← Volver a la lista de capítulos

Impulsión turbo

Publicado el 23 de enero de 2022

En este capítulo, explicaremos qué es Turbo Drive y cómo acelera nuestras aplicaciones Ruby on Rails al convertir todos los clics en enlaces y envíos de formularios en solicitudes AJAX.

Entendiendo qué es Turbo Drive

Turbo Drive es la **primera parte de Turbo** , que se instala de forma predeterminada en las aplicaciones Rails 7, como podemos ver en nuestro `Gemfile` archivo de manifiesto de JavaScript `application.js` :

```
# Gemfile

# Hotwire's SPA-like page accelerator [https://turbo.hotwired.dev]
gem "turbo-rails"
```

```
// app/javascript/application.js

// Entry point for the build script in your package.json
import "@hotwired/turbo-rails"
import "./controllers"
```

De forma predeterminada, Turbo Drive acelera nuestras aplicaciones Ruby on Rails al convertir todos los clics en enlaces y envíos de formularios en

solicitudes AJAX. Eso significa que ***nuestra aplicación CRUD del primer capítulo ya es una aplicación de una sola página y no tuvimos que escribir ningún código personalizado*** .

Con Turbo Drive, nuestras aplicaciones Ruby on Rails serán **rápidas por defecto** porque la página HTML que visitemos por primera vez no se actualizará por completo. Cuando Turbo Drive intercepta un clic en un enlace o el envío de un formulario, la respuesta a la solicitud AJAX solo servirá para reemplazar el `<body>` de la página HTML. En *la mayoría de los casos* , el `<head>` de la página HTML actual no cambiará, lo que se traducirá en una mejora considerable del rendimiento: **las solicitudes para descargar las fuentes, los archivos CSS y JavaScript solo se realizarán *una vez* cuando accedamos por primera vez al sitio web** .

¿Cómo funciona Turbo Drive?

Turbo Drive funciona interceptando eventos de "clic" en enlaces y eventos de "envío" en formularios.

Cada vez que se hace clic en un enlace, Turbo Drive intercepta el evento "clic", anula el comportamiento predeterminado transformando la solicitud HTML que el clic en el enlace desencadenaría normalmente en una solicitud AJAX. Cuando Turbo Drive recibe la respuesta, reemplaza el `<body>` de la página actual con el `<body>` de la respuesta, dejando el `<head>` sin cambios *en la mayoría de los casos* .

Por eso, las nuevas aplicaciones Ruby on Rails 7 son aplicaciones *de una sola página* de manera predeterminada. La página que visitemos primero no se reemplazará por completo, solo `<body>` se reemplazará la etiqueta.

Para los clics en enlaces, una implementación de pseudocódigo se vería así:

```
// Select all links on the page
const links = document.querySelectorAll("a");

// Add a "click" event listener on each link to intercept the click
// and override the default behavior
links.forEach((link) => {
  link.addEventListener("click", (event) => {
    // Override default behavior
```

```
    event.preventDefault()
    // Convert the click on the link into an AJAX request
    // Replace the current page's <body> with the <body> of the response
    // and leave the <head> unchanged
  }
)});
```

La misma lógica se aplica a los envíos de formularios. Cuando se envía un formulario, Turbo Drive intercepta el evento "enviar" y anula el comportamiento predeterminado transformando el envío del formulario en una solicitud AJAX y reemplazando el `<body>` de la página actual por el `<body>` de la respuesta, sin `<head>` modificarlo.

Para el envío de formularios, una implementación de pseudocódigo se vería así:

```
// Select all forms on the page
const forms = document.querySelectorAll("form");

// Add a "submit" event listener on each form to intercept the submissio
// and override the default behavior
forms.forEach((form) => {
  form.addEventListener("submit", (event) => {
  // Override default behavior
    event.preventDefault()
    // Convert the form submission into an AJAX request
    // Replace the current page's <body> with the <body> of the response
    // and leave the <head> unchanged
  }
)});
```

Como se explicó en el capítulo 1 , hay un cambio importante en Rails 7: **los envíos de formularios no válidos deben devolver un código de estado 422 para que Turbo Drive reemplace el `<body>` de la página y muestre los errores de formulario** . El alias para el código de estado 422 en Rails es `:unprocessable_entity` . Por eso, desde Ruby on Rails 7, el generador de andamiaje agrega `status: :unprocessable_entity` y `#create` acciones `#update` cuando el recurso no se pudo guardar debido a un envío de formulario no válido.

Nota : Si ha trabajado con Rails durante algún tiempo, es posible que conozca Turbolinks. Turbolinks es el antecesor de Turbo Drive: solo interceptaba clics en enlaces, pero no envíos de formularios. Ahora que Turbo también gestiona envíos de formularios, los autores cambiaron el nombre de la biblioteca de Turbolinks a Turbo Drive.

Desactivación del Turbo Drive

Es posible que queramos desactivar Turbo Drive para determinados clics en enlaces o envíos de formularios en algunos casos. Por ejemplo, este puede ser el caso cuando se trabaja con gemas que aún no son compatibles con Turbo Drive.

Al momento de escribir este capítulo, la gema Devise no es compatible con Turbo Drive. Una buena solución alternativa es deshabilitar Turbo Drive en los formularios de Devise, como los formularios de inicio de sesión y registro. Volveremos a tratar este problema en un capítulo futuro, pero por ahora, aprendamos a deshabilitar Turbo Drive en enlaces y formularios específicos.

Para deshabilitar Turbo Drive en un enlace o formulario, debemos agregarle el `data-turbo="false"` atributo de datos.

En la `Quotes#index` página, desactivemos Turbo Drive en el enlace "Nueva cotización":

```
<main class="container">
  <div class="header">
    <h1>Quotes</h1>
    <%= link_to "New quote",
                new_quote_path,
                class: "btn btn--primary",
                data: { turbo: false } %>
  </div>

  <%= render @quotes %>
</main>
```

Ahora, vamos a actualizar la página y hacer clic en el enlace "Nueva cotización". Notaremos el desagradable "parpadeo" blanco de la actualización completa de la página HTML. Si realizamos esta acción nuevamente con las

herramientas de desarrollo abiertas en la pestaña "Red", notaremos que el navegador realiza cuatro solicitudes al servidor:

- Una solicitud **HTML** para cargar la `Quotes#new` página HTML
- Una solicitud para cargar el paquete CSS
- Una solicitud para cargar el paquete de JavaScript
- Una solicitud para cargar el favicon de la página.

Nota : Estoy usando Google Chrome para realizar este experimento. Es posible que veas resultados ligeramente diferentes con otros navegadores.

Ahora, volvamos a agregar Turbo Drive eliminando el `data-turbo="false"` del enlace "Nueva cotización", refresquemos la página y realicemos el experimento nuevamente. No veremos el desagradable "parpadeo" blanco porque el navegador no recarga completamente la página. En segundo plano, Turbo Drive convierte el clic en una solicitud AJAX e intercambia el `<body>` de la página con el de la respuesta `<body>` . En las herramientas de desarrollo, deberíamos ver solo dos solicitudes:

- Una solicitud **AJAX** para obtener el HTML de la `Quotes#new` página
- Una solicitud para cargar el favicon de la página.

Podemos realizar el mismo experimento en el formulario de cotizaciones. Desactivemos Turbo Drive en él:

```
<%= simple_form_for quote,
                    html: {
                      class: "quote form",
                      data: { turbo: false }
                    } do |f| %>

  <% if quote.errors.any? %>
    <div class="error-message">
      <%= quote.errors.full_messages.to_sentence.capitalize %>
    </div>
  <% end %>

  <%= f.input :name, input_html: { autofocus: true } %>
```

```
  <%= f.submit class: "btn btn--secondary" %>
<% end %>
```

Actualicemos nuestra página y creemos una nueva cotización con las herramientas de desarrollo abiertas en la pestaña "Red". Deberíamos ver el desagradable "parpadeo" blanco durante toda la actualización de la página HTML y cinco solicitudes:

- Una solicitud **HTML** para enviar el formulario
- La redirección **HTML** a la `Quotes#index page`
- Una solicitud para cargar el paquete CSS
- Una solicitud para cargar el paquete de JavaScript
- Una solicitud para cargar el favicon de la página.

Eliminemos la línea que acabamos de agregar para volver a habilitar Turbo Drive, actualizar la página y crear otra cita con las herramientas de desarrollo abiertas. No deberíamos ver el desagradable "parpadeo" blanco y solo deberíamos ver tres solicitudes:

- Una solicitud **AJAX** para enviar el formulario
- La redirección **AJAX** a la `Quotes#index page`
- Una solicitud para cargar el favicon de la página.

Demostramos lo que Turbo Drive hace por nosotros en las nuevas aplicaciones Ruby on Rails 7.

- Convierte todos los clics en enlaces y envíos de formularios en solicitudes AJAX para acelerar nuestra aplicación.
- Evita que el navegador realice demasiadas solicitudes para cargar archivos CSS y JavaScript.

Lo mejor es que no tuvimos que escribir ningún código personalizado. ¡Obtenemos este beneficio gratis!

Nota : También es posible desactivar Turbo Drive para toda la aplicación, aunque no recomiendo hacerlo ya que perderá los beneficios de velocidad que proporciona Turbo Drive.

Para desactivar Turbo Drive en toda la aplicación, tenemos que añadir dos líneas de configuración a nuestro código JavaScript. Por ejemplo, puedes hacerlo directamente en el archivo de manifiesto:

```
// app/javascript/application.js

import { Turbo } from "@hotwired/turbo-rails"
Turbo.session.drive = false
```

Recargando la página con data-turbo-track="reload"

En *la mayoría de los casos* , Turbo Drive solo reemplaza el contenido `<body>` de la página HTML y lo deja `<head>` sin cambios. Digo en *la mayoría de los casos* porque hay situaciones en las que queremos que Turbo Drive note cambios en el contenido `<head>` de nuestras páginas web.

Tomemos como ejemplo una implementación en la que cambiamos el CSS de nuestra aplicación. Gracias a la canalización de activos, la ruta a nuestro paquete CSS cambiará de `/assets/application-oldfingerprint.css` a `/assets/application-newfingerprint.css` . Sin embargo, si `<head>` nunca cambiara, los usuarios que estaban en el sitio web antes de la implementación y que permanecieron en el sitio web después de la implementación seguirían usando el paquete CSS anterior, ya que no se enviaría ninguna solicitud para descargar el nuevo paquete. Esto podría dañar la experiencia del usuario, ya que los usuarios usarían CSS obsoleto. Tenemos el mismo problema con nuestro paquete JavaScript.

Para solucionar este problema, en cada nueva solicitud, Turbo Drive compara los elementos DOM `data-turbo-track="reload"` de `<head>` la página HTML actual y los `<head>` de la respuesta. Si hay diferencias, Turbo Drive recargará toda la página.

Si observamos el diseño de la aplicación de nuestra aplicación, notaremos que ambas etiquetas de activos se generaron con el atributo de datos `data-turbo-track="reload"` :

```
<%# app/views/layouts/application.html.erb %>

<%= stylesheet_link_tag "application", "data-turbo-track": "reload" %>
<%= javascript_include_tag "application", "data-turbo-track": "reload",
```

Ahora que sabemos qué `data-turbo-track="reload"` hace el atributo de datos, un buen ejercicio sería demostrar que funciona como se espera.

Con las herramientas de desarrollo abiertas, hagamos clic en algunos enlaces en nuestro editor de citas. Deberíamos ver que el navegador solo envía solicitudes AJAX para recuperar el HTML de la página siguiente, como se mencionó en la sección anterior.

Ahora, hagamos un cambio temporal tonto en nuestro manifiesto CSS para simular un cambio en nuestro paquete CSS y una implementación, por ejemplo, importando el código del `.btn` componente dos veces:

```
// app/assets/stylesheets/application.sass.scss

// Remove the double import after the experiment
@import "components/btn";
@import "components/btn";
```

La próxima vez que hagamos clic en un enlace, deberíamos ver que la página se recarga por completo. ¡Hagamos una prueba y veamos si funciona! ¡Turbo Drive es un software fantástico!

Nota : Pequeños experimentos como el que acabamos de realizar nos ayudan a entender en profundidad lo que está sucediendo. ¡Haremos otros experimentos en los siguientes capítulos de este tutorial!

Cambiar el estilo de la barra de progreso de Turbo Drive

Como Turbo Drive anula el comportamiento predeterminado del navegador para los clics en enlaces y envíos de formularios, la barra de progreso/los

cargadores predeterminados del navegador ya no funcionarán como se esperaba.

Turbo nos respalda y tiene un reemplazo integrado para la barra de progreso predeterminada del navegador, ¡y podemos diseñarla para que se adapte al sistema de diseño de nuestra aplicación! Diseñemos la barra de progreso de Turbo Drive antes de pasar al siguiente capítulo:

```
// app/assets/stylesheets/components/_turbo_progress_bar.scss

}
```


No olvidemos importar este archivo Sass a nuestro archivo de manifiesto:

```
// app/assets/stylesheets/application.sass.scss

// All the previous code
@import "components/turbo_progress_bar";
```

Una buena forma de ver que logramos agregar estilos a la barra de progreso Turbo es agregar **temporalmente** `sleep 3` a nuestro controlador acciones para que la barra de progreso aparezca durante al menos 3 segundos:

```
# app/controllers/application_controller.rb

class ApplicationController < ActionController::Base
  # Add this line to see the progress bar long enough
  # and remove it when it has the expected styles
  before_action -> { sleep 3 }
end
```

¡Eso es todo! Nuestra barra de progreso tiene estilo y coincide con nuestro sistema de diseño. Ahora podemos eliminar el `sleep 3` fragmento de código.

Conclusión

Ese fue un capítulo fácil. ¡Literalmente no tuvimos que hacer nada para que Turbo Drive funcionara! ¡Obtuvimos importantes beneficios de rendimiento de forma gratuita sin tener que escribir una sola línea de código personalizado!

En el próximo capítulo aprenderemos a transformar páginas complejas en piezas sencillas utilizando Turbo Frames. ¡Nos vemos allí!

← anterior Siguiente →