Custom Asset Tag Helpers

Shawn Toffel

March 19, 2015

1 Intro.

As the upcoming web component standards continue to grow I have noticed a trend where you will have a main HTML file and several component HTML files that are included into the main file. This is usually accomplished using a link tag in the head of the main file.

```
<link rel="import" href="component.html" />
```

However, when working with Rails (and many other frameworks), we don't want to just drop our files into the public directory and call it good. We want to be able to take advantage of the Rails Asset Pipeline.

2 Rails Asset Pipeline

The rails documentation has an excellent page describing the Asset Pipeline, but I'll describe some of the basics here. The Asset Pipeline allows us to perform tasks with assets. These tasks can include compression, concatenation, minification, and any pre or post processing you might need.

2.1 Pre-Processor

Sass, less, ERB, and CoffeeScript are all examples of pre-processors. Another important pre-processor is the directive processor. The Directive Processor is in charge of scanning manifest files and parsing and evaluating any directive comments it finds. A directive comment consists of a comment prefix, followed by an "=", then the directive name, then any arguments.

```
//= require foo
```

There are many directive names provided by Sprockets.

2.2 Post-Processor

Once we know where all the files live, we can begin post processing. Post processing is where all of the files found by the Directive Processor can be concatenated into a single file.

2.3 Compressor

Next we do any compression on the files such as removing whitespace.

2.4 Fingerprinting

This file is eventually finger printed and exported to the public assets directory. Fingerprinting is the technique of adding a hash which depends on the contents of the file to the filename itself. This way if the file contents changes the filename also changes. This is especially valuable in preventing the use of old cached files.

3 Asset Pipeline and Web Components

Now that we have a basic understanding of the Asset Pipeline the value it would provide when including web components should be obvious. We can include all of our components in a single manifest file using directives. We can then do any necessary processing with our JavaScript and styles, and recursively inline imports until we are down to a single file. We can compress the file and export a single fingerprinted HTML file to the public assets directory. But, if our filename will be different every time the content changes, how will we import the file? We will use a custom tag.

4 HTML import tag

In order to import our web components let's take a moment to think about what we're trying to accomplish. We would like to have a tag in our application layout file that imports all of our web components. We would like it to behave like the JavaScript and stylesheet import tags. That is, we provide a manifest file containing comment directives pointing to each of the HTML files we want to include. We would like all of the components to be concatenated and compressed into a single file to speed up load time. Additionally, we would like this file to be fingerprinted. So where do we begin? A good approach might be to separate our logic into a gem, which we will include in our Rails project.

4.1 Gem Structure

```
lib
├── html_import_tag
│   ├── helpers
│   │   └── asset_tag_helper.rb
│   ├── processors
│   │   └── directive_processor.rb
│   ├── compressors
│   │   └── html_compressor.rb
│   └── railtie.rb
└── html_import_tag.rb
```

4.2 Railtie

Railtie provides the hooks we will need to add our pre, post, and compression processors. We can be extend Railtie in our gem via the Rails::Railtie class. This will add our processors to the Asset Pipeline.

Listing 1: Railtie Example

```
module HtmlImportTag
  class Railtie < Rails::Railtie

    initializer :add_preprocessors do |app|
    end

    initializer :add_postprocessors do |app|
    end

    initializer :add_compressors do |app|
    end
  end
end
```

4.3 html_import_tag

In order to add our custom tag, we simply extend the ActionView AssetTagHelper class provided by Rails. The sources passed to the tag will be provided to us here. We then find our html files and create the link tag with the options to import html.

Listing 2: AssetTagHelper Example

```
module ActionView
  module Helpers
    module AssetTagHelper
      def html_import_tag(*sources)
        options = sources.extract_options!.stringify_keys
        path_options = options.extract!('protocol', 'extname').symbolize_keys

        sources.uniq.map do |source|
          tag_options = {
            rel: "import",
            href: path_to_asset(source, {:type => :html, extname: ".html"}.merge!
          }.merge!(options)

          tag(:link, tag_options)

        end.join("\n").html_safe
      end
    end
  end
end
```

5 Demo

I have created a very basic gem to demonstrate how the html_import_tag might be implemented. I have included the gem into a Gemfile in a Rails project. I placed a demo.html file in a components directory under app/assets. I then import the demo.html file into the main application layout with the following line:

```
<%= html_import_tag "demo" %>
```

This results in an import of the fingerprinted demo.html file. If we view the source we can see that the component was successfully imported!