20th October 2019

Attributes and properties

When the browser loads the page, it “reads” (another word: “parses”) the HTML and generates DOM objects from it. For element nodes, most standard HTML attributes automatically become properties of DOM objects.

For instance, if the tag is `<body id="page">`, then the DOM object has `body.id="page"`.

But the attribute-property mapping is not one-to-one! In this chapter we’ll pay attention to separate these two notions, to see how to work with them, when they are the same, and when they are different.

DOM properties

We’ve already seen built-in DOM properties. There are a lot. But technically no one limits us, and if there aren’t enough, we can add our own.

DOM nodes are regular JavaScript objects. We can alter them.

For instance, let’s create a new property in `document.body`:

```
1  document.body.myData = {
2    name: 'Caesar',
3    title: 'Imperator'
4  };
5
6  alert(document.body.myData.title); // Imperator
```

We can add a method as well:

```
1  document.body.sayTagName = function() {
2    alert(this.tagName);
3  };
4
5  document.body.sayTagName(); // BODY (the value of "this" in the method is doc
```

We can also modify built-in prototypes like `Element.prototype` and add new methods to all elements:

```
1  Element.prototype.sayHi = function() {
2    alert(`Hello, I'm ${this.tagName}`);
3  };
4
5  document.documentElement.sayHi(); // Hello, I'm HTML
6  document.body.sayHi(); // Hello, I'm BODY
```

So, DOM properties and methods behave just like those of regular JavaScript objects:

- They can have any value.
- They are case-sensitive (write `elem.nodeType`, not `elem.NoDeTyPe`).

HTML attributes

In HTML, tags may have attributes. When the browser parses the HTML to create DOM objects for tags, it recognizes *standard* attributes and creates DOM properties from them.

So when an element has `id` or another *standard* attribute, the corresponding property gets created. But that doesn't happen if the attribute is non-standard.

For instance:

```
<body id="test" something="non-standard">
  <script>
    alert(document.body.id); // test
    // non-standard attribute does not yield a property
    alert(document.body.something); // undefined
  </script>
</body>
```

Please note that a standard attribute for one element can be unknown for another one. For instance, `"type"` is standard for `<input>` (HTMLInputElement), but not for `<body>` (HTMLBodyElement). Standard attributes are described in the specification for the corresponding element class.

Here we can see it:

```
<body id="body" type="...">
  <input id="input" type="text">
  <script>
    alert(input.type); // text
    alert(body.type); // undefined: DOM property not created, because it's no
  </script>
</body>
```

So, if an attribute is non-standard, there won't be a DOM-property for it. Is there a way to access such attributes?

Sure. All attributes are accessible by using the following methods:

- `elem.hasAttribute(name)` – checks for existence.
- `elem.getAttribute(name)` – gets the value.
- `elem.setAttribute(name, value)` – sets the value.
- `elem.removeAttribute(name)` – removes the attribute.

These methods operate exactly with what's written in HTML.

Also one can read all attributes using `elem.attributes` : a collection of objects that belong to a built-in Attr class, with `name` and `value` properties.

Here's a demo of reading a non-standard property:

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

HTML attributes have the following features:

- Their name is case-insensitive (`id` is same as `ID`).
- Their values are always strings.

Here's an extended demo of working with attributes:

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

  <script>
    alert( elem.getAttribute('About') ); // (1) 'Elephant', reading

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

    alert( elem.outerHTML ); // (3), see if the attribute is in HTML (yes)

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

Please note:

1. `getAttribute('About')` – the first letter is uppercase here, and in HTML it's all lowercase. But that doesn't matter: attribute names are case-insensitive.
2. We can assign anything to an attribute, but it becomes a string. So here we have `"123"` as the value.
3. All attributes including ones that we set are visible in `outerHTML` .
4. The `attributes` collection is iterable and has all the attributes of the element (standard and non-standard) as objects with `name` and `value` properties.

Property-attribute synchronization

When a standard attribute changes, the corresponding property is auto-updated, and (with some exceptions) vice versa.

In the example below `id` is modified as an attribute, and we can see the property changed too. And then the same backwards:

```
<input>

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

  // attribute => property
  input.setAttribute('id', 'id');
  alert(input.id); // id (updated)

  // property => attribute
  input.id = 'newId';
  alert(input.getAttribute('id')); // newId (updated)
</script>
```

But there are exclusions, for instance `input.value` synchronizes only from attribute → to property, but not back:

```
<input>

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

  // attribute => property
  input.setAttribute('value', 'text');
  alert(input.value); // text

  // NOT property => attribute
  input.value = 'newValue';
  alert(input.getAttribute('value')); // text (not updated!)
</script>
```

In the example above:

- Changing the attribute `value` updates the property.
- But the property change does not affect the attribute.

That “feature” may actually come in handy, because the user actions may lead to `value` changes, and then after them, if we want to recover the “original” value from HTML, it’s in the attribute.

DOM properties are typed

DOM properties are not always strings. For instance, the `input.checked` property (for checkboxes) is a boolean:

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

<script>
  alert(input.getAttribute('checked')); // the attribute value is: empty stri
  alert(input.checked); // the property value is: true
</script>
```

There are other examples. The `style` attribute is a string, but the `style` property is an object:

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

<script>
  // string
  alert(div.getAttribute('style')); // color:red;font-size:120%

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

Most properties are strings though.

Quite rarely, even if a DOM property type is a string, it may differ from the attribute. For instance, the `href` DOM property is always a *full* URL, even if the attribute contains a relative URL or just a `#hash`.

Here's an example:

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

  // property
  alert(a.href ); // full URL in the form http://site.com/page#hello
</script>
```

If we need the value of `href` or any other attribute exactly as written in the HTML, we can use `getAttribute`.

Non-standard attributes, dataset

When writing HTML, we use a lot of standard attributes. But what about non-standard, custom ones? First, let's see whether they are useful or not? What for?

Sometimes non-standard attributes are used to pass custom data from HTML to JavaScript, or to "mark" HTML-elements for JavaScript.

Like this:

```
<!-- mark the div to show "name" here -->
<div show-info="name"></div>
<!-- and age here -->
<div show-info="age"></div>

<script>
  // the code finds an element with the mark and shows what's requested
  let user = {
    name: "Pete",
    age: 25
  };

```

```
  for(let div of document.querySelectorAll('[show-info]')) {
    // insert the corresponding info into the field
    let field = div.getAttribute('show-info');
    div.innerHTML = user[field]; // first Pete into "name", then 25 into "age
  }
</script>
```

Also they can be used to style an element.

For instance, here for the order state the attribute `order-state` is used:

```
<style>
  /* styles rely on the custom attribute "order-state" */
  .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>
```

Why would using an attribute be preferable to having classes like `.order-state-new`, `.order-state-pending`, `order-state-canceled`?

Because an attribute is more convenient to manage. The state can be changed as easy as:

```
// a bit simpler than removing old/adding a new class
div.setAttribute('order-state', 'canceled');
```

But there may be a possible problem with custom attributes. What if we use a non-standard attribute for our purposes and later the standard introduces it and makes it do something? The HTML language is alive, it grows, and more attributes appear to suit the needs of developers. There may be unexpected effects in such case.

To avoid conflicts, there exist data-* attributes.

All attributes starting with "data-" are reserved for programmers' use. They are available in the `dataset` property.

For instance, if an `elem` has an attribute named `"data-about"`, it's available as `elem.dataset.about`.

Like this:

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

Multiword attributes like `data-order-state` become camel-cased: `dataset.orderState`.

Here's a rewritten "order state" example:

```
<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>
  // read
  alert(order.dataset.orderState); // new

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

Using `data-*` attributes is a valid, safe way to pass custom data.

Please note that we can not only read, but also modify data-attributes. Then CSS updates the view accordingly: in the example above the last line `(*)` changes the color to blue.

Summary

- Attributes – is what's written in HTML.
- Properties – is what's in DOM objects.

A small comparison:

Properties	Attributes

	Properties	Attributes
Type	Any value, standard properties have types described in the spec	A string
Name	Name is case-sensitive	Name is not case-sensitive

Methods to work with attributes are:

- `elem.hasAttribute(name)` – to check for existence.
- `elem.getAttribute(name)` – to get the value.
- `elem.setAttribute(name, value)` – to set the value.
- `elem.removeAttribute(name)` – to remove the attribute.
- `elem.attributes` is a collection of all attributes.

For most situations using DOM properties is preferable. We should refer to attributes only when DOM properties do not suit us, when we need exactly attributes, for instance:

- We need a non-standard attribute. But if it starts with `data-`, then we should use `dataset`.
- We want to read the value "as written" in HTML. The value of the DOM property may be different, for instance the `href` property is always a full URL, and we may want to get the "original" value.

Tasks

Get the attribute

importance: 5

Write the code to select the element with `data-widget-name` attribute from the document and to read its value.

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

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

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

solution

Make external links orange

importance: 3

Make all external links orange by altering their `style` property.

A link is external if:

- Its `href` has `://` in it
- But doesn't start with `http://internal.com` .

Example:

```
<a name="list">the list</a>
<ul>
  <li><a href="http://google.com">http://google.com</a></li>
  <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.zip</a></li>
  <li><a href="http://nodejs.org">http://nodejs.org</a></li>
  <li><a href="http://internal.com/test">http://internal.com/test</a></li>
</ul>

<script>
  // setting style for a single link
  let link = document.querySelector('a');
  link.style.color = 'orange';
</script>
```

The result should be:

The list:

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

Open a sandbox for the task.

solution

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)