Trong CSS Flexbox, cả `flex-basis` và `flex-grow` đều được sử dụng để xác định kích thước của phần tử con trong container flex, nhưng chúng phục vụ các mục đích khác nhau. Dưới đây là so sánh chi tiết về hai thuộc tính này:

---

### \*\*1. `flex-basis`\*\*

- \*\*Chức năng\*\*:

- Xác định \*kích thước ban đầu\* của một phần tử con trước khi không gian còn lại trong container được phân phối (giống như kích thước "cơ sở").

- Kích thước này có thể là một `chiều rộng` hoặc `chiều cao`, phụ thuộc vào \*\*hướng dòng chính\*\* (main axis) của container.

- \*\*Giá trị\*\*:

- Có thể đặt bằng bất kỳ giá trị CSS hợp lệ nào cho chiều rộng/chiều cao (ví dụ: `px`, `%`, `em`, `auto`, v.v.).

- Mặc định là `auto`, khi đó kích thước của phần tử sẽ phụ thuộc vào chính nội dung của nó (hoặc chiều rộng/chiều cao được định rõ thông qua `width`/`height`).

- \*\*Cách hoạt động\*\*:

- `flex-basis` xác định "kích thước cơ sở", sau đó container Flexbox sẽ phân phối thêm hoặc giảm không gian (nếu có) dựa trên các thuộc tính `flex-grow` và `flex-shrink`.

- \*\*Ví dụ\*\*:

```css

.item {

flex-basis: 200px; /\* Kích thước cơ sở của mỗi phần tử là 200px \*/

}

```

=> Mỗi phần tử sẽ cố gắng chiếm 200px trên trục chính (nếu có đủ không gian).

---

### \*\*2. `flex-grow`\*\*

- \*\*Chức năng\*\*:

- Xác định tỷ lệ mà một phần tử có thể \*phóng to\* để lấp đầy không gian trống trong một hàng/dòng \*sau khi đã tính đến `flex-basis`\*.

- \*\*Giá trị\*\*:

- Là một số nguyên dương hoặc số thập phân:

- Nếu `flex-grow` là `0` (mặc định), phần tử sẽ không lấp thêm không gian trống nào.

- Nếu `flex-grow` lớn hơn `0`, phần tử sẽ nhận thêm không gian trống \*\*theo tỷ lệ\*\* với các phần tử khác.

- \*\*Cách hoạt động\*\*:

- Container Flexbox sẽ tính không gian còn thừa trên dòng chính, rồi phân phối nó cho các phần tử dựa theo giá trị `flex-grow`.

- \*\*Ví dụ\*\*:

```css

.item {

flex-grow: 2; /\* Phần tử này sẽ chiếm không gian trống gấp đôi phần tử với flex-grow: 1 \*/

}

```

=> Một phần tử với `flex-grow: 2` sẽ rộng gấp đôi phần tử có `flex-grow: 1`.

---

### \*\*So sánh `flex-basis` và `flex-grow`\*\*

| Thuộc tính | `flex-basis` | `flex-grow` |

|------------------|------------------------------------------|-----------------------------------------------|

| \*\*Chức năng\*\* | Xác định kích thước ban đầu của phần tử | Xác định phần tử có mở rộng chiếm không gian trống hay không |

| \*\*Giá trị\*\* | `px`, `%`, `auto`, v.v. | Số nguyên hoặc số thập phân (ví dụ: `0`, `1`, `2`) |

| \*\*Ý nghĩa\*\* | "Cơ sở ban đầu" | "Phóng to theo tỷ lệ không gian trống" |

| \*\*Tác động\*\* | Tác động ngay từ đầu trước khi phân phối không gian trống | Chỉ áp dụng khi phân bổ không gian trống |

---

### \*\*Ví dụ kết hợp `flex-basis` và `flex-grow`\*\*

```css

.container {

display: flex;

}

.item1 {

flex-basis: 100px;

flex-grow: 1;

}

.item2 {

flex-basis: 50px;

flex-grow: 2;

}

```

- \*\*`item1`\*\*:

- Có kích thước cơ sở là 100px.

- Khi có không gian trống, nó sẽ nhận thêm \*\*1 phần\*\*.

- \*\*`item2`\*\*:

- Có kích thước cơ sở là 50px.

- Khi có không gian trống, nó sẽ nhận thêm \*\*2 phần\*\* (gấp đôi `item1`).

Nếu container rộng 400px:

- Kích thước ban đầu của `item1` và `item2` lần lượt là 100px và 50px (tổng cộng 150

Trong CSS Flexbox, `flex-wrap` và `flex-grow` là hai thuộc tính hoàn toàn khác nhau, phục vụ các mục đích khác nhau liên quan đến bố cục và cách phần tử bên trong một \*\*Flexbox container\*\* được hiển thị.

---

### \*\*1. `flex-wrap`\*\*

- \*\*Chức năng\*\*:

- Quyết định xem các phần tử bên trong `flex-container` có được phép bẻ dòng nếu không đủ không gian để hiển thị trong một dòng hay không.

- Liên quan đến bố trí \*\*không gian nhiều dòng (wrapping)\*\*.

- \*\*Giá trị\*\*:

- `nowrap` \*(mặc định)\*: Các phần tử được buộc hiển thị trên \*\*một hàng\*\* duy nhất, bất kể có đủ chỗ hay không. Nếu không vừa, chúng sẽ bị nén lại.

- `wrap`: Khi không đủ không gian để hiển thị tất cả các phần tử trên một dòng, phần tử sẽ được bẻ dòng sang dòng mới.

- `wrap-reverse`: Giống `wrap`, nhưng dòng mới sẽ được sắp xếp \*\*ngược chiều\*\*, nghĩa là dòng dưới cùng sẽ nằm dưới phần dòng trên.

- \*\*Ví dụ\*\*:

```css

.container {

display: flex;

flex-wrap: wrap; /\* Cho phép các phần tử con bẻ dòng \*/

}

```

Với `flex-wrap: wrap`, nếu container không đủ rộng, các phần tử con sẽ tự động tràn sang dòng mới.

---

### \*\*2. `flex-grow`\*\*

- \*\*Chức năng\*\*:

- Quy định \*\*tỷ lệ\*\* mà các phần tử con sẽ "phình ra" để chiếm thêm không gian trống \*\*trong một dòng duy nhất\*\*.

- Áp dụng cho \*\*một phần tử con cụ thể\*\*, không ảnh hưởng đến việc bố trí nhiều dòng.

- \*\*Giá trị\*\*:

- Một giá trị \*\*số nguyên hoặc số thập phân\*\* (mặc định là `0`).

- Nếu `flex-grow` của một phần tử là `0`, nó không nhận thêm bất kỳ không gian trống nào, chỉ giữ kích thước ban đầu của nó.

- Phần tử với `flex-grow` lớn hơn sẽ phình to hơn các phần tử khác, dựa trên tỷ lệ.

- \*\*Ví dụ\*\*:

```css

.item {

flex-grow: 1; /\* Mỗi phần tử sẽ chia đều không gian trống \*/

}

```

Nếu bạn có 3 phần tử con trong một container với các thiết lập:

```css

.item1 {

flex-grow: 1;

}

.item2 {

flex-grow: 2;

}

.item3 {

flex-grow: 1;

}

```

Trong trường hợp này:

- `item2` sẽ chiếm gấp \*\*hai lần không gian trống\*\* so với `item1` và `item3`.

---

### \*\*Tóm tắt so sánh\*\*

| Thuộc tính | `flex-wrap` | `flex-grow` |

|------------------|------------------------------------------|---------------------------------------|

| \*\*Chức năng\*\* | Quy định bẻ dòng trong container flex | Quy định tỷ lệ mở rộng của phần tử con|

| \*\*Liên quan đến\*\*| Không gian dòng và container | Không gian trống trong dòng hiện tại |

| \*\*Giá trị\*\* | `nowrap`, `wrap`, `wrap-reverse` | Số nguyên hoặc số thập phân (mặc định: `0`) |

Hy vọng phần giải thích này giúp bạn hiểu cách hoạt động và chức năng khác biệt giữa hai thuộc tính! 😊

Trong Flexbox CSS, các thuộc tính bạn liệt kê phục vụ cho việc tạo và quản lý bố cục linh hoạt. Dưới đây sẽ là \*\*phân tích và so sánh chi tiết\*\* của từng thuộc tính trong Flexbox, bao gồm chức năng và mục đích sử dụng.

---

## \*\*1. Các thuộc tính liên quan đến bố cục chính của container\*\*

### \*\*1.1. `flex-direction`\*\*

- Quy định \*\*hướng\*\* sắp xếp các phần tử con dọc theo \*\*trục chính\*\* (main axis) của container. ( hướng trục main axis)

- Các giá trị:

- \*\*`row`\*\* (mặc định): Sắp xếp các phần tử theo hàng ngang, từ trái sang phải.

- \*\*`row-reverse`\*\*: Sắp xếp theo hàng ngang, từ phải sang trái.

- \*\*`column`\*\*: Sắp xếp theo cột dọc, từ trên xuống dưới.

- \*\*`column-reverse`\*\*: Sắp xếp theo cột dọc, từ dưới lên trên.

---

### \*\*1.2. `flex-wrap`\*\*

- Quyết định cách các phần tử xử lý nếu không đủ không gian trong container:

- \*\*`nowrap`\*\* (mặc định): Tất cả phần tử nằm trên cùng một dòng (phần tử có thể bị thu nhỏ để vừa vặn).

- \*\*`wrap`\*\*: Các phần tử con sẽ bẻ dòng và hiển thị trên nhiều hàng/cột.

- \*\*`wrap-reverse`\*\*: Giống như `wrap`, nhưng các dòng sẽ hiển thị theo chiều ngược lại.

---

### \*\*1.3. `flex-flow`\*\*

- Là \*\*từ khóa rút gọn\*\*, kết hợp của \*\*`flex-direction`\*\* và \*\*`flex-wrap`\*\*.

- Cú pháp:

```css

flex-flow: <flex-direction> <flex-wrap>;

```

- Ví dụ:

```css

flex-flow: row wrap; /\* Hướng hàng ngang và cho phép bẻ dòng \*/

```

---

### \*\*1.4. `justify-content`\*\*

- Điều chỉnh cách các phần tử được căn chỉnh \*\*trên trục chính\*\* (main axis - theo `flex-direction`).

- Các giá trị:

- \*\*`flex-start`\*\* (mặc định): Các phần tử nằm ở đầu trục chính.

- \*\*`flex-end`\*\*: Các phần tử nằm ở cuối trục chính.

- \*\*`center`\*\*: Các phần tử nằm ở giữa trục chính.

- \*\*`space-between`\*\*: Khoảng cách đều nhau, với phần tử đầu tiên và cuối cùng sát mép container.

- \*\*`space-around`\*\*: Khoảng cách đều nhau, nhưng có khoảng cách \*\*nhỏ hơn\*\* ở hai đầu.

- \*\*`space-evenly`\*\*: Khoảng cách đều nhau, bao gồm cả 2 đầu.

---

### \*\*1.5. `align-content`\*\*

- Điều chỉnh cách \*\*các dòng\*\* (nếu có nhiều dòng) được căn chỉnh trên \*\*trục phụ\*\* (cross axis - vuông góc với `flex-direction`).

- Chỉ hoạt động khi sử dụng `flex-wrap` với nhiều dòng.

- Các giá trị:

- \*\*`flex-start`\*\*: Các dòng dồn vào đầu trục phụ.

- \*\*`flex-end`\*\*: Các dòng dồn vào cuối trục phụ.

- \*\*`center`\*\*: Các dòng dồn vào giữa trục phụ.

- \*\*`space-between`\*\*: Khoảng cách đều giữa các dòng.

- \*\*`space-around`\*\*: Khoảng cách đều nhưng có khoảng cách nhỏ hơn ở hai cạnh.

- \*\*`stretch`\*\* (mặc định): Các dòng được kéo dãn để lấp đầy không gian trục phụ.

---

### \*\*1.6. `align-items`\*\*

- Điều chỉnh cách các phần tử được căn chỉnh \*\*trên trục phụ\*\* khi có \*\*dòng chính\*\*.

- Các giá trị:

- \*\*`flex-start`\*\*: Dồn các phần tử về phía \*\*đầu trục phụ\*\*.

- \*\*`flex-end`\*\*: Dồn về phía \*\*cuối trục phụ\*\*.

- \*\*`center`\*\*: Căn giữa trên trục phụ.

- \*\*`baseline`\*\*: Căn theo đường cơ sở của văn bản.

- \*\*`stretch`\*\* (mặc định): Phần tử được kéo dãn để lấp đầy trục phụ (nếu không có kích thước cố định).

---

### \*\*1.7. `align-self`\*\*

- Kiểm soát \*\*một phần tử con cụ thể\*\* trên \*\*trục phụ\*\*, ghi đè giá trị `align-items` của container.

- Các giá

### \*\*1.7. `align-self`\*\* \*(TIẾP TỤC)\*

- Thuộc tính này cho phép \*\*một phần tử con cụ thể\*\* kiểm soát vị trí căn chỉnh trên \*\*trục phụ\*\* (cross axis). Nó ghi đè giá trị của `align-items` được kế thừa từ container.

- Các giá trị:

- \*\*`auto`\*\* (mặc định): Sử dụng giá trị từ `align-items` của container.

- \*\*`flex-start`\*\*: Căn về phía đầu của trục phụ.

- \*\*`flex-end`\*\*: Căn về phía cuối của trục phụ.

- \*\*`center`\*\*: Căn giữa trên trục phụ.

- \*\*`baseline`\*\*: Căn theo đường cơ sở của phần tử.

- \*\*`stretch`\*\*: Phần tử được kéo dãn để lấp đầy trục phụ (nếu không có kích thước cố định).

- Ví dụ:

```css

.container {

display: flex;

align-items: center;

}

.item {

align-self: flex-end;

}

```

=> Trong ví dụ trên, các phần tử trong container đều được căn giữa trục phụ (`align-items: center`), nhưng một phần tử cụ thể có class `item` sẽ được căn ở cuối trục phụ (`align-self: flex-end`).

---

### \*\*5. `justify-self` (CSS Grid)\*\*

Mặc dù thuộc tính `justify-self` \*\*không phải thuộc tính của Flexbox\*\* mà là của \*\*CSS Grid\*\*, nhưng mình sẽ giải thích rõ ràng để bạn hiểu cách hoạt động của nó.

---

### \*\*5.1. `justify-self` là gì?\*\*

`justify-self` được sử dụng để canh chỉnh (align) \*\*một phần tử con\*\* của Grid Container theo \*\*trục ngang\*\* (horizontal axis) trong \*\*khung của ô mà phần tử đó chiếm\*\*.

Đối với Flexbox, bạn không sử dụng `justify-self` (bạn sẽ sử dụng `align-self` hoặc `justify-content` thay thế). Tuy nhiên, nếu bạn đang làm việc với \*\*CSS Grid\*\*, thì `justify-self` rất hữu ích.

---

### \*\*5.2. Giá trị của `justify-self`\*\*

Các giá trị chính:

- `start`: Căn phần tử ở \*\*lề trái\*\* của ô Grid.

- `end`: Căn phần tử ở \*\*lề phải\*\* của ô Grid.

- `center`: Căn phần tử ở \*\*giữa\*\* ô Grid (ngang).

- `stretch` (mặc định): Làm cho phần tử con \*\*kéo giãn hết chiều rộng của ô Grid\*\* nếu có thể.

---

### \*\*5.3. Cú pháp\*\*

```css

grid-item {

justify-self: value;

}

```

---

### \*\*5.4. Ví dụ\*\*

```html

<div class="grid-container">

<div class="item item1">1</div>

<div class="item item2">2</div>

<div class="item item3">3</div>

</div>

```

```css

.grid-container {

display: grid;

grid-template-columns: 100px 100px 100px; /\* Gồm 3 cột \*/

grid-template-rows: 100px; /\* 1 hàng \*/

gap: 10px;

}

.item1 {

justify-self: start; /\* Canh trái trong ô lưới \*/

}

.item2 {

justify-self: center; /\* Canh giữa trong ô lưới \*/

}

.item3 {

justify-self: end; /\* Canh phải trong ô lưới \*/

}

```

#### \*\*Kết quả\*\*:

- `1`: Căn trái.

- `2`: Ở giữa ô Grid.

- `3`: Căn phải.

---

### \*\*5.5. `justify-self` vs. `justify-items`\*\*

Thêm một điểm quan trọng:

- \*\*`justify-self`:\*\* Canh chỉnh từng phần tử riêng lẻ trong một ô Grid.

- \*\*`justify-items`:\*\* Canh chỉnh tất cả các phần tử bên trong Grid Container theo cùng một cách.

---

### \*\*5.6. Sử dụng kết hợp `justify-self` và `align-self`\*\*

Ngoài việc căn chỉnh trên \*\*trục ngang\*\* với `justify-self`, bạn cũng có thể căn chỉnh \*\*trục dọc\*\* với `align-self`.

Ví dụ:

```css

.item {

justify-self: center; /\* Căn ngang giữa ô Grid \*/

align-self: start; /\* Căn dọc phía trên ô Grid \*/

}

```

---

Nếu bạn muốn giải thích chi tiết hơn hoặc ví dụ cụ thể với dự án thực tế, cứ yêu cầu nhé! 😊

## \*\*2. Các thuộc tính liên quan đến phần tử con (flex items)\*\*

### \*\*2.1. `flex-basis`\*\*

- Xác định \*\*kích thước cơ bản ban đầu\*\* (chiều rộng trên trục chính hoặc chiều cao trên trục phụ). Sau khi xác định kích thước, không gian còn lại sẽ được phân phối dựa trên các giá trị `flex-grow` và `flex-shrink`.

- Giá trị:

- Các giá trị kích thước hợp lệ (ví dụ: `px`, `%`, `em`, `auto`).

- \*\*`auto`\*\* (mặc định): Sử dụng kích thước nội dung hoặc giá trị được đặt cụ thể qua `width` hoặc `height`.

- Ví dụ:

```css

.item {

flex-basis: 200px;

}

```

=> Phần tử này sẽ có kích thước ban đầu là 200px.

---

### \*\*2.2. `flex-grow`\*\*

- Xác định mức độ \*\*phóng to\*\* của phần tử con khi có không gian trống trên trục chính.

- Giá trị:

- Một số không âm (thông thường là `0`, `1`, `2`,...).

- \*\*`0`\*\* (mặc định): Phần tử không mở rộng để chiếm không gian thừa.

- Nếu phần tử có giá trị `flex-grow` lớn hơn, nó sẽ chiếm \*\*nhiều không gian thừa hơn\*\*.

- Ví dụ:

```css

.item {

flex-grow: 2;

}

```

=> Phần tử này sẽ chiếm không gian lớn hơn các phần tử có `flex-grow: 1`.

---

### \*\*2.3. `flex-shrink`\*\*

- Xác định mức độ \*\*thu nhỏ\*\* của phần tử con khi không có đủ không gian trong container.

- Giá trị:

- Một số không âm (thường là `0`, `1`, `2`,...).

- \*\*`1`\*\* (mặc định): Phần tử có thể thu nhỏ kích thước để vừa với container.

- \*\*`0`\*\*: Phần tử không thu nhỏ kích thước trong mọi trường hợp.

- Ví dụ:

```css

.item {

flex-shrink: 0;

}

```

=> Phần tử này sẽ không bị thu nhỏ ngay cả khi container không đủ chỗ.

---

### \*\*2.4. `flex`\*\*

- Là thuộc tính \*\*rút gọn\*\* kết hợp giữa \*\*`flex-grow`\*\*, \*\*`flex-shrink`\*\*, và \*\*`flex-basis`\*\*.

- Cú pháp:

```css

flex: <flex-grow> <flex-shrink> <flex-basis>;

```

- Một số ví dụ thường gặp:

- \*\*`flex: 1;`\*\* (rút gọn của `flex: 1 1 0%;`): Phần tử sẽ mở rộng và thu gọn linh hoạt để phù hợp với container.

- \*\*`flex: 0;`\*\* (rút gọn của `flex: 0 1 auto;`): Phần tử không mở rộng, nhưng có thể thu nhỏ.

- \*\*`flex: none;`\*\* (rút gọn của `flex: 0 0 auto;`): Không mở rộng

### \*\*2.5. `order`\*\*

- Xác định \*\*thứ tự hiển thị\*\* của các phần tử con trong container. Mặc định, các phần tử được sắp xếp theo thứ tự chúng xuất hiện trong HTML. Tuy nhiên, với thuộc tính `order`, bạn có thể thay đổi thứ tự trực quan mà không cần thay đổi cấu trúc HTML.

- Giá trị:

- \*\*Số nguyên\*\* (dương hoặc âm).

- \*\*Mặc định là `0`\*\*: Các phần tử có giá trị `order` thấp hơn sẽ được đặt trước.

- Lưu ý: `order` không ảnh hưởng đến luồng tài liệu thực của DOM.

- Ví dụ:

```css

.item1 {

order: 2;

}

.item2 {

order: 1;

}

.item3 {

order: 3;

}

```

=> Trong container, phần tử `.item2` sẽ hiển thị trước, sau đó là `.item1`, cuối cùng là `.item3`.

---

## \*\*3. Một số ví dụ áp dụng Flexbox trong thực tế\*\*

### \*\*3.1. Tạo menu ngang\*\*

```html

<div class="menu">

<div>Home</div>

<div>About</div>

<div>Services</div>

<div>Contact</div>

</div>

```

```css

.menu {

display: flex;

justify-content: space-between; /\* Phân bố đều các phần tử, khoảng cách giữa chúng bằng nhau \*/

}

```

---

### \*\*3.2. Tạo layout 2 cột, cột chính và cột phụ\*\*

```html

<div class="container">

<div class="main-column">Main Content</div>

<div class="sidebar">Sidebar</div>

</div>

```

```css

.container {

display: flex;

}

.main-column {

flex: 3; /\* Chiếm 3 phần không gian \*/

}

.sidebar {

flex: 1; /\* Chiếm 1 phần không gian \*/

}

```

---

### \*\*3.3. Canh giữa hoàn toàn phần tử\*\*

```html

<div class="container">

<div class="item">Centered</div>

</div>

```

```css

.container {

display: flex;

justify-content: center; /\* Canh giữa theo trục chính \*/

align-items: center; /\* Canh giữa theo trục phụ \*/

height: 100vh; /\* Chiều cao toàn màn hình \*/

}

```

---

### \*\*3.4. Responsive hàng và cột với `flex-wrap`\*\*

```html

<div class="container">

<div class="item">1</div>

<div class="item">2</div>

<div class="item">3</div>

</div>

```

```css

.container {

display: flex;

flex-wrap: wrap; /\* Đưa xuống dòng khi không đủ không gian \*/

column-gap: 10px; /\* Khoảng cách giữa các cột \*/

row-gap: 10px; /\* Khoảng cách giữa các dòng \*/

}

.item {

flex: 1 1 150px; /\* Không gian tối thiểu mỗi item là 150px \*/

}

```

---

## \*\*4. Kết hợp Flexbox và Media Queries\*\*

Sử dụng Media Queries để làm giao diện responsive khi thay đổi kích thước màn hình.

### Ví dụ: Layout responsive 3 cột chuyển thành 1 cột trên màn hình nhỏ

```html

<div class="container">

<div class="item">Header</div>

<div class="item">Content</div>

<div class="item">Footer</div>

</div>

```

```css

.container {

display: flex;

flex-direction: row; /\* Hiển thị ngang 3 cột \*/

}

.item {

flex: 1; /\* Tất cả các cột đều chiếm không gian bằng nhau \*/

}

@media (max-width: 768px) { /\* Khi màn hình nhỏ hơn 768px \*/

.container {

flex-direction: column; /\* Chuyển thành hiển thị dọc \*/

}

}

```

---

Flexbox là một công cụ mạnh mẽ để xây dựng giao diện linh hoạt (responsive), đặc biệt khi làm việc với các thiết kế hiện đại. Hy vọng bạn đã nắm vững các điểm cốt lõi để áp dụng vào thực tế! 😊 Nếu cần làm rõ thêm điều gì, đừng ngần ngại hỏi nhé!