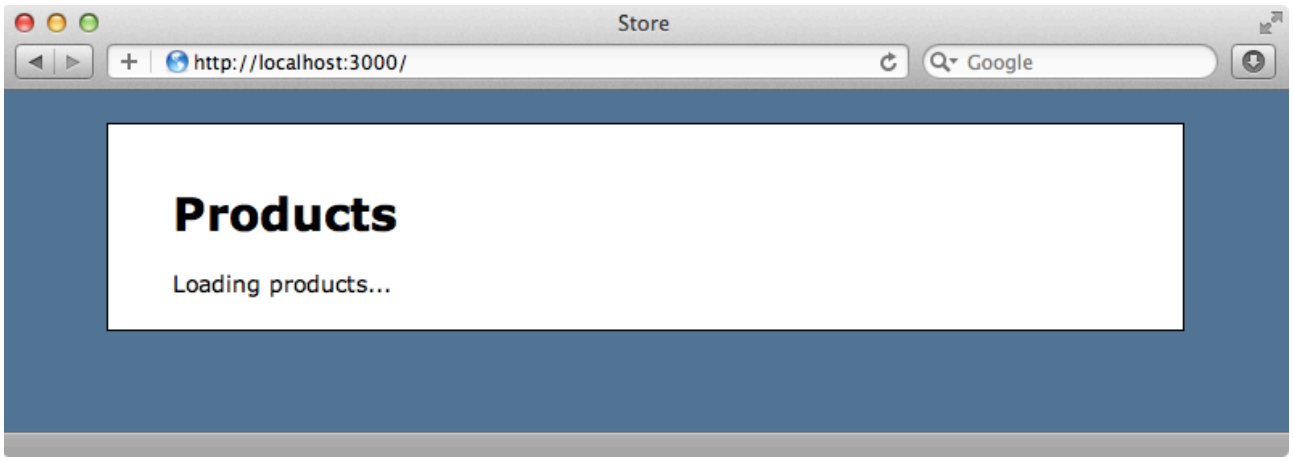
Episode 324

Passing Data to JavaScript

When JavaScript plays a large part in a Rails application it often becomes necessary to pass data from the app on the server to JavaScript to be used by the client. In this episode we'll explore some techniques for doing just that. Below is a simple page from a simple Rails application. The idea here is that we want JavaScript to take over and handle fetching and displaying the products. To do that we'll need to pass some information from our app to the JavaScript that runs on the client.



The view template for this page is simple as the page suggests.

/app/views/products/index.html.erb

```
<h1>Products</h1>

<div id="products">
  Loading products...
</div>
```

Let's say that the JavaScript in the app needs to know the URL to call to fetch the list of products. We don't want to hard-code this in the JavaScript code and so we'll generate it dynamically in our Rails app. One way we can do this is to use `javascript_tag` in the view.

/app/views/products/index.html.erb

```
<%= javascript_tag do %>
  window.productsURL = '<%= j products_url %>';
<% end %>
```

embedding in JavaScript. Now we can use this variable in any of the JavaScript or CoffeeScript files that this page references.

/app/assets/javascripts/products.js.coffee

```
jQuery ->
  alert productsURL
```

If we want to preload the page with an initial set of products instead of having fetch them in a separate request we could fetch, say, the first ten products as JSON and have the JavaScript display them straight away. We could do this by writing something like this:

/app/views/products/index.html.erb

```
<%= javascript_tag do %>
  window.productsURL = '<%= j products_url %>';
  window.products = <%= raw Product.limit(10).to_json %>
<% end %>
```

Rails will automatically try to HTML-escape the list of products and so we have to call `raw` on the JSON that's returned.

This approach can become awkward fairly quickly and using data attributes on HTML can often be a better alternative. We can pass in the products URL to one like this:

/app/views/products/index.html.erb

```
<h1>Products</h1>

<div id="products" data-url="<%= products_url %>">
  Loading products...
</div>
```

We can easily fetch this information with some jQuery code.

/app/assets/javascripts/products.js.coffee

```
jQuery ->
  alert $('#products').data('url')
```

This technique has the same effect but feels a little cleaner, apart from having to nest Erb tags within HTML attributes. Using `content_tag` is generally a better approach for inserting dynamic data into an HTML tag.

/app/views/products/index.html.erb

```
<h1>Products</h1>

<%= content_tag "div", id: "products", data: {url: products_url}
do %>
  Loading products...
<% end %>
```

Since Rails 3.1 we've been able to use a `data` hash to define data attributes which makes this approach even better. If we pass Ruby objects into this hash `to_json` is called so that the object or objects passed in are automatically converted to their JSON representation.

/app/views/products/index.html.erb

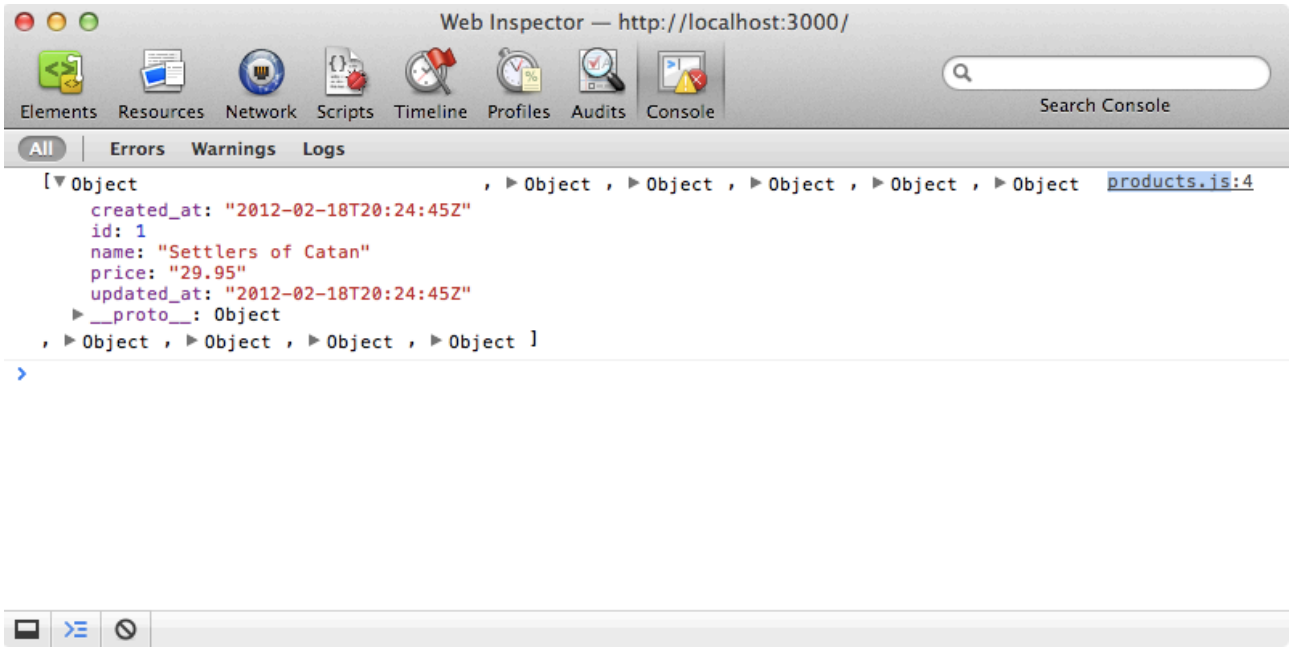
```
<h1>Products</h1>

<%= content_tag "div", id: "products", data: {url:
Product.limit(10) } do %>
  Loading products...
<% end %>
```

When we fetch this data it will be automatically parsed into a JavaScript object.

/app/assets/javascripts/products.js.coffee

```
jQuery ->
  console.log $('#products').data('url')
```

If we reload the page now we'll see the products listed in the console.



Gon

If we have a lot of data to pass to the JavaScript this technique can still become fairly cumbersome. Fortunately there's one more solution we can use: the Gon gem[1]. This allows us to set variables in our controllers and then access them from JavaScript. Gon is installed in the usual way, by adding it to the gemfile and running `bundle`.

`/Gemfile`

[1] https://github.com/gazay/gon

```
source 'https://rubygems.org'

gem 'rails', '3.2.1'
gem 'sqlite3'

# Gems used only for assets and not required
# in production environments by default.
group :assets do
  gem 'sass-rails',   '~> 3.2.3'
  gem 'coffee-rails', '~> 3.2.1'

  # See https://github.com/sstephenson/execjs#readme for more
supported runtimes
  # gem 'therubyracer'

  gem 'uglifier', '>= 1.0.3'
end

gem 'jquery-rails'
gem 'gon'
```

Next we'll need to update our application's layout file by adding `include_gon` somewhere inside the `head` section.

/app/views/layouts/application.html.erb

```
<head>
  <title>Store</title>
  <%= include_gon %>
  <%= stylesheet_link_tag    "application", media: "all" %>
  <%= javascript_include_tag "application" %>
  <%= csrf_meta_tag %>
</head>
```

If we do this before we load any other JavaScript files we'll be able to access the variables we've set through Gon without waiting for the entire DOM to load.

We can now set variables on a gon object in a controller action.

/app/controllers/products_controller.rb

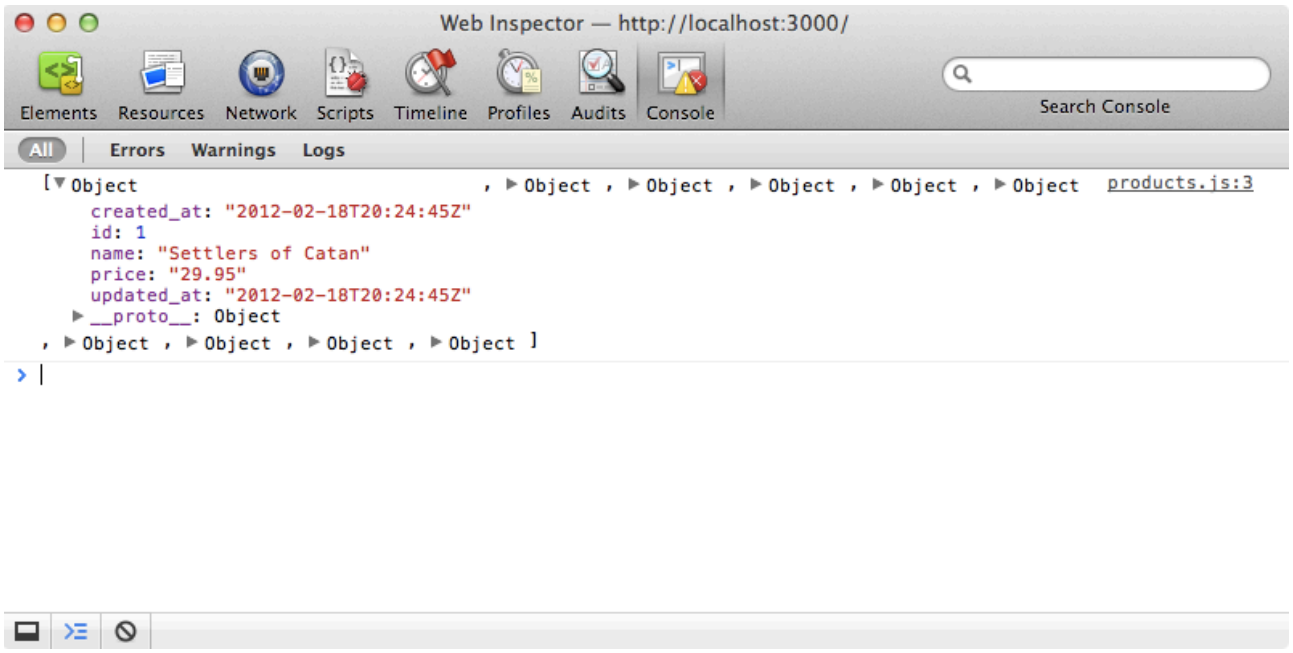
```
class ProductsController < ApplicationController
  def index
    gon.products = Product.limit(10)
  end
end
```

This list of products will now automatically be converted to JSON and be accessible by our application's JavaScript. We can access this list of products like this:

/app/assets/javascripts/products.js.coffee

```
console.log gon.products
```

Note that we don't have to wait for the DOM to load before we access this data. If we reload the page now we'll see the list of products in the console again.



If we view the page's source we'll see how this all works.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Store</title>
      <script>window.gon =
{};gon.products=[{"created_at":"2012-02-18T20:24:45Z","id":1,"name":"Settlers
of Catan","price":"29.95","updated_at":"2012-02-18T20:24:45Z"},
{"created_at":"2012-02-18T20:24:45Z","id":2,"name":"DVD
Player","price":"79.98999999999999","updated_at":"2012-02-18T20:24:45Z"},
{"created_at":"2012-02-18T20:24:45Z","id":3,"name":"Red
Shirt","price":"12.49","updated_at":"2012-02-18T20:24:45Z"}];</script>
    // Other products omitted.
    <link href="/assets/application.css?body=1" media="all" rel="stylesheet"
type="text/css" />
<link href="/assets/products.css?body=1" media="all" rel="stylesheet"
type="text/css" />
    <script src="/assets/jquery.js?body=1" type="text/javascript"></script>
<script src="/assets/jquery_ujs.js?body=1" type="text/javascript"></script>
<script src="/assets/products.js?body=1" type="text/javascript"></script>
<script src="/assets/application.js?body=1" type="text/javascript"></script>
    <meta content="authenticity_token" name="csrf-param" />
<meta content="KktOQAWKvaRho+IdWOiaBud0W7Vuv31rdn0LF38hBag=" name="csrf-
token" />
  </head>
  <body>
    <!-- Body omitted -->
  </body>
</html>
```

Gon simply creates a `script` tag then fills a gon variable with the data that we set in the controller. If we want to customize the JSON that's returned to the client Gon has support for both RABL and Jbuilder templates, both of which have been covered in recent episodes. To customize the data returned by the list of products we can create an `index.json.rabl` file and define the attributes we want to be returned there.

/app/views/products/index.json.rabl

```
collection Product.limit(10)

attributes :id, :name, :price
```

We can use `gon.rabl` to tell Gon to use this template.

/app/controllers/products_controller.rb

```
class ProductsController < ApplicationController
  def index
    gon.rabl "app/views/products/index.json.rabl", as: "products"
  end
end
```

This will store the response from the template in a `products` variable. When we reload the page now we'll see the same list of products, but their attributes will have been defined by the RABL template.

There's a small potential trap when using Gon. If we don't define any gon variables in a controller action and we try to call gon in the JavaScript we'll get an error as the gon object won't have been set. It's always best to ensure that this object exists before trying to call anything against it.

`/app/assets/javascripts/products.js.coffee`

```
console.log gon.products if gon
```

There's more information about Gon in its documentation[2] which is well worth reading if you're thinking of using it in your Rails applications.

[2] https://github.com/gazay/gon/blob/master/README.md