Web Browser Automator

ISOM3390: Business Programming in R

Selenium: Web Browser Automator

Selenium is a **Web browser automation tool**.

It opens a browser of our choice and “drives” it to perform tasks as a human being would, such as:

- Clicking buttons
- Entering information in forms
- Searching for specific information on the web pages

Because the Selenium server is a standalone JAVA program, we need to download and install the **Java SE Development Kit** to run it.

Download the latest Selenium standalone server binary manually. Look for `selenium-server-standalone-x.xx.x.jar`.

Starting a Selenium Server

- Open the OS console (e.g., **Command Prompt** in Windows or **Terminal** on a MacOS computer) and navigate to where the binary is stored and run: `java -jar selenium-server-standalone-x.xx.x.jar`.
- For some browsers, we also need a driver program (e.g., chromedriver) that acts as a bridge between the browser and the Selenium server.
- If the browser to use requires a driver program, execute the above command with the appropriate option to locate the required driver program, e.g., `-Dwebdriver.chrome.driver=[relative path to chromedriver]`.
- By default, the Selenium server listens for connections on port `4444`. We can change it to `4445` using the `-port` option.

Run: `java -Dwebdriver.chrome.driver=chromedriver -jar selenium-server-standalone-x.xx.x.jar -port 4445`

Working with Selenium in R

The **`RSelenium`** package allows us to connect to the Selenium server and program its behaviors from within R.

```
library(RSelenium)
```

To connect to the running server, use the `remoteDriver()` function to instantiate a new `remoteDriver` object with appropriate options:

```
remDr <- remoteDriver(remoteServerAddr = "localhost", port = 4445L,
  browserName = "chrome") # "firefox", "internet explorer", "iphone", etc.
str(remDr, max.level = 1)
```

Opening the Browser

Use `remDr`'s `open()` method to send a request to the Selenium server to start the browser:

```
remDr$open(silent = TRUE)
```

We can query the status of the remote server using the `getStatus()` method:

```
remDr$getStatus() %>% str()
```

Navigating through Webpages

Navigate to a url using `navigate()`:

```
remDr$navigate("http://www.imdb.com/title/tt1856010/reviews")
```

We navigate to a second page:

```
remDr$navigate("https://www.imdb.com/title/tt0944947/reviews")
remDr$getCurrentUrl()
```

We can go back and forth using the methods `goBack()` and `goForward()`.

We can use the `refresh()` method to refresh the current page:

```
remDr$refresh()
```

Locating HTML Elements

A number of methods can be used for searching. We can search by `id`, `name`, or `class`:

```
loadmore <- remDr$findElement(using = "id", value = "load-more-trigger")  # using = 'name' or 'class'
```

Or using css selector or XPath:

```
loadmore <- remDr$findElement(using = "css", ".ipl-load-more__button")  # using = 'xpath' when using XPa
```

The method returns a `webElement` object:

```
class(loadmore)
```

Sending Events to Elements

Mimic clicking the link to load new items (iteratively until all reviews are loaded):

```
loadmore$clickElement()
```

Supported events include:

- Sending mouse events
- Sending key strokes

```
bottom <- remDr$findElement("css", "body")
bottom$sendKeysToElement(list(key = "end"))  # Scrolls down to the bottom of a page
```

Sending Events to Elements (Continue)

Sending text

```
query <- remDr$findElement(using = "css", "[name = 'q']")
query$sendKeysToElement(list("Business Programming in R", key = "enter"))
```

More examples can be found here

Returning the Page Source

Use the `getPageSource()` to get the source of the last loaded page.

```
page_source <- remDr$getPageSource()[[1]]
```

Use `read_html()` to return an XML document as before:

```
# library(rvest)

page_source %>% read_html()
```

Close the conenction after use:

```
remDr$close()
```