The Complete Guide to Web Scraping with Java

Raluca Penciuc on Jul 07 2021

As opposed to the "time is money" mentality of the 20th century, now it's all about data. Particularly in the last decade, web scrapers have become extremely popular. It’s not hard to understand why - the Internet is brimming with valuable information that can make or break companies.

As companies are becoming aware of data extraction's benefits, more and more people are learning how to build their own scraper. In addition to having the potential to boost business, it may also act as a neat project for developers to improve their coding skills.

If you’re on team Java, but your work has nothing to do with web scraping, you will learn about a new niche where you can put your skills to good use. The article will provide a step-by-step tutorial on creating a simple web scraper using Java to extract data from websites and then save it locally SV format.

Understanding web scraping

What does web scraping refer to? Many sites do not provide their data under public APIs, so web scrapers extract data directly from the browser. It's a lot like a person copying text manually, but it's done in the blink of an eye.

When you consider that better business intelligence means better decisions, this process is more valuable than it seems at first glance. Websites are producing more and more content, so doing this operation entirely by hand is not advisable anymore.

You might be wondering, "What am I going to do with this data?". Well, let's see a few of the use cases where web scraping can really come in handy:

- Lead generation: an ongoing business requires lead generation to find clients.

- Price intelligence: a company's decision to price and market its products will be informed by competitors' prices.

- Machine learning: to make AI-powered solutions work correctly, developers need to provide training data.

Detailed descriptions and additional use cases are available in this well-written article that talks about the value of web scraping.

effectiveness of your business, creating a scraper is not that simple. Websites have many ways of identifying and stopping bots from accessing their data.

Here are some examples:

- **Completely Automated Public Turing Tests (CAPTCHAs)**: These logical problems are reasonably easy to solve for people but a significant pain for scrapers.

- **IP blocking**: if a website determines multiple requests are coming from the same IP address, it can block access to that website or greatly slow you down.

- **Honeypots**: invisible links that are visible to bots but invisible to humans; once the bots fall for the trap, the website blocks their IP address.

- **Geo-blocking**: the website may geo-block certain content For instance, you may be given regionally specific information when you asked for input from another area (for example, plane ticket prices).

Dealing with all these hurdles is no small feat. In fact, while it's not too hard to build an OK bot, it's damn difficult to make an excellent web scraper. Thus, APIs for web scraping became one of the hottest topics in the last decade.

WebScrapingAPI collects the HTML content from any website and automatically takes care of the problems I mentioned earlier. Furthermore, we are using Amazon Web Services, which ensures speed and scalability. Sounds like something you might like? Start your free WebScrapingAPI trial, and you will be able to make 5000 API calls for the first 14 days.

Understanding the Web

To understand the Web, you need to understand Hypertext Transfer Protocol (HTTP) which explains how a server communicates with a client. There are multiple pieces of information that a message contains that describe the client and how it handles data: method, HTTP version, and headers.

Web scrapers use the GET method for HTTP requests, meaning that they retrieve data from the server. Some advanced options also include the POST and the PUT methods. For details, you can view here a detailed list of the HTTP methods.

Several additional details about requests and responses can be found in HTTP headers. You can consult the complete list of them, but the ones relevant in web scraping are:

- User-Agent: indicates the application, operating system, software, and version; web scrapers rely on this header to make their requests seem more realistic.
- Host: the domain name of the server you accessed.
- Referrer: contains the source site the user visited; accordingly, the content displayed can differ, so this fact has to be considered as well.
- Cookie: keeps confidential information about a request and the server (such as authentication tokens).

Understanding Java

Java, an open-source, object-oriented language, making it one of the most popular programming languages. Almost two decades have passed since we first encountered Java, and the programming language has become increasingly accessible.

A lot of Java's changes have been aimed at decreasing the code implementation dependencies. Because of this, many developers favor the language, but it has other advantages as well:

- It’s open-source;
- It provides a variety of APIs;
- It’s cross-platform, providing more versatility;
- It has detailed documentation and reliable community support.

Making your own web scraper

Now we can start talking about extracting data. First things first, we need a website that provides valuable information. For this tutorial, we chose to

Step 1: Set up the environment

To build our Java web scraper, we need first to make sure that we have all the prerequisites:

- Java 8: even though Java 11 is the most recent version with Long-Term Support (LTS), Java 8 remains the preferred production standard among developers.

- Gradle: is a flexible open-source build automation tool with a wide range of features, including dependency management (requires Java 8 or higher);

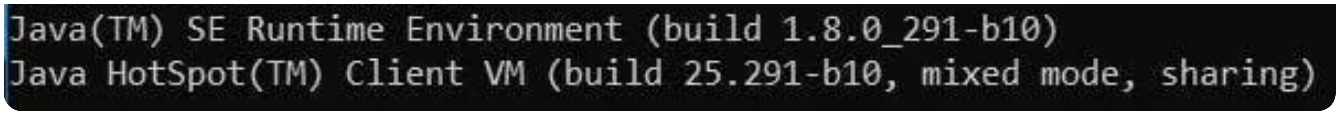
- A Java IDE: in this guide, we will use IntelliJ IDEA, as it makes the integration with Gradle pretty straightforward.

- HtmlUnit: can simulate browser events such as clicking and submitting forms when scraping and has JavaScript support.

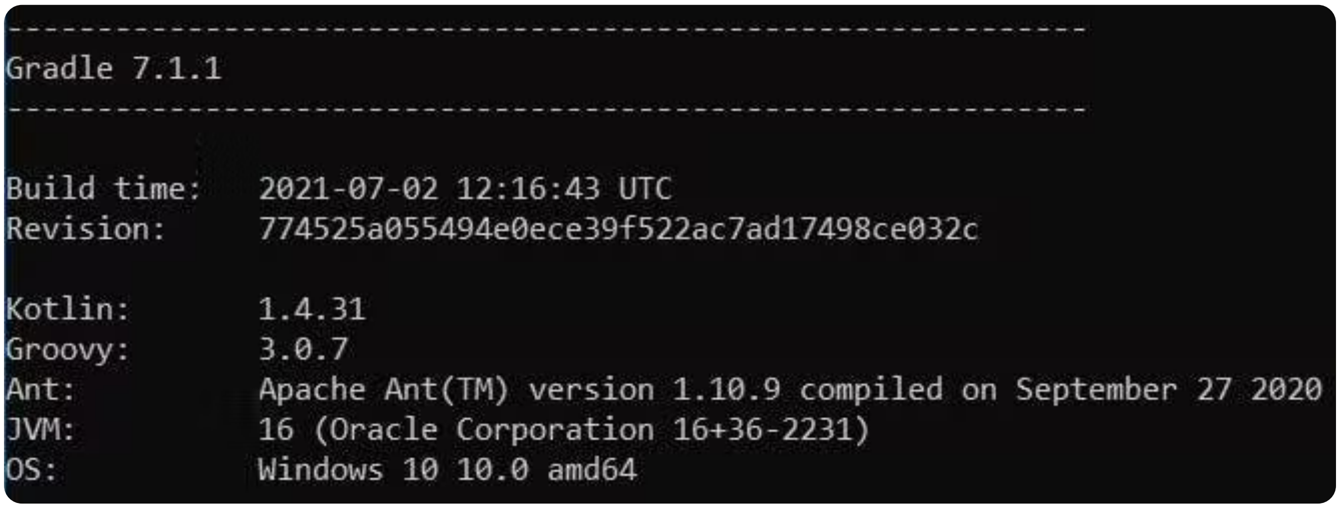
After installation, we should verify if we followed the official guides correctly. Open a terminal and run the following commands:

```
> java -version
> gradle -v
```

These should show you the versions of Java and Gradle that are installed on



```
Java(TM) SE Runtime Environment (build 1.8.0_291-b10)
Java HotSpot(TM) Client VM (build 25.291-b10, mixed mode, sharing)
```



```
------------------------------------------------------------
Gradle 7.1.1
------------------------------------------------------------

Build time:   2021-07-02 12:16:43 UTC
Revision:     774525a055494e0ece39f522ac7ad17498ce032c

Kotlin:       1.4.31
Groovy:       3.0.7
Ant:          Apache Ant(TM) version 1.10.9 compiled on September 27 2020
JVM:          16 (Oracle Corporation 16+36-2231)
OS:           Windows 10 10.0 amd64
```

If there is no error popping up, then we are good to go.

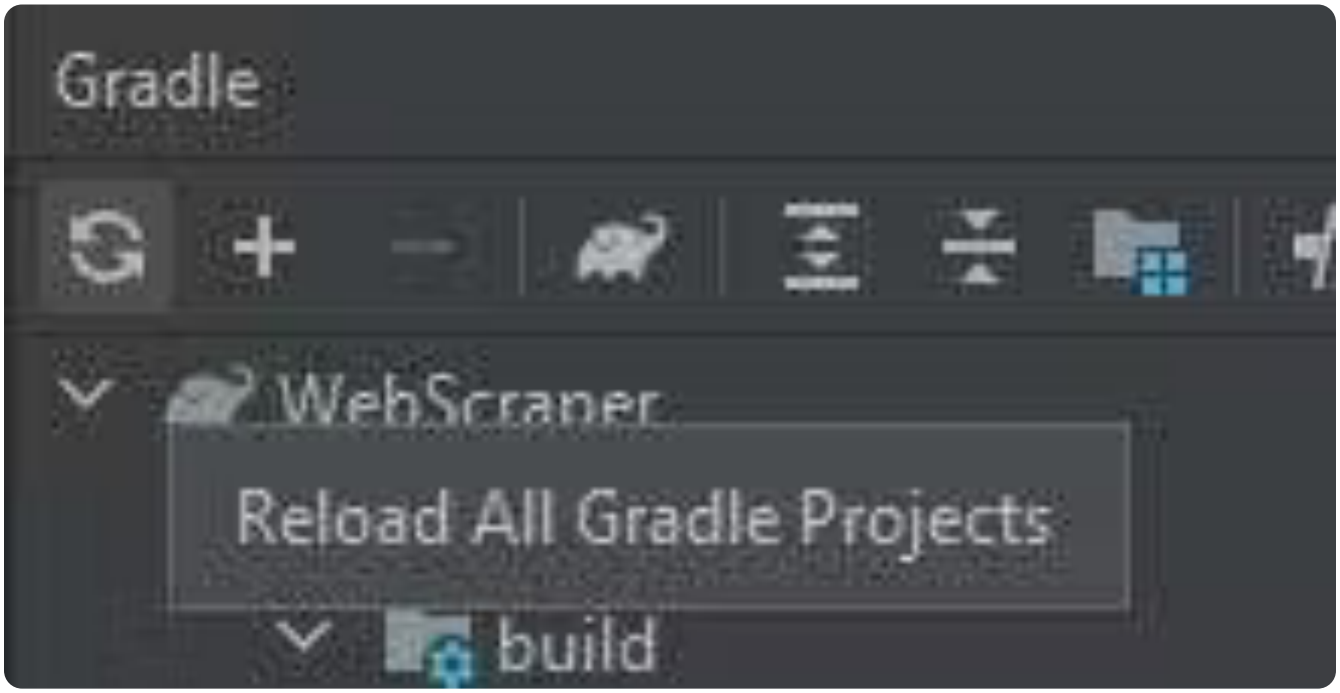
Now let's create a project so we can start writing the code. Luckily for us, JetBrains offers a well-written tutorial on how to get started with IntelliJ and Gradle, so we don't get lost throughout the configurations.

Ensure that once you create the project, let the IDE finish the first build, as you will have an automatically generated file tree.

Once it's done, open your "build.gradle" file and add the following line in the "dependencies" block:

```
implementation('net.sourceforge.htmlunit:htmlunit:2.51.0')
```

from the right-sided Gradle toolbox so you'll get rid of all the "Not found" warnings.



Step 2: Inspect the page you want to scrape

Cool, let's move on! Navigate to the page you want to scrape and right-click anywhere on it, then hit "Inspect element". The developer console will pop up, where you should see the HTML of the website.

Step 3: Send an HTTP request and scrape the HTML

Now, to get that HTML on our local machine, we have to send an HTTP request using HtmlUnit, that will return the document. Let's get back to the IDE, and put this idea into code.

First, write the imports that we need to use HtmlUnit:

```
import com.gargoylesoftware.htmlunit.*;
import com.gargoylesoftware.htmlunit.html.*;
import java.io.IOException;
import java.util.List;
```

Then we initialize a WebClient and send an HTTP request to the website that will return a HtmlPage. It's important to remember to close the connection

```
WebClient webClient = new WebClient(BrowserVersion.CHROME);

try {
   HtmlPage page = webClient.getPage("https://foodnetwork.co.uk
/italian-family-dinners/");


webClient.getCurrentWindow().getJobManager().removeAllJobs();
   webClient.close();
   recipesFile.close();

} catch (IOException e) {
   System.out.println("An error occurred: " + e);
}
```

It is worth mentioning that HtmlUnit will throw a bunch of error messages in the console that will make you think that your PC will explode. Well, have no fear because you can safely ignore 98% of them.

They are mainly caused by HtmlUnit trying to execute the Javascript code from the website's server. However, some of them can be actual errors that show a problem in your code, so it's better to pay attention to them when you run your program.

You can skip viewing a part of these useless errors by configuring some options to your WebClient:

```
webClient.getOptions().setCssEnabled(false);
webClient.getOptions().setPrintContentOnFailingStatusCode(false
);
```

Step 4: Extracting specific sections

So, we have an HTML document, but we want data, which means that we should parse the previous response into human-readable information.

Starting with baby steps, let's extract the title of the website. We can do this with the help of the built-in method getTitleText:

```
String title = page.getTitleText();
System.out.println("Page Title: " + title);
```

Moving forward, let's extract all the links from the website. For this, we have the built-in getAnchors and getHrefAttribute methods that will extract all the *<a>* tags from the HTML and then will retrieve the value of the *href* attribute:

```
List<HtmlAnchor> links = page.getAnchors();
for (HtmlAnchor link : links) {
   String href = link.getHrefAttribute();
   System.out.println("Link: " + href);
}
```

that spare you hours of reading documentation.

Let’s take a more realistic example. We need to extract all the recipes from the website, their title, and their address more precisely.

If you inspect one of the recipe cards, you can see that all the information we need is through the link’s attributes, which means that all we need to do is look for the links that have the class “card-link” and get their attributes.

```
List<?> anchors = page.getByXPath("//a[@class='card-link']");
for (int i = 0; i < anchors.size(); i++) {
   HtmlAnchor link = (HtmlAnchor) anchors.get(i);
   String recipeTitle = link.getAttribute("title").replace(',',
';');
   String recipeLink = link.getHrefAttribute();
}
```

This time we use an XPath expression to search for links at any depth through the HTML document. Then, we iterate through the result list and extract the title and the href attribute of each one of them.

Step 5: Export the data to CSV

application, a recipe aggregator in our case. So, to do that, we need to export the parsed data to an external file.

We will create a CSV file, as it can be easily read by another application and opened with Excel for further processing. First, just one more import:

```
import java.io.FileWriter;
```

Then we initialize our FileWriter that will create the CSV in "append" mode:

```
FileWriter recipesFile = new FileWriter("recipes.csv", true);
recipesFile.write("id,name,link\n");
```

After creation, we also write the first line of the CSV that will be the table's head. Now we get back to the previous loop, where we parsed all the recipe cards, and we complete with the following line:

```
recipesFile.write(i + "," + recipeTitle + "," + recipeLink +
"\n");
```

We're done writing to the file, so now it's time to close it:

```
recipesFile.close();
```

easy to forward way.

Conclusion and alternatives

That concludes our tutorial. I hope that this article was informative and gave you a better understanding of web scraping.

As you can imagine, this technology can do a lot more than fuel recipe aggregators. It’s up to you to find the correct data and analyze it to create new opportunities.

But, as I said at the start of the article, there are many challenges web scrapers need to face. Developers might find it exciting to solve these issues with their own web scraper as it’s a great learning experience and a lot of fun. Still, if you have a project to finish, you may want to avoid the costs associated with that (time, money, people).

possible blocking points such as javascript rendering, proxies, CAPTCHAs, etc., WebScrapingAPI overcomes them all and provides a customizable experience. There is also a free trial option, so if you aren't quite sure yet, why not give it a shot?

Related articles

Guides

Top 5 WebScraping C# Tools in 2022

WebScraping C# is a standard tool used by WebScraping brands. I will focus on the top brands offering the service in 2022 and why is WebScraping API coming on top of my list.

WebscrapingAPI
Nov 02 2022 · 10 min read

Guides

Web Scraper Service – Data Extraction Made Easy in 2022

Find out how a web scraper service can help you gain valuable insights and re-orient your marketing strategy for increasing profits.

WebscrapingAPI
Nov 03 2022 · 10 min read

Guides

Top 7 Best Charles Proxy alternatives for you

Charles proxy is one of the most common debugging tools today. Let’s explore its pros and cons with numerous alternatives

WebscrapingAPI
Oct 31 2022 · 9 min read

Your work email

See how it works

Products

Scraper API

Google Search Results Scraper API

Legal

Terms & Conditions

Privacy Policy

Cookie Policy

Service Agreement

GDPR

Support

Contact Us

Documentation

WebScrapingAPI SDK

Blog

Status

Web Scraping Guides

The Ultimate Guide to Web Scraping

Web Scraping with Python

Web Scraping with PHP

Web Scraping with NodeJS

Web Scraping with R

Web Scraping with C#

Web Scraping with Elixir

Web Scraping with Rust

Web Scraping with Go

Resources

The Beginner's Guide to Extracting Data With APIs

Best Web Scraping Tools

The Ultimate Web Scraping Tips & Tricks List

The 7 Best Web Scraping Dedicated and Shared Proxy Providers

The Top 7 Free Proxy Lists for Web Scraping

Web Scraping vs Web Crawling
