source=DataSourceName; User Id=Username; Password=myPassword;"

Ví dụ chuỗi kết nối đến cơ sở dữ liệu DBSV với username scott và mật khẩu là tiger sẽ là:

"Data Source=DBSV; User Id=scott; Password=tiger";

**Đối tượng Connection:**

Trong ODP.Net lớp kế thừa Connection là OracleConnection.

Tạo một đối tượng OracleConnection bằng từ khóa new và chỉ ra chuỗi kết nối Connection String của nó như sau:

string strConn = "Data Source=DataSourceName; User Id=Username; Password=myPassword;";

OracleConnection oracleConnection = new OracleConnection(strConn);

Ví dụ muốn kết nối đến cơ sở dữ liệu DBSV với username scott và mật khẩu là tiger, khai báo Connection String strConn và đối tượng OracleConnection objConnection như sau:

string strConn = "Data Source=DBSV; User Id=scott; Password=tiger";  
OracleConnection objConnection = new OracleConnection(strConn);

**Bước 4:** Tạo đối tượng Command

Sau khi kết nối vào nguồn dữ liệu, ta cần phải thực hiện thao tác các dữ liệu. Để thao tác được với các dữ liệu ta phải dùng đối tượng Command.

Cú pháp tạo đối tượng Command:

OracleCommand cmd = new OracleCommand();

Thuộc tính của đối tượng Command

* Connection: thiết lập/chỉ đến đối tượng Connection được sử dụng
* CommandType: khai báo kiểu của đối tượng lệnh (lệnh trực tiếp, stored procedure…)
* CommandText: khai báo lệnh thao tác với dữ liệu (câu lệnh truy vấn SQL, tên stored procedure)
* Parameters: các tham số truyền vào cho đối tượng Command (với loại CommandType.StoredProcedure)
* …

Phương thức:

* ExecuteReader : Thực thi câu lệnh CommandText của đối tượng Command và trả về kiểu DataReader (OracleDataReader ).
* ExecuteNonQuery : Thực thi câu lệnh CommandText của đối tượng Command, đây là dạng câu lệnh cập nhật cơ sở dữ liệu (xoá /sửa) nên chỉ trả về số dòng bị ảnh hưởng mà không trả về dòng dữ liệu nào.
* ExecuteScalar : Thực thi câu truy vấn của đối tượng Command và chỉ trả về cột đầu tiên của dòng đầu tiên của kết quả. Các kết quả còn lại bị bỏ qua.

**Tham số Parameters của đối tượng Command:**

Khai báo một tham số với cú pháp:

OracleParameter parameter = new OracleParameter();

Hoặc khai báo một mảng các tham số với cú pháp:

var parameters = new OracleParameter[5];

Các thuộc tính cần chú ý:

* Direction: Giá trị cho biết lọai tham số: Input: (mặc định) Loại tham số đầu vào; InputOutput: Loại tham số đầu vào và ra; Output: Loại tham số đầu ra; ReturnValue: Loại tham số nhận trị trả về.
* OracleDbType: Kiểu dữ liệu của tham số.
* ParameterName: Tên tham số
* Value: Giá trị tham số

Thêm tham số vào đối tượng Command: sử dụng phương thức Add() của thuộc tính Parameters của đối tượng Command

Ví dụ: với đối tượng Connection đã tạo ở bước 3, khai báo đối tượng Command

string strConn = "Data Source=DataSourceName; User Id=Username; Password=myPassword;";

OracleConnection oracleConnection = new OracleConnection(strConn);

OracleCommand cmd = new OracleCommand();

cmd.Connection = oracleConnection;

cmd.CommandText = "Insert into Table Values()"; //Câu lệnh SQL

cmd.CommandType = CommandType.Text;

**Bước 5:** Mở kết nối

Thực hiện kết nối Connection với các Connection đã tạo, sử dụng phương thức open() của đối tượng Connection.

**Bước 6:** Chạy command

Thực hiện các thao tác tới database bằng cách thực thi các phương thức của đối tượng Command

**Bước 7:** Đóng kết nối

Sau khi thực hiện thao tác với database xong thì đóng kết nối lại, sử dụng phương thức close() của đối tượng Connection

Ví dụ: với các đối tượng Connection và Command đã tạo ở bước 3, 4, thực hiện các bước 5, 6, 7

cnn.Open();

cmd.ExecuteNonQuery();

response.write("Thành công!“);

cnn.Close();

**Ví dụ:** truy xuất thông tin về các phòng ban trong hệ thống cơ sở dữ liệu DBSV (đây là hệ thống cơ sở dữ liệu minh họa để quản lý dữ liệu nguồn nhân lực) và đổ ra DataGrid dtgList bằng đoạn mã sau.

string strConn = "Data Source=DBSV; User Id=scott; Password=tiger";  
using (OracleConnection objConnection = new OracleConnection())  
{  
     objConnection.ConnectionString = strConn;  
     try  
     {  
         objConnection.Open();  
         OracleCommand objCommand = new OracleCommand();  
         objCommand.Connection = objConnection;  
         objCommand.CommandText = "select dept\_id, dept\_name, manage\_id from dept";  
         objCommand.CommandType = System.Data.CommandType.Text;  
         OracleDataAdapter objAdapter = new OracleDataAdapter(objCommand);  
         DataTable objTable = new DataTable();  
         objAdapter.Fill(objTable);  
         this.dtgList.DataSource = objTable;  
         objConnection.Close();  
     }  
     catch (Exception ex)  
     {  
         System.Windows.Forms.MessageBox.Show(ex.ToString());  
     }  
     finally  
     {  
         objConnection.Close();  
     }  
}

**Sử dụng Stored procedure**

Store procedures là tập hợp một hoặc nhiều câu lệnh SQL thành một nhóm đơn vị xử lý logic và được lưu trữ trên Database Server. Stored procedure cho tốc độ thực thi câu lệnh nhanh hơn so với các câu lệnh SQL thông thường. Nó được lưu sẵn tại Database Server nên chương trình không cần gửi hàng ngàn dòng lệnh đến Database Server mà chỉ cần gửi một dòng lệnh để thực thi Stored procedure.

Sử dụng Stored procedure không những giúp tốc độ thực thi và xử lý dữ liệu tốt hơn, có thể tái sử dụng các thủ tục đã được lưu mà còn giúp code ứng dụng ngắn gọn hơn, dễ debug hơn.

Để thực thi một Stored procedure đã lưu, ta thực hiện các bước từ 1-7 tương tự như phần trên. Riêng tại bước 4, khai báo các thuộc tính của đối tượng Command như sau:

command.CommandText = "<tên Stored Procedure>";

command.CommandType = CommandType.StoredProcedure;

Khai báo các tham số đầu vào, đầu ra của Stored Procedure (nếu có) thông qua thuộc tính Parameters của đối tượng Command.

**Ví dụ:** truy xuất thông tin về các phòng ban trong hệ thống cơ sở dữ liệu DBSV với stored procedure:

* Tên là sp\_dept\_get
* Đầu vào p\_dept\_name: là điều kiện tìm kiếm theo tên phòng ban, nếu tìm tất cả thì truyền ALL
* Đầu ra là con trỏ rs chứa danh sách các phòng ban theo điều kiện tìm kiếm

Lấy thông tin từ đầu ra của sp và hiển thị lên DataGrid dtgList



1. **Giới thiệu và cấu trúc chung về mô hình 3 lớp**

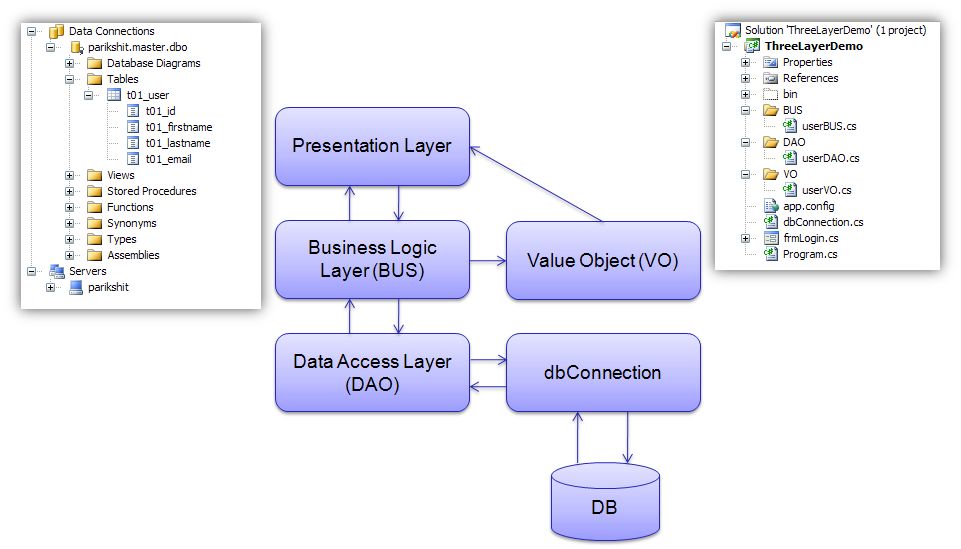
Khi làm việc với các ứng dụng, với những dự án nhỏ thì việc tạo ra một ứng dụng theo ý muốn rất dễ dàng, tuy nhiên khi làm việc với những dự án lớn hơn, cần nhiều người cùng làm hơn thì việc lập trình trở nên phức tạp hơn. Vậy nên để dễ dàng trong việc quản lý, bảo trì và nâng cấp các phần của hệ thống, ta thường nhóm các thành phần có cùng chức năng lại với nhau. Đó là lý do các mô hình lập trình được sinh ra, một trong những mô hình lập trình đó là mô hình 3 lớp (3 Layer).

Mô hình 3 lớp là sự phân chia các thành phần theo chức năng hoặc vai trò một cách logic và cơ bản được chia thành các lớp như sau: Lớp truy cập cơ sở dữ liệu Database Access Layer, Lớp logic nghiệp vụ Business Logic Layer và Lớp giao diện người dùng User Interface Layer.

Ưu điểm của mô hình 3 lớp:

* Code được tường minh hơn: Nhờ vào việc chia ra từng lớp đảm nhận các chức năng khác nhau và riêng biệt như giao diện, xử lý, truy vấn thay vì để tất cả lại một chỗ. Nhằm giảm sự kết dính.
* Dễ bảo trì: Khi cần thay đổi một thành phần của hệ thống thì việc thay đổi này có thể được cô lập trong 1 lớp, hoặc ảnh hưởng đến lớp gần nhất mà không ảnh hưởng đến cả chương trình.
* Dễ phát triển, tái sử dụng: khi muốn thêm một chức năng nào đó thì việc lập trình theo một mô hình sẽ dễ dàng hơn vì chúng ta đã có chuẩn để tuân theo. Và việc sử dụng lại khi có sự thay đổi giữa hai môi trường ( Winform sang Webfrom ) thì chỉ việc thay đổi lại lớp GUI.
* Dễ bàn giao. Nếu mọi người đều theo một quy chuẩn đã được định sẵn, thì công việc bàn giao, tương tác với nhau sẽ dễ dàng hơn và tiết kiệm được nhiều thời gian.
* Dễ phân phối khối lượng công việc. Mỗi một nhóm, một bộ phận sẽ nhận một nhiệm vụ trong mô hình 3 lớp. Việc phân chia rõ ràng như thế sẽ giúp các lập trình viên kiểm soát được khối lượng công việc của mình.

Chi tiết các lớp hơn như hình sau:



**dbConnection**

Lớp này chủ yếu được sử dụng để thực hiện hoạt động truy vấn tới cơ sở dữ liệu như Tìm kiếm, Thêm, Cập nhật, Xóa... Nó cũng kiểm tra xem kết nối cơ sở dữ liệu có đang mở hay không. Nếu kết nối cơ sở dữ liệu không được mở, thì nó sẽ mở kết nối và thực hiện truy vấn cơ sở dữ liệu. Kết quả cơ sở dữ liệu sẽ được nhận và chuyển vào Bảng dữ liệu trong lớp này.

**Database Access Layer**

Lớp truy cập cơ sở dữ liệu (DAO) xây dựng truy vấn dựa trên các tham số nhận được từ Business Logic Layer và chuyển cho lớp dbConnection để thực thi. Kết quả sau khi thực thi sẽ được trả về cho Business Logic Layer.

**Value Object**

Đây đơn giản là lớp với nội dung chỉ là các phương thức GET và SET. Nó chủ yếu được sử dụng để chuyển dữ liệu từ lớp này sang lớp khác. Nó được kết nối trực tiếp với Lớp Business Logic và Lớp Presentation. Như trong sơ đồ, giá trị của đối tượng được SET trong Business Logic Layer và GET ở Presentation Layer.

**Business Logic Layer**

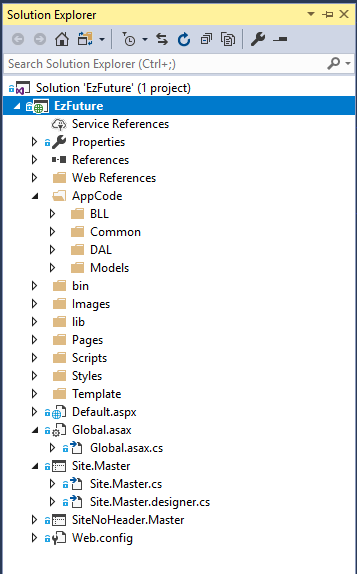
Lớp Logic nghiệp vụ (BUS) hoạt động như một cầu nối giữa Lớp Presentation và DAO. Tất cả các giá trị người dùng nhận được từ lớp Presentation được chuyển tới BUS. Kết quả nhận được từ DAO là các dòng dữ liệu trong bảng nhưng ở BUS, nó sẽ được chuyển đổi thành các Đối tượng giá trị (Value Object). Business Logic Layer (BUS) là lớp quan trọng nhất trong toàn bộ kiến trúc vì nó chứa hầu hết tất cả các logic nghiệp vụ của chương trình. Bất cứ khi nào người dùng muốn cập nhật logic nghiệp vụ của chương trình chỉ cần cập nhật lớp này.

**Presentation Layer**

Lớp Trình bày là lớp duy nhất được kết nối trực tiếp với người dùng. Vì vậy nó cũng là một lớp thực sự quan trọng cho mục đích tiếp thị. Presentation Layer chủ yếu được sử dụng để lấy dữ liệu người dùng và sau đó chuyển nó đến Lớp Business Logic để thực hiện các quy trình tiếp theo và khi dữ liệu được nhận là Value Object thì nó có trách nhiệm trình bày đối tượng giá trị này ở dạng thích hợp mà người dùng có thể hiểu được.

1. **Cấu trúc mô hình 3 lớp trong các ứng dụng internal của FPTS**

Ví dụ về cấu trúc thư mục của một dự án trong FPTS như sau:



Các thư mục cơ bản trong một dự án của FPTS thường là:

Thư mục AppCode chứa các thư mục:

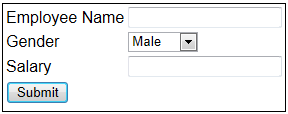
* Common: chứa các thủ tục dùng chung cho project
* Models: bên trong là các class tương ứng với các bảng dữ liệu trong Cơ sở dữ liệu (tương đương với Value Object).
* DAL: (tương ứng với Database Access Layer) chứa các thủ tục kết nối với Database: gọi đến các store procedure để lấy và xử lý dữ liệu phía Database khi được các thủ tục trong BLL yêu cầu, rồi chuyển kết quả lên cho các thủ tục trong BLL.
* BLL: (tương ứng với Business Logic Layer) chứa các thủ tục để xử lý dữ liệu phía ASP.Net sau khi lấy được dữ liệu từ DAL, được sử dụng khi các sự kiện trong trang .aspx gọi đến

Thư mục Pages: chứa các trang aspx, phần code behind khi xử lý các sự kiện trên trang sẽ gọi đến các thủ tục trong BLL (tương ứng với Presentation Layer)

Ngoài ra có các thư mục như Images, Script, Style, Templace: tương ứng để chứa ảnh, file JavaScript, file CSS, các Web User Control dùng cho dự án.

1. **Bài tập**

Ex1: Tạo bảng employee và lập trình một trang web để nhập thông tin nhân viên (sử dụng ODP.Net và store procedure) theo thiết kế sau:



Ex2: Tạo form Thiết lập bậc lương như hình dưới

Yêu cầu:

- Dùng trang master page đã tạo trong ex3 chapter VI làm master page của form này

- Lưới dữ liệu:

+ dùng datagrid

+ Sắp xếp lưới theo mức lương từ thấp đến cao

+ Ngày tạo, Ngày sửa hiển thị theo format 'DD/MM/YYYY HH24:MI:SS'

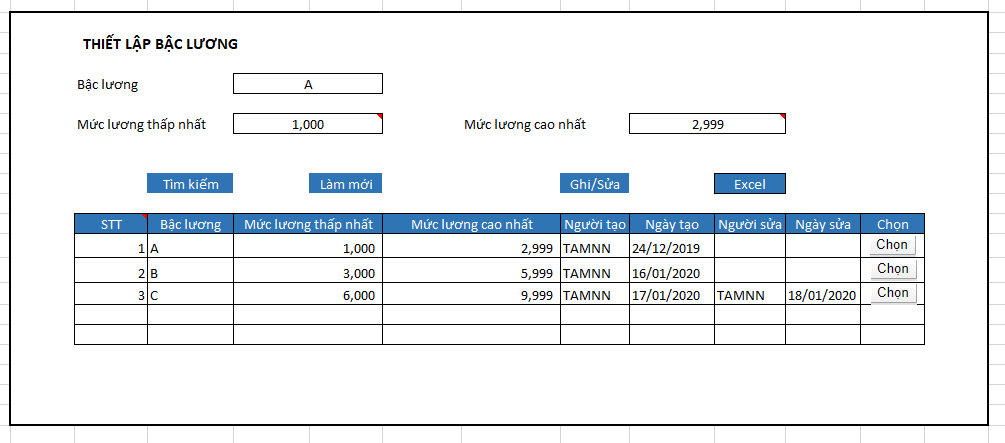
+ Mức lương thấp nhất, Mức lương cao nhất: Tự động thêm dấu phẩy ngăn cách hàng nghìn, triệu

+ Chữ trong các ô header căn giữa

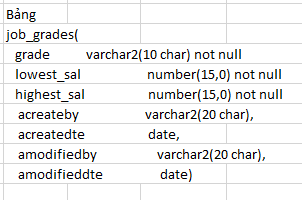
+ Các ô dưới lưới: Số căn phải, chữ căn trái, ngày căn giữa

- Ban đầu button 'Ghi/ Sửa' chỉ hiển thị là 'Ghi', khi click button 'Chọn' thì đẩy dữ liệu dưới lưới lên các ô textbox và nút 'Ghi' chuyển thành nút 'Sửa'

- Khi làm mới: xóa hết dữ liệu trên textbox và lưới, nút 'Sửa' chuyển thành 'Ghi'



Bảng dữ liệu trong oracle:



Ex3: Tạo form Thiết lập phòng ban như hình dưới

Yêu cầu:

- Dùng trang master page đã tạo trong ex3 chapter VI làm master page của form này

- Lưới dữ liệu:

+ dùng datagrid

+ Sắp xếp lưới theo mã phòng

+ Ngày tạo, Ngày sửa hiển thị theo format 'DD/MM/YYYY HH24:MI:SS'

+ Chữ trong các ô header căn giữa

+ Các ô dưới lưới: Số căn phải, chữ căn trái, ngày căn giữa

- Ban đầu button 'Ghi/ Sửa' chỉ hiển thị là 'Ghi', khi click button 'Chọn' thì đẩy dữ liệu dưới lưới lên các ô textbox và nút 'Ghi' chuyển thành nút 'Sửa'

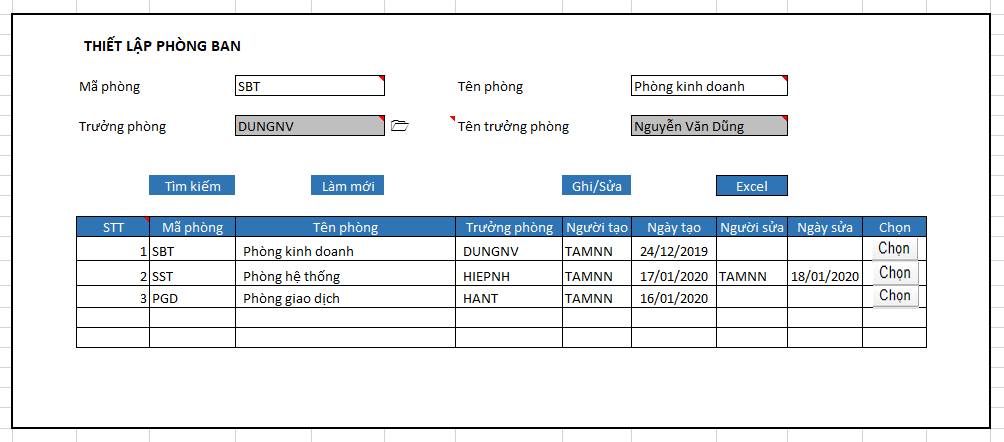
- Khi làm mới: xóa hết dữ liệu trên textbox và lưới, nút 'Sửa' chuyển thành 'Ghi'

- Khi tạo phòng: không cần khai báo trưởng phòng

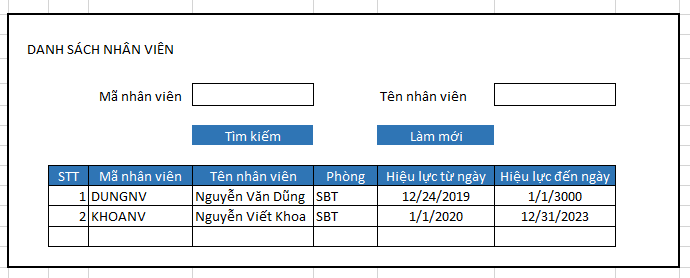
- Khi cập nhật phòng: cập nhật trưởng phòng thì lấy dữ liệu trưởng phòng từ bảng nhân viên

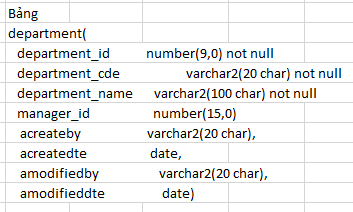
- Khi click vào hình ảnh tệp cạnh textbox trưởng phòng thì mở ra trang popup

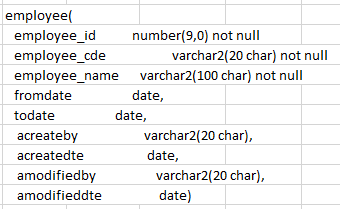
Form Thiết lập phòng ban:



Trang popup: lấy danh sách nhân viên từ bảng nhân viên







Bài tập 2, 3 chi tiết hơn trong file excel