Experimenting CSS (Part 1)

1. XHTML: Những điều cần biết trước khi thực hành viết CSS

1.1. Cấu trúc chuẩn của trang XHTML

1.2. Khai báo DOCTYPE

1.3. Kiểm duyệt XHTML

1.4. Một số thẻ XHTML thường gặp

2. Thực hành viết CSS

2.1. Kiểm duyệt CSS

2.2. CSS Selectors: Các phương pháp chọn đối tượng để áp dụng CSS

2.3. CSS Specificity: Cách xác định CSS nào được quyền ưu tiên áp dụng cho đối tượng

2.4. CSS Box Model: Cách xác định vùng hiển thị của đối tượng

2.5. Floating Elements: Cách trồi đối tượng về 2 phía và xử lý hiện tượng trồi

2.6. Positioning Elements: Cách định vị đối tượng

1. XHTML: Những điều cần biết trước khi thực hành viết CSS

1.1. Cấu trúc chuẩn của trang XHTML

Một trang XHTML khi được khởi tạo, cần có một cấu trúc chuẩn như sau:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Title of Page</title>
</head>
<body>
    <!-- Đặt nội dung và thẻ XHTML tại đây -->
</body>
</html>
```

Trong đó có 2 phần chính:

- `<!DOCTYPE>`: luôn được khai báo trên cùng và có thể tùy chọn loại DOCTYPE phù hợp với trang XHTML nhất.
- `<html>`: bao gồm 2 phần
  - `<head>`: dùng để khai báo tiêu đề trang, thông tin trang, gắn các liên kết với các tập tin JS/CSS bên ngoài,...
  - `<body>`: toàn bộ nội dung trang và các thẻ dùng để chưa nội dung sẽ được đặt trong thẻ này

Chú ý:

- Trong một trang XHTML, 4 đối tượng trên là duy nhất, theo chuẩn và quy định của W3C.
- Bên cạnh đó còn có một thẻ cũng được khuyến cáo là duy nhất và được khuyến cáo là luôn sử dụng trong mỗi trang XHTML, đó là thẻ `<h1>`. Thẻ này xác định tiêu đề hiện tại của trang (cùng nội dung giống với thẻ `<title>` được khai báo trong phần `<head>`).

1.2. Khai báo DOCTYPE

Điều đầu tiên trong quá trình bắt đầu viết một trang XHTML luôn là việc xác định sử dụng DOCTYPE nào cho trang XHTML đó. Và khai báo DOCTYPE luôn được đặt trên cùng trong mỗi trang XHTML.

Khai báo DOCTYPE là cách giúp cho các trình duyệt hiểu được chuẩn nào đang được sử dụng trong trang XHTML theo quy định của tổ chức Web W3C. Dựa trên DOCTYPE đó, các trình duyệt sẽ áp dụng các quy tắc chuẩn để dịch các thẻ XHTML và hiển thị NỘI DUNG mà các thẻ đó đang chứa.

Việc áp dụng và chọn đúng kiểu khai báo DOCTYPE sẽ giúp cho mỗi người lập trình XHTML xác định được một số quy tắc cơ bản (cách lồng các thẻ, cách đóng mở thẻ, các thuộc tính được sử dụng trên thẻ,...) trong quá trình viết thẻ XHTML để đạt được sử kiểm duyệt của hệ thống kiểm duyệt chuẩn XHTML từ W3C (*W3C Markup Validation*).

Có 3 loại khai báo DOCTYPE cho trang XHTML:

- **XHTML Strict**: sử dụng khai báo này khi muốn trang XHTML được viết theo cách chặt chẽ nhất và không được sử dụng những thẻ đã bị cấm (*deprecated tags*) hoặc thẻ trình diễn (*presentational tags*)
- **XHTML Transitional**: sử dụng khai báo này nếu vẫn muốn sử dụng các thẻ trình diễn trong trang XHTML
- **XHTML Frameset**: sử dụng khai báo này nếu trong trang XHTML có sử dụng `iframe` (không tính đến những thẻ `iframe` tự sinh ra trong quá trình chạy code)

Khuyến cáo: sử dụng **XHTML Strict** để đảm bảo tuyệt đối các yêu cầu cho một trang XHTML theo kiểm duyệt của W3C.

1.3. Kiểm duyệt XHTML

Kiểm duyệt XHTML là cách giúp người lập trình biết được trang XHTML đã được tạo ra có theo chuẩn W3C không?

Để theo chuẩn, trước hết phải khai báo DOCTYPE trên đầu mỗi trang XHTML. Sau khi đã hoàn thành việc viết nội dụng qua các thẻ trong trang XHTML, người lập trình sử dụng hệ thống kiểm duyệt của W3C để kiểm tra tính chính xác trang XHTML. Trong quá trình kiểm tra, hệ thống sẽ thông báo kết quả, trong trường hợp chưa đạt, hệ thống sẽ thông báo cụ thể dòng code nào bị lỗi, lỗi gì, thông tin về lỗi đó và cách xử lý lỗi.

Sử dụng hệ thống kiểm duyệt XHTML của W3C tại đây: http://validator.w3.org/

1.4. Một số thẻ XHTML thường gặp

Trong quá trình viết trang XHTML, người lập trình thường sử dụng một số thẻ phổ biến như sau:

- **Loại thẻ block** (*mặc định có chiều rộng chiếm đầy chiều rộng của đối tượng cấp cha trực tiếp chứa thẻ đó*):
  - `<h1>..<h6>`: chỉ chứa các thẻ inline (`<a>`, `<img>`,...) và text, dùng để tạo tiêu đề cho trang, cho một phần nào đó trong trang. `<h1>` chỉ được sử dụng 1 lần duy nhất cho trang. Các thẻ còn lại sẽ được sử dụng giảm dần theo cấp độ quan trọng của tiêu đề trong cùng một trang.
  - `<div>`: dùng để nhóm tất cả các loại thẻ để phân biệt giữa các phần nội dung khác nhau trong cùng một trang.
  - `<p>`: chỉ chứa các thẻ inline (`<a>`, `<img>`,...) và text, dùng để tạo ra các đoạn văn bản trong một khối nội dung
  - `<ul>`, `<ol>`, `<dl>`: dùng để nhóm tất cả các loại thẻ, tạo ra một danh sách cách nhóm nội dung, không theo thứ tự, theo thứ tự hoặc theo cách định nghĩa.
  - `<address>`: chỉ chứa các thẻ inline (`<a>`, `<img>`,...) và text, dùng để tạo nội dung là địa chỉ
  - `<table>`: dùng để tạo bảng các phần nội dung, dữ liệu cần hoặc có thể được hiển thị cho phép so sánh qua lại giữa các dòng (`<tr>`) và cột (`<td>`) trong cùng một bảng.
  - `<form>`: có thể chưa được tất cả các loại thẻ, tuy nhiên khuyến cáo chỉ chứa các thẻ block, dùng để tạo ra một form nhập liệu từ người dùng.
  - `<fieldset>`: có thể chưa được tất cả các loại thẻ, dùng để phân tách các phần nhập liệu theo cùng một chủ để trong cùng một form. Đi kèm với thẻ này thường là thẻ `<legend>`, chỉ chứa text và là tiêu đề cho phần nhập liệu trong `<fieldset>` tương ứng.
  - `<blockquote>`: có thể chứa được một số loại thẻ khác nhau, nhưng khuyến cáo chỉ chứa trục tiếp các thẻ `<p>`, dùng để tạo ra các đoạn văn được trích từ một nguồn nào đó hoặc phát ngôn của một nhân vật nào đó.
- **Loại thẻ inline** (*mặc định có thể nằm liền kề với các thẻ inline khác trên cùng một dòng và chỉ xuống hàng nếu chiều rộng của đối tượng cấp cha trực tiếp không đủ để chưa tất cả chúng*):
  - `<a>`: chỉ chứa các thẻ inline (`<a>`, `<img>`,...) và text, dùng để tạo ra các siêu liên kết hoặc bookmark trong cùng một trang hoặc tại một điểm nào đó ở trang khác.
  - `<span>`: chỉ chứa các thẻ inline (`<a>`, `<img>`,...) và text, dùng để nhóm một phần nội dung nào đó theo mục đích riêng của người lập trình, thường là được gán thêm CSS Selectors để áp dụng CSS cho phần nội dung này.
  - `<em>`: chỉ chứa text, dùng để tạo nhấn mạnh phần nội dung được bao
  - `<strong>`: giống thẻ `<em>` tuy nhiên mức độ nội dung được nhấn mạnh hơn
  - `<label>`: chỉ chứa các thẻ inline (`<a>`, `<span>`,...) và text, tạo nhãn cho một input nào đó trong form. Input đó có thể là text/radio/checkbox, option trong một dropdown hoặc là một vùng nhập liệu `<textarea>`

2. Thực hành viết CSS

2.1. Kiểm duyệt CSS

Kiểm duyệt CSS là cách giúp người lập trình biết được các thuộc tính CSS đã được tạo ra có theo chuẩn W3C không?

Sau khi đã hoàn thành việc viết các thuộc tính CSS, người lập trình sử dụng hệ thống kiểm duyệt của W3C để kiểm tra tính chính xác của các thuộc tính đó. Trong quá trình kiểm tra, hệ thống sẽ thông báo kết quả, trong trường hợp chưa là đạt, hệ thống sẽ thông báo cụ thể dòng code nào bị lỗi, lỗi gì, thông tin về lỗi đó và cách xử lý lỗi.

Hệ thống chỉ kiểm duyệt cách viết các thuộc tính, tức là cú pháp, chứ không thể giúp người lập trình xác định được cách viết đó có làm việc được trên các trình duyệt hay không?

Sử dụng hệ thống kiểm duyệt CSS của W3C tại đây: http://jigsaw.w3.org/css-validator/

2.2. CSS Selectors: Các phương pháp chọn đối tượng để áp dụng CSS

Đây là cách chính thống và duy nhất để áp dụng các thuộc tính CSS lên một đối tượng (các thẻ XHTML) trong một trang.

Có rất nhiều phương pháp để chọn lọc ra được đối tượng để áp dụng CSS, tuy nhiên một số phương pháp trong đó không được hỗ trợ bởi những trình duyệt củ (IE6-, Firefox2-,...) nên chúng ta không nghiên cứu sâu các phương pháp đó.

Một số quy tắc và yêu cầu trong quá trình đặt Selectors:

- Đặt tên với Class và ID
  - không được đặt số trước, luôn bắt đầu bằng một chữ cái
  - không nên sử dụng dấu gạch dưới "_" trong tên, nên sử dụng dấu gạch ngang "-"
  - đặt tên theo nội dung đối tượng đang chứa, không được sử dụng đặc tả CSS để đặt tên (`.blue { color: blue; }`), vì các giá trị của thuộc tính có thể thay đổi sau này, nên cách đặt này sẽ bất hợp lý. Trường hợp trên nên đặt là `.author { color: blue; }`
- Tên đặt hoặc nhóm Selectors không nên quá dài và cố gắng chọn tên hoặc nhóm sao cho tường minh nhất, giúp người lập trình dễ dàng hiểu, nhận dạng đối tượng và debug

Một số phương pháp phổ biến để chọn lọc đối tượng trong trang XHTML:

- **Type Selectors**: đây là phương pháp đơn giản nhất, tuy nhiên lại chứa đựng rất nhiều nguy hiểm không lường trước nếu chúng ta có rất nhiều thẻ trong một trang XHTML
  - Cú pháp:
    `x { css property 1: value; css property 2: value; }`
  - Giải thích: áp dụng các thuộc tính CSS lên tất cả các thẻ `x` như `p`, `h1`, `div`, `span`, `a`, `img`, `em`, `strong`,...
- **Class Selectors**: đây là phương pháp phổ biến nhất, cho phép chọn lọc một nhóm các đối tượng có thể hiện giống nhau để cùng áp dụng CSS lên đó
  - Cú pháp:
    `.classname { css property 1: value; css property 2: value; }`
  - Giải thích: áp dụng các thuộc tính CSS lên tất cả các thẻ có class chứa `classname`.
- **ID Selectors**: cũng phổ biến như phương pháp Class Selectors, tuy nhiên phương pháp cho phép chọn lọc ra một đối tượng duy nhất trong cùng một trang
  - Cú pháp:
    `#idname { css property 1: value; css property 2: value; }`
  - Giải thích: áp dụng các thuộc tính CSS lên thẻ có id là `idname`.
- **Descendant Selectors**: phương pháp này được sử dụng thường xuyên, cho phép chọn lọc ra một hoặc một nhóm đối tượng theo nhánh Selectors
  - Cú pháp:
    `.classname1 .classname2 { css property 1: value; css property 2: value; }`
  - Giải thích: áp dụng các thuộc tính CSS lên tất cả các thẻ có class chứa `classname2` và nằm trong thẻ có class chứa `classname1`.
- **Combine Selectors**: dùng phương pháp này để tinh giản số lượng khai báo CSS, đồng bộ hóa cách thức thể hiện các nhóm đối tượng và tiện lợi trong quá trình thay đổi và đồng nhất
  - Cú pháp:
    `.classname1, .classname2 { css property 1: value; css property 2: value; }`
  - Giải thích: áp dụng các thuộc tính CSS lên tất cả các thẻ có class chứa `classname1` hoặc `classname2`.

Tham khao chi tiết các phương pháp còn lại tại đây: http://css.maxdesign.com.au/selectutorial/

2.3. CSS Specificity: Cách xác định CSS nào được quyền ưu tiên áp dụng cho đối tượng

Các thuộc tính CSS được áp dụng lên đối tượng tuân theo thứ tự như sau:

- Inline styles: `<h2 style="color: red;">Title of Page</h2>` (cách này chủ yếu dùng để debug hoặc hotfix trong trường hợp không thể sửa các thuộc tính CSS vì lo ngại sẽ ảnh hưởng đến các phần khác mà chúng ta chưa kiểm soát hết)
- Inpage styles: các thuộc tính được đặt trong thẻ `<style>` nằm trong `<head>` (trên thực tế thì bước này bỏ qua, vì chúng ta không sử dụng phương pháp đặt CSS trong trang, mà phải đặt ở một file ngoài trang và liên kết nó bằng thẻ `<link>`)
- External styles: các thuộc tính được đặt ở file riêng (.css) và được liên kết vào trang bằng thẻ `<link>`.

Mặc định, các thuộc tính CSS sẽ được trình duyệt đọc và áp dụng lên một đối tượng nào đó từ trên xuống dưới. Điều đó đồng nghĩa với một đối tượng có thể được thừa hưởng các thuộc tính CSS ở nhiều nơi, thay vì chỉ tại một đoạn CSS code. Tuy nhiên cách viết này không hợp lý, bởi vì nếu cùng một đối tượng, cùng một kiểu khai báo nhưng lại tách các thuộc tính CSS ra thành nhiều phần và đặt ở nhiều nơi thì rất khó để kiểm soát.

Một đối tượng thường được thừa hưởng nhiều thuộc tính CSS khác nhau, phụ thuộc vào vị trị của đối tượng trong toàn trang, trong các khối nội dung khác nhau,...

Để hiểu chính xác các thuộc tính CSS nào được áp dụng sau cùng trên một đối tượng, chúng ta sử dụng phương pháp **CSS Specificity**.

Đặt quy ước:

- Inline styles: **1000**
- ID Selectors: **100**
- Attribute, Class hoặc Pseudo-class: **10**
- Element hoặc Pseudo-element: **1**

Dùng phép cộng đơn giản giá trị tổng của một selector, giá trị sau cùng lớn hơn thì các thuộc tính CSS trong selector đó sẽ được áp dụng cho đối tượng.

Một số ví dụ chi tiết:

- `* { }` **0**
- `li { }` **1** (1 đối tượng)
- `li:first-line { }` **2** (1 đối tượng và 1 giả đối tượng (*pseudo-element*))
- `ul li { }` **2** (2 đối tượng)
- `ul ol+li { }` **3** (3 đối tượng)
- `h1 + *[rel=up] { }` **11** (1 thuộc tính và 1 đối tượng)

- `ul ol li.red { }` **13** (1 class và 3 đối tượng)
- `li.red.level { }` **21** (2 class và 1 đối tượng)
- `style=""` **1000** (1 inline style)
- `#sith { }` **100** (1 id)
- `body #darkside .sith p { }` **112** (1 id, 1 class và 2 đối tượng)

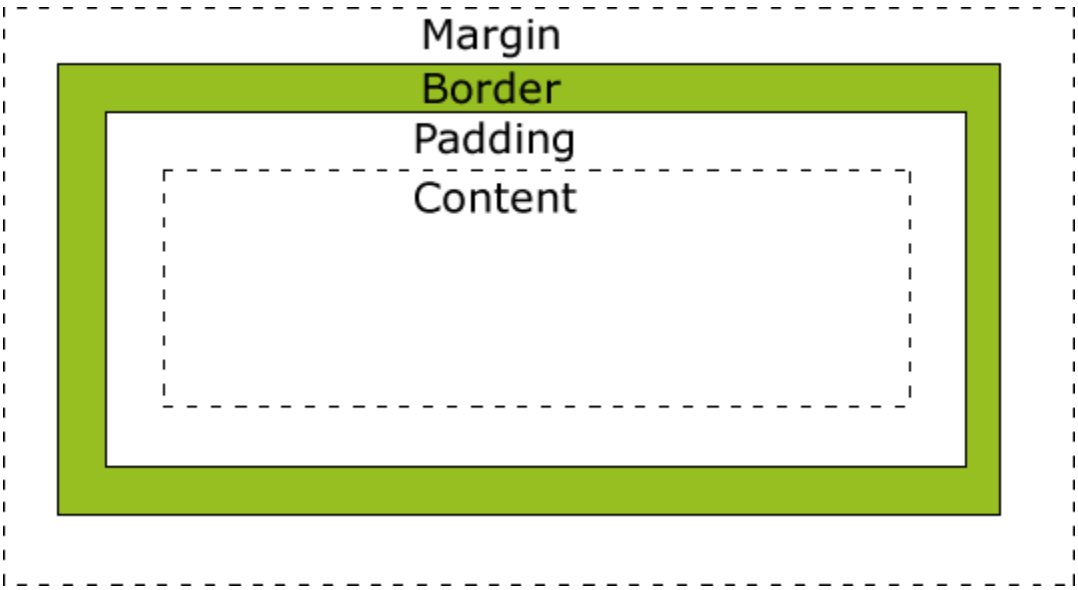
Tham khảo chi tiết hơn tại đây: http://www.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/

2.4. CSS Box Model: Cách xác định vùng hiển thị của đối tượng

Tất cả các đối tượng đều được xem như là những khối nội dung nằm trong một trang. Khái niệm 'Box Model' về bản chất đó là một khối chứa các đối tượng (các thẻ), bao gồm:

- **margins**: khoảng cách giữa đối tượng với các đối tượng xung quanh, vùng này hoàn toàn trong suốt và không thể áp dụng CSS lên vùng này từ chính đối tượng đang được áp dụng thuộc tính margin.
- **borders**: đường kẻ sát bao quanh phần padding và nội dung chứa trong đối tượng
- **paddings**: khoảng cách giữa đối tượng đến nội dụng thực
- **nội dung thực**: đó là nội dung đối tượng chứa, bao gồm nội dung, hình ảnh,...

Hình ảnh dưới đây sẽ minh họa trực quan khái niệm CSS Box Model:



Việc xác định được vùng hiển thị của đối tượng và khoảng cách margin/padding sẽ giúp người lập trình tính toán chính xác các giá trị trong quá trình tạo trang web từ các thiết kế.

2.5. Floating Elements: Cách trồi đối tượng về 2 phía và xử lý hiện tượng trồi

Float (trồi) một đối tượng là phương pháp đặt đối tượng về 2 phía so với đối tượng cấp cha chứa nó.

Chú ý: Như thế, không thể có định nghĩa trồi về giữa, bởi vì đối tượng cấp cha không xác định được không gian chiều ngang để canh vị trí tương ứng cho đối tượng có ý định trồi và nằm giữa.

- Cú pháp: `.author { float: left; } .counter { float: right; }`
- Giải thích: khi một đối tượng được trồi, nó mặc định sẽ không còn thuộc về đối tượng cấp cha chứa nó nữa. Nói cách khác, một đối tượng được xem là con thực sự của một đối tượng khác khi nó nằm trong đối tượng đó, về cấu trúc XHTML và không có thuộc tính Float hoặc Position (`relative` hoặc `absolute`), về đặc tả CSS.

Hiện tượng khi áp dụng trồi một đối tượng: khi đối tượng đã được trồi, không gian ban đầu của đối tượng này sẽ bị trống và bị chiếm bởi các đối tượng liền kề theo sau, bất chấp cấp độ của đối tượng (nằm trong hay nằm ngoài đối tượng cấp cha trực tiếp của đối tượng được trồi) và không có thuộc tính `clear`. Việc này khiến người lập trình khó kiểm soát được vị trí của các đối tượng trong cùng một trang cũng như trong một khối nội dung.

Để tránh hiện tượng này, sử dụng các phương pháp chống trồi phổ biến:

- Thiết lập giá trị chiều cao cho đối tượng cấp cha, đảm bảo ít nhất bằng chiều cao của đối tượng con được trồi
  - Ưu điểm: đơn giản khi áp dụng
  - Nhược điểm: khi nội dung khác trong đối tượng cấp cha nhiều hơn và vượt quá chiều cao đã định, thì xảy ra hiện tượng tràn nội dung.
- Thiết lập thuộc tính clear cho đối tượng bên dưới để tránh cho đối tượng này trồi lên phần không gian trống mà đối tượng trồi ngay trên tạo ra
  - Ưu điểm: đơn giản khi áp dụng
  - Nhược điểm: việc sử dụng thuộc tính clear đôi khi lại sinh ra các lỗi layout khác ở trình duyệt IE. Hơn nữa, việc này chỉ đảm bảo đối tượng bên dưới không trồi lên chiếm không gian.
- Sử dụng phương pháp khác:
  - Chèn một đối tượng trống có thuộc tính clear ngay sau đối tượng trồi: `<div class="clear"></div>` (`.clear { clear: both; }`): đây là cách làm an toàn nhất, tuy nhiên lại dẫn đến việc lạm dụng thẻ vô nghĩa cho mục đích CSS và làm tăng kích cỡ file HTML vì thường có rất nhiều đối tượng được áp dụng trồi trong một trang.
  - Thiết lập thuộc tính width cho đối tượng trồi sao cho nó chiếm hết không gian trống nó đã tạo ra, như vậy các đối tượng sau sẽ không còn không gian để chiếm.
  - Thiết lập thuộc tính `overflow: hidden` cho đối tượng cấp cha: đây là một khám phá nhỏ trong quá trình debug của cộng đồng web, nó không dựa trên nguyên tắc lý thuyết nào. Nhược điểm chính của phương pháp này là nếu đối tượng cấp cha có thuộc tính height cố định, thì phần nội dung tràn quá phần chiều cao đó sẽ bị che dấu.

- Riêng IE6, thiết lập thuộc tính `height: 1%` sẽ giải quyết được hiện tượng này, tuy nhiên sẽ có nảy sinh vấn đề khác, nếu đối tượng cấp cha trực tiếp có thiết lập giá trị height. Khi đó, đối tượng cấp cha tự động tính ra giá trị chiều cao của mình bằng 1% chiều cao của đối tượng cha trực tiếp.

Kết luận: Có nhiều phương pháp xử lý hiện tượng do kết quả một đối tượng trồi tạo nên, tuy nhiên mỗi phương pháp đều có ưu nhược điểm, nên tùy trường hợp thực tế mà áp dụng các phương pháp phù hợp.

2.6. Positioning Elements: Cách định vị đối tượng

Trong một số trường hợp, chúng ta cần phải định vị lại vị trí của một đối tượng so với vị trí thực tế của nó, hoặc so với toàn trang. Để định vị được một đối tượng, chúng ta phải thiết lập thuộc tính `position` (`relative` hoặc `absolute`) lên đối tượng, đi kèm là 4 thuộc tính khác `top/right/bottom/left` để xác định tọa độ vị trí mới.

Có 2 cách định vị một đối tượng:

- Định vị tương đối: thiết lập vị trí mới tương ứng với vị trí hiện tại.
  - Trong cách định vị này, đối tượng cần định vị được thiết lập thuộc tính `postion: relative`, cùng với nhóm thuộc tính vị trí hướng `top/right/bottom/left`.
  - Chú ý: chỉ được sử dụng 1 trong nhóm 4 thuộc tính vị trí ở trên, hoặc 2, trong đó 1 từ `top/bottom` (theo trục y tọa độ) và 1 từ `left/right` (theo trục x tọa độ).
- Định vị tuyệt đối: thiết lập vị trí mới so với đối tượng cấp cha gần nhất có thuộc tính `postion: relative`. Nếu không có đối tượng cấp cha nào có thuộc tính đó, thì đối tượng hiện tại sẽ lấy `body` làm gốc tọa độ (0, 0).
  Chú ý: Trên IE7-, việc sử dụng padding trên đối tượng làm gốc tọa độ có thể sẽ ảnh hưởng đến gốc tọa độ và khiến các giá trị định vị sẽ bị ảnh hưởng theo. Hiện tượng này không xuất hiện thường xuyên và khó để bắt được nên cần phải kiểm thử trên các trình duyệt khác nhau sau khi đã áp dụng định vị trí tuyệt đối cho đối tượng.