DOMWorker

A proposal for cross domain components

Elliott Sprehn (esprehn@chromium.org), 11-11-2013

Abstract

A DOMWorker is similar to a Worker except that it manages a collection of ShadowRoot instances associated with Elements of the owning page and has a DOMWindow object associated with it exposing the full DOM. This also allows the code inside the DOMWorker to register it's own custom elements and load imports.

ShadowRoots that are created by the DOMWorker by means of the rootcreated event are owned by the Document object in the DOMWorker's global scope, and no means is provided to get access to the parent page.

This tight security boundary allows authors to embed cross domain untrusted content in the same way as iframes into their pages. It also allows the authors of that content to write code without worrying about the embedding page modifying the global objects it depends on.

Currently pages create many <iframe>'s for this use case, for example the +1 button creates one iframe per button and then a single "worker" <iframe> that contains the JS that powers all the other iframes. This reduces the memory usage and load time by only loading the JS for the buttons once in a single place while still retaining the boundary that protects the JS for the buttons from the page that's embedding them.

The DOMWorker concept codifies this pattern into a platform primitive that allows custom elements to expose a public API to the embedding page while keeping the bulk of the widget's code inside the worker.

Note: No asynchronous guarantees are made for DOMWorker, specifically a busy loop inside the worker may hang the main page completely unlike a regular worker. Developers should assume that a DOMWorker is time sharing the same event loops as the owning document.

TBD: Some events should probably be forwarded (ex. hashchange) from the DOMWindow or Document to the DOMWorker's global object or document. It might be desirable that this is not the case though since it lets the embedding page expose only information it deems strictly necessary to expose.

Interfaces

```
[Constructor(DOMString url)]
interface DOMWorker : EventTarget { };
```

```
interface DOMWorkerGlobalScope : DOMWindow {
  attribute EventHandler onrootcreated;
};

interface Element {
  MessagePort createShadowRootWorker(DOMWorker worker);
};
```

Algorithms

When invoking DOMWorker(*url*):

1. let *worker* be a new Worker with the specified url and a new **DOMWorkerGlobalScope** as the global object.
2. return *worker.*

When invoking createShadowRootWorker(worker):

1. let *event* be a new MessageEvent of type “rootcreated”.
2. let *channel* be a new MessageChannel.
3. Set the *data* property of event to a new ShadowRoot created using the global scope of the worker.
4. Set the *source* property of *event* to *port1* of *channel.*
5. Post a task to the passed worker to fire a event at the worker global scope.
6. return *port2* of *channel*.

Example

```
// main document
var PlusOneButton = (function() {
  var worker = new DOMWorker(“//apis.google.com/plusone.js”);
  var proto = {
    createdCallback: function() {
      this.port_ = this.createShadowRootWorker(worker);
    },
    flash: function() {
      this.port_.postMessage(“flash”);
    },
  };
  return document.register(“g-plusone”, {prototype: proto});
}});

var button = document.createElement(“g-plusone”);
element.appendChild(button);
```

```
button.flash();
```

// plusone.js

```
onrootcreated = function(event) {
  var shadowRoot = event.data;
  shadowRoot.appendChild(document.createElement('div')).textContent = "+1";
  event.source.onmessage = function(event) {
    if (event.data == 'flash')
      playAnimationWith(shadowRoot);
  };
};
```