Extracted from:

Cucumber Recipes

Automate Anything with BDD Tools and Techniques

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Cucumber Recipes

Automate Anything with BDD Tools and Techniques

Ian Dees,
Matt Wynne,
and Aslak Hellesøy

Edited by Jacquelyn Carter

Printed in the United States of America.
ISBN-13: 978-1-937785-01-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—August 15, 2012

Recipe 29

Parse HTML Tables

Problem

You're testing a web page containing tabular data (or any repeating data, really), and you need to compare the contents to a table in your Cucumber scenario.

Ingredients

- Capybara[7] for testing web applications
- Capybara's arsenal of *finders*[8] for traversing patterns in HTML
- XPath[9] for describing the locations of objects on the page

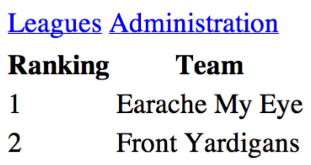
Solution

Capybara is a Ruby web testing library. It provides a simple API for visiting web pages and parsing the results. Behind the scenes, Capybara will either launch a real browser (for non-Ruby web apps) or just call directly into your server code (for Ruby apps built on Rails, Sinatra, or any other Rack framework).

In this recipe, you'll serve a simple static site using the Sinatra framework, then use Capybara to find the right table on the page and extract the contents.

Imagine you have a web page containing team rankings for a lawn darts league, something like Figure 27, *A Tale of Two Tables*, on page 4. You'd like to match the results against the ones you expect your algorithm to return. Any web testing library can scrape a bunch of raw HTML off the page and hand it to you for processing. But then it'd be up to you to use a DOM parsing library to loop through that HTML and extract the team names.

7. http://jnicklas.github.com/capybara
8. http://rubydoc.info/github/jnicklas/capybara/master/Capybara/Node/Finders:find
9. http://www.w3.org/TR/xpath



Leagues Administration

Ranking	Team
1	Earache My Eye
2	Front Yardigans

Figure 27—A Tale of Two Tables

Capybara's finders can spare you that agony. Let's see how.

The Application

For this recipe, we'll serve the data as a static HTML file. Put the following markup in public/lawn_darts.html:

html_tables/public/lawn_darts.html

```
<!doctype html>
<title>Lawn Darts</title>
<table>
  <tr>
    <td><a href="#">Leagues</a></td>
    <td><a href="#">Administration</a></td>
  </tr>
</table>
<table>
  <tr>
    <th>Ranking</th>
    <th>Team</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Earache My Eye</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Front Yardigans</td>
  </tr>
</table>
```

You could use Capybara with this file right now by connecting it to the Selenium browser-based framework. But let's wrap a trivial Ruby application around it instead, so that we can test through the much faster Rack interface.

First, install the Rack-based Sinatra web framework:

```
$ gem install sinatra
```

Now, create a file called lawn_darts_app.rb with the following contents:

html_tables/lawn_darts_app.rb
```
require 'sinatra/base'

class LawnDartsApp < Sinatra::Base
end
```

Now that we have a Ruby web interface, we can drive this static site from Cucumber.

Test Setup

Here's a Cucumber scenario that will check the contents of the table containing our teams. This code goes in features/league.feature:

html_tables/features/league.feature
```
Feature: Lawn darts league

  Scenario: View teams
    When I view the league page
    Then I should see the following teams:
      | Ranking | Team            |
      |       1 | Earache My Eye  |
      |       2 | Front Yardigans |
```

Because this test uses Capybara, now's a good time to install it:

```
$ gem install capybara
```

You'll need to connect Cucumber to Capybara by putting the following code in features/support/env.rb:

html_tables/features/support/env.rb
```
require 'capybara/cucumber'
require './lawn_darts_app'

Capybara.app = LawnDartsApp
```

Now that Cucumber can drive the site, it's time to add step definitions to retrieve and process the HTML.

Scraping HTML

In the first step definition, Capybara needs visit the league page. Create a file called features/step_definitions/league_steps.rb with the following contents:

html_tables/features/step_definitions/league_steps.rb
```
When /^I view the league page$/ do
  visit '/lawn_darts.html'
end
```

Once we've hit the page, Capybara has the contents ready for us to slice and dice. We'll do that in the Then step:

html_tables/features/step_definitions/league_steps.rb

```
Line 1 Then /^I should see the following teams:$/ do |expected|
     2   rows   = find('table:nth-of-type(2)').all('tr')
     3   actual = rows.map { |r| r.all('th,td').map { |c| c.text } }
     4   expected.diff! actual
     5 end
```

Let’s walk through that step line by line. At line 2, Capybara’s find() method retrieves the table element that contains the teams. This is actually the second table on the page (the first one contains navigation links), so we need to use XPath’s nth-of-type modifier.

Once we have the table, we call the all() method on it to retrieve all the <tr> elements on the page.

Each <tr> element may contain multiple cells in the form of <th> or <td> elements. On line 3, we loop through each row’s cells and retrieve the contents.

Finally, on line 4, we use Cucumber’s diff!() method to compare the actual table against the expected value, and report a test failure if there are any differences.

As we’ve seen, comparing HTML tables is just a matter of combining two simple pieces. A web scraping library like Capybara does the initial work of converting the HTML into a standard Ruby array. Cucumber takes over from there and compares the native Ruby data to what’s in the scenario.

Further Exploration

In this recipe, we tested a Ruby-based web app through a Ruby-specific test interface. For non-Ruby apps, you can use Capybara with a web browser through the Selenium layer; see Recipe 3, *Run Slow Setup/Teardown Code With Global Hooks*, on page ? for an example that uses Selenium.

For more information about comparing tables in Cucumber, see Recipe 1, *Compare and Transform Tables of Data*, on page ?.