Understanding Relationships Between HTML Attributes & DOM Object's Properties

Summary: in this tutorial, you will learn the relationships between HTML attributes and DOM object's properties.

When the web browser loads an HTML page, it generates the corresponding DOM objects based on the DOM nodes of the document.

For example, if a page contains the following `input` element:

```
<input type="text" id="username">
```

The web browser will generate an `HTMLInputElement` object.

The `input` element has two attributes:

- The `type` attribute with the value `text` .
- The `id` attribute with the value `username` .

The generated `HTMLInputElement` object will have the corresponding properties:

- The `input.type` with the value `text` .
- The `input.id` with the value `username` .

In other words, the web browser will automatically convert attributes of HTML elements to properties of DOM objects.

However, the web browser only converts the *standard* attributes to the DOM object's properties. The standard attributes of an element are listed on the element's specification.

Attribute-property mapping is not always one-to-one. For example:

```
<input type="text" id="username" secured="true">
```

In this example, the `secured` is a non-standard attribute:

```
let input = document.querySelector('#username');
console.log(input.secured); // undefined
```

Attribute methods

To access both standard and non-standard attributes, you use the following methods:

- `element.getAttribute(name)` – get the attribute value
- `element.setAttribute(name, value)` – set the value for the attribute
- `element.hasAttribute(name)` – check for the existence of an attribute
- `element.removeAttribute(name)` – remove the attribute

element.attributes

The `element.attributes` property provides a live collection of attributes available on a specific element. For example:

```
let input = document.querySelector('#username');

for(let attr of input.attributes) {
    console.log(`${attr.name} = ${attr.value}` )
}
```

Output:

```
type = text
id = username
secure = true
```

> Note that `element.attributes` is a `NamedNodeMap`, not an `Array`, therefore, it has no Array's methods.

Attribute-property synchronization

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

Suppose that you have the following `input` element:

```
<input type="text" id="username" tabindex="1">
```

The following example illustrates the change of the `tabindex` attribute is propagated to the `tabIndex` property and vice versa:

```
let input = document.querySelector('#username');

// attribute -> property
input.setAttribute('tabindex', 2);
console.log(input.tabIndex);  // 2



// property -> attribute
input.tabIndex = 3;
console.log(input.getAttribute('tabIndex')); // 3
```

The following example shows when the `value` attribute changes, it reflects in the `value` property, but not the other way around:

```
let input = document.querySelector('#username');

// attribute -> property: OK
input.setAttribute('value','guest');
console.log(input.value);  // guest
```

```
// property -> attribute: doesn't change
input.value = 'admin';
console.log(input.getAttribute('value')); // guest
```

DOM properties are typed

The value of an attribute is always a string. However, when the attribute is converted to the property of a DOM object, the property value can be a string, a boolean, an object, etc.

The following `checkbox` element has the `checked` attribute. When the `checked` attribute is converted to the property, it is a boolean value:

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

let checkbox = document.querySelector('#chkAccept');
console.log(checkbox.checked); // true
```

The following shows an `input` element with the `style` attribute:

```
<input type="password" id="password" style="color:red;with:100%">
```

The `style` attribute is a string while the `style` property is an object:

```
let input = document.querySelector('#password');

let styleAttr = input.getAttribute('style');
console.log(styleAttr);

console.dir(input.style);
```

Output:

```
[object CSSStyleDeclaration]
```

The data-* attributes

If you want to add a custom attribute to an element, you should prefix it with the `data-` e.g., `data-secured` because all attributes start with `data-` are reserved for the developer's uses.

To access `data-*` attributes, you can use the `dataset` property. For example, we have the following `div` element with custom attributes:

```
<div id="main" data-progress="pending" data-value="10%"></div>
```

The following shows how to access the `data-*` attributes via the `dataset` property:

```
let bar = document.querySelector('#main');
console.log(bar.dataset);
```

Output:

```
[object DOMStringMap] {
    progress: "pending",
    value: "10%"
}
```

Summary

- Attributes are specified in HTML elements.
- Properties are specified DOM objects.
- Attributes are converted to properties respectively.
- Use the `element.attributes` property to access standard and custom attributes of an element.
- Use the `element.dataset` property to access the `data-*` attributes.