Screen Scraping With Ruby

Paul Stadig

Why?

- Not all websites provide an API
- The most interesting data lies in the “deep web”
- Fun mash-ups await you

Case studies

- rubystuff.com
- code.whytheluckystiff.net
- reformedchurches.info
- Ideas
  - Access your router HTTP interfaces
  - Consolidate information from your bank websites

How?

POOR
(Plain Old Open-URI and RegExps)

- Pros
  - Simple
  - No extra libraries
- Cons
  - Very fragile/limited in extraction
  - You have to RYO browser (navigation, history, caching, cookies, authentication, forms, etc).
  - POST?

The Code

- This is string based extraction
- A better way is using a library that “understands” HTML (i.e. tree based extraction)

Normalization and REXML?

Don't do it!

POOH
(Plain Old Open-URI and Hpricot)

- Pros
  - Fast
  - Query using CSS selectors or XPath
  - Easy iteration over elements
  - Handles poorly constructed HTML pages
- Cons
  - Still have to RYO browser

The Code

Hpricot Up Close

```
doc = open("/tmp/afile") {|f| Hpricot(f) }
```

Hpricot Up Close

1. doc.search("//tr[@class='datarow']")
2. doc.search("tr.datarow")
3. doc.search("#submenu")
4. (doc/"#submenu”)
5. (doc/:tr).select{|tr| tr['class'] == 'datarow'}

Hpricot Up Close

1. doc.at("/html/body/table[2]/tr/td[2]")
2. doc.at("#submenu")
3. doc.at("/html/body/table[2]/tr/td")
4. ((doc/:html/:body/:table)[1]/:tr/:td).first
5. (doc/"#sitesearch/table/tr/td").first

Hpricot Up Close

1. doc.at("//a")['href']
2. doc.at("//a").attributes['href']

Hpricot Up Close

1. ((doc/:html/:body/:table)[1]/:tr/:td).first.xpath
   => "//form[@id='sitesearch']/table/tr[1]/td[1]"
2. ((doc/:html/:body/:table)[1]/:tr/:td).first.css_path
   => "#sitesearch > table > tr:nth(0) > td:nth(0)"

WWW::Mechanize

- Pros
  - Handles navigation, history, cookies, authentication, forms, etc.
  - Works with an HTTP proxy (this means sophisticated caching)
  - Automatically gives you access to an Hpricot document
- Cons
  - No JavaScript support

The Code

scRUBYt!

- Pros
  - Very simple to scrape simple data
- Cons
  - Still somewhat immature

The Code

WATIR & FireWatir

- Pros
  - Drives an actual browser
- Cons
  - Win32 only

scrAPI

- ?

Tips

- Use FireBug to find the XPath to an element
- However, beware of the tbody tag, missing p, and other problems
- You should be nice
  - Pay attention to robots.txt
  - Use a caching proxy (especially when you are testing your extraction code)

Resources

- http://www.rubyrailways.com/data-extraction-for-web-20-screen-scraping-in-rubyrails
- http://www.rubyrailways.com/data-extraction-for-web-20-screen-scraping-in-rubyrails-episode1
- http://scottstuff.net/blog/articles/2003/11/01/html-screen-scraping-in-ruby
- http://www.igvita.com/blog/2007/02/04/ruby-screen-scraper-in-60-seconds/