# Debug Visual studio

## Attach to process – ctrl +alt +p

Immediate window – ctrl + alt +i

## Show all brearkpoint – ctrl + alt + b

## To specify breackpoint – chột trái -> ctrl+F10

F11- nhẩy vào trong

Shift + F11 - nhẩy ra ngoài

F10 - nhẩy qua

Ctrl + ‘-‘ – quay ngược lại vị trí trước đó

# Công nghệ sử dụng

|  |  |  |
| --- | --- | --- |
|  | Client | Server |
|  | Angular + javascriptrịc |  |

# HTML

## Thẻ <tr> -<table>

Định nghĩa và sử dụng

* Tag <tr> được dùng để tạo một hàng trong bảng HTML.
* Một tag <tr> được sử dụng bên trong [<table>](https://hocwebchuan.com/reference/tag/tag_table.php), và thường kèm với [<th>](https://hocwebchuan.com/reference/tag/tag_th.php), tất cả được chứa bên trong [<td>](https://hocwebchuan.com/reference/tag/tag_td.php), trong đó:
  + <th> xác định phần tử tiêu đề của <table>
  + <td> xác định phần tử nội dung của <table>
* Tag <tr> có thể chứa một hoặc nhiều <th> và <td>

**Sự khác nhau giữa HTML và XHTML**

|  |  |  |
| --- | --- | --- |
| **HTML4.01** | **XHTML1.0** | **XHTML1.1** |
| Không có sự khác biệt | | |

Cấu trúc

<tr></tr>

Ví dụ

**Html viết:**

<table>  
<tr>  
<th>A</th> // thẻ th để làm tiêu đề  
<th>B</th>  
</tr>  
<tr>  
<td>1</td>  
<td>5</td>  
</tr>  
<tr>  
<td>9</td>  
<td>10</td>  
</tr>  
</table>

**Hiển thị trình duyệt:**

|  |  |
| --- | --- |
| **A** | **B** |
| 1 | 5 |
| 9 | 10 |

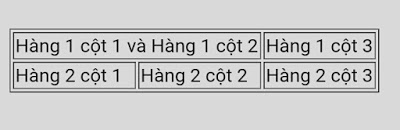
### Thuộc tính colspan và rowspan cho thẻ Table | Học HTML

[](https://4.bp.blogspot.com/-3qtc4AvqKU4/WMuO-mbKmuI/AAAAAAAAAEw/13_Juk-WKpEpN8YHIPjoHV_xm43qJY9cwCPcB/s1600/hoc_html_2.jpg)

Ở bài này chúng tôi sẽ giới thiệu đến các bạn hai thuộc tính colspan và rowspan cho thẻ tạo bảng table.  
  
Colspan: **Thuộc tính colspan** dùng để nối hai hoặc nhiều ô với nhau tính từ ô đặt thẻ này đi tới phía trước.  
Ví dụ: áp dụng thuộc tính colspan vào bảng table

<table border= "1">  
    <tr>  
        <td colspan="2">Hàng 1 cột 1 và Hàng 1 cột 2</td>  
        <td>Hàng 1 cột 3</td>  
    </tr>  
    <tr>  
        <td>Hàng 2 cột 1</td>  
        <td>Hàng 2 cột 2</td>  
        <td>Hàng 2 cột 3</td>  
    </tr>  
</table>

Kết quả khi chạy ví dụ trên

[](https://2.bp.blogspot.com/-caXCzrb4WPI/WNsL6H2qPlI/AAAAAAAAAJ0/IS1tY6we7EIICHql2qlplhPEyFudC1a0ACLcB/s1600/IMG_20170329_081913.JPG)

Rowspan: **Thuộc tính rowspan** dùng để nối hai hoặc nhiêu ô với nhau tính từ ô đặt thẻ đi xuống phía dưới.  
  
Ví dụ: áp dụng thuộc tính rowspan và bảng table

<table border= "1">  
    <tr>  
        <td rowspan="2">Hàng 1 cột 1</td>  
        <td>Hàng 1 cột 3</td>  
        <td>Hàng 1 cột 3</td>  
    </tr>  
    <tr>  
        <td>Hàng 2 cột 2</td>  
        <td>Hàng 2 cột 3</td>  
    </tr>  
</table>

Kết quả khi chạy ví dụ trên

[](https://3.bp.blogspot.com/-MhGE9dNedeA/WNsL6n3KhjI/AAAAAAAAAJ4/_ELwLtYrW-kX0NzUCHuIpOm1VNvteRDMgCLcB/s1600/IMG_20170329_081934.JPG)

Giá trị của 2 thuộc tính ở trên là số cột hoặc hàng cần gọp.  
  
✴ Lưu ý: hai thuộc tính trên bạn phải đằt nó nằm bên trong cặp thẻ td như ở 2 ví dụ trên.  
  
Và bạn cũng có thể kết hộp 2 thuộc tính trên và cùng một bảng table như ở ví dụ sau.

<table border="1" cellspacing="0" cellpadding="5">  
  
    <tr>  
        <td rowspan="2">Hàng 1 cột 1</td>  
        <td colspan= "2">Hàng 1 cột 3</td>  
  
  
    </tr>  
    <tr>  
        <td>Hàng 2 cột 2</td>  
        <td>Hàng 2 cột 3</td>  
    </tr>  
</table>

Kết quả khi chạy ví dụ trên

[](https://4.bp.blogspot.com/-w1niqakhrGc/WNsL8Lbf0QI/AAAAAAAAAJ8/pyWH5t1ve8o5pwh5tuDRuCWYzD69-4WWACLcB/s1600/IMG_20170329_081948.JPG)

### Kỹ thuật vẽ bảng

table tr:last-child td {

1.  border-bottom-width: 1px; // vì từng ô bị trồng lên nhau nên xóa border-bottom nhưng cell ở tr cuối thì đữ lại

}

table td {

1. 
2.  border: 1px solid #dee2e6;
3. ~~ border-bottom-width: 0;~~
4.  border-left-width: 0;

}

**! Tương tự với: border-left-width**

## Thẻ <FORM>

url tham khảo: <https://timoday.edu.vn/bai-25-the-form-trong-html/>

### action

**Thuộc tính action** xác định hành động được thực hiện khi form được gửi đi khi người sử dụng kích nút submit. Cách phổ biến nhất khi gửi dữ liệu của form tới server là sử dụng một nút submit. Thông thường form được gửi tới một trang web chạy trên máy chủ web. Trong ví dụ trên, một script chạy ở phía server được xác định để xử lý  form đã gửi:

<form **action="action\_page.php**">

Nếu thuộc tính action bỏ qua, action sẽ được thiết lập tới trang hiện tại.

### method

Thuộc tính **method**xác định kiểu phương thức HTTP (**GET**hoặc **POST**) được sử dụng gửi dữ liệu trên form:

<form action="action\_page.php" **method="get"**>

hoặc:

<form action="action\_page.php" **method="post"**>

#### get – sử dụng khi nào?

Bạn có thể sử dụng GET (mặc định phương thức này). Sử dụng GET nếu việc gửi form là thụ động (giống như bạn thực hiện truy vấn trên các máy tìm kiếm) và dữ liệu không cần mã hoá, không chứa các thông tin nhạy cảm như mật khẩu, v.v. Khi bạn sử dụng GET, dữ liệu form sẽ bị nhìn thấy trên thanh địa chỉ của trang như ví dụ dưới:

action\_page.php?txtHoDem=Trần+Hùng&txtTen=Dũng

|  |  |
| --- | --- |
|  | GET là phù hợp nhất với số lượng dữ liệu ít. Giới hạn về kích thước dữ liệu GET được thiết lập trong trình duyệt của bạn. |

#### post – sử dụng khi nào?

Bạn nên sử dụng POST: Trong trường hợp nếu form cập nhật dữ liệu hoặc dữ liệu trên form gửi đi bao gồm các thông tin nhạy cảm như mật khẩu, mã thẻ ngân hàng,v.v. POST cung cấp cơ chế bảo mật hơn bởi vì dữ liệu được gửi đi không được hiển thị trên thanh địa chỉ của trang.

### name

Để lấy được dữ liệu đúng khi gửi đi, mỗi trường phải có một thuộc tính name. Ví dụ này sẽ chỉ gửi trường “Họ đệm”, theo bạn tại sao lại không gửi trường “Tên”?:

Chỉ định một tên được sử dụng để nhận diện các thành phần trên form (với DOM sử dụng: document.forms.name).

Ví dụ

<!DOCTYPE html>

<html>

<body><form action="action\_page.php">

Họ đệm:<br>

<input type="text" name="txtHoDem" value="Trần Hùng"><br>

Tên:<br>

<input type="text" value="Dũng"><br>

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

</form>

</body>

</html>

### <fieldset> **<legend>**

Phần từ <fieldset> nhóm dữ liệu liên quan trong một form. Phần tử <legend> xác định một phụ đề cho phần tử <fieldset>.

vd: [file:///C:/Users/BlueMen/Desktop/Lap\_trinh\_web/Unit1.html](../../Users/BlueMen/Desktop/Lap_trinh_web/Unit1.html)

<form action="action\_page.php">

        <fieldset>

            <legend>Thông tin cá nhân:</legend>

            Họ đệm:<br>

            <input type="radio" name="firstname" value="Dũng"><br> Tên:

            <br>

            <input type="radio" name="lastname" value="Trần Hùng"><br>

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

        </fieldset>

        <fieldset>

            <input type="radio" name="firstname1" value="Dũng1"><br> Tên:

            <br>

            <input type="radio" name="lastname1" value="Trần Hùng1"><br>

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

        </fieldset>

    </form>

### autocomplete\*

### enctype\*

### novalidate\*

### targetse\*

### thẻ <output>

## Thẻ <INPUT>

url tham khảo: <https://timoday.edu.vn/bai28-cac-thuoc-tinh-cua-the-input/#Thuoc_tinh_formaction>

### type

#### text

**<input type=”text”>** sử dụng cho nhập dữ liệu đầu vào cho đầu vào dạng văn bản (một dòng):

#### password

**<**input type=”password”> định nghĩa một trường password:

Các kí tự trong trường password được ẩn đi (nó được mã hóa hiển thị thành hình sao hoặc các chấm tròn).

#### Submit <important!>

[<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input)các yếu tố của loại **submit**được hiển thị dưới dạng các nút. Khi [click](https://developer.mozilla.org/en-US/docs/Web/Events/click)sự kiện xảy ra (thường là do người dùng nhấp vào nút), [tác nhân người dùng sẽ](https://developer.mozilla.org/en-US/docs/Glossary/user_agent) cố gắng gửi biểu mẫu đến máy chủ.

<input type="submit" value="Send Request">

Nếu bạn không chỉ định value, nút sẽ có nhãn mặc định, được chọn bởi tác nhân người dùng. Nhãn này có thể là một cái gì đó dọc theo dòng "Gửi" hoặc "Gửi truy vấn." Dưới đây là ví dụ về nút gửi với nhãn mặc định trong trình duyệt của bạn:

<input type="submit">

**<input type=”submit”>** định nghĩa một nút để gửi dữ liệu từ form người sử dụng nhập tới nơi xử lý dữ liệu của form này (**form-handler**)

Form-handler thường mà một trang chạy ở phía server, được viết bằng các ngôn ngữ ở phía máy chủ như PHP, ASP.NET, JSP … cho xử lý dữ liệu đầu vào.

Form-handler được chỉ định bởi các thuộc tính  “action” trong form đó:

Ví dụ

<html>

<body>

<form action="action\_page.php">

Họ đệm:<br>

<input type="text" name="txtHoDem" value="Trần Hùng"><br>

Tên:<br>

<input type="text" name="txtTen" value="Dũng"><br>

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

</form>

</body>

</html>

Nếu bạn kích nút “Submit” dữ liệu trên form sẽ được gửi tới trang “action\_page.php”. Phía máy chủ sẽ thực hiện xử lý dữ liệu gửi và đưa ra đầu ra. Về xử lý phía server bạn có thể sử dụng [ngôn ngữ lập trình PHP](http://timoday.edu.vn/category/khoa-hoc/php/) Đoạn code HTML ở trên sẽ được hiển thị trên trình duyệt

#### radio

**<input type=”radio”>** định nghĩa một nút  **radio.**

Các nút Radio cho phép người sử dụng chọn duy nhất một lựa chọn trong danh sách các lựa chọn:

<input type="radio" name="gender" value="male" checked> Male<br>

Các radio trong cùng 1 form sẽ được lựa cọn duy nhất 1 radio!

#### checkbox

**<input type=”checkbox”>** định nghĩa một nút **checkbox**.

Checkboxes cho phép người sử dụng chọn không hoặc nhiều lựa chọn trong các lựa chọn đưa ra .

#### button

**<input type=”button”>** định nghĩa một **button**:

<input type="button" onclick="alert('Website Tìm ở đây chào bạn!')" value="Kíchh Tôi!"></body>

! nên chọn botton thay vì input type=”botton”

#### Nunber

Thẻ**<input type=”number”>** được sử dụng cho trường dữ liệu đầu vào có chứa các giá trị là số.

Bạn có thể cài đặt các giới hạn trong kiểu đầu vào này.

<input type="number" name="quantity" min="1" max="5">

#### date (hỗ trợ bảng chọn)

Thẻ **<input type=”date”>** được sử dụng khi trường dữ liệu đầu vào chứa các giá trị là ngày.

<input type="date" name="bday" max="1979-12-31">

#### color (hỗ trợ bảng chọn)

Thẻ **<input type=”color”>** được sử dụng khi trường dữ liệu đầu vào chứa các giá trị là màu sắc.

#### Range

Thẻ **<input type=”range”>** được sử dụng khi trường dữ liệu đầu vào chứa dữ liệu là giá trị trong phạm vi nhất định.

<input type="range" name="points" min="0" max="10">

#### month

Thẻ **<input type=”month”>** cho phép người sử dụng có thể lựa chọn một tháng và năm.

#### week

Thẻ **<input type=”week”>** cho phép người sử dụng lựa chọn tuần và năm.

#### time

Thẻ **<input type=”time”>**cho phép người dùng lựa chọn một thời gian (không có múi giờ).

#### datetime-local (hỗ trợ bảng chọn)

Thẻ **<input type=”datetime-local”>** cho phép người sử dụng lựa chọn ngày và thời gian (không có múi giờ).

#### email

Thẻ **<input type=”email”>** được sử dụng cho các trường dữ liệu đầu vào có chứa một địa chỉ e-mail.

Phụ thuộc vào sự hỗ trợ của các trình duyệt, địa chỉ e-mail có thể được tự động xác nhận khi gửi đến.

Một vài smartphones có thể nhận ra các loại email, và bổ sung thêm “.com” vào bàn phím để phù hợp với dữ liệu đầu vào của email.

#### search

Thẻ **<input type=”search”>**được sử dụng cho phép người dùng tìm kiếm các trường (một trường search hoạt động thông thường như một trường text).

#### Tel

Thẻ **<input type=”tel”>** được sử dụng cho các trường đầu vào có dữ liệu chứa một số điện thoại.

Phần tử  tel hiện nay chỉ được hỗ trợ trong trình duyệt Safari 8.

#### url

Thẻ **<input type=”url”>** được sử dụng cho các trường đầu vào mà chứa địa chỉ URL.

Phụ thuộc vào sự hỗ trợ của các trình duyệt, trường địa chỉ url có thẻ tự động xác nhận khi gửi đến.

Một vài smartphones nhận ra kiểu đầu vào url, và bổ sung thêm “.com” vào bàn phím để phù hợp với địa chỉ đầu vào url.

#### reset

reset lại các phần tử trong thẻ form có đi kèm event reset. [vd xem dưới đây](#_onreset)

#### giới hạn đầu vào của các thẻ input

chủ yếu với các thẻ number , range,..

|  |  |
| --- | --- |
| **Thuộc tính** | **Mô tả** |
| disabled | Xác định trường dữ liệu đầu vào bị vô hiệu hóa |
| max | Xác định giá trị lớn nhất cho trường dữ liệu đầu vào |
| maxlength | Xác định số lượng kí tự tối đa cho trường dữ liệu đầu vào |
| min | Xác định giá trị nhỏ nhất cho trường dữ liệu đầu vào |
| pattern | Xác định biểu thức chính quy để kiểm tra giá trị cho trường dữ liệu đầu vào |
| readonly | Xác định trường dữ liệu đầu vào là chỉ đọc (không thể thay đổi) |
| required | Xác định các yêu cầu của trường dữ liệu đầu vào(trường không thể để trống) |
| size | Xác định độ rộng (theo các kí tự) của trường dữ liệu đầu vào |
| step | Xác định các khoảng số hợp lí cho các trường dữ liệu đầu vào |
| value | Xác định giá trị mặc định cho trường dữ liệu đầu vào |

### readonly

Thuộc tính **readonly** chỉ đinh phần tử input chỉ có thể đọc (không thể thay đổi dữ liệu trên phần tử đó):

### disabled

Thuộc tính **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

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

(độ rộng của khung input, size càng nhỏ thì khung càng ngắn)

### maxlength

Thuộc tính **maxlength** chỉ định độ dài tối đa cho phép của trường input:

Vd: cho nhập tối đa 10 ký tự

<input type="text" name="firstname" maxlength="10">

### value autocomplate

Thuộc tính autocomplete quy định một form hoặc một thẻ input trong form có được sử dụng chức năng tự động điền, gợi ý thông tin hay không.

Khi thuộc tính autocomplete có giá trị “on”, trình duyệt sẽ tự động gợi ý những giá trị dựa trên thông tin người dùng nhập vào.

**Mẹo:** Bạn có thể thiết lập giá autocomplete là “on” cho thẻ form và “off” cho một vài thẻ input hoặc ngược lại.

Thuộc tính autocomplete có thể sử dụng trong thẻ <form> và thẻ <input> với các kiểu: text, search, url, tel, email, password, datepickers, range và color.

url demo: <https://webvn.com/example/vi-du-thuoc-tinh-autocomplete/>

### placeholder

Thuộc tính placeholder được sử dụng để đưa ra những gợi ý cho người dùng trước khi nhập liệu vào thẻ.

Thuộc tính placeholder được sử dung trong thẻ input với các type có giá trị: text, search, url, tel, email và password.

Thuộc tính placeholder trong thẻ input không được hỗ trợ trong Internet Explorer 9 và các phiên bản trước.

url demo: <https://webvn.com/example/vi-du-thuoc-tinh-placeholder/>

### for

vd: <label for="username">Click me</label>

<input type="text" id="username">

Liên kết thẻ input vào trong label cho phép vùng nhấn tăng lên,

* For giống tên thuộc tính id của thẻ input

### form

Thuộc tính form form để chỉ định một hay nhiều form mà chứa  phần tử <input> này. **Mách nhỏ:** Để ám chỉ nhiều hơn một form, sử dụng khoảng trắng để phân tách danh sách các id của form.

Thẻ sẽ được thêm vào dưới cùng của form

Ví dụ

Trường Last name được đặt bên ngoài form HTML (nhưng vẫn là một phần của form):

Thẻ input ám chỉ là 1 thành phần của form 1, nếu muốn thêm vào form có id là form2, form3 thì ta viết:

Last name: <input type="text" name="lname" form="form1 form2 form3">

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php" id="form1">

First name: <input type="text" name="fname"><br>

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

</form>

<p>Trường "Last name" bên dưới là ở bên ngoài phần tử form nhưng nó vẫn là một phần của form.</p>

Last name: <input type="text" name="lname" form="form1">

</body>

</html>

### formaction

Thuộc tính formaction chỉ định tới đường dẫn của một file mà sẽ xử lý dữ liệu của các control input khi form được submit. Thuộc tính formaction viết chồng với thuộc tính action của phần tử <form>. Thuộc tính formaction được sử dụng cho các thẻ input kiểu type=”submit” và type=”image”.

**Ví dụ**

Một form HTML với hai nút submit, thực hiện hai hành động khác nhau:

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php">

First name: <input type="text" name="fname"><br>

Last name: <input type="text" name="lname"><br>

<input type="submit" value="Submit"><br>

<input type="submit" formaction="demo\_admin.php" value="Submit as admin">

</form>

<p><strong>Chú ý:</strong> Thuộc tính formaction của thẻ input không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE.</p>

</body>

</html>

### formenctype

Thuộc tính formenctype xác định cách dữ liệu form nên được mã hoá khi gửi dữ liệu tới server (chỉ cho form với method=”post”). Thuộc tính formenctype viết chồng với thuộc tính enctype của thẻ <form>. Thuộc tính formenctype được sử dụng với các thẻ input kiểu type=”submit” và  type=”image”.

**Ví dụ**

Gửi dữ liệu của form với mã hoá mặc định (cho nút submit thứ nhất) và mã hoá kiểu “multipart/form-data” (cho nút submit thứ hai):

<!DOCTYPE html>

<html>

<body>

<form action="demo\_post.php" method="post">

First name: <input type="text" name="fname"><br>

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

<input type="submit" formenctype="multipart/form-data" value="Submit as Multipart/form-data">

</form>

<p><strong>Chú ý:</strong> Thuộc tính formenctype của thẻ input không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE.</p>

</body>

</html>

### formmethod

Thuộc tính formmethod định nghĩa phương thức HTTP cho gửi dữ liệu của form tới một URL thực hiện nó (đường dẫn file chứa code server). Thuộc tính formmethod viết chồng thuộc tính method của phần tử <form>. Thuộc tính formmethod có thể được sử dụng với <input> kiểu type=”submit” and type=”image”.

**Ví dụ**

Nút submit thứ hai viết chồng method HTTP của form:

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php" method="get">

First name: <input type="text" name="fname"><br>

Last name: <input type="text" name="lname"><br>

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

<input type="submit" formmethod="post" formaction="demo\_post.php" value="Submit using POST">

</form>

<p><strong>Chú ý:</strong> Thuộc tính formmethod của thẻ input không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE.</p>

</body>

</html>

### formnovalidate

Thuộc tính novalidate là một thuộc tính kiểu boolean. Khi xuất hiện trong phần tử <input> thì phần tử đó sẽ không bị kiểm tra tính hợp lệ khi gửi dữ liệu. Thuộc tính formnovalidate viết chồng thuộc tính novalidate của phần tử <form>. Thuộc tính formnovalidate có thể được sử dụng với thẻ <input> kiểu type=”submit”.

Ví dụ  
Một form với hai nút submit (với việc có và không có kiểm tra dữ liệu hợp lệ):

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php">

E-mail: <input type="email" name="email"><br>

<input type="submit" value="Submit"><br>

<input type="submit" formnovalidate value="Submit without validation">

</form>

<p><strong>Chú ý:</strong> Thuộc tính formnovalidate của thẻ input không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE hoặc Safari.</p>

</body>

</html>

### name

tên biến, chính là key trong key= value khi gửi dũ liệu

### formtarget

Thuộc tính *formtarget* chỉ định một tên hoặc một từ khoá mà cho biết nơi để hiển thị kết quả khi nhận được sau khi submit form. Thuộc tính *formtarget* viết đề thuộc tính *target* của phần tử <form>. Thuộc tính formtarget có thể được sử dụng với thẻ <input> kiểu type=”submit” và type=”image”.

Ví dụ

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php">

First name: <input type="text" name="fname"><br>

Last name: <input type="text" name="lname"><br>

<input type="submit" value="Submit as normal">

<input type="submit" formtarget="\_blank" value="Submit to a new window/tab">

</form>

<p><strong>Chú ý:</strong> Thuộc tính formtarget không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE.</p>

</body>

</html>

### height và width

Thuộc tính height và width chỉ định chiều rộng và cao của phần tử <input>. Thuộc tính height và width chỉ được sử dụng với <input type=”image”>.

|  |  |
| --- | --- |
| [Mách nhỏ](https://i1.wp.com/timoday.edu.vn/wp-content/uploads/2015/09/lamp.jpg) | Luôn luôn chỉ định kích thước của ảnh. Nếu trình duyệt không biết kích thước, trang nhấp nháy khi loạt các ảnh. |

Ví dụ

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php">

First name: <input type="text" name="fname"><br>

Last name: <input type="text" name="lname"><br>

<input type="image" src="img\_submit.gif" alt="Submit" width="48" height="48">

</form>

</body>

</html>

### list (~combobox)

Thuộc tính list tham chiếu đến phần tử <datalist> mà chứa các giá trị trị tuỳ chọn định nghĩa trước cho một phần tử <input>.

Giống như 1 combobox

**Ví dụ**

Phần tử <input> với các giá trị định nghĩa trước trong một <datalist>:

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php" method="get">

<input list="browsers" name="browser">

<datalist id="browsers">

<option value="Internet Explorer">

<option value="Firefox">

<option value="Chrome">

<option value="Opera">

<option value="Safari">

</datalist>

<input type="submit">

</form>

<p><b>Chú ý:</b> Thẻ datalist không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE hoặc Safari.</p>

</body>

</html>

#### <option> của <datalist>

Khác 1 chút so với [<option>](#_<option>) của [<select>](#_Thẻ_<SELECT>)

không sử dụng nội dung bên trong thẻ <option>

không cần thẻ đóng </option>

##### value

option sẽ hiển thị nội dung chính từ value. thẻ option không có value sẽ hiển thị nội dung bên trong thẻ <option>nôi dung</option>

##### label

nội dung của thẻ label sẽ được hiển thị bên dưới nội dung của thẻ value khi chúng ta show phần datalist, giống như phần mô tả vậy

##### title và selected

không có ý nghĩa trong này

##### disabled

thay vì làm mờ và không thể thao tác thì option đó sẽ mất muôn trong datalist

### multiple

Thuộc tính multiple là một thuộc tính kiểu boolean. Khi xuất hiện, phần tử sẽ cho người sử dụng nhập hơn một giá trị trong phần tử<input>. Thuộc tính multiple làm việc với các kiểu input sau: email và file.

Ví dụ

Một trường upload file chấp nhận nhiều giá trị, có thể lấy 1 lúc nhiều file, cò không khai báo multiple thì chỉ chọn được 1 file:

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php">

Select images: <input type="file" name="img" multiple>

<input type="submit">

</form>

<p>Thử chọn nhiều hơn một file khi browser các file.</p>

</body>

</html>

### min và max

Thuộc tính min và max xác định giá trị nhỏ nhất và lớn nhất cho một phần tử <input>. Các thuộc tính min và max làm việc với các kiểu thẻ <input> sau: number, range, date, datetime, datetime-local, month, time and week.

Ví dụ

Các phần tử <input> với các giá trị min và max, xem lại [bài 27](http://timoday.edu.vn/bai-27-cac-kieu-input-trong-html/):

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php">

Enter a date before 1980-01-01:

<input type="date" name="bday" max="1979-12-31"><br>

Enter a date after 2000-01-01:

<input type="date" name="bday" min="2000-01-02"><br>

Quantity (between 1 and 5):

<input type="number" name="quantity" min="1" max="5"><br>

<input type="submit">

</form>

<p><strong>Chú ý:</strong> Thuộc tính max và min của thẻ input không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE hoặc Firefox.</p>

<p><strong>Chú ý:</strong> Thuộc tính max và min sẽ không làm việc với ngày tháng và thời gian trong Internet Explorer 10.</p>

</body>

</html>

### pattern

huộc tính pattern chỉ định một biểu thức chính quy (regular expression) mà giá trị của phần tử <input> sẽ được kiểm tra lại. Thuộc tính pattern làm việc với các loại thẻ <input> sau: text, search, url, tel, email, và password.

**Mách nhỏ:** Sử dụng thuộc tính title để trợ giúp thông tin pattern cho người sử dụng.

**Ví dụ**

Một trường input có thể chỉ chứa 3 ký tự (không phải kiểu số hoặc các ký tự đặc biệt):

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php">

Country code: <input type="text" name="country\_code" pattern="[A-Za-z]{3}" title="Three letter country code">

<input type="submit">

</form>

<p><strong>Chú ý:</strong> Thuộc tính pattern của thẻ input không hỗ trợ trong Internet Explorer 9 và các phiên bản cũ hơn của IE hoặc Safari.</p>

</body>

</html>

### required

Thuộc tính required là một thuộc tính kiểu boolean. Khi xuất hiện nó xác định rằng trường input không được để trống trước khi submit form. Thuộc tính required làm việc với thẻ <input> kiểu: text, search, url, tel, email, password, date pickers, number, checkbox, radio, và file.

### step

Thuộc tính step xác định bước nhảy của một giá trị số cho phần tử <input>. Ví dụ: nếu step=”3″, các số có thể là -3, 0, 3, 6, v.v.  
**Mách nhỏ:** Thuộc tính step có thể được sử dụng cùng với các thuộc tính max và min để tạo khoảng các giá trị hợp lệ. Thuộc tính step làm việc với các thẻ <input> kiểu: number, range, date, datetime, datetime-local, month, time và week.

! thường dùng khi nhập giá trị tiền tệ với step 1000. Vì 1000 VNĐ là đơn vị nhỏ nhất

**Ví dụ**

Trường input với chỉ định bước nhảy của một giá trị số:

<input type="number" name="points" step="3">

### data-lpignore

### data-testid

### role

### aria-autocomplete

### aria-expanded

### aria-control

## Thẻ <SELECT>

Tag <select> được dùng để tạo một danh sách chọn lựa (danh sách thả xuống).

Tag [<option>](https://hocwebchuan.com/reference/tag/tag_option.php) bên trong thành phần select, định nghĩa giá trị tùy chọn trong danh sách.

Tag <select> và <option> được thường được đặt trong [<form>](https://hocwebchuan.com/reference/tag/tag_form.php) cho người dùng lựa chọn danh sách.

Cách sử dụng: <select thuoctinh="giatri"></select>

### disabled

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuộc tính** | **Giá trị** | **Ví dụ** | **Mô tả** |
| [disabled](_blank) | disabled | disabled="disabled" | Xác định danh sách thả xuống không hiển thị. |

### multiple

thay vì hiển thị 1 ô , nó sẽ hiển thị tất cả các tùy chọn

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuộc tính** | **Giá trị** | **Ví dụ** | **Mô tả** |
| [multiple](_blank) | multiple | multiple="multiple" | Hiển thị nhiều tùy chọn. |

**Khi lấy giá trị :**  var foo = $('#multiple').val(); => foo là 1 list

### name

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuộc tính** | **Giá trị** | **Ví dụ** | **Mô tả** |
| [name](_blank) | name | name="selName" | Xác định tên cho danh sách lựa chọn. |

### size

thay vì hiển thị 1 ô nó sẽ hiển thị theo số size

theo ví dụ dưới thì sẽ nhìn thấy 1 lúc 3 tùy chịn

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuộc tính** | **Giá trị** | **Ví dụ** | **Mô tả** |
| [size](_blank) | Số | size="3" | Xác định số tùy chọn có thể thấy trong danh sách lựa chọn. |

### <option>

#### disabled

Giống như Thuộc tính **disabled** của trường input. Nó 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).

#### label

nếu không có text trong thẻ option thì nội dung sẽ được hiển thị của label

vd: nếu bỏ chữ “Danh sách 01” thì option 1 sẽ hiển thị là label1

<select>  
<option label="label01">Danh sách 01</option>  
<option label="label02">Danh sách 02</option>  
<option label="label03">Danh sách 03</option>  
<option label="label04">Danh sách 04</option>  
</select>

#### selected

giống với các thuộc tính như checked, multiple, thì selected phải được viết như ở vd dưới, nó chỉ định option mặc định của 1 select

<select>  
<option>Danh sách 01</option>  
<option selected ">Danh sách 02</option>  
<option>Danh sách 03</option>  
<option>Danh sách 04</option>  
</select>

## Thẻ <OUTPUT>

Phần tử <output> thể hiện cho kết quả của phép tính (như được thực hiện bởi script).

Ví dụ:  
Thực hiện một phép tính và hiển thị kết quả thông qua phần tử <output>:

<!DOCTYPE html>

<html>

<body>

<form action="action\_page.php" oninput="x.value=parseInt(a.value)+parseInt(b.value)">

0<input type="range" id="a" name="a" value="50">100 + <input type="number" id="b" name="b" value="50"> =

<output name="x" for="a b"></output>

<br><br>

<input type="submit">

</form>

</body>

</html>

### for

Chỉ định mối quan hệ giữa kết quả của phép tính và các phần tử sử dụng trong phép tính

( mình chưa hiểu? vì khi mình xóa đi cũng không có ảnh hưởng gì

### form

Chỉ định một hoặc nhiều form mà phần tử output thuộc về, giống [form](#_form) của input

### name

Chỉ định tên cho phần tử đầu ra

### Thiết lập css mặc định

output {  
display: inline;  
}

## Thẻ <IFRAME>

**Cú pháp Iframe**  
Một iframe được định nghĩa bằng thẻ iframe như sau:

**<iframe src=”**[**URL**](https://www.dammio.com/glossary/url)**”></iframe>**

Thuộc tính **src** đặc tả đường dẫn website của trang bên trong iframe.

### **width, height**

dùng để định nghĩa chiều rộng và chiều cao của 1 Iframe. Các giá trị width, height có thể là giá trị pixel (hay dạng phần trăm như 80%).

## Thẻ <NAV>

Thẻ <nav> định nghĩa các liên kết điều hướng.

Lưu ý là KHÔNG phải tất cả các liên kết của tài liệu phải nằm bên trong phần tử <nav>. Phần tử <nav> chỉ dành cho khối liên kết chính có mục đích điều hướng.

Các trình duyệt, như màn hình đọc cho người khuyết tật, có thể sử dụng các phần tử này để xác định có nên đọc/xuất một nội dung nào đó hay không.

* Tag <nav> định nghĩa link danh mục (navigation).
* Tag <nav> thường sử dụng cho global menu, local link, topic path, pager link,...

<nav>

<a href="/html/">HTML</a> |

<a href="/css/">CSS</a> |

<a href="/js/">JavaScript</a> |

<a href="/jquery/">jQuery</a>

</nav>

Thiết lập CSS mặc định

Hầu hết các trình duyệt sẽ hiển thị thẻ <nav> với các giá trị mặc định sau:

*nav {*  
*display: block;*  
*}*

## Thẻ<SVG>

### SVG là gì?

SVG (Scalable Vector Graphics), là một định dạng hình ảnh (tương tự như JPG, PNG,... mà chúng ta vẫn thường dùng) sử dụng cấu trúc XML để hiển thị hình ảnh dưới dạng vector. Vì là hình ảnh dạng vector nên chúng ta có thể hiển thị, co giãn (scale) thoải mái mà không làm giảm chất lượng hình ảnh. Một ưu điểm của SVG là tất cả mọi element và attribute của các element đó đều có thể animateVí dụ một file SVG để vẽ hình tròn: :

<svg width="100" height="100">

<circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />

</svg>

### Dùng trong HTML thế nào?

#### Dùng trực tiếp:

Bạn có thể chèn trực tiếp nội dung file SVG vào trang HTML, cho vào 1 cái thẻ DIV chẳng hạn.

<div class="circle">

<svg width="100" height="100">

<circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />

</svg>

</div>

#### Dùng thông qua thẻ IMG

Hoặc bạn có thể dùng qua thẻ IMG như một hình ảnh bình thường.

<img src="circle.svg" />

### Dùng trong CSS thế nào?

<div class="icon-facebook"></div>

### Sử dụng Background Image

.icon-facebook {

background-image: url("facebook.svg");

...

}

### Những thành phần SVG cơ bản

Bạn có thể đối mặt với rất nhiều sự phức tạp với SVG, nhưng nó không cần thiết cho những icon mà chúng ta sẽ tạo ra. Danh sách dưới đây khái quát những khối mà chúng ta sẽ cần.

* <svg> Bao bọc và định nghĩa toàn bộ hình. <svg> chính là đồ hoạ vector mà <html> là một trang web.
* <line> Tạo những đường thẳng đơn.
* <polyline> Tạo những đường đa đoạn.
* <rect> Tạo hình chữ nhật và hình vuông.
* <ellipse> Tạo hình tròn và hình oval.
* <polygon> Tạo hình đa giác, với ba hoặc nhiều cạnh.
* <path> Tạo bất kỳ hình nào mà bạn thích bằng cách định nghĩa những điểm và đường thẳng giữa chúng.
* <defs> Định nghĩa những tài nguyên có thể sử dụng lại. Không có gì nằm trong phần <defs> được hiển thị. <defs> là vector mà <head> là một trang web.
* <g> Gom nhiều hình dạng thành một nhóm. Đặt các nhóm trong phần <defs> để cho phép chúng được sử dụng lại.
* <symbol> Liên kết một nhóm, những với một số tính năng phụ. Thông thường được đặt trong phàn <defs>.
* <use> Lấy những tài nguyên được định nghĩa bên trong phần <defs> và làm cho chúng hiển thị trong SVG.

### Ma thuật

### <https://webdesign.tutsplus.com/vi/tutorials/how-to-hand-code-svg--cms-30368>

## Thẻ <SECTION>

Thẻ <section> định nghĩa khu vực trong một tài liệu, chẳng hạn như các chương, phần đầu, phần chân hoặc bất cứ khu vực nào khác của tài liệu.

Nó cũng chả có gì ngoài việc phân ra các vùng chức năng, nhưng nên có section với các class riêng để đổ dữ liệu vào

## <article>

article là phần tử để bao bọc nội dung độc lập, nó có thể là một bài post của diễn đàn, một bài viết của trang, một bài báo, một bình luận ... hoặc bất kỳ một nội dung độc lập nào.

Thẻ <article> trong HTML5 được dùng thay thế cho thẻ <div> của HTML4.

<article>

<h1>Tiêu đề bài viết</h1>

<p>Các nội dung bài viết</p>

</article>

## <aside>

Các <aside>thẻ định nghĩa một số nội dung ngoài nội dung nó được đặt trong.

Nội dung sang một bên nên liên quan gián tiếp đến nội dung xung quanh.

**Mẹo:** Các <aside>nội dung thường được đặt như một thanh bên trong một tài liệu.

# EVENT trong HTML

url tham khảo: <https://webvn.com/thu-vien/thu-vien-html/thu-vien-su-kien/>

## Các bắt sự kiện

1 <Tên-phần-tử Tên-sự-kiện="đoạn mã sẽ được thực thi khi sự kiện xảy ra">

2 viết riêng trong file script không cần viết bên trong thẻ ở file html

cách này cho phép bạn phân chia rõ hơn giữa html và javascript

window.[onload](#_onload) = function() {

     var banner\_obj = this.document.getElementById("banner");

     banner\_obj.addEventListener('click', clickhere);

}

function clickhere() {

     alert('ok banner');

}

Thực tế bạn có thể sử dụng cách thêm sự như loại 1 nhưng sẽ không chuyên nghiệp và hay bằng cách sử dụng hàm addEventListener(). Từ hàm này bạn có thể liên tưởng qua Events trong jQuery có cách xử lý y chang như vậy nên sau này nếu đi phỏng vấn có hỏi câu này thì bạn trả lời bản chất xử lý sự kiện trong jQuery cũng sử dụng hàm này nhé.

## oninput

Thuộc tính oninput được kích hoạt khi người dùng nhập hoặc thay đổi thông tin trong thẻ. thay đổi ngay và luôn

Thuộc tính oninput kích hoạt khi giá trị của thẻ <input> và <textarea> thay đổi.

**Mẹo**: Sự kiện này tương tự như sự kiện [onchange](https://webvn.com/su-kien-onchange/). Điểm khác biệt là sự kiện oninput diễn ra ngay khi giá trị của thẻ thay đổi, trong khi thuộc tính onchange diễn ra khi bạn chuyển trỏ chuột sang một thành phần khác. Một sự khác biệt nữa đó là sự kiện onchange có thể sử dụng trong thẻ <keygen> và thẻ <select>.

Cú pháp

|  |
| --- |
| <element oninput = "script"> |

script có thể viết luôn như ví dụ của [thẻ <output>](#_Thẻ_<OUTPUT_>) hoặc có thể viết hàm trong file js:

VD:

<p>Nhập thông tin vào ô bên dưới để kích hoạt thuộc tính oninput.</p>

    <input type="text" id="myInput" oninput="myFunction()">

    <p id="demo"></p>

Nếu để oninput ở thẻ <form> thì tất cả các [thẻ <input>](#_Thẻ_<INPUT>) và <textarea> trong form đó đều có oninput: nghĩa là. bất kỳ thay đổi nào về giá trị của bất kỳ thẻ nào thì đều gọi thực thi “script”. Tương tự với các [<option>](#_<option>) trong [<select>](#_Thẻ_<SELECT>)

Nếu để oniput ở [thẻ <input>](#_Thẻ_<INPUT>) và <textarea> thì chỉ khi thay đổi giá trị thẻ đó thì mới thực thi “script”

## onbeforeunload

Sự kiện onbeforeunload được kích hoạt khi người dùng đóng cửa sổ hoặc chuyển tới trang khác.

Sự kiện này cho phép bạn hiển thị một hộp thoại thông báo tới người dùng muốn ở lại hay chuyển tới trang khác.

Thông báo mặc định sẽ xuất hiện trong hộp thoại xác nhận, ở trên mỗi trình duyệt sẽ xuất hiện một thông báo khác nhau. Tuy nhiên, thông báo chuẩn sẽ là “Are you sure you want to leave this page?”. Bạn không thể bỏ thông báo này.

Tuy nhiên, mình chưa tìm ta cách để đưa thông báo riêng theo câu chữ của mình vào

thường đặt sự kiện này ngay thẻ body

**Lưu ý:** Trong Firefox, chỉ xuất hiện thông báo mặc định (không xuất hiện thông báo tùy chỉnh (nếu có)).

## onload

Thuộc tính onload được kích hoạt khi đối tượng đã được tải xong.

onload thường được sử dụng nhất trong thẻ <body> để chạy một script khi trình duyệt của bạn đã tải đầy đủ tất cả thành phần của website (bao gồm ảnh, script, CSS, v.v…). Tuy nhiên, nó cũng có thể sử dụng trong những thẻ khác (xem phần “Chi tiết” bên dưới).

Thuộc tính onload có thể được sử dụng để kiểm tra phiên bảnvà kiểu trình duyệt của người dùng, sau đó tải các thành phần của website để tương thích với trình duyệt dựa trên thông tin thu được.

vd:

<!DOCTYPE html>

<html>

<body onload="checkCookies()">

<p id="demo"></p>

<script>

function checkCookies() {

var text = "";

if (navigator.cookieEnabled == true) {

text = "Cookies đã được bật.";

} else {

text = "Cookies chưa được bật.";

}

document.getElementById("demo").innerHTML = text;

}

</script>

</body>

</html>

## onblur

Thuộc tính onblur được kích hoạt ngay khi người dùng chuyển con trỏ nháy ra ngoài thẻ.

Onblur thường được sử dụng nhiều nhất trong việc kiểm tra tính hợp lệ của imail.

Định dạng lại forn chữ, hoa, thường hay tự động thêm vào đầu(sđt: +84) thêm vào cuối(.com, .es)

vd:

Nhập tên của bạn: <input type="text" name="fname" id="fname" onblur="myFunction()">

    <p>Khi bạn di chuyển con trỏ nháy ra khỏi ô nhập tên, một hàm sẽ được kích hoạt để chuyển toàn bộ ký tự đã nhập thành viết hoa.</p>

    <script>

        function myFunction() {

            var x = document.getElementById("fname");

            x.value = x.value.toUpperCase();

        }

    </script>

## onchange

Thuộc tính onchange được kích hoạt ngay khi giá trị của thẻ thay đổi.

**Mẹo:** Sự kiện này tương tự như sự kiện [oninput](#_oninput). Điểm khác biệt là sự kiện oninput diễn ra ngay khi giá trị của thẻ thay đổi, trong khi thuộc tính onchange diễn ra khi bạn chuyển trỏ chuột sang một thành phần khác. Một sự khác biệt nữa đó là sự kiện onchange có thể sử dụng trong thẻ <keygen> và thẻ <select>. Xem ví dụ bên dưới.

<p>Chọn tên xe từ danh sách dưới đây.</p>

<select id="mySelect" onchange="myFunction()">

<option value="Audi">Audi

<option value="BMW">BMW

<option value="Mercedes">Mercedes

<option value="Volvo">Volvo

</select>

<p>Khi bạn chuyển sang một xe mới, một hàm sẽ được kích hoạt để hiển thị giá trị ra ngoài.</p>

<p id="demo"></p>

<script>

function myFunction() {

var x = document.getElementById("mySelect").value;

document.getElementById("demo").innerHTML = "Bạn đã chọn xe: " + x;

}

</script>

! Cũng giống như form . Nêý đặt event ở select thì tất cả các option sẽ có tác dụng

## onfocus

trái ngược với [onblur](#_onblur)

Thuộc tính onfocus được kích hoạt ngay chi người dùng chuyển con trỏ vào thẻ.

Onfocus được sử dụng hầu hết trong thẻ <input>, <select> và <a>.

## onkeydown và onkeyup

onkeydown: Thuộc tính onkeydown kích hoạt khi người dùng nhấn một phím trên bàn phím.

onkeyup: Thuộc tính onkeyup được kích hoạt khi người dùng thả một phím (trên bàn phím).

VD:

<!DOCTYPE html>

<html>

<body>

<p>Khi bạn nhấn và giữ một phím trong thẻ bên dưới, màu nền sẽ chuyển sang màu đỏ. Thả phím ra, màu nền sẽ chuyển sang màu xanh lá cây.</p>

<input type="text" id="demo" onkeydown="keydownFunction()" onkeyup="keyupFunction()">

<script>

function keydownFunction() {

document.getElementById("demo").style.backgroundColor = "red";

}

function keyupFunction() {

document.getElementById("demo").style.backgroundColor = "green";

}

</script>

</body>

</html>

## onkeypress

Thuộc tính onkeypress được kích hoạt khi người dùng nhấn một phím (trên bàn phím) vào thẻ.

**Lưu ý:** Sự kiện onkeypress không kích hoạt với một số phím (v.d. ALT, CTRL, SHIFT, ESC) trong mọi trình duyệt. Để xác định người dùng đã nhấn một phím, bạn sử dụng sự kiện [onkeydown](#_onkeydown_và_onkeyup) vì sự kiện này làm việc với tất cả các phím.

## onreset

Thuộc tính onreset được kích hoạt khi thông tin trong form được xóa về mặc định (reset).

onreset sẽ được đặt trong thẻ form và trong form sẽ có thẻ [<input type=”reset”>](#_reset), khi người dùng ấn nút này. mọi dữ liệu trong form sẽ bị reset lại

! Bạn thường nên tránh bao gồm các nút đặt lại trong biểu mẫu của bạn. Chúng hiếm khi hữu ích và thay vào đó có nhiều khả năng làm người dùng thất vọng khi nhấp vào chúng do nhầm lẫn (thường là trong khi cố gắng nhấp vào [nút gửi](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/submit) ).

vd

<form>

<div>

<label for="example">Type in some sample text</label>

<input id="example" type="text">

</div>

<div>

<input type="reset" value="Reset the form"

accesskey="r">

</div>

</form>

## onresize

Thuộc tính onresize được kích hoạt khi cửa sổ trình duyệt thay đổi kích thước.

thẻ thường dùng: thẻ body

! event này ít dùng

## onsubmit

Thuộc tính onsubmit được kích hoạt khi thông tin từ form được gửi đi.

thuộc tính này nên để ở thẻ form

## querySelector

* Lấy ra phần tử đầu tiên của bộ chọn. Tương tự như [.first()](#_.first()) trong jquery

Vd: document.querySelector(".example");

-lấy ra phần thử đầu tiên có class là: example

# CSS

## selector[attribute\*=value]

vd: div[class\* = “temp”] {…}

chọn tất cả các thẻ div có class=”temp”;

vd: [class\* = “temp”] {…}

chọn tất cả các thẻ có class=”temp”;

## float

xác định thành phần dồn về 1 phía

* left - xác định thành phần được nổi về phía bên trái.
* right - xác định thành phần được nổi về phía bên phải.
* none - xác định thành phần không được nổi về phía nào(**mặc định**).
* inherit - xác định thành phần thừa hưởng thuộc tính từ thành phần cha.

muốn xóa thuộc tính float từ thẻ nào ta chỉ cần gọi thuộc tính [clear](#_clear) trong thẻ đó

## clear

Thuộc tính clear trong CSS có tác dụng loại bỏ các thuộc tính float (left - right) trên một thành phần. Để giải quyết vấn đề trong mục chú ý của [thuộc tính float](#_float) phía trên.

* left - xóa bỏ float: left.
* right - xóa bỏ float: right.
* both - xóa bỏ cả float:left và float: right.
* none - không xóa bỏ gì cả (**mặc định**).
* inherit - thừa hưởng giá trị từ thành phần cha.

## Position

Không gian của thẻ con(có parding) không bao gồm pading cuả thẻ cha

### absolute

các thẻ con trong thẻ cha có thuộc tính này sẽ không phân biệt với nhau khi di chuyển sẽ làm phá vỡ cấu trúc

các thẻ con này phải có left(right), top(botton) để xác định vị trí tương đối với thẻ cha

* muốn dùng absolute thì thẻ cha phải set position:.. . Nếu không nó sẽ lấy cha gần nhất được set position. Nếu không có cha nào thì nó sẽ lấy body

### relative

vd:

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<style>

.relative{

height: 300px;

width: 300px;

margin: 100px auto;

background: blue;

position: relative;

text-align: center;

}

.absolute{

position: relative ;

height: 40px;

width: 40px;

}

.red{

top: 0px;

left: 100px;

background: red;

}

.yellow{

top: 0px;

left: 110px;

background: yellow;

}

.white{

background: white;

margin:0 auto;

}

</style>

</head>

<body>

<div class="relative">

<div class="absolute red"></div>

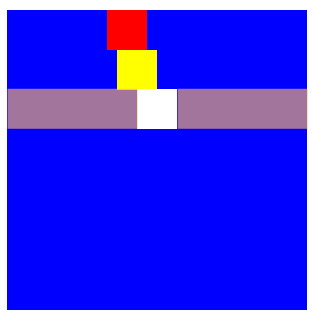
<div class="absolute yellow"></div>

<div class="absolute white"></div>

</div>

</body>

</html>



trong vd trên white là thẻ con thứ 3 nên top: 0 tưng ứng với cạnh dưới của thẻ vàng:

cụ thể cạnh trên của white sẽ cách cạnh trên của cha nó 1 đoạn: 40\*2=80px

ở đây relative đã phân vùng xác đinh cho từng thằng con. nên khi chúng di chuyển lên nhau( top có thể âm) chúng sẽ không ảnh hưởng tới nhau và không bị phá vỡ cấu trúc

### sự khác nhau giữ position relative và absolute <importand

|  |  |
| --- | --- |
| absolute | relative |
|  |  |
| Ngoài magin còn có 1 vùng positon. position(left, right, top, bottom). được tính từ cạnh của cha  margin:  không có, nếu không thiết lập cả left-right( hoặc top-bottom  thiết lập left-right( hoặc top-bottom) có ảnh hưởng tới margin   * nên thiết lập heigh, weith cho phần tử   **Lưu ý:**  cài đặt position(left, right, top, bottom) càng to thì margin(left, right, top, bottom) càng nhỏ  nếu muốn căn vị trí, căn giữa,..thì phải cài đặt lại position với left-right, top-bottom   1. Nếu không biết chính xác left và right căn giữa thì cứ để **left :0; right:0 margin: auto;** | position(left, right, top, bottom)=0  bất kể di chuyển đêì sẽ có thuộc tính âm  - di chuyển sang phải ; right<0  - di chuyển lên trên: top<0  margin:  luôn chiếm hết phần dư 2 bên và thiết lập position(left, right, top, bottom) không ảnh hưởng tới margin  vd: |

### Sticky

## animate< sẽ còn phát triển>

url: <https://hocwebchuan.com/reference/cssSection/css3/pr_animation.php>

vd:

<!DOCTYPE html>

<html>

<head>

<style>

div {

width: 100px;

height: 100px;

background: red;

position: relative;

animation: myonly 2s infinite;

}

@keyframes myonly {

from {left: 0px;}

to {left: 100px; background-color: blue;}

}

</style>

</head>

<body>

<h1>The @keyframes Rule</h1>

<p><strong>Note:</strong> The @keyframes rule is not supported in Internet Explorer 9 and earlier versions.</p>

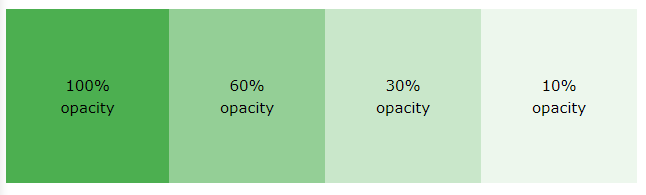
<div></div>

</body>

</html>

RGBA

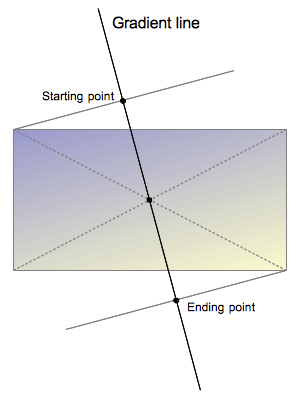
Nếu bạn không muốn áp dụng độ mờ đục cho các phần tử con, như trong ví dụ ở trên, sử dụng các giá trị màu RGBA. Ví dụ sau thiết lập độ mờ cho màu nền chứ không phải văn bản:



Ngoài RGB, CSS3 giới thiệu giá trị màu RGB với kênh alpha (RGBA) – xác định độ mờ của màu. Giá trị màu RGBA được chỉ định bằng: rgba (đỏ, xanh, xanh, alpha). Thông số alpha là một số giữa 0,0 (hoàn toàn trong suốt) và 1,0 (đục hoàn toàn).

1. div {
2. background: **rgba**(76, 175, 80, 0.3) /\* Green background with 30% opacity \*/
3. }

## Gradient



background-image: linear-gradient(

90deg, // góc của grandient line // mặc định “|” à 0deg;

rgba(#fff, 0) 0,

rgba(#fff, 0.2) 20%,

rgba(#fff, 0.5) 60%,

rgba(#fff, 0)

);// từ trái sang có 4 loại màu, loại 2 tọa độ tâm là 20% dọc theo đường gradient line màu 3 ở tọa độ 60 % dọc theo đường grandient line

&::after {

position: absolute;

top: 0;

right: 0;

bottom: 0;

left: 0;

transform: translateX(-100%);

background-image: linear-gradient(

90deg,

rgba(#fff, 0) 0,

rgba(#fff, 0.2) 20%,

rgba(#fff, 0.5) 60%,

rgba(#fff, 0)

);

animation: shimmer 2s infinite;

content: '';

}

@keyframes shimmer {

100% {

transform: translateX(100%);

}

}

}

animation: shimmer 2s infinite;

content: '';

}

@keyframes shimmer {

100% {

transform: translateX(100%);

}

}

}

## Box-sizing: border-box

Cố định kích thức border khi pading thay đổi

* Height(width) = content+ padding + border
* Nếu dom con được có position:absolute(true) top(left) tính từ padding, không tính border. Còn lại tính từ bên trong border

ví dụ:

|  |  |
| --- | --- |
| 01  02  03 | box-sizing: border-box;  -moz-box-sizing: border-box;  -webkit-box-sizing: border-box; |

Trong đó, nếu viết không có tiền tố là dành cho trình duyệt IE8, Opera 7, Firefox và Google chrome bản mới. -webkit là dành cho Google Chrome bản cũ và -moz là dành cho Firefox bản cũ.

## Display

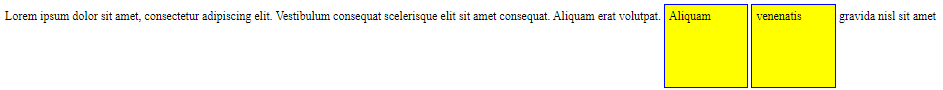
### Display:Inline

Với kiểu này thì các item sẽ nằm trên cùng một dòng. Nếu các items vượt quá độ dài của dòng thì item sẽ xuống dòng mới

Các item có kiểu display này **không thể set width và height**.

Các inline item sẽ chỉ có thể điều chỉnh margin và padding **left and right** (top và bottom thì không thể).

### Display: inline-block

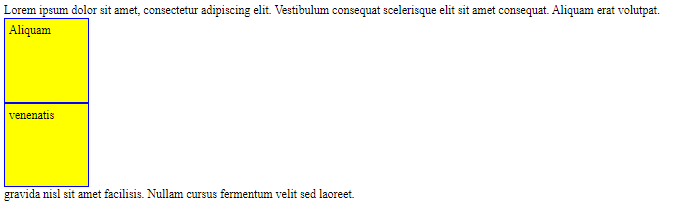


Kiểu display: inline-block sẽ được sắp xếp giống với kiểu display: inline, nghĩa là các items sẽ được xếp cùng nhau trên một dòng . Tuy nhiên các items sẽ có thuộc tính của display: block như là có set width, height, margin, padding đủ 4 hướng.

### Display: block

Khác với kiểu display: inline thì các item có kiểu display: block luôn được xuống dòng và chiếm toàn bộ width nếu width không được set

* nó tự động xuống dòng
* Đứng sau nó cũng tự động xuống dòng



## Flex

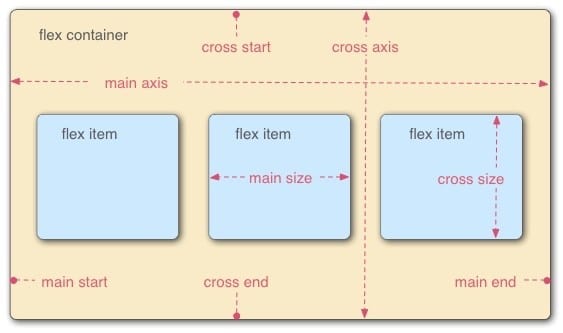
*lexbox* là một kiểu dàn trang (layout mode) mà nó sẽ tự cân đối kích thước của các phần tử bên trong để hiển thị trên mọi thiết bị. Nói theo cách khác, bạn không cần thiết lập kích thước của phần tử, không cần cho nó float, chỉ cần thiết lập nó hiển thị chiều ngang hay chiều dọc, lúc đó các phần tử bên trong có thể hiển thị theo ý muốn.

Hiện nay, theo lời khuyên từ Mozilla thì chúng ta sử dụng Flexbox để thiết lập bố cục trong phạm vi nhỏ (ví dụ như những khung trong website) và khi thiết lập bố cục ở phạm vi lớn hơn (như chia cột website) thì vẫn nên sử dụng kiểu thông thường là dàn trang theo dạng lưới (grid layout).

### Thuật ngữ các thành phần trong Flexbox

Trước khi đi vào tìm hiểu sâu hơn về Flexbox, chúng ta cần nắm qua cấu trúc của Flexbox là như thế nào và một số thuật ngữ liên quan.

Dưới đây là sơ đồ cấu trúc Flexbox từ Mozilla Developer Network.

[](https://thachpham.com/wp-content/uploads/2016/06/flex_terms.jpg)

*Nguồn: Mozilla Developer Network*

Hai thành phần quan trọng nhất trong một bố cục Flexbox là gồm container và item:

* container: là thành phần lớn bao quanh các phần tử bên trong, bạn sẽ thiết lập kiểu hiển thị inline (sắp xếp theo chiều ngang) hoặc kiểu sắp xếp theo chiều dọc. Khi đó, các item bên trong sẽ hiển thị dựa trên thiết lập của container này.
* item: Các phần tử con của container được gọi là item, ở item bạn có thể thiết lập nó sẽ sử dụng bao nhiêu cột trong một container, hoặc thiết lập thứ tự hiển thị của nó.

Ngoài hai thành phần chính đó, chúng ta có thể thấy hình trên sẽ có:

* main start, main end: Khi thiết lập flexbox, điểm bắt đầu của container gọi là main start và điểm kết thúc được gọi là main end. Điều này có nghĩa, các item bên trong sẽ heienr thị từ main start đến main end (hoặc là được phép hiển thị đến main end). Và chiều vuông góc của nó là **cross start**, **cross end** cũng có ý nghĩa tương tự nhưng luôn vuông góc với main start, main end.
* main axis: Trục này là trục chính để điều khiển hướng mà các item sẽ hiển thị. Như bạn thấy ở trên hình main axis là trục dọc nên các item sẽ hiển thị theo chiều dọc, tuy nhiên ta có thể sử dụng thuộc tính flex-direction để thay đổi trục của main axis và lúc đó các item sẽ hiển thị theo nó. Và cross axis luôn là trục vuông góc của main axis.
* main size: Bạn có thể hiểu đơn giản là kích thước (chiều rộng hoặc dọc) của mỗi item dựa theo trục main axis.
* cross size: Là kích thước (chiều rộng hoặc dọc) của mỗi item dựa theo trục cross axis.

**Bắt đầu với Flexbox**

Trước tiên mình bắt đầu với một cấu trúc đơn giản sau:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | <div class="container">         <div class="item item1">1</div>         <div class="item item2">2</div>         <div class="item item3">3</div>         <div class="item item4">4</div>      </div> |

Và một đoạn CSS ban đầu để thiết lập màu sắc và kích thước để dễ nhìn từng thành phần:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | /\*\* Global CSS \*\*/  .container {   background: red;   max-width: 960px;   max-height: 1000px;   margin: 0 auto;   padding: 5px;  }  .item {   background: blue;   margin: 5px;   color: white;   height: 50px;   text-align: center;   line-height: 50px;  } |

Bây giờ chúng ta sẽ bắt đầu làm việc với Flexbox ở đây. Trước tiên chúng ta sẽ đưa .container về hiển thị ở dạng flexbox với display: flex.

|  |  |
| --- | --- |
| 01  02  03  04 | /\*\* Flex layout \*\*/  .container {   display: flex;  } |

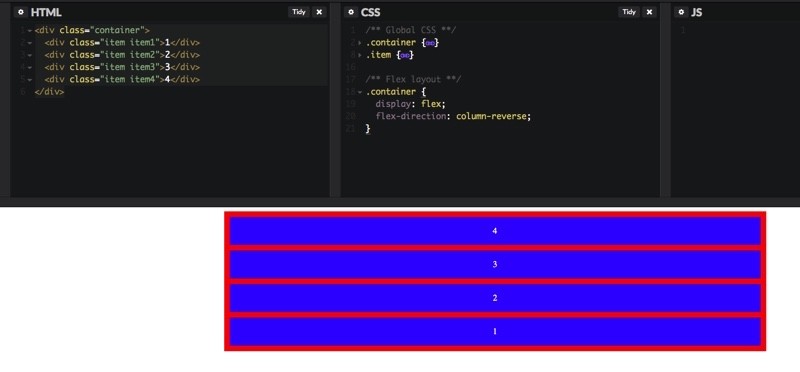
Bạn sẽ thấy các item bên trong đã tự hiển thị theo chiều dọc, tương ứng với trục main axis mặc định là chiều ngang.

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-01.jpg)

Nếu bạn muốn đổi trục thì chỉ cần thêm thuộc tính flex-direction vào container. Cụ thể:

|  |  |
| --- | --- |
| 01 | flex-direction: row | column | row-reverse | column-reverse |

* flex-direction:
  + row: Chuyển trục main axis thành chiều ngang, nghĩa là hiển thị theo hàng.
  + colum: Chuyển trục main axis thành chiều dọc, nghĩa là hiển thị theo cột.
  + row-reverse: Hiển thị theo hàng nhưng đảo ngược vị trí các item.
  + column-reverse: Hiển thị theo cột nhưng đảo ngược vị trí các item.

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-02.jpg)

Quá đơn giản đúng không?

**flex-wrap**

Bây giờ để hiểu cái này, chúng ta thử thêm chiều rộng cho mỗi item bên trong là 1000px xem chuyện gì sẽ xảy ra khi dùng flexbox.

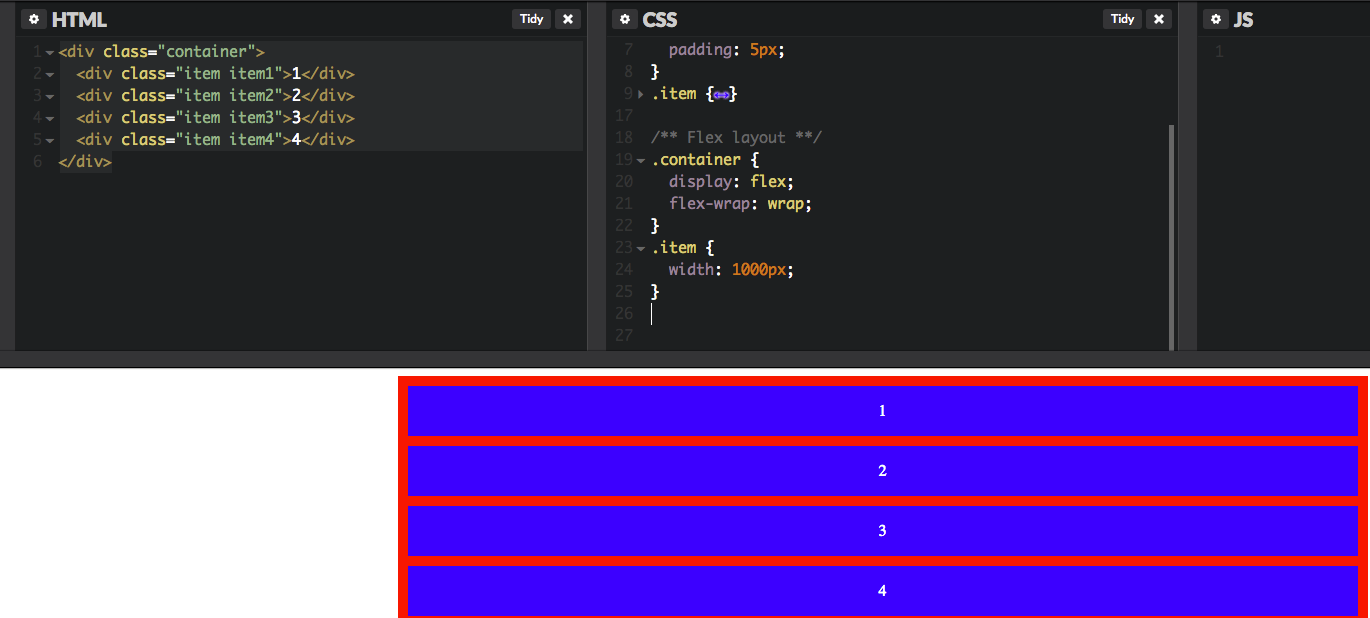
|  |  |
| --- | --- |
| 01  02  03 | .item {   width: 1000px;  } |

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-03.png)

Như bạn thấy, dù chúng ta có thêm chiều rộng cho mỗi item bên trong là 1000px nhưng nó vẫn hiển thị trên một hàng đều nhau. Lý do là mặc định, flexbox tự căn chỉnh các phần tử hiển thị đều nhau theo trục main axis của nó dựa theo chiều rộng của container. Vì vậy cho dù bạn có chỉnh chiều rộng vượt quá giới hạn của nó thì nó vẫn không bị nhảy lung tung.

Bây giờ hãy thử thêm thuộc tính flex-wrap: wrap vào container thử nhé.

|  |  |
| --- | --- |
| 01  02  03  04 | .container {   display: flex;   flex-wrap: wrap;  } |

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-04.png)

Nói nôm na là thuộc tính này cho phép container có thể bọc lại các item kể cả khi kích thước của item bị thay đổi, mặc định là nowrap. Thuộc tính này có thể áp dụng với cả chiều dọc của container và item. Các bạn xem video ở trên để rõ hơn.

**order**

Trong đoạn HTML ví dụ của mình trong bài này, mình có đặt số thứ tự cho mỗi phần tử là 1, 2, 3 và 4 với class tương ứng là .item1, .item, .item3 và .item4. Mặc định item này sẽ hiển thị theo thứ tự trong HTML, nhưng với thuộc tính order chúng ta có thể sắp xếp lại vị trí sắp xếp của các item.

Ví dụ mình có:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | .item1 {   order: 4;  }  .item2 {   order: 3;  }  .item3 {   order: 1;  }  .item4 {   order: 2;  } |

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-05.png)

Mặc định thứ tự sắp xếp sẽ bắt đầu từ bên trái qua phải, từ trên xuống dưới. Nếu bạn chỉnh lại trục của main axis với thuộc tính flex-direction nó sẽ thay đổi ngược lại.

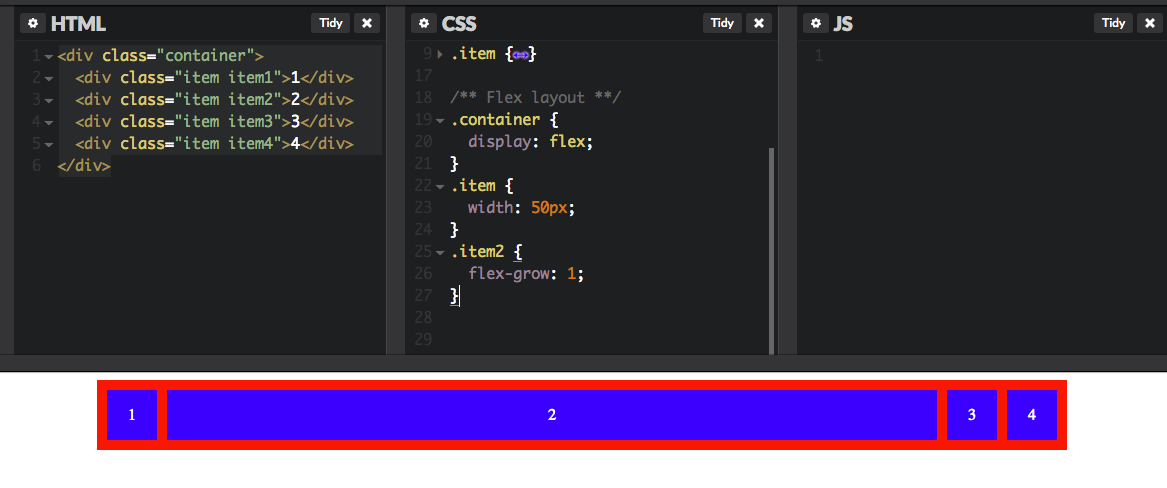
**flex-grow**

Để làm ví dụ này trước tiên mình hãy bỏ chức năng wrap đi và thiết lập chiều rộng của item là 50px.

|  |  |
| --- | --- |
| 01  02  03 | .item {   width: 50px;  } |

Bây giờ ở .item2, mình cho giá trị flex-grow là 1 thử nhé.

|  |  |
| --- | --- |
| 01  02  03 | .item2 {   flex-grow: 1;  } |

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-06.png)

Khi thiết lập nó flex-grow là 1, thì nó sẽ lấy phần trống còn lại của container đắp vào. Bây giờ hãy thử cho .item1 với flex-grow: 2 thử.

|  |  |
| --- | --- |
| 01  02  03 | .item1 {   flex-grow: 2;  } |

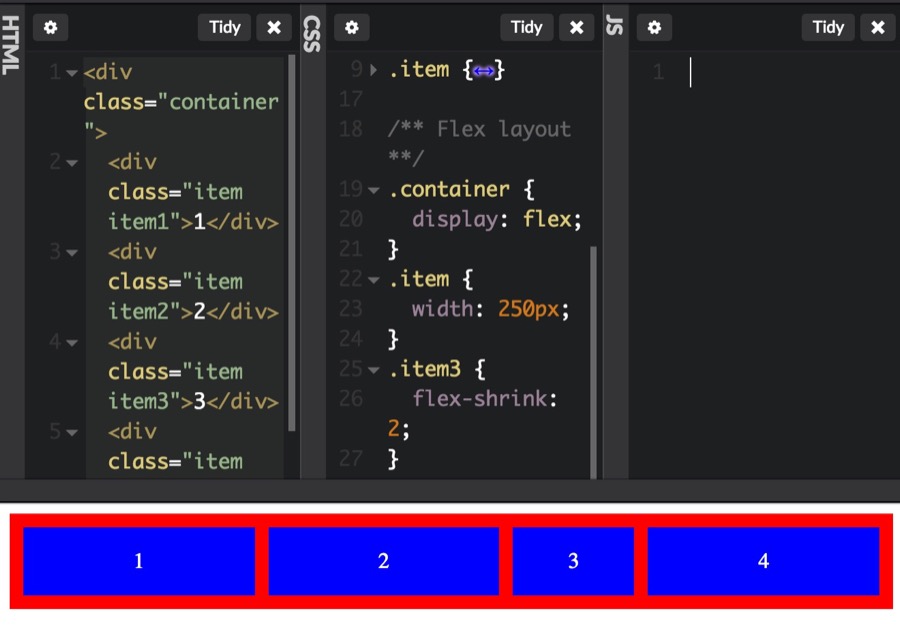
[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-07edited.jpg)

Lúc này giá trị flex-grow: 2 sẽ lấy phần dư lớn gấp đôi của flex-grow: 1.

**flex-shrink**

Bạn có thể hiểu mặc định tất cả các item đều có giá trị flex-shrink là 1. Điều này có nghĩa là khi chúng ta thu nhỏ trình duyệt lại, các phần tử đều co lại bằng nhau. Tuy nhiên giả sử mình muốn .item3 nó co lại nhiều hơn so với các item khác thì mình sẽ tăng giá trị flex-shrink của nó lên.

|  |  |
| --- | --- |
| 01  02  03 | .item3 {   flex-shrink: 2;  } |

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-08.jpg)

**flex-basis**

Cái này bạn có thể hiểu đơn giản nhất là gán cho item một kích thước nhất định. Bạn có thể sử dụng [giá trị tuyệt đối hoặc tương đối](https://thachpham.com/web-development/html-css/cac-don-vi-do-luong-trong-css.html) (căn cứ theo kích thước của container).

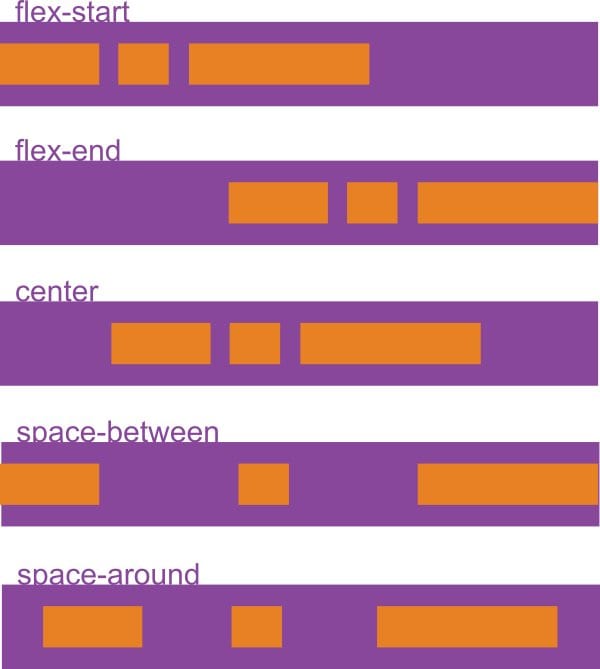
|  |  |
| --- | --- |
| 01  02  03 | .item3 {   flex-basis: 500px;  } |

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-09.jpg)

**justify-content**

Mặc định các item bên trong sẽ rải đều bắt đầu từ main start đến main end, tuy nhiên nếu container vẫn còn khoảng trống thì có thể dùng thuộc tính justify-content để điều chỉnh lại vị trí bắt đầu của nó.

Thuộc tính này có 5 giá trị và bạn có thể xem tấm ảnh bên dưới mình mượn của CSS Tricks để hiểu hơn về ý nghĩa các giá trị của justify-content.

[](https://thachpham.com/wp-content/uploads/2016/06/flexbox-justify-content.jpg)

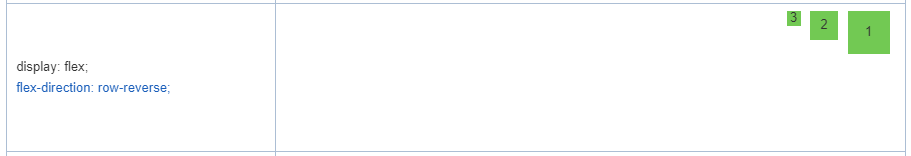
Đây là ví dụ:

|  |  |
| --- | --- |
| 01  02  03  04 | .container {   display: flex;   justify-content: flex-end;  } |

[](https://thachpham.com/wp-content/uploads/2016/06/css-flexbox-10.jpg)

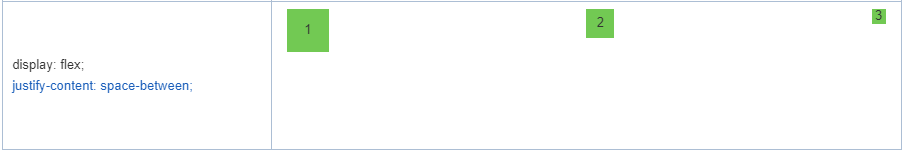
**Kết hợp 1 giá trị**

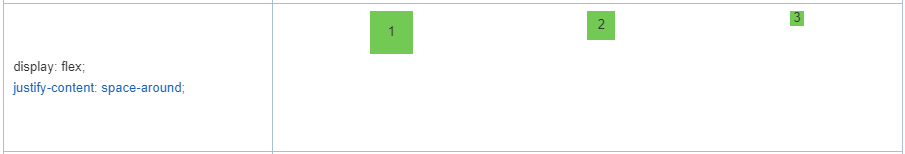








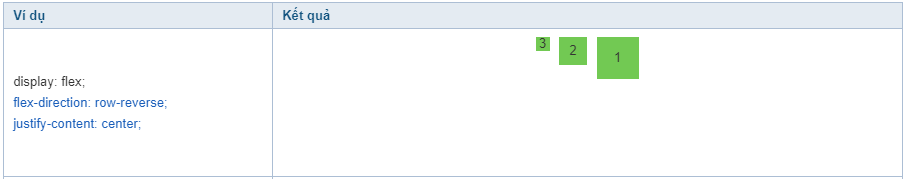




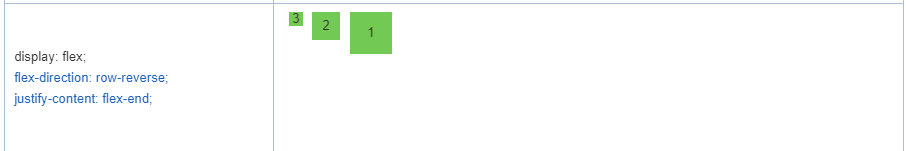


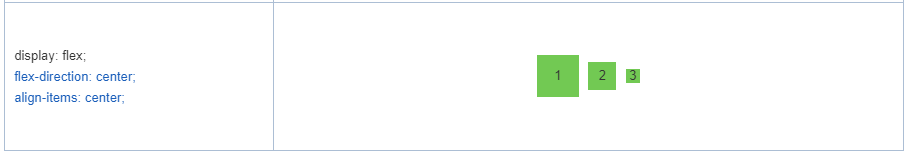


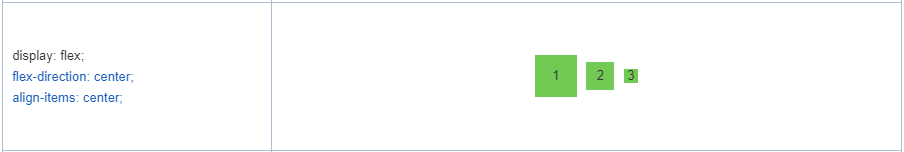


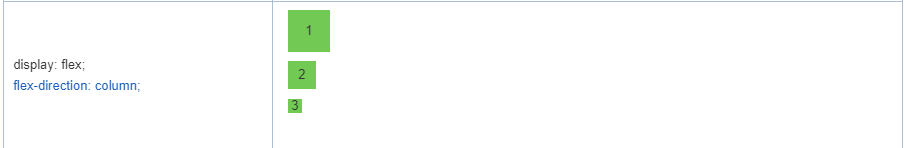


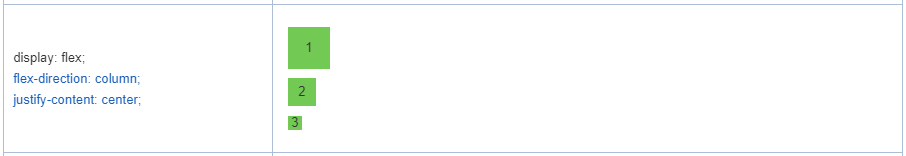




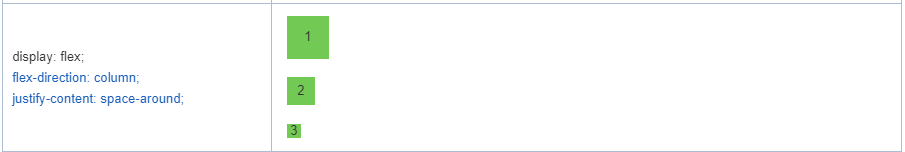


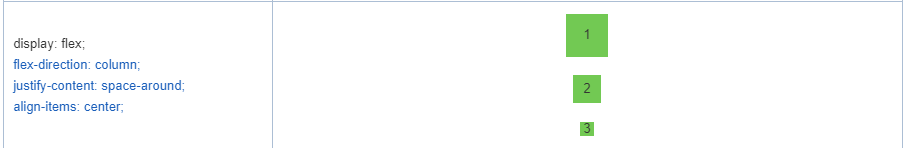


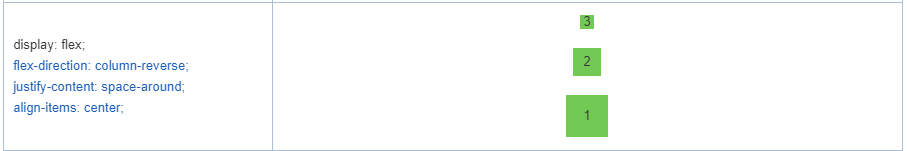




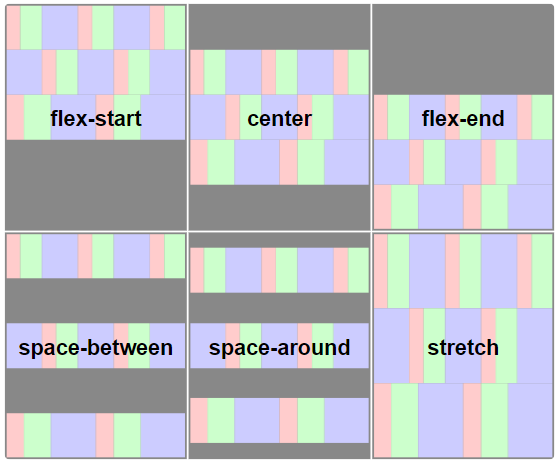








#### Align-content:

****

#### So sánh align-item và align-content

|  |  |
| --- | --- |
| **align-content : center** | **align-item: center** |
|  |  |
| **Có tác dụng cho nhiều dòng(cột)**  **Phạm vi là giống justify-content** | **Căn giữa ở đây là căn giữa cho từng dòng** |

## Grid

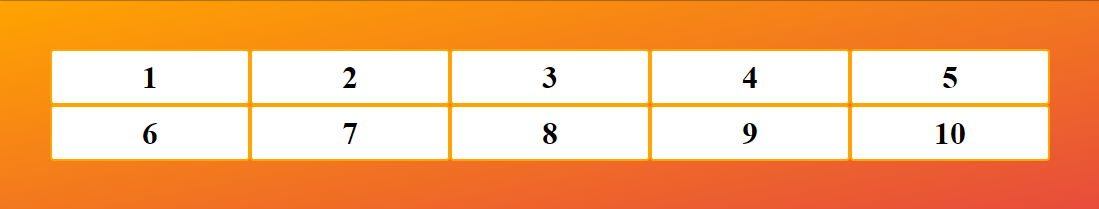
Tài liệu tham khảo : https://www.youtube.com/watch?v=t6CBKf8K\_Ac&t=1357s

### grid-template-columns

có thể dùng px, %, em rem, auto hay gì tùy, muốn điền bao cột thì điền bấy nhiêu giá trị

**container {**

**grid-template-columns: 20% 20% 20% 20% 20%;**

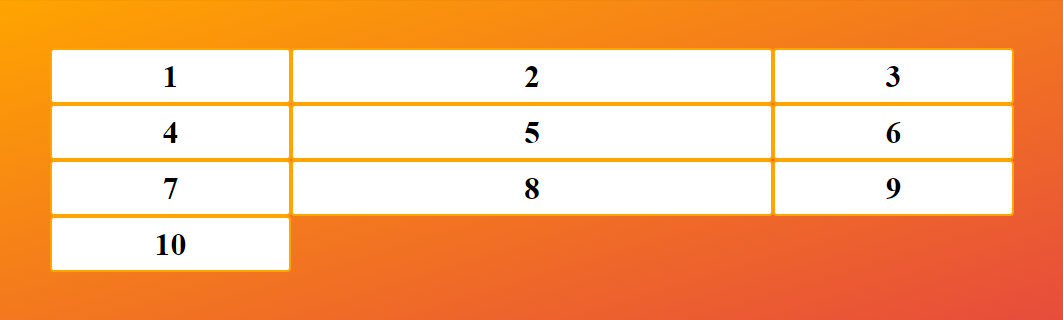
[](https://evondev.com/wp-content/uploads/2018/08/grid-template-columns1.png)

### fraction-unit

grid-template-columns: 1fr 2fr 1fr chẳng hạn thì layout nó sẽ tạo ra 3 cột với cột giữa là 2fr sẽ gấp đôi 2 cột còn lại

**container {**

**grid-template-columns: 1fr 2fr 1fr;**

**}**[](https://evondev.com/wp-content/uploads/2018/08/grid-fr.png)

### grid-auto-flow

Như ở trường hợp trên khi có một hoặc nhiều phần tử bị dư do không đủ cột hay hàng thì nó sẽ rớt xuống tạo nên hàng mới. Thì mình muốn set cho các phần tử bị rớt ra tạo nên Implicit Tracks có chiều cao nhất định nào đó. Chúng ta sẽ dùng thuộc tính grid-auto-flow: row kết hợp với thuộc tính grid-auto-rows để set cho các phần tử đó

grid-auto-flow: row;

grid-auto-rows: 20px; // những item rớt ra bên ngoài sẽ có height là 20px  
 grid-auto-columns:

Tương tự grid-auto-rows:

1. Nếu không set grid-template-columns thì sẽ tự động set tất cả các cột theo giá trị. Hoặc set những item dư ra

### grid-gap

Thuộc tính này giúp chúng ta tạo khoảng cách giữa các phần tử với nhau theo cột và hàng. Thuộc tính này bao gồm thuộc tính grid-row-gap và grid-column-gap được viết như sau grid-gap: grid-row-gap grid-column-gap.

### repeat

Hàm này giúp chúng ta tạo nhiều cột hoặc hàng có cùng giá trị một cách nhanh chóng. Ở bài trước khi chúng ta muốn tạo 5 cột bằng nhau thì chúng ta sẽ dùng grid-template-columns: 20% 20% 20% 20% 20% đúng không nào.

Thì với hàm repeat() mà CSS Grid cung cấp thì chúng ta có thể viết nó như thế này grid-template-columns: repeat(5, 20%) trong đó 5 là số cột muốn tạo còn 20% là đơn vị chúng ta sẽ dùng.

Nếu bạn muốn tạo 6 cột có cột đầu tiên và cột cuối cùng độ rộng là 10% còn 4 cột ở giữa mỗi cột 20% thì đơn giản chúng ta viết thế này grid-template-columns: 10% repeat(4,20%) 10%

### grid-column

Thuộc tính này giúp chúng ta thiết lập vị trí và kích thước của phần tử theo cột. Nó là viết tắt của 2 thuộc tính grid-column-start và grid-column-end. Cú pháp của nó như sau grid-column: grid-column-start / grid-column-end.

Trong đó giá trị của grid-column-start và grid-column-end là các tracks line

vd:

grid {

display: grid;

height: 200px;

grid-template-columns: repeat(6, 1fr);

grid-template-rows: 200px;

}

#item1 {

background-color: #07a787;

}

#item2 {

background-color: #ffa400;

grid-column: 2 / 4;

}

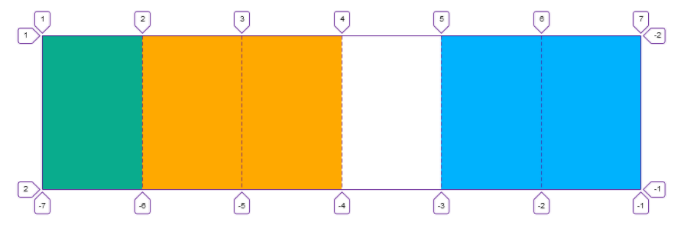
#item3 {

background-color: #00aefd;

grid-column: span 2 / 7;

}

-ttrước tiên set cho grid là 6 cột tương ứng 7 tracks line : grid-template-columns: repeat(6, 1fr);

Còn phần tử item3 có một giá trị là lạ đó là span. Và với grid-column: span 2 / 7 nên nó sẽ chiếm 2 cột tính từ track line số 7 chạy từ phải qua trái. Giá trị span sẽ tương ứng số cột mà các bạn muốn. Ví dụ span 2 nó sẽ chiếm 2 cột, span 1 là một cột.

Ở đây có 2 trường hợp. Trường hợp 1 nếu giá trị span nằm ở đầu như thế này(grid-colum-start) grid-column: span 2 / 7 thì sẽ tính ngược từ phải qua trái bắt đầu từ line số 7 và span 2(chiếm 2 cột).

Trường hợp 2 nếu grid-column: 5 / span 2  giá trị span nằm ở sau(grid-column-end) thì cũng tạo ra kết quả y Codepen ở trên. Nhưng phần tử sẽ tính từ line số 5 và chiếm 2 cột. Kết quả đều như nhau.

- tương tự vói Rows

### Grid areas

- 1 cách khác để căn bố cục thay vì dùng gird-column, grid-row

Vd:

trong grid:

.grid-2{

    display: grid;

    grid-template-columns: repeat(4,1fr) ; // vẫn cứ phải chia hàng chi cột

    grid-template-rows: repeat(3,40px) ;

    grid-template-areas: // set như map vậy

    "m h h h"

    "m c c c"

    "m f f f";

}

trong item ta chỉ việc gán với cú pháp grid-area:

.grid-2 div:nth-child(1){

  background-color: slateblue;

  grid-area: h;

}

.grid-2 div:nth-child(2){

  background-color: olivedrab;

   grid-area: m;

}

.grid-2 div:nth-child(3){

  background-color: olivedrab;

  grid-area: c;

}

.grid-2 div:nth-child(4){

  background-color: thistle;

   grid-area: f;

}



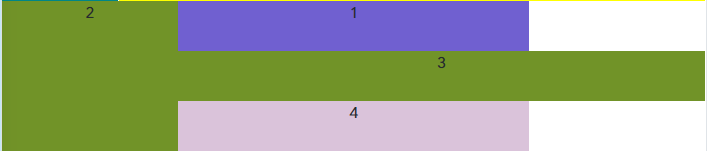
- thay dấu “.” nếu muốn để trống

 grid-template-areas:

    "m h h ."

    "m c c c"

    "m f f .";



### grid-auto-flow

Nếu bạn có một vài grid items bạn không xác định vị trí cụ thể trên grid. Một thuật toán sắp chỗ sẽ tự động làm việc đó. Thuộc tính này sẽ xác định cách thuật toán hoạt động.

Giá trị:

**row** – (mặc định) thuật toán sẽ thêm vào theo từng hàng, thêm hàng mới nếu cần  
**column** – thuật toán sẽ thêm vào theo từng cột, thêm hàng cột nếu cần  
**dense** – thuật toán sẽ lấp vào chỗ trống với nếu vừa với các grid items đứng sau

### justify-content

Căn phần tử theo chiều ngang khi tổng số phần tử nhỏ hơn kích thước grids

### align-content

Căn phần tử theo chiều dọc khi tổng số phần tử nhỏ hơn kích thước grid

### align-items

Căn chỉnh các items theo trục dọc nội tuyến (mặc định từ trên xuống dưới). Giá trị này áp dụng cho tất cả items nằm trong container.

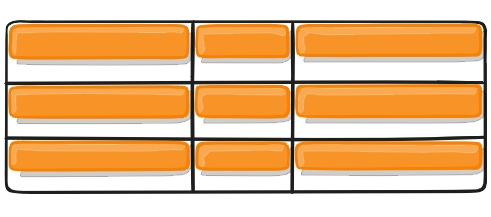
Giá trị:

start – căn chỉnh items dồn về phía cạnh bắt đầu (mặc định là cạnh trên) của cell  
end – căn chỉnh items dồn về phía cạnh kết thúc (mặc định là cạnh dưới) của cell  
center – căn chỉnh items dồn về vào giữa cell  
stretch – (mặc định) lấp đầy toàn bộ chiều rộng của cell

Ví dụ

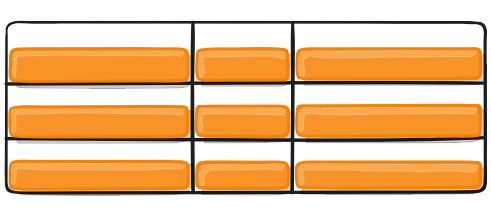
css

|  |  |
| --- | --- |
| 1  2  3 | .container {    align-items: start;  } |



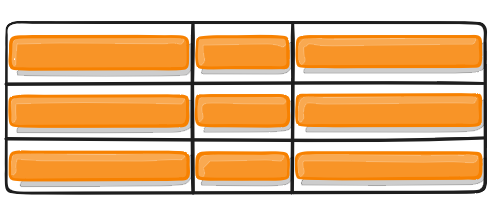
css

|  |  |
| --- | --- |
| 1  2  3 | .container {    align-items: end;  } |



css

|  |  |
| --- | --- |
| 1  2  3 | .container {    align-items: center;  } |



### justify-items

Căn chỉnh các items theo trục ngang nội tuyến (mặc định từ trái sang phải). Giá trị này áp dụng cho tất cả items nằm trong container.

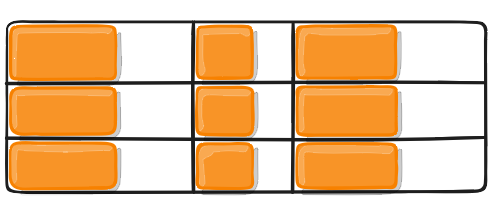
Giá trị:

**start** – căn chỉnh items dồn về phía cạnh bắt đầu (mặc định bên trái) của cell  
**end** – căn chỉnh items dồn về phía cạnh kết thúc (mặc định bên phải) của cell  
**center** – căn chỉnh items dồn về vào giữa cell  
**stretch** – (mặc định) lấp đầy toàn bộ chiều rộng của cell

Ví dụ:

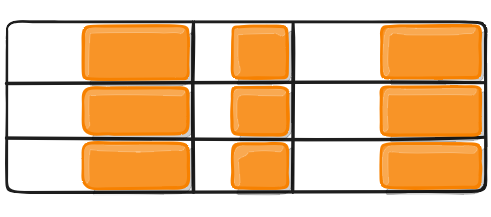
css

|  |  |
| --- | --- |
| 1  2  3 | .container {    justify-items: start;  } |



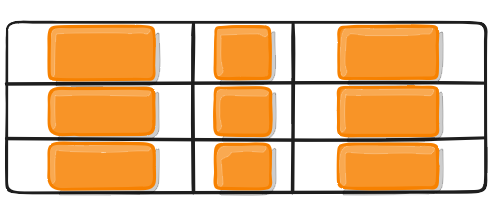
css

|  |  |
| --- | --- |
| 1  2  3 | .container {    justify-items: end;  } |



css

|  |  |
| --- | --- |
| 1  2  3 | .container {    justify-items: center;  } |



### justify-self

template

.item {

justify-self: start | end | center | stretch;

}

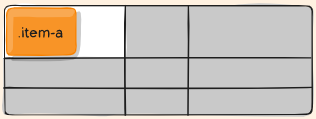
* Tương tự như justify- content nhưng là áp dụng trong 1 item

Vd:

.item-a {

justify-self: start;

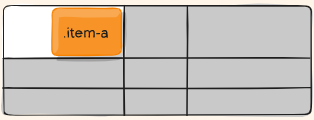
}



.item-a {

justify-self: end;

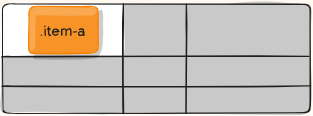
}



.item-a {

justify-self: center;

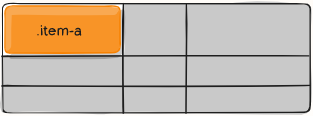
}



.item-a {

justify-self: stretch;

}



### align-self

template

.item {

align-self: start | end | center | stretch;

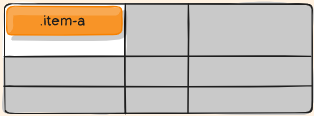
}

Tương tự như align-content nhưng mà áp dụng cho 1 item

.item-a {

align-self: start;

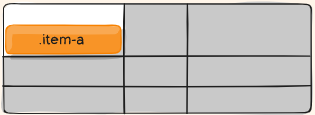
}



.item-a {

align-self: end;

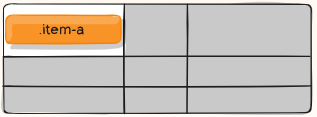
}



.item-a {

align-self: center;

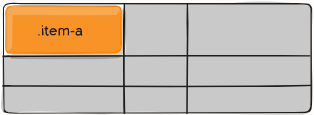
}



.item-a {

align-self: stretch;

}



### auto-fit và autofill

.grid-container--fill {

grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));

}

.grid-container--fit {

grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));

}

.grid-element {

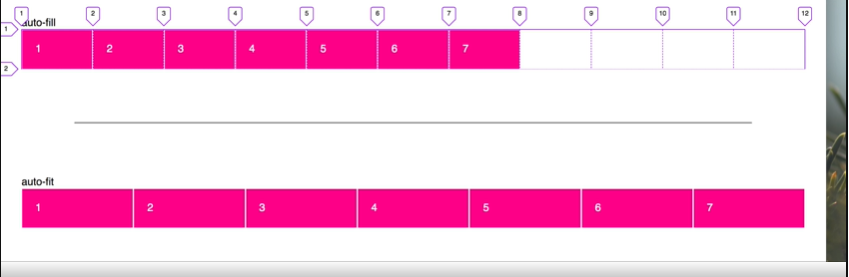
background-color: deepPink;

padding: 20px;

color: #fff;

border: 1px solid #fff;

}





### Fit là đầy luôn

## background-position: 5pxcter

; background-position: 5px center

background-position-x:: 5px x;

* 1. background-position-y:: center cen

height-width

# JavarScripts

## list

### tạo mới

### vd: var cartList = [];

### thêm phần tử vào mảng

vd:

cartList.push({

Amount: $(item).val(),

Products: {

id\_product: $(item).data('id')

}

});

## object

### tạo mới

var <tên đối tượng> = {

<tên thuộc tính>: <giá trị của thuộc tính>,

<tên thuộc tính>: <giá trị của thuộc tính>,

....

}

vd:

var bill = {

Item: cartList,

Id\_customer: $('#txtCustomer').data('id'),

Subtotal : $('#SubTotal').text(),

Total: $('#Total').text(),

DiscountCode: $('#txtdiscountCode').val()

}

hoặc;

## for()

*for (var i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

*for (let i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

## Function

Function <tên hàm> (tham số){..}

Nó không quan tâm tham số(thừa thiếu, sai kiểu dữ liệu không quan trọng). chỉ cần đúng tên hàm là nó sẽ gọi. Nếu nhiều hàm trùng tên, nó sẽ gọi hàm cuối cùng tìm thấy

(function(\*){}) (\*); khai báo xong gọi luôn;

(function(name){ consoler.Log(“hello” +name)(“hieu”); =>hellohieu

### Prototype

Mọi function đều có prototype

* + Cho phép thêm thuộc tính hoặc phương thức vào đối tượng

Vd: function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
Person.prototype.nationality = "English"; // thêm trường nationlity

Person.prototype.name = function() {// thêm hàm name()  
  return this.firstName + " " + this.lastName;  
};

Tổng quát hơn ta có \:

String.prototype.translate = function(){var content = this; …};

* + Sử dụng:var english = “tiếng việt”.stranstale(); // “tiếng việt” = this;

## Table

Khi render dữ liệu. vd có 1000 row thì nó sẽ raw 1000 row rồi mới hiển thị

Ta có thể dùng giải pháp render bằng thẻ dip raw đến đâu hiển thị đến đấy

## Biến

### var

Là từ khóa định nghĩa 1 biến có pham vi truy cập trong 1 function. **Biến var** có tác dụng trong function mà nó được định nghĩa.

### let

Là từ khóa định nghĩa 1 biến có phạm vi truy cập trong 1 block – khối code. **Biến let** có phạm vi trong dấu 1 cặp dấu {} bao quanh nó

Let có tác dụng hơn var trong trường hợp:

*for (let i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

Khi đó i chỉ được hiểu trong khối lệnh của for

### const

Là từ khóa định nghĩa 1 biến sẽ là hằng số. **Biến const** lưu trữ giá trị không thể thay đổi được trong suốt vòng đời của biến.

## Argument

Vd:  
Lấy giờ địa phương theo định dạng ISO

[https://www.techrepublic.com/article/convert-the-local-time-to-another-time-zone-with-this-javascript/#:~:text=getTimezoneOffset()%20\*%2060000%3B,the%20location%20is%20behind%20UTC.](https://www.techrepublic.com/article/convert-the-local-time-to-another-time-zone-with-this-javascript/#:~:text=getTimezoneOffset() * 60000%3B,the location is behind UTC.)

Công thức:

var tzoffset = (new Date()).getTimezoneOffset() \* 60000; //offset in milliseconds

var localISOTime = (new Date(Date.now() - tzoffset)).toISOString().slice(0, -1);

## Splice

\*không áp dụng cho chuỗi

Splice(Vị trí bắt đầu, số lượng bị xóa, item thay thế, item thay tthế 2,..)

* Splice(Vị trí bắt đầu) : xóa từ vị trí đó đến hết
* Splice(Vị trí bắt đầu, , số lượng bị xóa) : xóa và không thay thế
* Splice(Vị trí bắt đầu, số lượng bị xóa, item thay thế, item thay tthế 2,..): các item sẽ đc chèn vào vị trí xóa

Hàm sẽ xóa trực tiếp vào phần tử và trả về những phần tử bị xóa

Vd:

var subject = ["php", "css", "html", "js"];

subject.splice(1);

* php

subject.splice(1, 2);

* php,js

subject.splice(1, 1, "python", "c#", "ios");

* php,python,c#,ios,html,js

## slice

slice(start)

slice(start, end)

* trả về giá trị củavị trí bắt đầu đến vị trí kết thúc của chuỗi -1
* nếu không có end thì lấy đến hết

vd:

const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));

// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));

// expected output: Array ["camel", "duck"]

## toFixed ()

Định dạng số chữ số sau dấy phảy

let numObj = 12345.6789

numObj.toFixed() // Returns '12346': note rounding, no fractional part

numObj.toFixed(1) // Returns '12345.7': note rounding

numObj.toFixed(6) // Returns '12345.678900': note added zeros

(1.23e+20).toFixed(2) // Returns '123000000000000000000.00'

(1.23e-10).toFixed(2) // Returns '0.00'

2.34.toFixed(1) // Returns '2.3'

2.35.toFixed(1) // Returns '2.4'. Note it rounds up

2.55.toFixed(1) // Returns '2.5'. Note it rounds down - see warning above

-2.34.toFixed(1) // Returns -2.3 (due to operator precedence, negative number literals don't return a string...)

(-2.34).toFixed(1) // Returns '-2.3'

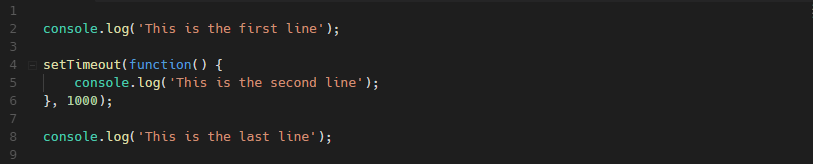
x = findMax(1, 123, 500, 115, 44, 88);  
  
function findMax() {  
  var i;  
  var max = -Infinity;  
  for (i = 0; i < arguments.length; i++) {  
    if (arguments[i] > max) {  
      max = arguments[i];  
    }  
  }  
  return max;  
}

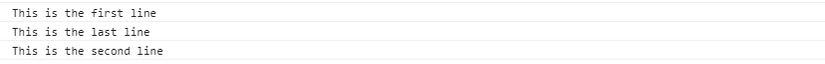
Nếu một hàm được gọi với **quá nhiều đối số** (nhiều hơn được khai báo), các đối số này có thể được truy cập bằng cách sử dụng **đối tượng đối số** .

## Bất đồng bộ

### số khái niệm

1 ví dụ về setTimeout quen thuộc

Kết quả chạy đoạn code:



#### CALL STACK

- là một dạng cấu trúc dữ liệu ghi lại vị trí các lệnh đang được thực hiện trong chương trình. Khi lệnh bắt đầu được thực hiện sẽ được đưa vào đỉnh của stack và sau khi thực hiện xong sẽ được lấy ra khỏi ngăn xếp.

#### **WEB APIs**

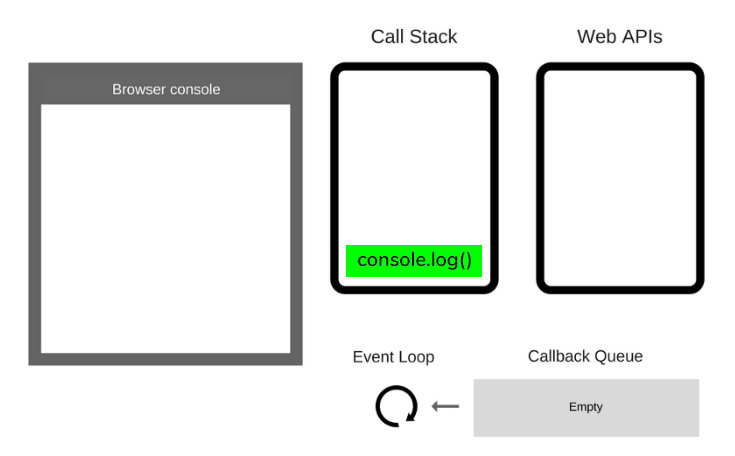
- vể bản chất đây chính là các thread mà ta không thể truy cập trực tiếp mà chỉ có thể gọi được đến nó. Các thread này do trình duyệt cung cấp.

#### **CALLBACK QUEUE**

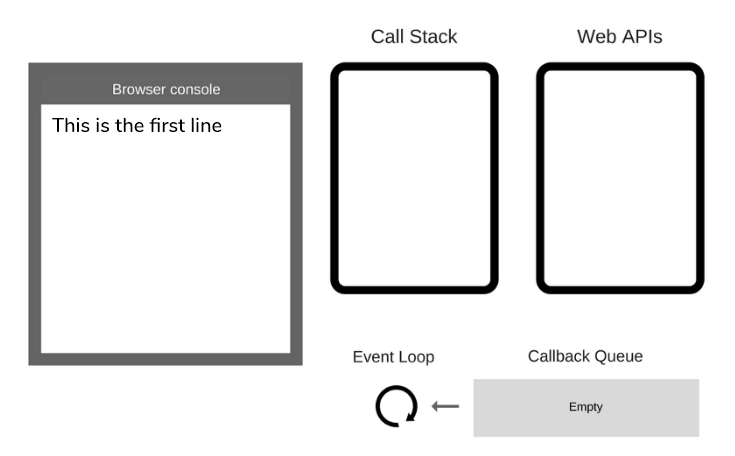
- là một dạng cấu trúc dữ liệu với nguyên tắc First-In-First-Out (vào trước ra trước).

#### EVENT LOOP

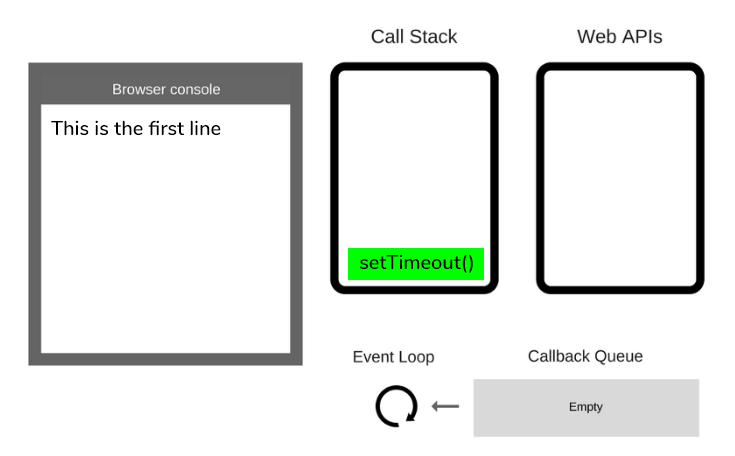
- có nhiệm vụ giám sát tình trạng của CALL STACK và CALLBACK QUEUE. Để hiểu được quá trình thực hiện của cơ chế bất đồng bộ ta sẽ đưa ví dụ thứ hai vào và thực hiện trong mô hình trên.



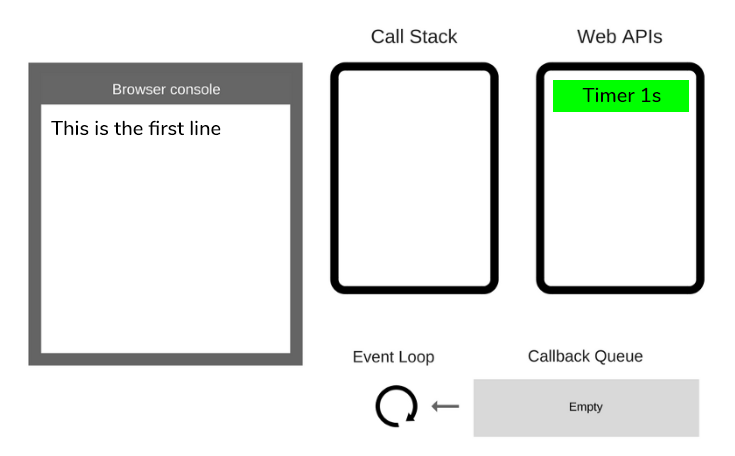
* Lệnh này lập tức trả về dòng chữ This is the first line đồng nghĩa với việc nó đã chạy xong và được đẩy ra khỏi **CALL STACK**.

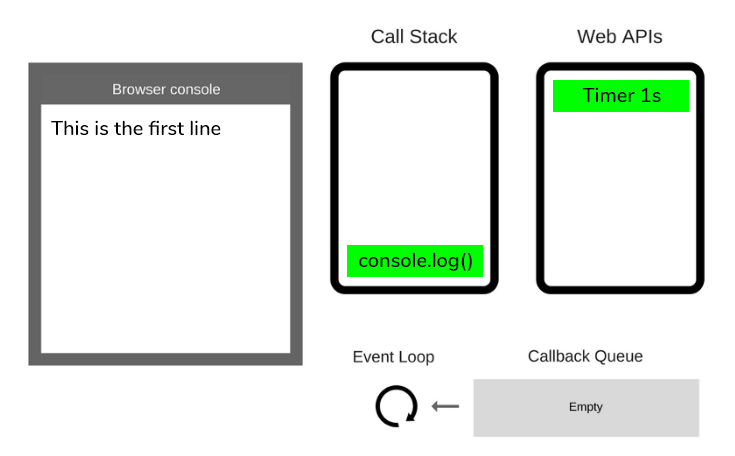
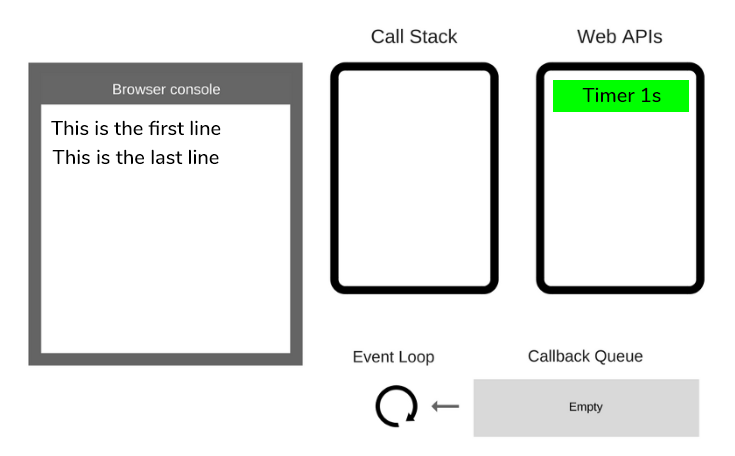


* Tiếp đến hàm setTimeout(function() { console.log('This is the second line'); }, 1000); được đưa vào trong **CALL STACK** để thực hiện.

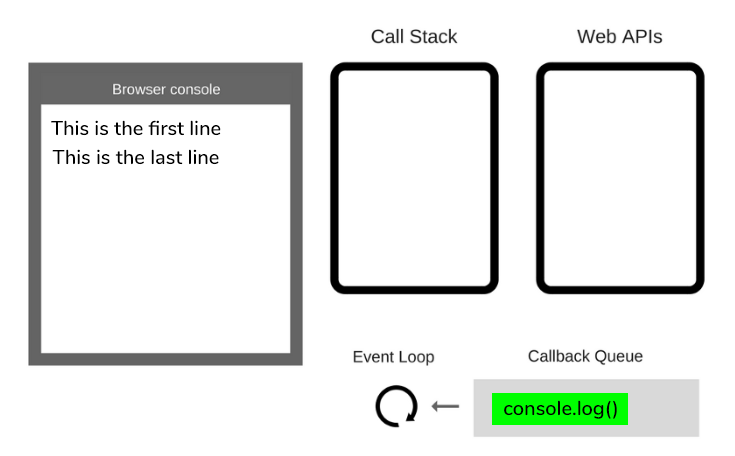


* Tuy nhiên hàm nay không trả về kết qua ngay mà phải đợi 1 giây. Hàm setTimeout() ở đây chính là một API mà **WEB APIs** cung cấp. Lập tức đoạn code này được chuyển vào trong **WEB APIs** và trình duyệt sẽ tạo ra một bộ hẹn giờ tương ứng với thời gian trên là 1 giây trước khi trả về kết quả

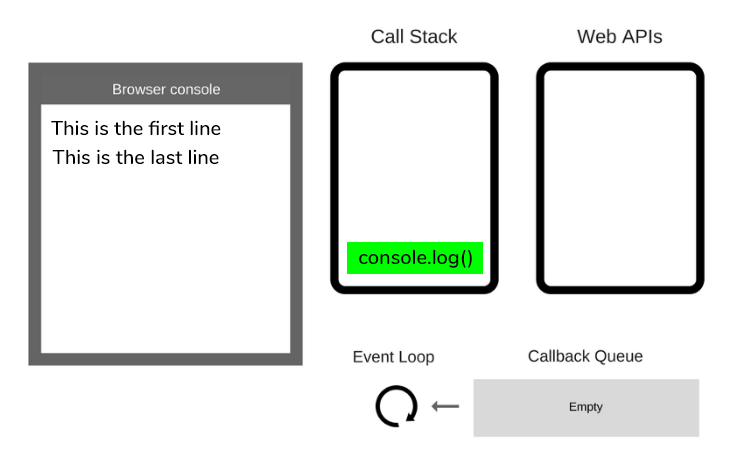


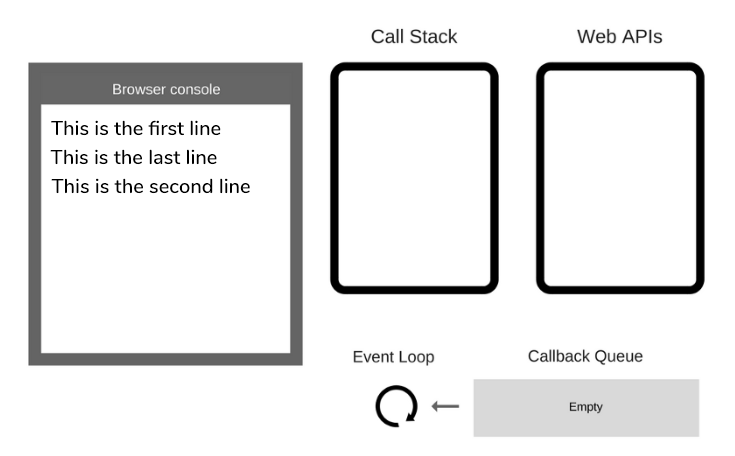
* Khi đoạn code thứ 2 được chuyển sang **WEB APIs** thì lập tức đoạn code cuối cùng console.log('This is the last line ') đã được đưa vào **CALL STACK** để thực
* hiện và trả về kết quả là dòng chữ This is the last line. Sau đó đoạn code này cũng được đẩy ra khỏi **CALL STACK**

* Sau khi bộ giờ trong **WEB APIs** chạy xong thì kết quả trả về lúc này không được in ngay ra màn hình mà nó được đẩy vào **CALLBACK QUEUE**



* **EVENT LOOP** với chức năng liên tục giám sát xem **CALL STACK** đã trống chưa và **CALLBACK QUEUE** có gì không. Lúc này **CALLBACK QUEUE** đang chư kết quả mà **WEB APIs** trả về nên và **CALL STACK** lúc này cũng đã trống do toàn bộ code trong chương trình đã được thực hiện nên nó sẽ đẩy kết quả trong **CALLBACK QUEUE** vào lại **CALL STACK** và đoạn code console.log(''This is the second line) được thực hiện và trả kết quả ra màn hình.



Đó là toàn bộ quá trình diễn ra trong cơ chế chạy bất đồng bộ của javascript, tuy nhiên ta cần chú ý một số điểm sau:

* **EVENT LOOP** chỉ tiến hành đẩy hàm trả về từ **CALLBACK QUEUE** sang **CALL STACK** khi **CALL STACK** đã thực hiện xong hết các đoạn code khác của chương trình
* Thứ tự các kết quả được trả về từ **WEB APIs** sang **CALLBACK QUEUE** sẽ không theo thứ tự đưa vào mà sẽ tùy thuộc hàm nào chạy xong trước sẽ được đẩy vào **CALLBACK QUEUE** trước đồng nghĩa với việc sẽ được chuyển qua **CALL STACK** trước.

### Call Back

Call Back là một hàm được truyền vào một hàm khác với tư cách như một tham số của hàm đó. Ví dụ như

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | function nauGa(callback)       nauNuocSoi();      vatLongGa();      callback();  }  function luocGa(){      //  }    function nuongGa(){  //  }  nauGa(luocGa);  nauGa(nuongGa); |

Ở đoạn mã trên, chúng ta thấy rằng, hàm luocGa và nuongGa được dùng như tham số trong hàm nauGa Với Javascript, một ngôn ngữ hướng sự kiện, call back được sử dụng rất nhiều khi xử lý các sự kiện, ví dụ như

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $('#button').click(function(){  alert("hê nô");  }) |

Chúng ta có thể áp dụng call back để đồng bộ hóa các đoạn mã không đồng bộ. Ví dụ như ở đoạn mã trên. Nấu nước sôi cần một khoảng thời gian chờ nước sôi, chúng ta không phải làm gì. Ta có thể biểu diễn thời gian chờ này bằng hàm setTimeout() trong javascript.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | function soCheGa(){     nauNuocSoi();     vatLongGa();  }  function nauNuocSoi(){     setTimeout(function(){       console.log("nau nuoc soi")     }, 1000)  }  function vatLongGa(){  console.log("vat long ga");  }  soCheGa(); |

Và nếu như theo đúng cách chạy của Javascript thì hành động vặt lông gà sẽ được thực hiện trước hành động nấu nước sôi. (à, quên mất, nấu nước sôi là để nhúng gà vào vặt lông chứ không phải để luộc gà đâu nhé ) .Mà nếu chúng ta vặt lông gà luôn mà không cần nhúng nước nóng thì tội cho bác gà quá. Vì thế để cho gà có thể ra đi thanh thản chúng ta cần đồng bộ hóa lại quy trình bằng callback như sau

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | function soCheGa(callback){        nauNuocSoi(vatLongGa);      }       function nauNuocSoi(callback){        setTimeout(function(){          console.log("nau nuoc soi");          callback();        }, 1000)      } |

Tuy nhiên, Callback cũng có nhược điểm. Đó là khi chúng ta muốn nhiều hành động bất đồng bộ thực hiện theo đúng thứ tự liên tiếp nhau, chúng ta phải gọi nhiều hàm callback lồng vào nhau nhiều lần, gây ra đoạn code rất khó kiểm soát và không tối ưu. Đây gọi là tình trạng Callback Hell. Ví dụ như muốn in các số từ 1 đến 10, mà mỗi hành động in đều là một hàm bất đồng bộ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | function printNumber(number, callback){    setTimeout(      () =&gt; {        console.log(number)        callback()      },      Math.floor(Math.random() \* 100) + 1    )  }    function printAll(){    printNumber(1, function(){      printNumber(2, function(){        printNumber(3, function(){          printNumber(4, function(){            printNumber(5, function(){              printNumber(6, function(){                printNumber(7, function(){                  printNumber(8, function(){                    printNumber(9, function(){                      printNumber(10, function(){                        })                    })                  })                })              })            })          })        })      })    })  } |

### Promise

Promise là một đối tượng bao hàm một hàm chứa các đoạn code không đồng bộ. Hàm này chứa 2 tham số là hai hàm callback để giải quyết sau khi mã đồng bộ thực hiện thành công hay thất bại. Promise cung cấp cho ta hai phương thức xử lý sau khi đoạn mã bất đồng bộ thực hiện thành công hoặc thất bại. Hàm then() dùng để xử lý sau khi mã bất đồng bộ được thực hiện thành công và hàm catch() dùng để xử lý sau khi mã bất đồng bộ thực hiện thất bại

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | function printNumber(number){      return new Promise((resolve, reject) =&gt; {        setTimeout(          () =&gt; {          if(number&lt;0){           reject();           }else{               console.log(number);               resolve();           }          },        1000        )      })  }  printNumber(1)  .then(()=&gt;printNumber(2))  .reject(()=&gt;console.log("number &lt; 0")) |

Phương thức then có thể thực thi một hàm, một Promise hay một đối tượng. Nếu chúng ta dùng then để trả về một Promise thì ta có thể tận dụng để xử lý tình trạng Callback Hell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | printNumber(1)  .then(()=&gt;printNumber(2))  .then(()=&gt;printNumber(3))  .then(()=&gt;printNumber(4))  .then(()=&gt;printNumber(5))  .then(()=&gt;printNumber(6))  .then(()=&gt;printNumber(7))  .then(()=&gt;printNumber(8))  .reject(()=&gt;console.log("number &lt; 0")) |

Tuy nhiên, dù Promise đã giải quyết được vấn đề Callback Hell, nhưng chúng ta có thể thấy, đoạn mã vẫn chưa thực sự rõ ràng và dễ hiểu. Trong phương thức chúng ta vẫn phải truyền vào một hàm, mà hàm đó trả về một hàm khác có giá trị trả về là 1 Promise. Chúng ta tạm gọi đây là tình trạng Promise Hell.

Asyn/Await

Async / Await là một tính năng ngôn ngữ là một phần của tiêu chuẩn ES8. Từ khóa Async để khai báo rằng hàm này sẽ xử lý các hàm bất đồng bộ, nó sẽ chờ kết quả của các hàm bất đồng bộ được trả về sau đó mới thực hiện tiếp. Hàm bất đồng bộ đó phải trả về một Promise và được khai báo với từ khóa Await

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | function printNumber(number){    return new Promise((resolve, reject) =&gt; {      setTimeout(        () =&gt; {         console.log(number)         resolve()        },       Math.floor(Math.random() \* 100) + 1      )    })  }    async function printAll(){    await printNumber(1);    await printNumber(2);    await printNumber(4);    await printNumber(5);    await printNumber(6);  }  printAll(); |

Đến đây thì chúng ta đã thấy Asyn/Await đã giải quyết triệt để được tình trạng Callback Hell cũng như Promise Hell.

-Với 1 hàm bất kỳ vd: async function say(){ return "hi"; }

Khi async đối tượng trả về luôn được chuyển thành promise (kể cả không return



Nếu muốn lấy kết quả của hàm async ta cần thêm await

### Promise.all

# jQuery

## list

### tạo mới

### vd: var cartList = [];

### thêm phần tử vào mảng

vd:

cartList.push({

Amount: $(item).val(),

Products: {

id\_product: $(item).data('id')

}

});

## object

### tạo mới

var <tên đối tượng> = {

<tên thuộc tính>: <giá trị của thuộc tính>,

<tên thuộc tính>: <giá trị của thuộc tính>,

....

}

vd:

var bill = {

Item: cartList,

Id\_customer: $('#txtCustomer').data('id'),

Subtotal : $('#SubTotal').text(),

Total: $('#Total').text(),

DiscountCode: $('#txtdiscountCode').val()

}

hoặc;

## for()

*for (var i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

*for (let i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

## Function

Function <tên hàm> (tham số){..}

Nó không quan tâm tham số(thừa thiếu, sai kiểu dữ liệu không quan trọng). chỉ cần đúng tên hàm là nó sẽ gọi. Nếu nhiều hàm trùng tên, nó sẽ gọi hàm cuối cùng tìm thấy

(function(\*){}) (\*); khai báo xong gọi luôn;

(function(name){ consoler.Log(“hello” +name)(“hieu”); =>hellohieu

### Prototype

Mọi function đều có prototype

* + Cho phép thêm thuộc tính hoặc phương thức vào đối tượng

Vd: function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
Person.prototype.nationality = "English"; // thêm trường nationlity

Person.prototype.name = function() {// thêm hàm name()  
  return this.firstName + " " + this.lastName;  
};

Tổng quát hơn ta có \:

String.prototype.translate = function(){var content = this; …};

* + Sử dụng:var english = “tiếng việt”.stranstale(); // “tiếng việt” = this;

## Table

Khi render dữ liệu. vd có 1000 row thì nó sẽ raw 1000 row rồi mới hiển thị

Ta có thể dùng giải pháp render bằng thẻ dip raw đến đâu hiển thị đến đấy

## Biến

### var

Là từ khóa định nghĩa 1 biến có pham vi truy cập trong 1 function. **Biến var** có tác dụng trong function mà nó được định nghĩa.

### let

Là từ khóa định nghĩa 1 biến có phạm vi truy cập trong 1 block – khối code. **Biến let** có phạm vi trong dấu 1 cặp dấu {} bao quanh nó

Let có tác dụng hơn var trong trường hợp:

*for (let i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

Khi đó i chỉ được hiểu trong khối lệnh của for

### const

Là từ khóa định nghĩa 1 biến sẽ là hằng số. **Biến const** lưu trữ giá trị không thể thay đổi được trong suốt vòng đời của biến.

## Argument

Vd:  
Lấy giờ địa phương theo định dạng ISO

[https://www.techrepublic.com/article/convert-the-local-time-to-another-time-zone-with-this-javascript/#:~:text=getTimezoneOffset()%20\*%2060000%3B,the%20location%20is%20behind%20UTC.](https://www.techrepublic.com/article/convert-the-local-time-to-another-time-zone-with-this-javascript/#:~:text=getTimezoneOffset() * 60000%3B,the location is behind UTC.)

Công thức:

var tzoffset = (new Date()).getTimezoneOffset() \* 60000; //offset in milliseconds

var localISOTime = (new Date(Date.now() - tzoffset)).toISOString().slice(0, -1);

## Splice

\*không áp dụng cho chuỗi

Splice(Vị trí bắt đầu, số lượng bị xóa, item thay thế, item thay tthế 2,..)

* Splice(Vị trí bắt đầu) : xóa từ vị trí đó đến hết
* Splice(Vị trí bắt đầu, , số lượng bị xóa) : xóa và không thay thế
* Splice(Vị trí bắt đầu, số lượng bị xóa, item thay thế, item thay tthế 2,..): các item sẽ đc chèn vào vị trí xóa

Hàm sẽ xóa trực tiếp vào phần tử và trả về những phần tử bị xóa

Vd:

var subject = ["php", "css", "html", "js"];

subject.splice(1);

* php

subject.splice(1, 2);

* php,js

subject.splice(1, 1, "python", "c#", "ios");

* php,python,c#,ios,html,js

## slice

slice(start)

slice(start, end)

* trả về giá trị củavị trí bắt đầu đến vị trí kết thúc của chuỗi -1
* nếu không có end thì lấy đến hết

vd:

const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));

// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));

// expected output: Array ["camel", "duck"]

## toFixed ()

Định dạng số chữ số sau dấy phảy

let numObj = 12345.6789

numObj.toFixed() // Returns '12346': note rounding, no fractional part

numObj.toFixed(1) // Returns '12345.7': note rounding

numObj.toFixed(6) // Returns '12345.678900': note added zeros

(1.23e+20).toFixed(2) // Returns '123000000000000000000.00'

(1.23e-10).toFixed(2) // Returns '0.00'

2.34.toFixed(1) // Returns '2.3'

2.35.toFixed(1) // Returns '2.4'. Note it rounds up

2.55.toFixed(1) // Returns '2.5'. Note it rounds down - see warning above

-2.34.toFixed(1) // Returns -2.3 (due to operator precedence, negative number literals don't return a string...)

(-2.34).toFixed(1) // Returns '-2.3'

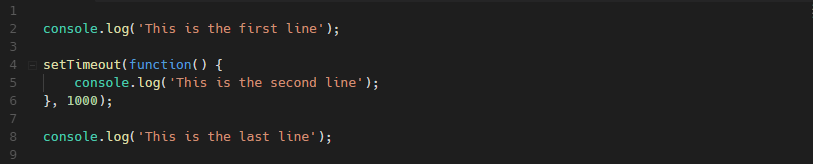
x = findMax(1, 123, 500, 115, 44, 88);  
  
function findMax() {  
  var i;  
  var max = -Infinity;  
  for (i = 0; i < arguments.length; i++) {  
    if (arguments[i] > max) {  
      max = arguments[i];  
    }  
  }  
  return max;  
}

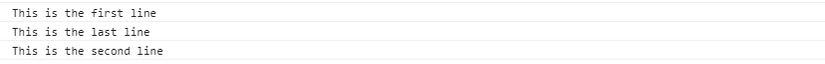
Nếu một hàm được gọi với **quá nhiều đối số** (nhiều hơn được khai báo), các đối số này có thể được truy cập bằng cách sử dụng **đối tượng đối số** .

## Bất đồng bộ

### số khái niệm

1 ví dụ về setTimeout quen thuộc

Kết quả chạy đoạn code:



#### CALL STACK

- là một dạng cấu trúc dữ liệu ghi lại vị trí các lệnh đang được thực hiện trong chương trình. Khi lệnh bắt đầu được thực hiện sẽ được đưa vào đỉnh của stack và sau khi thực hiện xong sẽ được lấy ra khỏi ngăn xếp.

#### **WEB APIs**

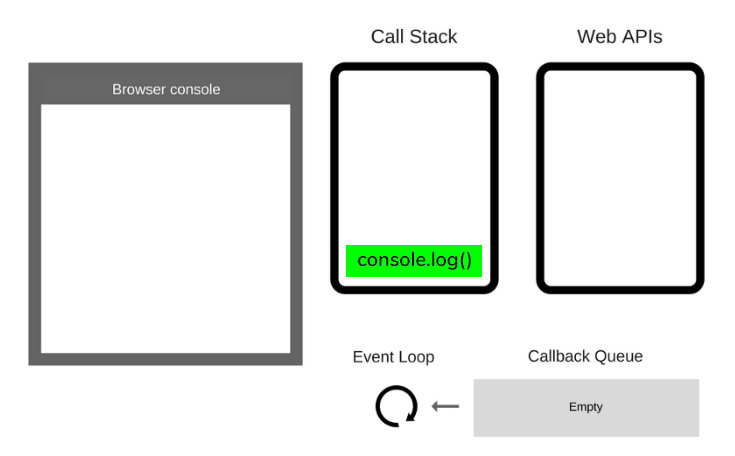
- vể bản chất đây chính là các thread mà ta không thể truy cập trực tiếp mà chỉ có thể gọi được đến nó. Các thread này do trình duyệt cung cấp.

#### **CALLBACK QUEUE**

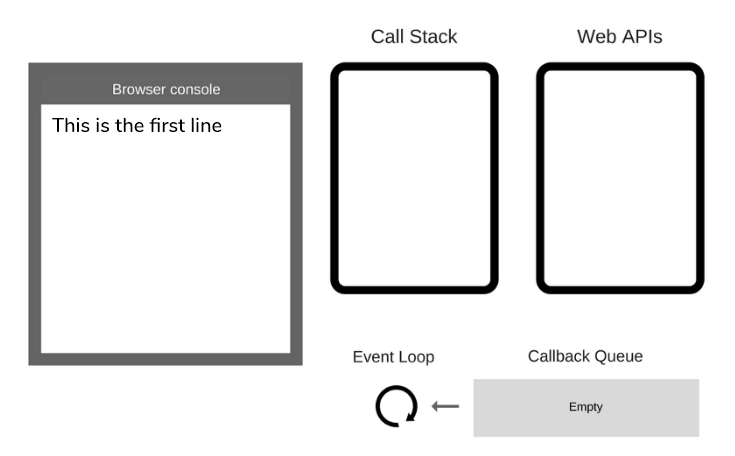
- là một dạng cấu trúc dữ liệu với nguyên tắc First-In-First-Out (vào trước ra trước).

#### EVENT LOOP

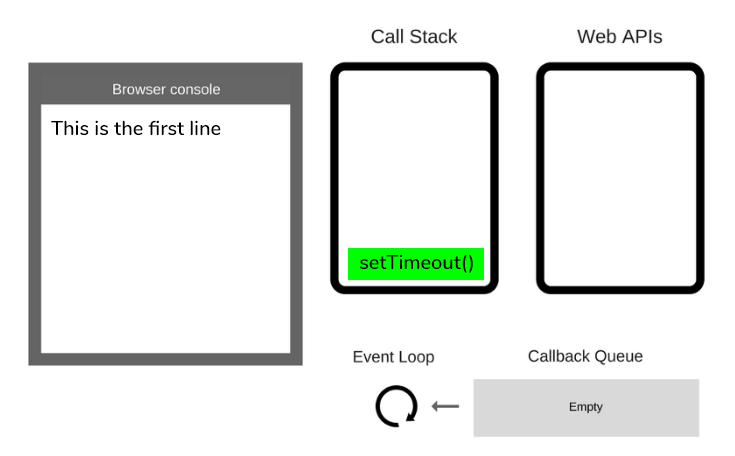
- có nhiệm vụ giám sát tình trạng của CALL STACK và CALLBACK QUEUE. Để hiểu được quá trình thực hiện của cơ chế bất đồng bộ ta sẽ đưa ví dụ thứ hai vào và thực hiện trong mô hình trên.



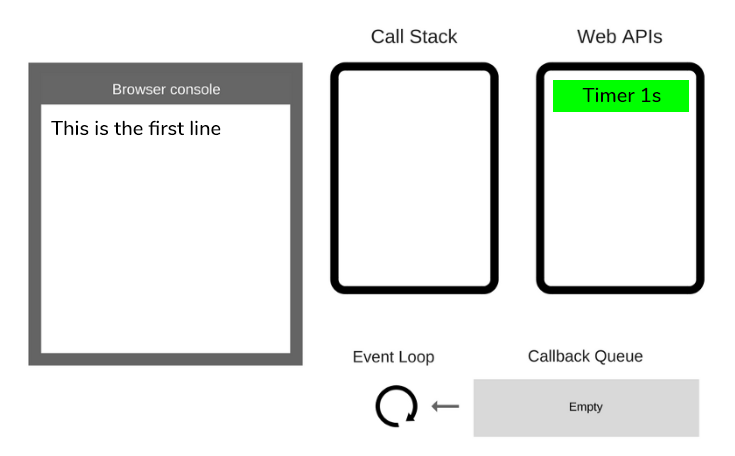
* Lệnh này lập tức trả về dòng chữ This is the first line đồng nghĩa với việc nó đã chạy xong và được đẩy ra khỏi **CALL STACK**.

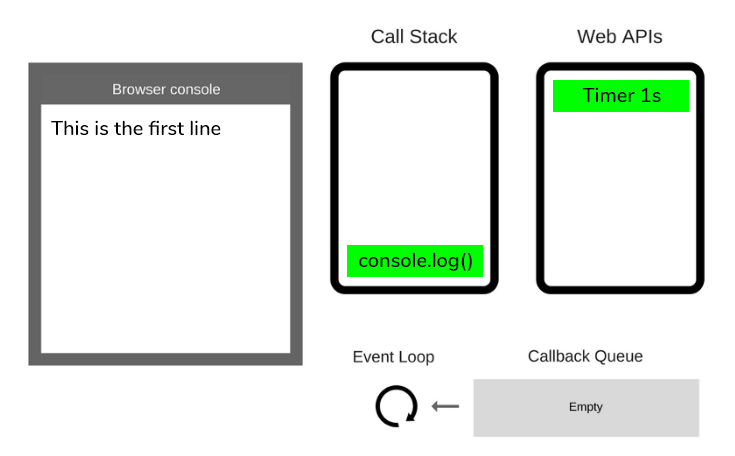
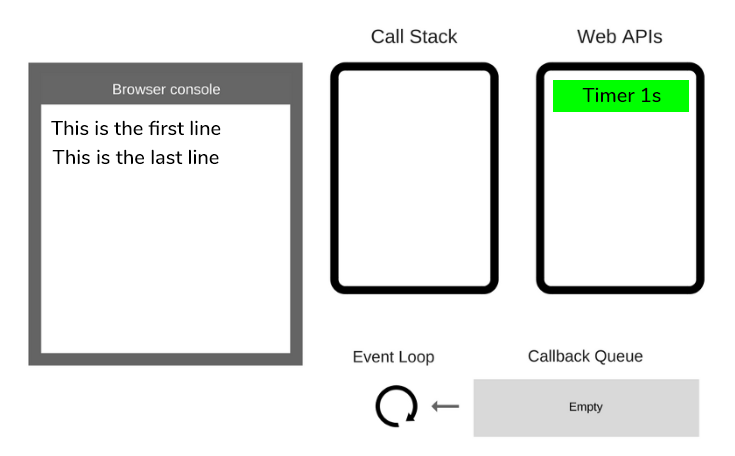


* Tiếp đến hàm setTimeout(function() { console.log('This is the second line'); }, 1000); được đưa vào trong **CALL STACK** để thực hiện.

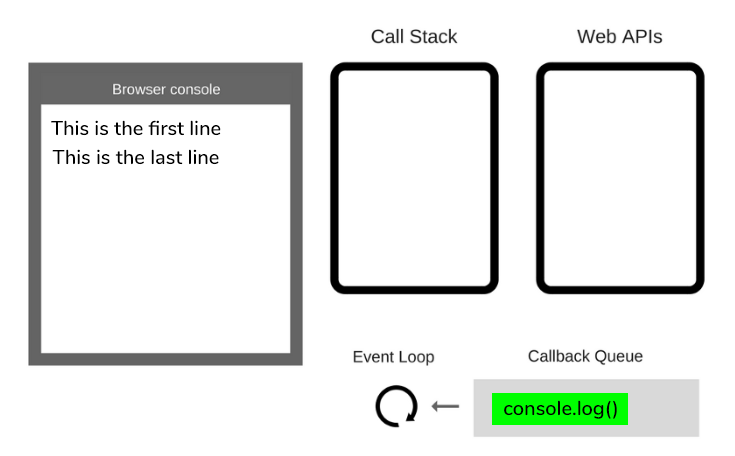


* Tuy nhiên hàm nay không trả về kết qua ngay mà phải đợi 1 giây. Hàm setTimeout() ở đây chính là một API mà **WEB APIs** cung cấp. Lập tức đoạn code này được chuyển vào trong **WEB APIs** và trình duyệt sẽ tạo ra một bộ hẹn giờ tương ứng với thời gian trên là 1 giây trước khi trả về kết quả

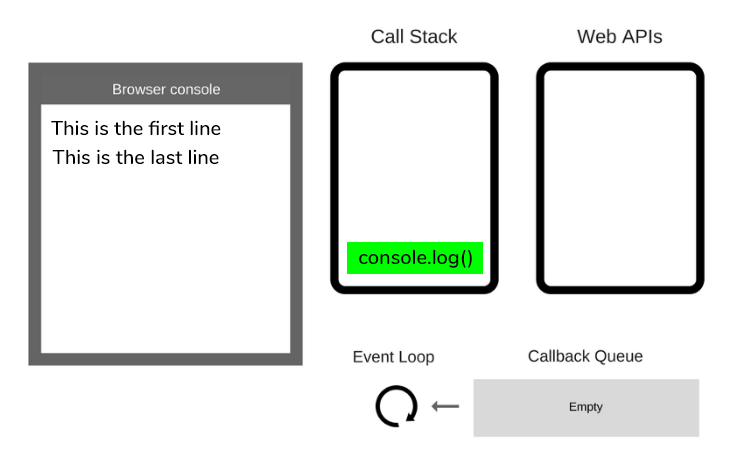


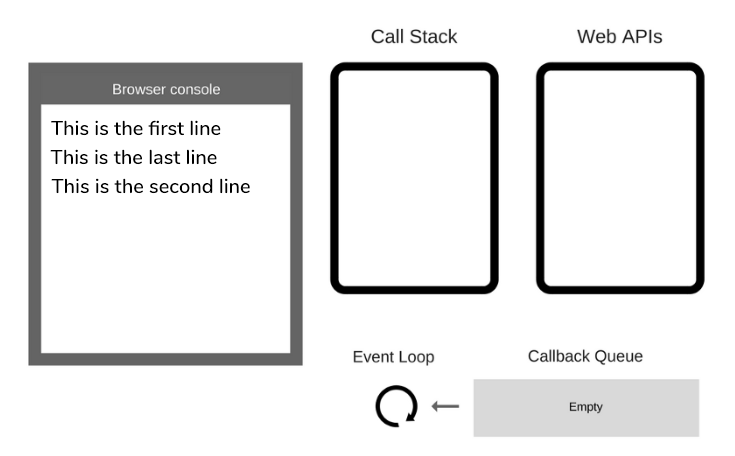
* Khi đoạn code thứ 2 được chuyển sang **WEB APIs** thì lập tức đoạn code cuối cùng console.log('This is the last line ') đã được đưa vào **CALL STACK** để thực
* hiện và trả về kết quả là dòng chữ This is the last line. Sau đó đoạn code này cũng được đẩy ra khỏi **CALL STACK**

* Sau khi bộ giờ trong **WEB APIs** chạy xong thì kết quả trả về lúc này không được in ngay ra màn hình mà nó được đẩy vào **CALLBACK QUEUE**



* **EVENT LOOP** với chức năng liên tục giám sát xem **CALL STACK** đã trống chưa và **CALLBACK QUEUE** có gì không. Lúc này **CALLBACK QUEUE** đang chư kết quả mà **WEB APIs** trả về nên và **CALL STACK** lúc này cũng đã trống do toàn bộ code trong chương trình đã được thực hiện nên nó sẽ đẩy kết quả trong **CALLBACK QUEUE** vào lại **CALL STACK** và đoạn code console.log(''This is the second line) được thực hiện và trả kết quả ra màn hình.



Đó là toàn bộ quá trình diễn ra trong cơ chế chạy bất đồng bộ của javascript, tuy nhiên ta cần chú ý một số điểm sau:

* **EVENT LOOP** chỉ tiến hành đẩy hàm trả về từ **CALLBACK QUEUE** sang **CALL STACK** khi **CALL STACK** đã thực hiện xong hết các đoạn code khác của chương trình
* Thứ tự các kết quả được trả về từ **WEB APIs** sang **CALLBACK QUEUE** sẽ không theo thứ tự đưa vào mà sẽ tùy thuộc hàm nào chạy xong trước sẽ được đẩy vào **CALLBACK QUEUE** trước đồng nghĩa với việc sẽ được chuyển qua **CALL STACK** trước.

### Call Back

Call Back là một hàm được truyền vào một hàm khác với tư cách như một tham số của hàm đó. Ví dụ như

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | function nauGa(callback)       nauNuocSoi();      vatLongGa();      callback();  }  function luocGa(){      //  }    function nuongGa(){  //  }  nauGa(luocGa);  nauGa(nuongGa); |

Ở đoạn mã trên, chúng ta thấy rằng, hàm luocGa và nuongGa được dùng như tham số trong hàm nauGa Với Javascript, một ngôn ngữ hướng sự kiện, call back được sử dụng rất nhiều khi xử lý các sự kiện, ví dụ như

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $('#button').click(function(){  alert("hê nô");  }) |

Chúng ta có thể áp dụng call back để đồng bộ hóa các đoạn mã không đồng bộ. Ví dụ như ở đoạn mã trên. Nấu nước sôi cần một khoảng thời gian chờ nước sôi, chúng ta không phải làm gì. Ta có thể biểu diễn thời gian chờ này bằng hàm setTimeout() trong javascript.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | function soCheGa(){     nauNuocSoi();     vatLongGa();  }  function nauNuocSoi(){     setTimeout(function(){       console.log("nau nuoc soi")     }, 1000)  }  function vatLongGa(){  console.log("vat long ga");  }  soCheGa(); |

Và nếu như theo đúng cách chạy của Javascript thì hành động vặt lông gà sẽ được thực hiện trước hành động nấu nước sôi. (à, quên mất, nấu nước sôi là để nhúng gà vào vặt lông chứ không phải để luộc gà đâu nhé ) .Mà nếu chúng ta vặt lông gà luôn mà không cần nhúng nước nóng thì tội cho bác gà quá. Vì thế để cho gà có thể ra đi thanh thản chúng ta cần đồng bộ hóa lại quy trình bằng callback như sau

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | function soCheGa(callback){        nauNuocSoi(vatLongGa);      }       function nauNuocSoi(callback){        setTimeout(function(){          console.log("nau nuoc soi");          callback();        }, 1000)      } |

Tuy nhiên, Callback cũng có nhược điểm. Đó là khi chúng ta muốn nhiều hành động bất đồng bộ thực hiện theo đúng thứ tự liên tiếp nhau, chúng ta phải gọi nhiều hàm callback lồng vào nhau nhiều lần, gây ra đoạn code rất khó kiểm soát và không tối ưu. Đây gọi là tình trạng Callback Hell. Ví dụ như muốn in các số từ 1 đến 10, mà mỗi hành động in đều là một hàm bất đồng bộ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | function printNumber(number, callback){    setTimeout(      () =&gt; {        console.log(number)        callback()      },      Math.floor(Math.random() \* 100) + 1    )  }    function printAll(){    printNumber(1, function(){      printNumber(2, function(){        printNumber(3, function(){          printNumber(4, function(){            printNumber(5, function(){              printNumber(6, function(){                printNumber(7, function(){                  printNumber(8, function(){                    printNumber(9, function(){                      printNumber(10, function(){                        })                    })                  })                })              })            })          })        })      })    })  } |

### Promise

Promise là một đối tượng bao hàm một hàm chứa các đoạn code không đồng bộ. Hàm này chứa 2 tham số là hai hàm callback để giải quyết sau khi mã đồng bộ thực hiện thành công hay thất bại. Promise cung cấp cho ta hai phương thức xử lý sau khi đoạn mã bất đồng bộ thực hiện thành công hoặc thất bại. Hàm then() dùng để xử lý sau khi mã bất đồng bộ được thực hiện thành công và hàm catch() dùng để xử lý sau khi mã bất đồng bộ thực hiện thất bại

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | function printNumber(number){      return new Promise((resolve, reject) =&gt; {        setTimeout(          () =&gt; {          if(number&lt;0){           reject();           }else{               console.log(number);               resolve();           }          },        1000        )      })  }  printNumber(1)  .then(()=&gt;printNumber(2))  .reject(()=&gt;console.log("number &lt; 0")) |

Phương thức then có thể thực thi một hàm, một Promise hay một đối tượng. Nếu chúng ta dùng then để trả về một Promise thì ta có thể tận dụng để xử lý tình trạng Callback Hell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | printNumber(1)  .then(()=&gt;printNumber(2))  .then(()=&gt;printNumber(3))  .then(()=&gt;printNumber(4))  .then(()=&gt;printNumber(5))  .then(()=&gt;printNumber(6))  .then(()=&gt;printNumber(7))  .then(()=&gt;printNumber(8))  .reject(()=&gt;console.log("number &lt; 0")) |

Tuy nhiên, dù Promise đã giải quyết được vấn đề Callback Hell, nhưng chúng ta có thể thấy, đoạn mã vẫn chưa thực sự rõ ràng và dễ hiểu. Trong phương thức chúng ta vẫn phải truyền vào một hàm, mà hàm đó trả về một hàm khác có giá trị trả về là 1 Promise. Chúng ta tạm gọi đây là tình trạng Promise Hell.

Asyn/Await

Async / Await là một tính năng ngôn ngữ là một phần của tiêu chuẩn ES8. Từ khóa Async để khai báo rằng hàm này sẽ xử lý các hàm bất đồng bộ, nó sẽ chờ kết quả của các hàm bất đồng bộ được trả về sau đó mới thực hiện tiếp. Hàm bất đồng bộ đó phải trả về một Promise và được khai báo với từ khóa Await

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | function printNumber(number){    return new Promise((resolve, reject) =&gt; {      setTimeout(        () =&gt; {         console.log(number)         resolve()        },       Math.floor(Math.random() \* 100) + 1      )    })  }    async function printAll(){    await printNumber(1);    await printNumber(2);    await printNumber(4);    await printNumber(5);    await printNumber(6);  }  printAll(); |

Đến đây thì chúng ta đã thấy Asyn/Await đã giải quyết triệt để được tình trạng Callback Hell cũng như Promise Hell.

-Với 1 hàm bất kỳ vd: async function say(){ return "hi"; }

Khi async đối tượng trả về luôn được chuyển thành promise (kể cả không return



Nếu muốn lấy kết quả của hàm async ta cần thêm await

### Promise.all

# JavarScripts

## list

### tạo mới

### vd: var cartList = [];

### thêm phần tử vào mảng

vd:

cartList.push({

Amount: $(item).val(),

Products: {

id\_product: $(item).data('id')

}

});

## object

### tạo mới

var <tên đối tượng> = {

<tên thuộc tính>: <giá trị của thuộc tính>,

<tên thuộc tính>: <giá trị của thuộc tính>,

....

}

vd:

var bill = {

Item: cartList,

Id\_customer: $('#txtCustomer').data('id'),

Subtotal : $('#SubTotal').text(),

Total: $('#Total').text(),

DiscountCode: $('#txtdiscountCode').val()

}

hoặc;

## for()

*for (var i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

*for (let i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

## Function

Function <tên hàm> (tham số){..}

Nó không quan tâm tham số(thừa thiếu, sai kiểu dữ liệu không quan trọng). chỉ cần đúng tên hàm là nó sẽ gọi. Nếu nhiều hàm trùng tên, nó sẽ gọi hàm cuối cùng tìm thấy

(function(\*){}) (\*); khai báo xong gọi luôn;

(function(name){ consoler.Log(“hello” +name)(“hieu”); =>hellohieu

### Prototype

Mọi function đều có prototype

* + Cho phép thêm thuộc tính hoặc phương thức vào đối tượng

Vd: function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
Person.prototype.nationality = "English"; // thêm trường nationlity

Person.prototype.name = function() {// thêm hàm name()  
  return this.firstName + " " + this.lastName;  
};

Tổng quát hơn ta có \:

String.prototype.translate = function(){var content = this; …};

* + Sử dụng:var english = “tiếng việt”.stranstale(); // “tiếng việt” = this;

## Table

Khi render dữ liệu. vd có 1000 row thì nó sẽ raw 1000 row rồi mới hiển thị

Ta có thể dùng giải pháp render bằng thẻ dip raw đến đâu hiển thị đến đấy

## Biến

### var

Là từ khóa định nghĩa 1 biến có pham vi truy cập trong 1 function. **Biến var** có tác dụng trong function mà nó được định nghĩa.

### let

Là từ khóa định nghĩa 1 biến có phạm vi truy cập trong 1 block – khối code. **Biến let** có phạm vi trong dấu 1 cặp dấu {} bao quanh nó

Let có tác dụng hơn var trong trường hợp:

*for (let i = 0; i < 5; i++) {*

*callbacks.push( function() { return i; } );*

*}*

Khi đó i chỉ được hiểu trong khối lệnh của for

### const

Là từ khóa định nghĩa 1 biến sẽ là hằng số. **Biến const** lưu trữ giá trị không thể thay đổi được trong suốt vòng đời của biến.

## Argument

Vd:  
Lấy giờ địa phương theo định dạng ISO

[https://www.techrepublic.com/article/convert-the-local-time-to-another-time-zone-with-this-javascript/#:~:text=getTimezoneOffset()%20\*%2060000%3B,the%20location%20is%20behind%20UTC.](https://www.techrepublic.com/article/convert-the-local-time-to-another-time-zone-with-this-javascript/#:~:text=getTimezoneOffset() * 60000%3B,the location is behind UTC.)

Công thức:

var tzoffset = (new Date()).getTimezoneOffset() \* 60000; //offset in milliseconds

var localISOTime = (new Date(Date.now() - tzoffset)).toISOString().slice(0, -1);

## Splice

\*không áp dụng cho chuỗi

Splice(Vị trí bắt đầu, số lượng bị xóa, item thay thế, item thay tthế 2,..)

* Splice(Vị trí bắt đầu) : xóa từ vị trí đó đến hết
* Splice(Vị trí bắt đầu, , số lượng bị xóa) : xóa và không thay thế
* Splice(Vị trí bắt đầu, số lượng bị xóa, item thay thế, item thay tthế 2,..): các item sẽ đc chèn vào vị trí xóa

Hàm sẽ xóa trực tiếp vào phần tử và trả về những phần tử bị xóa

Vd:

var subject = ["php", "css", "html", "js"];

subject.splice(1);

* php

subject.splice(1, 2);

* php,js

subject.splice(1, 1, "python", "c#", "ios");

* php,python,c#,ios,html,js

## slice

slice(start)

slice(start, end)

* trả về giá trị củavị trí bắt đầu đến vị trí kết thúc của chuỗi -1
* nếu không có end thì lấy đến hết

vd:

const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));

// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));

// expected output: Array ["camel", "duck"]

## toFixed ()

Định dạng số chữ số sau dấy phảy

let numObj = 12345.6789

numObj.toFixed() // Returns '12346': note rounding, no fractional part

numObj.toFixed(1) // Returns '12345.7': note rounding

numObj.toFixed(6) // Returns '12345.678900': note added zeros

(1.23e+20).toFixed(2) // Returns '123000000000000000000.00'

(1.23e-10).toFixed(2) // Returns '0.00'

2.34.toFixed(1) // Returns '2.3'

2.35.toFixed(1) // Returns '2.4'. Note it rounds up

2.55.toFixed(1) // Returns '2.5'. Note it rounds down - see warning above

-2.34.toFixed(1) // Returns -2.3 (due to operator precedence, negative number literals don't return a string...)

(-2.34).toFixed(1) // Returns '-2.3'

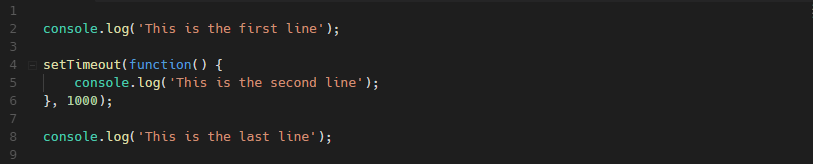
x = findMax(1, 123, 500, 115, 44, 88);  
  
function findMax() {  
  var i;  
  var max = -Infinity;  
  for (i = 0; i < arguments.length; i++) {  
    if (arguments[i] > max) {  
      max = arguments[i];  
    }  
  }  
  return max;  
}

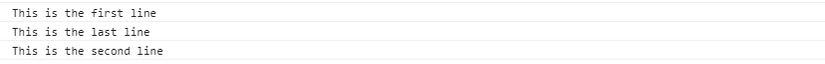
Nếu một hàm được gọi với **quá nhiều đối số** (nhiều hơn được khai báo), các đối số này có thể được truy cập bằng cách sử dụng **đối tượng đối số** .

## Bất đồng bộ

### số khái niệm

1 ví dụ về setTimeout quen thuộc

Kết quả chạy đoạn code:



#### CALL STACK

- là một dạng cấu trúc dữ liệu ghi lại vị trí các lệnh đang được thực hiện trong chương trình. Khi lệnh bắt đầu được thực hiện sẽ được đưa vào đỉnh của stack và sau khi thực hiện xong sẽ được lấy ra khỏi ngăn xếp.

#### **WEB APIs**

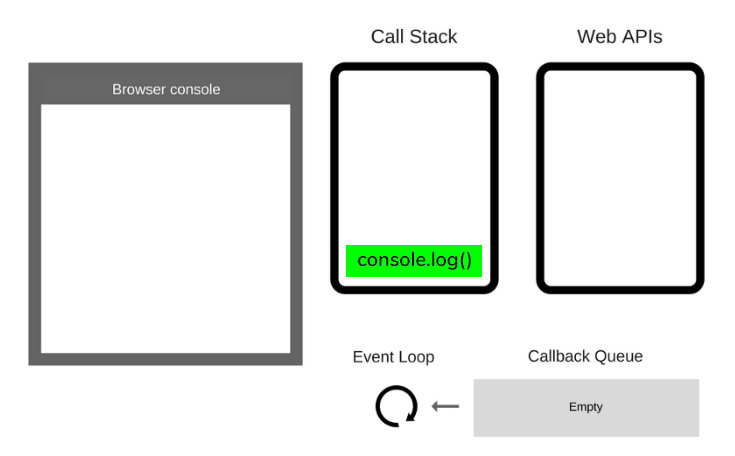
- vể bản chất đây chính là các thread mà ta không thể truy cập trực tiếp mà chỉ có thể gọi được đến nó. Các thread này do trình duyệt cung cấp.

#### **CALLBACK QUEUE**

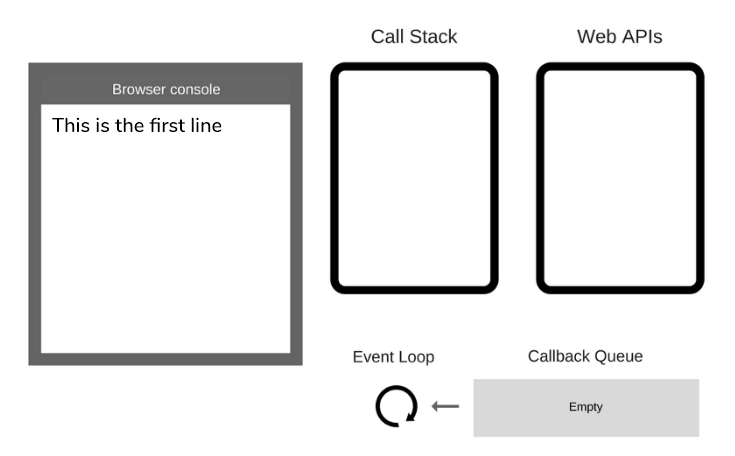
- là một dạng cấu trúc dữ liệu với nguyên tắc First-In-First-Out (vào trước ra trước).

#### EVENT LOOP

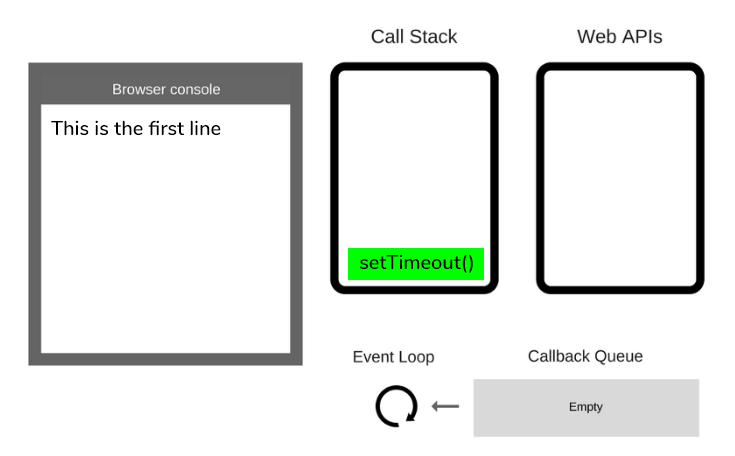
- có nhiệm vụ giám sát tình trạng của CALL STACK và CALLBACK QUEUE. Để hiểu được quá trình thực hiện của cơ chế bất đồng bộ ta sẽ đưa ví dụ thứ hai vào và thực hiện trong mô hình trên.



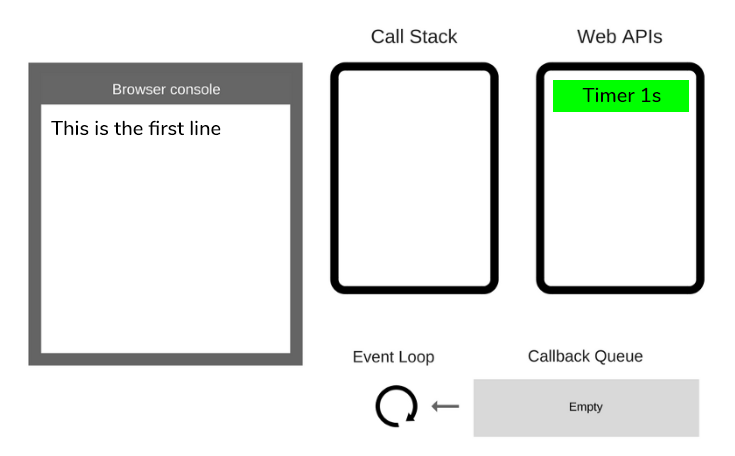
* Lệnh này lập tức trả về dòng chữ This is the first line đồng nghĩa với việc nó đã chạy xong và được đẩy ra khỏi **CALL STACK**.

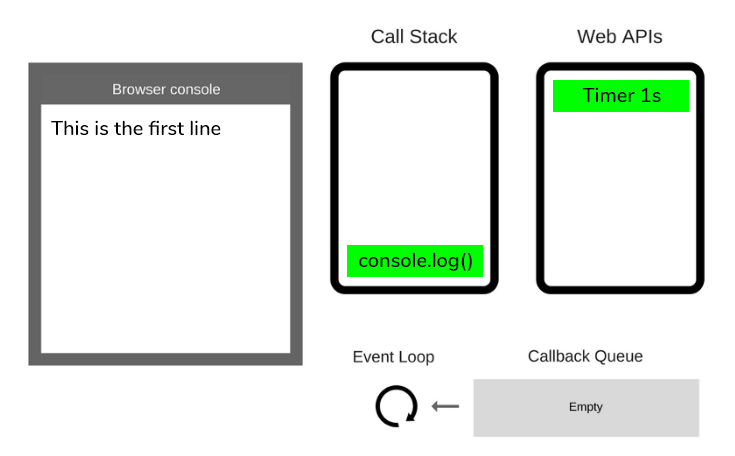
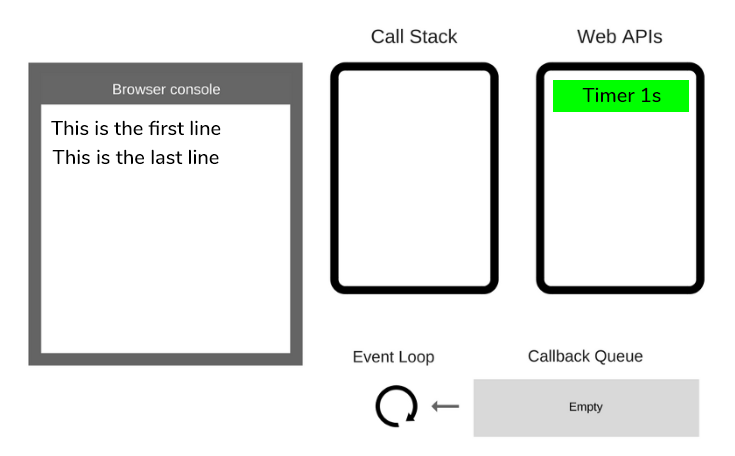


* Tiếp đến hàm setTimeout(function() { console.log('This is the second line'); }, 1000); được đưa vào trong **CALL STACK** để thực hiện.

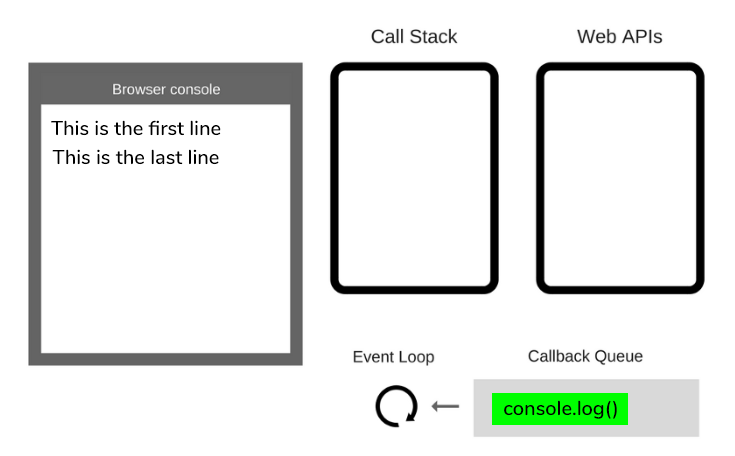


* Tuy nhiên hàm nay không trả về kết qua ngay mà phải đợi 1 giây. Hàm setTimeout() ở đây chính là một API mà **WEB APIs** cung cấp. Lập tức đoạn code này được chuyển vào trong **WEB APIs** và trình duyệt sẽ tạo ra một bộ hẹn giờ tương ứng với thời gian trên là 1 giây trước khi trả về kết quả

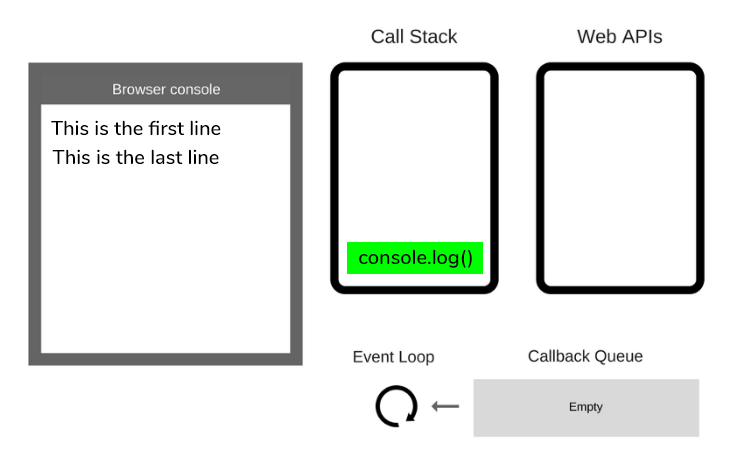


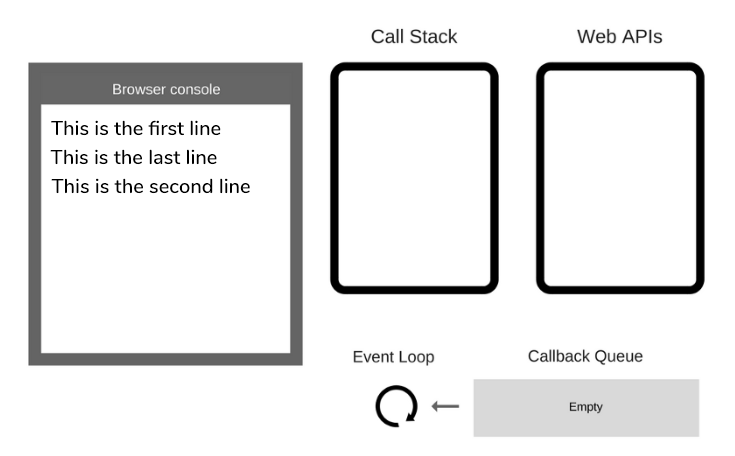
* Khi đoạn code thứ 2 được chuyển sang **WEB APIs** thì lập tức đoạn code cuối cùng console.log('This is the last line ') đã được đưa vào **CALL STACK** để thực
* hiện và trả về kết quả là dòng chữ This is the last line. Sau đó đoạn code này cũng được đẩy ra khỏi **CALL STACK**

* Sau khi bộ giờ trong **WEB APIs** chạy xong thì kết quả trả về lúc này không được in ngay ra màn hình mà nó được đẩy vào **CALLBACK QUEUE**



* **EVENT LOOP** với chức năng liên tục giám sát xem **CALL STACK** đã trống chưa và **CALLBACK QUEUE** có gì không. Lúc này **CALLBACK QUEUE** đang chư kết quả mà **WEB APIs** trả về nên và **CALL STACK** lúc này cũng đã trống do toàn bộ code trong chương trình đã được thực hiện nên nó sẽ đẩy kết quả trong **CALLBACK QUEUE** vào lại **CALL STACK** và đoạn code console.log(''This is the second line) được thực hiện và trả kết quả ra màn hình.



Đó là toàn bộ quá trình diễn ra trong cơ chế chạy bất đồng bộ của javascript, tuy nhiên ta cần chú ý một số điểm sau:

* **EVENT LOOP** chỉ tiến hành đẩy hàm trả về từ **CALLBACK QUEUE** sang **CALL STACK** khi **CALL STACK** đã thực hiện xong hết các đoạn code khác của chương trình
* Thứ tự các kết quả được trả về từ **WEB APIs** sang **CALLBACK QUEUE** sẽ không theo thứ tự đưa vào mà sẽ tùy thuộc hàm nào chạy xong trước sẽ được đẩy vào **CALLBACK QUEUE** trước đồng nghĩa với việc sẽ được chuyển qua **CALL STACK** trước.

### Call Back

Call Back là một hàm được truyền vào một hàm khác với tư cách như một tham số của hàm đó. Ví dụ như

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | function nauGa(callback)       nauNuocSoi();      vatLongGa();      callback();  }  function luocGa(){      //  }    function nuongGa(){  //  }  nauGa(luocGa);  nauGa(nuongGa); |

Ở đoạn mã trên, chúng ta thấy rằng, hàm luocGa và nuongGa được dùng như tham số trong hàm nauGa Với Javascript, một ngôn ngữ hướng sự kiện, call back được sử dụng rất nhiều khi xử lý các sự kiện, ví dụ như

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $('#button').click(function(){  alert("hê nô");  }) |

Chúng ta có thể áp dụng call back để đồng bộ hóa các đoạn mã không đồng bộ. Ví dụ như ở đoạn mã trên. Nấu nước sôi cần một khoảng thời gian chờ nước sôi, chúng ta không phải làm gì. Ta có thể biểu diễn thời gian chờ này bằng hàm setTimeout() trong javascript.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | function soCheGa(){     nauNuocSoi();     vatLongGa();  }  function nauNuocSoi(){     setTimeout(function(){       console.log("nau nuoc soi")     }, 1000)  }  function vatLongGa(){  console.log("vat long ga");  }  soCheGa(); |

Và nếu như theo đúng cách chạy của Javascript thì hành động vặt lông gà sẽ được thực hiện trước hành động nấu nước sôi. (à, quên mất, nấu nước sôi là để nhúng gà vào vặt lông chứ không phải để luộc gà đâu nhé ) .Mà nếu chúng ta vặt lông gà luôn mà không cần nhúng nước nóng thì tội cho bác gà quá. Vì thế để cho gà có thể ra đi thanh thản chúng ta cần đồng bộ hóa lại quy trình bằng callback như sau

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | function soCheGa(callback){        nauNuocSoi(vatLongGa);      }       function nauNuocSoi(callback){        setTimeout(function(){          console.log("nau nuoc soi");          callback();        }, 1000)      } |

Tuy nhiên, Callback cũng có nhược điểm. Đó là khi chúng ta muốn nhiều hành động bất đồng bộ thực hiện theo đúng thứ tự liên tiếp nhau, chúng ta phải gọi nhiều hàm callback lồng vào nhau nhiều lần, gây ra đoạn code rất khó kiểm soát và không tối ưu. Đây gọi là tình trạng Callback Hell. Ví dụ như muốn in các số từ 1 đến 10, mà mỗi hành động in đều là một hàm bất đồng bộ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | function printNumber(number, callback){    setTimeout(      () =&gt; {        console.log(number)        callback()      },      Math.floor(Math.random() \* 100) + 1    )  }    function printAll(){    printNumber(1, function(){      printNumber(2, function(){        printNumber(3, function(){          printNumber(4, function(){            printNumber(5, function(){              printNumber(6, function(){                printNumber(7, function(){                  printNumber(8, function(){                    printNumber(9, function(){                      printNumber(10, function(){                        })                    })                  })                })              })            })          })        })      })    })  } |

### Promise

Promise là một đối tượng bao hàm một hàm chứa các đoạn code không đồng bộ. Hàm này chứa 2 tham số là hai hàm callback để giải quyết sau khi mã đồng bộ thực hiện thành công hay thất bại. Promise cung cấp cho ta hai phương thức xử lý sau khi đoạn mã bất đồng bộ thực hiện thành công hoặc thất bại. Hàm then() dùng để xử lý sau khi mã bất đồng bộ được thực hiện thành công và hàm catch() dùng để xử lý sau khi mã bất đồng bộ thực hiện thất bại

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | function printNumber(number){      return new Promise((resolve, reject) =&gt; {        setTimeout(          () =&gt; {          if(number&lt;0){           reject();           }else{               console.log(number);               resolve();           }          },        1000        )      })  }  printNumber(1)  .then(()=&gt;printNumber(2))  .reject(()=&gt;console.log("number &lt; 0")) |

Phương thức then có thể thực thi một hàm, một Promise hay một đối tượng. Nếu chúng ta dùng then để trả về một Promise thì ta có thể tận dụng để xử lý tình trạng Callback Hell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | printNumber(1)  .then(()=&gt;printNumber(2))  .then(()=&gt;printNumber(3))  .then(()=&gt;printNumber(4))  .then(()=&gt;printNumber(5))  .then(()=&gt;printNumber(6))  .then(()=&gt;printNumber(7))  .then(()=&gt;printNumber(8))  .reject(()=&gt;console.log("number &lt; 0")) |

Tuy nhiên, dù Promise đã giải quyết được vấn đề Callback Hell, nhưng chúng ta có thể thấy, đoạn mã vẫn chưa thực sự rõ ràng và dễ hiểu. Trong phương thức chúng ta vẫn phải truyền vào một hàm, mà hàm đó trả về một hàm khác có giá trị trả về là 1 Promise. Chúng ta tạm gọi đây là tình trạng Promise Hell.

Asyn/Await

Async / Await là một tính năng ngôn ngữ là một phần của tiêu chuẩn ES8. Từ khóa Async để khai báo rằng hàm này sẽ xử lý các hàm bất đồng bộ, nó sẽ chờ kết quả của các hàm bất đồng bộ được trả về sau đó mới thực hiện tiếp. Hàm bất đồng bộ đó phải trả về một Promise và được khai báo với từ khóa Await

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | function printNumber(number){    return new Promise((resolve, reject) =&gt; {      setTimeout(        () =&gt; {         console.log(number)         resolve()        },       Math.floor(Math.random() \* 100) + 1      )    })  }    async function printAll(){    await printNumber(1);    await printNumber(2);    await printNumber(4);    await printNumber(5);    await printNumber(6);  }  printAll(); |

Đến đây thì chúng ta đã thấy Asyn/Await đã giải quyết triệt để được tình trạng Callback Hell cũng như Promise Hell.

-Với 1 hàm bất kỳ vd: async function say(){ return "hi"; }

Khi async đối tượng trả về luôn được chuyển thành promise (kể cả không return



Nếu muốn lấy kết quả của hàm async ta cần thêm await

### Promise.all

# jQuery

## Selection

<#id hoặc .class > <#id hoặc .class > …

.<class>[<tên thuộc tính =”<nội dung thuộc tính>”]

#<id>[<tên thuộc tính =”<nội dung thuộc tính>”]

## Đọc và đổi nội dung html

đổi nội dung HTML để tạo 1 hoặc nhiều thẻ mới tạo ra giao diện mới bê trong selector( selector vẫn còn

đổi nội dung text chỉ để thay đổi text bên trong selector

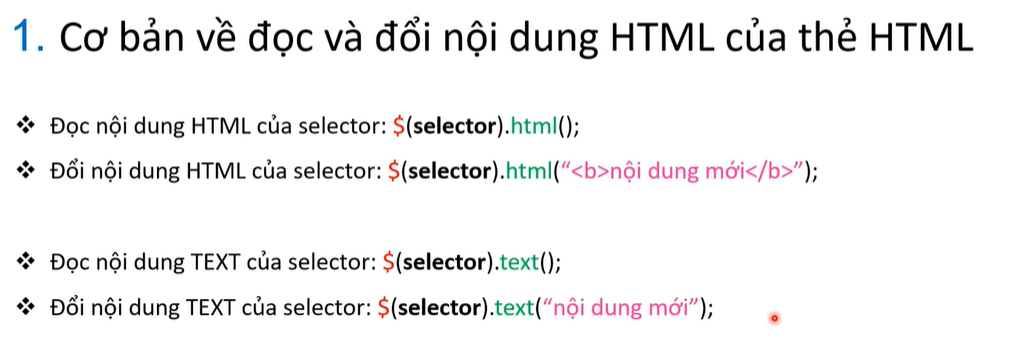
### VD Nội dung mới

let index = 5;

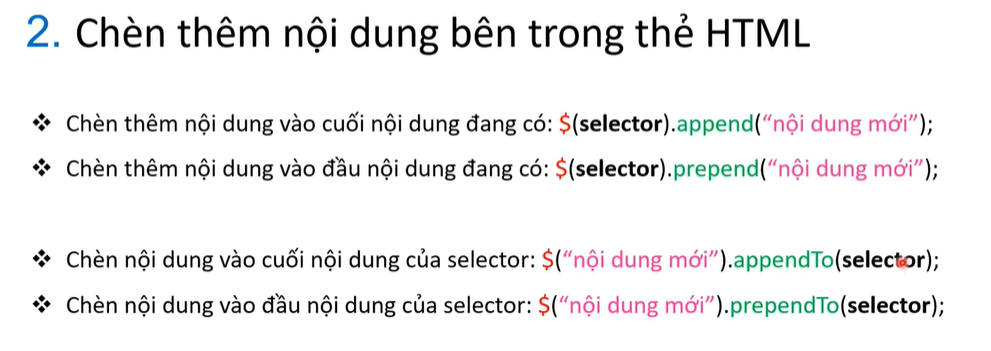
                for (; index <= 20; index += 1) {

                    $(".father").after("<Div>số:  +" + index + "</div>");

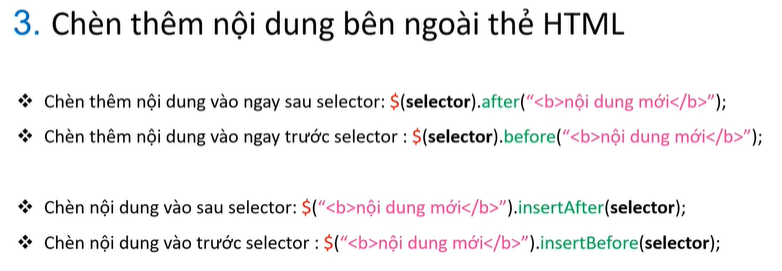
                }

nội dung mới sẽ là 1 string nhưng nếu muốn chuyền 1 tham số biến đổi như index ta có thể là như trên

## Chèn nội dung bên trong thẻ HTML



## Chèn nội dung bên ngoài thẻ HTML



## Xóa bỏ hoặc thay thế, làm rỗng nội dung và thẻ HTML

* làm rỗng nội dung selector: $(selector).empty();

toàn bộ nội dung bao gồm text và thẻ bên trong selector sẽ bị xóa, nhưng selector sẽ không bị xóa

* xóa bỏ selector khỏi tài liệu: $(selector).remove(); Hoặc $(selector).detach();

toàn bộ nội dung bên trong và thẻ sẽ bọ xóa

* Loại bỏ các phần tử cha của selector:$(selector).unwrap();

chỉ 1 có phần tử cha trực tiếpnó bị xóa. thẻ ông, thẻ cụ thì vẫn còn nguyên

* thay thế selector bằng nội dung mới: $(selector).replaceWith(“<b>nội dung mới</b>”);

hoặc: $(“<b>nội dung mới</b>”).replaceAll(selector);

nội dung mới ở đây phải là 1 hoặc nhiều thẻ mới thế vào vị trí selector bị xóa

## Get, Set thuộc tính thẻ

$(selector).attr(“thuộc tính”);

$(selector).attr(“thuộc tính”, “giá trị mới”);

$(selector).attr({ *attribute*:value, *attribute*:value, *attribute*:value, .... });

Vd: $("img").attr({ width:"500", height:"100" });

$(selector).prop(“thuộc tính”);

$(selector). prop(“thuộc tính”, “giá trị mới”);

**Hàm prop() trong jquery** cũng dùng để lấy các thuộc tính của thẻ, tuy nhiên cách trả về kết quả lại khác một xíu. Đối với những thuộc tính có dạng true/false thì thay vì trả về giá trị của thuộc tính như hàm attr() thì nó sẽ trả về true hoặc false.

vd

theImage.src = $("#pr\_draggable img").attr("src");

## Thay đổi CSS

### Thay thế class

Cách đơn giản nhất để chèn CSS class vào phần tử HTML đó là sử dụng method addClass() của jQuery. Có hai cách sử dụng method .addClass() của jQuery:

#### .addClass(className)

Với cách này chúng ta sẽ sử dụng đối số truyền vào là một chuỗi chứa một hoặc nhiều CSS class sẽ được thêm vào phần tử:

$('#my-el').addClass("new-class-1 new-class-2");

Kết thúc câu lệnh trên phần tử #my-el sẽ được gắn thêm hai CSS class là new-class-1 và new-class-2.

#### .addClass(function)

Với cách này chúng ta sẽ sử dụng đối số truyền vào là một hàm thay vì một chuỗi. Ví dụ như sau:

$("ul li").addClass(**function** (index) {

**return** "item-" + index;

});

Với cách sử dụng method .addClass() như trên thì jQuery sẽ lặp qua từng phần tử li có trong danh sách và index chính là thứ tự ứng với phần tử này trong danh sách. Kết quả trả về quả hàm sẽ xác định CSS class được thêm vào phần tử.

Kết thúc câu lệnh trên mỗi phần tử li sẽ được thêm lần lượt các CSS class bắt đầu từ phần tử đầu tiên (nếu có) sẽ là item-0, phần tử thứ hai (nếu có) là item-1, phần tử thứ 3 (nếu có) item-2...

#### .removeClass(*classname*);

 Phương thức **removeClass()** dùng để xóa *"tên lớp"* khỏi giá trị thuộc tính class của phần tử.

Vd: $("p").removeClass("background");

### Thay thế thuộc tính CSS

#### [**.css('**](https://hocwebchuan.com/reference/jquery/jquery_css.php#anchor01)*thuộc tính***','***giá trị***')**

vd: $('p.test').css('color','red');

#### [**.css({'**](https://hocwebchuan.com/reference/jquery/jquery_css.php#anchor02)*thuộc tính 01***':'***giá trị 01***','***thuộc tính 02***':'***giá trị 02***',...})**

vd: $('p.test').css({'color':'blue','border':'1px dashed red'});

#### [**.css('**](https://hocwebchuan.com/reference/jquery/jquery_css.php#anchor03)*thuộc tính***',function(index,***giá trị***){})**

vd: $('p.test').css('width', function(index,value){

return parseFloat( value ) \* 2;

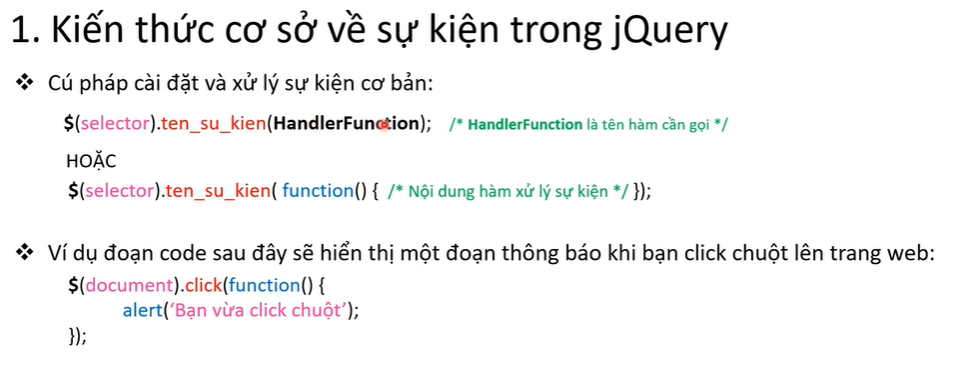
});

### Lấy giá trị *style* của thành phần:

#### [**.css('**](https://hocwebchuan.com/reference/jquery/jquery_css.php#anchor04)*thuộc tính***'**)

vd: var ulHeight = $('ul').css('height');

## Điều khiển sự kiện cơ bản



Hoặc [**.on('**](https://hocwebchuan.com/reference/jquery/jquery_on.php#anchor01)***Sự kiện*', function(){...})**

**// sự kiện: tên sự kiện trong jquenry(không có on)**

### .trigger (“event name”)

* Kích hoạt sự kiện

Vd: khi nhấn botton sẽ kích hoạt sự kiện chọn file

<script>

$(document).ready(function(){

$("input").select(function(){

$("input").after(" Text marked!");

});

$("button").click(function(){

$("input").trigger("click");

});

});

</script>

</head>

<body>

<input type="file" value="Hello World"><br><br>

<button>Trigger the select event for the input field</button>

</body>

### .error(HandlerFunction);

sự kiện: đã có lỗi xảy ra với selector

vd: $(window).resive(

### .resize(HandlerFunction);

sự kiện: selector đã bị thay đổi kích thước

vd: $(window).resive(ResiveHappend);

cửa sổ trình duyệt bị thay đổi thì sẽ gọi đến hàm ResiveHappend()

### .scroll(HandlerFunction);

sự kiện:đá có thao tác cuộn trên selector

áp dụng với những selector có thanh cuộn

### .ready(HandlerFunction);

sự kiện : selector đá tải xong nội dung và sẵn sàng

Thường dùng với toàn bộ trang HTML: nếu không muốn để link scrip xuống dưới cuối cùng trong HTML

vd: $(document).ready(function(){ //code});

### .load(HandlerFunction);

sự kiện: đã tải xong nội dung

### .unload(HandlerFunction);

sự kiện trang web bị tắt hoặc chuyển hướng đi web khác

### .holdReady(true/false);

đây là 1 hàm jquery giúp chúng ta có thể tạm giữ cho sự kiện ready khong xảy ra(khi cần thiết). Giá trị TRUE sẹ thực hiện việc chờ(tạm giữ) và giá trị (false) sẽ xóa bỏ sự tạm giữ

### .click(HandlerFunction);

sự kiện: click chuột

### .contextmenu(HandlerFunction);

sự kiện: click chuột phải

### .bdclick(HandlerFunction);

sự kiện Double click

### .hover(HandlerFunction);

sự kiện rê chuột lên đối tương

### .mousedown(HandlerFunction);

sự kiện: click nhưng chưa nhấc chuột lên

### .mouseup(HandlerFunction);

sự kiện: click đã nhấc chuột lên

### .mouseover(HandlerFunction);

sự kiện: rê chuột lên đối tượng

### .mouseout(HandlerFunction);

sự kiện: chột rời khỏi đối tượng đối tượng

### .mouseenter(HandlerFunction);

sự kiện: rê chuột vào đối tượng

### .mouseleave(HandlerFunction);

sự kiện: chột rời khỏi đối tượng đối tượng

### .mousemove(HandlerFunction);

sự kiện: chuột di chuyển trên đối tượng

### .toggetle(HandlerFunction1, HandlerFunction2, …);

Trường hợp đặc biệt: toggle giúp bạn điều khiển sự kiện click chuột theo số lần và lặp lại tuần tự

* VD: có 2 handler: click 1 lần gọi handler1, clịck 2 lần gọi hanler 2, click 3 lần gọi handler3, click lần 4 trở về handler1

### .keypress(HandlerFunction);

sự kiện: gõ phím lên seletor

### .key(HandlerFunction);

sự kiện: thao tác đè phím xuống

vd tính năng: ấn giữ phím a để xem thông tin

### .keyup(HandlerFunction);

sự kiện: thao tác nhả phím sau khi đè xuống

### .focus(HandlerFunction);

sự kiện: đang target vào selector( sẵn sàng) tương tự [onfocus](#_onfocus)

cụ thể: khi bạn click vào 1 thẻ input có type = [“text”](#_text) con trỏ chuyển thành dấu nháy văn bản. vừa xảy ra sự kiện click vừa xảy ra sự kiện focus

### .blur(HandlerFunction);

sự kiện: hủy taget selector(sau khi forcus) tương tự [onblur](#_onblur)

Cụ thể: khi bạn nhập xong dữ liệu bạn click ra 1 ngoài sự kiện blur sẽ được gọi>

>> có thể dùng để kiểm tra điều kiện của dữ liệu, thông báo hợp lệ

### .change(HandlerFunction);

sự kiện: thay đổi giá trị của selector( chỉ sau khi blur) tương tự [onchange](#_onchange)

### .select(HandlerFunction);

sự kiện: tô đen một đoạn văn bản trong selector

### .submit(HandlerFunction);

sự kiện: bắt đầu việc submit form

### .blind- đính kèm sự kiện

được sử dụng để đính kèm xử lý thêm sự kiện cho thành phần. tương tự [.on](#_Điều_khiển_sự)

các sự kiện trong đây có thể vừa dùng “JavascriptEventName”(có on+<tên sự kiện>) hoặc sự kiện trong jquenry(không on)

#### [**.bind(**](https://hocwebchuan.com/reference/jquery/jquery_bind.php#anchor01)JavascriptEventName**,function(){})**

#### [**.bind('**](https://hocwebchuan.com/reference/jquery/jquery_bind.php#anchor03)*JavascriptEventName(1) JavascriptEventName(2)***',function(){})**

vd: $(function(){

$('div.test').bind('onclick ondblclick',function(){

$('span').text('Đã click');

});

});

#### [**.bind({**](https://hocwebchuan.com/reference/jquery/jquery_bind.php#anchor04)*Sựkiện1***:function(){},***Sựkiện2***:function(){}})**

vd: $(function(){

$('div.test').bind({

click : function(){

$('span').text('Click');

},dblclick : function(){

$('span').text('Double click');

}

});

});

## .offset()

Lấy tọa độ hiện tại của thành phần đầu tiên, hoặc thiết lập các tọa độ của từng thành phần trong tập hợp các thành phần phù hợp liên quan đến văn bản.

* tọa độ được tính từ góc màn hình, không phụ thuộc vào position

có 2 thuộc tính của offset

.offset().top;

.offset().left;

set tọa độ:

### .offset({ top: 10, left: 20 });

set tọa độ: với top =10; left =20;

tọa độ này tính từ góc màn hình chứ không giống {left; right} của position

### .offset(function(index,tọa độ){...})

set tọa độ theo ý muốn

* **function phải có return**

vd: mỗi lần click thì tăng left và top thêm 10 px

$(this).offset(function(index, coordinates){

if (index == 0) {//Chỉ tác động lên thành phần đầu tiên

coordinates.top += 10;

coordinates.left += 10;

}

return $(this).offset(coordinates);

});

## .val()

-lấy dữ liệu val của thẻ. luôn trả về dạng string.

## .data-...

Không nên đặt tên là in háo vì chungd sẽ trở về in thường vd:

<button type="button" class="btn btn-default edit" data-IdGVDT="${item.IdGVDT}”>

thực chất thuộc tính data của thẻ trên là : data-idgvdt

## delegate()

Phương thức delegate() không được dùng nữa trong phiên bản 3.0. Sử dụng phương thức [on ()](https://www.w3schools.com/jquery/event_on.asp) để thay thế.

Vd:

Khi một phần tử <p> bên trong một phần tử <div> được nhấp, hãy thay đổi màu nền của tất cả các phần tử <p>:

$("div").delegate("p", "click", function(){  
    $("p").css("background-color", "pink");  
});

Mặc dù vậy nó đặc biệt hữu ích khi bắt sự kiện từ 1 thẻ được render từ lệnh [.append()](#_.append()) hoặc những lệnh thêm thẻ tương tự. vì trong trường hợp này những cách gọi khác không thể sử dụng được

→ Như cụ thể: thẻ p đc tạo ra như sau: $("div").append(<p>Hieu</P>)

## .prev()

.prev(): xác định thành phần cùng cấp ngay trước thành phần sử dụng .prev().

VD:

!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<title>Tiêu đề</title>

<script src="https://code.jquery.com/jquery-latest.js"></script>

<script>

$(function(){

$('li.second').prev().css('background-color','yellow');

});

</script>

</head>

<body>

<ul>

<li>li thứ nhất</li>

<li class="second">li thứ hai</li>

<li>li thứ ba</li>

<li>li thứ tư</li>

<li>li thứ năm</li>

</ul>

</body>

</html>

Hiển thị trình duyệt:

* li thứ nhất
* li thứ hai
* li thứ ba
* li thứ tư
* li thứ năm

## .next()

xác định thành phần cùng cấp ngay kế tiếp thành phần sử dụng . ngược với prev

.shoping\_\_cart\_\_tota

## .before()

Chèn nội dung, di chuyển thành phần vào ngay trước mỗi thành phần đã có.

vd: $('div').before('<p>nội dung thêm vào</p>');

## .after()

[**.after(**](https://hocwebchuan.com/reference/jquery/jquery_after.php#anchor01)***nội dung*)**

Chèn nội dung, di chuyển thành phần vào ngay sau mỗi thành phần đã có.

$('div').after('<p>nội dung thêm vào</p>');

## .find()

.find(): Giúp tìm thành phần trong thành phần cha.

[**.find(**](https://hocwebchuan.com/reference/jquery/jquery_find.php#anchor01)***bộ chọn*)**

$('li').find('.test');

* Đã được thêm vào từ phiên bản 1.6

[**.find(**](https://hocwebchuan.com/reference/jquery/jquery_find.php#anchor02)***jquery object*)**

$('li').find($('div.test')); tìm thành phần con là thẻ div có class=“test”

## .append()

thêm nội dung vào vị trí cuối cùng bên trong phần nội dung của phần tử được chọn.

<body>

<button type="button">Click here</button>

<ul>

<li>Tài liệu học HTML</li>

<li>Tài liệu học CSS</li>

~~<li>Tài liệu học CSS</li>~~ // nội dung sẽ được thêm

</ul>

<script>

$(document).ready(function(){

$("button").click(function(){

$("ul").append("<li>Tài liệu học JavaScript</li>");

});

});

</script>

</body>

## .prepend()

thêm nội dung vào vị trí đầu tiên bên trong phần nội dung của phần tử được chọn. tương tự [.append()](#_.append())

## .parent()

Lấy ra selecter của cha nó

## .hasClass()

hasClass(‘tên class’): trả về kiểu bool. Kiểm tra xem selector có class đó không

## [.first()](https://api.jquery.com/first/#first)

Chọn ra phần tử đầu tiên

! var countItem = $('.countItem').text(); trường hợp nếu không có first(). Có 2 thẻ mỗi thẻ có text là 5 => countItem= 55; 3 thẻ thì sẽ là 555

## [.last()](https://api.jquery.com/first/#first)

Chọn ra phần tử cuối cùng

## .map()

* Cơ bản giống each() dùng để lặp qua các phần tử trong DOM
* Tốt hơn về hiệu suất và code ngắn hơn.

VD:

$('p').append($('input').map(function(){

return this.id;

}).get().join(", "));

<body>

<p>Value: </p>

<input type="text" id="text01" />

<input type="text" id="text02" />

<input type="text" id="text03" />

<input type="text" id="text04" />

</body>

Hiển thị:

Value: text01, text02, text03, text04

Map thường dùng kết hợp với get() với join(“, “)-các phần tử cách nhau bởi dấu ‘,”

## .preventDefault()

* Ngăn nút submit gửi biểu mẫu
* Ngăn một liên kết theo dõi URL

## .stopPropagation ()

dừng việc tạo sự kiện cho các phần tử mẹ, ngăn không cho bất kỳ trình xử lý sự kiện mẹ nào được thực thi.

Vd:

<script>

$(document).ready(function(){

$("span").click(function(event){

event.stopPropagation();

alert("The span element was clicked.");

});

$("p").click(function(event){

alert("The p element was clicked.");

});

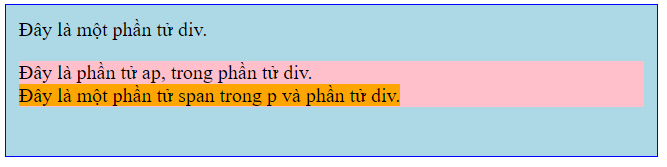
$("div").click(function(){

alert("The div element was clicked.");

});

});

</script>



**Lưu ý:** Nhấp vào từng phần tử ở trên. Khi nhấp vào phần tử **div** , nó sẽ cảnh báo rằng phần tử div đã được nhấp. Khi nhấp vào phần tử **p** , nó sẽ trả về cả phần tử p và phần tử div, vì phần tử p nằm bên trong phần tử div. Nhưng khi nhấp vào phần tử **span** , nó sẽ chỉ trả về chính nó chứ không phải p và phần tử div (mặc dù nó nằm bên trong các phần tử này). Event.stopPropagation () ngăn sự kiện nhấp chuột chuyển sang các phần tử mẹ.

## Hàm trong hàm

**1 Vd với preview image**

1 $('.addPhoto').click(function () {

2 var name = $('.modal img').attr('name');

3 var selectedImg = '.content img[name="' + name + '"]';

4 var parent = $(selectedImg).parent().parent();

5 var newFile = parent.find('input:file');

6 newFile.trigger('click');

7 newFile.change(function () {

8 if (this.files.length > 0) {

9 parent.find('input:checkbox').attr('checked', 'checked');

10 var reader = new FileReader();

11 reader.addEventListener('load', function () {

12 $(selectedImg).attr('src', this.result);

13 });

14 reader.readAsDataURL(this.files[0])

15 }

16 });

17 });

khác biệt nằm ở dòng 9; khi thao tác trực tiếp với thẻ input file *($('.addPhoto').click không được gọi* ) dòng 9 lúc này sẽ không được gọi vì nó không hiểu parent là ai

//

Luôn đúng: newFile.change có thể bắt sự kiện sau khi thao tác trực tiếp với thẻ input file hoặc trong khi trong $('.addPhoto').click(function () đang chạy

khác biệt khi dùng hàm trong hàm là sự kiện con sẽ hoạt động sau khi chạy hết chương trình(ngoại trừ các hàm sự kiện đồng cấp phía sau sẽ chạy sau)

như vd trên: với dòng 6: input file chỉ update khi hoàn thành *($('.addPhoto').click* Nếu không có hàm ở dòng 7. Thì …files.length luôn bằng 0 vì input file chưa update.

//

- việc dùng hàm trong hàm cho phép sử dụng lại các biến của hàm cha với điền kiện hàm con được gọi trong khi duyệt hàm cha. Hàm con sẽ được đánh dấu để duyệt sau khi chạy hết hàm cha.( nhưng vẫn giữ lại biến) . như vd trên: dòng 14 sẽ được duyệt trước hàm reader.addEventListener()

Lưu ý. Hàm con thường đc gọi với những sự kiện bị động như, load, chage, ..

## Animation

### show – hiệu ứng hiển thị

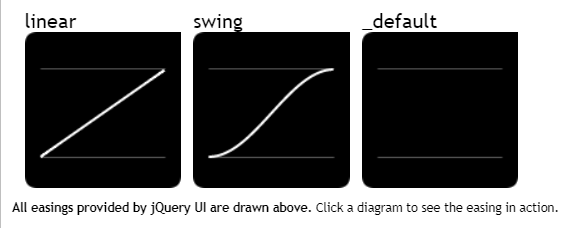
### các tham số bên trong mỗi hàm

mỗi tham số đều không bắt buộc

- độ bền là thời gian thực hiện animation

- function sẽ được gọi khi animation hoàn tất

- easing có thể sử dụng swing hoặc linear( <https://api.jqueryui.com/easings/>)



* "swing" - chậm ở đoạn đầu & cuối, nhanh ở đoạn giữa.(có gia tốc)
* "linear" - tốc độ không đổi.

[**.hide(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

### hire – Hiệu ứng mất

[**.hide(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

### toggle- hiệu ứng hiển thị hoặc biến mất

[**.toggle(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

*Lưu ý:*

* Một phần tử bị ẩn bởi phương thức toggle() thì cũng tương tự như việc phần tử đó được thiết lập thuộc tính display với giá trị none trong CSS.
* Phương thức toggle() chỉ có thể hiển thị những phần tử bị ẩn bởi thuộc tính display:none chứ không có tác dụng với phần tử bị ẩn bởi thuộc tính visibility:hidden trong CSS.

### slideDown- hiển thị với hiệu ứng trượt một phần tử đang ẩn

[**.slideDown(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

- *Lưu ý:* Phương thức **slideDown()** chỉ có tác dụng với phần tử bị ẩn bởi thuộc tính display:none chứ không có tác dụng với phần tử bị ẩn bởi thuộc tính visibility:hidden trong CSS.

### slideUp- ẩn với hiệu ứng trượt

[**.slideUp(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

- *Lưu ý:* Một phần tử bị ẩn bởi phương thức **slideUp()** thì cũng tương tự như việc phần tử đó được thiết lập thuộc tính display với giá trị none trong CSS.

### slideToggle- ẩn/hiện với hiệu ứng trượt

[**.slideToggle(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

- Phương thức **slideToggle()** dùng để thực hiện luân phiên giữa việc hiện & ẩn một phần tử.

- *Lưu ý:*

* Một phần tử bị ẩn bởi phương thức slideToggle() thì cũng tương tự như việc phần tử đó được thiết lập thuộc tính display với giá trị none trong CSS.
* Phương thức slideToggle() chỉ có thể hiển thị những phần tử bị ẩn bởi thuộc tính display:none chứ không có tác dụng với phần tử bị ẩn bởi thuộc tính visibility:hidden trong CSS.

### fadeln – tăng dần opacity đến 1

[**.fadeIn(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

*Lưu ý:* Phương thức **fadeIn()** chỉ có tác dụng với phần tử bị ẩn bởi thuộc tính display:none chứ không có tác dụng với phần tử bị ẩn bởi thuộc tính visibility:hidden trong CSS.

### fadeOut- giảm dần opacity đến 0

[**.fadeOut(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

khi opacity về 0, selector sẽ có thuộc tính display: node, các thẻ dưới sẽ bị đẩy lên thế chỗ selector đó

- *Lưu ý:* Một phần tử bị ẩn bởi phương thức **fadeOut()** thì cũng tương tự như việc phần tử đó được thiết lập thuộc tính display với giá trị none trong CSS.

### fadeToggle -  luân phiên giữa việc hiện & ẩn một phần tử.

[**.fadeToggle(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,'*easing*',function(){...})**

khi opacity về 0, selector sẽ có thuộc tính display: node, các thẻ dưới sẽ bị đẩy lên thế chỗ selector đó

- Lưu ý:

Một phần tử bị ẩn bởi phương thức fadeToggle() thì cũng tương tự như việc phần tử đó được thiết lập thuộc tính display với giá trị none trong CSS.

Phương thức fadeToggle() chỉ có thể hiển thị những phần tử bị ẩn bởi thuộc tính display:none chứ không có tác dụng với phần tử bị ẩn bởi thuộc tính visibility:hidden trong CSS.

[**.fadeToggle(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,x,'*easing*',function(){...})**

### fadeTo – opacity dần đến x thuộc[0;1]

[**.fadeToggle(**](https://hocwebchuan.com/reference/jquery/jquery_hide.php#anchor04)*độ bền***,x,'*easing*',function(){...})**

có thể đậm lên hoặc nhạt đi

- *Lưu ý:* Phương thức **fadeTo()** chỉ làm cho phần tử bị phai màu chứ không khiến phần tử bị ẩn đi.

### animation

#### animate()

$(selector).animate({style}, độ bền,'easing',function(){...});

ref: <http://webcoban.vn/jquery/phuong-thuc-animate-trong-jquery-cach-tao-hieu-ung-chuyen-dong-cho-phan-tu-trong-jquery.html>

{stype} : <! bắt buộc>

- Chỉ định cặp thuộc tính và giá trị được thay đổi.

*(giá trị phải được đặt bên trong cặp dấu nháy đơn hoặc nháy kép)*

- *Ví dụ:* Nếu bạn muốn thay đổi giá trị thuộc tính width của phần tử thành 500px thì bạn phải viết là *width:"500px"*

- *Lưu ý:* Nếu bạn muốn chỉ định nhiều cặp thuộc tính và giá trị được thay đổi thì bạn phải đặt một dấu phẩy ở giữa mỗi hai cặp.

- *Lưu ý:* Tên thuộc tính sử dụng trong phương thức animate() tương đối khác so với tên thuộc tính trong CSS. Tên thuộc tính trong phương thức animate() phải được viết dưới dạng camel-cased *(tức là ký tự đầu tiên của từ thứ hai trở đi phải viết hoa)*

- *Lưu ý:* Phương thức animate() không hỗ trợ những thuộc tính mà giá trị của nó xác định bởi tên gọi, điển hình như: background-color, color, font-family, ....

VD:

$("button").click(function(){

$("p").animate(

{width:"300px",height:"300px"},

3500,

"linear",

function(){

alert("KẾT THÚC HIỆU ỨNG CHUYỂN ĐỘNG");

}

);

});

VD: có thể thêm vòng lặp để chạy liên tục, Lưu ý: nhưng để vô hạn sẽ gặp lỗi

$("button").click(function(){

for(let index =1; index<=5; index++){

$(".child").animate({left:"400px"},2000);

$(".child").animate({top:"200px"},2000);

$(".child").animate({left:"0px"},2000);

$(".child").animate({top:"0px"},2000);

}

});

VD: và đâu là 1 vòng lặp vô hạn

$(document).ready(function(){

colors = ['#FFB30C', '#58EC00', '#0087EC', '#EEEEEE', '#FF5A00' ]

var i = 0;

animate\_loop = function() {

$('body').animate({backgroundColor:colors[(i++)%colors.length]

}, 500, function(){

animate\_loop();

});

}

animate\_loop();

});

#### danh sách các thuộc tính được hỗ trợ trong phương thức animate():

* [backgroundPositionX](http://webcoban.vn/jquery/demo?file=2737)
* [backgroundPositionY](http://webcoban.vn/jquery/demo?file=2738)
* [borderWidth](http://webcoban.vn/jquery/demo?file=2739)
* [borderBottomWidth](http://webcoban.vn/jquery/demo?file=2740)
* [borderLeftWidth](http://webcoban.vn/jquery/demo?file=2741)
* [borderRightWidth](http://webcoban.vn/jquery/demo?file=2742)
* [borderTopWidth](http://webcoban.vn/jquery/demo?file=2743)
* [borderSpacing](http://webcoban.vn/jquery/demo?file=2744)
* [margin](http://webcoban.vn/jquery/demo?file=2745)
* [marginBottom](http://webcoban.vn/jquery/demo?file=2746)
* [marginLeft](http://webcoban.vn/jquery/demo?file=2747)
* [marginRight](http://webcoban.vn/jquery/demo?file=2748)
* [marginTop](http://webcoban.vn/jquery/demo?file=2749)
* [outlineWidth](http://webcoban.vn/jquery/demo?file=2750)
* [padding](http://webcoban.vn/jquery/demo?file=2751)
* [paddingBottom](http://webcoban.vn/jquery/demo?file=2752)
* [paddingLeft](http://webcoban.vn/jquery/demo?file=2753)
* [paddingRight](http://webcoban.vn/jquery/demo?file=2754)
* [paddingTop](http://webcoban.vn/jquery/demo?file=2755)
* [height](http://webcoban.vn/jquery/demo?file=2756)
* [width](http://webcoban.vn/jquery/demo?file=2757)
* [maxHeight](http://webcoban.vn/jquery/demo?file=2758)
* [maxWidth](http://webcoban.vn/jquery/demo?file=2759)
* [minHeight](http://webcoban.vn/jquery/demo?file=2760)
* [minWidth](http://webcoban.vn/jquery/demo?file=2761)
* [fontSize](http://webcoban.vn/jquery/demo?file=2762)
* [bottom](http://webcoban.vn/jquery/demo?file=2763)
* [left](http://webcoban.vn/jquery/demo?file=2764)
* [right](http://webcoban.vn/jquery/demo?file=2765)
* [top](http://webcoban.vn/jquery/demo?file=2766)
* [letterSpacing](http://webcoban.vn/jquery/demo?file=2767)
* [wordSpacing](http://webcoban.vn/jquery/demo?file=2768)
* [lineHeight](http://webcoban.vn/jquery/demo?file=2769)
* [textIndent](http://webcoban.vn/jquery/demo?file=2770)

#### stop()

dừng animation

.stop(*stopAll*,*goToEnd*);

|  |  |  |  |
| --- | --- | --- | --- |
| topAll |  | Không bắt buộc | - Tham số stopAll xác định việc nếu phần tử được thiết lập nhiều hiệu ứng chuyển động thì phương thức stop() sẽ chỉ dừng hiệu ứng chuyển động hiện tại hay là dừng hiệu ứng chuyển động hiện tại lẫn tất cả những hiệu ứng chuyển động còn lại.  - Tham số stopAll có thể được xác định bởi một trong hai giá trị:   * *true:* dừng hiệu ứng chuyển động hiện tại lẫn tất cả những hiệu ứng chuyển động còn lại. * *false:* chỉ dừng hiệu ứng chuyển động hiện tại.   - *Lưu ý:* Mặc định thì phương thức stop() chỉ dừng hiệu ứng chuyển động hiện tại, các hiệu ứng chuyển động còn lại vẫn được tiếp tục. |
| goToEnd |  | Không bắt buộc | - Tham số goToEnd xác định việc khi hiệu ứng chuyển động hiện tại bị dừng lại thì giá trị thuộc tính của hiệu ứng chuyển động đó chỉ bằng ở điểm dừng lại hay được chuyển đến trạng thái cuối cùng.  - Tham số goToEnd có thể được xác định bởi một trong hai giá trị:   * *true:* giá trị thuộc tính được chuyển đến trạng thái cuối cùng. * *false:* giá trị thuộc tính chỉ bằng ở điểm dừng lại.   - *Lưu ý:* Mặc định, khi hiệu ứng chuyển động hiện tại bị dừng lại thì giá trị thuộc tính của hiệu ứng chuyển động đó chỉ bằng điểm dừng lại. |

vd:

<script>

$(document).ready(function(){

$("#start").click(function(){

$(".child").animate({left:"400px"},3000);

$(".child").animate({top:"200px"},3000);

$(".child").animate({left:"0px"},3000);

$(".child").animate({top:"0px"},3000);

});

$("#stop").click(function(){

$(".child").stop(false,true);

});

});

</script>

# jQueryUI

url: <https://api.jqueryui.com/>

## option

$(selector).”ten widget/ interaction” ("option", "thuộc tính của option")

**vd**

$(selector).progressbar("option", "max")

$(selector).progressbar("option", "max", 1024) – set giá trị mới

## Methods

$(selector).”ten widget/ interaction” ("thuộc tính của methods")

**vd:**

$("#my-prb").progressbar("value")

## Events

tạo function trong lúc khởi tạo

## Tiện ích (widget)

### dialog

giống như 1 messageBox trong C#

tương tự như 1 alert

cơ chế display: none <->block

### datepicker

### progressbar

Mình phải gắn vào 1 thẻ div

với max là giá trị lớn nhất

value tương ứng với độ chạy của thanh, 0<value<max

## Tương tác (interaction)

### draggable

Cho phép các selector được di chuyển bằng chuột.

### dropable

### kết hợp với draggable

* gắn vào 1 thẻ div
* với accept là 1 selector . Thường là selector của 1 draggble cho phép chấp nhận thả lên nó
* áp dụng với việc thêm vào rỏ trong bán hàng

### resizable

### selectable

### sortable

* gắn vào 1 thẻ <ul>, các <li> ở trong sẽ tự động được gắn

Cho phép sắp xếp lại các li trong ul

* xắp xếp lại ảnh,

#### option-connectWith: “selector”

selector: selector của 1 ul khác

cho phép liên kết với các ul khác nhau, có thể chuyển đổi <li> từ <ul> gốc sang <ul> trong connectWith

lưu ý: <ul> trong connectWith cũng phải là sortable

<li> sẽ thay đổi css theo những <li> bên <ul> connectWith

## Hiệu ứng (effects)

### easing

### color animation

# Cơ chế chèn icon

html: <botton> <span>thẻ span để icon</span> </botton>

background-image: url(..);

position: absolute;

overflow: hidden; background-repeat: no-repeat;

Nếu image là 1 ảnh to gồm nhiều icon trong đó ó icon ta cần lấy ta sẽ dùng cơ chế hiển thị 1 phần của image

background-position: -…px -…px;

background-position

kích thước icon là : height: …; width: …;

botton

icon

icon

icon

icon

icon

icon

# This.

## disabled\*

## form\*

## formAction\*

## formMethod\*

## formNoValidate\*

## formTarget\*

## name

get, set name của nó

## type

get, set type của nó

## value

get, set type của nó dùng với thẻ input

## willValidate\*

## validity\*

## validationMessage\*

## labels\*

## title

get, set title của nó

## lang\*

## translate\*

## dir\*

## hidden\*

## accessKey\*

## draggable\*

## spellcheck\*

## autocapitalize\*

## contentEditable\*

## isContentEditable: \*

## inputMode\*

## parentNode:

* 1 biến kiểu offsetParent cũng là 1 this
* Nó cho biết mọi thông tin về thẻ cha của nó
* cha nó là thẻ bao ngoài của nó

vd:  
<div id="parent\_bt" name="parent-offer" title="parent" style="position:relative;">

        <button type="button" style="position: relative;">Click here</button>

    </div>

với

 $("button").click(function() {

        var obj = $(".display\_background .display\_img").offset();

        $("#s1").text(this.offsetTop + "px");

        var str = $(this).offsetParent().attr("id");

        $("#s2").text(str);

    });

Có thể .attr(“title”); .attr(“name”); .attr(“value”, 5);

hoặc .offsetTop;

hoặc . offsetParent(); tiếp để lấy thông tin ông nội nó

## offsetParent

offsetParent là phần tử cha gần nhất có positive: relative or absolute hoặc phần nội dung của trang. Còn [parentNode](#_parentNode:): là cha trực tiếp

## offsetTop

get,set tọa độ top của nó theo position, # với [offset();](#_.offer())

## offsetLeft

get,set độ left của nó theo position, # với [offset();](#_.offer())

## offsetWidth

get,set width của nó

## offsetHeight

get,set height của nó

## style\*

## innerText

get,set nội dung bên trong thẻ

*giống js dom content*

## outerText

get, set nội dung bao gồm cả thẻ

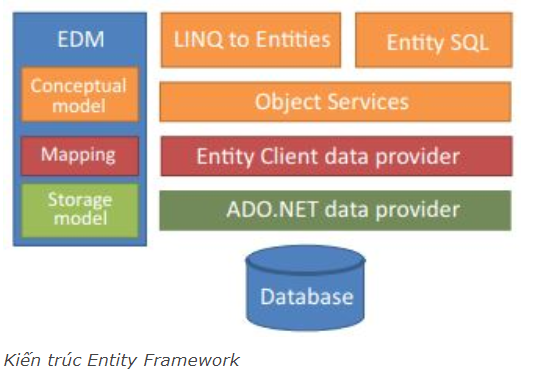
*giống js dom content*

# BACK-END (.Net MVC)

## ENTITY FRAMEWORK(EF)

* entity framework là gì?:

Tài liệu tham khảo: <https://tuhocict.com/gioi-thieu-tong-quan-kien-truc-cai-dat-entity-framework/>



* đây là 1 bộ chuyển đổi dữ liệu từ db sang model với việc

chuyển dữ liệu từ dạng bảng sang dữ liệu dạng object

* tách biệt model với csld và việc truy xuất dữ liệu hoàn toàn không dùng tới câu lệnh sql

**! nên dùng**

**Kiến trúc Entity Framework (Entity Data Model hay EDM)**

**for** once floor (yellow, red, green) match with one file xml

* Conceptual model (.csdl): be responsible save information about model classes and relation of them
* Storage model (.ssdl) : save information about database, include tables, view, stored proceduce, key, relation
* Mapping model (.msl) ; save ánh xạ information between conceptual and storage model

**LINQ to Entities và Entity SQL**

* đây là 2 ngôn ngữ dùng để truy xuất dữ liệu thay vì cài câu lệnh dạng string như trong ADO.NET

**Object services**

* chuyển đổi dữ liệu từ tầng dưới (Entity client data provider) về dạng object, dữ liệu nhận được từ tầng dưới không phải là dữ liệu dạng bảng lấy trực tiếp từ csdl mà là 1 dạng trung gian có tên model tabular structure.
* - đảm nhiệm chức năng **quản lý trạng thái:** hiểu đơn giản, đây là việc theo dõi thay đổi của từng object >> có vai trò especially important trong khi thực hiện các thao tác thêm, sửa, xóa dữ liệu. Nhờ có nó bạn có thể thoải mái thực hiện các thao tác trên dữ liệu và lưu lại tất cả thay đổi chỉ với 1 dòng code

**Entity client data provider:**

* chuyển đổi truy vấn LINQ hoặc ADO.NET về truy vấn SQL
* chuyển đổi kiết quả thu được từ truy vấn SQL về dạng trung gian (model tabular structure). kết quả trung gian được chuyển lên Object Services sử lý

**ADO.NET**

* Làm việc trực tiếp với csdl

## <connectionString>

VD: <connectionStrings>

<add name="DataFirstExampleContext" connectionString="data source=DESKTOP-L2JEIFM\SQLEXPRESS;initial catalog=DataFirstExample;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework" providerName="System.Data.SqlClient"/>

Đây là 1 quy ước máy tính với 1 name sẽ thay thế cho 1 connectionString.

tương ứng với 1 models tổng là VD:

public partial class DataFirstExampleContext : DbContext

{

public DataFirstExampleContext()

: base("name=DataFirstExampleContext")

{

}

public virtual DbSet<CategoryOfProduct> CategoryOfProducts { get; set; }

public virtual DbSet<Product> Products { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

}

string đượng truyền vào từ hàm khởi tạo chính là đại diện cho chuỗi kết nối connectionString

## LINQ to ENTITY FRAMEWORK

được hiểu như các method

! chuyển đổi toán tử

|  |  |
| --- | --- |
| sql | entity |
| and  =  or | &&  ==  || |

### OrderBy and OrderByDescending

db.CategoryOfProducts.OrderBy(c => c.Description)

OR: db.CategoryOfProducts.ToList().OrderBy(c => c.Description) (similar)

Nếu cần sắp xếp theo nhiều tiêu chí, bạn có thể ghép nối OrderBy (OrderByDescending) với ThenBy (ThenByDescending) như sau:

var query = db.Courses .OrderBy(c => c.Name) .ThenBy(c=>c.Credit) .ThenByDescending(c=>c.Description);

Trong truy vấn trên, bạn sắp xếp (tăng dần) theo tên, sau đó tới số tín chỉ, và cuối cùng (giảm dần) theo mô tả.

OrderByDescending: đây là giảm dần

### Where

like in sql server code

VD: db.CategoryOfProducts.ToList().Where(c =>c.CategoryOfProductID>=4)

#### Contains(“string”)

vd: db.CategoryOfProducts.ToList().Where(c =>c.Description.Contains("uốc");

tương tự như like “&string&”

Không có and như trong sql

bao nhiêu điều kiện thì bấy nhiêu where

db.CategoryOfProducts.ToList().Where(c =>c.Description.Contains("c")).Where(c=>c.CategoryOfProductID >4);

Or có thể dùng && VD:

var list = db.CategoryOfProducts.Where(c => c.Description.Contains("u") && c.CategoryOfProductID > 7);

Bạn có thể ghép nối cả Where và OrderBy (OrderByDescending) để vừa lọc vừa sắp xếp:

var query = db. CategoryOfProducts.Where(c => c. CategoryOfProductID >= 4) .OrderBy(c => c.Credit) .ThenBy(c => c.Name);

### Select

chỉ select ra những cột mình mong muốn

vd: var query = db.Courses .Where(c => c.Credit > 3) .OrderBy(c => c.Name) .Select(c => c.Name);

vd: var query = db.Courses .Where(c => c.Credit > 3) .OrderBy(c => c.Name) .Select(c => new{ c.Name, c.Credit });

Truy vấn này khác biệt ở chỗ trong phương thức Select bạn thực hiện projection từ kiểu Course sang một kiểu vô danh (anonymous type). Trong đó, kiểu vô danh chứa giá trị Name và Credit của Course.

mỗi object trong danh sách kết quả trả về giờ thuộc một kiểu vô danh chỉ chứa hai property Name (string) và Credit (int).

Dĩ nhiên bạn có thể truy xuất các property này như đối với các object bình thường:

var query = db.Courses .Where(c => c.Credit > 3) .OrderBy(c => c.Name) .Select(c => new Courses (){ c.Name, c.Credit });

### Single and SingOrDefault

.Single();

.SingleOrDefault();

Khi sử dụng Single (hoặc SingleOrDefault), bạn vẫn xây dựng truy vấn lọc dữ liệu LINQ như bình thường. Tuy nhiên, Single báo cho Entity Framework biết rằng bạn đang chờ đợi MỘT kết quả trả về duy nhất. Do vậy, nếu truy vấn trả về nhiều hơn một kết quả, hoặc không có kết quả nào, Single sẽ phát ra ngoại lệ System.InvalidOperationException: 'Sequence contains more than one element'.

Single và SingleOrDefault hoạt động cơ bản là giống nhau. Sự khác biệt là nếu trong trường hợp không có kết quả trả về, Single sẽ phát ra ngoại lệ trong khi SingleOrDefault đơn giản là trả về null. Nếu có nhiều hơn 1 kết quả trả về, cả Single và SingleOrDefault đều phát ra ngoại lệ.

! Single tương đối “cực đoan” khi đòi hỏi bạn phải đảm bảo rằng truy vấn chỉ trả về một kết quả duy nhất. Nó đặc biệt phù hợp khi bạn tìm kiếm object theo Id.

### First and FirstOrDefault

Sự khác biệt với Single(SingleOrDefault), nếu có nhiều kết quả trả về, First (FirstOrDefault) sẽ lấy object đầu tiên trong danh sách. Nếu không có kết quả trả về, First sẽ phát ra ngoại lệ, trong khi FirstOrDefault trả lại null (không phát ra ngoại lệ). Cú pháp sử dụng First (FirstOrDefault) giống hệt như Single (SingleOrDefault):

### Find

Find hoạt động khác biệt so với Single hoặc First ở các điểm sau: Find tìm kiếm trong những object đã tải từ cơ sở dữ liệu hoặc những object đã gắn với context (nhưng chưa lưu vào cơ sở dữ liệu).

Nói chung là Find tìm kiếm những object đang nằm trong bộ nhớ. Nếu object chưa tải vào bộ nhớ, Find cũng tìm kiếm trong cơ sở dữ liệu. Như vậy, Find hiệu quả hơn vì nó không nhất thiết phải thực hiện truy vấn tới cơ sở dữ liệu.

Ở khía cạnh khác, Single và First luôn tạo ra truy vấn tới cơ sở dữ liệu để tìm bản ghi phù hợp. Những object mới đang được context quản lý nhưng chưa lưu vào cơ sở dữ liệu sẽ không xuất hiện trong kết quả tìm kiếm.

! Lưu ý rằng, mặc dù Find chấp nhận không giới hạn tham số

đầu vào nhưng bạn chỉ có thể truyền giá trị khóa chính cho Find. Nếu bạn truyền những giá trị khác Find sẽ báo lỗi khi chạy (mặc dù vẫn dịch thành công).

### Take

.Take(số bản ghi muốn lấy lấy ra)

### Include

chính là việc gộp trường của các bảng select \* ..join

var customers = context.Customers.ToList();

tương ứng với: SELECT \* FROM Customers;

var customersWithOrderDetail = context.Customers.Include("Orders").ToList();

tương ứng với: SELECT \*

FROM Customers JOIN Orders ON Customers.Id = Orders.CustomerId;

var phieuChis = db.PhieuChis.Include(p => p.MucChi).Include(p => p.NhanVien);

tương ứng với: SELECT \*

FROM Phieuchi JOIN MucChi ON Phieuchi.maMC = MucChi.maMC JOIN NhanVien ON PhieuChi.maNV = NhanVien.maNV;

chú ý: 2 cái này vẫn có thể hiển thự như nhau

var departments = db.Departments.Include(d => d.Administrator); (háo hức)

var departments = db.Departments;(lười biếng)

giải quyết:

<https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/reading-related-data-with-the-entity-framework-in-an-asp-net-mvc-application>

### GroupBy

tương tự group by trong sql

var result = db.Courses.GroupBy(c => c.Credit);

**\*có thể kết hợp tất cả như trong câu lệnh sql**

vd: var result = db.Courses .GroupBy(c => c.Credit) .OrderByDescending(g => g.Key) .Select( g=> new { GroupName = g.Key, GroupItems = g.OrderBy(c=>c.Name)})

Dùng && như là and trong sql:

vd

var result = db.Courses .GroupBy(c => c.Credit) .OrderByDescending(g => g.Key) .Select( g=> new { GroupName = g.Key, GroupItems = g.OrderBy(c=>c.Name && c.key >5)})

VD kinh điển

var ps = db.Products.GroupBy(p => new {p.CategoryOfProductID }).OrderByDescending(o =>o.Key.CategoryOfProductID).Select(s => new {id = s.Key.CategoryOfProductID, Prd =s.OrderBy(x =>x.NameOfProduct) });

List<Product> List = new List<Product>();

foreach(var item in ps)

{

foreach(var p in item.Prd)

{

List.Add( new Product { ProductID = p.ProductID, NameOfProduct = p.NameOfProduct, CategoryOfProductID = p.CategoryOfProductID, CategoryOfProduct = p.CategoryOfProduct});

}

}

return View(List.ToList());

Kết luận:

key: chính là các thuộc tính gom nhóm, muốn lấy thông tin của các thuộc tính gom nhóm, chỉ cần .key.<thuộc tính>

var ps:

* id
* Prd( chính là product)
  + ProductID
  + NameOfProduct
  + CatelogyOfProducdID
  + CatelogyOfProducd

Tài liệu tham khảo: <https://www.codeproject.com/Questions/1280419/How-to-use-groupby-data-in-view-to-display-a-resul>

### ToList

ToList copy dữ liệu thu được vào một danh sách cục bộ để tái sử dụng. Tuy nhiên, cách làm này có một số nhược điểm.

* ToList trên truy vấn LINQ to Entities chỉ lưu trữ những dữ liệu lấy được từ cơ sở dữ liệu. Trong quá trình hoạt động, những object mới có thể đã được thêm vào danh mục quản lý của context(EF or Model tùy vào cách mình đặt tên thư mục) nhưng chưa được lưu trở lại cơ sở dữ liệu thì không xuất hiện trong kết quả truy vấn.

Any (AnyAsnyc)

* trả về kiểu boole
* là cách nhanh nhất để kiểm tra xem đối tượng có tồn tại trong cơ sở dữ liệu hay không

### Lưu trữ Local

Trong mô hình này, dữ liệu tải từ bất kỳ truy vấn LINQ to Entities nào sẽ được lưu lại trong chương trình. Bạn có thể truy vấn trực tiếp các object này mà không cần tương tác với cơ sở dữ liệu.

Context sẽ theo dõi tất cả các object (thêm mới, cập nhật, xóa). Kết quả các truy vấn dữ liệu cục bộ sẽ bao gồm cả các object cũ và object mới.

Khi cần lưu dữ liệu, context căn cứ vào trạng thái của các object nó quản lý và chỉ tạo ra các truy vấn tương ứng nếu object có thay đổi.

Local data thực chất chỉ là một danh sách object thông thường, các truy vấn LINQ phát đến nó không còn là LINQ to Entities nữa mà chính là LINQ to Objects quen thuộc. Do đó truy vấn thực hiện nhanh chóng hơn nhiều (vì không cần dịch sang SQL và thực hiện trên cơ sở dữ liệu).

Local data được lưu ở dạng ObservableCollection. Loại cấu trúc này phát ra sự kiện mỗi khi có sự thay đổi dữ liệu. Do đó nó đặc biệt phù hợp để xây dựng các ứng dụng desktop với cơ chế Data Binding và mô hình sự kiện.

Ngoài ra khi sử dụng Local data bạn thậm chí có thể tự mình theo dõi sự biến đổi của object thông qua các thông tin theo dõi (tracking) của context. Nó rất có ích nếu bạn cần thực hiện các thao tác khôi phục dữ liệu (ví dụ lỡ bị thay đổi nhầm, giờ bạn muốn trả lại giá trị cũ).

VD : count = db.Courses.Local.Count;

## LINQ Quere Expression

được hiểu như các truy vấn thuần

cũng để sử dụng để truy vấn dữ liệu như [LINQ to ENTITY FRAMEWORK](#_LINQ_to_ENTITY) nhưng nó gần giống với câu lệnh sql hơn.

vd:

var list = from c in db.CategoryOfProducts where c.Description.Contains("u") where c.CategoryOfProductID>7 select c;

return View(list);

### Select

lợi thế so với câu lệnh sql là có thể chỉnh sửa dữ liệu ngay khi gọi từ db như ‘LoaiDeTai”

listItems = (from k in db.DeTais

join gv\_dt in db.GV\_DeTai on k.Id equals gv\_dt.IdDeTai

where gv\_dt.IdGiaoVien == idGiaoVien && k.NamHoc == nam && k.KiHoc == ki

select new DeTaiModel()

{

Id = k.Id,

Ma = k.Ma,

Ten = k.Ten,

CoQuanQuanLy = k.CoQuanQuanLy,

NamHoc = k.NamHoc,

KiHoc = k.KiHoc,

SoThanhVien = k.SoThanhVien,

IdLoaiDeTai = k.IdLoaiDeTai,

SoGio = gv\_dt.SoGio,

LaChuTri = gv\_dt.LaChuTri,

LoaiDeTai = k.IdLoaiDeTai == null ? null : new LoaiDTModel()

{

Id = k.LoaiDeTai.Id,

Ten = k.LoaiDeTai.Ten

}

}).ToList<DeTaiModel>();

### Where

có thể thay nhiều where như ở vd trên thành **&&**

vd: var list = from c in db.CategoryOfProducts where c.Description.Contains("u") && c.CategoryOfProductID>7 select c;

tương thự thay thế OR trong sql thành **||**

**vd:** var list = from c in db.CategoryOfProducts where c.Description.Contains("u") || c.CategoryOfProductID>7 select c;

### OderBy

vd: Thứ tự sắp xếp từ name sau đó đến birthday

    var queryLondonCustomers3 =

    from cust in customers

    where cust.City == "Đà Lạt"

    orderby cust.Name ascending, cust.Birthday descending

    select cust;

ascending: tăng dần

descending: giảm dần

### Group\*

**chính là GroupBy**

vd: gom nhóm City và Name

var queryCustomersByCity = from cust in customers

                           group cust by cust.City , cust.Name;

### Into

khi bạn truy vấn với 1 câu lệnh dài để ra 1 data. Bạn muốn lấy data đó để tiếp tục truy vấn. Giống như truy vấn lồng trong sql, ta thay (…) **bằng into:**

**vd:**

var= from u in db.User

from t= in db.TieuSus

where u.IDUser = t.IDUser

orderby u.IDUser descending

select new {  
 ID = u.IDUser

Ten = u.Name

TieuSu= t.Info

} **into** newselect where newselect.ID.CompareTo(“1”) select newselect;

## Razor

### @

### có thể là 1 [thẻ](#_@html.ActionLink), 1 [biến](#_model)

### @html.ActionLink

Tương tự như thẻ <a>, các parameters cách nhau bởi dấu ‘,’

@html.ActionLink(“string hiển thị”, “ string actionName- tên của ActionResult”, “string controllerName – tên của controller điều khiển”, new { @(có thể có @ hoặc không)parameters = model.<thuộc tính> }, new {area =”<tên area>”,(orther override)..)

vd: area

1. @Html.ActionLink("Admin", "Index", "Meets", new { area = "Admin" }, null) ( tham số cuối phải có null nó mới chạy)
2. @Html.ActionLink("Admin", "Index", "Meets", new { area = "" }, null)

### BeginForm

@using (Html.BeginForm("ImformationOfAccount","ImformationOfAccount", new { @FullName = "Vũ Minh Hiếu" }))

{ //code}

string 1: ImformationOfAccount: tên của action

string 1: ImformationOfAccount: tên của controller chứa action

new { @FullName = "Vũ Minh Hiếu" })): có thẻ tạo ra vô số biến để truyền vào action, chúng cách nhau bởi dấu ‘,’

* public ActionResult ImformationOfAccount(FormCollection collection, string FullName)

Lưu ý: <@FullName> phải trùng tham số <string FullName> trong action

* <@FullName> không cần có @đằng trước cũng được

### So sánh các loại thẻ

#### **TextBox và TextBoxFor**

Phương thức **Html.TextBox ()** tạo phần tử **<input type="text" />** với tên được chỉ định, giá trị và thuộc tính html.

Ví dụ:

@Html.TextBox(“StudentName”, “hello everyone”, new { @class = “form-control” })

sẽ tạo ra **<input class="”form-control”" id="”StudentName”" name="”StudentName”" type="”text”" value="”hello" everyone”="">**

**Html.TextBoxFor** tạo ra một phần tử input text với thuộc tính model xác định sử dụng một biểu thức lambda.

Ví dụ:

@model Student @Html.TextBoxFor(m => m.StudentName, new { @class = “form-control” })

sẽ tạo ra **<input class="”form-control”" id="”StudentName”" name="”StudentName”" type="”text”" value="”This" is="" value”="">**

## ActionResult

mỗi action result sẽ tương ứng với 1 gì đó chưa hiểu, anything may is view. And so that return something that is knowed, usually return view();

## Truyền giá trị vào ActionResult

### form

[form](#_BeginForm) sẽ tìm đến 1 methor dạng ActionResult tương ứng. nếu muốn lấy dữ liệu từ form. có 2 cách

1. tạo [FormCollection](#_FormCollection)
2. tạo các biến có tên trùng với tên của [parameter](#_BeginForm) được khai báo trong form

vd: từ vd [BeginForm](#_BeginForm) ta có trong collection 1 hàm với parameter FullName:

public ActionResult ImformationOfAccount(FormCollection collection, string FullName)

{

ViewBag.Name = FullName;

ViewBag.Email = collection.Get("txtEmail").ToString(); return View();

}

Hoặc binding dữ liệu trực tiếp như code mẫu trong mvc

public ActionResult Edit([Bind(Include = "CategoryOfProductID,Description")] CategoryOfProduct categoryOfProduct){}

### ActionLink

[ActionLink](#_@html.ActionLink) khi gọ đến 1 method

@Html.ActionLink("Details", "Details", new { id=item.ProductID , @age = 21 })

trong controller details có ActionResult Details có parameter id để nhận kết quả

public ActionResult Details(int? id, int? age){}

* **có thể thêm bao nhiêu parameter tùy ý**
* **nếu các hàm overload gọi đến có parameter htmlAttributes thì để là null**

### Action

Cú pháp tương tự như actionlink(Thường dùng để load 1 partialView)

### window.location.href

VD: window.location.href = "/ShopCheckout/ConfirmOrder?address=" + Address + "&phone=" + Phone;

Câu lệnh để trong file js; khác với action hoặc actionlink, nó dùng để gửi về dữ liệu động (Address, Phone)

Lưu ý. Không nên bắt sự kiện của thẻ có type = submit và được đặt trong form vì nó sẽ chạy form trước sau đó mới chạy lệnh window.location.href nên ấn lần 2 mới được. Nhớ bỏ thẻ form đi

trường hợp gửi dữ liệu bất kỳ không tuân thủ form. Bên controller nhớ [HTTP request methods](#_HTTP_request_methods) là get. Vì dữ liệu được thêm tường minh vào url

### truyền json với ajax

dữ liệu json trả về chỉ cần đúng tên thuộc tính của model cần ép kiểu sang. Thừa thiếu thuộc tính không quan trọng. như vd dưới. thuộc tính Products không cần có đủ tất cả các thuộc tính con mà chỉ cần id.

#### trên view

$('#btnUpdate').off('click').on('click', function () {

var listProduct = $('.txtquantity');

var cartList = [];

var dcCode = $('#txtdiscountCode').val();

$.each(listProduct, function (i, item) { // cách thức lấy từng phần tử của 1

cartList.push({ list các thành phần giống nhau//

Amount: $(item).val(), // I: index của phần tử

Products: { // item: từng phần tử được lấy ra;

id\_product: $(item).data('id')

}

});

});

$.ajax({

url: '/ShopCart/Update',

data: { cartModel: JSON.stringify(cartList) }, // ép kiểu json sang chuỗi

dataType: 'json',

type: "POST",

success: function (res) {

if (res.status == true) {

window.location.href = "/ShopCart/Index?discountCode=" + dcCode;

}

}

});

});

#### Trong controller

public JsonResult Update(string cartModel)

{

var jsonCart = new JavaScriptSerializer().Deserialize<List<CartItem>>(cartModel); // lệnh ép kiểu sang 1 model có sẵn

var sessionCart = (List<CartItem>)Session[CartSession];

foreach (var item in sessionCart)

{

var jsonItem = jsonCart.SingleOrDefault(x => x.Products.id\_product == item.Products.id\_product);

if (jsonItem != null)

{

item.Amount = jsonItem.Amount;

item.Price = item.Products.price.GetValueOrDefault(0);

item.IntoMoney = item.Price \* item.Amount;

}

}

Session[CartSession] = sessionCart;

return Json(new

{

status = true // , tên thuộc tính = giá trị, …

});

#### }

## truyền giá trị cho view

### ViewBag

* là 1 đối tượng kiểu dynamic
* cho phép taoj ra đối tượn con truyền dữ liệu
* thêm đối tượng thoải mái với ViewBag, không bị giới hạn sử dụng như model của view

vd :

public ActionResult ImformationOfAccount(FormCollection collection, string FullName)

{

ViewBag.Name = FullName;

ViewBag.Email = collection.Get("txtEmail").ToString();

return View();

}

ở view ta có:

<h3 style="margin-left: 20px; margin-top: 20px">Name: @ViewBag.Name</h3>

<h3 style="margin-left: 20px; margin-top: 20px">You singed in with eamil: @ViewBag.Email</h3>

### model

sử dụng : return View(<tên model>);

VD

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Product product = db.Products.Find(id);

if (product == null)

{

return HttpNotFound();

}

return View(product);

}

-ở đây ta đã truyề tới view 1 kiểu object là product

ở phần view ta sẽ khai báo:

@model CodeFirstExample.Models.Product

Trường hợp ActionResult trả về dạng mảng: như return View(List.ToList()); ta thêm vào đầu trang view 1 @model Ienumerable<…>

vd: @model IEnumerable<CodeFirstExample.Models.Product>

Hoặc có thể viết chặt chẽ hơn. cho phép tạo các biến kiểu Product hoặc truy xuất dễ dàng với model

vd: @using CodeFirstExample.Models;

@model IEnumerable<Product>

***Lưu ý:***

*nếu ở trong 1 cú pháp razor nào đó thì có thể thêm @ hoặc không phía trước tên biến*

*còn lại phải bắt buộc thêm @trước tên biến*

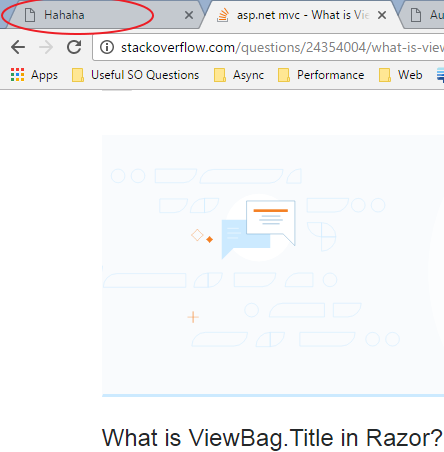
### ViewBag.Title=”tên của view”;

Trong trang \_Layout.cshtml bạn sẽ thấy dòng mã này:

<title>@ViewBag.Title</title>

### FormCollection

- Form collection đựơc sử dụng để tập hợp dữ liệu được chứa trong các phần tử của form chuyển từ client đến server bằng phương thức POST  
- Lấy dữ liệu từ các phần tử của form theo cú pháp:  
+ ActionResult ActionName(FormCollection frm) => cách lấy dữ liệu như sau: frm["fieldName"] với fieldName là Name của các filed name cần gửi về để xử lý.  
\*\*\* Form Collection thuận tiện cho các bạn lập trình khi cần gửi một form dữ liệu với nhiều filedName



Khi bạn đang ở view nào thì sẽ lấy title của view ấy

### Truyền từ trong 1 namespase với bất kỳ biến static nào

vd: <li class="@Html.IsSelected(controller: "group\_role", action: "Index")"><a href="@lte.MVC.Pages.QLKS.Home"><i class="fa fa-circle-o"></i> Lễ tân</a></li>

*! đây cũng là 1 cú pháp hay hơn thay cho @actionlink*

với @lte.MVC.Pages.QLKS.Home chính là 1 biến string là đường dẫn đến 1 trang cố định

namespace lte.MVC

{

public static class Pages

{

public static class QLKS

{

public const string Home = "/TTNhom\_QLKS/Home/Index";

}

## View(object model)

tham khảo [model](#_model)

## View(string viewName)

vd : [HttpPost]

public ActionResult Login(FormCollection collection)

{

//code

return View("Index");

}

view sẽ hiểm thị Index.cshtml thay vì view mặc định của login

## View(string viewName, object model)

tương tự 

## session

lưu nội dung thông tin tài khoản có thể dùng ở mọi chỗ(view, controller,…) như là 1 biến toàn cục

vd:

Session.Add(“tên biến”, biến(kiểu dữ liệu cơ bản int, string, …));

Session.Add(“USER\_SESSION”, userSession.UserName);

Session.Add(“ID\_SESSION”, userSession.UserID);

trong layout:

@Session["USER\_SESSION"] => lấy userSession.UserName

@Session["ID\_SESSION "] => lấy userSession.UserID

### RenderSection

truyền giá trị có trong renderBody lên \_layout.cshtml

VD:

<html>

<body>

@RenderBody()

@RenderSection("header", required: true)

</body>

</html>

then you can have an index.cshtml content view like this

@section header {

Trang chủ

}

## OnActionExecuted\*

ActionExecutedContext\*

### filterContext\*

BaseController

vd:

public class BaseController : Controller

{

protected override void OnActionExecuted(ActionExecutedContext filterContext)

{

var session = Session[CommonConstants.USER\_SESSION];

if(session == null)

{

filterContext.Result = new RedirectToRouteResult(new System.Web.Routing.RouteValueDictionary(new { Controller = "login", action = "Index", Area = "admin" }));

}

base.OnActionExecuted(filterContext);

}

}

dùng để kiểmt tra đăng nhập . nếu đăng nhập rồi thì cho vào

* các lớp controller khác phải kế thừa từ BaseController nếu cần đăng nhập trước

## RedirectToAction\*

## PartialView

### trong [Action](#_Action)

Vd:

trong .htmlcs thêm

@Html.Action("HeaderCart", "ShopCart")

Trong

public ActionResult HeaderCart()

{

var cart = Session[CommonConstants.CartSession];

var list = new List<CartItem>();

if (cart != null)

{

list = (List<CartItem>)cart;

}

return PartialView(list);

}

* + render ra partial view có model hoặc không

### Trực tiếp trong .htmlcs

### Vd

@Html.Partial("\_MainLayoutHeader")

\_MainLayoutHeader.htmlcs Là 1 view mà không cần tới controller và đương nhiên nó không có model

* + nếu muốn hiển thị dữ liệu trong file này có thể sử dụng session hoặc thêm 1 [action()](#_trong_Action) bên trong

## Html.RenderAction\*

## Kỹ thuật phân trang trong MVC

!cài đặt pagedlist trong nuget packet

sử dụng

db.User.ToPageList(trang số, số lượng của 1 trang);

## 10 cách để liên kết nhiều mô hình trên một khung nhìn trong MVC – CodeProject

<https://www.codeproject.com/Articles/1108855/ways-to-Bind-Multiple-Models-on-a-View-in-MVC>

## ValidateAntiForgeryToken

<https://tedu.com.vn/bao-mat/series-bao-mat-trong-aspnet-mvc-2-cross-site-request-forgery-csrf-96.html>

## Phân trang trong MVC

Cài đặt gói PagedList và PagedList.mvc

## Webconfig

### <appSettings>

* Cài đặt các key value cho project tương tự như 1 biến local cho toàn bộ project

vd:

<appSettings>

<!--Email Config-->

<add key="FromEmailAddress" value="chiminhtoi0112@gmail.com"/>

<add key="FromEmailDisplayName" value="Đơn hàng mới"/>

<add key="FromEmailPassword" value="linh160999"/>

<add key="SMTPHost" value="smtp.gmail.com"/>

<add key="SMTPPort" value="587"/>

<add key="EnabledSSL" value="true"/>

<add key="ToEmailAddress" value="vuhieupro1999@gmail.com"/>

</appSettings>

ta có thể gọi chngs ở bất kỳ đâu với cú pháp:

ConfigurationManager.AppSettings["FromEmailAddress"].ToString();

# Ajax

* .done() thay thế cho .success()
* .fail() thay thế cho .error()
* .always() thay thế cho .complete() Tương ứng với đoạn code ở trên, ta có thể viết lại như sau:

$.ajax({

data: someData,

dataType: 'json',

url: '/path/to/script'

}).done(function(data) {

// If successful

console.log(data);

}).fail(function(jqXHR, textStatus, errorThrown) {

// If fail

console.log(textStatus + ': ' + errorThrown);

});

## Ép kiểu sang json trong C#

Cài nuget packet : Newtonsoft.json

Vd:

Public JsonResult Country(){

var json = JsonConvert.SerializeObject(bd.Customer.toList());

Return Json(json, JsonRequestBehavior.AllowGet);

}

## Ép kiểu json sang đối tượng trong C#

JavaScriptSerializer().Deserialize<List<CartItem>>(cartModel);

## @Ajax.ActionLink(..)

Công dụng: render 1 partialView mà không cần load lại trang:

Gửi kiểu text

data: { <tên biến>: <giá trị> } //tên biến phải trùng với parameter của hàm nhận phía backend

Vd:

$('#userName').on('input', function () {

var userName = this.value;

$.ajax({

url : '/Admin/Customer/CheckUserName',

data: { userName: userName },

dataType : 'text',

type: "POST",

success: function (res) {

if (res.status == true) {

$('#CreateCustomer .text-success').remove();

$(this).parentNode.after('<p class="text-danger">.Tài khoản đã tồn tại</p>');

}

else {

$('#CreateCustomer .text-danger').remove();

$(this).parentNode.after('<p class="text-success">.Có thẻ sử dụng</p>');

}

}

});

});

Gửi kiểu json

data: { <tên biến>: JSON.stringify(<giá trị>) } //tên biến phải trùng với parameter của hàm nhận phía backend

Vd:

$('#btnUpdate').off('click').on('click', function () {

var listProduct = $('.txtquantity');

var cartList = [];

var dcCode = $('#txtdiscountCode').val();

$.each(listProduct, function (i, item) {

cartList.push({

Amount: $(item).val(),

Products: {

id\_product: $(item).data('id')

}

});

});

$.ajax({

url: '/ShopCart/Update',

data: { cartModel: JSON.stringify(cartList) },

dataType: 'json',

type: "POST",

success: function (res) {

if (res.status == true) {

window.location.href = "/ShopCart/Index?discountCode=" + dcCode;

}

}

});

});

phía backend

JavaScriptSerializer().Deserialize<<kiểu muốn chuyển>>(<dữ liệu truyền vào>);

Vd:

public JsonResult Update(string cartModel)//tên của parameter phải trùng bên phía frontend

{

var jsonCart = new JavaScriptSerializer().Deserialize<List<CartItem>>(cartModel);

// code

return Json(new

{

status = true

});

}

## contentType

là loại dữ liệu bạn đang gửi, loại dữ liệu application/json; charset=utf-8phổ biến cũng vậy, là loại dữ liệu application/x-www-form-urlencoded; charset=UTF-8mặc định.

## dataType

là những gì bạn đang mong trở lại từ máy chủ: json, html, textvv jQuery sẽ sử dụng để tìm ra cách để cư tham số chức năng thành công của.

# Storage

## Local storage (angular)

Kiểm tra trình duyệt có hỗ trợ storage : if(typeof Storage!== “underfined”)

Lấy value: window.localStorage.getItem(<key>)

Set value: window.localStorage.setItem(<key>, <value>)

Xóa value: window.localStorage.removeItem(<key>, <value>)

Đặc điểm

* Lưu trữ trên trình duyệt
* Tối đa đến 5 mb
* Thời lượng sử dụng vô thời hạn
* Người dùng có thể xóa
* Nên dùng cho những thông tin không nhạy cảm

Nhựơc điểm:

* Chỉ lưu được ở dạng string
* Bảo mật kém vì bất kỳ 1 js nà cũng có thể truy cập được

## sesstioStorage

* Tương tự như localstorage nhưng sẽ bị mất nếu người dùng lose tab hay close browser

Kiểm tra trình duyệt có hỗ trợ storage : if(typeof Storage!== “underfined”)

Lấy value: window.sessionStorage.getItem(<key>)

Set value: window. sessionStorage.setItem(<key>, <value>)

Xóa value: window. sessionStorage.removeItem(<key>, <value>)

## Cookies

Với local storage chỉ được dùng ở brower client, cookies cos theer access được ở cả brower client và server. Khi tạo 1 http request thì cookies của brower sẽ được attach vào hearder Cookie, tờ đó phía server có thể parse header này và get được data cookie.

* cookies có thời gian hết hạn expires, sau thời gian này thì cookies sẽ biến mất khỏi browser
* cookies chỉ cho phép lưu tối đa khoảng 4 KB, vì vậy ta nên sử dụng cookies với mục đích save những loại data simple ví dụ như token cho authentication,...

# C#

Kết nối db luôn đặt trong try- catch và finally luôn phải đóng kết nối

## Base

### **Compare Abstract Class and Interface**

Giống nhau:

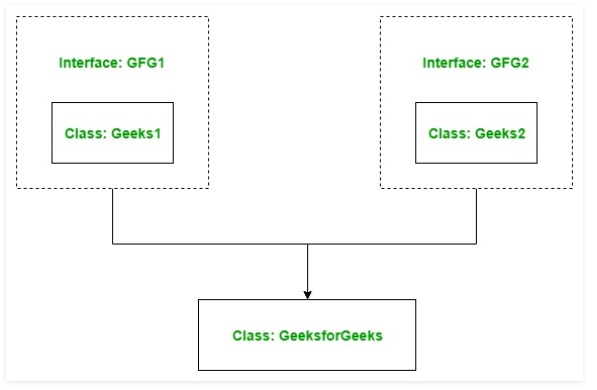
* Abstract class và interface đều không thể khởi tạo đối tượng bên trong được.
* Abstract class và interface đều có thể khai báo các phương thức nhưng không thực hiện chúng.
* Abstract class và interface đều bao gồm các phương thức abstract.
* Abstract class và interface đều được thực thi từ các class con hay còn gọi kế thừa, dẫn xuất.
* Abstract class và interface đều có thể kế thừa từ nhiều interface.

Khác nhau:

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| Cho phép khai báo field | Không cho phép |
| Các phương thức có thể có thân hàm hoặc không có thân hàm. | Chỉ khai báo không có thân hàm |
| Class dẫn xuất chỉ kế thừa được từ 1 abstract class và nhiều interface. | Class triển khai có thể triển khai nhiều interface. |
| Có chứa constructor | Không có |
| Các phương thức có từ khóa access modifier | Không có |

Đó chính là các điểm giống và khác nhau của Abstract class và interface ở góc độ ngôn ngữ và lập trình. Còn bản chất và mục đích sử dụng khác nhau mới là phần chính chúng ta phải hiểu.

### Kế thừa



Ta có thể lấy interface thông qua kế thừa từ lớp cha

Hoặc có thể triển khai đồng thời

class GeeksforGeeks: Geeks2, GFG2

**Lưu ý:** Lớp cơ sở phải đứng trước danh sách tên giao diện.

Tất nhiên, bạn vẫn cần phải triển khai tất cả các thành viên mà các giao diện xác định. Tuy nhiên, nếu lớp cha chứa một thành viên phù hợp với một thành viên giao diện, thì thành viên lớp cah có thể hoạt động như việc triển khai thành viên giao diện và bạn không bắt buộc phải triển khai lại theo cách thủ công.

### Params

Trong nhiều trường hợp, nhiều lúc chúng ta muốn viết một hàm, mà không biết trước được là hàm đó có bao nhiêu tham số truyền vào.

Và hàm này có thể cho phép người dùng truyền vào nhiều tham số.

Mình ví dụ:

Chúng ta cần viết một hàm sau, cho phép truyền vào nhiều số tự nhiên, và trả về tổng kết quả của các số nhập vào.

public static int Sum(params int[] arr)

{

int sum = 0;

foreach (int item in arr)

sum += item;

return sum;

}

C#

Copy

Từ khóa params chúng ta truyền vào là một mảng object.

Dưới đây là sử dụng hàm trên:

int sum2 = Sum( 32, 8, 5, 20, 1, 9, 10);

Kỹ thuật mapping 2 class thông qua json

var resData = JObject.Parse(JsonConvert.SerializeObject(shop));

var datas = resData.ToObject<shopTest>(new JsonSerializer());

or var data11 = JsonConvert.DeserializeObject<shopTest>(JsonConvert.SerializeObject(shop));

delegate

### Định nghĩa

**1** Đầu tiên cần khai báo một **delegate**, khai báo giống như cách khai báo phương thức nhưng có thêm từ khóa delegate và không có thân phương thức. Ví dụ sau khai báo một delegate có tên là **ShowLog**

public delegate void ShowLog(string message);

**2** Khi đã có **ShowLog**, nó dùng như một kiểu dữ liệu để khai báo các biến, các biến này có thể gán vào nó các hàm có sự tương đồng về tham số và kiểu trả về với khai báo delegate ví dụ khai báo biến:

ShowLog showLog;

Thi hành delegate

Sau khi biến delegate được gán hàm vào, có thể dùng biến delegate để thi hành bằng cách gọi:  
**varDelegate.Invoke(các-tham-số)** hoặc **varDelegate(các-tham-số)**

**3** Tạo hai phương thức Info và Warning có tham số giống với ShowLog, nghĩa là có một tham số kiểu string, trả về void:

static public void Info(string s)

{

// ...

}

static public void Warning(string s)

{

// ...

}

Do Info, Warning có tương đồng về tham số với delegate trên, nên hai hàm này có thể dùng để gán vào biến kiểu ShowLog, xem đoạn mã đầy đủ sau:

Logs.cs

using System;

namespace CS008\_Anonymous

{

public class Logs

{

// Khai báo một delegate

public delegate void ShowLog(string message);

// Phương thức tương đồng với ShowLog (tham số, kiểu trả về)

static public void Info(string s)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine(string.Format("Info: {0}", s));

Console.ResetColor();

}

// Phương thức tương đồng với ShowLog (tham số, kiểu trả về)

static public void Warning(string s)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(string.Format("Waring: {0}", s));

Console.ResetColor();

}

public static void TestShowLog()

{

ShowLog showLog;

showLog = Info; // showLog gán bằng phương thức Info

showLog("Thông báo"); // Thi hành delegate chính là thi hành Info

showLog = Warning; // showLog gán bằng phương thức Warning

showLog("Thông báo"); // Thi hành delegate chính là thi hành Info

}

}

}

Kết quả chạy đoạn, khi gọi hàm Logs.TestShowLog();:

Waring: Thông báo

Info: Thông báo

### Gán nhiều phương thức vào delegate

Khi dùng delegate chạy một phương thức, cần đảm bảo biến delegate đó đã được gán phương thức (biến khác null), có thể bạn kiểm tra trước khi gọi ví dụ: if (showLog != null) showLog("Mgs") hoặc gắn gọn hơn showLog?.Invoke("Mgs");

**4** Một delegate có thể đưa vào nó nhiều phương thức để một lần gọi thi hành tất cả các phương thức nó chứa

* **Toán tử +=** Nối thêm một phương thức vào delegate, ví dụ delegatevar += method1
* **Toán tử -=** : Loại bỏ 1 phương ở cuối (nếu phương thức đó có trong delegate, tính từ cuối) , ví dụ delegatevar -= method1

Ví dụ:

public static void TestShowLogMulti()

{

ShowLog showLog;

showLog = null;

showLog += Warning; // Nối thêm Warning vào delegate

showLog += Info; // Nối thêm Info vào delegate

showLog += Warning; // Nối thêm Warning vào delegate

//Một lần gọi thi hành tất cả các phương thức trong chuỗi delegate

showLog("TestLog"); //Hoặc an toàn: showLog?.Invoke("TestLog");

}

Gọi phương thức TestShowLogMulti thì kết quả:

Waring: TestLog

Info: TestLog

Waring: TestLog

Các delegate cùng kiểu có thể kết hợp lại với nhau bằng toán tử +, ví dụ:

// Cộng nhiều Delegate

public static void TestShowLogPlus()

{

ShowLog showLog1 = (x)=> {Console.WriteLine($"-----{x}-----");};

ShowLog showLog2 = Warning;

ShowLog showLog3 = Info;

var all = showLog1 + showLog2 + showLog3 + showLog1;

all("Xin Chào");

}

Gọi phương thức TestShowLogPlus kết quả là:

-----Xin Chào-----

Waring: Xin Chào

Info: Xin Chào

-----Xin Chào-----

### Func và Action

Func và Action là hai mẫu delegate định nghĩa sẵn, giúp bạn nhanh chóng tạo ra biến kiểu delegate mà không mất công khai báo, xem lại ví dụ trên nếu sử dụng đến Func, Action thì không cần có dòng khai báo:

public delegate void ShowLog(string message);

#### Sử dụng Func

**Func** là mẫu delegate có kiểu trả về. Để khai báo biến delegate dùng cú pháp như sau:

Func<kiểu\_tham\_số\_1, kiểu\_tham\_số\_2, ..., kiểu\_trả\_về> var\_delegate;

Kiểu cuối cùng trong khai báo Func là kiểu trả về của hàm, có thể thiếu tham số nhưng không được thiếu kiểu trả về

Ví dụ muốn có biến delegate tên bien1 tương đương với hàm có 2 tham số, tham số 1 kiểu int, tham số 2 kiểu string, và hàm trả về kiểu bool thì tạo biến đó như sau:

Func<int, string, bool> bien1;

Khai báo trên nếu bạn dùng cách thông thường tương ứng với:

// Khai báo delegate ở lớp

delegate bool DelegateName(int a, string b);

// Khai báo biến trong phương thức

DelegateName bien1;

**Ví dụ:**

using System;

namespace CS008\_Anonymous

{

class FuncAction

{

static int Sum(int x, int y)

{

return x + y;

}

public static void TestFunc(int x, int y)

{

Func<int,int,int> tinhtong; // biến tinhtong phù hợp với các hàm trả về kiểu int, có 2 tham số kiểu int

tinhtong = Sum; // Hàm Sum phù hợp nên có thể gán cho biến

var ketqua = tinhtong(x, y);

Console.WriteLine(ketqua);

}

}

}

Khi gọi phương thức TestFunc kết quả:

FuncAction.TestFunc(5, 6); // In ra: 11

#### Sử dụng Action

Action tương tự như Func, điều khác duy nhất là nó không có kiểu trả về, khai báo cơ bản như sau:

Action<kiểu\_tham\_số\_1, kiểu\_tham\_số\_2, ... > var\_delegate;

Nghĩa là biến kiểu Action có thể gán bằng các hàm có kiểu trả về void

Trở lại ví dụ cho hai hàm Info và Warning ở trên, có thể sử dụng ngay đoạn code sau, để có kết quả tương tự:

public static void TestAction(string s)

{

Action<string> showLog = null;

showLog += Logs.Warning; // Nối thêm Warning vào delegate

showLog += Logs.Info; // Nối thêm Info vào delegate

showLog += Logs.Warning; // Nối thêm Warning vào delegate

// Một lần gọi thi hành tất cả các phương thức trong chuỗi delegate

showLog("TestLog");

}

Sử dụng Delegate làm tham số hàm

Có thể sử dụng delegate làm tham số của phương thức, nó có vai trò như những hàm callback linh hoạt. Xem ví dụ sau:

// Sử dụng Delegate làm tham số phương thức, truyền callback

static void TinhTong(int a, int b, Action callback)

{

int c = a + b;

// Gọi callback

callback(c.ToString());

}

public static void TestTinhTong()

{

TinhTong(5,6, (x) => Console.WriteLine($"Tổng hai số là: {x}"));

TinhTong(1,3, Logs.Info);

}

## HTTP request methods

GET: được sử dụng để lấy thông tin từ sever theo URI đã cung cấp.  
HEAD: giống với GET nhưng response trả về không có body, chỉ có header  
POST: gửi thông tin tới sever thông qua các biểu mẫu http( đăng kí chả hạn..)-thường dung để tạo mới  
PUT: ghi đè tất cả thông tin của đối tượng với những gì được gửi lên – thường dùng để ghi đè -PUT chính là phương thức bảo toàn dữ liệu nên khi chúng ta ấn submit bao nhiêu lần đi chăng nữa thì dữ liệu cũng chỉ tính như chúng ta submit 1 lần  
PATCH: ghi đè các thông tin được thay đổi của đối tượng.  
DELETE: xóa tài nguyên trên server.  
CONNECT: thiết lập một kết nối tới server theo URI.  
OPTIONS: mô tả các tùy chọn giao tiếp cho resource.  
TRACE: thực hiện một bài test loop - back theo đường dẫn đến resource.

<https://viblo.asia/p/cung-tim-hieu-ve-http-request-methods-djeZ1xBoKWz>

### ActionLink\*

## int?

Trừ kiểu string, tất cả kiểu dữ liệu trên đều không được có giá trị null:

* Null là giá trị rỗng, không tham chiếu đến vùng nhớ nào.
* Để có thể gán giá trị null cho biến thì ta thêm ký tự ? vào sau tên kiểu dữ liệu là được. Ví dụ: int? hay bool? . . .

trả về view là trả về htm. trả về redict… là chạy tiếp vào ActionResult chỉ định

Lỗi đôi khi chỉ đơn giản là sử dụng File script nào trước script nào sau

## Dispose()

protected override void Dispose(bool disposing)

{

    db.Dispose();

    base.Dispose(disposing);

}

Để đảm bảo các kết nối đến CSDL đã được đóng và tài nguyên do chúng chiếm giữ đã được giải phóng, bạn phải đảm bảo đối tượng context phải bị hủy. Đó là vì sao chúng ta sửa lại phương thức Dispose ở cuối lớp StudentController trong file StudentController.cs, như bạn thấy trong ví dụ trên.

Lớp Controller cơ sở đã implement IDisposable, do vậy đoạn code này đơn giản thêm một phương thức override lại Dispose(bool) để thực hiện việc giải phóng đối tượng context.

## context.Entry(entity).State = EntityState.Modified;

Khi bạn làm vậy context.Entry(entity).State = EntityState.Modified;, bạn không chỉ gắn thực thể vào DbContext, mà bạn còn đánh dấu toàn bộ thực thể là bẩn. Điều này có nghĩa là khi bạn thực hiện context.SaveChanges(), EF sẽ tạo một câu lệnh cập nhật sẽ cập nhật ***tất cả*** các trường của thực thể.

Điều này không phải lúc nào cũng mong muốn.

Mặt khác, DbSet.Attach(entity)gắn thực thể vào ngữ cảnh ***mà không*** đánh dấu nó là bẩn. Nó tương đương với làmcontext.Entry(entity).State = EntityState.Unchanged;

Khi đính kèm theo cách này, trừ khi sau đó bạn tiến hành cập nhật thuộc tính trên thực thể, trong lần gọi tiếp theo context.SaveChanges(), EF sẽ không tạo cập nhật cơ sở dữ liệu cho thực thể này.

Ngay cả khi bạn đang lên kế hoạch cập nhật một thực thể, nếu thực thể đó có nhiều thuộc tính (cột db) nhưng bạn chỉ muốn cập nhật một vài thuộc tính, bạn có thể thấy thuận lợi khi thực hiện một DbSet.Attach(entity)và sau đó chỉ cập nhật một vài thuộc tính cần cập nhật. Làm theo cách này sẽ tạo ra một câu lệnh cập nhật hiệu quả hơn từ EF. EF sẽ chỉ cập nhật các thuộc tính mà bạn đã sửa đổi (ngược lại context.Entry(entity).State = EntityState.Modified;sẽ khiến tất cả các thuộc tính / cột được cập nhật)

Tài liệu liên quan: [Thêm / Đính kèm và Thực thể Quốc gia](https://msdn.microsoft.com/en-us/data/jj592676) .

**Mã ví dụ**

Giả sử bạn có thực thể sau:

public class Person

{

public int Id { get; set; } // primary key

public string FirstName { get; set; }

public string LastName { get; set; }

}

Nếu mã của bạn trông như thế này:

context.Entry(personEntity).State = EntityState.Modified;

context.SaveChanges();

SQL được tạo sẽ trông giống như sau:

UPDATE person

SET FirstName = 'whatever first name is',

LastName = 'whatever last name is'

WHERE Id = 123; -- whatever Id is.

Lưu ý cách câu lệnh cập nhật ở trên sẽ cập nhật tất cả các cột, bất kể bạn đã thực sự thay đổi các giá trị hay chưa.

Ngược lại, nếu mã của bạn sử dụng Đính kèm "bình thường" như thế này:

context.People.Attach(personEntity); // State = Unchanged

personEntity.FirstName = "John"; // State = Modified, and only the FirstName property is dirty.

context.SaveChanges();

Sau đó, câu lệnh cập nhật đã tạo khác:

UPDATE person

SET FirstName = 'John'

WHERE Id = 123; -- whatever Id is.

Như bạn có thể thấy, câu lệnh cập nhật ***chỉ*** cập nhật các giá trị đã thực sự thay đổi sau khi bạn gắn thực thể vào ngữ cảnh. Tùy thuộc vào cấu trúc bảng của bạn, điều này có thể có tác động tích cực đến hiệu suất.

## Attach

Update một số trường dữ liệu mà không update tất

Khi bạn sử dụng Attachphương thức trên một thực thể, trạng thái của nó sẽ được đặt thành Unchanged, điều này sẽ dẫn đến không có lệnh cơ sở dữ liệu nào được tạo ra. Tất cả các thực thể có thể truy cập khác với các giá trị chính được xác định cũng sẽ được đặt thành Unchanged. Những người không có giá trị chính sẽ được đánh dấu là Added. Tuy nhiên, bây giờ thực thể đang được theo dõi bởi ngữ cảnh, bạn có thể thông báo cho ngữ cảnh những thuộc tính nào đã được sửa đổi để SQL chính xác để cập nhật chỉ những giá trị đó được tạo ra:

db.Customers.Attach(customer);

db.Entry(customer).Property("name").IsModified = true;

// hoặc gán trực tiếp: customer.name = “123”

db.SaveChanges();

return RedirectToAction("Index");

Đoạn mã trên sẽ dẫn đến thực thể tác giả được đánh dấu là Modifiedvà SQL được tạo để chỉ cập nhật thuộc tính  name:

[Vd khác](#_context.Entry(entity).State_=_Entit)

## DbContext.Configuration.ProxyCreationEnabled

Nếu DbContext.Configuration.ProxyCreationEnabled được đặt thành false, DbContext sẽ không tải các đối tượng con cho một số đối tượng mẹ trừ khi Include phương thức được gọi trên đối tượng mẹ. Đặt DbContext.Configuration.LazyLoadingEnabled thành true hoặc false sẽ không ảnh hưởng đến hành vi của nó.

Nếu DbContext.Configuration.ProxyCreationEnabled được đặt thành true, các đối tượng con sẽ được tải tự động và DbContext.Configuration.LazyLoadingEnabled giá trị sẽ kiểm soát thời điểm các đối tượng con được tải.

Cụ thể:

public partial class FastFood : DbContext

{

public FastFood()

: base("name=FastFood")

{

this.Configuration.ProxyCreationEnabled = false;

}

….

## Eager Loading

Eager loading giúp bạn load tất cả các entity trong 1 câu leenhk, tất cả các entity con sẽ được load ra trong 1 lần gọi duy nhất. Điều này có thể được thực hiện thông qua phương thức Include, sẽ trả về các entity liên quan như một phần của câu query và một lượng lớn dữ liệu sẽ được load ra 1 lần.

Ví dụ, bạn có một bản User và một bảng UserDetails (bảng này liên kết đến bảng User), sau đó bạn sẽ viết code như dưới đây. Chúng ta sẽ load ra user với Id của nó bằng UserId ở bảng UserDetails.

User usr = dbContext.Users.Include(a => a.UserDetails).FirstOrDefault(a => a.UserId == userId);

Nếu có nhiều cấp độ hơn chúng ta có thể sử dụng câu lệnh sau:

User usr = dbContext.Users.Include(a => a.UserDetails.Select(ud => ud.Address)).FirstOrDefault(a => a.UserId == userId);

## Lazy Loading

Đây là một hành vi mặc định của Entity Framework, khi mà các entity con được load ra chỉ khi chúng được truy cập lần đầu tiên. Đơn giản là hoãn lại việc load các dữ liệu ở các enttiy liên quan cho đến khi bạn yêu cầu nó.

Ví dụ, khi chúng ta chạy câu lện bên dưới, bảng UserDetails sẽ không được tải ra theo bảng User:

User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);

Nó sẽ chỉ được load ra khi bạn công khai gọi nó:

UserDeatils ud = usr.UserDetails; // UserDetails are loaded here

Xét thêm 1 ví dụ nữa:

**public** **class** **Person**{

**public** String Name{**get**; **set**;}

**public** String Email {**get**; **set**;}

**public** **virtual** Employer employer {**get**; **set**;}

}

**public** List<EF.Person> GetPerson(){

**using**(EF.DbEntities db = **new** EF.DbEntities()){

**return** db.Person.ToList();

}

}

Sau khi phương thức GetPerson() được gọi, bạn không thể dùng lazyloading để truy cập vào thuộc tính employer để lấy ra đối tượng Employer khi ra ngoài phương thức nữa. Vì sao? Vì đối tượng db đã bị hủy sau khi ra ngoài khỏi khối using. Vây bạn phải dùng Person.Include(x=>x.employer) để load sẵn dữ liệu của Employer vào thuộc tính của person trước khi đối tượng db bị hủy.

## Explicit Loading

Có nhiều tùy chọn để disable tính năng Lazyloading trong Entity Framework. Sau khi tắt Lazy Loading, bạn có thể vẫn load các enttiy liên quan bằng cách gọi tường minh phương thức Load() để tải các entity liên quan. Có 2 cách sử dụng phương thức Load, Reference và Collection (để load collections):

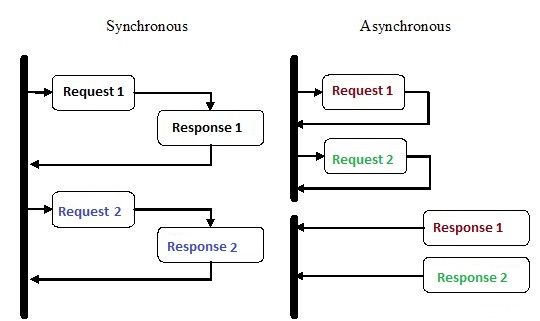
User usr = dbContext.Users.FirstOrDefault(a => a.UserId == userId);

dbContext.Entry(usr).Reference(usr => usr.UserDetails).Load();

Khi nào sử dụng cái gì:

1. Sử dụng Eager loading khi các mối quan hệ không quá nhiều. Vì thế Eager loading là cách tối để giảm lượng query đến server vì chỉ có 1 query duy nhất.
2. Sử dụng Eager loading khi bạn chắc chắn bạn sẽ sử dụng các entity liên quan với entity chính ở nhiều nơi và nhiều lần.
3. Sử dụng Lazyloading khi bạn sử dụng qua nhệ 1-N
4. Sử dụng Lazyloading khi bạn chắc chắn bạn không sử dụng các entity liên quan ngay lập tức.
5. Khi bạn tắt tính năng Lazyloading, sử dụng Explicit loading khi bạn không chắc rằng bạn sẽ có sử dụng entity trước đó hay không.

## Asynchronous trong C#



### TQ: đây chính là đa luồng

url: <https://viblo.asia/p/lap-trinh-bat-dong-bo-trong-c-DZrGNDoWkVB>

Trong rất nhiều ngôn ngữ lập trình hiện nay, việc hỗ trợ lập trình bất đồng bộ(Asynchronous programing) đã trở nên khá phổ biến. Ví dụ thường gặp nhất là việc giao tiếp với server thông qua Ajax của javascript. Như vậy có thể hiểu đơn giản lập trình bất đồng bộ (Asynchronous) là khả năng thực thi các tác vụ độc lập nhau, có nghĩa là chúng không nhất thiết phải chạy một cách tuần tự (có thể chạy song song với nhau) hoặc xử lý này không phải đợi xử lý khác giúp cho việc cải thiện hiệu suất hoạt động của ứng dụng. Trong C# bắt đầu thực thi từ phương thức Main và kết thúc khi phương thức Main được trả về. Trong đó tất cả các xử lý thực hiện một cách tuần tự hết cái này đến cái khác. Một xử lý phải đợi xử lý trước đó hoàn thành.

static void Main(string[] args)

{

DoTaskOne();

DoTaskTwo();

}

Ta có thể thấy. Với cách lập trình đồng bộ thì "DoTaskTwo" không thể bất đầu cho đến khi "DoTaskOne" hoàn thành. Trong lập trình bất đồng bộ phương thức được gọi sẽ được chạy trong background và việc gọi thread là không bị block. Sau khi gọi phương thức thực thi, luồng trở lại gọi và thực thi những tác vụ khác. Thông thường chúng sử dụng Thread hoặc Task.

Trong trường hợp của chúng ta, nếu chúng ta chạy "DoTaskOne" theo cách bất đồng bộ thì sau khi gọi "DoTaskOne" luồng thực thi sẽ ngay lập tức quay lại phương thức Main và bất đầu "DoTaskTwo"

Chúng ta có thể tạo thread bằng việc sử dụng Thread class hoặc sử dụng asynchronous patterns được cung cấp bởi .Net để thực hiện lập trình bất đồng bộ

**2. Những điểm nổi bật của Asynchronous trong C#**

**2.1 Cải thiện hiệu suất ứng dụng**

Lập trình bất đồng bộ thường được sử dụng trong các xử lý tiềm ẩn blocking, giống như truy cập tới website, việc truy cập tài nguyên trên web đôi lúc là chậm trễ. Ngoài ra có một số xử lý khác nên được xây dựng bất động bộ như: Làm việc trên file, ảnh,..

Từ .Net Framework 4.5 và Windows Runtime sẽ có một số phương thức trong các APIs hỗ trợ bất đồng bộ như ở dưới:

* Web access: **HttpClient, SyndicationClient**
* Working with files: **StorageFile, StreamWriter, StreamReader, XmlReader**
* Working with images: **MediaCapture, BitmapEncoder, BitmapDecoder**
* WCF programming: **Synchronous and Asynchronous Operations**

**2.2 Phương thức bất đồng bộ trong C# dễ dàng để viết**

Chúng ta có thể thông nhất rằng lập trình bất đồng bộ là rất tốt để cải thiện performance cho ứng dụng chúng ta. Tuy nhiên với công nghệ trước đây, việc viết ứng dụng bất đồng bộ là khá phức tạp, khó khăn để viết, debug và bảo trì.

Do vậy, .Net Framework 4.5 ra đời có một cách tiếp cập đơn giản để thúc đẩy lập trình bất đồng bộ trở nên dễ dàng hơn bao giờ hết. Những phần việc khó khăn nhất mà trước kia lập trình viên từng làm sẽ được thực hiện bởi compiler. Như vậy cấu trúc logic của ứng dụng sẽ vẫn giữ như lập trình đồng bộ thông thường.

Trong C#, hai từ khóa asyn và await là trung tâm của lập trình bất đồng bộ. Bằng việc sử dụng hai từ khóa này, bạn có thể sử dụng tài nguyên trong .Net hoặc Windows Runtime để tạo một phương thức bất đồng bộ một cách dễ dàng. Chúng ta cùng theo dõi ví dụ bên dưới:

async Task<int> AccessTheWebAsync()

{

HttpClient client = new HttpClient();

// GetStringAsync returns a Task<string>. That means that when you await the

Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");

// You can do work here that doesn't rely on the string from GetStringAsync.

DoIndependentWork();

// - The await operator then retrieves the string result from getStringTask.

string urlContents = await getStringTask;

// The return statement specifies an integer result.

// Any methods that are awaiting AccessTheWebAsync retrieve the length value.

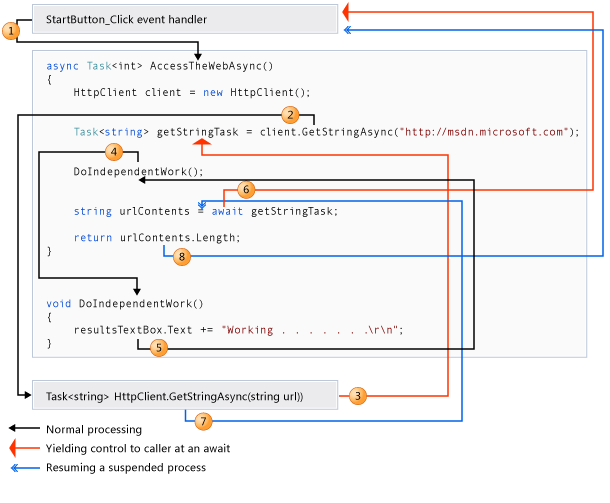
return urlContents.Length;

}

Gọi phương thức trên:

string urlContents = await client.GetStringAsync();

Hình ảnh bên dưới là luồng thực thi của phương thức 'GetStringAsync':



**2.3 Awaiting nhiều phương thức bất đồng bộ**

Trước tiên, bạn hãy quan sát code sau:

private async void CallWithAsync()

{

string result = await DoTask1Async("Task1");

string result1 = await DoTask2Async("Task2");

Console.WriteLine(result);

Console.WriteLine(result1);

}

Ở trên, chúng ta đợi hai phương thức bất đồng bộ một cách tuần tự. Việc gọi hàm thứ 2 sẽ bắt đầu sau khi công việc hàm đầu hoàn thành. Đây không phải là ý tưởng tốt trong thực tế nếu chúng không phụ thuộc vào kết quả của nhau bởi vì "DoTask1Async" có thể block chương trình. Để giải quyết vấn đề này trong C# chúng ta có thể sử dụng Task.WhenAll

private async static void MultipleAsyncMethodsWithCombinators()

{

Task<string> t1 = DoTask1Async("Task1");

Task<string> t2 = DoTask2Async("Task2");

await Task.WhenAll(t1, t2);

Console.WriteLine("Finished both methods.\n " +

"Result 1: {0}\n Result 2: {1}", t1.Result, t2.Result);

}

Với code trên cả hai phương thức trên được thực thi một cách song song, không block lẫn nhau.

**2.4 Canceling tác vụ**

Trước đây, nếu chúng ta sử dụng Thread thì việc hủy tác vụ của nó là không thể. Từ Net 4.0 trở lên C# cung cấp một cách để hủy các Task đang thực thi dựa trên CancellationTokenSource class

static void Main(string[] args)

{

CallWithAsync();

Console.ReadKey();

}

async static void CallWithAsync()

{

try

{

CancellationTokenSource source = new CancellationTokenSource();

source.CancelAfter(TimeSpan.FromSeconds(1));

var t1 = await GreetingAsync("HiepHV", source.Token);

}

catch (OperationCanceledException ex)

{

Console.WriteLine(ex.Message);

}

}

static Task<string> GreetingAsync(string name, CancellationToken token)

{

return Task.Run<string>(() =>

{

return Greeting(name, token);

});

}

static string Greeting(string name, CancellationToken token)

{

Thread.Sleep(3000);

token.ThrowIfCancellationRequested();

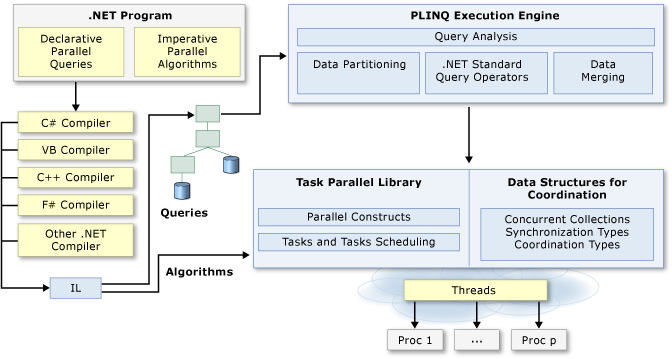
return string.Format("Hello, {0}", name);

}

Đây là một tính năng rất hay trong lập trình bất động của .Net. Trong ứng dụng thực thế, có những xử lý như đang upload file, xử lý document,.. và các xử lý bất động khác đôi khi rất khó để hủy tác vụ của nó khi đang thực hiện. CancellationTokenSource class có thể giúp chúng ta control điều này dễ dàng hơn.

**2.5 Lập trình song song**

Rất nhiều máy tính cá nhân và các máy trạm có hai hoặc bốn nhân của CPU, điều đó cho phép nhiều luồng xử lý thực hiện đồng thời. Trong tương lai gần, phần cứng máy tính sẽ ngày càng được nâng cấp(có nhiều nhân CPU hơn,..). Vì vậy để tận dụng những lợi thế của phần cứng hiện nay và sau này, bạn có thể làm song song hóa các tác vụ trong code thông qua nhiều trình xử lý. Trong quá khứ, lập trình song song hay còn gọi là đa luồng yêu cầu một sự nắm bắt ở mức low-level của thread và locks. VS 2010 và .Net 4.0 ra đời đã mở rộng hỗ trợ lập trình song song bằng việc cung cấp new runtime, new class library types và new diagnostic tools. Những tính năng này nhằm giúp đơn giản hóa việc phát triển lập trình song song. Hình ảnh bên dưới sẽ minh họa kiến trúc 'parallel programing' trong .Net 4.0:



Example code:

ParallelLoopResult result =

Parallel.For(0, 100, async (int i) =>

{

Console.WriteLine("{0}, task: {1}, thread: {2}", i,

Task.CurrentId, Thread.CurrentThread.ManagedThreadId);

await Task.Delay(10);

});

**3. Asynchronous và threading**

Trước khi Asynchronous programing ra đời ở những phiên bản cũ hơn của .Net, chúng ta có công nghệ lập trình đa luồng(multiple thread). Theo hiểu biết của tôi, với mỗi thread chúng ta tạo ra trong code sẽ cần một thread tương ứng của CPU. Nó cũng thể coi là một dạng lập trình bất đồng bộ hay có thể gọi là một bộ phận của Asynchronous programing hiện nay. Với công nghệ Asynchronous trong .Net hiện nay, nó có thể chạy nhiều taks trong cùng một thread hoặc nhiều thread khác nhau tùy tình huống cự thể. Rõ ràng đây là một ưu điểm lớn giúp cải thiện hiệu suất tốt hơn cho ứng dụng so với sử 'dụng Multiple Thread'

**4. Kết**

Hầu hết các lập trình viên hiện nay đều khá quen thuộc với lập trình bất đồng bộ trong các ngôn ngữ khác nhau như: javascript, java, python,ruby..Trong bài viết này tôi muốn trình bày những hiểu biết của mình về lập trình bất đồng bộ trong C# và .Net. Hy vọng sẽ góp một góc nhìn có ích cho các bạn.

### async

là từ khóa được đặt trước một method để chỉ định với compiler rằng method này được chạy bất đồng bộ, ở bên trong phần cài đặt nó sẽ được phép dùng từ khóa **await**.

### await

là từ khóa được đặt trước một lời gọi hàm nhằm chỉ định với compiler rằng phải chờ cho đến khi hàm được await thực hiện xong mới tiếp tục chạy các chỉ thị còn lại.

### VD:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | class Program      {          private static Random \_rnd;          static void Main(string[] args)          {              \_rnd = new Random();                // cho phép tính 10 + 10              // thằng bé A bắt đầu tính              Run("A", SumAsync(10, 10));                // thằng bé B bắt đầu tính              Run("B", SumAsync(10, 10));                // thằng bé đếm thời gian              for (int i = 0; i < 60; i++)              {                  Thread.Sleep(50);                  Console.WriteLine(i);              }                Console.Read();          }            private static async void Run(string name ,Task<int> task)          {              var result = await task;              Console.WriteLine(name + " has got the answer =" + result);          }            private static Task<int> SumAsync(int a, int b)          {              return Task.Factory.StartNew(() => Sum(a, b));          }            private static int Sum(int a, int b)          {              var calculatingTime = \_rnd.Next(3000);              Thread.Sleep(calculatingTime);                return a + b;          }      } |

### Task.WhenAll

Vd:

class Program

{

private static Random \_rnd;

static void Main(string[] args)

{

Program p = new Program();

Task t1 = Task.Factory.StartNew(() => p.SyncFuncA());

Task t2 = Task.Factory.StartNew(() => p.SyncFuncB());

Task.WhenAll(new[] { t1, t2 });

Console.Read();

}

private void SyncFuncA()

{

for(int i= 1; i<10; i++)

{

Console.WriteLine("sync A: " + i);

Thread.Sleep(1000);

}

}

private void SyncFuncB()

{

for (int i = 1; i < 10; i++)

{

Console.WriteLine("sync B: " + i);

Thread.Sleep(3000);

}

}

}

Hoặc bạn cũng có thể thay bằng hàm async

private async Task SyncFuncA()

{

for(int i= 1; i<10; i++)

{

Console.WriteLine("sync A: " + i);

Thread.Sleep(1000);

}

}

private async Task SyncFuncB()

{

for (int i = 1; i < 10; i++)

{

Console.WriteLine("sync B: " + i);

Thread.Sleep(3000);

}

}

### So sánh Task.Delay và Task.Sleep

Hãy thử chuyển vd trên tành await Task.Delay()

* Delay được khuyến cáo sử dụng với bất đồng bộ và không có ý nghĩa khi chạy đồng bộ
* Sleep thực sự dừng 1 luồng đang thực thi.  Tuy nhiên, do mã không đồng bộ thường chạy trên threadpool, nên việc chặn một luồng không cần thiết Thread.Sleep()sẽ tiêu tốn toàn bộ luồng có thể được sử dụng ở nơi khác. Nếu nhiều tác vụ được chạy với Thread.Sleep (), có khả năng cao sẽ làm cạn kiệt tất cả các luồng threadpool và cản trở hiệu suất nghiêm trọng.
* Không dùng Sleep cho luồng bất đồng bộ

### So sánh Task.run và Task.Factory.StartNew

2 cái là như nhau về chức năng

* Task.run được khuyến cáo sử dụng cho những phiên bản sau này nó gọn gàng hơn
* Task.Factory.StartNew lắm tùy chọn hơn
* nói chung nếu bạn không sử dụng tác vụ con lồng nhau và luôn muốn các tác vụ của mình được thực thi trên Thread Pool thì tốt hơn nên sử dụng Task.Run, trừ khi bạn có một số tình huống phức tạp hơn.d

# Tạo server Web

## Public web

<https://www.youtube.com/watch?v=xmyv6xqNMutro>

## Cài đặt IIS

<https://www.youtube.com/watch?v=GA6HoPx28pc>

lưu ý: phần host không cần cài;

ip address nên để là ip của máy mình: 192.168.1.43

## Set identity

Trong ISS: Application -> Name của web( chuột phải) - > Advanced Settings -> Identity: chọn LocalSystem.

## Thêm quyền cho NT AUTHORITY\SYSTEM

Trong sql management: Security ->Login -> NT AUTHORITY\SYSTEM -> Properties -> Server Roles: tích sysadmin

# WEB API

Form

Nhiệm vụ đầu tiên của bạn là thiết lập biểu mẫu HTML sẽ thu thập dữ liệu từ người dùng. Cung cấp <form>ID cho phần tử của bạn ajax-contact, đặt methodthuộc tính thành postvà actionthuộc tính thành mailer.php.

<form id="ajax-contact" method="post" action="mailer.php">

<div class="field">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required>

</div>

<div class="field">

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

</div>

<div class="field">

<label for="message">Message:</label>

<textarea id="message" name="message" required></textarea>

</div>

<div class="field">

<button type="submit">Send</button>

</div>

</form>

Tại đây, bạn đã thiết lập một biểu mẫu đơn giản có các trường để thu thập tên, địa chỉ email và tin nhắn của người dùng. Lưu ý rằng mỗi trường biểu mẫu có một requiredthuộc tính. Trong các trình duyệt hỗ trợ xác thực biểu mẫu HTML5, điều này sẽ ngăn biểu mẫu được gửi cho đến khi hoàn thành tất cả các trường.

Tiếp theo, bạn cần tạo một <div>phần tử sẽ được sử dụng để hiển thị thông báo thành công và lỗi cho người dùng. Tạo phần tử này phía trên phần <form>đánh dấu HTML của bạn và cung cấp cho nó ID form-messages.

<div id="form-messages"></div>

Bây giờ bạn cần [tải xuống các tài nguyên mã](http://cl.ly/290k070a1m1Z) và sao chép style.csstệp vào thư mục dự án của bạn. Bạn cũng sẽ cần thêm một <link>phần tử yêu cầu trình duyệt tải biểu định kiểu.

<link rel="stylesheet" href="style.css">

Cuối cùng, bạn cần tạo hai <script>phần tử tham chiếu đến tệp jquery-2.1.0.min.jsvà app.js. Thêm chúng ngay trước </body>thẻ đóng .

<script src="jquery-2.1.0.min.js"></script>

<script src="app.js"></script>

Điều quan trọng là bạn phải tải jquery-2.1.0.min.jstệp trước vì mã bên trong app.jsyêu cầu jQuery.

Đó là tất cả HTML mà bạn sẽ cần trong bài đăng này. Tiếp theo, chúng ta hãy xem xét JavaScript.

Gửi biểu mẫu bằng AJAX

Tạo một tệp mới trong thư mục dự án của bạn có tên app.js. Điều này sẽ chứa tất cả mã chịu trách nhiệm gửi dữ liệu biểu mẫu bằng AJAX.

Sao chép mã sau vào app.jstệp.

$(function() {

// Get the form.

var form = $('#ajax-contact');

// Get the messages div.

var formMessages = $('#form-messages');

// TODO: The rest of the code will go here...

});

Tại đây, bạn đã tạo hai biến mới formvà formMessagestham chiếu đến các phần tử tương ứng trong đánh dấu HTML của bạn.

Tiếp theo, bạn cần tạo một trình nghe sự kiện sẽ chặn submitcác sự kiện trên biểu mẫu. Bạn có thể thực hiện việc này bằng submitphương thức jQuery .

// Set up an event listener for the contact form.

$(form).submit(function(event) {

// Stop the browser from submitting the form.

event.preventDefault(); Ngăn nút gửi gửi biểu mẫu

// TODO

});

Ở đây bạn đã chuyển một hàm cho submitphương thức sẽ được thực thi khi người dùng gửi biểu mẫu liên hệ. Bạn cũng đã yêu cầu trình duyệt không gửi biểu mẫu như bình thường bằng cách gọi preventDefaultphương thức trên sự kiện.

Tiếp theo, bạn cần tuần tự hóa dữ liệu biểu mẫu. Thao tác này sẽ chuyển đổi dữ liệu người dùng đã nhập vào biểu mẫu thành một chuỗi khóa / giá trị có thể được gửi cùng với yêu cầu AJAX. Sử dụng serializephương thức jQuery để tuần tự hóa dữ liệu biểu mẫu và sau đó lưu trữ kết quả trong một biến được gọi formData.

// Serialize the form data.

var formData = $(form).serialize();

Bây giờ bạn đã sẵn sàng để viết mã chịu trách nhiệm gửi dữ liệu biểu mẫu đến máy chủ và xử lý phản hồi. Sao chép mã sau vào app.jstệp của bạn .

// Submit the form using AJAX.

$.ajax({

type: 'POST',

url: $(form).attr('action'),

data: formData

})

Ở đây bạn đang sử dụng ajaxphương thức của jQuery để tạo một yêu cầu AJAX mới. Bạn đã chuyển một đối tượng cho ajaxphương thức có chứa một số thuộc tính được sử dụng để định cấu hình yêu cầu. Các typebất động sản quy định các phương thức HTTP sẽ được sử dụng cho các yêu cầu, trong trường hợp của chúng tôi POST. Các urlbất động sản là vị trí của kịch bản mà các dữ liệu mẫu sẽ được gửi đến. Bạn đã điền actionthuộc tính này bằng cách tìm nạp thuộc tính từ biểu mẫu. Cuối cùng, thuộc datatính đã được điền bằng cách sử dụng formDatabiến mà bạn đã tạo trước đó.

Tiếp theo, bạn cần xử lý phản hồi thành công từ máy chủ. Sao chép mã sau ngay sau dấu ngoặc đóng của ajaxcuộc gọi. Lưu ý rằng tôi đã cố tình bỏ dấu chấm phẩy.

.done(function(response) {

// Make sure that the formMessages div has the 'success' class.

$(formMessages).removeClass('error');

$(formMessages).addClass('success');

// Set the message text.

$(formMessages).text(response);

// Clear the form.

$('#name').val('');

$('#email').val('');

$('#message').val('');

})

Đây donephương pháp sẽ được gọi nếu yêu cầu hoàn tất thành công. Ở đây, trước tiên bạn phải đảm bảo rằng formMessagesphần tử có successlớp và sau đó đặt nội dung văn bản của phần tử này bằng cách sử dụng dữ liệu được trả về bởi tập lệnh thư. Để kết thúc, bạn xóa các giá trị khỏi mỗi trường biểu mẫu.

Bit JavaScript cuối cùng bạn cần viết xử lý những gì sẽ xảy ra nếu có lỗi xảy ra. Sao chép phần sau vào app.js.

.fail(function(data) {

// Make sure that the formMessages div has the 'error' class.

$(formMessages).removeClass('success');

$(formMessages).addClass('error');

// Set the message text.

if (data.responseText !== '') {

$(formMessages).text(data.responseText);

} else {

$(formMessages).text('Oops! An error occured and your message could not be sent.');

}

});

Đây failphương pháp được gọi là nếu kịch bản bưu phẩm trả về một lỗi. Ở đây, trước tiên bạn phải đảm bảo rằng formMessagesphần tử có errorlớp. Sau đó, bạn kiểm tra xem liệu yêu cầu AJAX có trả lại bất kỳ yêu cầu nào không responseText. Nếu có, bạn sử dụng văn bản này để đặt nội dung cho formMessagesphần tử; nếu không sử dụng một thông báo lỗi chung.

Điều đó hoàn thành mã HTML và JavaScript cần thiết để tạo biểu mẫu liên hệ AJAX. Trong phần tiếp theo, bạn sẽ tìm hiểu về tập lệnh bưu phẩm chịu trách nhiệm xử lý dữ liệu biểu mẫu và gửi email.

## Khắc phục: Duy trì tham chiếu vòng tròn trên toàn cầu

Cách sửa thứ hai này tương tự như lần đầu tiên.trong hàm Register của file WebApiConfig Chỉ cần thay đổi mã thành:

config.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling

= Newtonsoft.Json.ReferenceLoopHandling.Serialize;

config.Formatters.JsonFormatter.SerializerSettings.PreserveReferencesHandling

= Newtonsoft.Json.PreserveReferencesHandling.Objects;

Hình dạng dữ liệu sẽ được thay đổi sau khi áp dụng cài đặt này.

[

{

"$id":"1",

"Category":{

"$id":"2",

"Products":[

{

"$id":"3",

"Category":{

"$ref":"2"

},

"Id":2,

"Name":"Yogurt"

},

{

"$ref":"1"

}

],

"Id":1,

"Name":"Diary"

},

"Id":1,

"Name":"Whole Milk"

},

{

"$ref":"3"

}

]

$ Id và $ ref giữ tất cả các tham chiếu và làm phẳng mức đồ thị đối tượng, nhưng mã máy khách cần biết sự thay đổi hình dạng để sử dụng dữ liệu và nó cũng chỉ áp dụng cho bộ tuần tự JSON.NET.

## Gọi api

Vd với hàm: public async Task<IHttpActionResult> GetCustomer(int name){…}

ta sẽ gọi <https://localhost:44399/api/Customers?name=1>

với cấu hình router như sau

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

* Có thể thêm trước hàm GetCustomer
  + [HttpGet] cho phép đặt tên hàm có thể bỏ Get ở đầu mà chương trình vẫn hiểu hành động là Get. Tương tự với Post, Put, Delete,..(không liên quan đến parameter)
  + [Route("Customers")]. Thay cho routeTemplate mặc đinh api/Customers?id=1

Thành Customers?id=1 (không liên quan đến parameter)

Cũng có thể cấu hình tương tự như sau: [Route("Categories/{id}")]

-> url: …/Categories/1

Gọi nhiều parameter bằng cách [Route("Categories/{id}&{name}")]

-> url: …/Categories/1&hieu

Vd: [Route("Categories/{id}")]

[ResponseType(typeof(Category))]

public async Task<IHttpActionResult> GetCategory(int id, string test)

{..}

Khi gọi: https://localhost:44399/Categories/5?test=hi

Vậy với điều kiện gọi api là theo routeTemplate mặc định với parameters khi gọi phải trùng tên với tất cả parameter của hàm api (không được thiếu). Không quan tâm thứ tự giữa các parameter

Lưu ý . có thể gọi thừa parameter thì chương trình vẫn nhận và chọn ra api trùng tên và có nhiều parameter trùng nhất nhưng trả ai lại đi gọi thừa  và cũng không đúng trong [.net core](#_.NET_CORE)

Mẹo: có thể vừa gọi parameter dạng [FromUri] và dạng [FromBody] bằng các gán parameter dạng [FromUri] vào phần khai báo router trước hàm api như vd dưới

[Route("api/CustomerVD/{status}")]

[HttpPost]

public Customer PostCustomerVD([FromUri]int status, [FromBody]Customer customer)

{

return customer;

}

### trong ajax

**data: {"id": id}**, truyền parameter được gắn trực tiếp vào url : với PUT, POST hoặc GET, nó hoạt động rất tốt.

* với delete chỉ có cách gán vào url dưới dạng text:

VD:

url: 'https://localhost:44399/Customer/' + id, <=> [Route("Customer/{id}")]

hay url: 'https://localhost:44399/Customer?id=' + id <=> [Route("Customer")]

## Phân trang trong API

Chỉ cần cài đặt gói PagedList thôi

## Load list data into datatable

Vd:

$("#example1").DataTable({

"processing": true,

"ajax" : {

"url" : "https://localhost:44399/api/Customers",

"dataType" : "JSON",

"type": "GET",

"dataSrc": ""

},

"columns": [

{ "data": "id\_customer" },

{ "data": "name" },

{ "data": "phone" },

{ "data": "address" },

{ "data": "userName" },

{

data: 'avatar',

render: function (data, type, row, meta){

return '<img height="20" src="https://localhost:44399/Areas/Admin/Content/Photos/' + data + '" alt="Chưa có" />';

}

}

],

"paging": false,

"lengthChange": false,

"searching": false,

"ordering": true,

"info": false,

"autoWidth": false,

"responsive": false

});

### [- Định dạng lại hiển thị với render](https://datatables.net/examples/basic_init/data_rendering.html)

Thêm định dạng

Trong DataTable của chúng tôi, nếu chúng tôi muốn có một cột hiển thị giá, thì việc đặt trước nó bằng một ký hiệu tiền tệ là điều tương đối phổ biến. Trong trường hợp này, chúng tôi sử dụng ký hiệu đô la (xem thêm trình kết xuất số tích hợp bên dưới cung cấp các tùy chọn định dạng nâng cao):

[Javascript](https://datatables.net/manual/data/renderers)

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | {      data: 'price',      render: function ( data, type, row ) {          return '$'+ data;      }  }  Khác :  Như vd trên {  data: 'avatar',  render: function (data, type, row, meta){  return '<img height="20" src="https://localhost:44399/Areas/Admin/Content/Photos/' + data + '" alt="Chưa có" />';  }  Vd: định dạng tiền :   |  | | --- | | $ 162.700,00 |   {                  data: "salary",                  render: $.fn.dataTable.render.number( ',', '.', 2, '$' )              } |

Nối chuỗi

Trong DataTable của chúng tôi, nếu chúng tôi muốn có một cột duy nhất hiển thị tên đầy đủ của người tạo, chúng tôi có thể nối các chuỗi bằng cách sử dụng [columns](https://datatables.net/reference/option/columns)định nghĩa sau (đặc biệt lưu ý cách createđối tượng được truyền vào làm tham số đầu tiên do việc gán nó bằng [columns.data](https://datatables.net/reference/option/columns.data)tùy chọn):

lưu ý 'creator' là 1 đối tượng có thuộc tính firstName, lastName

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {      data: 'creator',      render: function ( data, type, row ) {          return data.firstName +' '+ data.lastName;      }  } |

Chuyển đổi dữ liệu

Đối với một cột khác, chúng tôi muốn hiển thị createdgiá trị nhưng được định dạng bằng định dạng MM-DD-YYYY tiêu chuẩn của Hoa Kỳ. Điều này có thể được thực hiện đơn giản bằng cách tách chuỗi và sắp xếp lại các bộ phận thành phần. Chúng tôi cũng muốn ngày có thể sắp xếp được và vì DataTables đã được tích hợp hỗ trợ cho chuỗi có định dạng ISO8601 (định dạng ban đầu), chúng tôi muốn chỉ thực hiện chuyển đổi cho kiểu displayvà filterdữ liệu - xem [dữ liệu trực giao](https://datatables.net/manual/data/orthogonal-data) để biết thêm):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | {      data: 'created',      render: function ( data, type, row ) {          var dateSplit = data.split('-');          return type === "display" || type === "filter" ?              dateSplit[1] +'-'+ dateSplit[2] +'-'+ dateSplit[0] :              data;      }  } |

Tính toán các giá trị

Cuối cùng, để tạo cột lề từ các trường pricevà, costchúng ta có thể sử dụng một hàm để tính toán các giá trị bắt buộc - lưu ý rằng trong trường hợp [columns.data](https://datatables.net/reference/option/columns.data)này là null- kết quả là tham số đầu tiên được truyền vào [columns.render](https://datatables.net/reference/option/columns.render)phương thức cũng vậy null, nhưng tham số thứ ba cung cấp quyền truy cập vào đối tượng nguồn dữ liệu ban đầu, vì vậy chúng tôi có thể tiếp tục sử dụng dữ liệu từ đó:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {      data: null,      render: function ( data, type, row ) {          return Math.round( ( row.price - row.cost ) / row.price \* 100 )+'%';      }  } |

#### Dây

Một tùy chọn ít phổ biến hơn cho các trình định dạng là như một chuỗi để chỉ vào dữ liệu sẽ được sử dụng trong bảng. Điều này tương tự như cách [columns.data](https://datatables.net/reference/option/columns.data)thường được sử dụng, mặc dù hãy nhớ rằng trình kết xuất sẽ chỉ có quyền truy cập vào dữ liệu được trỏ tới [columns.data](https://datatables.net/reference/option/columns.data)chứ không phải toàn bộ hàng.

Tiếp tục các ví dụ sử dụng cấu trúc dữ liệu JSON ở trên, hãy xem xét một cột sẽ hiển thị tên của người tạo:

|  |  |
| --- | --- |
| 1  2  3  4 | {      data: 'creator',      render: 'firstName'  } |

Phương pháp này không có lợi thế hơn so với việc sử dụng đơn giản data: 'creator.firstName'trong ví dụ được trình bày ở đây, nhưng nếu bạn có dữ liệu phức tạp với dữ liệu trực giao được bao gồm trong đối tượng nguồn dữ liệu, điều này đôi khi có thể hữu ích.

### “dataSrc”

Dưới dạng một chuỗi (ví dụ dataSrc: 'myData') - lấy dữ liệu từ một thuộc tính khác trong đối tượng nguồn.

Vd:

|  |
| --- |
| {<font></font>          "client\_id": 11,<font></font>      "from\_date": "2019-03-01",<font></font>      "to\_date": "2019-03-08",<font></font>      "data": [{<font></font>              "id": "1",<font></font>              "company\_name": "ABC",<font></font>              "num\_of\_people": 3,<font></font>              "num\_of\_page\_visits": 4,<font></font>              "time\_on\_site\_in\_secs": 62,<font></font>              "last\_visited\_datetime": "2018-12-06 13:00:00+00:00"<font></font>          }, {<font></font>              "id": "2",<font></font>              "company\_name": "XYZ",<font></font>              "num\_of\_people": 1,<font></font>              "num\_of\_page\_visits": 2,<font></font>              "time\_on\_site\_in\_secs": 42,<font></font>              "last\_visited\_datetime": "2018-12-04 14:00:00+00:00"<font></font>          }, {<font></font>              "id": "3",<font></font>              "company\_name": "PQR",<font></font>              "num\_of\_people": 1,<font></font>              "num\_of\_page\_visits": 1,<font></font>              "time\_on\_site\_in\_secs": 151,<font></font>              "last\_visited\_datetime": "2018-12-03 12:04:45+01:00"<font></font>          }<font></font>      ]<font></font>  }<font></font> |

Tôi chỉ muốn lấy nội dung datatrong bảng:

|  |  |
| --- | --- |
| 1  2  3  4 | ajax: {<font></font>     url: "/api\_endpoint/",<font></font>     dataSrc: "data"<font></font>  },<font></font> |

* Là một chuỗi rỗng (ví dụ dataSrc: '') - nguồn dữ liệu không phải là một đối tượng mà là một mảng.
* Là một hàm (ví dụ dataSrc: function(json) {}) - một hàm có thể được sử dụng để chuyển đổi dữ liệu từ định dạng nguồn này sang định dạng nguồn khác (ví dụ: bạn có thể chuyển đổi từ XML sang đối tượng Javascript). Giá trị trả về từ hàm được sử dụng làm dữ liệu cho bảng.

### Load data phức tạp

Vd : json

{

  "data": [

    {

      "name": "Tiger Nixon",

      "hr": {

        "position": "System Architect",

        "salary": "$320,800",

        "start\_date": "2011/04/25"

      },

      "contact": [

        "Edinburgh",

        "5421"

      ]

    }

- javascrip:

(document).ready(function() {

    $('#example').DataTable( {

        "processing": true,

        "ajax": "data/objects\_deep.txt",

        "columns": [

            { "data": "name" },

            { "data": "hr.position" },

            { "data": "contact.0" },

            { "data": "contact.1" },

            { "data": "hr.start\_date" },

            { "data": "hr.salary" }

        ]

    } );

} );

Với những trường hợp thuộc tính con là 1 list: nhớ kiểm tra null trước

data: 'Bills.0.id\_bill', // Bills[0].id\_bill

render: function (data, type, row) {

if (data == null) {

return "";

}

return data.toString();

}

## Index Row

Show  entries

Search:

|  | **Name** | **Position** | **Office** | **Age** | **Salary** |
| --- | --- | --- | --- | --- | --- |
| 1 | Airi Satou | Accountant | Tokyo | 33 | $162,700 |
| 2 | Angelica Ramos | Chief Executive Officer (CEO) | London | 47 | $1,200,000 |
| 3 | Ashton Cox | Junior Technical Author | San Francisco | 66 | $86,000 |
| 4 | Bradley Greer | Software Engineer | London | 41 | $132,000 |
| 5 | Brenden Wagner | Software Engineer | San Francisco | 28 | $206,850 |
| 6 | Brielle Williamson | Integration Specialist | New York | 61 | $372,000 |
| 7 | Bruno Nash | Software Engineer | London | 38 | $163,500 |
| 8 | Caesar Vance | Pre-Sales Support | New York | 21 | $106,450 |
| 9 | Cara Stevens | Sales Assistant | New York | 46 | $145,600 |
| 10 | Cedric Kelly | Senior Javascript Developer | Edinburgh | 22 | $433,060 |
|  | **Name** | **Position** | **Office** | **Age** | **Salary** |

Showing 1 to 10 of 57 entries

Previous123456Next

The Javascript shown below is used to initialise the table shown in this example:

[Javascript](https://datatables.net/examples/api/counter_columns.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | $(document).ready(function() {      var t = $('#example').DataTable( {          "columnDefs": [ {              "searchable": false,              "orderable": false,              "targets": 0          } ],          "order": [[ 1, 'asc' ]]      } );        t.on( 'order.dt search.dt', function () {          t.column(0, {search:'applied', order:'applied'}).nodes().each( function (cell, i) {              cell.innerHTML = i+1;          } );      } ).draw();  } ); |
|  |  |

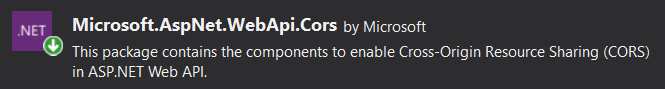
## RoutePrefix: tiền tố điều hướng

## Chia sẻ tài nguyên xuyên nguồn

vd: mình đang có web mvc có cổng [https://localhost:44399](https://localhost:44399/) và đang gọi tới 1 API có địa chỉ khác là http://localhost:54555

**Chia sẻ tài nguyên xuyên nguồn** ( [CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS) ) là một cơ chế sử dụng tiêu đề [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP) bổ sung để yêu cầu trình duyệt cung cấp cho ứng dụng web đang chạy tại một [nguồn](https://developer.mozilla.org/en-US/docs/Glossary/origin) , quyền truy cập vào các tài nguyên được chọn từ một nguồn khác. Một ứng dụng web thực thi một yêu cầu HTTP có nguồn gốc chéo khi nó yêu cầu một tài nguyên có nguồn gốc khác (miền, giao thức hoặc cổng) từ chính nó.

trước tiên phải cài gói này vào server api



sau đó trong: WebApiConfig.cs

thêm: config.EnableCors(new EnableCorsAttribute("http://localhost:4200, https://localhost:44398/", "\*", "\*"));

**thuộc tính 1**: tên adress cho phép truy cập vào server, mỗi địa chỉ cách nhau bởi dấu ‘.’ và tất cả đc viết dưới dạng chuỗi

* “\*” nếu muốn access all

**thuộc tính 2:** headers-vd "Accept, Content-Type, Origin, X-My-Header"

* “\*” nếu muốn access all

**thuộc tính 3:** method - gọi đến “GET, POST, PUT,..”

* “\*” nếu muốn access all

chỉ định chia sẻ co từng controller

phía trên class muốn chỉ đinh:

**[EnableCorsAttribute(“\*” , ”\*” , ”\*”)]**

public class EmployeeController: AipController{

...

// nếu muốn vô hiệu hóa 1 method chỉ định:

**[DisableCors]**

public IHttpActionResult GetLoaiDT(){

...

}

...

}

## HTTPClient vs SendAsync

### Get

Vd:

using (var client = new HttpClient())

{

client.BaseAddress = new Uri("http://206.189.90.147/");

//HTTP GET

var responseTask = client.GetAsync("api/getallproduct");

responseTask.Wait();

var result = responseTask.Result;

if (result.IsSuccessStatusCode)

{

var readTask = result.Content.ReadAsAsync<List<SanPhamModel>>();

readTask.Wait();

model.SP = readTask.Result;

}

else //web api sent error response

{

//log response status here..

model.SP = Enumerable.Empty<SanPhamModel>();

ModelState.AddModelError(string.Empty, "Lỗi server.");

}

}

Type(responseTask) = HttpResponseMessage

Vd: var responseTask = client.GetAsync("api/getProduct?id=" +id.toString());

### Post

Vd:

using (var client = new HttpClient())

{

client.BaseAddress = new Uri("http://206.189.90.147/");

//HTTP POST

var postTask = client.PostAsJsonAsync<NhaCungCapModel>("api/create\_nhacc", nhaCungCap);

postTask.Wait();

var result = postTask.Result;

if (result.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

}

### Put

Vd:

using (var client = new HttpClient())

{

client.BaseAddress = new Uri("http://206.189.90.147/api/");

//HTTP POST

var postTask = client.PutAsJsonAsync<SanPhamModel>("updateproduct", sanPham);

postTask.Wait();

var result = postTask.Result;

if (result.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

}  
Delete

### Delete

Vd:

using (var client = new HttpClient())

{

client.BaseAddress = new Uri("http://206.189.90.147/api/");

//HTTP DELETE

var deleteTask = client.DeleteAsync("deletenhacc?id=" + id.ToString());

deleteTask.Wait();

var result = deleteTask.Result;

if (result.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

}

### Function liên quan

#### Content.ReadAsStringAsync()--- -Lấy chuỗi json

HttpResponseMessage response = await client.GetAsync("Products");

//get data as Json string

string data = await response.Content.ReadAsStringAsync();

//use JavaScriptSerializer from System.Web.Script.Serialization

JavaScriptSerializer JSserializer = new JavaScriptSerializer();

//deserialize to your class

products = JSserializer.Deserialize<List<Product>>(data);

## Các lỗi thường gặp

500: lỗi server thực thi

404: Lỗi không tìm thấy server=> có thể là url không đúng hoặc server đó đã bị chết

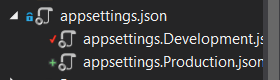
401: lỗi unAuth => có thể do chưa hawcoj đăng nhập sai dẫn đến sai auth

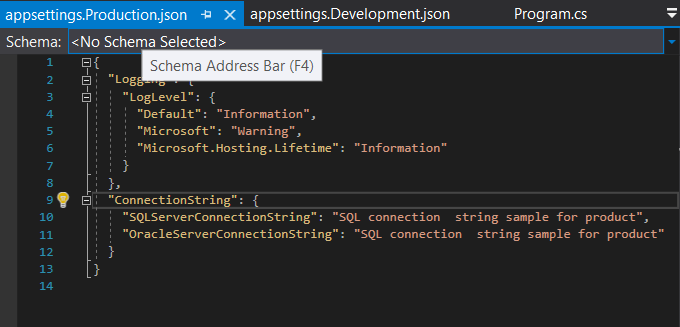
# .NET CORE

## Configurate environment

Nội dung được viết trong file json.

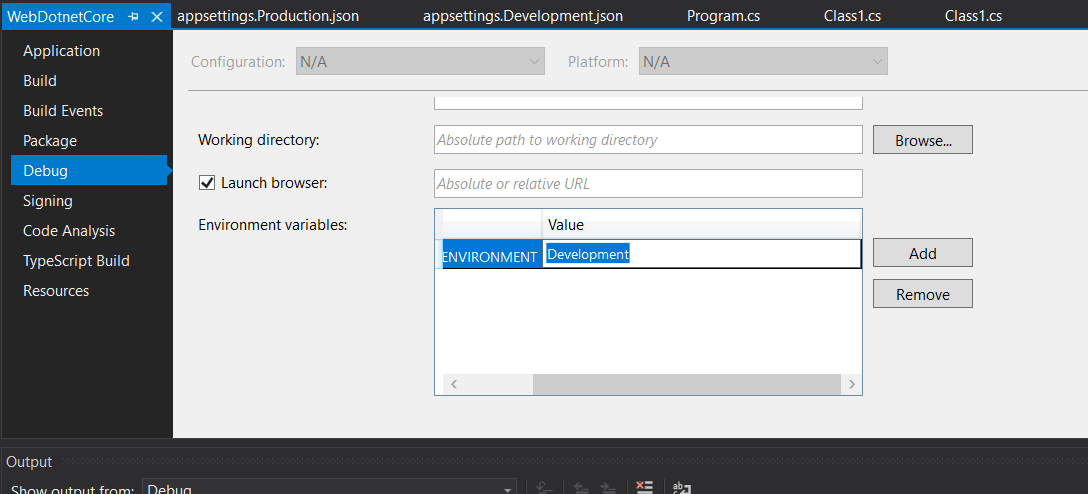
Bên dưới là 2 môi trường trong project : Dvelopment là mt có sẵn. Ứng với 1 môi trường là 1 file json



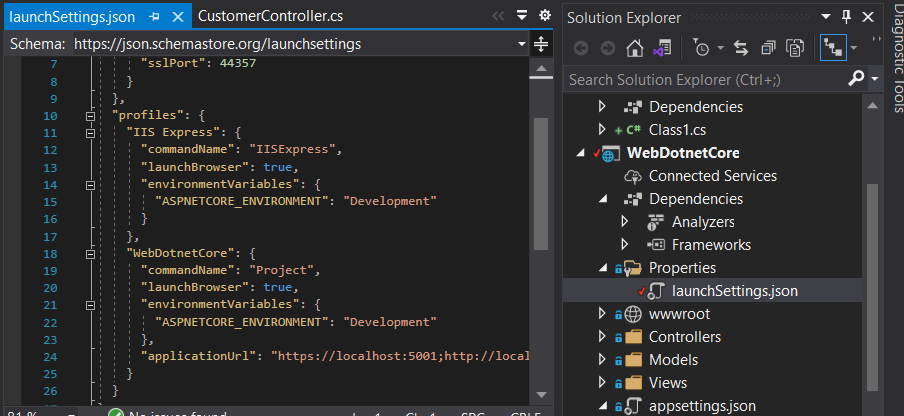


Chúng ta có thể cấu hình theo dạng key-value theo cấu trúc json để lưu trữ tương tự như [<appSetting>](#_<appSettings>) trong Web.config của .NET

* Thay đổi mô trường bằng cách: set in value column <- debug <-property of project



Hay thay trực trong file launchSetting.json



-lấy giữ liệu ra bằng Configuration.Section(key: “.:.:.:.” ).Value);

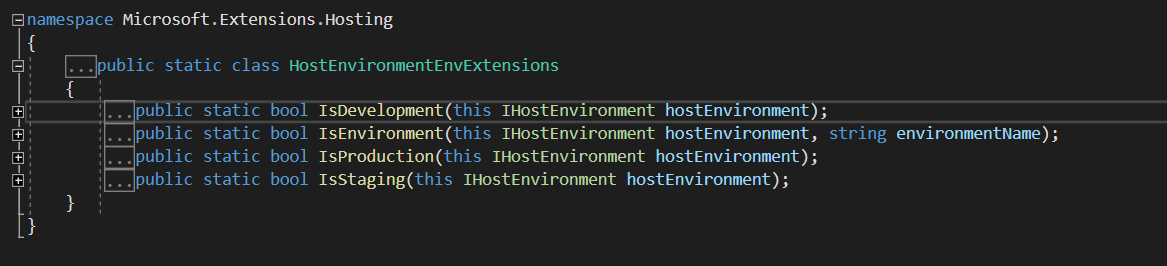
Mỗi phần con tiếp theo tương ứng 1 dấu “:”, với mảng ta có phần tử đầu tiên “<tên mảng>:0:…”



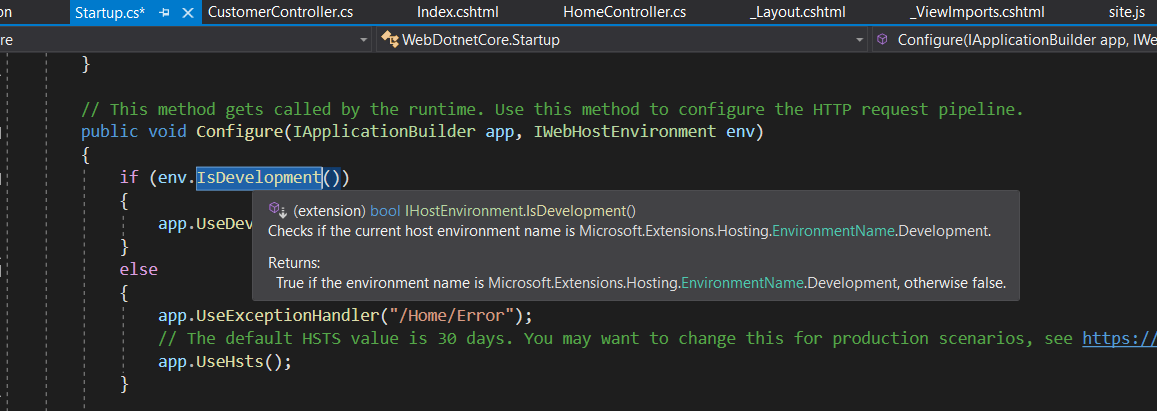
### View

* Ta có thể setting cho mỗi môi trường trong view như sau
* 
* With each tag <enviroment> in <head> tương ứng with if – else ifStartup.cs

Chúng ta có các môi trường



Trong Startuo.cs để kiểm tra tên môi trường đang sử dụng ta dùng biến env



## Model Validation

https://tedu.com.vn/lap-trinh-aspnet-core/model-validation-trong-aspnet-core-253.html

## Fluent API Configuration

Cấu hình entity cho giống hệt table trong sqlserver

Được cấu hình trong function: OnModelCreating(DbModelBuilder modelBuilder)

https://www.learnentityframeworkcore.com/configuration/fluent-api

## Chuyển đổi giữa Fluent API Configuration và DataAnnotations

<https://docs.microsoft.com/en-us/ef/core/modeling/entity-properties?tabs=data-annotations%2Cwithout-nrt>

### Lưu ý 1:

EF tự động xóa các bản ghi có liên quan trong bảng quan hệ của các thực thể có quan hệ nhiều-nhiều nếu một trong hai thực thể bị xóa. Nên không cần set là .WillCascadeOnDelete(); những trường hợp không khải bảng quan hệ trung gian thì cần set nếu trong sql để “delete rule = cascade”

### Lưu ý 2:

Nói chung, bạn không cần phải định cấu hình mối quan hệ một-nhiều trong khung thực thể vì các quy ước quan hệ một-nhiều bao hàm tất cả các kết hợp. Tuy nhiên, bạn có thể định cấu hình các mối quan hệ bằng cách sử dụng API Fluent tại một nơi để làm cho nó dễ bảo trì hơn.( <https://www.entityframeworktutorial.net/code-first/configure-one-to-many-relationship-in-code-first.aspx>)

## Gọi api

Gần tương tự [asp.net](#_Gọi_api)

Có 2 cách viết

[HttpGet("test/{id}")] == [HttpGet][Route("test/{id}")]

Ta có vd

[HttpGet("test/{id}&{name}")]

public IEnumerable<string> Get(int id, string name)

{

return new string[] { id.ToString(), name };

}

* url: http://localhost:51708/api/Values/test/1&hieu

[HttpGet]

public IEnumerable<string> Get(string name)

{

return new string[] { name};

}

* Url: <http://localhost:51708/api/Values?name=linh>

Lưu ý: 1 api có thể gọi thừa , thiếu tham số hay bị đảo ngược thứ tự tham số thì đều gọi được.

* [FromBody] 1class cũng có thừa thiếu property tùy ý. Chương trình sẽ chỉ mapping dữ liệu khi trùng tên

## Dapper

### 1 số lỗi thường gặp

Lỗi 1: không thể convert từ guid sang string khi sử dụ dapper kết nối với sql server

https://github.com/StackExchange/Dapper/issues/447

### Lưu ý:

Không dùng $emit trong hàm then() và catch();

Emit đc dùng trong tài liệu [vue](VueJS/VueJS%20Document.docx)

Excude

* Gần tương tự ado: sẻ dụng string query để thao tác dữ liệu
* Dùng với insert, update, delete, procedude, function, select

Query

* Thường dùng với select

VD: var selectObject = dbConnection.Query<TEntity>(storeName, parameters, commandType: CommandType.StoredProcedure);

# Dependency Injection

<https://xuanthulab.net/dependency-injection-di-trong-c-voi-servicecollection.html>

## Không áp dụng kỹ thuật DI

public class Horn {

public void Beep () => Console.WriteLine ("Beep - beep - beep ...");

}

public class Car {

public void Beep () {

// chức năng Beep xây dựng có định với Horn

// tự tạo đối tượng horn (new) và dùng nó

**Horn horn = new Horn ();**

horn.Beep ();

}

}

Code viết như trên khá thông dụng và vẫn chạy tốt. Bấm còi xe vẫn kêu Beep, beep ...

var car = new Car();

car.Beep(); // Beep - beep - beep ...

Nhưng code trên có một vấn đề là tính linh hoạt khi sử dụng. Chức năng Beep() của Car nó tự tạo ra đối tượng Horn và sử dụng nó - làm cho Car gắn cứng vào Horn với cấu trúc khởi tạo hiện thời.

[Source code - car-horn-nodi](https://github.com/xuanthulabnet/learn-cs-netcore/tree/master/CS027_Dependencyinjection/car_horn_nodi)

Nếu lớp Horn sửa lại, ví dụ muốn khởi tạo Horn phải chỉ ra một tham số nào đó, ví dụ như độ lớn tiêng còi level

public class Horn {

int level; // độ lớn của còi xe

**public Horn (int level) => this.level = level;** // thêm khởi tạo level

public void Beep () => Console.WriteLine ($"(level {level}) Beep - beep - beep ...");

}

Việc thay đổi Horn làm cho Car không còn dùng được nữa, nếu muốn Car hoạt động cần sửa lại code của Car, ví dụ tại Beep sửa thành

Horn horn = new Horn(10); // Khởi tạo với Horn với tham số level

horn.Beep();

Như vậy không sử dụng DI, việc thay đổi code của lớp này, dịch vụ này kéo theo phải cập nhật sửa đổi các đối tượng sử dụng nó. Điều này rất phức tạp nếu dự án code lớn lên. Hãy khắc phục nó bằng kỹ thuật DI

[Source code - car\_horn\_nodi\_update](https://github.com/xuanthulabnet/learn-cs-netcore/tree/master/CS027_Dependencyinjection/car_horn_nodi_update)

## Áp dụng kỹ thuật DI

Xây dựng lại ví dụ trên có sử dụng kỹ thuật DI xem sao. Khái niệm DI thì trừu tượng, nhưng bạn chỉ việc sử đổi cách viết code một chút là đã áp dụng DI.

Giữ code trong trường hợp thứ nhất (Horn khởi tạo không cần level). Nhưng lớp Car cập nhật lại:

public class Horn {

public void Beep () => Console.WriteLine ("Beep - beep - beep ...");

}

public class Car {

// horn là một Dependecy của Car

Horn horn;

// horn trong có được qua hàm tạo, ta nói

// horn Inject (bơm vào) bằng hàm khởi tạo

public Car(Horn horn) => this.horn = horn;

public void Beep () {

// Sử dụng Dependecy đã được Inject

horn.Beep ();

}

}

Khi sử dụng:

Horn horn = new Horn();

var car = new Car(horn); // horn inject vào car

car.Beep(); // Beep - beep - beep ...

Code trên hoạt động tương tự trường hợp thứ nhất. Bằng cách khai bảo Horn là một biến thành viên trong Car, Car đã có một dependency là đối tượng lớp Horn, dependency này không phải do Car tạo ra, nó được bơm vào (cung cấp) thông qua phương thức khởi tạo của nó.

[Source code - car\_horn\_with\_di](https://github.com/xuanthulabnet/learn-cs-netcore/tree/master/CS027_Dependencyinjection/car_horn_with_di)

Kết quả gọi car.Beep(); có vẻ kết quả vẫn như trên, nhưng code mới này có một số lợi ích. Ví dụ, nếu sửa cập nhật lại Horn bằng cách sửa phương thức khởi tạo của nó, thì lớp Car không phải sửa gì!

public class Horn {

int level; // thêm độ lớn còi xe

public Horn (int level) => this.level = level; // thêm khởi tạo level

public void Beep () => Console.WriteLine ("Beep - beep - beep ...");

}

Horn horn = new Horn(10);

var car = new Car(horn); // horn inject vào car

car.Beep(); // Beep - beep - beep ...

Như vậy, kỹ thuật DI giúp các đối tượng độc lập nhau một cách tối đa (lớp Car độc lập lớp Horn - cập nhật code Horn mà không cần cập nhật lại code Car)

## Các kiểu Dependency Injection

Cơ bản thì có 3 kiểu Inject bơm các Dependency vào đối tượng:

* **Inject thông qua phương thức khởi tạo:** cung cấp các Dependency cho đối tượng thông qua hàm khởi tạo ( như đã thực hiện ở ví dụ trên)
* **Inject thông qua setter:**tức các Dependency như là thuộc tính của lớp, sau đó inject bằng gán thuộc tính cho Depedency object.denpendency = obj;
* **Inject thông qua các Interface** - áp dụng Interface cho các Dependency - nó có thể dùng setter, phương thức khởi tạo để thực hiện Inject

Trong ba kiểu Inject thì Inject qua Interface rất phổ biến vì tính linh hoạt, mềm dẻo ...

## xây dựng lại code phần trên với kỹ thuật Inject qua Interface.

Đầu tiên xây dựng một interface là IHorn

public interface IHorn {

void Beep ();

}

Lớp Car được xây dựng để sử dụng IHorn như là Dependency

public class Car {

IHorn horn; // IHorn (Interface) là một Dependecy của Car

public Car (IHorn horn) => this.horn = horn; // Inject từ hàm tạo

public void Beep () => horn.Beep ();

}

Với cách triển hai DI với Interface như vậy, sử dụng Car rất linh hoạt và độc lập với nhiều loại đối tượng triển khai IHorn, ví dụ thử tạo ra hai loại còi một cái loại lơn và một cái loại nhẹ

public class HeavyHorn : IHorn

{

public void Beep() => Console.WriteLine("(kêu to lắm) BEEP BEEP BEEP ...");

}

public class LightHorn : IHorn

{

public void Beep() => Console.WriteLine("(kêu bé lắm) beep bep bep ...");

}

Lúc này khi sử dụng, Car của bạn có dùng loại còi nào thì dùng - logic code giống nhau

Car car1 = new Car(new HeavyHorn());

car1.Beep(); // (kểu to lắm) BEEP BEEP BEEP ...

Car car2 = new Car(new LightHorn());

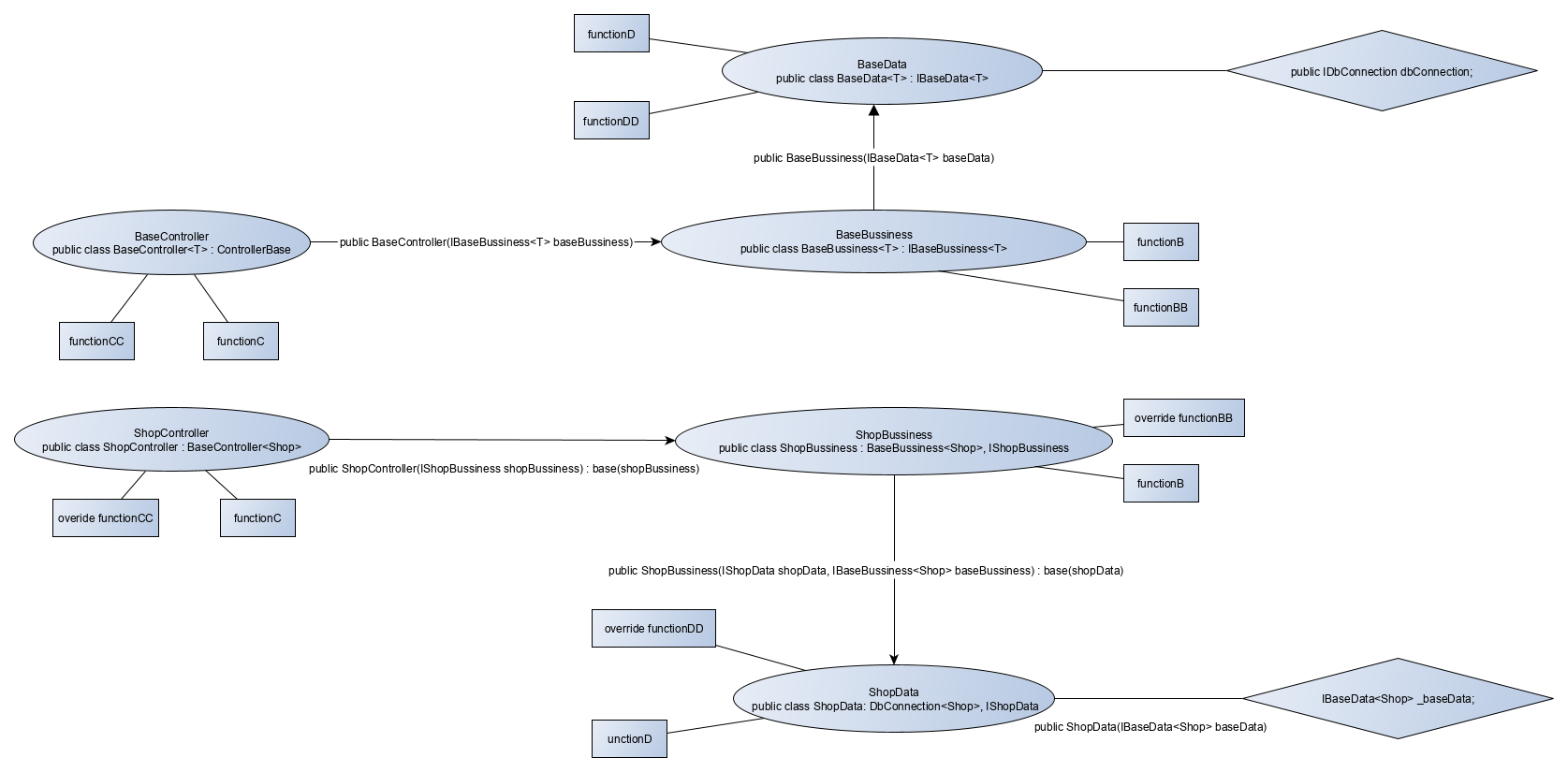
car2.Beep(); // (kểu bé lắm) beep bep bep ...

[Source code - car\_horn\_di\_interface](https://github.com/xuanthulabnet/learn-cs-netcore/tree/master/CS027_Dependencyinjection/car_horn_with_di_update)

**Inject** bằng phương thức khởi tạo nên tập trung vào đó, vì các thư viện DI hỗ trợ tốt

Lưu ý; những function của riêng class **HeavyHorn** và **LightHorn** mà không được định nghĩa trong interface **Ihorn** sẽ không gọi được

## Áp dụng với cấu trúc dự án 3 lớp



Với luồng xử lý của “Shop” ta có thể override lại mọi hàm để đáp ứng nghiệp vụ của luồng này

* Cấu hình phụ thuộc trong public void ConfigureServices(IServiceCollection services). File Status.cs

// cấu hình Dependency Injection

services.AddScoped(typeof(IBaseData<>), typeof(BaseData<>));

services.AddScoped(typeof(IBaseBussiness<>), typeof(BaseBussiness<>));

services.AddScoped<IShopData, ShopData>();

services.AddScoped<IShopBussiness, ShopBussiness>();

# MySQL

## Insert

* Giống ms sql

## Update

* Giống ms sql

## Delete

* Giống ms sql

## Proceduce

<https://freetuts.net/truyen-tham-so-vao-mysql-stored-procedure-108.html>

### Truyền tham số

! gần tương tự sql server

#### In

 Đây là chế độ mặc định (nghĩa là nếu bạn không định nghĩa loại nào thì nó sẽ hiểu là IN). Khi bạn sử dụng mức này thì nó sẽ được bảo vệ an toàn, có nghĩa là sẽ không bị thay đổi nếu như trong Procedure có tác động đến

**Ví dụ**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | DELIMITER $$  CREATE PROCEDURE getById(IN id INT(11))  BEGIN      /\*Code\*/  END; $$  DELIMITER; |

#### Out

Chế độ này nếu như trong Procedure có tác động thay đổi thì nó sẽ thay đổi theo. Nhưng có điều đặc biệt là dù trước khi truyền vào mà bạn gán giá trị cho biến đó thì vẫn sẽ không nhận được vì mặc định nó luôn hiểu giá trị truyền vào là NULL.

Vd:

|  |
| --- |
| DELIMITER $$  DROP PROCEDURE IF EXISTS changeTitle $$  CREATE PROCEDURE changeTitle(OUT title VARCHAR(255))  BEGIN      SET title = 'Hoc lap trinh online tai freetuts.net';  END; $$  DELIMITER; |

Bây giờ ta gọi Procedure này như sau:

|  |  |
| --- | --- |
| 1  2  3 | CALL changeTitle(@title);    SELECT @title; |

Thì kết quả sẽ như sau:

call store procedure in mysql 1 png

#### Inout

Đây là sự kết hợp giữa IN và OUT. Nghĩa là **có thể gán giá trị trước** và **có thể bị thay đổi** nếu trong Procedure có tác động tới

**Gọi Stored Procedure**

Tạo xong rồi bây giờ làm thế nào để gọi đến Store này? Đơn giản để gọi tới Store nào thì ta chỉ cần dùng cú pháp như sau:

|  |  |
| --- | --- |
| 1 | CALL storeName();   * + Khác vói exec trong mssql |

## Function

## Parameter

## Variable

Không có kiểu này trongmysql

DECLARE rate int(11) = 2;

Mà thay bằng

DECLARE rate int(11) default 2;

Ta có: set rate = 2;

## Trigger

NV-55455

## View

# MS SQL

https://www.sqlservertutorial.net/sql-server-stored-procedures/stored-procedure-output-parameters/

## Proceduce

### Truyền tham số

#### In

#### Out

Chế độ này nếu như trong Procedure có tác động thay đổi thì nó sẽ thay đổi theo. Nhưng có điều đặc biệt là dù trước khi truyền vào mà bạn gán giá trị cho biến đó thì vẫn sẽ không nhận được vì mặc định nó luôn hiểu giá trị truyền vào là NULL.

Vd:

|  |
| --- |
| DELIMITER $$  DROP PROCEDURE IF EXISTS changeTitle $$  CREATE PROCEDURE changeTitle(OUT title VARCHAR(255))  BEGIN      SET title = 'Hoc lap trinh online tai freetuts.net';  END; $$  DELIMITER; |

Bây giờ ta gọi Procedure này như sau:

|  |  |
| --- | --- |
| 1  2  3 | CALL changeTitle(@title);    SELECT @title; |

Thì kết quả sẽ như sau:

call store procedure in mysql 1 png

#### Inout

Đây là sự kết hợp giữa IN và OUT. Nghĩa là **có thể gán giá trị trước** và **có thể bị thay đổi** nếu trong Procedure có tác động tới

**Gọi Stored Procedure**

Tạo xong rồi bây giờ làm thế nào để gọi đến Store này? Đơn giản để gọi tới Store nào thì ta chỉ cần dùng cú pháp như sau:

|  |  |
| --- | --- |
| 1 | CALL storeName();   * + Khác vói exec trong mssql |

## Phân trang trong sql

ALTER proc [dbo].[SelectShopResult\_Proc](@pageIndex int , @pageSize int, @shopCode nvarchar(30) = N'', @shopName nvarchar(100) = N'', @address nvarchar(400) = N'', @phone nvarchar(20) = N'', @statusId nvarchar(8) = N'')

AS

BEGIN

declare @start int;

set @start = (@pageIndex-1)\*@pageSize

select s.shopId , s.shopCode, s.shopName, s.shopName, s.address, s.phone, s.taxCode, s.statusId, ss.statusName, s.street, s.createDate, u.userName as createUserName from shop as s left join StatusShop as ss on s.statusId = ss.statusId left join dbo.[User] as u on s.createUserId = u.userId

where shopCode LIKE '%'+@shopCode+ '%'

and shopName LIKE '%'+@shopName+ '%'

and [address] LIKE '%'+@address+ '%'

and ([phone] LIKE '%'+@phone+ '%' or [phone] is null)

and (s.statusId LIKE '%'+@statusId+ '%' or s.statusId is null)

order by shopCode OFFSET @start ROWS FETCH NEXT @pageSize ROWS ONLY;

END

GO

# 1 số lỗi thường gặp

* Lỗi không tìm thấy dll của project <https://stackoverflow.com/questions/1421862/metadata-file-dll-could-not-be-found>

using Misa.Common;

using Misa.Common.Entities;

using Misa.Common.Enum;

using Misa.Common.Results;

using Misa.Data;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Misa.Bussiness

{

public class EmployeeBussiness

{

private EmployeeData employeeData;

public EmployeeBussiness()

{

employeeData = new EmployeeData();

}

/// <summary>

/// lấy ra toàn bộ danh sách nhân viên

/// </summary>

/// <returns>ServiceResult</returns>

/// update: 3/2/2021

public ServiceResult GetData()

{

try

{

var listEmployee = employeeData.GetData();

return new ServiceResult()

{

Data = listEmployee,

Error = null,

MISACukCukCode = MISACukCukServiceCode.Success

};

}

catch(Exception ex)

{

var errorResult = new List<ErrorResult>();

errorResult.Add(new ErrorResult()

{

DevMsg = ex.Message,

UserMsg = Properties.Resources.ErrorServive\_Employee\_GetData,

});

return new ServiceResult()

{

Data = null,

Error = errorResult,

MISACukCukCode = MISACukCukServiceCode.BadRequest

};

}

}

/// <summary>

/// thêm mới nhân viên

/// </summary>

/// <param name="employee">thông tin nhân viên</param>

/// <returns>ServiceResult</returns>

/// update: 3/2/2021

public ServiceResult InsertEmloyee(Employee employee)

{

try

{

ServiceResult serviceResult = new ServiceResult();

// check trùng mã nhân viên

if (CheckCustomerCodeDuplicate(employee.EmployeeCode))

{

serviceResult.Error.Add(new ErrorResult()

{

DevMsg = Properties.Resources.ErrorService\_Employee\_EmployeeCode\_duplicate,

UserMsg = Properties.Resources.ErrorService\_Employee\_EmployeeCode\_duplicate

});

serviceResult.MISACukCukCode = MISACukCukServiceCode.BadRequest;

}

// Check trùng số điện thoại

if (CheckPhoneNumberDuplicate(employee.PhoneNumber))

{

serviceResult.Error.Add(new ErrorResult()

{

DevMsg = Properties.Resources.ErrorService\_Employee\_PhoneNumber\_duplicate,

UserMsg = Properties.Resources.ErrorService\_Employee\_PhoneNumber\_duplicate

});

serviceResult.MISACukCukCode = MISACukCukServiceCode.BadRequest;

}

// Check trùng email

if (CheckEmailDuplicate(employee.Email))

{

serviceResult.Error.Add(new ErrorResult()

{

DevMsg = Properties.Resources.ErrorService\_Employee\_Email\_duplicate,

UserMsg = Properties.Resources.ErrorService\_Employee\_Email\_duplicate

});

serviceResult.MISACukCukCode = MISACukCukServiceCode.BadRequest;

}

// valiable dữ liệu

return serviceResult;

}

catch (Exception ex)

{

var errorResult = new List<ErrorResult>();

errorResult.Add(new ErrorResult()

{

DevMsg = ex.Message,

UserMsg = Properties.Resources.ErrorServive\_Employee\_Insert,

});

return new ServiceResult()

{

Data = null,

Error = errorResult,

MISACukCukCode = MISACukCukServiceCode.Exception

};

}

//return employeeData.Insert(employee);

}

/// <summary>

/// cập nhậ nhân viên

/// </summary>

/// <param name="employee">thông tin nhân viên</param>

/// <returns></returns>

public int UpdateEmloyee(Employee employee)

{

var oldEmployee = employeeData.GetByI(employee.EmployeeId);

// check thuộc tính

return employeeData.Update(employee);

}

public int DeleteEmloyee(string id)

{

// check thuộc tính

return employeeData.Delete(id);

}

/// <summary>

/// Kiểm tra tính hợp lệ của dữ liệu

/// </summary>

/// <param name="employee"> thông tin nhân viên</param>

/// create by : vu minh hieu (3/2/2021)

public void ValidateObject(Employee employee)

{

}

/// <summary>

/// Kiểm tra mã nhân viên đã tồn tại hay chưa

/// </summary>

/// <param name="entity">Kiểm tra mã nhân viên có tồm tại</param>

/// <returns>trùng là true, không trùng là false</returns>

/// Created by: 3/2/2021

public bool CheckCustomerCodeDuplicate(string employeeCode)

{

if(employeeData.GetByEmployeeCode(employeeCode).Count() > 0)

{

return true;

}

return false;

}

/// <summary>

/// Kiểm tra số điện thoại đã tồn tại hay chưa

/// </summary>

/// <param name="employeeCode"></param>

/// <returns>trùng là true, không trùng là false</returns>

/// update: 3/2/2021

public bool CheckPhoneNumberDuplicate(string phoneNumber)

{

if (employeeData.GetByPhoneNumber(phoneNumber).Count() > 0)

{

return true;

}

return false;

}

public bool CheckEmailDuplicate(string email)

{

if (employeeData.GetByEmail(email).Count() > 0)

{

return true;

}

return false;

}

}

}

using Misa.Common;

using Misa.Common.Entities;

using Misa.Data;

using System;

using System.Collections.Generic;

namespace Misa.Bussiness

{

public class CustomerBussiness

{

private CustomerData customerData;

public CustomerBussiness()

{

customerData = new CustomerData();

}

public IEnumerable<Customer> GetAllData()

{

return customerData.GetData();

}

public int InsertCustomer(Customer customer)

{

// check thuộc tính

return customerData.Insert(customer);

}

public void UdpdateCustomer()

{

;

}

}

}

|  |  |
| --- | --- |
| ErrorService\_Employee\_Email\_duplicate | Email đã tồn tại |
| ErrorService\_Employee\_EmployeeCode\_duplicate | Mã khách hàng đã tồn tại |
| ErrorService\_Employee\_PhoneNumber\_duplicate | Số điện thoại đã tồn tại |
| ErrorServive\_Employee\_GetData | Không thể lấy thông tin danh sách nhân viên |
| ErrorServive\_Employee\_Insert | Không thể thêm mới nhân viên |