Upgrade to Rails 4

A thorough guide for upgrading your Rails 3 application

Philip De Smedt

Upgrade to Rails 4

A thorough guide for upgrading your Rails 3 app

Philip De Smedt

This book is for sale at http://leanpub.com/upgradetorails4

This version was published on 2013-09-22

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Philip De Smedt by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I am reading Upgrade to Rails 4 #upgradetorails4

The suggested hashtag for this book is #upgradetorails4.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#upgradetorails4

Contents

Turbolinks

A significant amount of effort was spent on making Rails 4 a lot faster than previous versions. One of these improvements is Turbolinks. When a visitor clicks a link on your website, Turbolinks makes your application faster by using JavaScript to replace only the relevant content instead of doing a full page request. Your browser will not reinterpret all JavaScript and CSS, which should result in a significant speed boost. At the end of this chapter, we will have a quick look at the performance when Turbolinks is enabled versus the performance when Turbolinks is disabled.

How does it work?

What Turbolinks essentially does is listen to a click event on each link on your page. Turbolinks makes a JavaScript request every time such a click event happens. After the response comes in, Turbolinks will use JavaScript to update the relevant content on the current page. It uses the HTML5 `pushState` API to change the URL of the page, so it looks as if we are going to a new page. This technique is very similar to pjax3[1], which is a jQuery plugin that uses `ajax` and `pushState` to deliver a fast browsing experience with real permalinks, page titles and a working back button. The difference between `pjax` and Turbolinks is that with `pjax` it is necessary to specify which part of the page is to be replaced. Turbolinks simply replaces the entire body and title of your page.

Turbolinks is enabled by default in Rails 4. If you are upgrading an existing Rails app to Rails 4, you will have to change your existing front-end JavaScript. Since pages will change without doing a full page reload, you won't be able to rely on `DOMContentLoaded` or `jQuery.ready()` to trigger your JavaScript code.

Consider the following jQuery code, which listens to a submit button and submits a form when the button is clicked:

```
$(document).ready(function() {
  $('.submit-button').click(function() {
    $(this).parent('form').submit();
  });
});
```

Imagine this piece of JavaScript is loaded on the page `/forms` and our root page `/` contains a link to the `/forms` page. When we initially load the root page and click the link that takes us to `/forms`, Turbolinks will handle this page request using JavaScript. However, jQuery assumes we are still on

[1] https://github.com/defunkt/jquery-pjax

the root URL `/`, which means the JavaScript code above will not be loaded, since we're changing page URLs with HTML5 `pushState` and the actual URL is not loaded. The code above would work fine when we do an initial page load on the `/forms` URL, but after the first Turbolinks request has been processed, subsequent calls to `jQuery.ready()` won't be processed, since the `document.ready` event is not called.

Luckily, Turbolinks has a workaround for this, so your custom JavaScript will keep on working. Turbolinks gives you a range of events to deal with the lifecycle of the page:

- `page:fetch` - starting to fetch the target page
- `page:load` - fetched page is being retrieved fresh from the server
- `page:restore` - fetched page is being retrieved from the 10-slot client-side cache
- `page:change` - page has changed to the newly fetched version

Looks like we will need to trigger our code on a `page:load` event, as well as on the `jQuery.ready()` event. Change your code as follows:

```
var ready = function() {
  $('.edit-button').click(function() {
    $(this).parent('form').submit();
  });
};
$(document).ready(ready);
$(document).on('page:load', ready);
```

Whether or not we're using Turbolinks, we are now sure that our ready event will be called. If you have a lot of existing JavaScript that binds elements on `jQuery.ready()`, you can use the `jquery.turbolinks` gem[2] that will automatically trigger `jQuery.ready()` when Turbolinks triggers a `page:load` event.

While Turbolinks might be a nifty feature that speeds up your pages significantly, it is a good idea to keep an eye on the Turbolinks issues on Github[3].

Disable Turbolinks

If for some reason you want to disable Turbolinks on a link, you can opt out of it by using the data-no-turbolink option on an existing link tag.

[2] https://github.com/rails/turbolinks
[3] https://github.com/rails/turbolinks/issues

```
<a data-no-turbolink="true" href="/blog">Visit my Blog</a>
```

Additionally, you can opt out of Turbolinks completely by removing the gem `turbolinks` from your Gemfile and removing the Turbolinks directive (`//= require turbolinks`) from your `application.js` manifest file.

How much faster is Turbolinks?

Steve Klabnik has done a quick and dirty benchmark[4] on execution times of page loads with and without Turbolinks enabled. The following is the result of following a link 1000 times in a blank Rails app:

- 1000 pages without Turbolinks: 138.66 seconds
- 1000 pages with Turbolinks: 80.44 seconds

While there are a bunch of things missing that invalidate this as a scientific test (this was ran in development mode and there is no JS or any other big assets), it's still a good measure that Turbolinks significantly speeds up your Rails app. More benchmarks can be found on Steve Klabniks blog[5].

Using Turbolinks in Rails 3

Turbolinks is also compatible with Rails 3 through the `turbolinks` gem. Add the gem to your Gemfile (`gem 'turbolinks'`) and run the Bundle command to install it (`bundle install`). Next, you will have to add the Turbolinks directive to your `application.js` manifest file:

```
// This is a manifest file that'll be compiled into application.js,
// which will include all the files listed below.
//
// Any JavaScript/Coffee file within this directory,
// lib/assets/javascripts, vendor/assets/javascripts, or
// vendor/assets/javascripts of plugins, if any, can be referenced
// here using a relative path.
//
// It's not advisable to add code directly here, but if you do, it'll
// appear at the bottom of the the compiled file.
//
// WARNING: THE FIRST BLANK LINE MARKS THE END OF WHAT'S TO BE PROCESSED,
```

[4] https://github.com/steveklabnik/turbolinks_test

[5] http://blog.steveklabnik.com/posts/2012-09-27-seriously--numbers--use-them-

```
// ANY BLANK LINE SHOULD GO AFTER THE REQUIRES BELOW.
//
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

Turbolinks **does not** depend on jQuery, so you can still use it if you're not using jQuery itself. Turbolinks requires a modern browser that supports `pushState` and all related APIs (Safari 6.0+, IE10, Chrome 5.0+ or Firefox 4.0+). However, if Turbolinks is not supported, it will degrade gracefully and your app will still work as expected. If you're testing Turbolinks in development, make sure you use one of the supported browsers to get rid of bugs that may occur.