Encuentro de Tecnologías Libres, Mendoza 2016

Scrapping de Sitios Web con Ruby

Copyleft 2016 - Sergio A. Alonso (pancutan) http://about.me/elbunker

Razones para hacer scrapping

- Minería de datos
- Algoritmos para Data Science
- Para aprender a automatizar y controlar cosas
- Para armar paneles de control
- Para casos en los que no hay apis, o nos restringen incluso si hemos pagado por ello: de fotos
- Comparar cosas: nombres de dominio (poderopedia, nicathon)
- Bases para automatizar secuencias: encontrar cosas y proceder en consecuencia, ya sea con Ruby o ya sea lanzando comandos del sistema operativo
- Porque hay mucho laburo freelance

Que aporta Ruby a la disciplina

- Naturalidad en las expresiones (Ruby es *"natural"*, no *"fácil"*)
- Normas no restrictivas para facilitar la distinción de piezas de codigo tanto a GIT como a nuestros compañeros de trabajo
- Muchas librerias para hacer lo mismo
- Muchas ideas cruzadas y mejoradas junto a otros lenguajes (Beatiful Soup de Python, Markdown de Ruby y mejorado en R, Watin de .NET y Watir, etc)

Que aportan los debuggers de Ruby

- Colores, autocompletado, breakpoints en caliente, watchpoints
- Exploración (en serio) de objetos y métodos
- Logging de lo creado
- Debugger en el momento que las cosas se rompen: expresiones como try-again, edit -c (llamar al editor), cd-cause, etc
- Debug remoto
- Debug en sus propios frameworks web
- Desagregado de largos strings en piezas comprensibles visualmente

El siguiente tutorial hace uso intenso de pry. Se sugiere para aprovecharlo, instalar mediante gem las librerías pry, pry-debug, y por supuesto open-uri, nokogiri y mechanize

Scrapping en su forma primitiva: open-uri

© Copyright Monday, Dec 19, 2016, 12:46 PM by COMPANYNAME

```
$ irb
```

o mejor

```
$ pry

require 'open-uri'
open("http://www.ruby-lang.org/") do |f|
  f.each_line {|line| p line}
end
```

Esto conviene ponerlo en un archivo *ejemplo-open-uri.rb* Se puede ejecutar con

```
ruby ejemplo-open-uri.rb
```

Usemos toda la batería de pry

```
require 'open-uri'

require 'pry'
require 'pry-byebug'

# Use los términos continue, next, break o help para continuar
binding.pry
open("http://www.ruby-lang.org/") do |f|
    f.each_line {|line| p line}
end
```

Jugando con pry, nos creamos una mejor versión, ejemplo-open-uri-2.rb

```
require 'open-uri'

require 'pry'
require 'pry-byebug'

# Use los términos continue, next, break o help para continuar
binding.pry
a = Array.new
open("http://www.ruby-lang.org/") do |f|
  f.each_line do |line|
    a.push line
  end
end

puts "Programa terminado"
```


© Copyright Monday, Dec 19, 2016, 12:46 PM by COMPANYNAME

Si jugamos con pry, podriamos establecer un breakpoint en la ultima linea, e identificar elementos del array: a[0], a[3], etc

_ Todo cambia _ (esencia de Heráclito):

El problema con este enfoque, es que es muy laborioso de mantener. El menor cambio en la pagina requerirá que tengamos que buscar patrones una y otra vez para detectar que tanto se han movido los elementos que recuperamos diariamente.

Nokogiri

Nokogiri nos permite seleccionar mediante Xpath o mediante CSS elementos específicos de la pagina, navegando directamente el árbol DOM. En su forma mas simple, Nokogiri permite ir navegando lo obtenido directamente con su propia abstracción de objetos, tal y cual lo dicta la www3. Es decir, no necesitamos estrictamente a Xpath o a CSS. Pero estaremos aplazando el problema anterior de los elementos que pueden repentinamente cambiar de lugar, o reinventando la rueda al tratar de ir preguntando con if si ciertos elementos se encuentran o no, a fin de usarlos de ancla.

Xpath

Si intentamos seleccionar elementos mediante esta norma, nos podemos ayudar mediante alguna funcion del navegador, o algún plugin como Xpath Helper, presente en https://chrome.google.com/webstore/detail/xpath-helper/hgimnogjllphhhkhlmebbmlgjoejdpjl?hl=es

Xpath Helper se activa con Ctrl + Shift + X, y luego se escanea con el Shift apretado sobre la zona que queremos escrapear, por ejemplo las Efemerides que se publican todos los días en https://es.wikipedia.org/wiki/Wikipedia:Portada

Mediante el plugin mencionado, se puede obtener esta cadena:

```
/html[@class='client-js ve-not-available']/body[@class='mediawiki ltr sitedir-ltr mw-
hide-empty-elt ns-4 ns-subject page-Wikipedia_Portada rootpage-Wikipedia_Portada skin-
vector action-view']/div[@id='content']/div[@id='bodyContent']/div[@id='mw-content-
text']/table[3]/tbody/tr/td[@class='MainPageBG'][2]/table/tbody/tr[4]/td[@id='wp-
port']/ul/li[1]
```

Esta cadena es solo una ayuda. Es posible que haya que limpiarla. Yo le saco el //html del principio y algunos tbody. Lo mas cómodo es pararse con el debugger en alguna linea despues de

```
html_data = open('https://es.wikipedia.org/wiki/Wikipedia:Portada').read
nokogiri_object = Nokogiri::HTML(html_data)
```

Aqui podemos hacer nuestras pruebas. Por ejemplo el código anterior levemente adaptado a Xpath se vería de la siguiente manera:

```
//body[@class='mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-4 ns-subject page-
Wikipedia_Portada
rootpage-Wikipedia_Portada skin-vector
```


© Copyright Monday, Dec 19, 2016, 12:46 PM by COMPANYNAME

```
action-view']/div[@id='content']/div[@id='bodyContent']/div[@id='mw-content-
text']/table[3]/tr/td[@class='MainPageBG'][2]/table/tr[4]/td[@id='wp-port']/ul/li[1]")
```

Para este momento, el fuente de nuestro programa de ejemplo, con la inclusión de Pry Debugger, se estaría viendo así:

```
require 'nokogiri'
require 'open-uri'

require 'pry'
require 'pry-byebug'

# Use los términos continue, next, break o help para continuar
binding.pry

html_data = open('https://es.wikipedia.org/wiki/Wikipedia:Portada').read
nokogiri_object = Nokogiri::HTML(html_data)
elements = nokogiri_object.xpath("//body[@class='mediawiki ltr sitedir-ltr mw-hide-empty-
elt ns-4 ns-subject page-Wikipedia_Portada rootpage-Wikipedia_Portada skin-vector action-
view']/div[@id='content']/div[@id='bodyContent']/div[@id='mw-content-
text']/table[3]/tr/td[@class='MainPageBG'][2]/table/tr[4]/td[@id='wp-port']/ul/li").map
{|v| v.text}

elements.each do |e|
  puts e
end

puts "Programa terminado"
```

Como ejercicio recomendado, tal que muestre la forma en que se puede seguir aislando la información que nos interesa, aprovechamos que Nokogiri implementa Enumerable, y le agregamos .to_a al final para tratarlo como un cómodo array. Con esto tenemos una vista DOM. Y podremos usar las facilidades de Ruby, por ejemplo .each do |una variable| o algo mas sintetico como .map

La manera de realizar este ejercio es arrancar el programa. Pry detendrá la ejecución al principio. Establecemos un breakpoint hacia la linea 11 (*break 11*), dejamos que continúe (*continue*) el programa, y en ese punto probamos

```
nokogiri_object.xpath("//body[@class='mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-4
ns-subject page-Wikipedia_Portada rootpage-Wikipedia_Portada skin-vector action-
view']/div[@id='content']/div[@id='bodyContent']/div[@id='mw-content-
text']/table[3]/tr/td[@class='MainPageBG'][2]/table/tr[4]/td[@id='wp-port']/ul/li")
```

Obtendremos a cambio un confuso string con esta forma:

```
=> [#<Nokogiri::XML::Element:0x185b3fc name="li" children=[#
<Nokogiri::XML::Element:0x175e328 name="b" children=[#<Nokogiri::XML::Element:0x175e044
name="a" attributes=[#<Nokogiri::XML::Attr:0x1759f80 name="href" value="/wiki/1810">, #
<Nokogiri::XML::Attr:0x1759f6c name="title" value="1810">] children=[#<Nokogir
(etc etc)
```


© Copyright Monday, Dec 19, 2016, 12:46 PM by COMPANYNAME

Sin embargo, si acompañamos la expresión anterior con .to_a

```
nokogiri_object.xpath("//body[@class='mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-4
ns-subject page-Wikipedia_Portada rootpage-Wikipedia_Portada skin-vector action-
view']/div[@id='content']/div[@id='bodyContent']/div[@id='mw-content-
text']/table[3]/tr/td[@class='MainPageBG'][2]/table/tr[4]/td[@id='wp-port']/ul/li").to_a
```

Obtendremos en cambio una vista mas parecida al DOM tal y cual como lo dicta la www3

```
=> [#(Element:0x185b3fc {
   name = "li",
   children = [
     #(Element:0x175e328 {
       name = "b",
       children = [
         #(Element:0x175e044 {
           name = "a",
           attributes = [ #(Attr:0x1759f80 { name = "href", value = "/wiki/1810" }), #
(Attr:0x1759f6c { name = "title", value = "1810" })],
           children = [ #(Text "1810")]
           })]
       }),
       (etc etc)
```

Estos objetos son muy fáciles de recorrer con Ruby. Puede aislarse uno o todos los nodos que se desee en forma muy precisa. Por ejemplo si queremos traer la primera de las efemérides:

```
[15] pry(main):1> nokogiri_object.xpath("//body[@class='mediawiki ltr sitedir-ltr mw-
hide-empty-elt ns-4 ns-subject page-Wikipedia_Portada rootpage-Wikipedia_Portada skin-
vector action-view']/div[@id='content']/div[@id='bodyContent']/div[@id='mw-content-
text']/table[3]/tr/td[@class='MainPageBG'][2]/table/tr[4]/td[@id='wp-port']/ul/li").map
{|m| m.text}[0]

=> "1810.– Nace Theodor Schwann, fisiólogo y botánico alemán, descubridor de la pepsina."
```

Selectores CSS

Este método es el mismo del anterior, con la salvedad que aprovechamos las caracteristicas de Firefox en su inspector de código. En este caso nos paramos sobre cualquier palabra de la primera efeméride, la inspeccionamos, y en la linea hacia donde nos lleva Firefox en el inspector de código, le hacemos Botón derecho → Copiar → Selector CSS

En el portapapeles nos queda algo así

```
#mw-content-text > table:nth-child(3) > tbody:nth-child(1) >
tr:nth-child(1) > td:nth-child(3) > table:nth-child(1) >
tbody:nth-child(1) > tr:nth-child(4) > td:nth-child(1) > ul:nth-child(2)
> li:nth-child(1)
```


© Copyright Monday, Dec 19, 2016, 12:46 PM by COMPANYNAME

Le limpiamos los elementos tbody tal como hiciéramos con xpath, de modo que nos quede así:

```
#mw-content-text > table:nth-child(3) > tr:nth-child(1) >
td:nth-child(3) > table:nth-child(1) > tr:nth-child(4) > td:nth-child(1)
> ul:nth-child(2) > li:nth-child(1)
```

Ahora podemos realizar nuestra búsqueda mediante css. La línea anterior nokogiri_object.xpath(" .... podemos cambiarla por esta versión:

```
nokogiri_object.css("#mw-content-text > table:nth-child(3) >
tr:nth-child(1) > td:nth-child(3) > table:nth-child(1) > tr:nth-child(4)
> td:nth-child(1) > ul:nth-child(2) > li:nth-child(1)").map {|m| m.text}
```

Con esto obtenemos una de las efemerides. Pero queremos todo el listado: para terminar el ejemplo, limpiamos el ultimo elemento li:nth-child(1), y tratamos de capturar todos los li que penden de ul:

```
nokogiri_object.css("#mw-content-text > table:nth-child(3) >
tr:nth-child(1) > td:nth-child(3) > table:nth-child(1) > tr:nth-child(4)
> td:nth-child(1) > ul:nth-child(2) > li").map {|m| m.text}

=> ["1810.– Nace Theodor Schwann, fisiólogo y botánico alemán, descubridor de la
pepsina.",
 "1941.– Japón bombardea la base de Pearl Harbor y provoca la entrada de Estados Unidos
en la Segunda Guerra Mundial (en la imagen, propaganda de guerra después del ataque).",
 "1990.– Se suicida en Nueva York el activista anticastrista Reinaldo Arenas, novelista,
dramaturgo y poeta cubano."]
```

En pocas palabras la técnica es acercarse a los datos con Xpath o CSS y luego afinar la búsqueda recorriendo los nodos con el mapeo estilo DOM que hace Nokogiri.

© Copyright Monday, Dec 19, 2016, 12:46 PM by COMPANYNAME