JavaScript

Contents

Individual or compiled

Plugins can be included individually (using Bootstrap's individual `js/dist/*.js`), or all at once using `bootstrap.js` or the minified `bootstrap.min.js` (don't include both).

If you use a bundler (Webpack, Rollup...), you can use `/js/dist/*.js` files which are UMD ready.

Using Bootstrap as a module

We provide a version of Bootstrap built as `ESM` (`bootstrap.esm.js` and `bootstrap.esm.min.js`) which allows you to use Bootstrap as a module in your browser, if your targeted browsers support it.

```
<script type="module">
  import { Toast } from 'bootstrap.esm.min.js'

  Array.from(document.querySelectorAll('.toast'))
```

```
    .forEach(toastNode => new Toast(toastNode))
</script>
```

{{< callout warning >}} ## Incompatible plugins

Due to browser limitations, some of our plugins, namely Dropdown, Tooltip and Popover plugins, cannot be used in a `<script>` tag with `module` type because they depend on Popper. For more information about the issue see here. {{< /callout >}}

Dependencies

Some plugins and CSS components depend on other plugins. If you include plugins individually, make sure to check for these dependencies in the docs.

Our dropdowns, popovers and tooltips also depend on Popper.

Still want to use jQuery? It's possible!

Bootstrap 5 is designed to be used without jQuery, but it's still possible to use our components with jQuery. **If Bootstrap detects `jQuery` in the `window` object** it'll add all of our components in jQuery's plugin system; this means you'll be able to do `$('[data-bs-toggle="tooltip"]').tooltip()` to enable tooltips. The same goes for our other components.

Data attributes

Nearly all Bootstrap plugins can be enabled and configured through HTML alone with data attributes (our preferred way of using JavaScript functionality). Be sure to **only use one set of data attributes on a single element** (e.g., you cannot trigger a tooltip and modal from the same button.)

{{< callout warning >}} ## Selectors

Currently to query DOM elements we use the native methods `querySelector` and `querySelectorAll` for performance reasons, so you have to use valid selectors. If you use special selectors, for example: `collapse:Example` be sure to escape them. {{< /callout >}}

Events

Bootstrap provides custom events for most plugins' unique actions. Generally, these come in an infinitive and past participle form - where the infinitive (ex. `show`) is triggered at the start of an event, and its past participle form (ex. `shown`) is triggered on the completion of an action.

All infinitive events provide `preventDefault()` functionality. This provides the ability to stop the execution of an action before it starts. Returning false from an event handler will also automatically call `preventDefault()`.

```
var myModal = document.getElementById('myModal')

myModal.addEventListener('show.bs.modal', function (event) {
  if (!data) {
    return event.preventDefault() // stops modal from being shown
  }
})
```

{{< callout warning >}} ## jQuery events

Bootstrap will detect jQuery if `jQuery` is present in the `window` object and there is no `data-bs-no-jquery` attribute set on `<body>`. If jQuery is found, Bootstrap will emit events thanks to jQuery's event system. So if you want to listen to Bootstrap's events, you'll have to use the jQuery methods (`.on`, `.one`) instead of `addEventListener`.

```
$('#myTab a').on('shown.bs.tab', function () {
  // do something...
})
```

{{< /callout >}}

Programmatic API

All constructors accept an optional options object or nothing (which initiates a plugin with its default behavior):

```
var myModalEl = document.getElementById('myModal')

var modal = new bootstrap.Modal(myModalEl) // initialized with defaults
var modal = new bootstrap.Modal(myModalEl, { keyboard: false }) // initialized with no keyb
```

If you'd like to get a particular plugin instance, each plugin exposes a `getInstance` method. In order to retrieve it directly from an element, do this: `bootstrap.Popover.getInstance(myPopoverEl)`.

CSS selectors in constructors

You can also use a CSS selector as the first argument instead of a DOM element to initialize the plugin. Currently the element for the plugin is found by the `querySelector` method since our plugins support a single element only.

```
var modal = new bootstrap.Modal('#myModal')
var dropdown = new bootstrap.Dropdown('[data-bs-toggle="dropdown"]')
```

Asynchronous functions and transitions

All programmatic API methods are **asynchronous** and return to the caller once the transition is started but **before it ends**.

In order to execute an action once the transition is complete, you can listen to the corresponding event.

```
var myCollapseEl = document.getElementById('myCollapse')

myCollapseEl.addEventListener('shown.bs.collapse', function (event) {
  // Action to execute once the collapsible area is expanded
})
```

In addition a method call on a **transitioning component will be ignored**.

```
var myCarouselEl = document.getElementById('myCarousel')
var carousel = bootstrap.Carousel.getInstance(myCarouselEl) // Retrieve a Carousel instance

myCarouselEl.addEventListener('slid.bs.carousel', function (event) {
  carousel.to('2') // Will slide to the slide 2 as soon as the transition to slide 1 is fin
})

carousel.to('1') // Will start sliding to the slide 1 and returns to the caller
carousel.to('2') // !! Will be ignored, as the transition to the slide 1 is not finished !!
```

Default settings

You can change the default settings for a plugin by modifying the plugin's `Constructor.Default` object:

```
// changes default for the modal plugin's `keyboard` option to false
bootstrap.Modal.Default.keyboard = false
```

No conflict (only if you use jQuery)

Sometimes it is necessary to use Bootstrap plugins with other UI frameworks. In these circumstances, namespace collisions can occasionally occur. If this happens, you may call `.noConflict` on the plugin you wish to revert the value of.

```
var bootstrapButton = $.fn.button.noConflict() // return $.fn.button to previously assigned
$.fn.bootstrapBtn = bootstrapButton // give $().bootstrapBtn the Bootstrap functionality
```

Version numbers

The version of each of Bootstrap's plugins can be accessed via the `VERSION` property of the plugin's constructor. For example, for the tooltip plugin:

```
bootstrap.Tooltip.VERSION // => "{{< param current_version >}}"
```

No special fallbacks when JavaScript is disabled

Bootstrap's plugins don't fall back particularly gracefully when JavaScript is disabled. If you care about the user experience in this case, use `<noscript>` to

explain the situation (and how to re-enable JavaScript) to your users, and/or add your own custom fallbacks.

{{< callout warning >}} ##### Third-party libraries

Bootstrap does not officially support third-party JavaScript libraries like Prototype or jQuery UI. Despite `.noConflict` and namespaced events, there may be compatibility problems that you need to fix on your own. {{< /callout >}}

Sanitizer

Tooltips and Popovers use our built-in sanitizer to sanitize options which accept HTML.

The default `allowList` value is the following:

```
var ARIA_ATTRIBUTE_PATTERN = /^aria-[\w-]*$/i
var DefaultAllowlist = {
  // Global attributes allowed on any supplied element below.
  '*': ['class', 'dir', 'id', 'lang', 'role', ARIA_ATTRIBUTE_PATTERN],
  a: ['target', 'href', 'title', 'rel'],
  area: [],
  b: [],
  br: [],
  col: [],
  code: [],
  div: [],
  em: [],
  hr: [],
  h1: [],
  h2: [],
  h3: [],
  h4: [],
  h5: [],
  h6: [],
  i: [],
  img: ['src', 'srcset', 'alt', 'title', 'width', 'height'],
  li: [],
  ol: [],
  p: [],
  pre: [],
  s: [],
  small: [],
  span: [],
  sub: [],
  sup: [],
  strong: [],
```

```
  u: [],
  ul: []
}
```

If you want to add new values to this default `allowList` you can do the following:

```
var myDefaultAllowList = bootstrap.Tooltip.Default.allowList

// To allow table elements
myDefaultAllowList.table = []

// To allow td elements and data-bs-option attributes on td elements
myDefaultAllowList.td = ['data-bs-option']

// You can push your custom regex to validate your attributes.
// Be careful about your regular expressions being too lax
var myCustomRegex = /^data-my-app-[\w-]+/
myDefaultAllowList['*'].push(myCustomRegex)
```

If you want to bypass our sanitizer because you prefer to use a dedicated library, for example DOMPurify, you should do the following:

```
var yourTooltipEl = document.getElementById('yourTooltip')
var tooltip = new bootstrap.Tooltip(yourTooltipEl, {
  sanitizeFn: function (content) {
    return DOMPurify.sanitize(content)
  }
})
```