Josh Pullen
Posted on Jan 3, 2019

37 11

How to Load HTML, CSS, and JS Code into an iFrame

#javascript #webdev

If you're just here for the answer, not the story, the solution is at the bottom.

If you've ever used JSFiddle, Codepen, or others, this problem will be familiar to you: The goal is to take some HTML, CSS, and JS (stored as strings) and create an iframe with the code loaded inside.

This problem should be easy, but it isn't. At least... It wasn't, until I found the golden ticket I had been waiting for all along.

But more on that later. Let's start with all the things that didn't work, because that's more fun.

Attempt #1: Using srcdoc

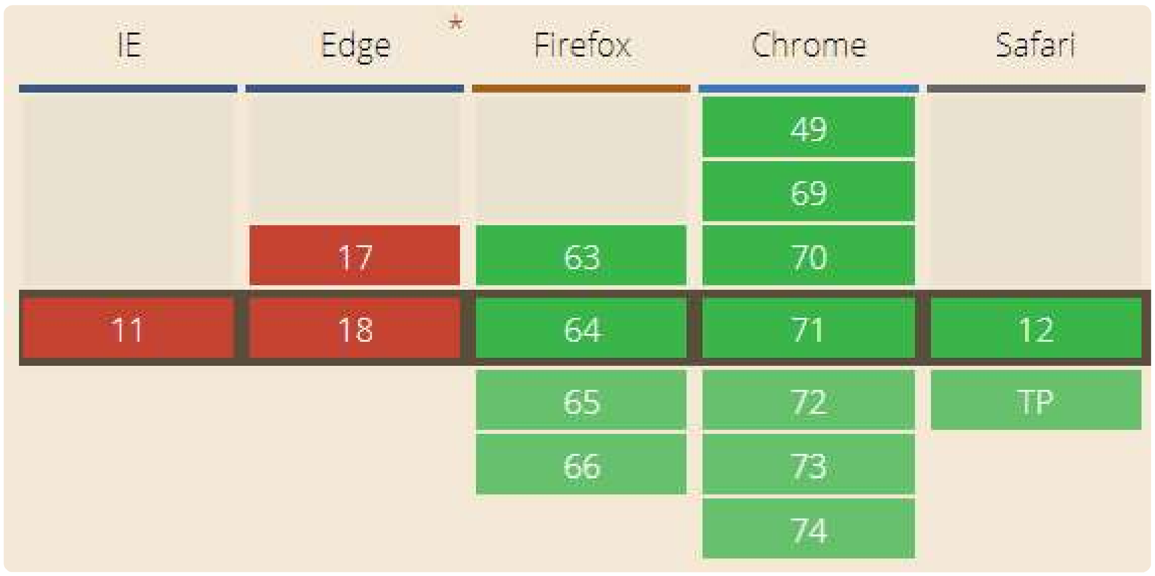
After doing a bit of research, I was thrilled to discover that it's possible to add a `srcdoc` attribute to iframes.

If you pass in an HTML string, the iframe will load with that HTML content inside:

```
<iframe srcdoc="<p>This text will appear in the iframe!</p>"></iframe>
```

Unfortunately, there are two main problems with this approach:

1. Browser Support for srcdoc is not great



IE	Edge *	Firefox	Chrome	Safari
			49	
			69	
	🟥 17	🟩 63	🟩 70	
🟥 11	🟥 18	🟩 64	🟩 71	🟩 12
		🟩 65	🟩 72	🟩 TP
		🟩 66	🟩 73	
			🟩 74	

If we want to support IE or Edge, we'll need a different approach (or a polyfill).

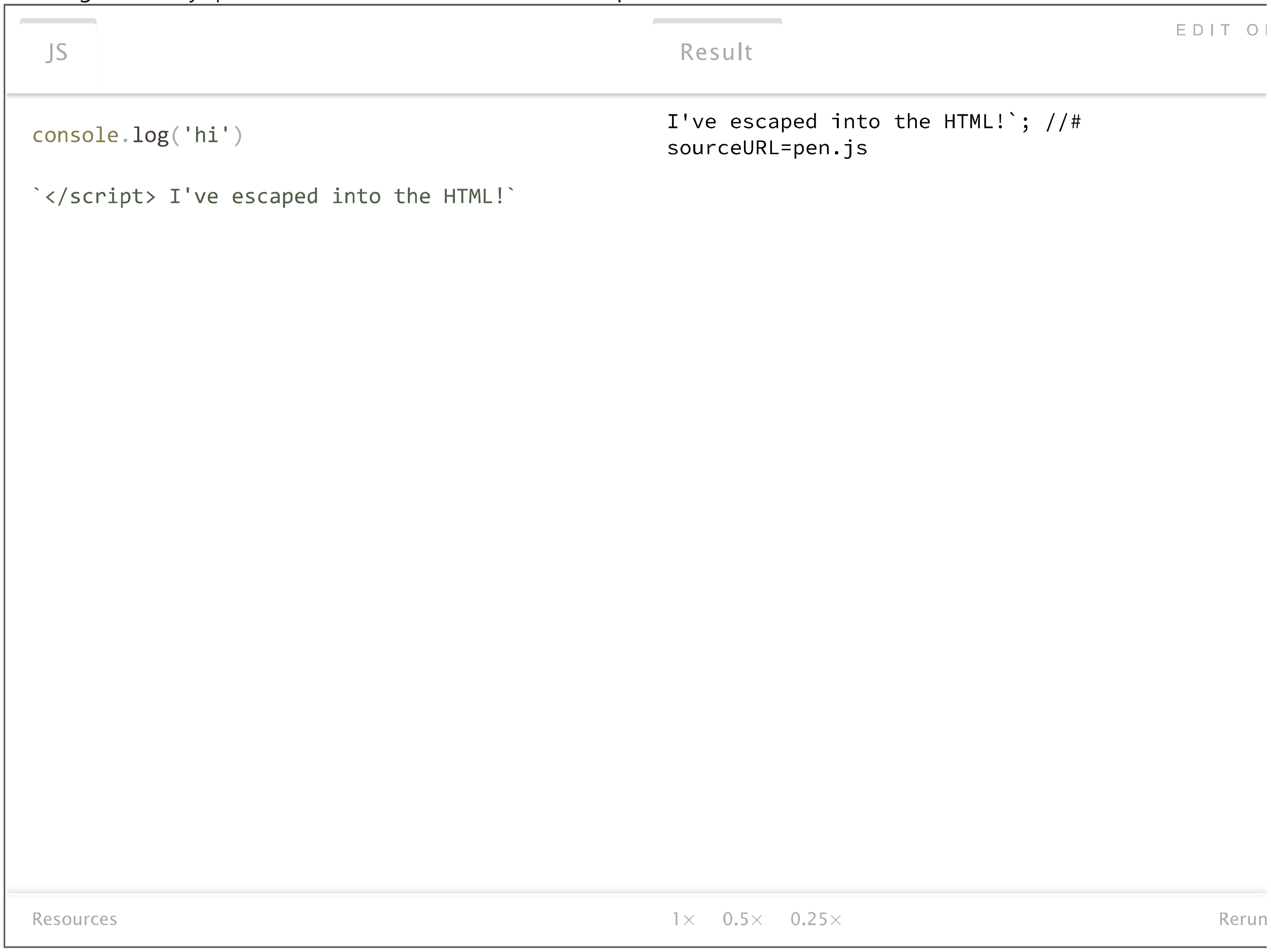
2. It's possible to "escape" from CSS/JS

Here's roughly how my implementation using srcdoc looked:

```
function setIframeContent(iframe, { html, css, js }) {
  const source = `
    <html>
      <head><style>${css}</style></head>
      <body>
        ${html}
        <script>${js}</script>
      </body>
    </html>
  `
  iframe.srcdoc = source
}
```

The problem? When writing CSS or JS, it's possible to "escape" out into HTML land, simply by including `</style>` or `</script>` in the code, respectively.

This bug is actually quite common; both JSFiddle and Codepen are affected:



```
console.log('hi')

`</script> I've escaped into the HTML!`
```

```
I've escaped into the HTML!`; //#
sourceURL=pen.js
```

Attempt #2: Serverless Boomerang

To fix the browser support issue, let's replace `srcdoc` with a regular `src` attribute. In order to do this, we'll need to pass a real URL instead of just code.

Perhaps we can set up a page which takes HTML, CSS, and JS "GET" parameters and spits out the same type of page as before, but this time loaded from an actual URL.

This is a perfect time to use a serverless architecture, because we just want a single endpoint that does one thing. Here's my attempt:

```
module.exports = (req, res) => {
  // Code comes from GET params in URL
  const { html = '', css = '', js = '' } = req.query

  // Generate and send HTML page
  return res.send(`
    <html>
      <head><style>${css}</style></head>
      <body>
        ${html}
        <script>${js}</script>
      </body>
    </html>
  `)
}
```

This works across virtually all browsers, but is not without its own issues:

1. "Escaping" from CSS/JS into HTML is still a problem
2. The entire source code is passed in a URL, which isn't ideal.

Attempt #3: Serverless Boomerang (redux)

Our first boomerang solved the browser support problem, but still has the "escaping" issue to deal with.

Fortunately, due to the way that we pass in the code, this can actually be solved. Rather than inserting the CSS and JS into the page on the server, we can do it on the client! This works because the URL GET parameters are still accessible to the client's computer.

The source here is a bit longer, but it does work:

```
module.exports = (req, res) => {
  return res.send(`
    <html>
      <head>
        <script type="text/javascript">
          window.addEventListener('load', function() {
            function getUrlParameter(name) {
              name = name.replace(/[\\[]/, '\\\\[').replace(/[\\]]/, '\\\\]');
              var regex = new RegExp('[\\\\?&]' + name + '=([^&#]*)');
              var results = regex.exec(location.search);
              return results === null ? '' : decodeURIComponent(results[1].replace(/\\+/g, ' '));
            };

            // Load JS from GET params (on client)
            var js = getUrlParameter('js');
            if (js) {
              var script = document.createElement('script');
              script.type = 'text/javascript';
              script.text = js;
              document.body.appendChild(script);
            }

            // Load CSS from GET params (on client)
            var css = getUrlParameter('css');
            if (css) {
              var style = document.createElement('style');
              style.type = 'text/css';
              if (style.styleSheet) {
                style.styleSheet.cssText = css;
              } else {
                style.appendChild(document.createTextNode(css));
              }
              document.head.appendChild(style);
            }

            // Remove the currently running script tag
            document.currentScript.parentNode.removeChild(document.currentScript);
          });
        </script>
      </head>
      <body>
        ${req.query.html || ''}
      </body>
    </html>
  `)
}
```

Now, if a script or style includes scary HTML characters, the browser will handle them for us when inserting said script/style into the document.

This solution is... fine. It works, technically. But we still have the soft URL length limit to consider. Plus, we're now dealing with something server-side that feels like it should happen on the client.

There must be a better way.

Solution: Blob URLs

This entire time, we've been trying to simulate loading data from a URL:

- First we used srcdoc to load data instead of loading from a URL
- Then we used the boomerang to load code from a URL
- Next we updated our boomerang to attempt to simulate the "loading CSS/JS from an external URL" behavior, despite every resource coming from one URL.

It turns out that Javascript has a feature to do just this: **Blob URLs**.

Blobs

We can use the `Blob` constructor to create a pseudo-file. It's not a real file loaded from disk or from a URL -- it's just stored in memory. But in many ways, it functions just like a real loaded file.

Then, we can use `URL.createObjectURL(blob)` to create a URL that can be used to load the contents of the blob.

Here's how that works in practice:

```
const getBlobURL = (code, type) => {
  const blob = new Blob([code], { type })
  return URL.createObjectURL(blob)
}

console.log(getBlobURL('<p>My webpage</p>', 'text/html'))
// blob:https://dev.to/9ca05e31-05ea-48f8-838d-cc1ad0949ec8
```

Try running the above code in the console to see it for yourself! It will log a URL. If you paste the URL into a new tab (including the `blob:` bit at the beginning), it will load a page containing the HTML.

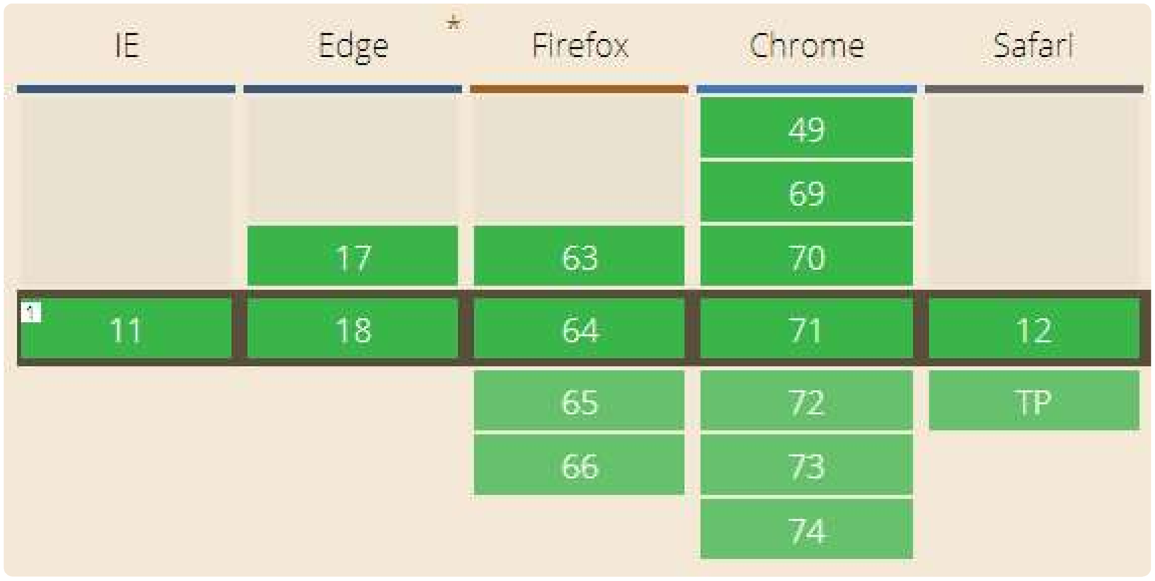
Notice the `'text/html'` passed to `getBlobURL`? We can change that too. Generating a CSS or JS blob is easy: Just pass `text/css` or `text/javascript` respectively.

Another benefit of blob URLs is that they stick around, and can be accessed any way that you would access a regular URL. Which means that we can *actually* load our CSS and JS files from a separate URL, so the "escaping" trick is no longer a problem.

Here's a bare-bones implementation of this in practice:

```
const getGeneratedPageURL = ({ html, css, js }) => {
  const getBlobURL = (code, type) => {
    const blob = new Blob([code], { type })
    return URL.createObjectURL(blob)
  }

  const cssURL = getBlobURL(css, 'text/css')
  const jsURL = getBlobURL(js, 'text/javascript')

  const source = `
    <html>
      <head>
        ${css && `<link rel="stylesheet" type="text/css" href="${cssURL}" />`}
        ${js && `<script src="${jsURL}"></script>`}
      </head>
      <body>
        ${html || ''}
      </body>
    </html>
  `

  return getBlobURL(source, 'text/html')
}

const url = getGeneratedPageURL({
  html: '<p>Hello, world!</p>',
  css: 'p { color: blue; }',
  js: 'console.log("hi")'
})

const iframe = document.querySelector('#iframe')
iframe.src = url
```

Oh, and browser support for Blob URLs is much better than srcdoc. ;)



IE	Edge *	Firefox	Chrome	Safari
			49	
			69	
	17	63	70	
1 11	18	64	71	12
		65	72	TP
		66	73	
			74	

The Moral?

Don't fight the language, I guess.

I knew what I wanted to do: Load data from URLs. It just never occurred to me to look for a non-hacky way to do just that!

Top comments (17)

Pratik Naik • Jul 16 '20

This was one of the most amazing solutions I have seen. I was constantly trying to load html contents inside iframe and all methods were failing. This method worked like a charm. Thank you for such marevellous solution and a great story.

Hendrik de Graaf • Apr 11 '19 • Edited

IE11 does support Blob URLs but not for type text/html and also not for usage in an iframe: caniuse.com/#feat=bloburls
Just putting this here so others don't waste time trying this solution as an alternative for srcDoc in IE11.
Try this one in IE11 codepen.io/grimen/pen/lBuiG/

Dragos Tudorache • May 22 '19

Thank you very very much, Hendrik! Indeed, text/html cannot be used in IE11

johnaweiss • Apr 27 '20 • Edited

Do blobs or the "text/css" type have a 32k-character limit? Varies between browsers?
stackoverflow.com/questions/695151...
ibm.com/support/knowledgecenter/SS...

Veer Bhadra Singh Solanki • May 12 '21 • Edited

Hi Josh, This was an amazing article and help me to resolve the error but the line for js inside the source variable, inside the head tag generates an error when I load my page then it shows js on the web pages. When I remove the js script "${js && `<script src="${jsURL}"></script>`}" then it works properly. Also console show error SyntaxError: unterminated string literal.

Victor Aremu • May 1 '19

Wow, I found this really helpful. Thanks!

Josh Pullen • May 1 '19

That's awesome to hear! Let me know if you have any questions. 😀

johnaweiss • Apr 27 '20 • Edited

Hi Josh, i'm eager to use blobs! To use your barebone code, don't i need to put something in the page-load event, so the iframe gets rendered before the blob-code runs? I mean, couldja put a lil' meat on the bones? :) Thx!

Josh Pullen • Apr 28 '20 • Edited

If your `<script>` is placed at the bottom of the `<body>`, it should run after the iframe exists in the DOM, so you're all good. If you're placing the `<script>` inside the `<head>` then you would need to grab the iframe and set its source after the body loads.

EhsanAgha • Jun 10 '19

hi
i have read this paper several times but i couldn't use it.
actually i need to insert a simple style CSS (.caption-overflow {overflow: hidden;}) to my iframe (<iframe allow='autoplay' id='iframe_1011998129' src="affstat.adro.co/imp/bXhIQnd0dUFxSE....) that i have inserted into my page (avang.ir/ssd-landing/).
i can't do it. i have used your code and no success yet.

Nick Pantelidis • May 22 '20

Thank you very much for this helpful article. It helped me a lot in my project. Thanks!

Josh Pullen • Jun 4 '20

That's good to hear! :)

Johnson awah Alfred • Dec 10 '19

Help me in my react project, Thanks!

Muhaimin CS • Jun 4 '19

right now I'm using zoid as it promise security as first layer. Do you have other alternative

Nenad V. • Jul 30 '20

Nice solution, thanks
Do you know maybe is it possible to add CDN reference somehow?

Rahul Patel • Aug 18 '22

I have tried this solution in my project but I got the [object HTMLDivElement] inside the iframe, why??

R Giraffe • Mar 2 '22

Hi Josh thanks a lot for this - is it possible to load into an iframe a combination of html and scripts with one long string using this method? Thanks! R

Code of Conduct • Report abuse

MongoDB PROMOTED

[MongoDB University | Get Started]

Learn the fundamentals - the document model, how to create CRUD operations, and more!

Start for Free

Josh Pullen

I love education and technology! If you ever want help with anything, please message me here on Dev, on Twitter (@PullJosh), or by email (hello@joshuapullen.com)

JOINED

Apr 25, 2017

More from Josh Pullen

The Transformative Power of Free
#webdev #showdev #startup

Project Idea: The RSS Reader for the End Times
#webdev #javascript #startup #productivity

Concerns with Separation of Concerns
#react #jsx #webdev

MongoDB PROMOTED

Build
Build
^ CTRL
MongoDB