Extract data from an html document with CSS package

François Guillem

January 31, 2013

Many data are available on internet bat they often are encapsulated in html pages and their extraction is difficult. In R, you can use the package XML to perform this extraction but it is quite tedious.

Package CSS provides wrapper functions that facilitate extraction of data from html document : these functions behave like the one in package XML, but instead of using xpath query, they use CSS paths to select elements ; moreover some functions facilitate extraction of specific information like numbers or urls.

What is an html document ?

An html document is a text document with a special structure that look like this :

```
<html>
<head></head>
<body>
    <h1> Title of the page </h1>
    <div>
        Some text.
        <a>Some link.</a>
    </div>
    <div>
        <div>Some other text.</div>
    </div>
</body>
</html>
```

It composed of a set of elements that contain text and or other elements. An element has the following form : "`<XXX> .... </XXX>`" where "XXX" is the name of the element.

An html element can have attributes that are defined like this :

```
<XXX attr1="V1" attr2="V2"> ... </XXX>
```

In html files, there are two attributes that appear very often and that are very useful for information extraction : the "id" is a unique name that identify the element in the document and the "class". Two elements cannot have the same "id" in a document, but they can have the same class. This is often the case when they have the same role and contain the same kind of information.

CSS selectors

In order to extract information from an html document, we need a way to indicate were the information is located in the document. To do so we will use what is called "CSS selectors". Here are some examples :

`XXX`	select all "XXX" elements
`#III`	select the element which id is "III"
`.CCC`	select all elements with class "CCC"
`XXX#III`	select the "XXX" element with id "III"
`XXX.CCC`	select all "XXX" elements with class "CCC"
`.CCC.DDD`	select all elements with class "CCC" and "DDD"
`XXX>YYY`	select all "YYY" elements contained in "XXX" elements
`XXX YYY`	select all "YYY" elements which have an "XXX" ancestor

Getting started with CSS package

First let's create a fake html page.

```
doc <- "<html>
<head></head>
<body>
  <div id='player1' class='player'>
    <span class='name'>Mike</span>
    <span class='level'>10</span>
    <a href='http://someurl.com'>Complete profile</a>
  </div>
  <div id='player2' class='player'>
    <span class='name'>Stan</span>
    <a href='http://someurl2.com'>Complete profile</a>
  </div>
  <div id='player3' class='player'>
    <span class='name'>Bruce</span>
    <span class='level'>21</span>
    <a href='http://someurl3.com'>Complete profile</a>
  </div>
</body>
```

```
</html>"
```

The document contains information about three players. Information for each player is contained in a div of class "player" which contains the name of the player, a link to his profile and eventually its level. Before extracting this information, one need to parse the document with the function "htmlParse":

```
library(CSS)
doc <- htmlParse(doc)
```

To extract information, we need to use the function "cssApply" and specify which elements we want to select and which function to use to perform the extraction. For instance, assume we want to know the name of the player. It is contained in an element "span" of class"name" which is contained in a div of class "player". So we can extract the names with the following command :

```
names <- cssApply(doc, ".player>.name", cssCharacter)
```

Now let's try to get the links to their profiles. they are in "a" elements. Since we want urls, we use "cssLink" instead of "cssCharacter".

```
urls <- cssApply(doc, ".player>a", cssLink)
```

Finally, to get the level of players, we use the following command :

```
levels <- cssApply(doc, ".player>.level", cssNumeric)
```

But, here the level is missing for the second player is missing, so the variable we just created contains only two values, but we would have prefered to have a vector with three values, the second one being a NA value. To do so, we need to use the function "cssApplyInNodeSet".

This function takes as input two CSS paths : the first one is the path of the elements that may contain the element containing the information we want (here ".player"), the second one is the relative path of the element containing the information (here ".level") :

```
levels <- cssApplyInNodeSet(doc, ".player", ".level", cssNumeric)
```

Finally, we can create a table containing the data we extracted :

```
data <- data.frame(Name = names, Level = unlist(levels), Profile = urls)
```

And here is the result :

```
> data
   Name Level             Profile
1  Mike    10  http://someurl.com
2  Stan    NA http://someurl2.com
3 Bruce    21 http://someurl3.com
```