26th December 2019

Shadow DOM slots, composition

Many types of components, such as tabs, menus, image galleries, and so on, need the content to render.

Just like built-in browser `<select>` expects `<option>` items, our `<custom-tabs>` may expect the actual tab content to be passed. And a `<custom-menu>` may expect menu items.

The code that makes use of `<custom-menu>` can look like this:

```html
<custom-menu>
  <title>Candy menu</title>
  <item>Lollipop</item>
  <item>Fruit Toast</item>
  <item>Cup Cake</item>
</custom-menu>
```

…Then our component should render it properly, as a nice menu with given title and items, handle menu events, etc.

How to implement it?

We could try to analyze the element content and dynamically copy-rearrange DOM nodes. That's possible, but if we're moving elements to shadow DOM, then CSS styles from the document do not apply in there, so the visual styling may be lost. Also that requires some coding.

Luckily, we don't have to. Shadow DOM supports `<slot>` elements, that are automatically filled by the content from light DOM.

Named slots

Let's see how slots work on a simple example.

Here, `<user-card>` shadow DOM provides two slots, filled from light DOM:

```html
<script>
customElements.define('user-card', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.innerHTML = `
      <div>Name:
        <slot name="username"></slot>
      </div>
      <div>Birthday:
        <slot name="birthday"></slot>
      </div>
    `;
  }
});
</script>

<user-card>
  <span slot="username">John Smith</span>
  <span slot="birthday">01.01.2001</span>
</user-card>
```

```
Name: John Smith
Birthday: 01.01.2001
```

In the shadow DOM, `<slot name="X">` defines an “insertion point”, a place where elements with `slot="X"` are rendered.

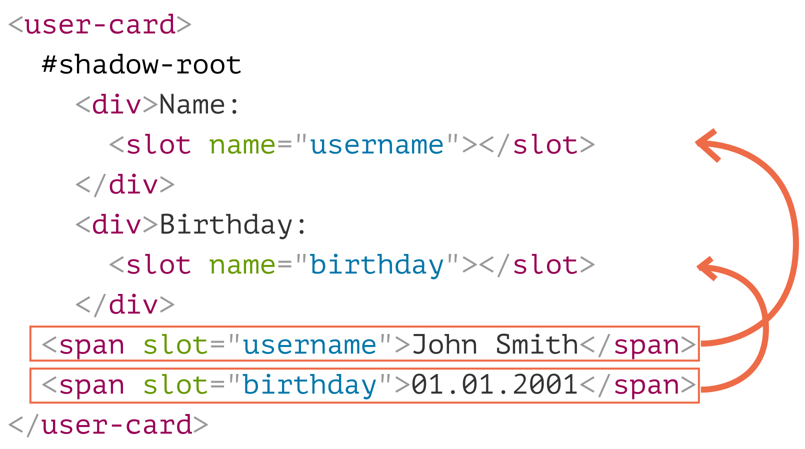
Then the browser performs “composition”: it takes elements from the light DOM and renders them in corresponding slots of the shadow DOM. At the end, we have exactly what we want – a component that can be filled with data.

Here’s the DOM structure after the script, not taking composition into account:

```
<user-card>
  #shadow-root
    <div>Name:
      <slot name="username"></slot>
    </div>
    <div>Birthday:
      <slot name="birthday"></slot>
    </div>
  <span slot="username">John Smith</span>
  <span slot="birthday">01.01.2001</span>
</user-card>
```

We created the shadow DOM, so here it is, under `#shadow-root`. Now the element has both light and shadow DOM.

For rendering purposes, for each `<slot name="...">` in shadow DOM, the browser looks for `slot="..."` with the same name in the light DOM. These elements are rendered inside the slots:



```
<user-card>
  #shadow-root
    <div>Name:
      <slot name="username"></slot>
    </div>
    <div>Birthday:
      <slot name="birthday"></slot>
    </div>
  <span slot="username">John Smith</span>
  <span slot="birthday">01.01.2001</span>
</user-card>
```

The result is called “flattened” DOM:

```
<user-card>
  #shadow-root
    <div>Name:
      <slot name="username">
        <!-- slotted element is inserted into the slot -->
        <span slot="username">John Smith</span>
      </slot>
    </div>
    <div>Birthday:
      <slot name="birthday">
        <span slot="birthday">01.01.2001</span>
      </slot>
    </div>
</user-card>
```

…But the flattened DOM exists only for rendering and event-handling purposes. It’s kind of “virtual”. That’s how things are shown. But the nodes in the document are actually not moved around!

That can be easily checked if we run `querySelectorAll` : nodes are still at their places.

```
// light DOM <span> nodes are still at the same place, under `<user-card>`
alert( document.querySelectorAll('user-card span').length ); // 2
```

So, the flattened DOM is derived from shadow DOM by inserting slots. The browser renders it and uses for style inheritance, event propagation (more about that later). But JavaScript still sees the document “as is”, before flattening.

⚠ Only top-level children may have slot="…" attribute

The `slot="..."` attribute is only valid for direct children of the shadow host (in our example, `<user-card>` element). For nested elements it’s ignored.

For example, the second `<span>` here is ignored (as it’s not a top-level child of `<user-card>`):

```
<user-card>
  <span slot="username">John Smith</span>
  <div>
    <!-- invalid slot, must be direct child of user-card -->
    <span slot="birthday">01.01.2001</span>
  </div>
</user-card>
```

If there are multiple elements in light DOM with the same slot name, they are appended into the slot, one after another.

For example, this:

```
<user-card>
  <span slot="username">John</span>
  <span slot="username">Smith</span>
</user-card>
```

Gives this flattened DOM with two elements in `<slot name="username">` :

```
<user-card>
  #shadow-root
    <div>Name:
      <slot name="username">
        <span slot="username">John</span>
        <span slot="username">Smith</span>
      </slot>
    </div>
    <div>Birthday:
      <slot name="birthday"></slot>
    </div>
</user-card>
```

Slot fallback content

If we put something inside a `<slot>` , it becomes the fallback, “default” content. The browser shows it if there’s no corresponding filler in light DOM.

For example, in this piece of shadow DOM, `Anonymous` renders if there’s no `slot="username"` in light DOM.

```
<div>Name:
  <slot name="username">Anonymous</slot>
</div>
```

Default slot: first unnamed

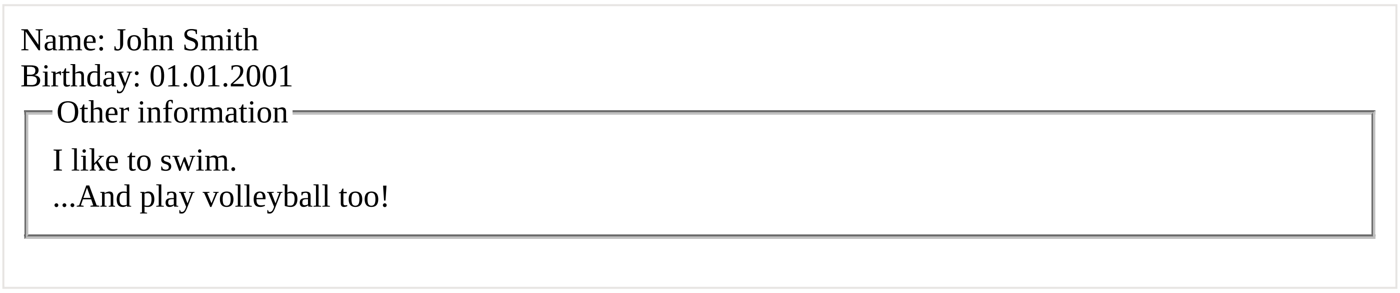
The first `<slot>` in shadow DOM that doesn’t have a name is a “default” slot. It gets all nodes from the light DOM that aren’t slotted elsewhere.

For example, let’s add the default slot to our `<user-card>` that shows all unslotted information about the user:

```
<script>
customElements.define('user-card', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.innerHTML = `
    <div>Name:
      <slot name="username"></slot>
    </div>
    <div>Birthday:
      <slot name="birthday"></slot>
    </div>
    <fieldset>
```

```
      <legend>Other information</legend>
      <slot></slot>
    </fieldset>
    `;
  }
});
</script>

<user-card>
  <div>I like to swim.</div>
  <span slot="username">John Smith</span>
  <span slot="birthday">01.01.2001</span>
  <div>...And play volleyball too!</div>
</user-card>
```


Name: John Smith
Birthday: 01.01.2001
Other information
I like to swim.
...And play volleyball too!

All the unslotted light DOM content gets into the “Other information” fieldset.

Elements are appended to a slot one after another, so both unslotted pieces of information are in the default slot together.

The flattened DOM looks like this:

```
<user-card>
  #shadow-root
    <div>Name:
      <slot name="username">
        <span slot="username">John Smith</span>
      </slot>
    </div>
    <div>Birthday:
      <slot name="birthday">
        <span slot="birthday">01.01.2001</span>
      </slot>
    </div>
    <fieldset>
      <legend>About me</legend>
      <slot>
        <div>Hello</div>
        <div>I am John!</div>
      </slot>
    </fieldset>
</user-card>
```

Menu example

Now let’s back to `<custom-menu>` , mentioned at the beginning of the chapter.

We can use slots to distribute elements.

Here's the markup for `<custom-menu>` :

```html
<custom-menu>
  <span slot="title">Candy menu</span>
  <li slot="item">Lollipop</li>
  <li slot="item">Fruit Toast</li>
  <li slot="item">Cup Cake</li>
</custom-menu>
```

The shadow DOM template with proper slots:

```html
<template id="tmpl">
  <style> /* menu styles */ </style>
  <div class="menu">
    <slot name="title"></slot>
    <ul><slot name="item"></slot></ul>
  </div>
</template>
```

1. `<span slot="title">` goes into `<slot name="title">` .
2. There are many `<li slot="item">` in the template, but only one `<slot name="item">` in the template. So all such `<li slot="item">` are appended to `<slot name="item">` one after another, thus forming the list.

The flattened DOM becomes:

```html
<custom-menu>
  #shadow-root
    <style> /* menu styles */ </style>
    <div class="menu">
      <slot name="title">
        <span slot="title">Candy menu</span>
      </slot>
      <ul>
        <slot name="item">
          <li slot="item">Lollipop</li>
          <li slot="item">Fruit Toast</li>
          <li slot="item">Cup Cake</li>
        </slot>
      </ul>
    </div>
</custom-menu>
```

One might notice that, in a valid DOM, `<li>` must be a direct child of `<ul>` . But that's flattened DOM, it describes how the component is rendered, such thing happens naturally here.

We just need to add a `click` handler to open/close the list, and the `<custom-menu>` is ready:

```js
customElements.define('custom-menu', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});

    // tmpl is the shadow DOM template (above)
    this.shadowRoot.append( tmpl.content.cloneNode(true) );

    // we can't select light DOM nodes, so let's handle clicks on the slot
    this.shadowRoot.querySelector('slot[name="title"]').onclick = () => {
      // open/close the menu
      this.shadowRoot.querySelector('.menu').classList.toggle('closed');
    };
  }
});
```

Here’s the full demo:

Candy menu
Lollipop
Fruit Toast
Cup Cake

Of course, we can add more functionality to it: events, methods and so on.

Updating slots

What if the outer code wants to add/remove menu items dynamically?

The browser monitors slots and updates the rendering if slotted elements are added/removed.

Also, as light DOM nodes are not copied, but just rendered in slots, the changes inside them immediately become visible.

So we don’t have to do anything to update rendering. But if the component code wants to know about slot changes, then `slotchange` event is available.

For example, here the menu item is inserted dynamically after 1 second, and the title changes after 2 seconds:

```
<custom-menu id="menu">
  <span slot="title">Candy menu</span>
</custom-menu>

<script>
customElements.define('custom-menu', class extends HTMLElement {
  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.innerHTML = `<div class="menu">
      <slot name="title"></slot>
      <ul><slot name="item"></slot></ul>
    </div>`;

    // shadowRoot can't have event handlers, so using the first child
    this.shadowRoot.firstElementChild.addEventListener('slotchange',
      e => alert("slotchange: " + e.target.name)
    );
  }
});

setTimeout(() => {
  menu.insertAdjacentHTML('beforeEnd', '<li slot="item">Lollipop</li>')
}, 1000);

setTimeout(() => {
  menu.querySelector('[slot="title"]').innerHTML = "New menu";
}, 2000);
</script>
```

The menu rendering updates each time without our intervention.

There are two `slotchange` events here:

1. At initialization:

   `slotchange: title` triggers immediately, as the `slot="title"` from the light DOM gets into the corresponding slot.

2. After 1 second:

   `slotchange: item` triggers, when a new `<li slot="item">` is added.

Please note: there's no `slotchange` event after 2 seconds, when the content of `slot="title"` is modified. That's because there's no slot change. We modify the content inside the slotted element, that's another thing.

If we'd like to track internal modifications of light DOM from JavaScript, that's also possible using a more generic mechanism: MutationObserver.

Slot API

Finally, let's mention the slot-related JavaScript methods.

As we've seen before, JavaScript looks at the "real" DOM, without flattening. But, if the shadow tree has `{mode: 'open'}`, then we can figure out which elements assigned to a slot and, vise-versa, the slot by the element inside it:

- `node.assignedSlot` – returns the `<slot>` element that the `node` is assigned to.
- `slot.assignedNodes({flatten: true/false})` – DOM nodes, assigned to the slot. The `flatten` option is `false` by default. If explicitly set to `true`, then it looks more deeply into the flattened DOM, returning nested slots in case of nested components and the fallback content if no node assigned.
- `slot.assignedElements({flatten: true/false})` – DOM elements, assigned to the slot (same as above, but only element nodes).

These methods are useful when we need not just show the slotted content, but also track it in JavaScript.

For example, if `<custom-menu>` component wants to know, what it shows, then it could track `slotchange` and get the items from `slot.assignedElements`:

```
<custom-menu id="menu">
  <span slot="title">Candy menu</span>
  <li slot="item">Lollipop</li>
  <li slot="item">Fruit Toast</li>
```

```
</custom-menu>

<script>
customElements.define('custom-menu', class extends HTMLElement {
  items = []

  connectedCallback() {
    this.attachShadow({mode: 'open'});
    this.shadowRoot.innerHTML = `<div class="menu">
      <slot name="title"></slot>
      <ul><slot name="item"></slot></ul>
    </div>`;

    // slottable is added/removed/replaced
    this.shadowRoot.firstElementChild.addEventListener('slotchange', e => {
      let slot = e.target;
      if (slot.name == 'item') {
        this.items = slot.assignedElements().map(elem => elem.textContent);
        alert("Items: " + this.items);
      }
    });
  }
});

// items update after 1 second
setTimeout(() => {
  menu.insertAdjacentHTML('beforeEnd', '<li slot="item">Cup Cake</li>')
}, 1000);
</script>
```

Summary

Usually, if an element has shadow DOM, then its light DOM is not displayed. Slots allow to show elements from light DOM in specified places of shadow DOM.

There are two kinds of slots:

- Named slots: `<slot name="X">...</slot>` – gets light children with `slot="X"` .
- Default slot: the first `<slot>` without a name (subsequent unnamed slots are ignored) – gets unslotted light children.
- If there are many elements for the same slot – they are appended one after another.
- The content of `<slot>` element is used as a fallback. It's shown if there are no light children for the slot.

The process of rendering slotted elements inside their slots is called “composition”. The result is called a “flattened DOM”.

Composition does not really move nodes, from JavaScript point of view the DOM is still same.

JavaScript can access slots using methods:

- `slot.assignedNodes/Elements()` – returns nodes/elements inside the `slot` .
- `node.assignedSlot` – the reverse meethod, returns slot by a node.

If we'd like to know what we're showing, we can track slot contents using:

- `slotchange` event – triggers the first time a slot is filled, and on any add/remove/replace operation of the slotted element, but not its children. The slot is `event.target` .
- MutationObserver to go deeper into slot content, watch changes inside it.

Now, as we know how to show elements from light DOM in shadow DOM, let's see how to style them properly. The basic rule is that shadow elements are styled inside, and light elements – outside, but there are notable exceptions.

We'll see the details in the next chapter.

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…)