Stim.js

Roy de Jong

Feb 10, 2023

CONTENTS

https://github.com/SoftwarePunt/stim.js

Warning: This project is still being prototyped and developed, and is not suitable for production use right now. Things may change or break at any time.

CHAPTER

ONE

INTRODUCTION

Stim.js is a lightweight, framework agnostic, JavaScript front-end framework designed to accelerate and add interactivity to server side rendering (SSR) apps.

Be productive! Write your application the “old fashioned” way with the server rendering HTML and receiving postbacks, and then use Stim.js to make it feel like a modern single-page application.

CHAPTER TWO

STIM.JS IS…

- **Lightweight:** a single file with zero dependencies and written in pure JavaScript.
- **Framework agnostic**: does not require any server side integration, and any optional integration is done through HTML attributes and generic request parameters.
- **An optional drop in:** it encourages you to design your application so that even if JavaScript is disabled, your application's core functionality will continue to work.

CHAPTER

THREE

CONTENTS

3.1 Installation

Stim.js ships as a single file with no dependencies. Just require it in a `<script>` tag and you're good to go!

3.1.1 Download

You can download the latest **stable version** from the GitHub releases page.

Or, you can download the latest **bleeding edge build** from GitHub actions. Find the latest successful build, and download it from the "Artifacts" section.

Also, the package is available on npm if that's your kind of thing: **stim.js**.

3.1.2 Setup

Simply require the script on your page:

```
<script src="stim.js" async defer></script>
```

Stim.js does not require any dependencies and will execute on `DOMContentLoaded`.

It is recommended to use `async` and `defer` so the script does not delay page loading or rendering in any way.

3.2 Links and inline loading

Stim.js accelerates internal page loads by (pre)loading them over XHR. This avoids a full page reload, which makes your app frontend feel significantly faster - like an SPA!

For links, preloading begins when the user *begins* clicking on the link (on mouse down or touch start). By the time the user releases their mouse button, the page is usually done loading, so it can be shown immediately.

Inline loading is also used by *Forms*.

Most of the Stim.js components will be registered or updated whenever a page loads, both on a hard reload an on an inline reload.

3.2.1 Binding links

When Stim.js *is first loaded*, and whenever it (pre)loads another page, all `<a>` tags in the DOM will be checked and bound for preloading, if they are compatible.

For a link to qualify, it must be a relative URL that opens in the current tab.

If `download` or `stim-ignore` attributes are set on the link, the link will always be ignored.

3.2.2 Preloading behavior

Preloading begins when a user begins clicking down (`mousedown`) on a bound link. For touch devices, this is on `touchstart`.

Stim.js will send an XHR `GET` request to the server and keep its response in memory. When the user releases the mouse button or otherwise finishes the "click", the page will be presented whenever it's ready - usually instantly.

If the link-click is aborted by the user, the preload will also be aborted if possible, and the DOM won't change and no redirects will happen under any circumstance.

If the link-click is finalized, but the XHR request failed, the browser will hard-navigate to the URL without XHR.

3.2.3 LoadingBar

Between release of the link click and the final page load, a loading bar appears on the top of the page by default. This helps the page feel responsive while the background load completes.

See *LoadingBar* for customization and options.

3.2.4 Integration

Manual navigation (`Stim.navigate()`)

You can use `Stim.navigate(url)` to manually perform an inline load from a script. Note that the URL will not be validated.

Opening links externally (`stim-ignore`)

If this attribute is set on a link, it will be ignored by Stim.js.

Breaking out of inline loads (`stim-kill` and `stim-zone`)

These attributes can be set on the `<body>` element to form a blacklist or whitelist for which pages can be loaded inline by Stim.js.

Whitelist mode: Once `stim-zone` is found on a body, a hard reload will be performed if a page is loaded that does *not* have the `stim-zone` attribute on its body.

Blacklist mode: If `stim-kill` is found on a body, a hard reload will be performed.

This is useful to prevent your page layout from breaking when Stim.js inadvertently loads an error page or external page that should not be loaded inline.

Updating actual URLs (`rel=canonical`)

The server can optionally send the canonical URL as a meta tag. If this is encountered on an inline load, it will be used for the address bar and browser history. This may be useful because Stim.js may not be able to detect URL redirects properly.

Executing scripts (`stim-run`)

By adding a `stim-run` attribute to `<script>` tag, that script will run whenever a page is loaded inline.

Event: Before commit (`stim-before-commit`)

The `stim-before-commit` event is our internal equivalent of `beforeunload`. It is triggered when a preload is committed (i.e. on mouse click release).

If default is prevented on this event, the commit will not proceed. Instead, you should manually commit or abort the preload.

For example, if you want a nice integrated prompt:

```
window.addEventListener('stim-before-commit', (e) => {
  e.preventDefault();
  this.exampleAsyncPromptFunction("Are you sure you want to do that?")
    .then((result) => {
      if (result)
        e.detail.preload.commit(true); // arg true for `skipEvent`, to prevent loop
      else
        e.detail.preload.abort();
    });
});
```

Event: Before DOM change (`stim-before-change`)

The `stim-before-change` event will be emitted right before the DOM is changed.

Event: Inline load complete (`stim-load`)

The `stim-load` event will be emitted on the `document` as soon as an inline load is fully complete and the DOM has been changed. This event is not emitted on the initial page load.

3.3 Forms

Stim.js can accelerate form submissions, similar to *link preloading*.

If configured, forms will load inline and the POST request will be sent over XHR to avoid a full page reload.

3.3.1 Configuration

To enable this feature, set the `stim-post` property on a `<form>` tag:

```
<form method="post" action="/target" stim-post>
```

The value of `stim-post` can be set to a custom target URL. If set without a value, the `action` will be used instead.

Only `post` (default) and `get` methods are supported:

POST forms

A request is sent to the target URL with `FormData`. This mimics a regular browser form submission.

GET forms

The target URL will be loaded with its query string. This is an internal page load via `Stim.navigate()`.

3.3.2 API

Forms will be scanned and configured whenever Stim.js first loads, and whenever it navigates to another page through *preloading*.

If the contents of your page have dynamically changed, you can force a manual update as well:

```
Stim.Forms.update();
```

3.3.3 stim-before-submit

Before a form is submitted, Stim.js will emit the `stim-before-submit` event on the document. This event is cancelable to prevent the submission from going ahead.

- `event.detail.form` - references the form element
- `event.detail.url` - the URL Stim.js will submit the form to via POST

3.4 Templates

Templates are a core part of Stim.js. They are used for re-usable and dynamic components like *Tooltips* and *ContextMenu*.

3.4.1 Structure

A template is defined in HTML and must be rendered by the server at least once before it can be used.

Note: Stim.js will cache a template once it encounters one, so the server can optimize its renders by only sending static templates on the initial full-page load.

Here's what a template might look like, as an example for a tooltip:

```html
<div stim-template="my-tooltip-template" class="custom-tooltip">
    <span stim-data="title">Placeholder text</span>
</div>

<a href="#" title="My tooltip text" stim-tooltip="my-tooltip-template">Hover me!</a>
```

All templates follow this same general format with `stim-` attributes.

The component you use affects the available data and options on a template. In this example, with a *Tooltip component*, we'll have `title` data available to us and `stim-position` becomes a possible option.

3.4.2 Attributes

Here's an explanation of the `stim-` attributes that can be used with templates:

3.4.3 Variables

Variables can be injected into templates so it's easier to reuse them.

Which variables are available depends on the component you're using templates with. In most cases this is a set of predefined variables, plus any `data-` properties you have set on the bound element.

These variables can be injected in the template in one of two ways:

- The **`stim-data` attribute** can be used to set the text content of a specific element to the value of the variable.
- The **variable syntax** can be used to replace the value in the raw template HTML before it is instantiated on the DOM.

The variable syntax looks like this: `[%%varname%%]`.

Here is what that might look like when combined with a context menu:

```html
<div stim-template="my-ctx-template">
    <a href="/goto/entity/[%%id%%]">Go to the entity</a>
</div>

<a href="#" stim-menu="my-ctx-template" data-id="123">Open the menu</a>

<!-- The above will render in the DOM as follows when instantiated: -->
<div stim-mounted="true" stim-instance-id="0">
    <a href="/goto/entity/123">Go to the entity</a>
</div>
```

3.4.4 Styling

The server is responsible for rendering HTML and applying styles itself.

It's good practice to make template elements invisible by default:

```
*[stim-template] {
    display: none !important;
}
```

You can approach it the other way around as well - when a template is instantiated, the `stim-mounted` attribute will be set on that instance. This allows you to add specific visibility styles:

```
.my-template[stim-mounted] {
    display: flex;
}
```

3.5 Tooltips

Stim.js can add fancy tooltips to your application, acting as a drop-in replacement for `title` properties on various HTML tags.

3.5.1 Configuration

To enable this feature, set the `stim-tooltip` property on an existing HTML tag:

```
<a href="#" title="Plain old fallback tooltip" stim-tooltip="left">Example tag</a>
```

The value of `stim-tooltip` should be set to a *Template ID*. If the template cannot be resolved, a warning will be logged and fancy tooltips won't be enabled for that element.

This approach has some benefits. Fancy tooltips will only appear in the places you want them to, using a specific template. And if scripts are disabled, native tooltips will simply continue to work through the title attribute.

3.5.2 API

Tooltips will be scanned and configured whenever Stim.js first loads, and whenever it navigates to another page through *preloading*.

If the contents of your page have dynamically changed, you can force a manual update as well:

```
Stim.Tooltips.update();
```

Note: You do not need to call this function if the title has changed, only if new tooltip elements need to be registered.

3.5.3 Template

Attributes

The following attributes can be set on the template as defaults, or the tooltip source element itself:

stim-position

Controls how the tooltip should be positioned. Options:

- `hover` (default): the element will be positioned automatically, like a regular tooltip.

Data

title

The original value of the title attribute.

3.5.4 Example

```
<div stim-template="tt-left" stim-position="hover" class="my-tooltip">
    <span stim-data="title">...</span>
</div>
<a href="#" title="Tooltip text" stim-tooltip="tt-left">Hover me!</a>
```

3.6 ContextMenu

Pop-out menus that can be opened to reveal additional context options. They are bound to links or programmatically opened.

3.6.1 Usage

Creating menus

To set up a context menu, first define a *Template* for it:

```
<div stim-template="my-menu-template" class="my-menu-class">
    <a href="#">Some link</a>
    <a href="#">Another link</a>
    <div class="my-separator"></div>
    <a href="#">One more link</a>
</div>
```

Binding to links

You can bind a link to a context menu by setting the `stim-menu` property on it, with a value referring to the menu template:

```
<a href="#" stim-menu="my-menu-template">
    <img src="open_context_menu.png"/>
</a>
```

When the user clicks the link, a context menu will open, its location anchored to the link clicked.

Scripting menus

You can also manually open and close context menus.

To open a context menu at an element's position:

```
Stim.ContextMenu.openOnElement(element, 'my-menu-template');
```

To close all menus:

```
Stim.ContextMenu.closeAll();
```

Binding data

You can bind data to the trigger element by setting `data-` properties on it. These will become available in the template for injection. This makes it easier to re-use the same menu template for multiple items.

For example:

```
<div stim-template="my-ctx-template">
    <a href="/goto/entity/[%%id%%]">Go to the entity</a>
</div>

<a href="#" stim-menu="my-ctx-template" data-id="123">Open the menu</a>
<a href="#" stim-menu="my-ctx-template" data-id="456">Open another menu</a>

<!-- The above will render in the DOM as follows when the first link is clicked: -->
<div stim-mounted="true" stim-instance-id="0">
    <a href="/goto/entity/123">Go to the entity</a>
</div>
```

3.7 Autoscroll

Autoscroll is a small convenience function that will automatically scroll to the bottom of a scrollable container on page (re)load.

This is useful for certain UIs, like when showing a page with a conversation view.

3.7.1 Configuration

To enable autoscroll on an element, add the `stim-autoscroll` attribute:

```html
<div class="my-scroll-container" stim-autoscroll>
    ...
</div>
```

3.7.2 API

You can use the API to manually control or apply autoscrolling behavior.

Update all components

```
Stim.Autoscroll.update();
```

This normally happens on page (re)load; all elements on the page with `stim-autoscroll` are processed and scrolled.

3.8 Tabs

Tabs can be used to switch the content of an element on click.

```html
<ul class="my-tab-links">
    <a class="my-tab-link" href="#tc3" stim-tab>Contact</a>
    <a class="my-tab-link" href="#tc3" stim-tab>Contact</a>
</ul>
<div class="my-tab-contents">
    <div class="my-tab-pane" id="tc1">Tab content 1</div>
    <div class="my-tab-pane" id="tc2">Tab content 2</div>
    <div class="my-tab-pane" id="tc3">Tab content 3</div>
</div>
```

All switchable elements should have a common parent, and an `id` attribute that is unique on the page.

The clickable links should have the `stim-tab` attribute, and their `href` should point to the id-anchor of the content element.

When a tab link is clicked, all other elements under the common parent will be hidden, and only the corresponding content element will be made visible.

3.8.1 Styling

When switching between tabs, the content style will be toggled between `display: block` and `display: none` by Stim.js automatically.

Active tabs and active content elements will get the `stim-active` attribute, which allows you to apply custom styles as desired:

```css
.my-tab-link[stim-active] {
    font-weight: bold;
    color: blue;
}

.my-tab-pane[stim-active] {
    display: flex !important;
}
```

3.8.2 API

Update all components

```
Stim.Tabs.update();
```

This normally happens on page (re)load; all elements on the page with `stim-tab` are evaluated and bound as needed.

3.9 Modals

Modals are internal dialog / pop-up windows.

3.9.1 Draggable windows

To turn an element into a draggable window, simply add the `stim-modal` attribute.

Then, define regions the user can grab and drag by adding the `stim-handle` attribute to one or more child elements.

Persisting positions

You can persist modal positions by adding a unique `stim-id` attribute, and setting the `stim-modal` attribute to `persist`. For example:

```html
<div class="my-modal" stim-id="my-unique-modal-id" stim-modal="persist">
    <div stim-handle style="padding: 15px; border: 1px solid red;">
        <h1>Drag me around and remember me!</h1>
    </div>
</div>
```

3.10 LoadingBar

The loading bar is a horizontal progress bar that appears at the top of the page. It simulates fake progress with a simple animation. This helps the page feel "alive" and responsive while a background request completes.

3.10.1 Behavior

When loading begins, the bar fades in and will animate from about 20% width towards 100%.

The animation will slow down the longer the request takes to complete, but will keep animating until it reaches 95%.

If the request completes fairly quickly (~250ms), the loading bar will not appear at all. This is fairly common for Stim.js preloads.

3.10.2 Stim Loading

By default, the loading bar appears whenever Stim.js *internally loads a page*. For link clicks, the loading bar only appears between the click release and final load.

You can completely disable this behavior if you prefer:

```
Stim.LoadingBar.setHandlePageLoads(false); // don't show for internal loads
```

3.10.3 Manual Use

To start the progress bar:

```
Stim.LoadingBar.start(); // fade in and begin animation (~250ms)
```

To finish the progress bar:

```
Stim.LoadingBar.finish(); // run to 100% and then fade out
```

To immediately cancel/abort the progress bar:

```
Stim.LoadingBar.stop(); // immediately stop and fade out
```

3.10.4 Styles

You can choose which built-in style to use:

```
Stim.LoadingBar.setStyle(LoadingBarStyle.Blue); // default style
Stim.LoadingBar.setStyle(LoadingBarStyle.None); // remove all styles
```

You can manually define/override CSS styles by targeting the `.stim-loadingbar` class.