1.11.12. Скрипты

Опубликовано: 23 декабря 2021 Обновлено: 27 января 2022 0

Данная спецификация описывает элементы, которые делают сайты и веб-приложения интерактивными, динамически добавляют содержимое и участвуют в создании веб-компонентов.

HTML-элементы, поддерживающие скрипт

СОДЕРЖАНИЕ:

1. Элемент <script>

Категории контента: мета содержимое; потоковое содержимое; текстовое содержимое; элементы, поддерживающие скрипт.

Контекст, в котором этот элемент может быть использован: где ожидается мета содержимое; текстовое содержимое; элементы, поддерживающие скрипт.

Пропуск тегов: ни один из тегов не может быть пропущен.

Для элемента доступны глобальные атрибуты, а также атрибуты, приведенные в таблице:

ТАБЛИЦА 1. АТРИБУТЫ ЭЛЕМЕНТА <SCRIPT>

Атрибут	Описание, принимаемое значение
`src`	Задает URL-адрес ресурса. **Синтаксис:** `src="js/main.js"`
`type`	Позволяет настроить MIME-тип представленного скрипта. **Разрешенные значения:** Пропуск атрибута, установка его на пустую строку или установка на соответствие JavaScript означает, что сценарий является классическим. На классические сценарии влияют атрибуты `async` и `defer` , но только если установлен атрибут `src` . В этом случае атрибут следует опускать вместо его избыточной установки. `module` — указывает, что файл является модулем. Один файл содержит один модуль. Каждый модуль — это фрагмент кода, который выполняется после загрузки. В этом коде могут быть объявления (объявления переменных, объявления функций и т.д.). По умолчанию эти объявления остаются локальными для модуля. Вы можете пометить некоторые из них как экспортируемые, тогда другие модули смогут их импортировать. Модуль может импортировать объявления из других модулей. Он обращается к этим модулям через спецификаторы модулей. Даже если модуль импортируется несколько раз, существует только один его «экземпляр». Такой подход к модулям позволяет избежать глобальных переменных, единственное, что является глобальным, — это спецификаторы модуля. На модули не влияет атрибут `defer` , но на них влияет атрибут `async` (независимо от состояния атрибута `src`). Установка для атрибута любого другого значения означает, что сценарий представляет собой **блок данных**, который не обрабатывается.

	Синтаксис: `type="text/javascript"` `type="module"`
`nomodule`	Логический атрибут. Указывает, что скрипт не должен выполняться в современных браузерах, поддерживающих ES2015 модули, то есть элемент `<script>` с атрибутом `nomodule` будет игнорироваться. **Синтаксис:** `nomodule`
`async`	Логический атрибут, указывает браузеру, что скрипт должен быть выполнен после того, как он будет получен параллельно с обработкой исходного кода страницы. **Синтаксис:** `async`
`defer`	Логический атрибут, указывает браузеру, что скрипт должен быть выполнен после того, как HTML-документ будет полностью разобран. **Синтаксис:** `defer`
`crossorigin`	Позволяет загружать видео в `<canvas>` с ресурсов другого домена с помощью CORS-запросов. Видеофайлы, загруженные с помощью CORS-запросов, могут быть использованы повторно. **Разрешенные значения:** `anonymous` — запрос выполняется с помощью HTTP-заголовка, при этом учетные данные не передаются (cookie, сертификат X.509, логин/пароль для базовой аутентификации по HTTP). Если сервер не даёт учетные данные серверу, с которого запрашивается контент, то изображение будет испорчено и его использование будет ограничено. `use-credentials` — запрос выполняется с передачей учетных данных. **Синтаксис:** `crossorigin="anonymous"`
`integrity`	Атрибут содержит криптографический хэш файла. Используя алгоритм криптографического хеширования, браузер проверяет целостность данных загружаемого файла и отсутствие вредоносного содержимого. Браузер сравнивает скрипт с ожидаемым хешем, и если он не соответствует связанному с ним значению `integrity`, то скрипт не выполняется. В этом случае браузер возвращает сетевую ошибку, указывающую на то, что выборка этого скрипта завершилась неудачно. Значение атрибута состоит из строки, которая включает префикс, указывающий на конкретный алгоритм хеширования (в настоящее время разрешенные префиксы — `sha256`, `sha384` и `sha512`), за которым следует тире, и заканчивается фактическим хешем в кодировке base64. Значение атрибута может содержать несколько хешей, разделенных пробелом. Ресурс будет загружен, если он соответствует одному из этих хешей. Для генерации хеша можно воспользоваться SRI Hash Generator. **Синтаксис:** `integrity="sha384-Li9vy3DqF8tnTXuiaAJuML3ky+er10rcgNR/VqsVpcw+ThHmYcwiB1pbOxEbzJr7"` `integrity="sha384-Li9vy3DqF8tnTXuiaAJuML3ky+er10rcgNR/VqsVpcw+ThHmYcwiB1pbOxEbzJr7 sha384-+/M6kredJcxdsqkczBUjMLvqyHb1K/JThDXWsBVxMEeZHEaMKEOEct339VItX1zB"`
`referrerpolicy`	Устанавливает политику HTTP-заголовка — количество информации об исходной странице, с которой осуществлен переход на целевую страницу. **Разрешенные значения:** `no-referrer` — никакая информация о Referer не должна отправляться вместе с

	запросами к какому-либо источнику. `no-referrer-when-downgrade` — Referer указывается при выполнении запроса между HTTPS. `same-origin` — Referer указывается при выполнении запроса в пределах одного источника. `origin` — указывается только источник запроса (например, Referer документа `https://example.com/page.html` будет `https://example.com/`). `strict-origin` — при выполнении запроса между HTTPS и HTTPS указывается только источник запроса. `origin-when-cross-origin` — при выполнении запроса в пределах одного источника указывается полный URL, иначе указывается только источник. `strict-origin-when-cross-origin` — политика по умолчанию, отправляет полный URL-адрес при выполнении запроса в пределах одного источника, отправляет только источник при запросе между HTTPS и HTTPS и не отправляет заголовок между HTTPS и HTTP. `unsafe-url` — всегда указывается полный URL. **Синтаксис:** `referrerpolicy="origin"`

Элемент `<script>` позволяет включать исполняемый код в документы. Элемент не представляет содержимое для пользователя.

Классические сценарии и JavaScript-модули могут быть встроены в строку или импортированы из внешнего файла с использованием атрибута `src` .

HTML

```
<script nomodule defer src="classic-app-bundle.js"></script>
```

HTML

```
<script type="module">
 import { walkAllTextNodeDescendants } from "./dom-utils.mjs";

 const substitutions = new Map([
   ["witnesses", "these dudes I know"]
   ["allegedly", "kinda probably"]
   ["new study", "Tumblr post"]
   ["rebuild", "avenge"]
   ["space", "spaaace"]
   ["Google glass", "Virtual Boy"]
   ["smartphone", "Pokédex"]
   ["electric", "atomic"]
   ["Senator", "Elf-Lord"]
   ["car", "cat"]
   ["election", "eating contest"]
   ["Congressional leaders", "river spirits"]
   ["homeland security", "Homestar Runner"]
   ["could not be reached for comment", "is guilty and everyone knows it"]
```

```
]);

function substitute(textNode) {
  for (const [before, after] of substitutions.entries()) {
    textNode.data = textNode.data.replace(new RegExp(`\\b${before}\\b`, "ig"), after);
  }
}

walkAllTextNodeDescendants(document.body, substitute);
</script>
```

HTML

```
<script type="text/x-game-map">
........U.........e
o............A....e
.....A.....AAA....e
.A..AAA...AAAAA...e
</script>
```

2. Элемент <noscript>

Категории контента: мета содержимое, потоковое содержимое, текстовое содержимое.

Контекст, в котором этот элемент может быть использован: в разделе `<head>`; где ожидается текстовое содержимое.

Пропуск тегов: ни один из тегов не может быть пропущен.

Для элемента доступны глобальные атрибуты.

Если в браузере не отключен JavaScript, элемент `<noscript>` сам по себе ничего не представляет. Если JavaScript отключен — представляет свои дочерние элементы.

Элемент `<noscript>` используется для отображения альтернативного содержимого. Внутри раздела `<head>` элемент `<noscript>` должен содержать только элементы `<link>`, `<style>` и `<meta>`.

В теле документа элемент `<noscript>` может содержать любые элементы, кроме `<noscript>`.

Элемент `<noscript>` нельзя использовать в XML-документах.

HTML

```
<form action="calcSquare.php">
 <p>
  <label for=x>Number</label>:
  <input id="x" name="x" type="number">
 </p>
 <script>
  var x = document.getElementById('x');
  var output = document.createElement('p');
```

```
  output.textContent = 'Type a number; it will be squared right then!';
  x.form.appendChild(output);
  x.form.onsubmit = function () { return false; }
  x.oninput = function () {
    var v = x.valueAsNumber;
    output.textContent = v + ' squared is ' + v * v;
  };
 </script>
 <noscript>
  <input type=submit value="Calculate Square">
 </noscript>
</form>
```

3. Элемент <template>

Категории контента: мета содержимое; потоковое содержимое; текстовое содержимое; элементы, поддерживающие скрипт.

Контекст, в котором этот элемент может быть использован: где ожидается мета содержимое; где ожидается текстовое содержимое; где ожидаются элементы, поддерживающие скрипт; как дочерний элемент элемента `<colgroup>`, у которого нет атрибута `span`.

Пропуск тегов: ни один из тегов не может быть пропущен.

Для элемента доступны глобальные атрибуты.

Элемент `<template>` используется для объявления фрагментов HTML-разметки, которые можно клонировать и вставлять в документ с помощью сценария. При загрузке страницы элемент `<template>` не отображается. Содержимое элемента не является его дочерним элементом.

HTML

```
<template id="template">
   <p>Smile!</p>
</template>
<script>
   let num = 3;
   const fragment = document.getElementById('template').content.cloneNode(true);
   while (num-- > 1) {
      fragment.firstChild.before(fragment.firstChild.cloneNode(true));
      fragment.firstChild.textContent += fragment.lastChild.textContent;
   }
   document.body.appendChild(fragment);
</script>
```

4. Элемент <slot>

Категории контента: потоковое содержимое, текстовое содержимое.

Контекст, в котором этот элемент может быть использован: где ожидается текстовое содержимое.

Пропуск тегов: ни один из тегов не может быть пропущен.

Для элемента доступны глобальные атрибуты, а также атрибут `name`, задающий имя слота. При рендеринге страницы элемент `<slot>` будет заменен тем элементом, у которого есть атрибут `slot` со значением, равным значению атрибута `name` элемента `<slot>`.

Элемент `<slot>` определяет ячейку, представляя назначенные ему узлы, если таковые имеются, или свое содержимое в противном случае. Элемент создает дерево с ограниченной областью видимости (Shadow DOM), которое имеет собственный корень (Shadow root) и может иметь собственные стили с ограниченной областью действия.

Shadow DOM, или теневое дерево, — один из трех стандартов веб-компонентов. С его помощью изменяется отрисовка элемента без изменения DOM (`document.querySelector()` не будет возвращать узлы теневого дерева). Shadow DOM — это иерархия элементов, такая же, как иерархия любого HTML-документа, которая сосуществует с деревом документа и другими деревьями, но полностью независима от них. Помимо прочего, эта характеристика позволяет деревьям иметь свои собственные классы и идентификаторы, которые не мешают классам и идентификаторам других деревьев. Shadow DOM особенно полезен при создании пользовательских элементов.

Shadow root, или теневой корень, представляет собой фрагмент документа, который прикрепляется к элементу.

Элемент `<slot>` может быть пустыми или содержать резервное содержимое:

HTML

```
<!-- Слот по умолчанию. Если имеется более одного слота по умолчанию, используется первый. -->
<slot></slot>

<slot>fallback content</slot> <!-- Слот по умолчанию с резервным содержимым -->

<slot> <!-- Слот по умолчанию с резервным DOM -->
  <h2>Title</h2>
  <summary>Description text</summary>
</slot>
```

Можно создать именованный слот, комбинируя его с обычным слотом:

HTML

```
<style>
   ::slotted(span) {
      background: pink;
   }
   ::slotted(.content) {
      font-family: monospace;
   }
</style>
```

```
<div>
   <slot name="title"></slot>
   <hr>
   <slot></slot>
</div>
```

Затем использовать этот элемент следующим образом:

HTML

```
<my-info-box>
   <span slot="title">🔍 Fancy title</span>
   <p class="content">I'm going straight to the anonymous slot.</p>
</my-info-box>
```

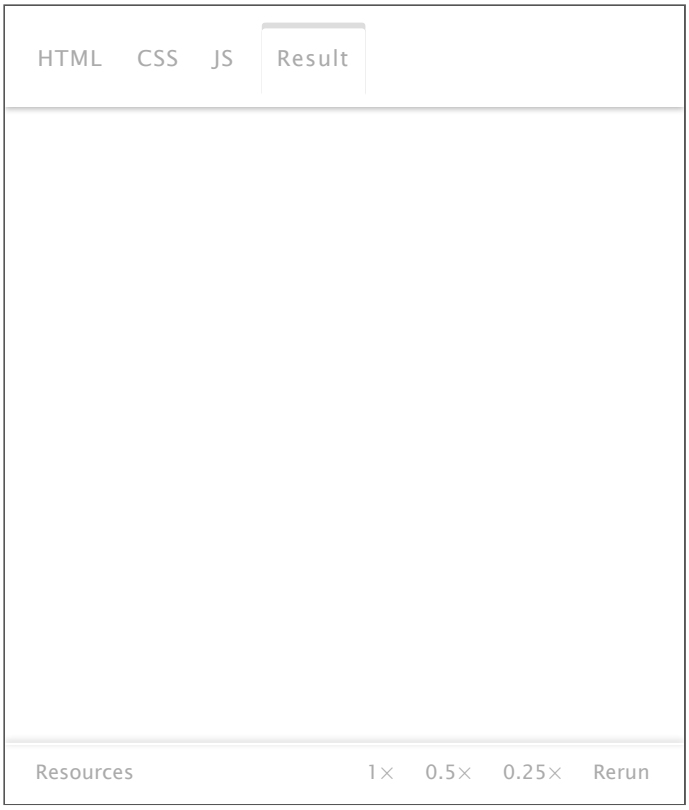
5. Элемент <canvas>

Категории контента: потоковое содержимое, текстовое содержимое, встраиваемое содержимое, видимое содержимое.

Контекст, в котором этот элемент может быть использован: где ожидается встраиваемое содержимое.

Пропуск тегов: ни один из тегов не может быть пропущен.

Для элемента доступны глобальные атрибуты, а также атрибуты `width` и `height`, задающие ширину и высоту холста. По умолчанию ширина равна `300px`, а высота — `150px`. Размеры холста могут быть изменены с помощью CSS-свойств, а его изображение подчиняется свойству `object-fit`.



Элемент `<canvas>` определяет пустую растровую область для рисования. Он не имеет собственных объектов, только инструкции Canvas API о том, что рисовать на любом отдельном кадре. Canvas API в основном ориентирован на 2D-графику. Его можно использовать для рисования графики, анимации, игровой графики, визуализации данных, обработки фотографий и видео в реальном времени.

API WebGL, который также использует элемент `<canvas>`, рисует 2D- и 3D-графику с аппаратным ускорением.

Доступ к самому элементу `<canvas>` осуществляется в браузере через DOM с помощью метода `getElementById()`, но отдельные графические элементы, созданные на Canvas, недоступны для DOM.

Чтобы настроить хост для рисования, необходимо получить контекст рисования, для чего используется метод `getContext()`. Изначально у элемента нет привязанного контекста, поэтому его нужно указать.

Контекст определяет набор функций API и возможностей рисования. HTML-спецификация определяет контексты `2d` и `bitmaprenderer`. Спецификации WebGL определяют контексты `webgl` и `webgl2`. WebGPU определяет контекст `webgpu`.

HTML

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
    function draw() {
      var canvas = document.getElementById('canvas');
      if (canvas.getContext) {
        var ctx = canvas.getContext('2d');
```

```
        ctx.fillStyle = 'rgb(200, 0, 0)';
        ctx.fillRect(10, 10, 50, 50);

        ctx.fillStyle = 'rgba(0, 0, 200, 0.5)';
        ctx.fillRect(30, 30, 50, 50);
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

По материалам Scripting

Интерактивные элементы

HTML5-аудио

Поделиться: