How to create web crawlers with Selenium

Site crawling is useful for automating certain tasks that are performed regularly on websites. You can write a crawler to interact with a website.

Site crawling is useful for automating certain tasks that are performed regularly on websites. You can write a crawler to interact with a website.

One way to create a web crawler is to use the python, scrapy framework. The drawback of this method is that the crawler does not support Javascript. It won't work with websites that use a lot of Javascript to manage the user interface. For such sites, you can write crawlers using Google Chrome and therefore can handle Javascript just like regular users use.

Automate Google Chrome with a tool called Selenium. This is a component of the software between the program and the browser, helping users redirect the browser through the program. This article will guide you through the entire process of automating Google Chrome. The general steps include:

1. Set Selenium
2. Use Google Chrome Inspector to identify parts of the site
3. Write a Java program to automate Google Chrome

The article will also teach you how to read Google Mail from Java. Although Google provides API (Application Programming Interface) to read mail, in this article we will use Selenium to interact with Google Mail because it uses a lot of Javascript.

Set Selenium

Web Driver

As explained above, Selenium includes a software component that runs as a separate process and performs alternate actions for the Java program. This component is called **Web Driver** and needs to be downloaded to the computer.

Click here to visit the Selenium download page, select the latest version and download the appropriate file for the computer operating system (Windows, Linux or MacOS). This is a ZIP archive that contains the **chromedriver.exe** file, extracting it to an appropriate location like the **C: WebDriverschromedriver.exe** drive.

Java module

The next step is to set up the necessary Java module to use Selenium. Suppose you are using Maven to build a Java program, add the following line to the **POM.xml** file.

```
org.seleniumhq.selenium

selenium-java

3.8.1
```

When running the build process, all modules require to be downloaded and installed on the computer.

Now, we will start working with Selenium. The first step is to create a **ChromeDriver entity** :

```
WebDriver driver = new ChromeDriver();
```

A Google Chrome window will open, navigating to Google's search page.

```
driver.get("http://www.google.com");
```

Get a reference to the text input elements to be able to perform a search, the text entry element named **q.** We will place HTML elements on the page using the **WebDriver.findElement () method .**

```
WebElement element = driver.findElement(By.name("q"));
```

You can send text to any element using the **sendKeys () method** . Here, we will send a search phrase and end with a new line to start searching immediately.

```
element.sendKeys("terminatorn");
```

Now doing the search, we just need to wait for the results page.

```
new WebDriverWait(driver, 10)

.until(d -> d.getTitle().toLowerCase().startsWith("terminator"));
```

This code is basically used to tell Selenium to wait 10 seconds and come back when the page title starts with the **terminator** . We will use a lambda function to determine the waiting condition.

```
System.out.println("Title: " + driver.getTitle());
```

Then, close the browser window with the command:

```
driver.quit();
```

This is a simple browser-controlled session using Java via Selenium. It looks pretty simple, but it allows you to program many things that often have to be done manually.

Use Google Chrome Inspector

Google Chrome Inspector is a tool for identifying elements used with Selenium. It allows targeting exact elements from Java to extract information as well as interactive actions such as clicking a button.

Open Google Chrome and navigate to the Justice League IMDb page (2017). Here, we will find the element that summarizes the movie content by right-clicking on the summary and selecting " **Inspect** " from the pop-up menu.

From the ' **Elements** ' tab, you can see the summary text is a **div** with a **summary_text** class.

Use CSS or XPath

Selenium supports selecting elements from the page using CSS. (Supported CSS dialer is CSS2). For example, to select the summary text from the IMDb page above, we will write:

```
WebElement summaryEl = driver.findElement(By.cssSelector("div.summary_text"));
```

You can also use XPath to select elements in the same way. To select summary text, we will write:

```
WebElement summaryEl = driver.findElement(By.xpath("//div[@class='summary_text']"
```

XPath and CSS are similar, so you can use either.

Read Google Mail from Java

Start Chrome Driver, navigate to gmail.com and wait until the page is loaded.

```
WebDriver driver = new ChromeDriver();
driver.get("https://gmail.com");new WebDriverWait(driver, 10)
.until(d -> d.getTitle().toLowerCase().startsWith("gmail"));
```

Next, find the email field (the name is set with the **identifierId** ID) and enter the email address. Then, click the **Next** button and wait for the password page to load.

```
/* Type in username/email */{
driver.findElement(By.cssSelector("#identifierId")).sendKeys(email);
driver.findElement(By.cssSelector(".RveJvd")).click();}
new WebDriverWait(driver, 10)
.until(d ->! d.findElements(By.xpath("//div[@id='password']")).isEmpty());
```

Now, enter the password, click the **Next** button again and wait for the Gmail page to load.

```
/* Type in password */{
driver
.findElement(By.xpath("//div[@id='password']//input[@type='password']"))
.sendKeys(password);
driver.findElement(By.cssSelector(".RveJvd")).click();}
```

```
new WebDriverWait(driver, 10)
.until(d ->! d.findElements(By.xpath("//div[@class='Cp']")).isEmpty());
```

Find the list of email rows and repeat each entry.

```
List rows = driver
.findElements(By.xpath("//div[@class='Cp']//table/tbody/tr"));for(WebElement tr
```

For each entry, find the **From** field. Note, some **From** items can have multiple elements depending on the number of people in the conversation.

```
{
/* From Element */
System.out.println("From: ");
for(WebElement e : tr
.findElements(By.xpath(".//div[@class='yW']/*"))){
System.out.println(" " +
e.getAttribute("email") + ", " +
e.getAttribute("name") + ", " +
e.getText());
}}
```

Now, find the object.

```
{
/* Subject */
System.out.println("Sub: " + tr.findElement(By.xpath(".//div[@class='y6']")).get
```

And the date and time of the message.

```
{
/* Date/Time */
WebElement dt = tr.findElement(By.xpath("./td[8]/*"));
```

```
System.out.println("Date: " + dt.getAttribute("title") + ", " +

dt.getText());}
```

This is the total number of emails in the page.

```
System.out.println(rows.size() + " mails.");
```

And finally, close the browser when you're done.

```
driver.quit();
```

In short, you can use Selenium with Google Chrome to crawl websites that use a lot of Javascript. And with Google Chrome Inspector, it's easy to find CSS or XPath to extract or interact with an element.

See more:

1. 5 free application building platforms do not need code
2. 17 simple HTML codes you can learn in 10 minutes
3. 10 simple CSS codes you can study in 10 minutes