Атрибуты и свойства

Когда браузер загружает страницу, он «читает» (также говорят: «парсит») HTML и генерирует из него DOM-объекты. Для узлов-элементов большинство стандартных HTML-атрибутов автоматически становятся свойствами DOM-объектов.

Например, для такого тега `<body id="page">` у DOM-объекта будет такое свойство `body.id="page"`.

Но преобразование атрибута в свойство происходит не один-в-один! В этой главе мы уделим внимание различию этих двух понятий, чтобы посмотреть, как работать с ними, когда они одинаковые и когда разные.

DOM-свойства

Ранее мы уже видели встроенные DOM-свойства. Их много. Но технически нас никто не ограничивает, и если этого мало – мы можем добавить своё собственное свойство.

DOM-узлы – это обычные объекты JavaScript. Мы можем их изменять.

Например, создадим новое свойство для `document.body`:

```
document.body.myData = {
  name: 'Caesar',
  title: 'Imperator'
};

alert(document.body.myData.title); // Imperator
```

Мы можем добавить и метод:

```
document.body.sayTagName = function() {
  alert(this.tagName);
};

document.body.sayTagName(); // BODY (значением "this" в
```

Также можно изменять встроенные прототипы, такие как `Element.prototype` и добавлять новые методы ко всем элементам:

```
Element.prototype.sayHi = function() {
  alert(`Hello, I'm ${this.tagName}`);
};

document.documentElement.sayHi(); // Hello, I'm HTML
document.body.sayHi(); // Hello, I'm BODY
```

Итак, DOM-свойства и методы ведут себя так же, как и обычные объекты JavaScript:

- Им можно присвоить любое значение.
- Они регистрозависимы (нужно писать `elem.nodeType`, не `elem.NoDeTyPe`).

HTML-атрибуты

В HTML у тегов могут быть атрибуты. Когда браузер парсит HTML, чтобы создать DOM-объекты для тегов, он распознаёт *стандартные* атрибуты и

создаёт DOM-свойства для них.

Таким образом, когда у элемента есть `id` или другой *стандартный* атрибут, создаётся соответствующее свойство. Но этого не происходит, если атрибут нестандартный.

Например:

```
<body id="test" something="non-standard">
  <script>
    alert(document.body.id); // test
    // нестандартный атрибут не преобразуется в свойств
    alert(document.body.something); // undefined
  </script>
</body>
```

Пожалуйста, учтите, что стандартный атрибут для одного тега может быть нестандартным для другого. Например, атрибут `"type"` является стандартным для элемента `<input>` (HTMLInputElement), но не является стандартным для `<body>` (HTMLBodyElement). Стандартные атрибуты описаны в спецификации для соответствующего класса элемента.

Мы можем увидеть это на примере ниже:

```
<body id="body" type="...">
  <input id="input" type="text">
  <script>
    alert(input.type); // text
    alert(body.type); // undefined: DOM-свойство не соз
  </script>
</body>
```

Таким образом, для нестандартных атрибутов не будет соответствующих DOM-свойств. Есть ли способ получить такие атрибуты?

Конечно. Все атрибуты доступны с помощью следующих методов:

- `elem.hasAttribute(name)` – проверяет наличие атрибута.
- `elem.getAttribute(name)` – получает значение атрибута.
- `elem.setAttribute(name, value)` – устанавливает значение атрибута.
- `elem.removeAttribute(name)` – удаляет атрибут.

Эти методы работают именно с тем, что написано в HTML.

Кроме этого, получить все атрибуты элемента можно с помощью свойства `elem.attributes` : коллекция объектов, которая принадлежит ко встроенному классу Attr со свойствами `name` и `value` .

Вот демонстрация чтения нестандартного свойства:

```
<body something="non-standard">
  <script>
    alert(document.body.getAttribute('something')); //
  </script>
</body>
```

У HTML-атрибутов есть следующие особенности:

- Их имена регистронезависимы (`id` то же самое, что и `ID`).
- Их значения всегда являются строками.

Расширенная демонстрация работы с атрибутами:

```
<body>
  <div id="elem" about="Elephant"></div>

  <script>
```


```
    alert( elem.getAttribute('About') ); // (1) 'Elepha

    elem.setAttribute('Test', 123); // (2), запись

    alert( elem.outerHTML ); // (3), посмотрим, есть ли

    for (let attr of elem.attributes) { // (4) весь спи
      alert( `${attr.name} = ${attr.value}` );
    }
  </script>
</body>
```

Пожалуйста, обратите внимание:

1. `getAttribute('About')` – здесь первая буква заглавная, а в HTML – строчная. Но это не важно: имена атрибутов регистронезависимы.
2. Мы можем присвоить что угодно атрибуту, но это станет строкой. Поэтому в этой строчке мы получаем значение `"123"`.
3. Все атрибуты, в том числе те, которые мы установили, видны в `outerHTML`.
4. Коллекция `attributes` является перебираемой. В ней есть все атрибуты элемента (стандартные и нестандартные) в виде объектов со свойствами `name` и `value`.

Синхронизация между атрибутами и свойствами

Когда стандартный атрибут изменяется, соответствующее свойство автоматически обновляется. Это работает и в обратную сторону (за некоторыми исключениями).

В примере ниже `id` модифицируется как атрибут, и можно увидеть, что свойство также изменено. То же самое работает и в обратную сторону:

```
<input>

<script>
  let input = document.querySelector('input');

  // атрибут => свойство
  input.setAttribute('id', 'id');
  alert(input.id); // id (обновлено)

  // свойство => атрибут
  input.id = 'newId';
  alert(input.getAttribute('id')); // newId (обновлено)
</script>
```

Но есть и исключения, например, `input.value` синхронизируется только в одну сторону – атрибут → значение, но не в обратную:

```
<input>

<script>
  let input = document.querySelector('input');

  // атрибут => значение
  input.setAttribute('value', 'text');
  alert(input.value); // text

  // свойство => атрибут
  input.value = 'newValue';
  alert(input.getAttribute('value')); // text (не обнов
</script>
```

В примере выше:

- Изменение атрибута `value` обновило свойство.

- Но изменение свойства не повлияло на атрибут.

Иногда эта «особенность» может пригодиться, потому что действия пользователя могут приводить к изменениям `value`, и если после этого мы захотим восстановить «оригинальное» значение из HTML, оно будет в атрибуте.

DOM-свойства типизированы

DOM-свойства не всегда являются строками. Например, свойство `input.checked` (для чекбоксов) имеет логический тип:

```
<input id="input" type="checkbox" checked> checkbox

<script>
  alert(input.getAttribute('checked')); // значение атр
  alert(input.checked); // значение свойства: true
</script>
```

Есть и другие примеры. Атрибут `style` – строка, но свойство `style` является объектом:

```
<div id="div" style="color:red;font-size:120%">Hello</d

<script>
  // строка
  alert(div.getAttribute('style')); // color:red;font-s

  // объект
  alert(div.style); // [object CSSStyleDeclaration]
  alert(div.style.color); // red
</script>
```

Хотя большинство свойств, всё же, строки.

При этом некоторые из них, хоть и строки, могут отличаться от атрибутов. Например, DOM-свойство `href` всегда содержит *полный* URL, даже если атрибут содержит относительный URL или просто `#hash`.

Ниже пример:

```
<a id="a" href="#hello">link</a>
<script>
  // атрибут
  alert(a.getAttribute('href')); // #hello

  // свойство
  alert(a.href ); // полный URL в виде http://site.com/
</script>
```

Если же нужно значение `href` или любого другого атрибута в точности, как оно записано в HTML, можно воспользоваться `getAttribute`.

Нестандартные атрибуты, dataset

При написании HTML мы используем много стандартных атрибутов. Но что насчёт нестандартных, пользовательских? Давайте посмотрим, полезны они или нет, и для чего они нужны.

Иногда нестандартные атрибуты используются для передачи пользовательских данных из HTML в JavaScript, или чтобы «помечать» HTML-элементы для JavaScript.

Как тут:

```
<!-- пометить div, чтобы показать здесь поле "name" -->
<div show-info="name"></div>
```


```
<!-- а здесь возраст "age" -->
<div show-info="age"></div>

<script>
  // код находит элемент с пометкой и показывает запрош
  let user = {
    name: "Pete",
    age: 25
  };

  for(let div of document.querySelectorAll('[show-info]
    // вставить соответствующую информацию в поле
    let field = div.getAttribute('show-info');
    div.innerHTML = user[field]; // сначала Pete в name
  }
</script>
```

Также они могут быть использованы, чтобы стилизовать элементы.

Например, здесь для состояния заказа используется атрибут `order-state`:

```
<style>
  /* стили зависят от пользовательского атрибута "order
  .order[order-state="new"] {
    color: green;
  }

  .order[order-state="pending"] {
    color: blue;
  }

  .order[order-state="canceled"] {
    color: red;
  }
</style>

<div class="order" order-state="new">
  A new order.
</div>

<div class="order" order-state="pending">
  A pending order.
</div>

<div class="order" order-state="canceled">
  A canceled order.
</div>
```

Почему атрибут может быть предпочтительнее таких классов, как `.order-state-new`, `.order-state-pending`, `order-state-canceled`?

Это потому, что атрибутом удобнее управлять. Состояние может быть изменено достаточно просто:

```
// немного проще, чем удаление старого/добавление новог
div.setAttribute('order-state', 'canceled');
```

Но с пользовательскими атрибутами могут возникнуть проблемы. Что если мы используем нестандартный атрибут для наших целей, а позже он появится в стандарте и будет выполнять какую-то функцию? Язык HTML живой, он растёт, появляется больше атрибутов, чтобы удовлетворить потребности разработчиков. В этом случае могут возникнуть неожиданные эффекты.

Чтобы избежать конфликтов, существуют атрибуты вида data-*.

Все атрибуты, начинающиеся с префикса «data-», зарезервированы для использования программистами. Они доступны в свойстве `dataset`.

Например, если у `elem` есть атрибут `"data-about"`, то обратиться к нему можно как `elem.dataset.about`.

Как тут:

```
<body data-about="Elephants">
<script>
  alert(document.body.dataset.about); // Elephants
</script>
```

Атрибуты, состоящие из нескольких слов, к примеру `data-order-state`, становятся свойствами, записанными с помощью верблюжьей нотации: `dataset.orderState`.

Вот переписанный пример «состояния заказа»:

```
<style>
  .order[data-order-state="new"] {
    color: green;
  }

  .order[data-order-state="pending"] {
    color: blue;
  }

  .order[data-order-state="canceled"] {
    color: red;
  }
</style>

<div id="order" class="order" data-order-state="new">
  A new order.
</div>

<script>
  // чтение
  alert(order.dataset.orderState); // new

  // изменение
  order.dataset.orderState = "pending"; // (*)
</script>
```

Использование `data-*` атрибутов – валидный, безопасный способ передачи пользовательских данных.

Пожалуйста, примите во внимание, что мы можем не только читать, но и изменять data-атрибуты. Тогда CSS обновит представление соответствующим образом: в примере выше последняя строка `(*)` меняет цвет на синий.

Итого

- Атрибуты – это то, что написано в HTML.
- Свойства – это то, что находится в DOM-объектах.

Небольшое сравнение:

	Свойства	Атрибуты
Тип	Любое значение, стандартные свойства имеют типы, описанные в спецификации	Строка
Имя	Имя регистрозависимо	Имя регистронезависимо

Методы для работы с атрибутами:

- `elem.hasAttribute(name)` – проверить на наличие.
- `elem.getAttribute(name)` – получить значение.

- `elem.setAttribute(name, value)` – установить значение.
- `elem.removeAttribute(name)` – удалить атрибут.
- `elem.attributes` – это коллекция всех атрибутов.

В большинстве ситуаций предпочтительнее использовать DOM-свойства. Нужно использовать атрибуты только тогда, когда DOM-свойства не подходят, когда нужны именно атрибуты, например:

- Нужен нестандартный атрибут. Но если он начинается с `data-`, тогда нужно использовать `dataset`.
- Мы хотим получить именно то значение, которое написано в HTML. Значение DOM-свойства может быть другим, например, свойство `href` – всегда полный URL, а нам может понадобиться получить «оригинальное» значение.

Задачи

Получите атрибут

важность: 5

Напишите код для выбора элемента с атрибутом `data-widget-name` из документа и прочитайте его значение.

```
<!DOCTYPE html>
<html>
<body>

  <div data-widget-name="menu">Choose the genre</div>

  <script>
    /* your code */
  </script>
</body>
</html>
```

решение

Сделайте внешние ссылки оранжевыми

важность: 3

Сделайте все внешние ссылки оранжевыми, изменяя их свойство `style`.

Ссылка является внешней, если:

- Её `href` содержит `://`
- Но не начинается с `http://internal.com`.

Пример:

```
<a name="list">the list</a>
<ul>
  <li><a href="http://google.com">http://google.com</a>
  <li><a href="/tutorial">/tutorial.html</a></li>
  <li><a href="local/path">local/path</a></li>
  <li><a href="ftp://ftp.com/my.zip">ftp://ftp.com/my.z
  <li><a href="http://nodejs.org">http://nodejs.org</a>
  <li><a href="http://internal.com/test">http://interna
</ul>

<script>
  // добавление стиля для одной ссылки
  let link = document.querySelector('a');
  link.style.color = 'orange';
</script>
```

Результат должен быть таким:

The list:

- http://google.com
- /tutorial.html
- local/path
- ftp://ftp.com/my.zip
- http://nodejs.org
- http://internal.com/test

Открыть песочницу для задачи.

решение

Комментарии

перед тем как писать...