СОВРЕМЕННЫЕ МЕТОДЫ ВЕРСТКИ ВЕБ ПРИЛОЖЕНИЙ
Кравцов Е.П.

Кравцов Евгений Павлович– старший разработчик программного обеспечения, Sberdevices, г. Москва

Аннотация: *в статье проводится детальный обзор новейших возможностей CSS, которые были введены для улучшения процесса верстки веб-страниц. Освещаются такие аспекты, как CSS Grid Layout, переменные CSS, функции min(), max() и clamp(), а также поддержка темной темы через prefers-color-scheme. Также рассматриваются методы для создания более динамичных интерфейсов с помощью CSS Scroll Snap и CSS Transitions. Статья представляет примеры кода, демонстрирующие применение новых свойств и селекторов, что позволяет разработчикам на практике оценить их преимущества и возможности интеграции в современные веб-проекты. В завершение, обсуждается, как эти инновации могут способствовать повышению гибкости и доступности веб-страниц.*
Ключевые слова: *CSS Grid Layout, CSS переменные, CSS функции, медиа-запросы, темная тема, адаптивный дизайн, CSS Scroll Snap, CSS Transitions, верстка веб-страниц, доступность веб-интерфейсов.*

Для веб-разработчиков, постоянно следящих за последними тенденциями и обновлениями в технологиях, CSS (Cascading Style Sheets) остается одним из наиболее динамично развивающихся инструментов. Появление новых свойств CSS может значительно упростить создание более чистого, более функционального и интерактивного пользовательского интерфейса. В этой статье мы рассмотрим некоторые из самых важных и обсуждаемых новинок в мире CSS, которые были представлены в последнее время. Эти свойства предлагают улучшенные возможности для анимации, более гибкое управление макетами, а также усиленный контроль над визуальным оформлением элементов на веб-страницах.

Одним из таких новых свойств является content-visibility [1], предназначенное для оптимизации производительности загрузки контента на веб-страницах. Это свойство позволяет браузерам пропускать рендеринг содержимого, которое не находится во вьюпорте пользователя, что существенно ускоряет загрузку страницы и улучшает производительность без потери доступности контента. Свойство content-visibility предлагает несколько значений, каждое из которых обеспечивает различные уровни видимости и рендеринга:

auto: Это значение указывает браузеру пропускать рендеринг элемента и его потомков, если они не видны в области просмотра. Как только элемент приближается к видимой области, браузер начинает его рендеринг, обеспечивая таким образом необходимую производительность без задержек.

visible: Элемент и его потомки всегда рендерятся независимо от того, находятся ли они в области просмотра. Это значение аналогично стандартному поведению CSS без использования content-visibility.

hidden: Браузер полностью пропускает рендеринг элемента и его потомков, даже если они находятся в области просмотра. Это может быть полезно для временного скрытия определенных частей интерфейса без удаления их из DOM-дерева [2].

Преимущества

Улучшенная загрузка страницы: Так как браузер рендерит только видимые элементы, время загрузки страницы сокращается. Это особенно заметно на страницах с большим объемом контента, таких как длинные статьи или фотогалереи.

Экономия ресурсов: Пропуск рендеринга невидимого контента позволяет браузеру экономить ресурсы, такие как CPU и память, что особенно актуально для устройств с ограниченными возможностями.

Плавное прокручивание: Уменьшение количества рендеринга на начальном этапе загрузки страницы способствует более плавному прокручиванию и улучшению пользовательского опыта.

Псевдоселектор :is()

Псевдоселектор :is() в CSS представляет собой полезный инструмент для упрощения и улучшения читаемости таблиц стилей. Этот селектор позволяет группировать различные селекторы, которые имеют общие стили, тем самым уменьшая необходимость в повторении одних и тех же CSS правил. Функционал :is() особенно полезен в больших и сложных проектах, где поддержание чистоты и организованности кода становится критически важным.

Принцип работы

Псевдоселектор :is() принимает список селекторов как аргументы и применяет стили к элементам, которые соответствуют хотя бы одному из этих селекторов [3]. С его помощью можно значительно

сократить количество кода, например, объединив селекторы с общими стилями в один блок. Это не только упрощает поддержку кода, но и делает его более гибким при изменениях дизайна.

Пример использования

Допустим, вы хотите применить одинаковые стили к заголовкам h1, h2, h3, а также к параграфам с классом .important. Без использования :is(), вам пришлось бы написать несколько отдельных правил:

```
css
h1 {
  font-size: 24px;
  color: blue;
}
h2 {
  font-size: 24px;
  color: blue;
}
h3 {
  font-size: 24px;
  color: blue;
}

.important {
  font-size: 24px;
  color: blue;
}
```

С использованием :is(), код упрощается до одного блока стилей:

```
css
:is(h1, h2, h3, .important) {
  font-size: 24px;
  color: blue;
}
```

Этот пример показывает, как :is() может сделать CSS более компактным и легким для поддержки. Стили внутри блока :is() применяются ко всем элементам, которые соответствуют любому из перечисленных селекторов, делая код более чистым и организованным.

Новое свойство CSS: container

Одним из перспективных нововведений в CSS является свойство container, которое является частью спецификации CSS Container Queries. Это свойство позволяет создавать компоненты, которые адаптируют свой дизайн и макет на основе размеров их собственного контейнера, а не весь вьюпорт. Таким образом, container призвано улучшить модульность и реюзабельность компонентов на веб-страницах, обеспечивая более тонкий контроль над адаптивным дизайном.

Пример использования

Представим, что у вас есть карточка продукта, которая должна изменять свой макет в зависимости от размера контейнера, в котором она находится. С помощью свойства container, вы можете указать контейнер как "контейнерный контекст", и затем использовать контейнерные запросы для определения стилей внутри этого контекста.

Для начала, вы должны указать, по какому параметру контейнера будут применяться запросы. Это может быть размер (size), стиль (style), или оба:

```
css
.card {
  container-type: inline-size; /* Создает контейнерный контекст по горизонтальному размеру */
}

.card {
  width: 100%;
  padding: 1rem;
  box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}

/* Контейнерный запрос, который применяется, когда минимальная ширина контейнера 300px */
@container (min-width: 300px) {
```

```
    .card {
      padding: 2rem;
      display: flex;
      flex-direction: row;
    }

    .card-image {
      flex: 0 0 auto;
      width: 30%;
      margin-right: 20px;
    }

    .card-content {
      flex: 1;
      display: block;
    }
  }
```

В этом примере, карточка .card объявляется как контейнер, реагирующий на изменения своей ширины. Если ширина карточки превышает 300 пикселей, применяются новые стили, делая её содержимое более горизонтально ориентированным, с картинкой слева и описанием справа. Это демонстрирует гибкость container, позволяя карточке адаптироваться к своему контейнеру и не зависеть от размера вьюпорта.

Селектор :has() в CSS: создание контекстно-зависимых стилей

Селектор :has() представляет собой мощное дополнение к CSS, позволяя разработчикам стилизовать элементы на основе наличия или отсутствия определённых потомков или сиблингов. Это свойство значительно расширяет возможности каскадирования стилей, позволяя создавать более динамичные и адаптивные интерфейсы без использования JavaScript.

Принцип работы

Селектор :has() является "родственным" селектором, что означает, что он проверяет наличие одного или более элементов внутри элемента, к которому применяется стиль [4]. Это позволяет стилизовать родительский элемент на основе его содержимого, что до появления :has() было возможно только с помощью JavaScript.

Пример использования

Допустим, вы хотите стилизовать только те списки, которые содержат элементы с классом .important. Без :has(), вам бы пришлось использовать JavaScript для добавления стилей или классов к таким спискам. С использованием :has(), вы можете легко добавить стили непосредственно через CSS:

```
css
Copy code
ul:has(li.important) {
  border: 2px solid red;
  padding: 10px;
}
```

В этом примере, все списки <ul>, содержащие хотя бы один элемент <li> с классом .important, будут иметь красную границу и дополнительный отступ. Это простой способ динамически изменять стиль элементов на основе их содержимого.

Применение

Селектор :has() идеально подходит для случаев, когда нужно стилизовать контейнер на основе его содержимого. Например, если вы хотите изменить стиль заголовков, которые содержат ссылки, или подсветить формы, содержащие поля с ошибками, :has() сделает эти задачи значительно проще. Однако стоит отметить, что на момент написания этой статьи :has() все еще находится на стадии экспериментального использования во многих браузерах, поэтому рекомендуется проверять поддержку и готовность к использованию в продакшене.

Улучшенная гибкость в CSS с помощью селектора :where()

Селектор :where() в CSS является относительно новым добавлением, который предлагает уникальную возможность для стилизации элементов без увеличения специфичности селекторов. Это особенно полезно в крупных проектах, где поддержание легкости и гибкости CSS кода может стать сложной задачей[5]. Селектор :where() позволяет группировать различные селекторы и применять к ним стили, не влияя на общую специфичность, что упрощает переопределение стилей в будущем.

Принцип работы

Селектор :where() функционирует аналогично селектору :is(), позволяя включать в себя несколько селекторов. Однако ключевое отличие заключается в том, что :where() не добавляет вес специфичности к селекторам, включенным в него. Это значит, что любые стили, применяемые через :where(), могут быть легко переопределены другими селекторами, даже если они менее специфичны.

Пример использования

Допустим, вы хотите применить стиль к заголовкам первого уровня, второго и третьего, но также хотите иметь возможность легко переопределить эти стили в определенных разделах сайта. С использованием :where(), ваш CSS код мог бы выглядеть так:

```
css
:where(h1, h2, h3) {
  color: gray;
  font-weight: normal;
}
```

В этом примере, все заголовки h1, h2, и h3 будут иметь серый цвет и нормальное начертание шрифта. Но если вы решите, что заголовки в определенном классе .special должны быть жирными и синими, вы можете легко добавить новое правило:

```
css
.special h1, .special h2, .special h3 {
  color: blue;
  font-weight: bold;
}
```

Благодаря нулевой специфичности :where(), переопределение стилей происходит без трудностей, даже если новый селектор менее специфичен по сравнению с обычными селекторами.

Применение

Использование :where() особенно ценно в библиотеках компонентов и больших CSS фреймворках, где необходимо предоставить пользователям возможность легкого переопределения стилей без конфликтов и необходимости увеличивать специфичность селекторов. Это помогает поддерживать чистоту и организованность кода, упрощая его поддержку и модификацию.

Список литературы

1. Анимация и переход с помощью content-visibility // https://developer.mozilla.org - [Электронный ресурс]: URL: https://developer.mozilla.org/en-US/docs/Web/CSS/content-visibility (дата обращения: 01.04.2024)
2. Использование CSS-видимости контента для повышения производительности рендеринга // https://blog.logrocket.com - [Электронный ресурс]: URL: https://blog.logrocket.com/using-css-content-visibility-boost-rendering-performance/ (дата обращения: 03.04.2024)
3. Как работают псевдоселекторы :is, :where и :has // https://www.cat-in-web.ru - [Электронный ресурс]: URL:https://www.cat-in-web.ru/is-where-has/ (дата обращения: 03.04.2024)
4. Уникальный селектор, позволяющий стилизовать родителя при наличии конкретного ребёнка. // https://doka.guide/ - [Электронный ресурс]: URL: https://doka.guide/css/has/ (дата обращения: 15.04.2024)
5. Селектор :where // https://css-tricks.com/ - [Электронный ресурс]: https://css-tricks.com/almanac/selectors/w/where/ (дата обращения: 20.04.2024)