2nd August 2019

Template element

A built-in `<template>` element serves as a storage for HTML markup templates. The browser ignores it contents, only checks for syntax validity, but we can access and use it in JavaScript, to create other elements.

In theory, we could create any invisible element somewhere in HTML for HTML markup storage purposes. What's special about `<template>` ?

First, its content can be any valid HTML, even if it normally requires a proper enclosing tag.

For example, we can put there a table row `<tr>` :

```
<template>
  <tr>
    <td>Contents</td>
  </tr>
</template>
```

Usually, if we try to put `<tr>` inside, say, a `<div>` , the browser detects the invalid DOM structure and "fixes" it, adds `<table>` around. That's not what we want. On the other hand, `<template>` keeps exactly what we place there.

We can put styles and scripts into `<template>` as well:

```
<template>
  <style>
    p { font-weight: bold; }
  </style>
  <script>
    alert("Hello");
  </script>
</template>
```

The browser considers `<template>` content "out of the document": styles are not applied, scripts are not executed, `<video autoplay>` is not run, etc.

The content becomes live (styles apply, scripts run etc) when we insert it into the document.

Inserting template

The template content is available in its `content` property as a DocumentFragment – a special type of DOM node.

We can treat it as any other DOM node, except one special property: when we insert it somewhere, its children are inserted instead.

For example:

```html
<template id="tmpl">
  <script>
    alert("Hello");
  </script>
  <div class="message">Hello, world!</div>
</template>

<script>
  let elem = document.createElement('div');

  // Clone the template content to reuse it multiple times
  elem.append(tmpl.content.cloneNode(true));

  document.body.append(elem);
  // Now the script from <template> runs
</script>
```

Let's rewrite a Shadow DOM example from the previous chapter using `<template>` :

```html
<template id="tmpl">
  <style> p { font-weight: bold; } </style>
  <p id="message"></p>
</template>

<div id="elem">Click me</div>

<script>
  elem.onclick = function() {
    elem.attachShadow({mode: 'open'});

    elem.shadowRoot.append(tmpl.content.cloneNode(true)); // (*)

    elem.shadowRoot.getElementById('message').innerHTML = "Hello from the sha
  };
</script>
```

Click me

In the line `(*)` when we clone and insert `tmpl.content` , as its `DocumentFragment` , its children (`<style>` , `<p>`) are inserted instead.

They form the shadow DOM:

```html
<div id="elem">
  #shadow-root
    <style> p { font-weight: bold; } </style>

```

```
      <p id="message"></p>
</div>
```

Summary

To summarize:

- `<template>` content can be any syntactically correct HTML.
- `<template>` content is considered “out of the document”, so it doesn’t affect anything.
- We can access `template.content` from JavaScript, clone it to reuse in a new component.

The `<template>` tag is quite unique, because:

- The browser checks HTML syntax inside it (as opposed to using a template string inside a script).
- …But still allows use of any top-level HTML tags, even those that don’t make sense without proper wrappers (e.g. `<tr>`).
- The content becomes interactive: scripts run, `<video autoplay>` plays etc, when inserted into the document.

The `<template>` element does not feature any iteration mechanisms, data binding or variable substitutions, but we can implement those on top of it.

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…)