Сбор данных и отправка в Apache Kafka

Введение.

Для анализа потоковых данных необходимы источники этих данных. Так же важна сама информация, которая предоставляется источниками. А источники с текстовой информацией, к примеру, еще и редки.

Из интересных источников можно выделить следующие: [twitter](<https://developer.twitter.com/en/docs/twitter-api/getting-started/guide>), [vk](<https://vk.com/dev/methods>). Но эти источники подходят не под все задачи.

Есть источники с нужными данными, но эти источники не потоковые. Здесь можно привести следующее сслыки: [public-apis](<https://github.com/public-apis/public-apis>).

При решении задач, связанных с потоковыми данными, можно воспользоваться старым способом.

Скачать данные и отправить в поток.

Для примера можно воспользоваться следующим источником: [imdb](<https://www.imdb.com/>).

Следует отметить, что [imdb](https://www.imdb.com/) предоставляет данные самостоятельно. См. [IMDb Datasets](https://www.imdb.com/interfaces/). Но можно принять, что данные собранные напрямую содержат более актуальную информацию.

Язык: Java 1.8.

Библиотеки: kafka 2.6.0, jsoup 1.13.1.

Сбор данных.

Сбор данных представляет из себя сервис, который по входным данным загружает html-страницы, ищет нужную информацию и преобразует в набор объектов.

Итак источник данных: [imdb](<https://www.imdb.com/>). Информация будет собираться о фильмах и будет использован следующий запрос: [https://www.imdb.com/search/title/?release\_date=%s,%s&countries=%s](https://www.imdb.com/search/title/?release_date=%25s,%25s&countries=%25s)

Где 1, 2 параметр – это даты. 3 параметр – страны.

Для лучшего понимания источника данных можно обратится к следующему ресурсу: [imdb-extensive-dataset](<https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset>).

Интерфейс для сервиса:

**public interface** MovieDirectScrapingService {  
 Collection<Movie> scrap();  
}

Класс Movie – это класс, которые содержит информацию об одном фильме (или о шоу и т.п.).

**class** Movie {  
 **public final** String **titleId**;  
 **public final** String **titleUrl**;  
 **public final** String **title**;  
 **public final** String **description**;  
 **public final** Double **rating**;  
 **public final** String **genres**;  
 **public final** String **runtime**;  
 **public final** String **baseUrl**;  
 **public final** String **baseNameUrl**;  
 **public final** String **baseTitleUrl**;  
 **public final** String **participantIds**;  
 **public final** String **participantNames**;  
 **public final** String **directorIds**;  
 **public final** String **directorNames**;

…

Для реализации понадобятся следующие методы.

Анализ данных на одной странице.

Информация собирается следующим образом. Данные закачиваются с помощью jsoup. Далее ищутся нужные html-элементы и трансформируются в экземпляры для фильмов.

String scrap(String url, List<Movie> items) {  
 Document doc = **null**;  
 **try** {  
 doc = Jsoup.*connect*(url).header(**"Accept-Language"**, **language**).get();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **if** (doc != **null**) {  
 collectItems(doc, items);  
 **return** nextUrl(doc);  
 }  
 **return ""**;  
}

Поиск ссылки на следующею страницу.

String nextUrl(Document doc) {  
 Elements nextPageElements = doc.select(**".next-page"**);  
 **if** (nextPageElements.size() > 0) {  
 Element hrefElement = nextPageElements.get(0);  
 **return baseUrl** + hrefElement.attributes().get(**"href"**);  
 }  
 **return ""**;  
}

Тогда основной метод будет таким. Формируется начальная строка поиска. Закачиваются данные по одной странице. Если есть следующая страница, то идет переход к ней. По окончании передаются накопленные данные.

@Override  
**public** Collection<Movie> scrap() {  
 String url = String.*format*(  
 **baseUrl** + **"/search/title/?release\_date=%s,%s&countries=%s"**,  
 **startDate**, **endDate**, **countries** );  
 List<Movie> items = **new** ArrayList<>();  
 String nextUrl = url;  
 **while** (**true**) {  
 nextUrl = scrap(nextUrl, items);  
 **if** (**""**.equals(nextUrl)) {  
 **break**;  
 }  
 **try** {  
 Thread.*sleep*(50);  
 } **catch** (InterruptedException e) {  
 }  
 }  
 **return** items;  
}

Подробности по остальным методам можно найти в ссылках на ресурсы.

Отправка данных в топик.

Формируется следующий сервис: MovieProducer. Здесь будет один единственный публичный метод: run.

Создается продюсер для кафки. Загружаются данные из источника. Трансформируются и отправляются в топик.

**public void** run() {  
 **try** (SimpleStringStringProducer producer = **new** SimpleStringStringProducer(  
 **bootstrapServers**, **clientId**, **topic**)) {  
 Collection<Data.Movie> movies = **movieDirectScrapingService**.scrap();  
 List<SimpleStringStringProducer.KeyValueStringString> kvList = **new** ArrayList<>();  
 **for** (Data.Movie move : movies) {  
 Map<String, String> map = **new** HashMap<>();  
 map.put(**"title\_id"**, move.**titleId**);  
 map.put(**"title\_url"**, move.**titleUrl**);  
 …  
 String value = JSONObject.*toJSONString*(map);  
 String key = UUID.*randomUUID*().toString();  
 kvList.add(**new** SimpleStringStringProducer.KeyValueStringString(key, value));  
 }  
 producer.produce(kvList);  
 }  
}

Теперь все вместе.

Формируются нужные параметры для поиска. Загружаются данные и отправляются в топик.

Для этого понадобится еще один класс: MovieDirectScrapingExecutor. С одним публичным методом: run.

В цикле создаются данные для поиска из текущей даты. Происходит загрузка и отправка данных в топик.

**public void** run() {  
 **int** countriesCounter = 0;  
 List<String> countriesSource = Arrays.*asList*(**"us"**);  
  
 **while** (**true**) {  
 **try** {  
 LocalDate localDate = LocalDate.*now*();  
  
 **int** year = localDate.getYear();  
 **int** month = localDate.getMonthValue();  
 **int** day = localDate.getDayOfMonth();  
  
 String monthString = month < 9 ? **"0"** + month : Integer.*toString*(month);  
 String dayString = day < 9 ? **"0"** + day : Integer.*toString*(day);  
  
 String startDate = year + **"-"** + monthString + **"-"** + dayString;  
 String endDate = startDate;  
  
 String language = **"en"**;  
 String countries = countriesSource.get(countriesCounter);  
  
 execute(language, startDate, endDate, countries);  
  
 Thread.*sleep*(1000);  
  
 countriesCounter += 1;  
 **if** (countriesCounter >= countriesSource.size()) {  
 countriesCounter = 0;  
 }  
  
 } **catch** (InterruptedException e) {  
 }  
 }  
}

Для запуска потребуется экземпляр класса MovieDirectScrapingExecutor, который можно запустить с нужными параметрами, к примеру, из метода main.

Пример данных.

{

"base\_name\_url": "https:\/\/www.imdb.com\/name",

"participant\_ids": "nm7947173~nm2373827~nm0005288~nm0942193~",

"title\_id": "tt13121702",

"rating": "0.0",

"base\_url": "https:\/\/www.imdb.com",

"description": "It's Christmas time and Jackie (Carly Hughes), an up-and-coming journalist, finds that her life is at a crossroads until she finds an unexpected opportunity - to run a small-town newspaper ... See full summary »",

"runtime": "",

"title": "The Christmas Edition",

"director\_ids": "nm0838289~",

"title\_url": "\/title\/tt13121702\/?ref\_=adv\_li\_tt",

"director\_names": "Peter Sullivan~",

"genres": "Drama, Romance",

"base\_title\_url": "https:\/\/www.imdb.com\/title",

"participant\_names": "Carly Hughes~Rob Mayes~Marie Osmond~Aloma Wright~"

}

Подробности можно найти в ссылках на ресурсы.

Тесты.

Для тестирования основной логики, которая связана с отправкой данных, можно воспользоваться юнит-тестами. В тестах предварительно создается kafka-сервер.

См. [Apache Kafka и тестирование с Kafka Server](https://habr.com/ru/post/527712/).

Сам тест: MovieProducerTest.

**public class** MovieProducerTest {  
 @Test  
 **void** simple() **throws** InterruptedException {  
 String brokerHost = **"127.0.0.1"**;  
 **int** brokerPort = 29092;  
 String zooKeeperHost = **"127.0.0.1"**;  
 **int** zooKeeperPort = 22183;  
 String bootstrapServers = brokerHost + **":"** + brokerPort;  
 String topic = **"q-data"**;  
 String clientId = **"simple"**;  
 **try** (KafkaServerService kafkaServerService = **new** KafkaServerService(  
 brokerHost, brokerPort, zooKeeperHost, zooKeeperPort  
 )  
 ) {  
 kafkaServerService.start();  
 kafkaServerService.createTopic(topic);

MovieDirectScrapingService movieDirectScrapingServiceImpl = () -> Collections.*singleton*(  
 **new** Data.Movie(…)  
 );  
 MovieProducer movieProducer =  
 **new** MovieProducer(bootstrapServers, clientId, topic, movieDirectScrapingServiceImpl);  
 movieProducer.run();  
  
 kafkaServerService.poll(topic, **"simple"**, 1, 5, (records) -> {  
 *assertTrue*(records.count() > 0);  
 ConsumerRecord<String, String> record = records.iterator().next();  
 JSONParser jsonParser = **new** JSONParser();  
 JSONObject jsonObject = **null**;  
 **try** {  
 jsonObject = (JSONObject) jsonParser.parse(record.value());  
 } **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
 *assertNotNull*(jsonObject);

…  
 });  
  
 Thread.*sleep*(5000);  
 }  
 }  
}

Заключение.

Конечно, описанный здесь способ получения источника потоковых данных, строго потоковым не является. Но для исследований и прототипов вполне может сойти.

Ссылки и ресурсы.

[Исходный код]( https://github.com/kartzum/intro\_to\_apache\_kafka/tree/main/story/resources/k-plain-sources).