

Introducción

En este informe explico brevemente mi proceso para completar la etapa individual del laboratorio 3 por medio de la metodología elegida: una mezcla de ChatGPT, Documentación y Stack Overflow. Luego paso a expandir sobre qué problemas surgieron y como estos recursos me sirvieron para solucionarlos.

Método

Elegí trabajar como usualmente lo hago: leo documentación introductoria sobre las tecnologías / herramientas que no conozco y trato de tener un mapa mental de "lo que no sé" y luego voy llenando esos "huecos" según los problemas que van surgiendo.

Recientemente empecé a usar ChatGPT para encontrar información conceptual(no tanto para generar código) más rápido que a través de buscadores como Google.

Las herramientas que usé fueron:

- Inteligencia artificial(**ChatGPT**) para **preguntas conceptuales**.
- **Documentación** para **ejemplos** de código.
- **Documentación** para how-to relacionado a herramientas(**IDE, Build Tool**).
- StackOverflow para preguntas sobre mensajes de error y ChatGPT en su defecto.

Proceso

En un principio anticipé que instalar spark incluiría, además, otras dependencias de Spark y la librería de JSON(dependencias de Spark principalmente) así que busqué sobre **Maven**, una herramienta para administrar proyectos en Java.

Welcome
License
ABOUT MAVEN
What is Maven?
Features
Download
Use
Release Notes
DOCUMENTATION
Maven Plugins
Maven Extensions
Index (category)
User Centre
Plugin Developer Centre
Maven Repository

Welcome to Apache Maven


Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

If you think that Maven could help your project, you can find out more information in the "About Maven" section of the navigation. This includes an in-depth description of what Maven is and a [list of some of its main features](#).

This site is separated into the following sections, depending on how you'd like to use Maven:


Use	Download, Install, Configure, Run Maven	Maven Plugins and Maven Extensions
	Information for those needing to build a project that uses Maven	Lists of plugins and extensions to help with your builds.
Extend	Write Maven Plugins	Improve the Maven Central Repository
	Information for developers writing Maven plugins.	Information for those who may or may not use Maven, but are interested in getting project metadata into the central repository .
Contribute	Help Maven	Develop Maven
	Information if you'd like to get involved. Maven is an open source community and welcomes contributions.	Information for those who are currently Maven developers, or who are interested in contributing to the Maven project itself.

También sabía sobre **IntelliJ Idea**, un IDE para desarrollar proyectos en Java.

 **IntelliJ IDEA** 2023.1

Shortcuts: Windows

[Get IntelliJ IDEA](#)



Getting started

Accessibility

IntelliJ IDEA overview

Installation guide

Create your first Java application

Learn IDE features

Install plugins

Work offline

Migrate from Eclipse

Migrate from NetBeans

IntelliJ IDEA for Education

IDE configuration


Project configuration

Write and edit source code

Compilation and building


Getting started

Check out the most useful and popular topics to get you started with IntelliJ IDEA.



Accessibility

Use a screen reader, adjust the colors and font size to assist users with sight deficiency



Discover IntelliJ IDEA

Discover the main features and possibilities of the IDE

Entre sus features(listados en su página) confirmé que incluía soporte para Maven. Así que decidí usar el IDE.

Conocimiento básico de Spark

No tenía idea de spark, así que lo primero que hice fue pedirle a ChatGPT que me resuma los principales conceptos. La respuesta fue la siguiente:

G

what are the most important concepts to learn about Apache Spark?



Apache Spark is a powerful open-source distributed computing system designed for big data processing and analytics. To effectively work with Spark, it is crucial to understand several key concepts:



1. Resilient Distributed Datasets (RDDs): RDDs are the fundamental data structure in Spark. They represent an immutable distributed collection of objects across a cluster. RDDs provide fault tolerance and can be processed in parallel across nodes.
2. Transformations and Actions: Spark operations are divided into transformations and actions. Transformations are operations that create a new RDD from an existing one (e.g., map, filter, reduceByKey), while actions compute a result or return data to the driver program (e.g., count, collect, save).
3. Lazy Evaluation: Spark uses lazy evaluation, meaning transformations on RDDs are not executed immediately. Instead, Spark builds up a logical execution plan (DAG) and optimizes it before triggering the actual computation when an action is called. This approach improves performance and reduces unnecessary computations.

[Regenerate response](#)

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. [ChatGPT May 24 Version](#)

Instalación de spark

En la documentación de IntelliJ leí como manejar un proyecto Maven desde el IDE, que básicamente consiste en declarar las dependencias del proyecto a un archivo `pom.xml`. Este archivo contiene los datos necesarios sobre el proyecto para que Maven pueda instalar las dependencias necesarias y compilar nuestro código.

Uso de Spark en el contexto del lab anterior

El lab anterior realizamos(al menos en la implementación de nuestro grupo) tareas de forma **secuencial**. Mi objetivo fue entonces hacer uso de `map` y `reduce` para realizar estas tareas de forma paralelizada.

Ejemplos

Para esto usé principalmente ejemplos de la [página sobre RDD's](#) y nociones sobre `map` y `reduce` que ya tenía.

Map

Por ejemplo, en las requests y parsing de los Feeds:

```

JavaRDD<SimpleSubscription> rSubList =
sc.parallelize(subscriptions.getSubscriptionList());

var feeds = rSubList.flatMap((simpleSubscription -> {
    List<Function0<Tuple2<Feed, String>>> frs = new ArrayList<>
();

    for (int j = 0, szj =
simpleSubscription.getUrlParametersSize(); j < szj; j++) {
        final int i = j;
        frs.add(() -> {
            try {
                return new
Tuple2(simpleSubscription.parse(i), null);
            } catch (Exception e) { // Omito todos los
casos de error
                return new Tuple2(null,
                    "Error while getting
feed in "
                                +
simpleSubscription.getFormattedUrlForParameter(i));
            });
        }

        return frs.iterator();
    })).mapToPair(Function0::apply);

```

En el snippet de arriba vemos como:

- Usamos nuestra lista de subscripciones simples para conseguir una estructura de datos propia de Spark: RDD. Esta estructura es inmutable y es usada por el framework para distribuir computo a traves de distintos CPU's o incluso distintas computadoras.
- A partir de esta lista, aplicamos un `map` para obtener una **lista de funciones "fetcher"**
- Paralelizamos estos "fetchers" para obtener los feeds en simultaneo, usando otro tipo de `map`

Reduce

En el siguiente snippet podemos ver un ejemplo de como introduce `reduce` para sumar las frecuencias de cada entidad nombrada. En este caso usamos `reduceByKey` que reduce todos los elementos de la colección con la misma clave a una única tupla clave-valor. Usamos este snippet para conseguir la frecuencia global de cada named entity.

```
parsedFeeds.flatMap(feed -> {
    if(feed.getArticleList() != null) {
        return feed.getArticleList().iterator();
    } else {
        return Collections.emptyIterator();
    }
}).flatMap(article -> {
    Heuristic heur = new QuickHeuristic();
    article.computeNamedEntities(heur);
    if(article.getNamedEntityList() != null) {
        return article.getNamedEntityList().iterator();
    } else {
        return Collections.emptyIterator();
    }
}).filter(Objects::nonNull).mapToPair(namedEntity -> new Tuple2<>
(namedEntity.getName(), namedEntity)).reduceByKey((n1, n2) -> {
    var n = new NamedEntity(n1.getName(), n1.getCategory(),
n1.getFrequency() + n1.getFrequency());
    n.setTheme(n1.getTheme());
    return n;
}).map(Tuple2::_2).foreach(namedEntity -> {
    System.out.println(namedEntity.getName());
    System.out.println(namedEntity.getFrequency());
    System.out.println(namedEntity.getCategory());
    System.out.println(namedEntity.getTheme());
    System.out.println(namedEntity.getClass().toString());
    System.out.println("-----");
});
```

Integración de OOP con map-reduce y Spark

A pesar de ser posible escribir código más conciso usando map-reduce en Spark, se presentan ciertas dificultades al integrar con OOP. Principalmente en torno al

encapsulamiento. Si la naturaleza de cierto comportamiento dentro de una de nuestras clases es secuencial, entonces no tenemos otra que:

- Replicar este comportamiento(y el estado del que depende!) fuera para poder usar Spark sobre su estado.
- Cambiar la implementación de la clase

Esto puede llegar a ser un problema si no controlamos o entendemos gran parte de un proyecto, limitando las mejoras que podemos hacer usando paralelismo.

Conclusiones

Trabajar con Spark en Java puede tener sus desafíos a la hora de integrar código orientado a objetos que haga uso de paralelismo. De todas formas, usando la documentación para obtener ejemplos y modelos predictivos para sintetizar información rápidamente esta tarea se hace posible y -al menos durante este laboratorio- llevadera.