



하둡 wordcount 실습



필요한 것!

1. Docker Desktop
2. git
3. jdk(환경변수 설정까지)

0. 환경 설정

```
git clone https://github.com/big-data-europe/docker-hadoop-spark-workbench.git
```

Hadoop, Hive, Spark 를 띄울 수 있는 레포를 클론해옵니다

Hive? 하이브는 하둡 에코시스템 중에서 데이터를 모델링하고 프로세싱하는 경우 가장 많이 사용하는 **데이터 웨어하우징용 솔루션**입니다

그리고 클론해온 레포로 이동해줍니다

```
cd docker-hadoop-spark-workbench
```

```
# 1. namenode
docker-compose -f docker-compose-hive.yml up -d namenode

# 2. datanode
docker-compose -f docker-compose-hive.yml up -d datanode
```

datanode와 namenode를 띄웁니다

```
docker ps
```


잘 떠있는지 확인

```
# 컨테이너 접속
docker exec -it namenode /bin/bash

# hadoop 명령어 잘 되나 확인
hadoop fs -ls /
```

1.1 Volume 확인

클론해온 레포에서 docker-compose.yml 파일 확인

 docker-compose.yml - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
version: '2'
services:
  namenode:
    image: bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8
    container_name: namenode
    volumes:
      - ./data/namenode:/hadoop/dfs/name
    environment:
      - CLUSTER_NAME=test
    env_file:
      - ./hadoop.env
    ports:
      - 50070:50070
  datanode:
    image: bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8
    depends_on:
      - namenode
    volumes:
      - ./data/datanode:/hadoop/dfs/data
    env_file:
      - ./hadoop.env
    ports:
```

datanode volumes 부분을 확인 datanode가 어디에 있는지 확인

datanode volumes: Docker에서 Volume을 간단하게 설명하면, 내 컴퓨터(로컬)의 특정 경로와 컨테이너 안에서의 특정 경로를 연동시켜 놓은 것

(:을 기준으로 [내 컴퓨터의 경로] : [컨테이너 안에서의 경로])

1.2 Text 파일 준비

```
# 경로 이동
cd ../data/datanode

# 단어를 셀 파일 아무거나 작성하고 :wq로 저장
vi words.txt

Deer Bear River
Car Car River
Deer Car Bear
```

2. WordCount.java 작성

```
vi WordCount.java
```

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
}
```

```

    }
    }
}

public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                        Context context
                        ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

▼ 주석 있는 버전

```

import java.io.IOException; // 예외처리
import java.util.StringTokenizer; //스트링 토큰 처리기

import org.apache.hadoop.conf.Configuration; // 하둡 구성 정보
import org.apache.hadoop.fs.Path; // 파일 시스템 경로
import org.apache.hadoop.io.IntWritable; // 정수형 쓰기 가능 데이터 삽입
import org.apache.hadoop.io.Text; //텍스트 데이터타입
import org.apache.hadoop.mapreduce.Job; //맵리듀스 Job
import org.apache.hadoop.mapreduce.Mapper; // 맵리듀스 매퍼
import org.apache.hadoop.mapreduce.Reducer; //맵리듀스 리듀서
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; //파일 입력 포맷
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; //파일 입력 출력

public class WordCount {

```

```

public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{
    // <inputkey, inputvalue, outputkey, outputvalue>
    private final static IntWritable one = new IntWritable(1);
    // IntWritable : 하둡 시스템에서 만들어 놓은 특별한 타입, 병렬을 직렬화
    private Text word = new Text(); //한줄씩 들어옴

    // mapper 부분
    public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
        // 각 라인을 읽어들이어서 토큰화 한다.
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken()); // 단어를 하나씩 추출
            context.write(word, one); // (word, 1)의 쌍으로 출력
        }
    }

    // reducer 부분
    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
        ) throws IOException, InterruptedException {
            // 정렬이 끝난 상태에서의 리듀스 작업을 진행한다.
            // 각각의 키에 대한 value값들이 하나로 합쳐지고 쓰여진다.
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get(); // value의 값을 얻어서 합계
            }
            result.set(sum);
            context.write(key, result); //(ket, result)의 쌍으로 출력
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "word count");
        // word count라는 이름을 가진 job을 만든다.
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        // mapper 와 reducer 사이에 combiner 함수가 실행된다.
        // combiner 함수는 중복된 키들의 value 값을 하나로 모아준다.
        // 이후에는 셔플, 파티셔닝, 정렬과정을 거쳐 reducer로 넘어간다.
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0])); // 입력 파일 지정
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // 목적지 파일 지정

        System.exit(job.waitForCompletion(true) ? 0 : 1); // 매퍼듀스 job제출 기다림
    }
}

```

```
}  
}
```

3.1 컨테이너 접속

```
# 컨테이너 내부 접속  
docker exec -it docker-hadoop-spark-workbench_datanode_1 /bin/bash  
  
# 경로 이동  
cd /hadoop/dfs/data
```

```
C:\Users\SAMSUNG#docker-hadoop-spark-workbench#data#datanode>docker exec -it docker-hadoop-spark-workbench_datanode_1 /bin/bash  
root@ed540467f5e2:/# cd /hadoop/dfs/data  
root@ed540467f5e2:/hadoop/dfs/data# ls  
WordCount.java WordCount.java~ current in_use.lock words.txt words.txt~  
wc-1540467f5e2 /hadoop/dfs/data#
```

3.2 환경변수 설정

```
# 환경변수 설정  
# JAVA_HOME은 되어있는 것을 확인하고, 나머지를 설정해준다.  
echo $JAVA_HOME  
# /usr/lib/jvm/java-8-openjdk-amd64 가 출력됨  
  
export PATH=${JAVA_HOME}/bin:${PATH}  
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

```
root@ed540467f5e2:/hadoop/dfs/data# echo $JAVA_HOME  
/usr/lib/jvm/java-8-openjdk-amd64  
root@ed540467f5e2:/hadoop/dfs/data# export PATH=${JAVA_HOME}/bin:${PATH}  
root@ed540467f5e2:/hadoop/dfs/data# export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

3.3 컴파일 및 jar 파일 생성

```
hadoop com.sun.tools.javac.Main WordCount.java  
jar cf wc.jar WordCount*.class
```

```
root@ed540467f5e2:/hadoop/dfs/data# export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar  
root@ed540467f5e2:/hadoop/dfs/data# hadoop com.sun.tools.javac.Main WordCount.java
```

4.1 HDFS에 파일 올리기

```
# HDFS에 올리기  
hadoop fs -put ./words.txt /
```

```
# 확인
hadoop fs -ls /
```

```
root@ed540467f5e2:/hadoop/dfs/data# hadoop fs -ls /
Found 1 items
-rw-r--r--  3 root supergroup          51 2022-08-23 16:57 /words.txt
```

4.2 jar 파일 실행

```
hadoop jar wc.jar WordCount /words.txt /output
```

```
root@ed540467f5e2:/hadoop/dfs/data# hadoop jar wc.jar WordCount /words.txt /output
22/08/23 17:00:42 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
22/08/23 17:00:42 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
22/08/23 17:00:42 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement
t the Tool interface and execute your application with ToolRunner to remedy this.
22/08/23 17:00:42 INFO input.FileInputFormat: Total input files to process : 1
22/08/23 17:00:42 INFO mapreduce.JobSubmitter: number of splits:1
22/08/23 17:00:42 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local906533528_0001
22/08/23 17:00:43 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
22/08/23 17:00:43 INFO mapreduce.Job: Running job: job_local906533528_0001
22/08/23 17:00:43 INFO mapred.LocalJobRunner: OutputCommitter set in config null
22/08/23 17:00:43 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
22/08/23 17:00:43 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under out
put directory:false, ignore cleanup failures: false
22/08/23 17:00:43 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutp
utCommitter
22/08/23 17:00:43 INFO mapred.LocalJobRunner: Waiting for map tasks
22/08/23 17:00:43 INFO mapred.LocalJobRunner: Starting task: attempt_local906533528_0001_m_000000_0
22/08/23 17:00:43 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
```

4.3 확인

```
hadoop fs -ls /output
hadoop fs -cat /output/part-r-00000
```

```
root@ed540467f5e2:/hadoop/dfs/data# hadoop fs -ls /output
Found 2 items
-rw-r--r--  3 root supergroup          0 2022-08-23 17:00 /output/_SUCCESS
-rw-r--r--  3 root supergroup        46 2022-08-23 17:00 /output/part-r-00000
root@ed540467f5e2:/hadoop/dfs/data# hadoop fs -cat /output/part-r-00000
a      1
b      1
best   2
boaz   3
cc      1
d      1
e      1
is     2
num    1
```

Reference

[Hadoop] Docker 기반 Hadoop 실행 환경 설치/구축

bde(big-data-europe)라는 아주 유명한 곳에서 관리하는 Docker Image와 dockek compose 파일을 통해 매우 간단하게 Hadoop 실행환경을 구축할 수 있다. 오늘 다뤄볼 내용은 Docker 지식이 없어도

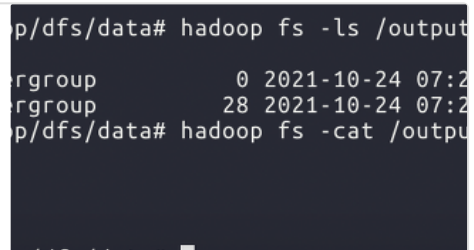
🔗 <https://so-easy-coding.tistory.com/21>



[Hadoop] MapReduce 프로그래밍 실습 예제 - Word Count

지난 글 에서 Docker기반으로 아주 빠르고 간단하게 Hadoop 실습 환경을 구축해봤다. 이번 글에서는 Hadoop의 핵심인 MapReduce 공식 튜토리얼을 따라 해 보면서, MapReduce 시작의 정석인 WordCount

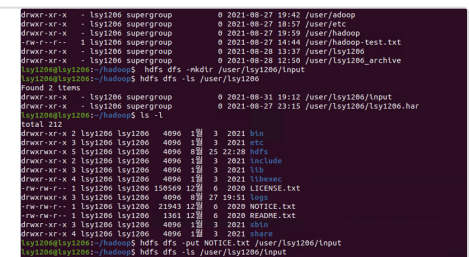
🔗 <https://so-easy-coding.tistory.com/15>



[Hadoop/기록] 7. Hadoop map-reduce (WordCount / Java)

map-reduce 예제 실행과정] 1. Hadoop home을 기준으로, 데이터를 넣을 input 폴더를 생성하고, 폴더안에 리눅스 시스템에 있던 아무 텍스트파일이나 집어넣는다. 예제는 wordcount이므로, 나는 NOTICE.txt

🔗 <https://buildabetterworld.tistory.com/166>



Hadoop WordCount 소스 코드 레벨에서 살펴보기

코드출처: <https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html> import java.io.IOException; // 예외

🔗 <https://exmemory.tistory.com/63>

