Extracted from:

Rails Recipes

Rails 3 Edition

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

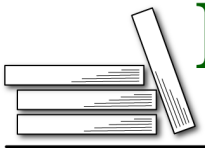
Rails Recipes

Rails 3 Edition

Chad Fowler

Edited by John Osborn

The Facets of Ruby Series



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

John Osborn (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-93435-677-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2012

Problem

Before the introduction of HTML5, if you were creating a rich, JavaScript-driven HTML user interface and wanted to store data in the browser for use by elements on the page, you were on your own. You could certainly store data in a JavaScript structure and access it by element id. Or you could create a custom XHTML namespace. But any solution required some manual work, and anyone who followed you on the project had to learn your way of doing things.

HTML5 introduces a standard solution to this problem with its new data-* attributes. How can we best use these from Rails applications?

Solution

You can create tags that store local data with the new Rails 3.1+ :data option. This gives us a convenient mechanism for generating HTML5 tags with data-* attributes, a feature available with HTML 5. In this recipe, we'll generate a page with data-* and then show how to read those options using CoffeeScript.

The first step is to generate an HTML template containing data attributes. As an example, we'll create a simple contact management app. We want to generate a sparse list of contact names but show more detailed information about the contact when a user hovers over the name with a mouse. We'll start with the default scaffolded index() action and a basic Contact model containing fields for name, city, state, and country.

Here's code to list the Contact records like this:

rr2/html5-data/app/views/contacts/index.html.erb

```
<h1>Listing contacts</h1>
<ul>
<% @contacts.each do |contact| %>
  <%= content_tag_for(:li,
                      contact,
                      :data => {
                        :city => contact.city,
                        :name => contact.name,
                        :country => contact.country}) do %>
    <%= contact.name %> from <%= contact.city %>
  <% end %>
<% end %>
</ul>

<br />

<%= link_to 'New Contact', new_contact_path %>
<div id='tip' style='display:none'>
```

```
</div>
```

There isn't much new here. Except for one feature, it's all the same view code Rails developers have been writing for years. The exception is that when we generate the <li> elements, we use content_tag_for()'s :data option to specify a Hash of data elements we want embedded in the generated element. So, though our <li> element shows only the contact's name, the element itself has more information embedded. An example contact list's source might look like this:

```
<li class="contact" data-city="New Orleans"
  data-country="USA" data-name="Chad Fowler" id="contact_1">
    Chad Fowler from New Orleans
</li>
<li class="contact" data-city="Oak Park" data-country="USA"
  data-name="Donald Shimoda" id="contact_2">
    Donald Shimoda from Oak Park
</li>
<li class="contact" data-city="Tokyo" data-country="Japan"
  data-name="Toru Okada" id="contact_3">
    Toru Okada from Tokyo
</li>
```

Now that we've successfully embedded data into the list elements, we'll write some CoffeeScript to display it. We have prepared an empty, hidden at the bottom of our index() view with the id set to <tip>. Our CoffeeScript code will fill this element with additional contact information when a user hovers over the corresponding list element. Here's our CoffeeScript file, which we've put in app/assets/javascripts/contacts.js.coffee:

rr2/html5-data/app/assets/javascripts/contacts.js.coffee

```
$ ->
  $('.contact').bind 'mouseenter', (event) =>
    contact = event.target
    $('#tip').html text_summary_for(contact)
    $('#tip').show()
    $('.contact').bind 'mouseleave', (event) =>
      $('#tip').hide()
```

Here we're using jQuery's ability to run code when the document is ready. The code binds the <mouseenter> event to set the tip element's HTML to a text summary of the contact record and show it. Then on a <mouseleave>, we re-hide the tip. The text_summary_for() function simply concatenates a string of text to be rendered into the tip element:

rr2/html5-data/app/assets/javascripts/contacts.js.coffee

```
text_summary_for = (contact) =>
  contact.dataset['name'] +
    " lives in " +
```

Automatic CoffeeScript Compilation

As of Rails 3.1, Rails will automatically compile CoffeeScript files into JavaScript in development mode. Simply edit CoffeeScript files in your app/assets/javascripts directory, and Rails will compile them into JavaScript and serve them as requests are made.

```
contact.dataset['city'] +
" in " +
contact.dataset['country']
```

That's all there is to it! As you can see, HTML5 data attributes make in-element data storage simple, and the addition of the :data option in Rails 3.1 makes it even cleaner.

Also See

- For more information on new features of HTML5, see Brian Hogan's *HTML5 and CSS3: Develop with Tomorrow's Standards Today* [Hog10].
- To learn more about CoffeeScript, check out the CoffeeScript website at http://jashkenas.github.com/coffee-script/. Another alternative, Trevor Burnham's *CoffeeScript: Accelerated JavaScript Development* [Bur11], is an excellent guide to the language.