**Apache Spark 소개**

Spark 2.1.0

단 계 계

설계

작성자

홍길동

작성일

2009. .

소 속 속

단 계 계

작성자

엄만진

작성일

2017. 02. 15

**개 정 이 력**

|  |  |  |  |
| --- | --- | --- | --- |
| **버전** | **변경날짜** | **변경내용** | **작성자** |
| 1.0 | 2017.02.15 | 최초 작성 | 엄만진 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**목 차**

[1. 소개 5](#_Toc477362527)

[1.1. 개요 5](#_Toc477362528)

[1.2. Spark stack 5](#_Toc477362529)

[2. Spark 설치 6](#_Toc477362530)

[2.1. 다운로드 6](#_Toc477362531)

[2.2. 설치 6](#_Toc477362532)

[2.3. 실행 7](#_Toc477362533)

[2.4. 간단 테스트 8](#_Toc477362534)

[2.5. 추가테스트 8](#_Toc477362535)

[3. Spark 란 무엇인가 9](#_Toc477362536)

[3.1. Spark 구조 9](#_Toc477362537)

[3.1.1. Spark Driver 9](#_Toc477362538)

[3.1.2. Cluster Worker(Executor) 9](#_Toc477362539)

[3.1.3. ClusterManger 9](#_Toc477362540)

[3.2. RDD(Resilient Distirubted Datasets) 란 무엇인가 10](#_Toc477362541)

[3.3. RDD Operator 11](#_Toc477362542)

[4. Spark 예제 13](#_Toc477362543)

[4.1. 기본환경 13](#_Toc477362544)

[4.2. Transformation 13](#_Toc477362545)

[4.2.1. Map 13](#_Toc477362546)

[4.2.2. FlatMap 14](#_Toc477362547)

[4.2.3. Filter 15](#_Toc477362548)

[4.2.4. Distinct 17](#_Toc477362549)

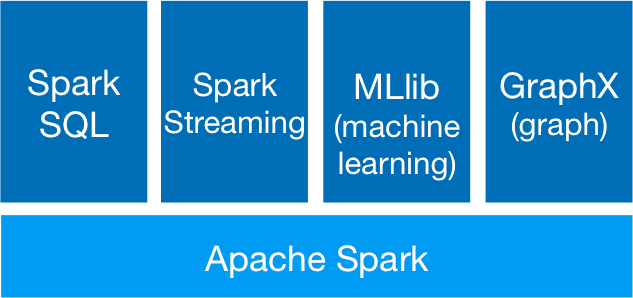
[4.2.5. Sample 17](#_Toc477362550)

[5. 참고자료 19](#_Toc477362551)

1. 소개
   1. 개요

Apache Spark은 범용의 고성능 클러스터 컴퓨팅 시스템이다. Spark은 Java, Scala 그리고 Python 으로 작성된 상위 수준의 API를 제공하고, 일반적인 실행 그래프를 지원하는 최적화된 엔진이다. 또한, SQL과 구조화된 데이터 처리를 위한 Spark SQL, 기계 학습을 위한 MLlib, 그래프 처리를 위한 GraphX 그리고 Spark Steaming 과 같은 풍부한 고급 도구들을 지원한다.

* 1. Spark stack



위 그림은 Spark의 구성을 나타낸것이다. 상황에 따라 다르겠지만 Spark 하나 만으로 기존에 필요했던 많은 기능을 통합 및 대체 할수 있다.

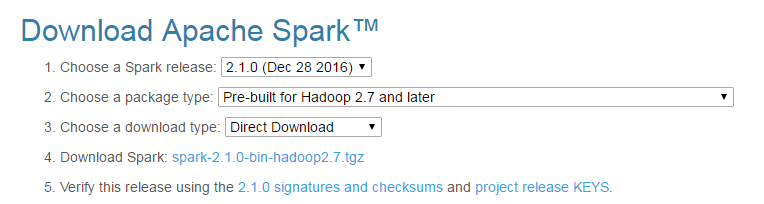
* Spark SQL : 정형 데이터 처리를 위한 Module
* Spark Streaming : 실시간 데이터 처리를 위한 Module
* MLib : 머신러닝 Module
* GraphX : 그래프 알고리즘 지원

1. Spark 설치
   1. 다운로드

다음의 주소로 접속하면 최신 버전의 Spark를 다운 받을 수 있다.

<https://spark.apache.org/downloads.html>

현재 최신 버전은 2.1.0으므로 최신 버전을 설치 했으며 설치 장비의 OS는 MINT Linux에 설치 하였다.



위 URL에 접속하면 위 이미지 처럼 Spark의 버전과 package type을 선택 할 수 있는데 기본 설정인 최신버전과 Hadoop 2.7 이상을 선택한 후 spark-2.1.0-bin-hadoop2.7.tgz를 클릭하여 다운로드 한다.

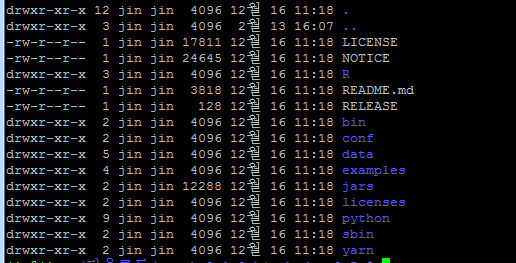
* 1. 설치

home> tar –zxvf spark-2.1.0-bin-hadoop2.7.tgz

다운받은 \*.tgz 파일을 위 명령으로 설치한다. 다운 받은 파일을 설치하고자 하는 directory로 이동한 후 명령을 실행하면 되고 여기서는 /home 밑으로 이동하였다.

/home/ 밑에 spark-2.1.0-bin-hadoop2.7 directory가 생성되면서 아래의 디렉토리 구조로 설치된다.

(원하는 directory에 설치해도 상관없다)

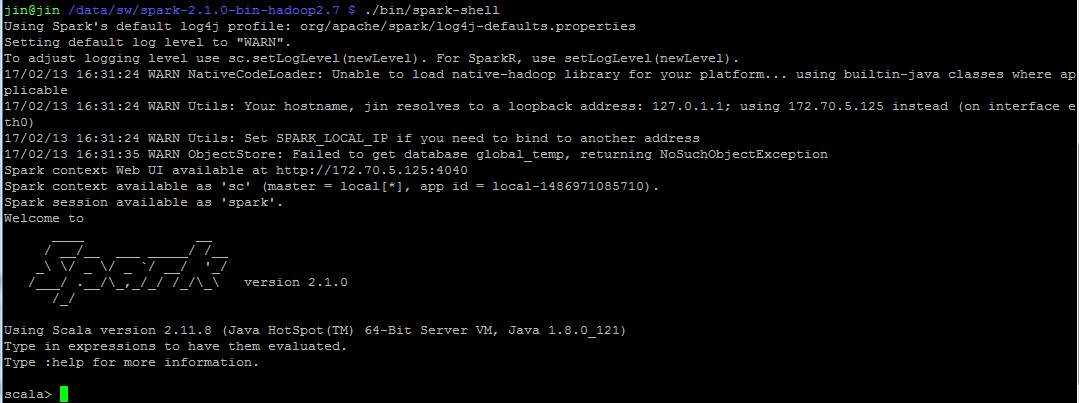


* 1. 실행

Spark를 실행하려면 ./bin 디렉토리의 spark-shell shell을 실행 하면된다.

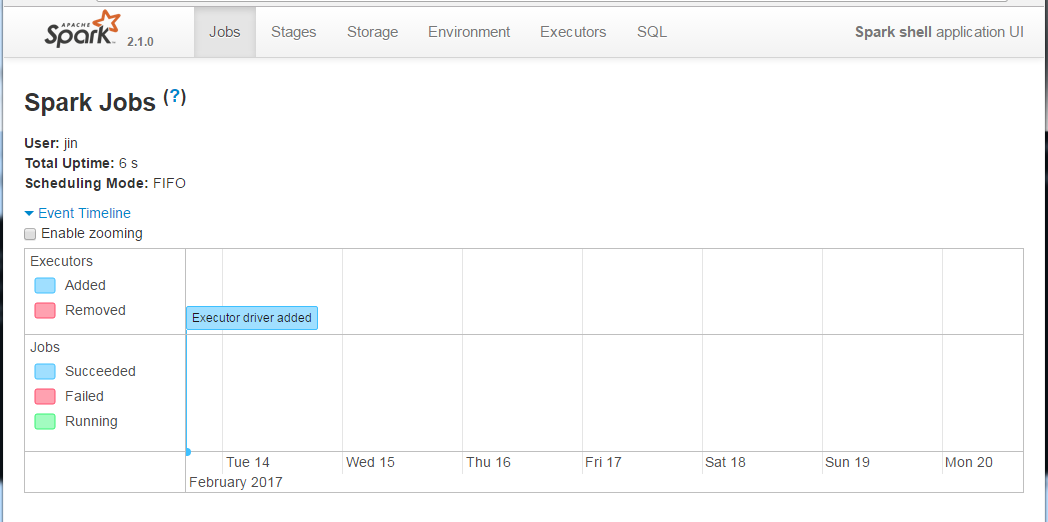
home/ spark-2.1.0-bin-hadoop2.7> ./bin/spark.shell

로그 설정이 되어 있지 않아서 default로그 설정으로, 로그 level이 WARN으로 되어 있어 WARN이상 로그만 표시된다.



위 그림처럼 표시되면 실행이 완료되었다. 실행이 완료되면 Web UI를 확인할수 있으며 접속 정보를 spark가 실행될 때 표시된다.

기본 설정은 <http://IP:4040/jobs> 이다. Web UI에 접속하면 아래와 같은 화면을 볼 수 있다.



로그와 Web UI를 확인 했으면 정상적으로 실행이 완료 되었으므로 Scala를 통해 간단하게 테스트 할 수 있다.

* 1. 간단 테스트

테스트를 위해 아래 명령을 실행하면 된다.

scala>var textFile = sc.textFile(“README.md”)

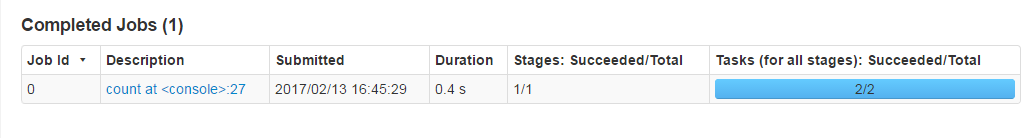
….

scala>textFile.count()

….

scala 가 interpreter 언어이므로 명령을 실행 하는 라인 단위로 상태를 표시하므로 바로 알수가 있다.

실행 후 Web UI를 보면 아래의 그림과 같이 Job이 추가된 것을 확인 할수 있다.



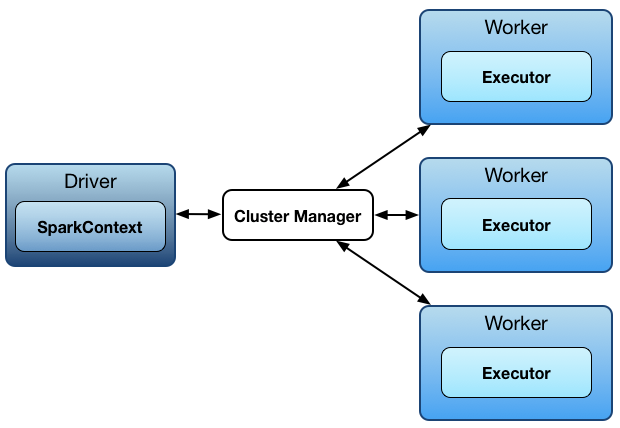
Description의 을 선택하면 상세 정보를 확인 할 수 있다.

* 1. 추가테스트

추가적으로 더 많은 기능을 테스트 하고 싶으면 아래 주소로 접속하면 word count와 caching등을 테스트 해볼 수 있다.

<https://spark.apache.org/docs/latest/quick-start.html>

1. Spark 란 무엇인가
   1. Spark 구조

간단하게 구성된 그림이다. ( 사이트에서 가장 간단하게 표시된 그림을 구한 것임)

분산 모드일 때 왼쪽의 그림과 같이 하나의 조정자(coordinator)와 여러 개의 Worker로 구성되어 있는 Master / Slave 구조로 되어 있으며 Driver(Spark Driver) 가 coordinator이고 ClusterManager를 통해 각 Worker들을 관리한다.

* + 1. Spark Driver

Driver란 SparkContext를 생성하고 File, HDFS등과 같은 저장소를 통해 RDD를 만든 후 Transformation과 Action을 기술한 사용자 코드를 정의. 전반적인 작업이 기술되어 있는 main() method를 정의한다고 생각하면 된다.

Driver의 역할은 다음과 같다

* 사용자 프로그램을 Task로 변환

1. RDD의 Operation을 통해 DAG(Directed Acyclic Graph)를 생성한다.
2. Driver가 실행될 때 Physical Plan으로 변환된다.
3. DAG를 최적화를 통해 여러 개의 Stage로 변경한다.
4. 각 Stage는 여러 개의 Task로 이루어지며 단위 작업들은 묶여서 Cluster에게 전달된다.

* Executor에서 Task들의 스케줄링

1. Executor는 시작 시 Driver에 등록하여 Driver가 Executor를 적절하게 스케줄링한다.
   * 1. Cluster Worker(Executor)

Task들를 실행하고 그 결과를 Driver에게 전달한다.

RDD가 Executor내부에 캐시 되므로 메모리 저장소를 제공한다.

* + 1. ClusterManger

Executor들을 실행하는 역할을 담당하며 Hadoop Yarn, Apache Mesos와 Stand-alone 으로 구성할수 있다.

**Spark Application 실행 단계**

1. 사용자는 spark-submit을 사용하여 애플리케이션을 제출

2. spark-submit은 Driver를 실행하고 사용자가 정의한 main() 메소드를 호출

3. Driver는 Cluster Manager에게 Executor 실행을 위한 리소스 요청

4. Cluster Manager는 Driver를 대신해 Executor들을 실행

5. Driver는 작성된 RDD의 Transformation과 Action에 기반하여 작업 내역을 단위 작업 형태로 나눠 Executor들에게 전송

6. Executor들은 단위 작업들 결과를 계산하고 저장

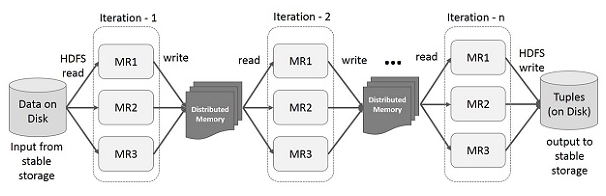
7. Driver의 main()이 끝나거나 SparkContext.stop()이 호출되면 Executor들은 중지되고

Cluster Manager에 사용했던 자원을 반환

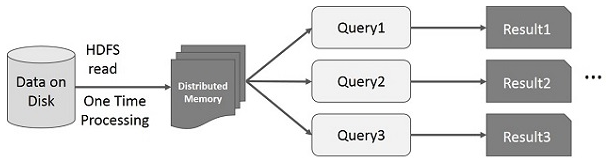
* 1. RDD(Resilient Distirubted Datasets) 란 무엇인가

HDFS의 MapReduce 방식은 MapReduce를 수행할 때 마다 HDFS에 저장하여 수행하므로 처리 속도가 느리다. 복잡하고 여러 단계 처리등 기능적으로 부족함이 있어 메모리에 정보를 저장하여 처리 속도 개선하기 위해 만들어진 변경이 불가능한 분산 객체이다.

아래 그림과 같이 MapReduce한 결과를 Memory에 보관하여 다음 MapReduce에서는 Memory에서 데이터를 읽어서 처리한다.



또한 참조 데이터를 미리 메모리에 보관하여 수행 할수 있다.



이러한 방법으로 100배 빠르게 작업을 수행 할수 있다.

특정 작업 단위로 나누어서 변경하는 find-grained update가 지원되는 RAMCloud, Piccolo등과 같이 메모리의 데이터를 update 해서 더 좋은 효과를 보려고 했었다. 하지만 메모리의 값을 변경함으로써 속도가 느려지는 현상이 발생되었다.

고장 방지(fault-tolerance)를 위해서 복사가 필요해 해지고 복사를 위해 멈추는 현상을 피할수 없었다.

이런 문제를 해결하기 위해 RDD를 Read-Only로 구성하였다. 이런 이유로 Spark에서 생성한 RDD를 수정할수 없다. 가공이 필요할 경우 Transformation 과 Actions를 이용하여 새로운 RDD를 생성하면 된다.

**Coarse-grained 와 find-grained**

자바 웹서비스 PDF 책의 저자에 따르면 grain은 원래 보리나 밀 같은 곡식을 낟알로 만드는 작업이나 표면을 우둘투둘하게 하는 일을 뜻 하는데, 그때아주 곱고 섬세하게 하느냐, 아니면 듬성 듬성 크게 하느냐에 따라 fine와 coarse라 는형용사를 붙인다. 이것이 소프트웨어공학에 도입되어 어떠한 프로세스를 아주 잘게 쪼개느냐 아니면 굵게 쪼개서 뭉뚱그려 놓느냐를 표현할 때 쓰게 되었다. 웹서비스에서의서비스라는 단위에 비해 상대적으로 자바의 메소드는 더 세분화되어있다는 뜻에서 코스그레인드 서비스-파인그레인드 메소드라는 대칭이 생겼다고 봐도 좋다.

Coarse는 큰 작업 단위로 이루어지는 것을 말하고 find은 세부적인 작업 단위로 나누는 것을 말하는 것같다.(google에서 검색해보는게 좋을 듯..ㅎㅎ)

* 1. RDD Operator

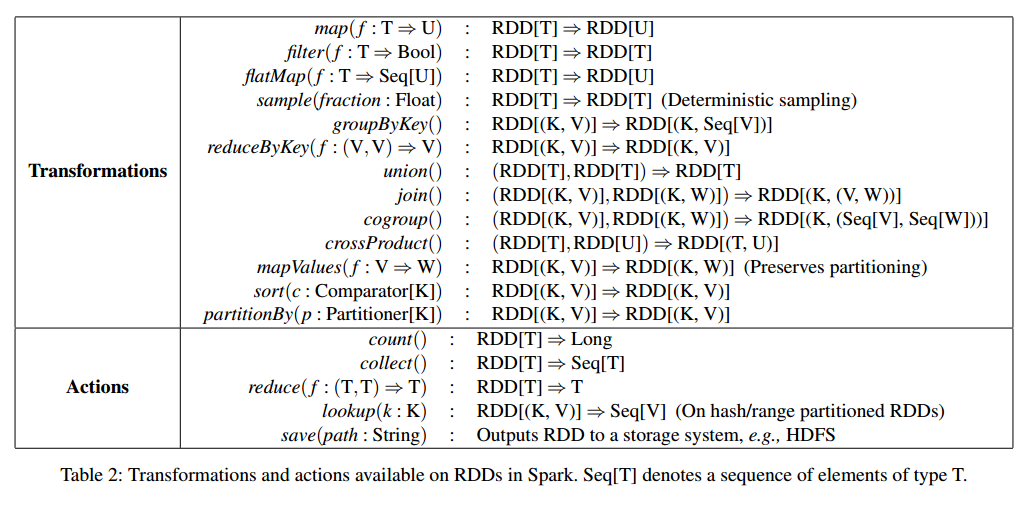
RDD의 Operator는 크게 두개로 구성되어 있다. Transformation와 Action이다.

Transformation은 map, reduce, join 등과 같이 데이터를 가공하는 Operator이고 Action은 최종 결과를 얻기 위한 Operator이다.

고딩을 할때는 Transforamtion은 여러 개를 나열할수 있으나 Action은 한번만 기술한다.

(transformation -> transformation -> … -> transformation -> Action 이런 형식)

Transformation과 Action의 종류는 아래 그림과 같다.



각 항목에 대한 설명은 Spark homepage에서 확인 가능하다.

Transformation : <https://spark.apache.org/docs/latest/programming-guide.html#transformations>

Action : <https://spark.apache.org/docs/latest/programming-guide.html#actions>

여기서 중요한 점은 실행 방식인데 Transformation후 Action까지의 과정을 lazy-execution이라고 한다. Transformation에서 RDD가 변경이 발생되는 것이 아니고 변경 이력만 생성한다(DAG - Directed Acyclic Graph를 저장 – Logical graph of RDD operations). DAG를 저장함으로써 cost가 저렴하고 fault\_tolerant의 문제가 발생 시 이전에 생성한 RDD를 보고 빠르게 복구가 가능하다.

실제 Action에서 Transformation에 의해 생성된 DAG를 바탕으로 한번에 수행된다.

1. Spark 예제
   1. 기본환경

eclipse에서 Spark 이용한 간단한 예제를 만드는 방법에 대해 작성한다. Maven Project 기준으로 작성하며 Maven에 대해서는 따로 설명하지 않는다.

소스는 <https://github.com/eum123/spark-study.git> 에서 확인 할수 있다.

Eclipse Project에 Spark core library dependency를 추가한다.

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.10</artifactId>

<version>2.1.0</version>

</dependency>

Version은 최신 version을 사용하면 된다.

위 설정을 추가하면 Maven에서 필요한 library를 repository에서 다운로드 받아서 eclipse build환경에 추가한다.

JDK는 1.8을 사용하였다.

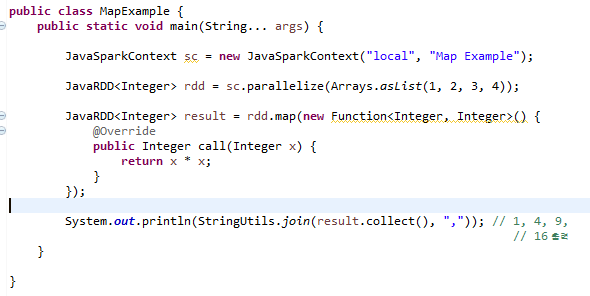
주의. 아래 예제는 로컬 환경에서 동작하게 구성된것이라 spark-submit할때는 동작하지 않을수 있다. Spark-submit할 경우 JavaSparkContext를 생성할 때 아래와 같은 방식으로 생성하면 된다.

SparkConf conf = **new** SparkConf().setAppName("Simple Application");

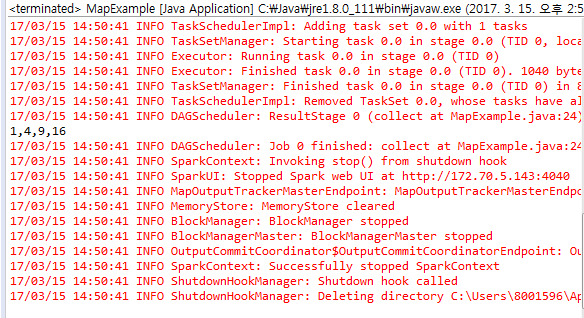
conf.setMaster("spark://172.70.5.125:4040");

* 1. Transformation
     1. Map

RDD의 각 요소에 함수를 적용하여 결과 RDD반환하는 방식.

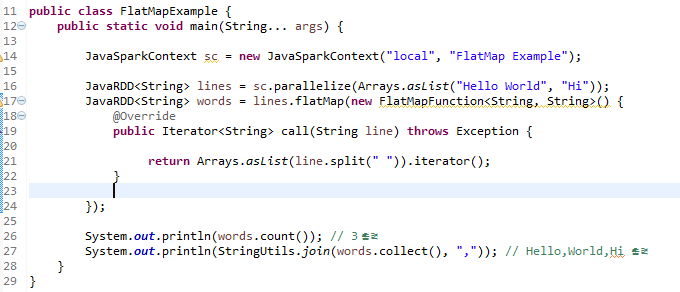


[1,2,3,4]로 구성된 RDD를 call() method을 통해 각 항목의 값을 연산하여 반환하는 예제이다. 이 예제를 실행하면 아래와 같은 결과를 볼수 있다.



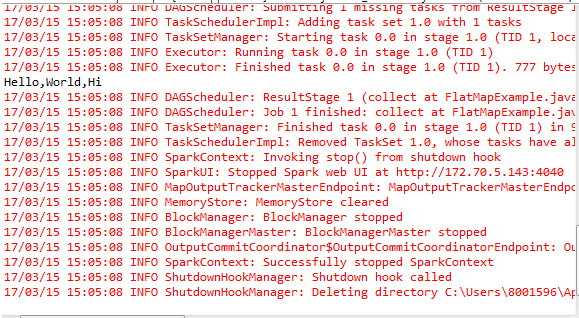
* + 1. FlatMap

Map과 동일하게 RDD의 요소에 함수를 적용하는 방식이나 call() method의 반환값이 Iterator(반복자)라는 부분이 차이가 있다. 주로 단어 분해에 사용된다.



“Hello World” 와 “Hi” 문장들을 “ “ 구분자로 단어를 분리하는 예제이다.

실행하면 아래와 같은 결과를 확인 할수 있다.



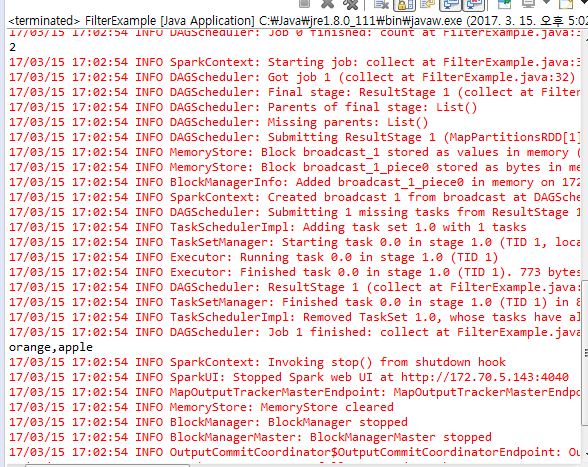
* + 1. Filter

RDD의 요소 중 원하지 않는 요소를 제외하는 기능.



“orange”, “apple”, “book” 단어로 구성되어 있는 RDD에서 “book”을 제거하는 예제이다.

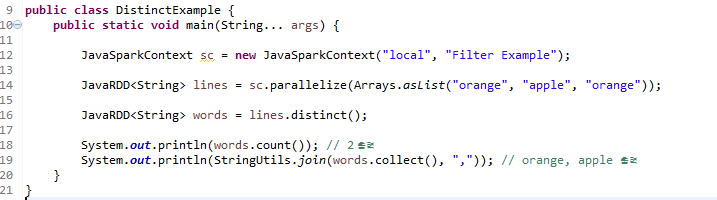
실행 결과를 아래와 같다.



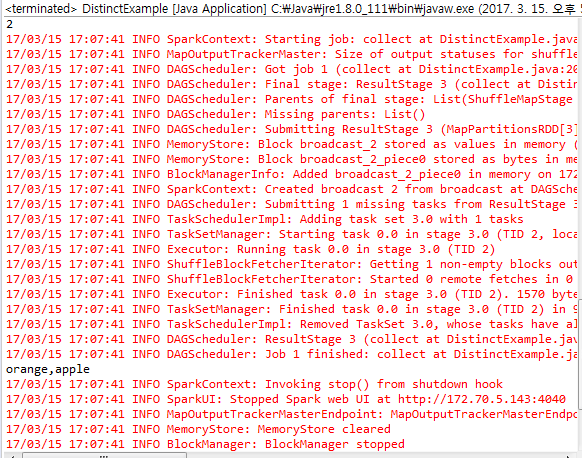
* + 1. Distinct

RDD요소에서 중복을 제거하는 기능.

Distinct는 셔플링(shuffling) 발생으로 속도 저하될수 있으니 유의 해야 된다.



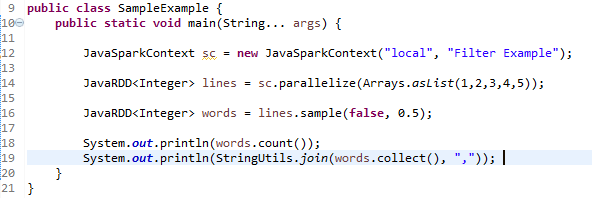
실행 결과는 아래와 같다.



* + 1. Sample

RDD에서 표본을 추출하는 기능으로 sample() method의 첫번째 argument 값에 따라 복원추출(true), 비복원추출(false) 방법을 사용한다.

두번째 argument는 추출 비율(분수)을 입력받는다. 1이면 최대 전체 개수만큼 추출된다.



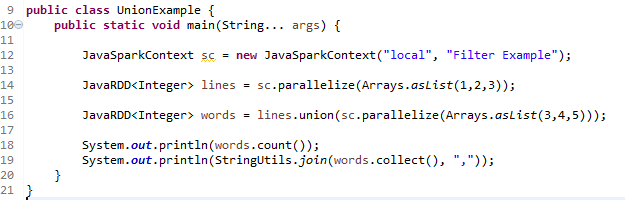
실행 결과는 임의의 값을 추출하여 표시된다.

**복원추출과 비복원추출**

복원 추출은 한번 시행한 사상의의 결과를 다시 얻을수 있도록 모집단에 다시 포함시켜 추출하는 방법이고 비복원 추출은 한번 시행한 사상은 다시 모집단에 포함시키지 않는 추출 방법이다.

* + 1. Union

현재 RDD와 입력한 새로운 RDD를 합쳐서 새로운 RDD를 만든다.

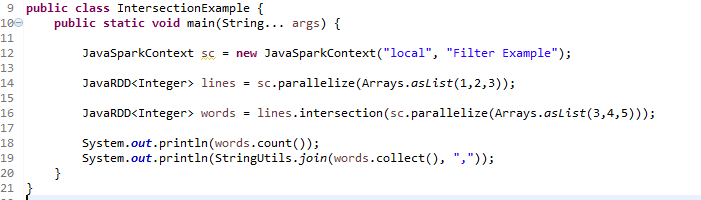


실행 결과는 1,2,3,3,4,5 가 출력된다.

* + 1. Intersection

두 RDD에서 공통으로 갖고 있는 요소만으로 구성된 RDD를 만든다.

Intersection는 셔플링(shuffling) 발생으로 속도 저하될수 있으니 유의 해야 된다.

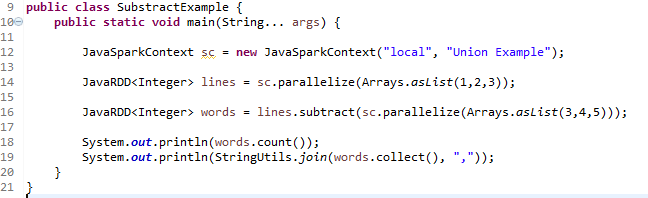


실행 결과는 3이 출력된다.

* + 1. Substract

현재 RDD에서 입력한 RDD의 요소를 제거한 새로운 RDD를 생성한다.

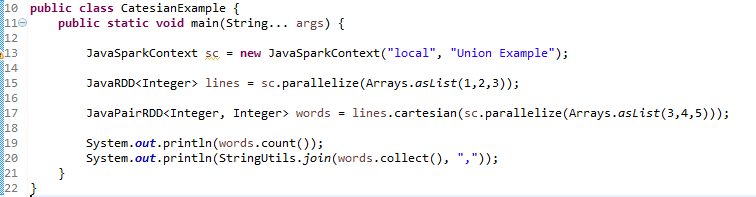
Substract는 셔플링(shuffling) 발생으로 속도 저하될수 있으니 유의 해야 된다.



실행 결과는 1,2,3에서 3,4,5를 제외한 1,2만 출력된다.

* + 1. Cartesian

Cartesian은 셔플링(shuffling) 발생으로 속도 저하될수 있으니 유의 해야 된다.



실행 결과는 (1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,3),(3,4),(3,5) 이렇게 출력된다.

**곱집합(Cartesian Product)**

두개 이상의 집합을 합쳐서 하나의 집합으로 만드는 것으로 구성 원소가 같더라도 순서가 다르면 다른 원소로 구성하는 방식.

A집합이 3개이고 B 집합이 4이면 총 12개로(3 X 4) 구성된 집합이 만들어 진다.

http://blog.naver.com/PostView.nhn?blogId=woney48&logNo=220906523467

1. 참고자료

<http://kireen.com/220738828944>

<https://spark.apache.org>

<https://www.slideshare.net/yongho/rdd-paper-review>

<http://godblessyk.tistory.com/entry/sparkSpark-Architecture>

<http://ourcstory.tistory.com/139>

<http://bcho.tistory.com/1026>