Tài liệu: Inheritance và Specificity trong CSS

Mục tiêu

- Hiểu rõ **quy tắc kế thừa** và **tính ưu tiên** trong CSS, cách chúng hoạt động, và vai trò trong việc giải quyết xung đột kiểu dáng.
- Nắm được cách trình duyệt áp dụng kiểu dáng khi nhiều quy tắc CSS cạnh tranh.
- Cung cấp ví dụ thực tế và bài tập để áp dụng vào dự án.

Đối tượng

- Sinh viên hoặc lập trình viên đã có kiến thức cơ bản về CSS (selector, property, value).
- Người muốn hiểu sâu hơn về cách CSS quản lý kiểu dáng trong các tình huống phức tạp.

---

1. Quy tắc kế thừa (Inheritance)

Định nghĩa

Kế thừa (inheritance) là cơ chế trong CSS cho phép một số thuộc tính của phần tử cha được tự động áp dụng cho các phần tử con, giúp giảm lặp lại mã và duy trì tính nhất quán trong giao diện.

Cách hoạt động

- **Cơ chế**: Khi một thuộc tính CSS được áp dụng cho phần tử cha, các phần tử con sẽ kế thừa giá trị của thuộc tính đó, trừ khi chúng được ghi đè bởi một quy tắc cụ thể hơn.
- **Thuộc tính kế thừa**: Một số thuộc tính CSS được kế thừa tự nhiên, ví dụ:
  - **Text-related**: `color`, `font-family`, `font-size`, `line-height`, `text-align`.
  - **List-related**: `list-style`, `list-style-type`.
  - **Others**: `visibility`, `cursor`.
- **Thuộc tính không kế thừa**: Các thuộc tính liên quan đến bố cục hoặc kích thước thường không kế thừa, ví dụ:
  - `margin`, `padding`, `border`, `width`, `height`, `background`.
- **Kiểm soát kế thừa**:
  - **`inherit`**: Buộc phần tử con sử dụng giá trị của cha.
  - **`initial`**: Đặt lại giá trị mặc định của thuộc tính.
  - **`unset`**: Nếu thuộc tính kế thừa tự nhiên, lấy giá trị cha; nếu không, lấy giá trị mặc định.
- **Cơ chế trình duyệt**: Trình duyệt kiểm tra thuộc tính của phần tử cha và áp dụng cho con nếu thuộc tính đó thuộc nhóm kế thừa, trừ khi có quy tắc khác ghi đè.

Cú pháp

```
/* Thuộc tính kế thừa tự nhiên */
body {
  font-family: Arial, sans-serif;
  color: #333;
}

/* Buộc kế thừa */
.element {
  font-family: inherit;
}

/* Đặt lại giá trị mặc định */
.element {
  color: initial;
}
```

Ví dụ minh họa

Ví dụ cơ bản

Tạo một div với các phần tử con kế thừa `color` và `font-family`.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Inheritance Basic Example</title>
  <style>
    .container {
      font-family: Arial, sans-serif;
      color: navy;
    }
    .child {
      /* Kế thừa font-family và color từ .container */
    }
    .override {
      color: red; /* Ghi đè color kế thừa */
    }
  </style>
</head>
<body>
  <div class="container">
    <p class="child">Đoạn văn kế thừa font và màu.</p>
    <p class="override">Đoạn văn ghi đè màu.</p>
  </div>
```

```
</body>
</html>
```

Ví dụ nâng cao 1: Kế thừa với `inherit`

Buộc một phần tử con kế thừa thuộc tính từ cha.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Inheritance Advanced Example 1</title>
  <style>
    .parent {
      font-family: Helvetica, sans-serif;
      color: #007bff;
      text-align: center;
    }
    .child {
      font-family: inherit; /* Kế thừa font-family từ parent */
      color: inherit; /* Kế thừa color từ parent */
    }
    .child-override {
      font-family: Georgia, serif; /* Ghi đè font-family */
    }
  </style>
</head>
<body>
  <div class="parent">
    <p class="child">Kế thừa font và màu từ parent.</p>
    <p class="child-override">Ghi đè font, nhưng vẫn kế thừa màu.</p>
  </div>
</body>
</html>
```

Ví dụ nâng cao 2: Sử dụng `initial` và `unset`

Sử dụng `initial` và `unset` để kiểm soát kế thừa.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Inheritance Advanced Example 2</title>
  <style>
    body {
      font-family: Arial, sans-serif;
```

```
      color: navy;
    }
    .container {
      color: green; /* Ghi đè color của body */
    }
    .child-initial {
      color: initial; /* Đặt lại về giá trị mặc định (thường là black) */
    }
    .child-unset {
      color: unset; /* Kế thừa từ container (green) */
    }
  </style>
</head>
<body>
  <div class="container">
    <p class="child-unset">Kế thừa màu xanh từ container.</p>
    <p class="child-initial">Màu đen mặc định (initial).</p>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Nhất quán giao diện**: Đảm bảo các phần tử con (như văn bản, danh sách) có font và màu sắc thống nhất với cha.
- **Giảm mã lặp**: Không cần định nghĩa lại `font-family` hoặc `color` cho từng phần tử con.
- **Tùy chỉnh linh hoạt**: Sử dụng `inherit`, `initial`, hoặc `unset` để kiểm soát kiểu dáng trong các tình huống cụ thể.

Lưu ý

- Không phải tất cả thuộc tính đều kế thừa; kiểm tra tài liệu MDN để biết thuộc tính nào kế thừa tự nhiên.
- Kế thừa có thể bị ghi đè bởi tính ưu tiên (specificity) hoặc `!important`.

---

2. Tính ưu tiên (Specificity)

Định nghĩa

Tính ưu tiên (specificity) là cơ chế CSS sử dụng để xác định quy tắc nào được áp dụng khi nhiều quy tắc cạnh tranh (xung đột) cho cùng một phần tử.

Cách hoạt động

- **Cơ chế**: Trình duyệt tính toán điểm specificity của mỗi selector để quyết định quy tắc nào có ưu tiên cao hơn. Nếu có xung đột, quy tắc với specificity cao hơn sẽ được áp dụng.

- **Cách tính specificity**:
  - Specificity được biểu diễn bằng 4 giá trị (a, b, c, d):
    - **a**: Số lượng ID selectors (`#id`).
    - **b**: Số lượng class selectors (`.class`), pseudo-classes (`:hover`), và attribute selectors (`[type="text"]`).
    - **c**: Số lượng type selectors (như `div`, `p`) và pseudo-elements (`::before`).
    - **d**: Không tính (dùng để so sánh khi các giá trị khác bằng nhau, như inline styles hoặc `!important`).
  - Công thức tính: `(a, b, c)` được so sánh từ trái sang phải.
    - Ví dụ: `#id .class div` có specificity `(1, 1, 1)`; `.class div` có specificity `(0, 1, 1)`.
- **Quy tắc ưu tiên**:
  1. Inline styles (`style="..."`) có ưu tiên cao nhất (specificity `(1, 0, 0, 0)`).
  2. `!important` vượt qua tất cả (trừ inline styles với `!important`).
  3. Nếu specificity bằng nhau, quy tắc xuất hiện sau trong mã CSS được áp dụng.
- **Cơ chế trình duyệt**: Trình duyệt duyệt qua tất cả các quy tắc CSS, tính specificity, và áp dụng quy tắc có điểm cao nhất.

Cú pháp

```
/* Specificity thấp */
p {
  color: blue;
}

/* Specificity cao hơn */
.class-name {
  color: red;
}

/* Specificity cao nhất */
#id-name {
  color: green;
}

/* !important vượt qua tất cả */
p {
  color: black !important;
}
```

Ví dụ minh họa

Ví dụ cơ bản

So sánh specificity của các selector.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Specificity Basic Example</title>
  <style>
    p {
      color: blue; /* Specificity: (0, 0, 1) */
    }
    .text {
      color: red; /* Specificity: (0, 1, 0) */
    }
    #main-text {
      color: green; /* Specificity: (1, 0, 0) */
    }
  </style>
</head>
<body>
  <p id="main-text" class="text">Màu xanh lá (ID có specificity cao
nhất).</p>
</body>
</html>
```

Ví dụ nâng cao 1: Xung đột selector phức tạp

So sánh specificity với các selector phức tạp.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Specificity Advanced Example 1</title>
  <style>
    .container p {
      color: blue; /* Specificity: (0, 1, 1) */
    }
    .container .text {
      color: red; /* Specificity: (0, 2, 0) */
    }
    #main .container p.text {
      color: green; /* Specificity: (1, 2, 1) */
    }
  </style>
</head>
<body>
  <div id="main" class="container">
```

```
    <p class="text">Màu xanh lá (selector #main .container p.text có
specificity cao nhất).</p>
  </div>
</body>
</html>
```

Ví dụ nâng cao 2: Sử dụng !important và thứ tự mã

Kiểm tra ưu tiên khi specificity bằng nhau và sử dụng `!important`.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Specificity Advanced Example 2</title>
  <style>
    .text {
      color: blue; /* Specificity: (0, 1, 0) */
    }
    .text {
      color: red; /* Specificity: (0, 1, 0), thắng vì xuất hiện sau */
    }
    .text-important {
      color: green !important; /* Vượt qua tất cả do !important */
    }
  </style>
</head>
<body>
  <p class="text text-important">Màu xanh lá (do !important).</p>
  <p class="text">Màu đỏ (quy tắc sau thắng khi specificity bằng
nhau).</p>
</body>
</html>
```

Ứng dụng thực tế

- **Quản lý CSS phức tạp**: Specificity giúp xác định rõ ràng quy tắc nào được áp dụng trong các dự án lớn.
- **Ghi đè kiểu dáng**: Sử dụng selector cụ thể hơn để ghi đè kiểu dáng mặc định (ví dụ: tùy chỉnh giao diện framework như Bootstrap).
- **Debug CSS**: Hiểu specificity giúp tìm ra lý do tại sao một kiểu dáng không được áp dụng.

Lưu ý

- **Tránh lạm dụng `!important`**: Dùng `!important` chỉ khi thực sự cần thiết, vì nó làm khó bảo trì mã.

- **Giữ specificity thấp**: Sử dụng selector đơn giản (như class) thay vì ID hoặc selector phức tạp để dễ ghi đè sau này.
- **Kiểm tra specificity**: Sử dụng công cụ như DevTools (Chrome/Firefox) để kiểm tra quy tắc nào đang được áp dụng.

Kết hợp Inheritance và Specificity

Cách hoạt động

- **Kế thừa ưu tiên trước**: Nếu một thuộc tính kế thừa từ cha và không có quy tắc nào ghi đè, giá trị kế thừa được áp dụng.
- **Specificity giải quyết xung đột**: Khi nhiều quy tắc áp dụng cho cùng một phần tử, specificity quyết định quy tắc nào thắng. Nếu specificity bằng nhau, quy tắc xuất hiện sau thắng.
- **!important vượt qua tất cả**: Một quy tắc với `!important` sẽ bỏ qua cả kế thừa và specificity (trừ inline styles với `!important`).

Ví dụ minh họa

Kết hợp kế thừa và specificity để định dạng văn bản.

```
<!DOCTYPE html>
<html lang="vi">
<head>
  <meta charset="UTF-8">
  <title>Inheritance and Specificity Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      color: navy; /* Kế thừa cho tất cả phần tử con */
    }
    .container {
      color: green; /* Kế thừa cho con, specificity (0, 1, 0) */
    }
    #main-text {
      color: red; /* Specificity (1, 0, 0), ghi đè kế thừa */
    }
    .text-important {
      color: black !important; /* Vượt qua tất cả */
    }
  </style>
</head>
<body>
  <div class="container">
    <p>Kế thừa màu xanh từ .container.</p>
    <p id="main-text">Màu đỏ do ID có specificity cao.</p>
```

```
    <p class="text-important">Màu đen do !important.</p>
  </div>
</body>
</html>
```

Ứng dụng thực tế

- **Theme website**: Kết hợp kế thừa để áp dụng font và màu sắc mặc định, sử dụng specificity để tùy chỉnh các phần tử cụ thể.
- **Framework CSS**: Ghi đè kiểu dáng mặc định của Bootstrap hoặc Tailwind bằng selector có specificity cao hơn.
- **Debug và bảo trì**: Hiểu cả hai cơ chế giúp nhanh chóng tìm ra lỗi khi kiểu dáng không được áp dụng như mong muốn.

Bài tập thực hành

1. **Kế thừa**:
   - Tạo một trang HTML với một div cha có `font-family: Helvetica` và `color: blue`. Thêm 3 đoạn văn con, trong đó một đoạn ghi đè `color` bằng `red`.
   - Sử dụng `inherit` để buộc một đoạn văn con kế thừa `font-family` từ một div khác.
2. **Specificity**:
   - Tạo một đoạn văn với 3 quy tắc CSS có specificity khác nhau (type, class, ID). Kiểm tra xem màu nào được áp dụng.
   - Tạo một nút với 2 quy tắc class có specificity bằng nhau, thêm `!important` vào một quy tắc và quan sát kết quả.
3. **Kết hợp**:
   - Tạo một giao diện với một div cha (có `color` và `font-family`) và các phần tử con. Sử dụng selector với specificity cao để ghi đè màu sắc cho một phần tử con, và `!important` cho một phần tử khác.

Tài nguyên bổ sung

- **MDN Web Docs**: Tài liệu về kế thừa (developer.mozilla.org/en-US/docs/Web/CSS/inheritance) và specificity (developer.mozilla.org/en-US/docs/Web/CSS/Specificity).
- **CSS Tricks**: Bài viết về specificity (css-tricks.com/specifics-on-css-specificity).
- **WebAIM**: Công cụ kiểm tra tương phản và accessibility (webaim.org/resources/contrastchecker).
- **DevTools**: Sử dụng Chrome/Firefox DevTools để kiểm tra quy tắc CSS được áp dụng.