A Very Simple LaTeX 2_ε Template

Chao Pan Xiaosu Tong Jiasen Yang

December 15, 2015

Abstract

This is the paper's abstract ...

1 Data Download

In this section we will illustrate the process of data preparation of the project. Concretely, we will detail how to download text articles from different websites in parallel, and then save the content of articles on HDFS as text files. There are two main sources of articles included in our analysis: Public Corruption section from the FBI website and New York Times articles Archive website.

1.1 Corruption Articles from FBI

Public Corruption articles from FBI website includes 2,000 articles about different types of public corruption all over the United States as of November 11 2015. The FBI website organized those articles as following: every 20 URL links, each of which links to one article, are demonstrated on one front web page. For instance, the URL links for the first 20 articles are on front website of `https://www.fbi.gov/collections/public-corruption?b_start:int=0`, and the second 20 articles' URL links are listed on `https://www.fbi.gov/collections/public-corruption?b_start:int=20`, until the last 20 URL links are listed on front website `https://www.fbi.gov/collections/public-corruption?b_start:int=1980`. It is noticeable that all those websites which contain articles' URL are sharing the same prefix, and only varies the last integer from 0 to 1980 by 20. So the first task is to parse these 100

websites and save those articles' URL as a `RDD` object using Spark API in Scala.

1.1.1 Get URL Links of Articles

In order to parse these 100 front websites and extract the articles' URL in parallel, we defined a function named `getLinksFromOnePage` which take an integer `Index` as input argument. In the function, we first concatenated the `Index` with the predefined prefix string object "`https://www.fbi.gov/collections/public-corruption?b_start:int=`" to be a new string object named `url`. And then we utilized the `jsoup` package to create a `Connection` object, specifically, called `Jsoup.connect` function using `url` as input argument. However, we cannot just directly call this `Jsoup.connect` function, since the FBI website can identify our query to their web pages as a robot behavior, and will return empty content for the query. The reason for this is because the query we generated based on `Jsoup.connect` function does not include basic query information which identifies the browser and provides certain system details to the server hosting the websites we are trying to visit, in this case, the FBI server. The solution to this problem is very trivial. We defined a string object named `userAgentString` which contains our browser and system details information, and passed these information to the connect by calling the method `userAgent` of `Jsoup.connect(url)`. Once the connect object is created, we called the `get` method of connect to fetch and parse the HTML file of the `url`, which is saved as a `Document` object. `Document` object has several methods which is used to extract specific elements from a HTLM file. For example, if the HTML file contains following structure:

```
<dl class="portletCollectionRecentSpeeches" id="0">
  <dd class="contenttype-pressrelease">
  <span class="created">12.11.15</span>
  <div class="portletItemSpeechContent">
  <span class="RecentSpeechesContent">
  <a href="https://www.fbi.gov/indianapolis/press-releases/
    2015/anderson-indiana-police-officer-faces-drug-
    distribution-charges" class="">
```

```
    <h3>
      Anderson, Indiana Police Officer Faces Drug
      Distribution Charges
    </h3>
  </a></span></div></dd>
</dl>
```

we can easily extract the hyper-reference, which is the url of an article, by calling `getElementsByTag("dl").select("a").attr("href")`. But in the FBI websites, each web page contains 20 articles' URL. So we considered `map` function to get all "href". Next we demonstrate the whole picture of the `getLinksFromOnePage` function:

```
def getLinksFromOnePage(index: Int) = {
  val url: String = baseUrl + index
  val connToLinkPage: Connection = Jsoup.connect(url)
  .userAgent(userAgentString)
  val docOfLinkPage: Document = connToLinkPage.get()
  // extract all article links on the page
  val articleLinks: Elements = docOfLinkPage
  .getElementsByTag("dl").select("a")
  articleLinks.toList.map {link => link.attr("href")}
}
```

Moreover, we would like to parallelly parse those url links of articles, and save the content of articles as text string. In order to parse url links in parallel, we have to generate an `RDD` of string object, each of which contains a string of url. So what we did is first create an array of integer which is a sequence from 0 to 1980 increment by 20. Each integer represents one front website of the FBI. Then parallelize this array object to be an RDD using Spark API. Finally we utilized `flatMap` method of `RDD` object to generate multiple url string elements from one input integer of `RDD`. Details are demonstrated as following:

```
val pageIndexRDD =  sc.parallelize(pageIndexArray)
val articleLinksRDD = pageIndexRDD.
  flatMap(ind => getLinksFromOnePage(ind)).persist()
```

1.1.2 Save Articles on HDFS

After the links of articles are available, we parsed each article from the link `RDD` in parallel and save content of each article as a JSON text file on HDFS.

1.2 Articles from The New York Times