jso-dom Documentation

Release latest

Jun 18, 2018

Contents:

This library makes the creation of (part of) a Document Object Model (DOM) easier by providing a function that converts a nested JavaScript Object (JSO) to a DOM. This eliminates the need for manual element and text creation, hierarchical placement of elements, attribute setting and configuration of callback functions, resulting in more readable and compact code.

This library was originally developed for Greasemonkey user scripts, but it should be usable in any JavaScript program that runs in a browser.

Please see ReadTheDocs for the latest documentation.

CHAPTER 1

Introduction

Suppose we want to add a section containing a title with an id, some verbatim text and a list to a website, the following code would be required:

```
var root, temp, parent, child;

root = document.createElement("div");
parent = document.createElement("h3");
parent.setAttribute("id", "title_1");
child = document.createTextNode("A title");
parent.appendChild(child);
root.appendChild(parent);

parent = document.createElement("pre");
child = document.createTextNode("Verbatim text.");
parent.appendChild(child);
root.appendChild(parent);

temp = document.createElement("ul");
parent = document.createElement("li");
child = document.createTextNode("one");
parent.appendChild(child);
temp.appendChild(parent);

parent = document.createElement("li");
child = document.createTextNode("two");
parent.appendChild(child);
temp.appendChild(parent);

parent = document.createElement("li");
child = document.createTextNode("three");
parent.appendChild(child);
temp.appendChild(parent);
root.appendChild(temp);
```

Resulting in the following HTML tree:

```
<div>
  <h3 id="title_1">A title</h3>
  <pre>Verbatim text.</pre>
  <ul>
    <li>one</li>
    <li>two</li>
    <li>three</li>
  </ul>
</div>
```

Some drawbacks of this manual creation are immediately apparent:

- Quite some code is required to create a modestly sized HTML tree.
- We need to keep track of quite a number of variables, this number grows as the depth of the tree increases.
- The nested structure of HTML tree is not apparent which makes code maintenance difficult.

To address these problems, this library provides the function `JSOToDOM()` which takes a nested JSO as input and outputs a DOM. The following call to this function creates the HTML tree from our example:

```
var root = JSOToDOM({
  "div": {
    "h3": {
        "attrs": {"id": "title_1"},
      "text": "A title"},
    "pre": {
      "text": "Verbatim text."},
    "ul": [
      {"li": {"text": "one"}},
      {"li": {"text": "two"}},
      {"li": {"text": "three"}}]}});
```

This code is a lot more compact than the original and arguably more readable and therefore more maintainable.

CHAPTER 2

Installation

In a Greasemonkey user script, add the following line to the `UserScript` header:

```
// @require     https://github.com/jfjlaros/jso-dom/raw/master/src/jso_dom.js
```

This makes the function `JSOToDOM()` available.

CHAPTER 3

Usage

The function `JSOToDOM()` converts a nested JSO to a DOM recursively. Since the output is a tree, the top level of the JSO can only contain one element.

Apart from the exceptions described below, for every key in the JSO an element is created, the type of which is determined by name of the key. For example, the following call will create a single `div` element:

```
var root = JSOToDOM({"div": {}});
```

And a `div` within a `div` is created as follows:

```
var root = JSOToDOM({"div": {"div": {}}});
```

3.1 Lists

Since a JSO can not have multiple keys with the same name, we use lists to create multiple objects of the same type.

```
var root = JSOToDOM({
  "div": [
    {"div": {}},
    {"div": {}}]});
```

Of course mixing of element types is not forbidden:

```
var root = JSOToDOM({
  "div": [
    {"div": {}},
    {"pre": {}}]});
```

But can be written a bit easier because all key names are unique:

```
var root = JSOToDOM({
  "div": {
    "div": {},
    "pre": {}}});
```

3.2 The attrs key

The attrs key indicates that its value is a set of key-value pairs that is to be applied to the parent element.

In the following example, we create a section with id section_1 and we set the background colour of this section to blue.

```
var root = JSOToDOM({
  "div": {
      "attrs": {"id": "section_1", "style": "background-color: blue;"}}});
```

Note that we double indent the attrs key to emphasise that this key is not a subelement, but rather a set of parameters. This improved readability is shown in the following example:

```
var root = JSOToDOM({
  "div": {
      "attrs": {"id": "section_1"},
    "div": {}}});
```

3.3 The text key

The text key indicates that instead of a normal element, a text node should be created. The content of the text node is given by the value of the key.

```
var root = JSOToDOM({"text": "Just some text."});
```

This can be combined with the attrs key:

```
var root = JSOToDOM({
  "div": {
      "attrs": {"style": "background-color: blue;"},
    "text": "Am I blue yet?"}});
```

3.4 The event key

Callbacks for interactive elements can be implemented in two ways: via the onclick or related attributes, or by adding an event listener.

Using attributes, simple pieces of JavaScript code can be executed:

```
var root = JSOToDOM({
  "button": {
      "attrs": {"onclick": "alert(\"Hi there.\");"},
    "text": "Click me"}});
```

To make a proper callback function in Greasemonkey user scripts, an event listener s needed.

```
function callback() {
  alert("Hi again.");
}

var root = JSOToDOM({
  "button": {
      "event": {"type": "click", "listener": callback},
    "text": "Click me too"}});
```

CHAPTER 4

Contributors

- Jeroen F.J. Laros <jlaros@fixedpoint.nl> (Original author, maintainer)

Find out who contributed:

```
git shortlog -s -e
```