RoR - Turbolinks

Overview

Turbolinks provides the performance benefits of a single page application, without the complexity of a client side JavaScript framework. When a user follows a link, turbolinks fetches the server rendered html as normal, then swaps in its body and merges the head without a full page load.

Ruby on Rails v6 includes turbolinks as standard, and while the gem is generally plug and play, due to the page not reloading the design of some client side features such as JavaScript packs must be changed.

How it Works

Turbolinks intercepts user clicks on <a> tags, changes the browser url using the History API, performs a XMLHttpRequest, then renders the html response. Each user navigation is called a visit, of which there are two types: application (advance - standard or replace - no added history) and restore (refresh).

Application visits are performed when a user clicks a link, or Turbolinks.visit(location) is called. Replace visits can be specified using the turbolinks action tag in the <a> element, or action hash:

<a href="/edit" data-turbolinks-action="replace">Edit</a>

Turbolinks.visit("/edit", { action: "replace" })

Restore visits can be performed with a similar syntax action to the above.

Disabling Turbolinks

Turbolinks can be disabled on specific <a> elements using the turbolinks data tag either on the specific link, or in its ancestor:

<a href="/" data-turbolinks="false">Disabled</a>

<div data-turbolinks="false">

<a href="/">Disabled</a>

</div>

Similarly use the true option to enable a tag with a disabled ancestor.

JavaScript Elements

The initial page load will load and evaluate script elements as normal, however on new turbolink visits, JavaScript actions are different.

When navigating to a new page using turbolinks, any script elements contained in the head of the new page will appeded to the current head, where they are loaded and evaluated. Script elements contained in the body will be loaded and evaluated each time the page is visited.

Navigation Events

Since turboloinks effectivily makes the webiste into a single page app, the using traditional window.onload will only work the first time the window is loaded. Therefore, different event listeners must be used to achieve the desired JavaScript window.onload behaviour.

Turbolinks has various different events which occur throughout user navigation:

* click - on each click of an enabled link, url in event.data.url
* before-visit - before visiting a location
* visit - immediately after visit starts
* request-start - before xmlhttprequest, xml object access in event.data.xhr
* request-end - after xmlhttprequest completes, xml object access in event.data.xhr
* before-cache - just before rendering a new page as turbolinks will cache the current page to allow for quicker perceived access when revisiting the current page.
* before-render - before rendering the page, new <body> access with event.data.newBody
* render - after new page render, twice during application visit to cached location, initial is cached version, second is fresh version
* load - on initial page load and after each turbolinks visit

JavaScript - Cleanup

Since turbolinks does not reload the page, the application DOM may not be in clean condition when the turbolinks navigation events such as 'load' are fired. Therefore, if there are event listeners included from previous pages which are not removed, they could error on the load of a new page with a different body.

Where possible functions which transform element in the DOM should be idempotent, meaning they can be performed more than once without any ill effect. To do this, it is common to add a data attribute to the element to confirm whether an action has been performed on it or the element is processed. The data attribute can then be checked and updated as required on page reloads. If the function changes or adds an element, it can be easier to simply check the for change during the element selection.

However, if the application relies on event listeners, javascript packs which are used on a specific page will need to clean themselves up (remove added event listeners) once a user visits another page. The 'turbolinks:before-cache' event can be used trigger cleanup, since it is called before each visit, except for the initial page load. For example:

export default function adminDashboardToggle() {

var dashboardToggleDiv = document.querySelector('.admin\_dashboard\_toggle')

function toggleDashboard() {

if (dashboardToggleDiv.classList.contains('open')) {

dashboardToggleDiv.classList.remove('open')

} else {

dashboardToggleDiv.classList.add('open')

}

}

dashboardToggleDiv.addEventListener('click', toggleDashboard)

function cleanUpToggleDashboard() {

if (dashboardToggleDiv.classList.contains('open')) {

dashboardToggleDiv.classList.remove('open')

}

dashboardToggleDiv.removeEventListener('click', toggleDashboard)

document.removeEventListener('turbolinks:before-cache', cleanUpToggleDashboard)

}

document.addEventListener('turbolinks:before-cache', cleanUpToggleDashboard)

}

JavaScript - Event Delegation

Another way to prevent issues with event listeners is by using javascript event delegation to select the correct function based on the event, normally by element clicked.

To perform event delegation create an event listener for the document click event and pass through the event variable. The event variable will then contain the information of the click target, which can be compared against the targets of application events:

document.addEventListener('click', event => {

if (event.target.class == 'desired\_target') {

// run event

}

})

Permanent Elements

Elements can be marked as permanent, making them persist across page visits. An example would be a shopping cart, not reducing number of items due to a cached copy being loaded on the user pressing the back button.

<div id="cart-counter" data-turbolinks-permanent>1 item</div>

Application JavaScript

Javascript added to the application.js file in Rails will only be run once, on the initial page load. Therefore javascript which is general to the application, such as a navbar action can be placed in a 'turbolinks:load' event listener, and it will be run on each page visit but not duplicated.