Apache Spark basic

Kiến thức cơ bản về Apache Spark

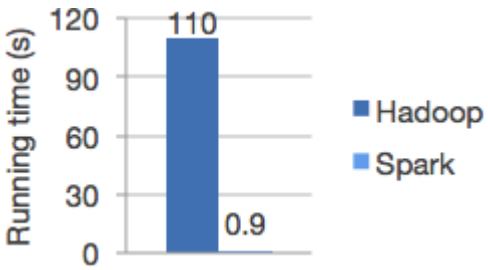
- Apache Spark overview
- Apache Spark RDD
- Hướng dẫn tạo Spark Scala Maven project trong Intellij
- Spark - SparkSession

Apache Spark overview

Ngày nay có rất nhiều hệ thống xử lý dữ liệu thông tin đang sử dụng Hadoop rộng rãi để phân tích dữ liệu lớn. Ưu điểm lớn nhất của Hadoop là được dựa trên một mô hình lập trình song song với xử lý dữ liệu lớn là MapReduce, mô hình này cho phép khả năng tính toán có thể mở rộng, linh hoạt, khả năng chịu lỗi, chi phí rẻ. Điều này cho phép tăng tốc thời gian xử lý các dữ liệu lớn nhằm duy trì tốc độ, giảm thời gian chờ đợi khi dữ liệu ngày càng lớn.

Hadoop đã được nền tảng tính toán cho rất nhiều cho một bài toàn xử lý dữ liệu lớn và các vấn đề về mở rộng tính toán song song trong các bài toàn xếp hạng. Apache Haddop cũng được sử dụng tại rất nhiều công ty lớn như Yahoo, Google. Dù có rất nhiều điểm mạnh về khả năng tính toán song song và khả năng chịu lỗi cao nhưng Apache Haddop có một nhược điểm là tất cả các thao tác đều phải thực hiện trên ổ đĩa cứng điều này đã làm giảm tốc độ tính toán đi gấp nhiều lần.

Để khắc phục được nhược điểm này thì Apache Spark được ra đời. Apache Spark có thể chạy nhanh hơn 10 lần so với Haddop ở trên đĩa cứng và 100 lần khi chạy trên bộ nhớ RAM, hình dưới biểu thị thời gian chạy của tính toán hồi quy Logistic trên Haddop và Spark.



Giới thiệu Apache Spark

Apache Spark là một open source cluster computing framework được phát triển sơ khởi vào năm 2009 bởi AMPLab tại đại học California, Berkeley.

Sau này, Spark đã được trao cho Apache Software Foundation vào năm 2013 và được phát triển cho đến nay. Apache Spark được phát triển nhằm tăng tốc khả năng tính toán xử lý của Haddop.

Spark cho phép xây dựng và phân tích nhanh các mô hình dự đoán. Hơn nữa, nó còn cung cấp khả năng truy xuất toàn bộ dữ liệu cùng lúc, nhờ vậy ta không cần phải lấy mẫu dữ liệu đòi hỏi bởi các ngôn ngữ lập trình như R.

Thêm vào đó, Spark còn cung cấp tính năng streaming, được dùng để xây dựng các mô hình real-time bằng cách nạp toàn bộ dữ liệu vào bộ nhớ. Khi ta có một tác vụ nào đó quá lớn mà không thể xử lý trên một laptop hay một server, Spark cho phép ta phân chia tác vụ này thành những phần dễ quản lý hơn. Sau đó, Spark sẽ chạy các tác vụn ày trong bộ nhớ, trên các cluster của nhiều server khác nhau để khai thác tốc độ truy xuất nhanh từ RAM.

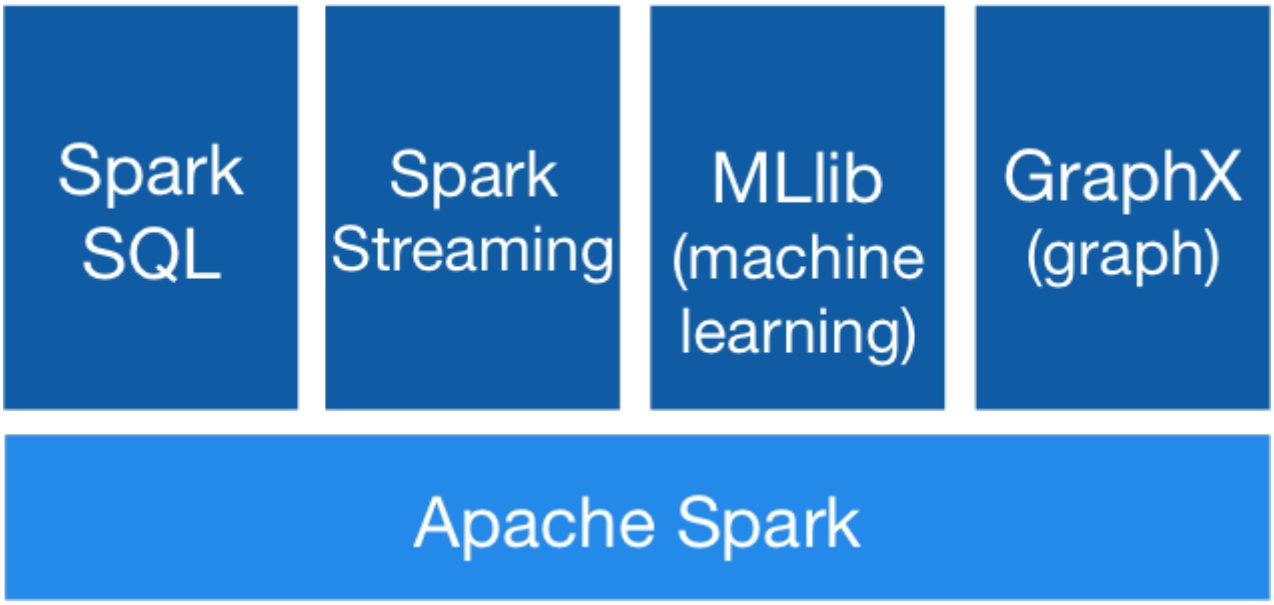
Spark sử dụng API Resilient Distributed Dataset (RDD) để xử lý dữ liệu. Spark nhận được nhiều sự hưởng ứng từ cộng đồng Big Data trên thế giới do cung cấp khả năng tính toán nhanh và nhiều thư viện hữu ích đi kèm như Spark SQL (với kiểu dữ liệu DataFrames), Spark Streaming, MLlib (machine learning: classification, regression, clustering, collaborative filtering, và dimensionality reduction) và GraphX (tính toán song song trên dữ liệu đồ thị)

Tính năng của Apache Spark

Apache Spark có các tính năng đặc trưng sau đây:

- Tốc độ: Spark có thể chạy trên cụm Hadoop và có thể chạy nhanh hơn 100 lần khi chạy trên bộ nhớ RAM, và nhanh hơn 10 lần khi chạy trên ổ cứng. Bằng việc giảm số thao tác đọc nghi lên đĩa cứng. Nó lưu trữ trực tiếp dữ liệu xử lý lên bộ nhớ.
- Hỗ trợ đa ngôn ngữ: Spark cung cấp các API có sẵn cho các ngôn ngữ Java, Scala, hoặc Python. Do đó, bạn có thể viết các ứng dụng bằng nhiều các ngôn ngữ khác nhau. Spark đi kèm 80 truy vấn tương tác mức cao.
- Phân tích nâng cao: Spark không chỉ hỗ trợ Map và Reduce, nó còn hỗ trợ truy vấn SQL, xử lý theo Stream, học máy, và các thuật toán đồ thị (Graph)

Các thành phần của Apache Spark



- **Apache Spark Core**: Spark Core là thành phần cốt lõi thực thi cho tác vụ cơ bản làm nền tảng cho các chức năng khác. Nó cung cấp khảnăng tính toán trên bộ nhớ và dataset trong bộ nhớ hệ thống lưu trữ ngoài.
- **Spark SQL**: Là một thành phần nằm trên Spark Core, nó cung cấp một sựảo hóa mới cho dữ liệu là SchemaRDD, hỗ trợ các dữ liệu có cấu trúc và bán cấu trúc.
- **Spark Streaming**: Cho phép thực hiện phân tích xử lý trực tuyến xử lý theo lô.
- **MLlib** (Machine Learning Library): MLlib là một nền tảng học máy phân tán bên trên Spark do kiến trúc phân tán dựa trên bộ nhớ. Theo các so sánh benchmark Spark MLlib nhanh hơn 9 lần so với phiên bản chạy trên Hadoop (Apache Mahout).
- **GrapX**: Grapx là nền tảng xử lý đồ thị dựa trên Spark. Nó cung cấp các Api để diễn tảcác

tính toán trong đồ thị bằng cách sử dụng Pregel Api.

Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) là một cấu trúc dữ liệu cơ bản của Spark. Nó là một tập hợp bất biến phân tán của một đối tượng. Mỗi dataset trong RDD được chia ra thành nhiều phần vùng logical. Có thể được tính toán trên các node khác nhau của một cụm máy chủ (cluster).

RDDs có thể chứa bất kỳ kiểu dữ liệu nào của Python, Java, hoặc đối tượng Scala, bao gồm các kiểu dữ liệu do người dùng định nghĩa. Thông thường, RDD chỉ cho phép đọc, phân mục tập hợp của các bản ghi. RDDs có thể được tạo ra qua điều khiển xác định trên dữ liệu trong bộ nhớ hoặc RDDs, RDD là một tập hợp có khả năng chịu lỗi mỗi thành phần có thể được tính toán song song.

Có hai cách để tạo RDDs:

- Tạo từ một tập hợp dữ liệu có sẵn trong ngôn ngữ sử dụng như Java, Python, Scala.
- Lấy từ dataset hệ thống lưu trữ bên ngoài như HDFS, Hbase hoặc các cơ sở dữ liệu quan hệ.

Xem thêm về Spark RDD

Apache Spark RDD

Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) là một cấu trúc dữ liệu cơ bản của Spark. Nó là một tập hợp bất biến phân tán của một đối tượng. Mỗi dataset trong RDD được chia ra thành nhiều phần vùng logical. Có thể được tính toán trên các node khác nhau của một cụm máy chủ (cluster).

RDDs có thể chứa bất kỳ kiểu dữ liệu nào của Python, Java, hoặc đối tượng Scala, bao gồm các kiểu dữ liệu do người dùng định nghĩa. Thông thường, RDD chỉ cho phép đọc, phân mục tập hợp của các bản ghi. RDDs có thể được tạo ra qua điều khiển xác định trên dữ liệu trong bộ nhớ hoặc RDDs, RDD là một tập hợp có khả năng chịu lỗi mỗi thành phần có thể được tính toán song song.

Có hai cách để tạo RDDs:

- Tạo từ một tập hợp dữ liệu có sẵn trong ngôn ngữ sử dụng như Java, Python, Scala.
- Lấy từ dataset hệ thống lưu trữ bên ngoài như HDFS, Hbase hoặc các cơ sở dữ liệu quan hệ.

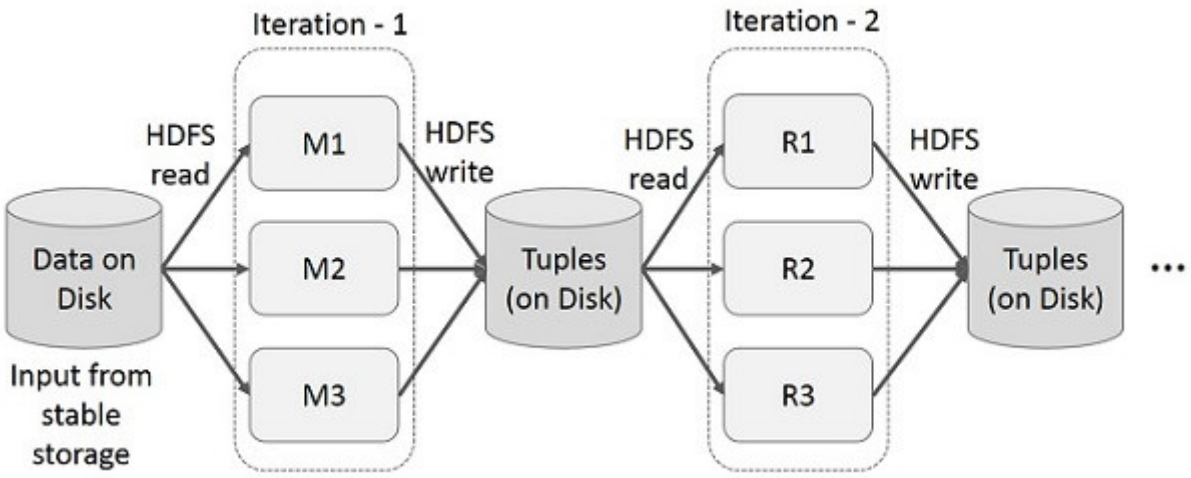
Thực thi trên MapRedure

MapReduce được áp dụng rộng rãi để xử lý và tạo các bộ dữ liệu lớn với thuật toán xử lý phân tán song song trên một cụm. Nó cho phép người dùng viết các tính toán song song, sử dụng một tập hợp các toán tử cấp cao, mà không phải lo lắng về xử lý/phân phối công việc và khả năng chịu lỗi.

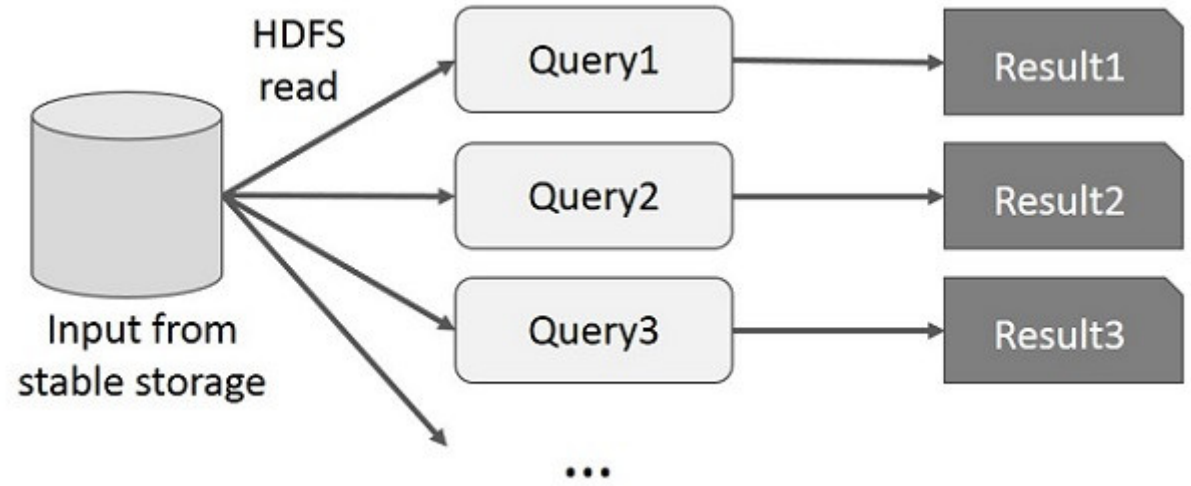
Tuy nhiên, trong hầu hết các framework hiện tại, cách duy nhất để sử dụng lại dữ liệu giữa các tính toán (Ví dụ: giữa hai công việc MapReduce) là ghi nó vào storage (Ví dụ: HDFS). Mặc dù framework này cung cấp nhiều hàm thư viện để truy cập vào tài nguyên tính toán của cụm Cluster, điều đó vẫn là chưa đủ.

Cả hai ứng dụng Lặp (Iterative) và Tương tác (Interactive) đều yêu cầu chia sẻ truy cập và xử lý dữ liệu nhanh hơn trên các công việc song song. Chia sẻ dữ liệu chậm trong MapReduce do sao chép tuần tự và tốc độ I/O của ổ đĩa. Về hệ thống lưu trữ, hầu hết các ứng dụng Hadoop, cần dành hơn 90% thời gian để thực hiện các thao tác đọc-ghi HDFS.

- Iterative Operation trên MapReduce:



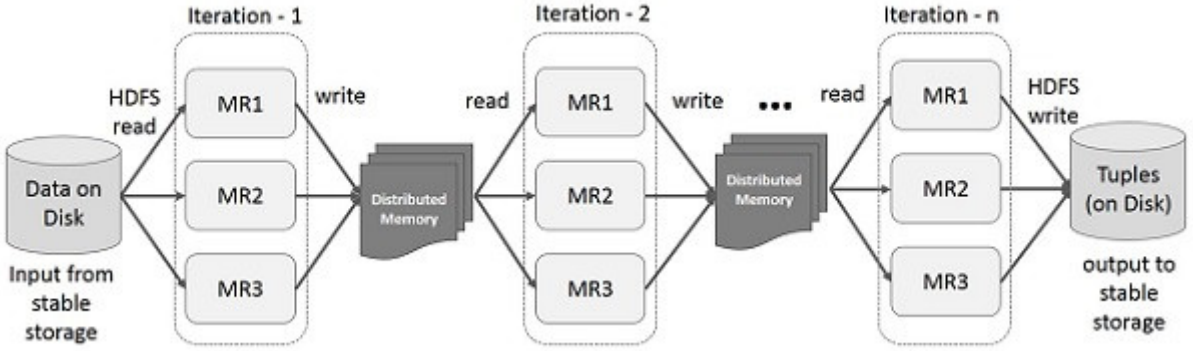
- Interactive Operations trên MapReduce:



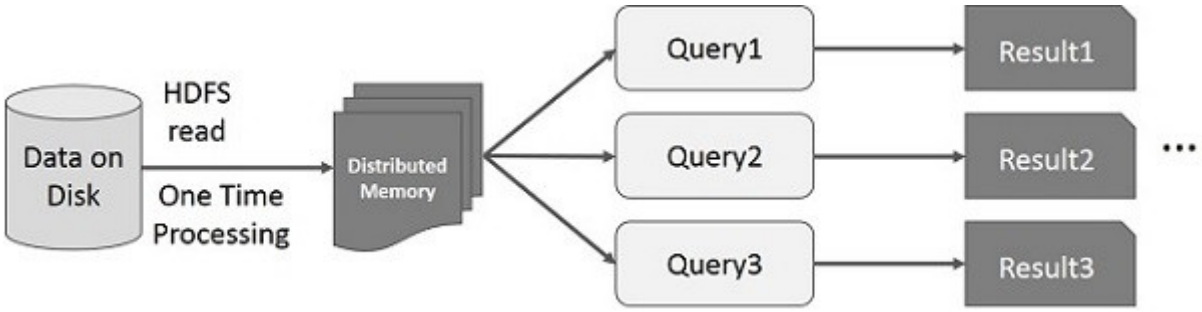
Thực thi trên Spark RDD

Để khắc phục được vấn đề về MapRedure, các nhà nghiên cứu đã phát triển một framework chuyên biệt gọi là Apache Spark. Ý tưởng chính của Spark là Resilient Distributed Datasets (RDD); nó hỗ trợ tính toán xử lý trong bộ nhớ. Điều này có nghĩa, nó lưu trữ trạng thái của bộ nhớ dưới dạng một đối tượng trên các công việc và đối tượng có thể chia sẻ giữa các công việc đó. Việc xử lý dữ liệu trong bộ nhớ nhanh hơn 10 đến 100 lần so với network và disk.

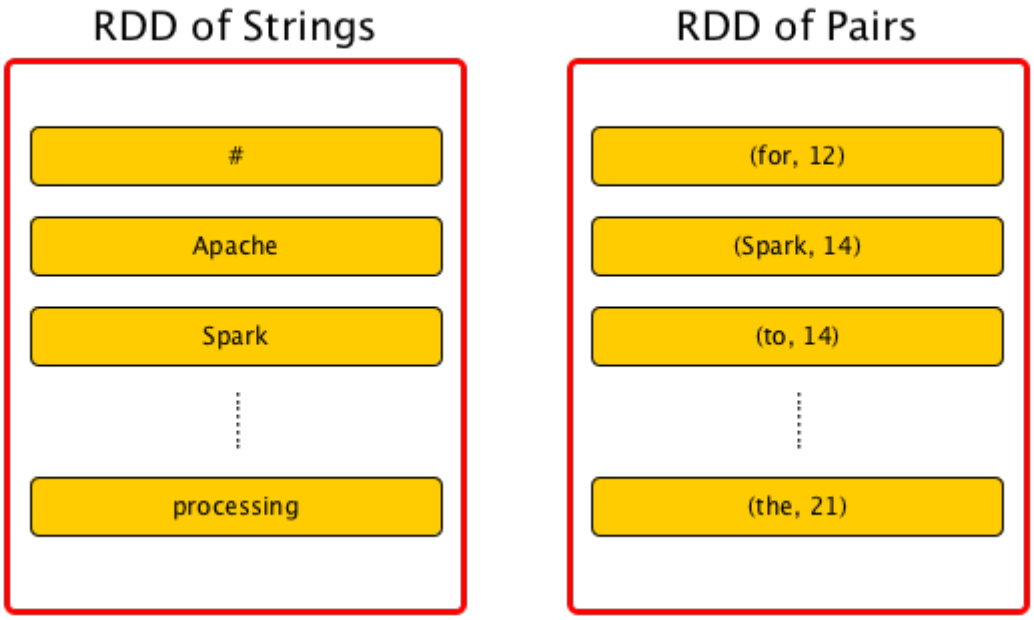
- Iterative Operation trên Spark RDD:



- Interactive Operations trên Spark RDD:



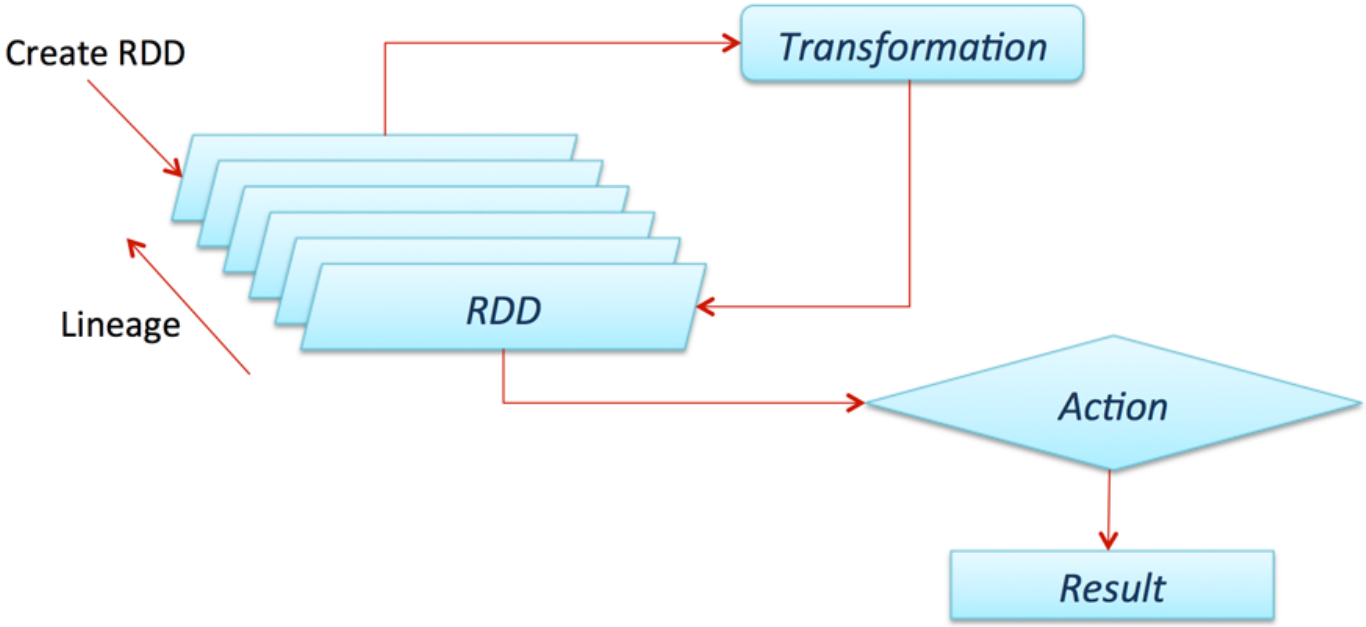
Các loại RDD



- Các RDD biểu diễn một tập hợp cố định, đã được phân vùng các record để có thể xử lý song song.
- Các record trong RDD có thể là đối tượng Java, Scale hay Python tùy lập trình viên chọn. Không giống như DataFrame, mỗi record của DataFrame phải là một dòng có cấu trúc chứa các field đã được định nghĩa sẵn.
- RDD đã từng là API chính được sử dụng trong series Spark 1.x và vẫn có thể sử dụng trong version 2.X nhưng không còn được dùng thường xuyên nữa.
- RDD API có thể được sử dụng trong Python, Scala hay Java:
  - Scala và Java: Perfomance tương đương trên hầu hết mọi phần. (Chi phí lớn nhất là khi xử lý các raw object)
  - Python: Mất một lượng performance, chủ yếu là cho việc serialization giữa tiến trình Python và JVM

Các transformation và action với RDD

RDD cung cấp các transformation và action hoạt động giống như DataFrame lẫn DataSets. Transformation xử lý các thao tác lazily và Action xử lý thao tác cần xử lý tức thời.



- Một số transformation:

Nhiều phiên bản transformation của RDD có thể hoạt động trên các Structured API, `transformation` xử lý lazily, tức là chỉ giúp dựng execution plans, dữ liệu chỉ được truy xuất thực sự khi thực hiện `action`

- **distinct**: loại bỏ trùng lắp trong RDD
- **filter**: tương đương với việc sử dụng where trong SQL – tìm các record trong RDD xem những phần tử nào thỏa điều kiện. Có thể cung cấp một hàm phức tạp sử dụng để filter các record cần thiết – Như trong Python, ta có thể sử dụng hàm lambda để truyền vào filter
- **map**: thực hiện một công việc nào đó trên toàn bộ RDD. Trong Python sử dụng lambda với từng phần tử để truyền vào map
- **flatMap**: cung cấp một hàm đơn giản hơn hàm map. Yêu cầu output của map phải là một structure có thể lặp và mở rộng được.
- **sortBy**: mô tả một hàm để trích xuất dữ liệu từ các object của RDD và thực hiện sort được từ đó.
- **randomSplit**: nhận một mảng trọng số và tạo một random seed, tách các RDD thành một mảng các RDD có số lượng chia theo trọng số.

- Một số action:

Action thực thi ngay các transformation đã được thiết lập để thu thập dữ liệu về driver để xử lý hoặc ghi dữ liệu xuống các công cụ lưu trữ.

- **reduce**: thực hiện hàm reduce trên RDD để thu về 1 giá trị duy nhất
- **count**: đếm số dòng trong RDD

- **countApprox:** phiên bản đếm xấp xỉ của count, nhưng phải cung cấp timeout vì có thể không nhận được kết quả.
- **countByValue**: đếm số giá trị của RDD
  chỉ sử dụng nếu map kết quả nhỏ vì tất cả dữ liệu sẽ được load lên memory của driver để tính toán
  chỉ nên sử dụng trong tình huống số dòng nhỏ và số lượng item khác nhau cũng nhỏ.
- **countApproxDistinct**: đếm xấp xỉ các giá trị khác nhau
- **countByValueApprox**: đếm xấp xỉ các giá trị
- **first**: lấy giá trị đầu tiên của dataset
- **max và min:** lần lượt lấy giá trị lớn nhất và nhỏ nhất của dataset
- **take và các method tương tự**: lấy một lượng giá trị từ trong RDD. take sẽ trước hết scan qua một partition và sử dụng kết quả để dự đoán số lượng partition cần phải lấy thêm để thỏa mãn số lượng lấy.
- **top và takeOrdered**: top sẽ hiệu quả hơn takeOrdered vì top lấy các giá trị đầu tiên được sắp xếp ngầm trong RDD.
- **takeSamples**: lấy một lượng giá trị ngẫu nhiên trong RDD

Một số kỹ thuật đối với RDD

- Lưu trữ file:

- Thực hiện ghi vào các file plain-text
- Có thể sử dụng các codec nén từ thư viện của Hadoop
- Lưu trữ vào các database bên ngoài yêu cầu ta phải lặp qua tất cả partition của RDD – Công việc được thực hiện ngầm trong các high-level API
- sequenceFile là một flat file chứa các cặp key-value, thường được sử dụng làm định dạng input/output của MapReduce. Spark có thể ghi các sequenceFile bằng các ghi lại các cặp key-value
- Đồng thời, Spark cũng hỗ trợ ghi nhiều định dạng file khác nhau, cho phép define các class, định dạng output, config và compression scheme của Hadoop.

- Caching: Tăng tốc xử lý bằng cache

- Caching với RDD, Dataset hay DataFrame có nguyên lý như nhau.
- Chúng ta có thể lựa chọn cache hay persist một RDD, và mặc định, chỉ xử lý dữ liệu trong bộ nhớ

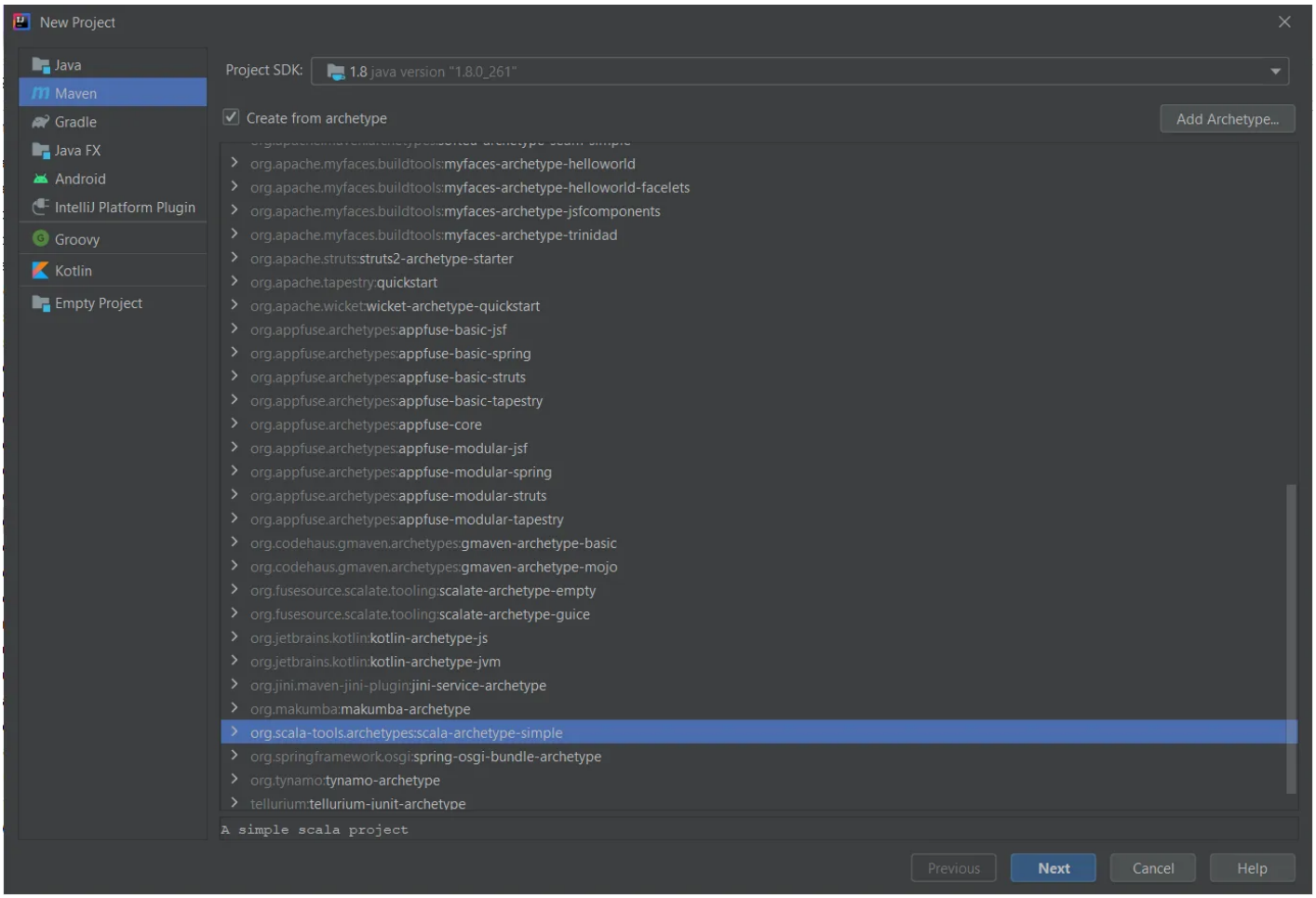
- Checkpointing: Lưu trữ lại các bước xử lý để phục hồi

- Checkpointing lưu RDD vào đĩa cứng để các tiến trình khác để thể sử dụng lại RDD point này làm partition trung gian thay vì tính toán lại RDD từ các nguồn dữ liệu gốc
- Checkpointing cũng tương tự như cache, chỉ khác nhau là lưu trữ vào đĩa cứng và không dùng được trong API của DataFrame
- Cần sử dụng nhiều để tối ưu tính toán.

Hướng dẫn tạo Spark Scala Maven project trong Intellij

1. Tạo Scala project trong Intellij

Click **New Project**

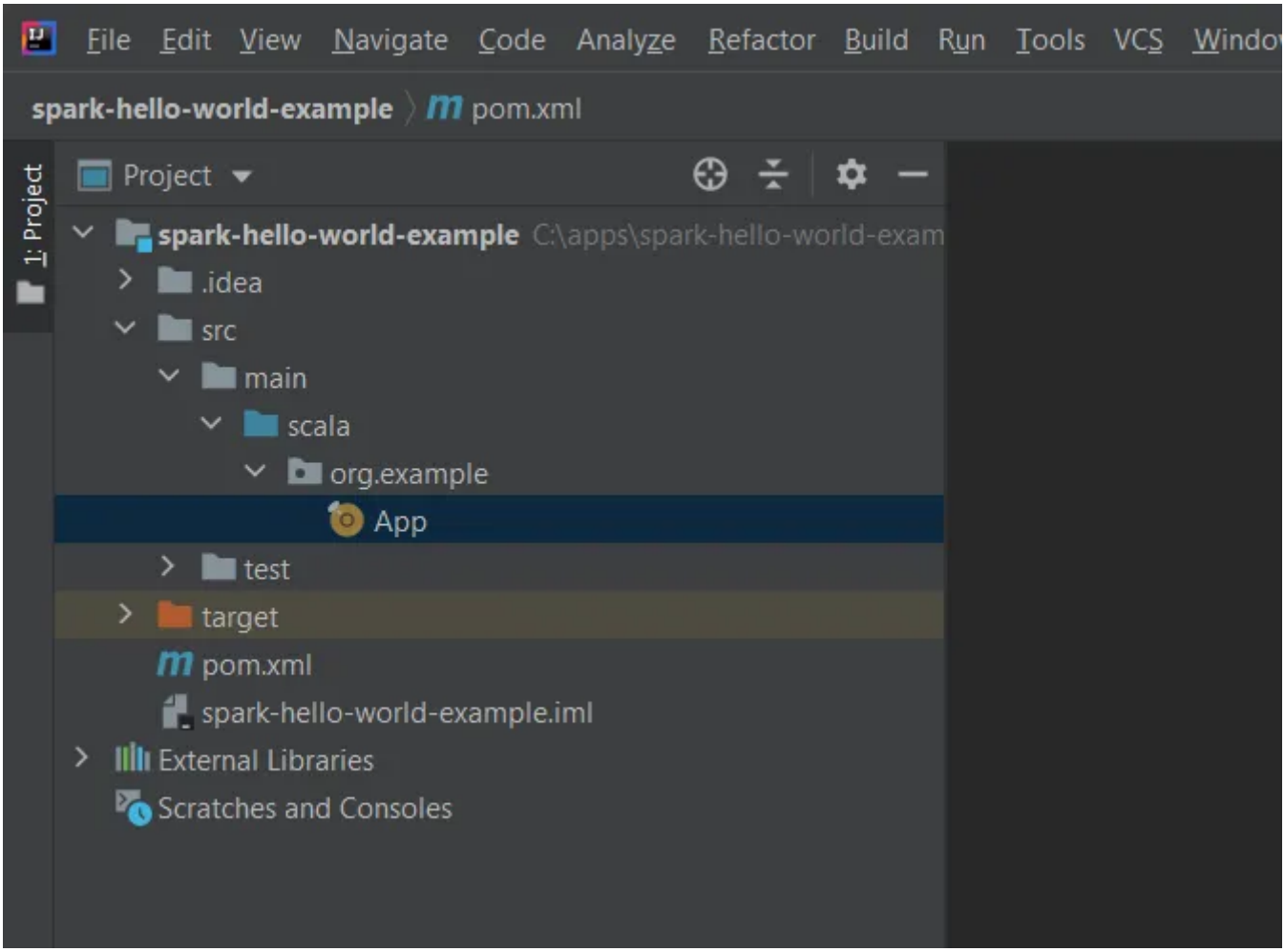


- Chọn Maven -> **Create from archetype** -> **org.scala-tools.archetypes:scala-archetypes-simple**

- Màn hình tiếp theo, nhập tên project, ví dụ: **spark-hello-world-example**

- Màn hình tiếp theo, nhập thông tin artifact-id và group-id của project

- Chọn **Finish**

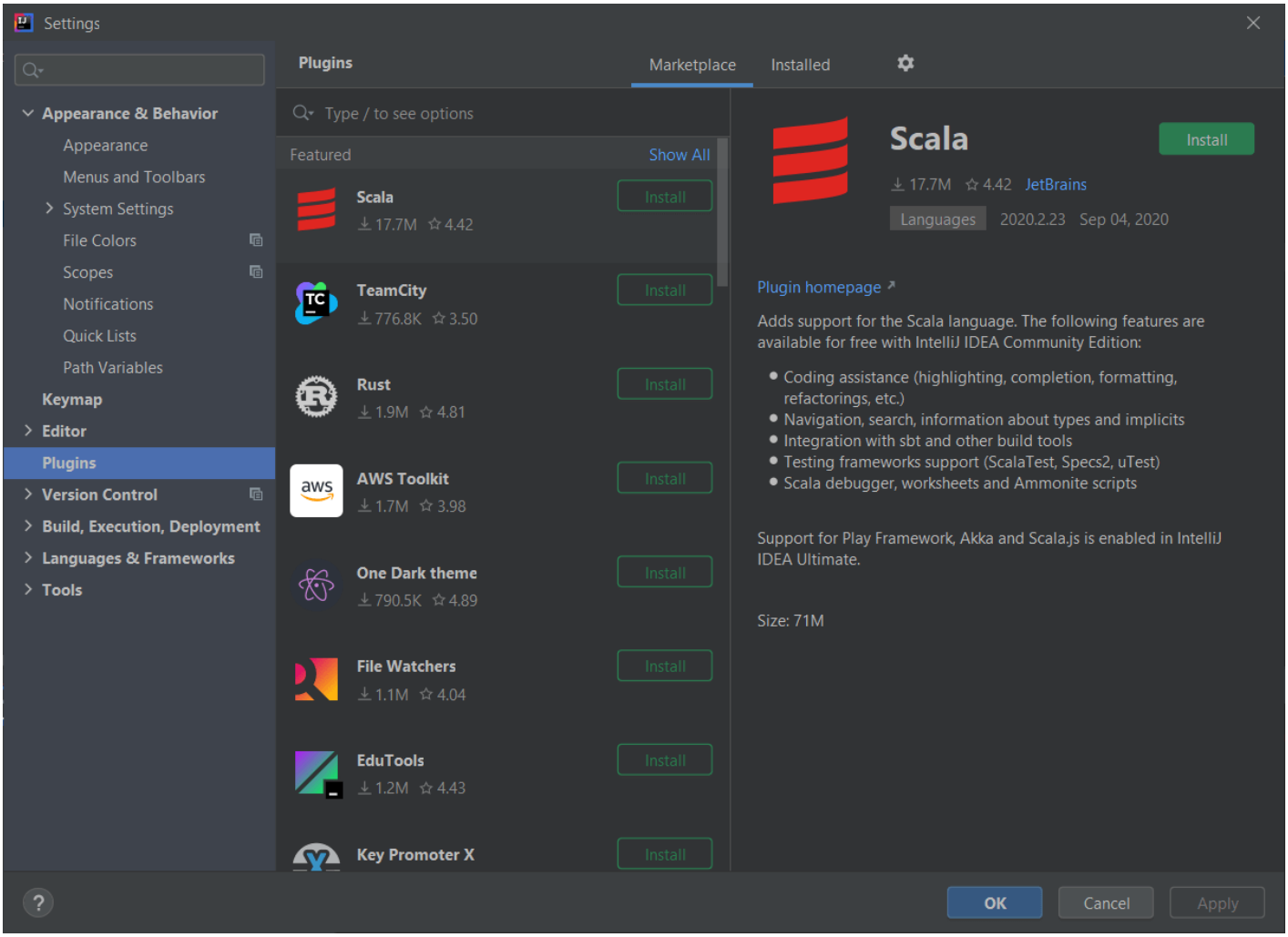


2. Cài đặt Scala Plugin

- Chọn **File** > **Settings** (hoặc Ctrl + Alt + s)

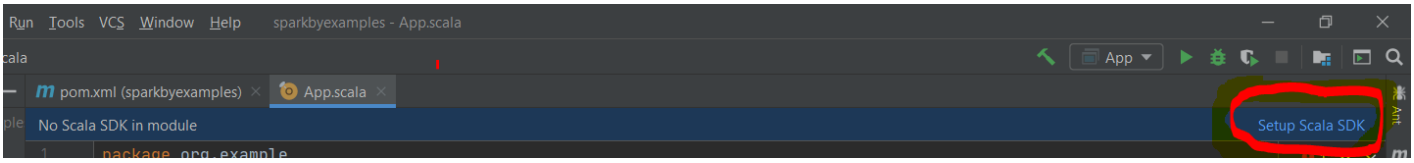
- Chọn **Plugins** từ cột bên trái.

- Chọn **Install** để cài đặt Scala plugin.



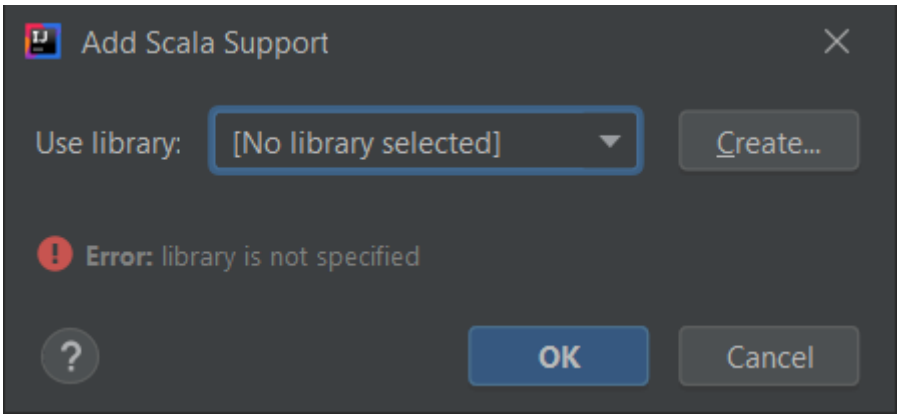
3. Cài đặt Scala SDK

- Intellij sẽ hiển thị thông báo về **Setup Scala SDK**



- Chọn **Setup Scala SDK**

- Chọn **Create**



- Màn hình tiếp theo, chọn **Download** -> Chọn **Scala version** (2.12.12 tại thời điểm bài viết)

4. Cấu hình pom.xml

- Cập nhật Scala version:

```
<properties>
    <scala.version>2.12.12</scala.version>
</properties>
```

- Thêm dependency:

```
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.12</artifactId>
      <version>3.0.0</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_2.12</artifactId>
      <version>3.0.0</version>
      <scope>compile</scope>
    </dependency>
```

- Xóa build plugin

```
      <plugin>
        <groupId>org.scala-tools</groupId>
        <artifactId>maven-scala-plugin</artifactId>
        <executions>
```

```
          <execution>
            <goals>
              <goal>compile</goal>
              <goal>testCompile</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <scalaVersion>${scala.version}</scalaVersion>
          <args>
            <arg>-target:jvm-1.5</arg>
          </args>
        </configuration>
      </plugin>
```

5. Xóa file không cần thiết

- src/test

- src/main/scala/org.example.App

6. Hello World với Spark Scala

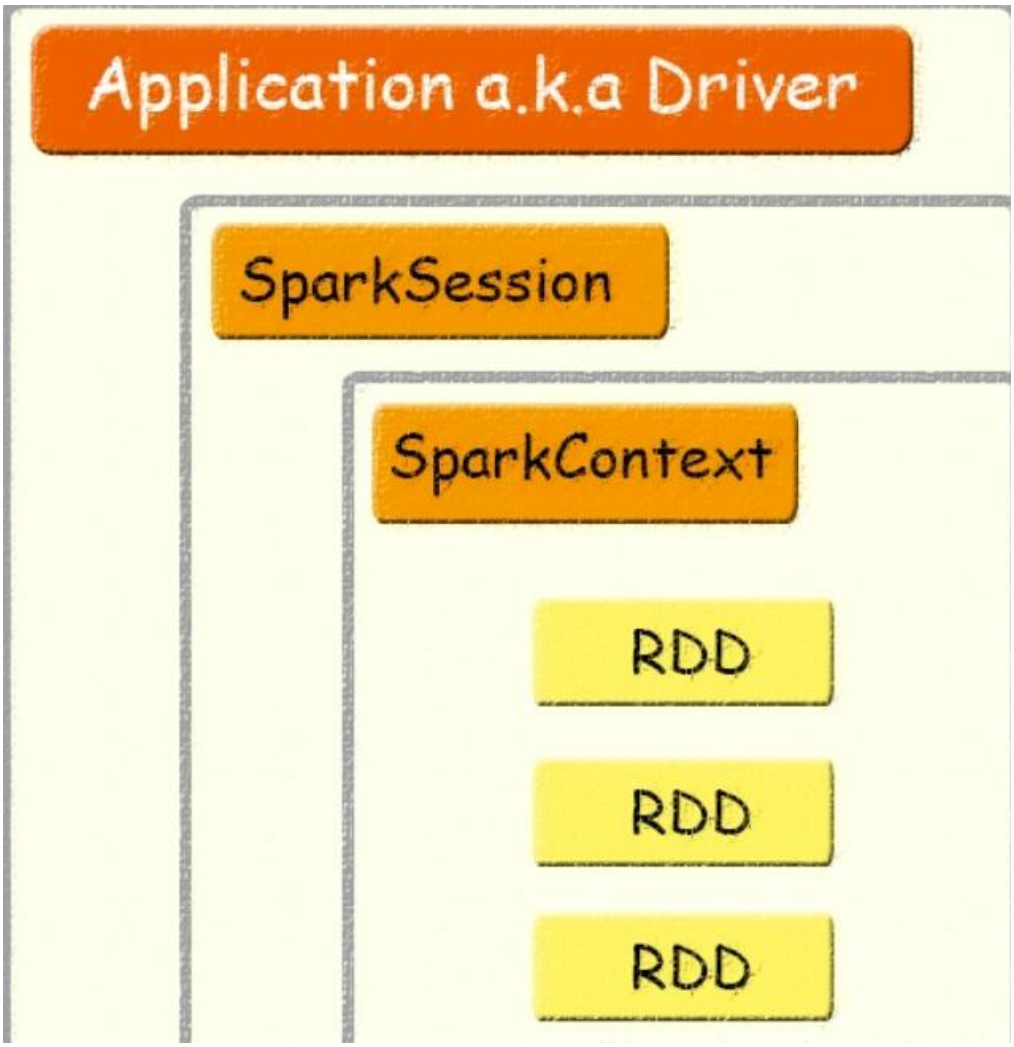
- Click chuột phải vào package của bạn, chọn **New** -> **Scala Class** -> Nhập tên và chọn loại Scala Object

```
package org.example
import org.apache.spark.sql.SparkSession
object SparkSessionTest extends App{
    val spark = SparkSession.builder()
      .master("local[1]")
      .appName("Laptrinh")
      .getOrCreate();

    println("First SparkContext:")
    println("APP Name :"+spark.sparkContext.appName);
    println("Deploy Mode :"+spark.sparkContext.deployMode);
    println("Master :"+spark.sparkContext.master);

    val sparkSession2 = SparkSession.builder()
      .master("local[1]")
      .appName("Laptrinh-VN")
```

```
      .getOrCreate();

    println("Second SparkContext:")
    println("APP Name :"+sparkSession2.sparkContext.appName);
    println("Deploy Mode :"+sparkSession2.sparkContext.deployMode);
    println("Master :"+sparkSession2.sparkContext.master);
}
```

Spark - SparkSession

Spark session: Đại diện cho khả năng tương tác với executors trong 1 chương trình. Spark session chính là entry point của mọi chương trình Spark. Từ SparkSession, có thể tạo RDD/ DataFrame/ DataSet, thực thi SQL... từ đó thực thi tính toán phân tán.



Spark Session bao gồm tất cả các API context có sẵn như:

- Spark Context
- SQL Context
- Streaming Context
- Hive Context

Khởi tạo SparkSession trong Spark-shell

Mặc định, Spark shell cung cấp một `spark` object, có thể sử dụng để khởi tạo các biến:

```
scala> val sqlcontext = spark.sqlContext
```

Khởi tạo SparkSession trong chương trình Scala

Để khởi tạo SparkSession trong Scala hoặc Python, bạn cần sử dụng phương thức builder() và getOrCreate():

```
val spark = SparkSession.builder()
      .master("local[1]")
      .appName("Laptrinh.vn")
      .getOrCreate();
```

`master()` - Nếu bạn đang chạy trên cluster, bạn cần sử dụng master name như một đối số của master(), thông thường nó là `yarn` hoặc `mesos`

- Sử dụng `local[x]` khi chạy ở chế độ standalone, x (số int > 0) - là số partition được tạo khi sử dụng RDD, DataFrame, and Dataset. Lý tưởng nhất, x là số CPU core bạn có.

`appName()` - Tên của application.

`getOrCreate()` - Return a SparkSession object nếu đã tồn tại, ngược lại sẽ tạo mới.

SparkSesssion Method

STT	Method	Mô tả
1	version	Return Spark version
2	builder	Khởi tạo new SparkSession
3	createDataFrame	Khởi tạo DataFrame từ collection và RDD
4	createDataset	Khởi tạo Dataset từ DataFrame, Collection và RDD
5	emptyDataFrame	Khởi tạo một empty DataFrame
6	emptyDataset	Khởi tạo một empty Dataset
7	getActiveSession	Return một active SparkSession
8	implicits	Truy cập một nested Scala object
9	read	Trả về một thể hiện của lớp DataFrameReader, được sử dụng để đọc các bản ghi từ csv, parquet, avro và các định dạng tệp khác vào DataFrame.

10	readStream	Trả về một thể hiện của lớp DataStreamReader, nó được sử dụng để đọc dữ liệu truyền trực tuyến, có thể được sử dụng để đọc dữ liệu truyền trực tuyến vào DataFrame.
11	sparkContext	Trả về một SparkContext.
12	sql	Trả về DataFrame sau khi thực thi SQL được đề cập.
13	sqlContext	Trả về SQLContext.
14	stop	Dừng SparkContext hiện tại.
15	table	Trả về DataFrame của một bảng hoặc dạng xem.
16	udf	Tạo một UDF Spark để sử dụng nó trên DataFrame, Dataset và SQL.