Removing and Replacing Elements

The DOM API provides a few operations that allow you to remove existing elements or replace a node of the document tree with another one. In this lesson, you will learn how to use these operations.

Earlier you learned that Firefox does not support the `insertAdjacent...()` operations, which are not the part of the DOM API standard, although all other major browsers support it.

To wrap all links with square brackets, you can find a general solution instead of the one demonstrated in Listing 6-9, which uses the script in Listing 6-10.

Listing 6-10: Browser independent way of fixing issues with Listing 6-9

index.html

```
<!-- Other markup elements are omitted -->
<script>
  function decorateLinks() {
    for (var i = 0; i < document.links.length; i++) {
      var link = document.links[i];
      var span = document.createElement('span');
      span.appendChild(document.createTextNode('['));
      span.appendChild(link.cloneNode(true));
      span.appendChild(document.createTextNode(']'));
      link.parentNode.replaceChild(span, link);
    }
  }
</script>
```

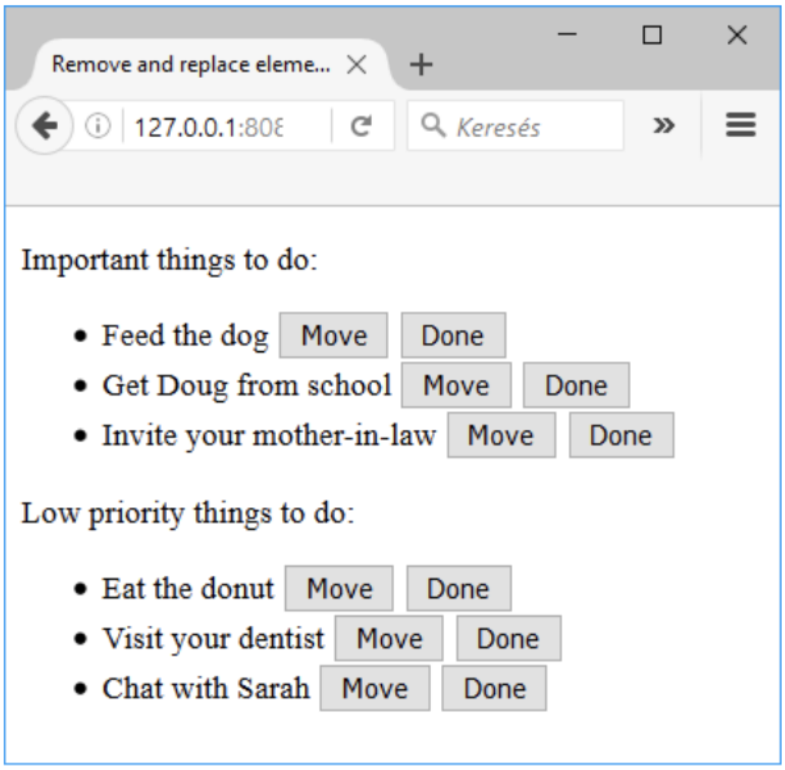
This code simply replaces the links with a `<span>` that contains three children, the text for `“[”`, the original link element, and the text for `“]”`. The code snippet uses the `replaceChild()` method of the link's parent node and passes

two arguments, the new child node (`span`) and the old child node to replace (`link`).

However, to make it work, another important thing needs to be done. The original link node cannot be at two places simultaneously, in its original location and within the newly created `<span>` element. You cannot append it to the `<span>`, but you can add the clone of the original node. This is exactly what the `cloneNode()` method does in the second invocation of `appendChild()`. Its argument with the true value indicates that the node should be cloned together with all of its children.

Let's see a more complex sample that not only replaces, but removes document tree nodes, too.

Figure 6-14 displays a simple web page that works as a simple to-do-list application. You have tasks that can be moved between two lists, important and low priority tasks. Each task can be marked as completed with clicking the `Done` button.



Important things to do:

- Feed the dog Move Done
- Get Doug from school Move Done
- Invite your mother-in-law Move Done

Low priority things to do:

- Eat the donut Move Done
- Visit your dentist Move Done
- Chat with Sarah Move Done

A simple to-do-list page

Listing 6-11 shows the complete HTML markup of the page. As you expect, it contains two unordered list, but the original markup does not declare buttons.

The Move and Done buttons are created at page load time, with the `decorateItems()` method (observe, the `onload` attribute of `<body>` is set to invoke this method).

Listing 6-11: Remove and replace elements

```
<!DOCTYPE html>
<html>
<head>
  <title>Remove and replace elements</title>
</head>
<body onload="decorateItems()">
  <p>Important things to do:</p>
  <ul id="high">
    <li>Feed the dog</li>
    <li>Get Doug from school</li>
    <li>Invite your mother-in-law</li>
  </ul>
  <p>Low priority things to do:</p>
  <ul id="low">
    <li>Eat the donut</li>
    <li>Visit your dentist</li>
    <li>Chat with Sarah</li>
  </ul>
  <script>
    function moveItem(node) {
      var divNode = node.parentNode;
      var liNode = divNode.parentNode;
      var ulNode = liNode.parentNode;
      var fromUl = ulNode.id;
      var toUl = fromUl == 'high'
        ? document.getElementById('low')
        : document.getElementById('high');
      var clone = liNode.cloneNode(true);
      toUl.appendChild(clone);
      ulNode.removeChild(liNode)
    }

    function closeItem(node) {
      var divNode = node.parentNode;
      var liNode = divNode.parentNode;
      var ulNode = liNode.parentNode;
      ulNode.removeChild(liNode)
    }

    function decorateItems() {
      var liNodes = document.getElementsByTagName('li');
      for (i = 0; i < liNodes.length; i++) {
        var liNode = liNodes[i];
        liNode.innerHTML = '<div>' + liNode.textContent
          + ' <button onclick="moveItem(this)">'
          + 'Move</button> '
          + '<button onclick="closeItem(this)">'
          + 'Done</button></div>';
      }
    }
  </script>

</body>
</html>
```

There is nothing new in `decorateItems()`, you can understand it according to what you have already learned.

After the page has been loaded and `decorateItems()` is completed, a list item’s markup looks like this:

```
<li>
  <div>
  Feed the dog
  <button onclick="moveItem(this)">
    Move
  </button>
  <button onclick="closeItem(this)">
    Done
  </button>
  <div>
</li>
```

When the `Move` or `Done` buttons are clicked, `moveItem()` or `closeItem()` are invoked. Both methods receive the object representing the appropriate `<button>` tag, and both initializes variables for nodes in this hierarchy:

```
var divNode = node.parentNode;
var liNode = divNode.parentNode;
var ulNode = liNode.parentNode;
```

To remove the `<li>` node, `moveItem()` and `closeItem()` both use the `removeChild()` method:

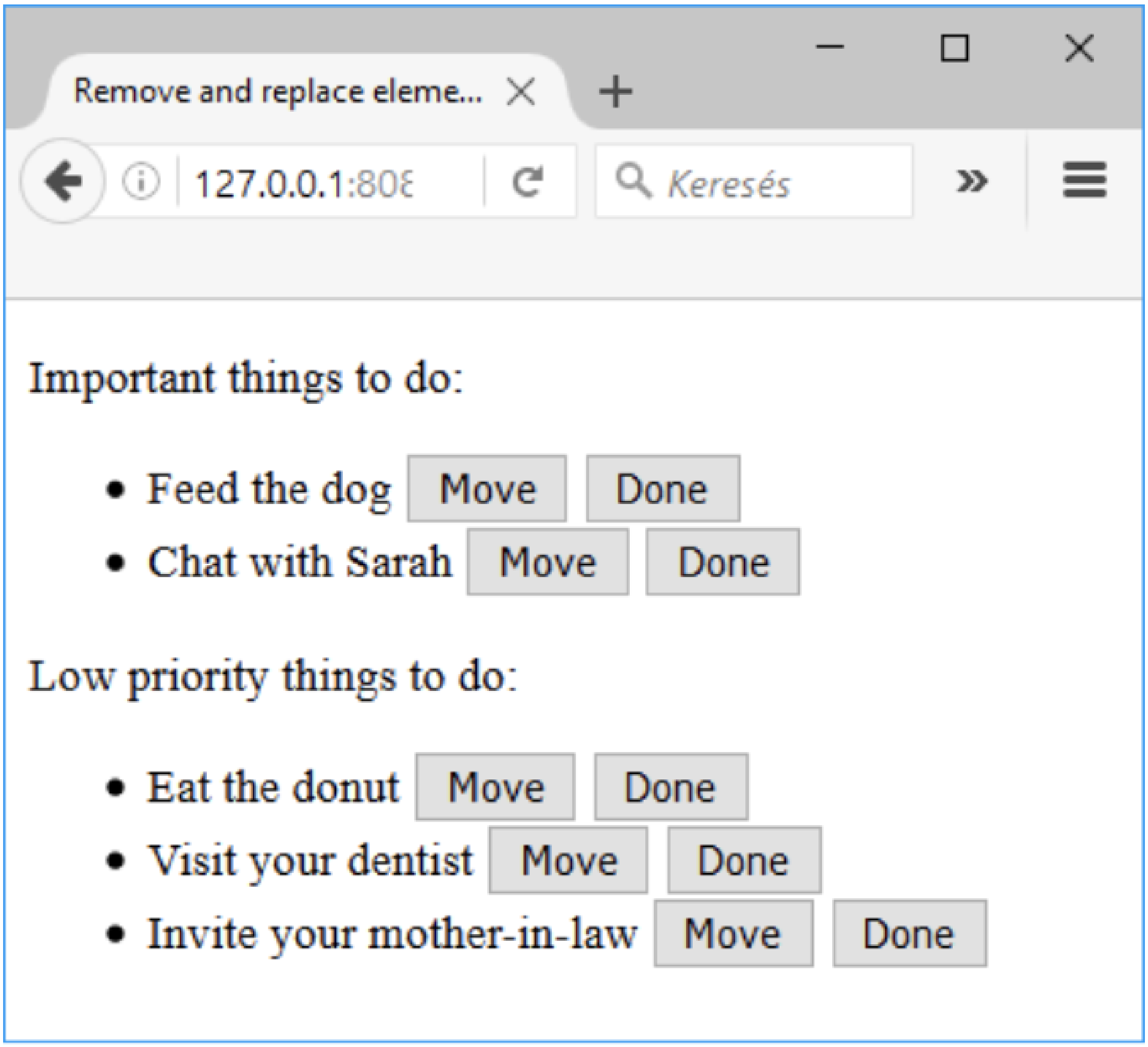
```
ulNode.removeChild(liNode)
```

Before removing a task, `moveItem()` adds a clone of that task to the other list:

```
var toUl = fromUl == 'high'
  ? document.getElementById('low')
  : document.getElementById('high');
var clone = liNode.cloneNode(true);
toUl.appendChild(clone);
```

As this code snippet shows, it first obtains the object representing the target list (`toUl`), then uses the `cloneNode()` method to create a full deep copy of the source `<li>` node. Finally, leveraging `appendChild()`, the code appends the cloned node to the target list.

The image below shows this web page in action. As you see, a few tasks have been moved between the priority lists and one task has already been completed.



Important things to do:

- Feed the dog Move Done
- Chat with Sarah Move Done

Low priority things to do:

- Eat the donut Move Done
- Visit your dentist Move Done
- Invite your mother-in-law Move Done

The to-do-list page in action

In the next lesson we change the style of these attributes and elements.