Create a Basic Loader with JavaScript Promises

By David Walsh on February 10, 2016   8

I've used JavaScript loaders for years; whether it was the Dojo loader, curl.js, or even using jQuery as a JavaScript loader, it's incredibly useful to request a set of resources and react once they've completed loading. Each JavaScript loader is feature-packed, efficient, and does a wonderful job of shimming the Promise API which didn't exist in the browser when the loader is created. The following is not that type of loader.

This *super simple* loader allows for loading of image, CSS, and JavaScript files, using the Promise API, and fires a callback upon success or failure. This tiny "loader" (I shouldn't even call it that) does **not**:

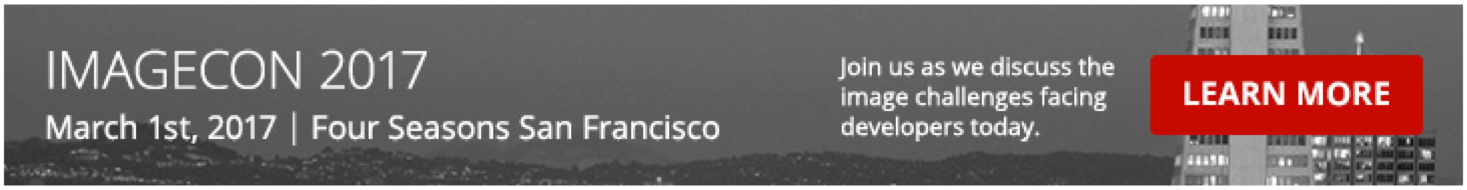
- cache results (though that would be easy)
- provide a module/object back
- do AJAX calls (though a XHR-to-Promise shim is available, or you can use fetch)
- ... or anything else advanced

Here is the tiny "loader" in all of its glory:

```
var load = (function() {
  // Function which returns a function: https://davidwalsh.name/javascript-functions
  function _load(tag) {
    return function(url) {
      // This promise will be used by Promise.all to determine success or failure
      return new Promise(function(resolve, reject) {
        var element = document.createElement(tag);
        var parent = 'body';
        var attr = 'src';

        // Important success and error for the promise
        element.onload = function() {
          resolve(url);
        };
        element.onerror = function() {
          reject(url);
        };

        // Need to set different attributes depending on tag type
        switch(tag) {
          case 'script':
            element.async = true;
            break;
          case 'link':
            element.type = 'text/css';
            element.rel = 'stylesheet';
            attr = 'href';
            parent = 'head';
        }

        // Inject into document to kick off loading
        element[attr] = url;
        document[parent].appendChild(element);
      });
    };
  }

  return {
    css: _load('link'),
    js: _load('script'),
    img: _load('img')
  }
})();

// Usage:  Load different file types with one callback
Promise.all([
    load.js('lib/highlighter.js'),
    load.js('lib/main.js'),
    load.css('lib/highlighter.css'),
    load.img('images/logo.png')
  ]).then(function() {
    console.log('Everything has loaded!');
  }).catch(function() {
    console.log('Oh no, epic failure!');
  });
```

A `load` object is created with `js`, `css`, and `img` functions which accept a URL to load. Each function returns a Promise and the `onload` or `onerror` event of the resource's tag triggers `resolve` or `reject` for the promise. `Promise.all` collects the resources to be loaded and `then` triggers upon successful load of all resources, `catch` if any of them fail.

I have to stress that this is meant to be a very, very simple "loader"; please save the comments about how it doesn't have bells and whistles that other loaders have. I love how awesome the Promise API makes async and resource loading management, as does the ServiceWorker API and fetch API. Do yourself a favor and check out these awesome APIs!


IMAGECON 2017
March 1st, 2017 | Four Seasons San Francisco
Join us as we discuss the image challenges facing developers today.
LEARN MORE

Recent Features

Creating Scrolling Parallax Effects with CSS

Introduction For quite a long time now websites with the so called "parallax" effect have been really popular. In case you have not heard of this effect, it basically includes different layers of images that are moving in different directions or with different speed. This leads to a...

Animated 3D Flipping Menu with CSS

CSS animations aren't just for basic fades or sliding elements anymore -- CSS animations are capable of much more. I've showed you how you can create an exploding logo (applied with JavaScript, but all animation is CSS), an animated Photo Stack, a sweet...

Incredible Demos

Dynamically Create Charts Using jQuery Flot and Google Analytics

Earlier in the week I published a popular article titled Dynamically Create Charts Using MooTools MilkChart and Google Analytics. My post showed you how to use MooTools MilkChart and a splash of PHP to create beautiful charts of Google Analytics data. I was...

Parallax Sound Waves Animating on Scroll

Scrolling animations are fun. They are fun to create and fun to use. If you are tired of bootstrapping you might find playing with scrolling animations as a nice juicy refreshment in your dry front-end development career. Let's have a look how to create animating...

Discussion

Rick Carlino

Great article! Would love to see this on a CDN somewhere. This wouldn't be a bad way to go for smaller projects that don't need all the bells and whistles of a full blown build system.

Kado

Isn't making `async` as default for you js scripts a bit risky. In your example, you might end up parsing "main.js" earlier that "highlighter.js" which might lead to ReferenceErrors thrown around.

David Walsh

You're right; maybe I can add a second argument to `load.js` where someone can say they want something sync.

samarjit

Have a declarative set of script dependencies. Look at require.js config shim. Having only this bit makes it scalable.

```
requirejs.config({
    shim: {
        'backbone': {
            //These script dependencies should be loaded before loading
            //backbone.js
            deps: ['underscore', 'jquery'],
           }
    }
});
```

Max

I have investigated this topic more extensively when I wrote my own loader (http://w3core.github.io/import.js/) and I see one serious gap here.

Unfortunately, we can not trust for the “load” event of the “link” tag, because most of mobile browsers does not dispatch the “load” event for this tag. A large number of mentions for this issue you can find on the stackoverflow site. There is no universal solution that can inform us that stylesheet is realy loaded, BUT we can be informed when HTTP request is completed and this is better then waiting of the event that never will be dispatched.

Max

There is a example that describes technique of event handling:

```
var callback = function(){
   alert("stylesheet request complete");
 };
 var url = "//path/to/stylesheet.file";
 var link = document.createElement("img");
 link.addEventListener("error", callback, !1);
 link.src = url;
```

Practically same loader can be implemented without any jQuery/Promise/etc toolkits (there is an example: http://w3core.github.io/import.js/). It can be more powerful, flexible and simple. Trust me.

Christof

This is short and sweet, but wouldn’t you also want to know if it failed (for more complex implementations), something which only Promises can give you.

Dirk

Although not that small it might still be an alternative https://github.com/dlueth/qoopido.demand/tree/feature/genie?files=1 which I wrote some time ago.

Name	Email	Website

Wrap your code in `<pre class="{language}"></pre>` *tags, link to a GitHub gist, JSFiddle fiddle, or CodePen pen to embed!*

☐ **Continue this conversation via email**

Post Comment! Use Code Editor