

Раздел

[Документ](#)

Навигация по уроку

className и classList

Element style

Сброс стилей

Следите за единицами измерения


Вычисленные стили: `getComputedStyle`

Итого

Задачи (1)

Комментарии

Поделиться

[Редактировать на GitHub](#)[🏠](#) → [Браузер: документ, события, интерфейсы](#)  
→ [Документ](#) 15-го июля 2020

## Стили и классы

До того, как начнёте изучать способы работы со стилями и классами в JavaScript, есть одно важное правило. Надеемся, это достаточно очевидно, но мы все равно должны об этом упомянуть.

Как правило, существует два способа задания стилей для элемента:

1. Создать класс в CSS и использовать его: `<div class="...">`
2. Писать стили непосредственно в атрибуте `style`: `<div style="...">`.

JavaScript может менять и классы, и свойство `style`.

Классы – всегда предпочтительный вариант по сравнению со `style`. Мы должны манипулировать свойством `style` только в том случае, если классы «не могут справиться».

Например, использование `style` является приемлемым, если мы вычисляем координаты элемента динамически и хотим установить их из JavaScript:

```
1 let top = /* сложные расчёты */;  
2 let left = /* сложные расчёты */;  
3  
4 elem.style.left = left; // например, '123px', значение  
5 elem.style.top = top; // например, '456px'
```



В других случаях, например, чтобы сделать текст красным, добавить значок фона – описываем это в CSS и добавляем класс (JavaScript может это сделать). Это более гибкое и лёгкое в поддержке решение.



### className и classList

Изменение класса является одним из наиболее часто используемых действий в скриптах.

Когда-то давно в JavaScript существовало ограничение: зарезервированное слово типа `"class"` не могло быть свойством объекта. Это ограничение сейчас отсутствует, но в то время было невозможно иметь свойство `elem.class`.

Поэтому для классов было введено схожее свойство `"className"`: `elem.className` соответствует атрибуту `"class"`.

Например:

```
1 <body class="main page">  
2   <script>  
3     alert(document.body.className); // main page  
4   </script>  
5 </body>
```



Если мы присваиваем что-то `elem.className`, то это заменяет всю строку с классами. Иногда это то, что нам нужно, но часто мы хотим добавить/удалить один класс.

Для этого есть другое свойство: `elem.classList`.

`elem.classList` – это специальный объект с методами для добавления/удаления одного класса.

Например:



Раздел

[Документ](#)

Навигация по уроку

className и classList

Element style

Сброс стилей

Следите за единицами измерения

Вычисленные стили: getComputedStyle

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
1 <body class="main page">
2   <script>
3     // добавление класса
4     document.body.classList.add('article');
5
6     alert(document.body.className); // main page article
7   </script>
8 </body>
```

Так что мы можем работать как со строкой полного класса, используя `className`, так и с отдельными классами, используя `classList`.

Выбираем тот вариант, который нам удобнее.

Методы `classList`:

- `elem.classList.add/remove("class")` – добавить/удалить класс.
- `elem.classList.toggle("class")` – добавить класс, если его нет, иначе удалить.
- `elem.classList.contains("class")` – проверка наличия класса, возвращает `true/false`.

Кроме того, `classList` является перебираемым, поэтому можно перечислить все классы при помощи `for...of`:

```
1 <body class="main page">
2   <script>
3     for (let name of document.body.classList) {
4       alert(name); // main, затем page
5     }
6   </script>
7 </body>
```



## Element style

Свойство `elem.style` – это объект, который соответствует тому, что написано в атрибуте `"style"`. Установка стиля `elem.style.width="100px"` работает так же, как наличие в атрибуте `style` строки `width:100px`.

Для свойства из нескольких слов используется `camelCase`:

```
1 background-color => elem.style.backgroundColor
2 z-index          => elem.style.zIndex
3 border-left-width => elem.style.borderLeftWidth
```

Например:

```
1 document.body.style.backgroundColor = prompt('background-color');
```



### Свойства с префиксом

Стили с браузерным префиксом, например, `-moz-border-radius`, `-webkit-border-radius` преобразуются по тому же принципу: дефис означает заглавную букву.

Например:

```
1 button.style.MozBorderRadius = '5px';
2 button.style.WebkitBorderRadius = '5px';
```

## Сброс стилей

Иногда нам нужно добавить свойство стиля, а потом, позже, убрать его.

Раздел

Документ

Навигация по уроку

className и classList

Element style

Сброс стилей

Следите за единицами измерения

Вычисленные стили: getComputedStyle

Итого

Задачи (1)

Комментарии

Поделиться



Редактировать на GitHub



Например, чтобы скрыть элемент, мы можем задать `elem.style.display = "none"`.

Затем мы можем удалить свойство `style.display`, чтобы вернуться к первоначальному состоянию. Вместо `delete elem.style.display` мы должны присвоить ему пустую строку: `elem.style.display = ""`.

```
1 // если мы запустим этот код, <body> "мигнёт"
2 document.body.style.display = "none"; // скрыть
3
4 setTimeout(() => document.body.style.display = "", 1000)
```

Если мы установим в `style.display` пустую строку, то браузер применит CSS-классы и встроенные стили, как если бы такого свойства `style.display` вообще не было.

### **i** Полная перезапись `style.cssText`

Обычно мы используем `style.*` для присвоения индивидуальных свойств стиля. Нельзя установить список стилей как, например, `div.style="color: red; width: 100px"`, потому что `div.style` – это объект, и он доступен только для чтения.

Для задания нескольких стилей в одной строке используется специальное свойство `style.cssText`:

```
1 <div id="div">Button</div>
2
3 <script>
4   // можем даже устанавливать специальные флаги д
5   div.style.cssText=`color: red !important;
6     background-color: yellow;
7     width: 100px;
8     text-align: center;
9   `;
10
11   alert(div.style.cssText);
12 </script>
```

Это свойство редко используется, потому что такое присваивание удаляет все существующие стили: оно не добавляет, а заменяет их. Можно ненароком удалить что-то нужное. Но его можно использовать, к примеру, для новых элементов, когда мы точно знаем, что не удалим существующий стиль.

То же самое можно сделать установкой атрибута: `div.setAttribute('style', 'color: red...')`.

## Следите за единицами измерения

Не забудьте добавить к значениям единицы измерения.

Например, мы должны устанавливать `10px`, а не просто `10` в свойство `elem.style.top`. Иначе это не сработает:

```
1 <body>
2   <script>
3     // не работает!
4     document.body.style.margin = 20;
5     alert(document.body.style.margin); // '' (пустая ст
6
7     // сейчас добавим единицу измерения (px) - и зарабо
8     document.body.style.margin = '20px';
9     alert(document.body.style.margin); // 20px
10
11     alert(document.body.style.marginTop); // 20px
12     alert(document.body.style.marginLeft); // 20px
```

Раздел

[Документ](#)

Навигация по уроку

[className и classList](#)

[Element style](#)

[Сброс стилей](#)

[Следите за единицами измерения](#)

[Вычисленные стили: getComputedStyle](#)

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



```
13     </script>
14 </body>
```

Пожалуйста, обратите внимание, браузер «распаковывает» свойство `style.margin` в последних строках и выводит `style.marginLeft` и `style.marginTop` из него.

## Вычисленные стили: `getComputedStyle`

Итак, изменить стиль очень просто. Но как его *прочитать*?

Например, мы хотим знать размер, отступы, цвет элемента. Как это сделать?

**Свойство `style` оперирует только значением атрибута `"style"`, без учёта CSS-каскада.**

Поэтому, используя `elem.style`, мы не можем прочитать ничего, что приходит из классов CSS.

Например, здесь `style` не может видеть отступы:

```
1 <head>
2   <style> body { color: red; margin: 5px } </style>
3 </head>
4 <body>
5
6   Красный текст
7   <script>
8     alert(document.body.style.color); // пусто
9     alert(document.body.style.marginTop); // пусто
10  </script>
11 </body>
```

...Но что, если нам нужно, скажем, увеличить отступ на 20px? Для начала нужно его текущее значение получить.

Для этого есть метод: `getComputedStyle`.

Синтаксис:

```
1 getComputedStyle(element, [pseudo])
```

### **element**

Элемент, значения для которого нужно получить

### **pseudo**

Указывается, если нужен стиль псевдоэлемента, например `::before`. Пустая строка или отсутствие аргумента означают сам элемент.

Результат вызова – объект со стилями, похожий на `elem.style`, но с учётом всех CSS-классов.

Например:

```
1 <head>
2   <style> body { color: red; margin: 5px } </style>
3 </head>
4 <body>
5
6   <script>
7     let computedStyle = getComputedStyle(document.body)
8
9     // сейчас мы можем прочитать отступ и цвет
10
11     alert( computedStyle.marginTop ); // 5px
12     alert( computedStyle.color ); // rgb(255, 0, 0)
13  </script>
14
15 </body>
```

Раздел

[Документ](#)

Навигация по уроку

className и classList

Element style

Сброс стилей

Следите за единицами измерения

Вычисленные стили: getComputedStyle

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



## Вычисленное (computed) и окончательное (resolved) значения

Есть две концепции в CSS:

1. *Вычисленное* (computed) значение – это то, которое получено после применения всех CSS-правил и CSS-наследования. Например, `height:1em` или `font-size:125%`.
2. *Окончательное* (resolved) значение – непосредственно применяемое к элементу. Значения `1em` или `125%` являются относительными. Браузер берёт вычисленное значение и делает все единицы измерения фиксированными и абсолютными, например, `height:20px` или `font-size:16px`. Для геометрических свойств разрешённые значения могут иметь плавающую точку, например, `width:50.5px`.

Давным-давно `getComputedStyle` был создан для получения вычисленных значений, но оказалось, что окончательные значения гораздо удобнее, и стандарт изменился.

Так что, в настоящее время `getComputedStyle` фактически возвращает окончательное значение свойства, для геометрии оно обычно в пикселях.

## `getComputedStyle` требует полное свойство!

Для правильного получения значения нужно указать точное свойство. Например: `paddingLeft`, `marginTop`, `borderTopWidth`. При обращении к сокращённому: `padding`, `margin`, `border` – правильный результат не гарантируется.

Например, если есть свойства `paddingLeft/paddingTop`, то что мы получим вызывая `getComputedStyle(elem).padding`? Ничего, или, может быть, «сгенерированное» значение из известных внутренних отступов? Стандарта для этого нет.

Есть и другие несоответствия. Например, некоторые браузеры (Chrome) отображают `10px` в документе ниже, а некоторые (Firefox) – нет:

```
1 <style>
2   body {
3     margin: 10px;
4   }
5 </style>
6 <script>
7   let style = getComputedStyle(document.body);
8   alert(style.margin); // пустая строка в Firefox
9 </script>
```

Раздел

[Документ](#)

Навигация по уроку

className и classList

Element style

Сброс стилей

Следите за единицами измерения

Вычисленные стили: `getComputedStyle`

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



### Стили, применяемые к посещённым `:visited` ссылкам, скрываются!

Посещённые ссылки могут быть окрашены с помощью псевдокласса `:visited`.

Но `getComputedStyle` не даёт доступ к этой информации, чтобы произвольная страница не могла определить, посещал ли пользователь ту или иную ссылку, проверив стили.

JavaScript не видит стили, применяемые с помощью `:visited`. Кроме того, в CSS есть ограничение, которое запрещает в целях безопасности применять к `:visited` CSS-стили, изменяющие геометрию элемента. Это гарантирует, что нет обходного пути для «злой» страницы проверить, была ли ссылка посещена и, следовательно, нарушить конфиденциальность.

## Итого

Для управления классами существуют два DOM-свойства:

- `className` – строковое значение, удобно для управления всем набором классов.
- `classList` – объект с методами `add/remove/toggle/contains`, удобно для управления отдельными классами.

Чтобы изменить стили:

- Свойство `style` является объектом со стилями в формате `camelCase`. Чтение и запись в него работают так же, как изменение соответствующих свойств в атрибуте `"style"`. Чтобы узнать, как добавить в него `important` и делать некоторые другие редкие вещи – смотрите [документацию](#).
- Свойство `style.cssText` соответствует всему атрибуту `"style"`, полной строке стилей.

Для чтения окончательных стилей (с учётом всех классов, после применения CSS и вычисления окончательных значений) используется:

- Метод `getComputedStyle(elem, [pseudo])` возвращает объект, похожий по формату на `style`. Только для чтения.

## Задачи

### Создать уведомление

важность: 5

Напишите функцию `showNotification(options)`, которая создаёт уведомление: `<div class="notification">` с заданным содержимым. Уведомление должно автоматически исчезнуть через 1,5 секунды.

Пример объекта `options`:

```
1 // показывает элемент с текстом "Hello" рядом с правой
2 showNotification({
3   top: 10, // 10px от верхней границы окна (по умолчанию)
4   right: 10, // 10px от правого края окна (по умолчанию)
5   html: "Hello!", // HTML-уведомление
6   className: "welcome" // дополнительный класс для div
7 });
```

### [Демо в новом окне](#)

Используйте CSS-позиционирование для отображения элемента в заданных координатах. Исходный документ имеет необходимые стили.

[Открыть песочницу для задачи.](#)

Раздел

[Документ](#)

Навигация по уроку

className и classList

Element style

Сброс стилей

Следите за единицами измерения

Вычисленные стили: `getComputedStyle`

Итого

Задачи (1)

Комментарии

Поделиться



[Редактировать на GitHub](#)



Проводим [курсы по JavaScript и фреймворкам](#).



## Комментарии

перед тем как писать...

