Ⅲ. 맵리듀스를 이용한 빅데이터 분석

- Author: JinKyoung Heo
- Issue Date: 2014.07.31
- Revision #: Rev 29
- Homepage: www.javaspecialist.co.kr
- Email: hjk7902@gmail.com

이 페이지는 여백 페이지입니다.

1 Hadoop 맵리듀스

Objectives

- 맵과 리듀스를 이용하여 대용량 데이터를 분산 시스템에서 분석하는 방법을 배웁니다.

MapReduce

MapReduce 용어

MapReduce 동작 과정

WordCount

HDFS 접근

Mapper

Reducer

DataType class

InputFormat/OutputFormat

이 페이지는 여백 페이지입니다.

MapReduce 용어

- ◈ Job
  - ➢ Mapper와 Reducer를 실행하는 전체 프로그램
- ◈ Task
  - ➢ 1개의 데이터 조각을 처리하는 1개의 Mapper 또는 Reducer의 실행
- ◈ Task attempt
  - ➢ 머신 위에서 1개의 태스크를 실행하는 특정 시도
- ◈ Map
  - ➢ 어떤 데이터의 집합을 받아들여 데이터를 생성하는 프로세스
- ◈ Reduce
  - ➢ Map에 의해서 만들어진 데이터를 모아서 최종적으로 원하는 결과로 만들어 내는 프로세스

• 작업(Job)
데이터 집합을 이용하여 Mapper와 Reducer를 실행하는 "전체 프로그램"입니다.
20개의 파일로부터 "Word Count"를 실행하는 것은 1개의 작업(Job)입니다.

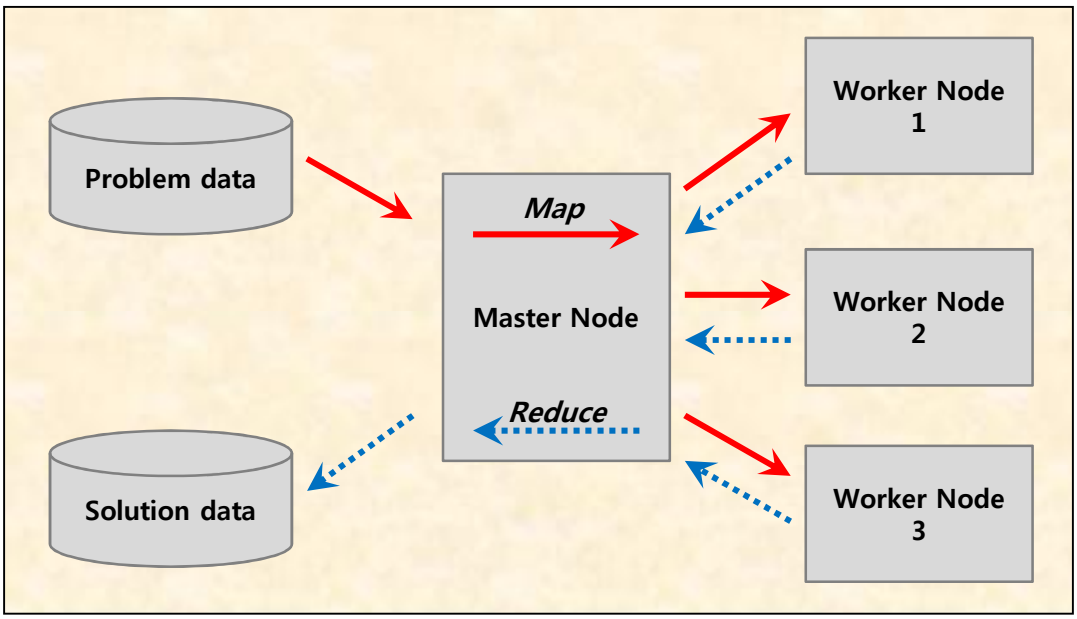
• 태스크(Task)
1개의 데이터 조각을 처리하는 1개의 Mapper 또는 Reducer의 실행입니다.
20개의 파일은 20개의 Map 태스크에 의해 처리됩니다.

• 태스크 시도(Task Attempt)
머신 위에서 1개의 태스크를 실행하는 특정 시도입니다.
최소한 20개의 Map 태스크 시도들이 수행됩니다. 서버 장애 시에는 더 많은 시도들이 수행됩니다.

• Map
어떤 데이터의 집합을 받아들여 데이터를 생성하는 프로세스입니다.

• Reduce
Map에 의해서 만들어진 데이터를 모아서 최종적으로 원하는 결과로 만들어 내는 프로세스입니다.

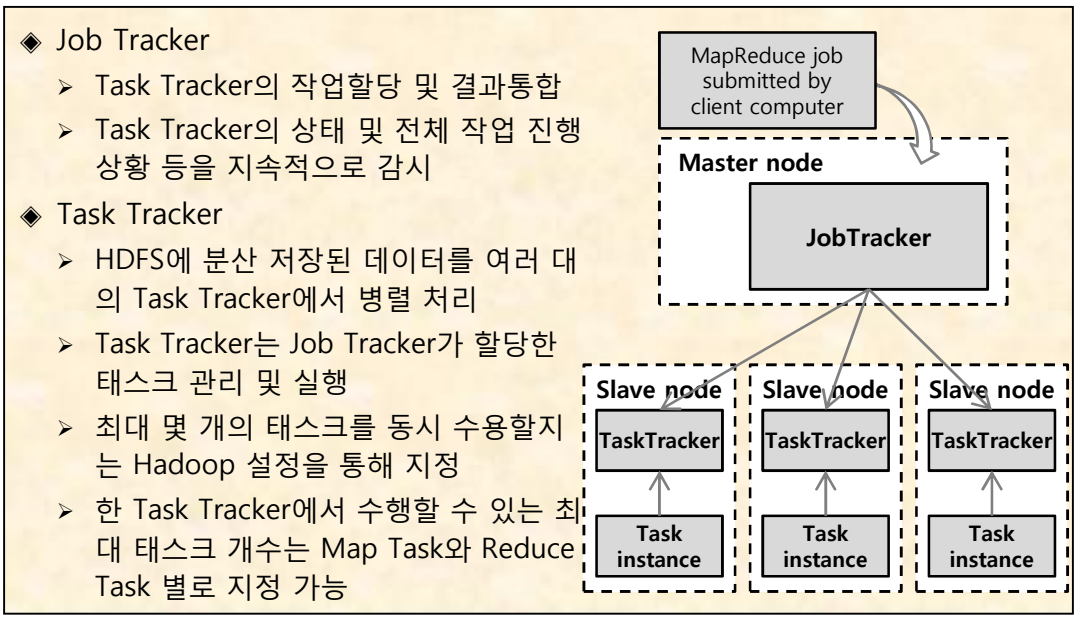
MapReduce 흐름



맵리듀스 특징

- 데이터 처리를 위한 프로그래밍 모델
- 분산컴퓨팅에 적합한 함수형 프로그래밍
- 배치형 데이터 처리 시스템
- 자동화된 병렬처리 및 분산처리
- Fault-tolerance(내고장성, 결함허용)
- 상태 및 모니터링 툴들
- 프로그래머를 위한 추상클래스

MapReduce 용어



- 마스터(Job Tracker)

  마스터 노드에서 Job Tracker는 시작합니다. Job Tracker 는 클라이언트로부터 작업요청을 받으면 실행합니다.

  Task Tracker의 작업 할당 및 결과를 통합합니다.

  Task Tracker의 상태 및 전체 작업 진행 상황 등을 지속적으로 감시합니다.

- 슬레이브(Task Tracker)

  Task Tracker는 슬레이브 노드에서 실행됩니다.

  Task Tracker는 태스크를 실행할 자바 프로세스를 발생 시킵니다.

  HDFS에 분산 저장된 데이터를 여러 대의 TaskTracker에서 병렬 처리합니다.

  TaskTracker는 JobTracker가 할당한 태스크 관리 및 실행을 수행합니다.

  최대 몇 개의 태스크를 동시 수용할지는 사용자가 Hadoop 설정을 통해 지정할 수 있습니다.

  한 TaskTracker에서 수행할 수 있는 최대 태스크 개수는 Map Task와 Reduce Task 별로 지정이 가능합니다.

MapReduce의 역할

◈ Map
- Map에는 키가 되는 데이터와 그에 대응하는 값 2가지 존재
- Map 안에서는 주어진 키를 사용하여 새로운 키와 값을 원하는 만큼 생성
- Map에 의해서 만들어진 새로운 키 값은 자동적으로 정리 한곳으로 모임
- 새로운 키와 모든 값들이 Reduce로 전달 됨.

◈ Reduce
- 같은 wordID(키')를 가진 모든 단어 정보(값')를 사용하여 하나로 모으면 Reduce
- 모여진 wordID(키')와 단어 정보(값')를 사용하여 검색을 위한 역 인덱스 생성

◈ 개발자가 준비할 것은 Map과 Reduce라는 2가지 처리
- Map을 다수의 머신으로 분산하여 실행하는 것과 같은 키의 데이터를 모아 Reduce를 호출하는 등 귀찮은 작업은 모두 시스템에 맡기는 것

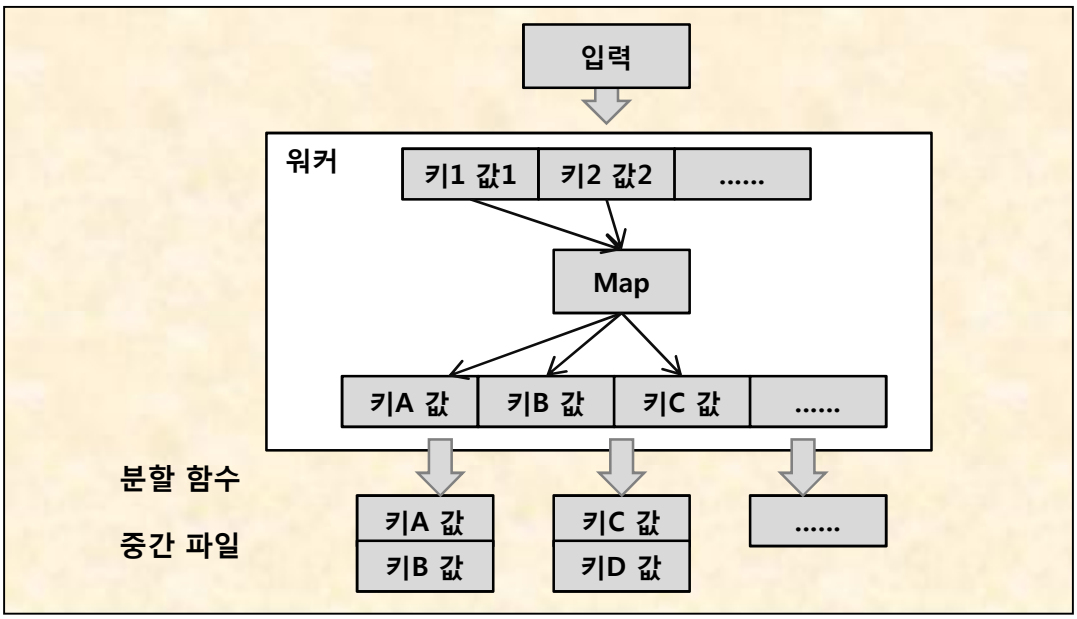
맵과 리듀스를 구현한 클래스로 Mapper와 Reducer 클래스가 있습니다. Mapper 클래스에는 map() 메서드가 정의 되어 있으며, Reducer 클래스에는 reduce() 메서드가 정의되어 있습니다. 개발자는 이들 메서드들을 재정의 하여 원하는 자료 셋을 가공할 수 있습니다.

다음은 map()메서드와 reduce() 메서드를 사용할 때 입니다.

• map()
리스트의 각 원소들에게 어떤 공통된 작업을 처리하고자 할 때 사용됩니다.
여러 입력 자료 셋으로부터 여러 중간 value 들을 생성합니다.

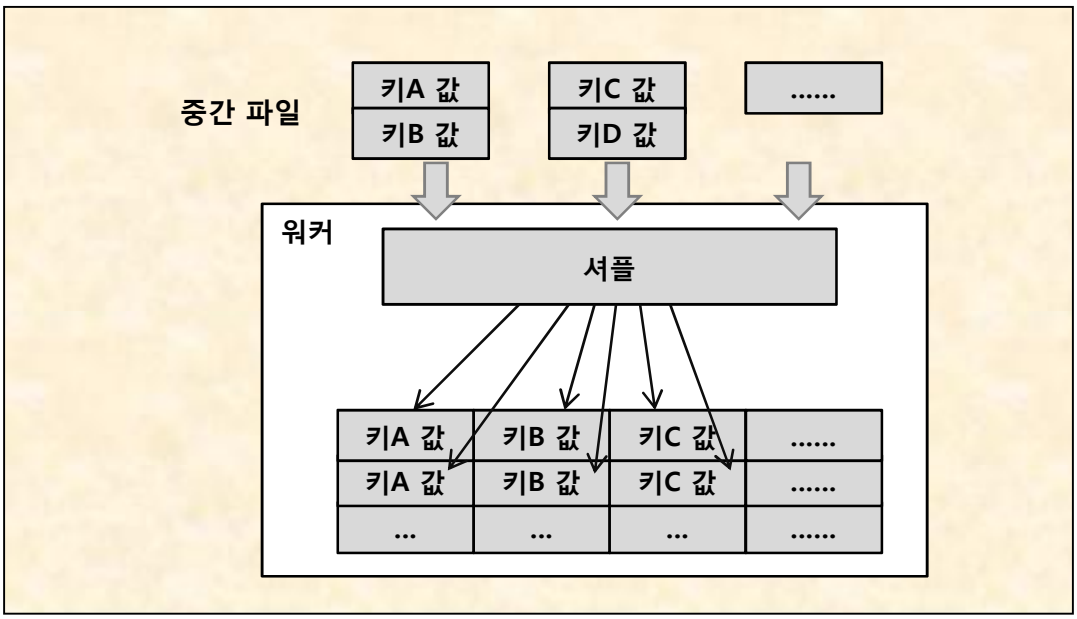
• reduce()
리스트 전체 원소를 모아 하나의 결과를 출력하고자 할 때 사용됩니다.
출력 key를 기준으로 각각 작업을 수행할 수 있습니다.

MapReduce 동작 과정 - 1단계 과정 (Map)



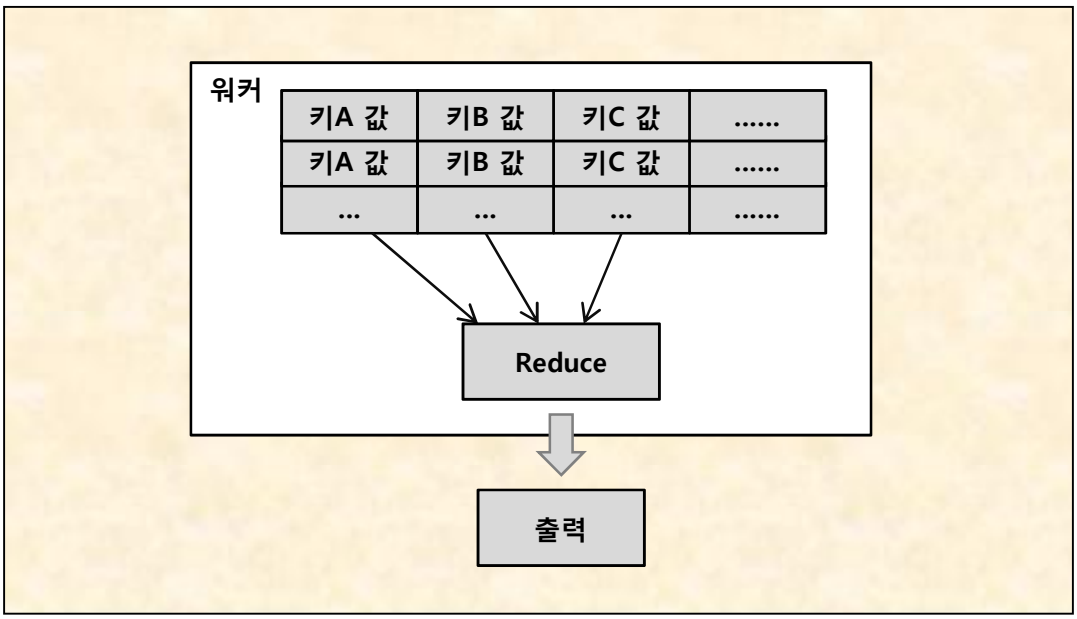
1. 마스터는 입력 파일을 복수의 단편으로 분할한 다음
2. 하나하나의 처리를 순차적을 워커에 할당합니다.
3. 워커는 단편에 기록된 키와 값을 Map호출하여
4. Map은 새로운 키와 값을 출력합니다.
5. 워커는 잠시 메모리상 중간파일로 저장하는데
6. 중간파일은 일시적으로만 사용하며, 효율을 높이기 위해 GFS가 아닌 로컬파일로 저장합니다.

MapReduce 동작 과정 – 2단계 과정 (Shuffle)



1. 워커에서 중간파일이 생성이 되면 마스터를 경유해 Reduce 워커에 전달합니다.
2. Reduce 워커는 네트워크를 경유해 중간파일을 가져오고 바로 셔플을 시작합니다.
3. 중간파일에 기록된 키에 따라 모든 데이터가 정렬되며,
4. 중간파일이 작으면 메모리상에서 정렬, 크면 임시파일로 보냅니다.
5. 중간 파일이 모일 때까지 셔플이 완료 되지 않기 때문에 Map이 계속 되는 한 셔플도 끝나지 않습니다.
6. Map쪽에서 중간파일이 생성될 때마다 순차적으로 셔플이 이루어지고 모든 Map의 처리가 완료가 되면 셔플도 끝납니다.

MapReduce 동작 과정 – 3단계 과정 (Reduce)



1. Reduce는 셔플이 끝난 그룹부터 차례로 시작합니다.
2. Reduce에 건네진 키는 사전 순으로 작은 것부터 차례로 선택됩니다.
3. 중간파일에 기록된 키에 따라 모든 데이터가 정렬되며,
4. Reduce의 출력은 그룹별로 하나의 파일로서 GFS로 보내집니다.
5. 그러므로 최종적으로 그룹의 수만큼 출력 파일이 만들어 집니다.

Hadoop MapReduce 프로그래밍 요소

- ◈ 데이터 타입
  - ➢ 맵리듀스 프로그램에서 키와 값으로 사용되는 데이터 타입
  - ➢ 반드시 WritableComparable 인터페이스를 구현해야 함
- ◈ InputFormat
  - ➢ 입력 스플릿을 맵 메서드의 입력 파라미터로 사용할 수 있도록 함
- ◈ Mapper
  - ➢ 키와 값으로 구성된 입력 데이터를 전달받아 데이터를 가공하고 분류해서 새로운 데이터 목록을 생성
- ◈ Partitioner
  - ➢ 맵 태스크의 출력 데이터가 어떤 리듀스 태스크로 전달될지 결정
- ◈ Reducer
  - ➢ 맵 태스크의 출력 데이터를 입력 데이터로 전달받아 집계 연산을 수행
- ◈ OutputFormat
  - ➢ setOutputFormatClass 메서드로 설정하는 맵 리듀스 잡의 출력 데이터 포맷

데이터 타입

- 맵리듀스 프로그램에서 키와 값으로 사용되는 데이터 타입 클래스 입니다.
- 데이터 타입 클래스는 개발자가 직접 구현할 수도 있으며, 반드시 WritableComparable 인터페이스를 구현해야 합니다.
- 주요 데이터 타입 클래스 : BooleanWritable, ByteWritable, DoubleWritable, FloatWritable, IntWritable, LongWritable, Text, NullWritable

InputFormat

- 입력 스플릿을 맵 메서드의 입력 파라미터로 사용할 수 있도록 합니다.
- 주요 InputFormat 클래스 : TextInputFormat, KeyValueTextInputFormat, NLineInputFormat, DelegatingInputFormat, CombineFileInputFormat, SequenceFileInputFormat

Mapper

- 입력 스플릿에서 key/value 쌍마다 실행됩니다.
- 키와 값으로 구성된 입력 데이터를 전달받아 데이터를 가공하고 분류해서 새로운 데이터 목록을 생성합니다.
- Mapper 클래스의 메서드는 4개가 있습니다.
  protected void setup(Mapper.Context context) throws IOException
  → 태스크가 시작할 때 한번만 실행됩니다.
  → setup 메서드에는 자원획득, 초기화, 사용자 파라미터를 읽는 코드 등이 들어갑니다.

Hadoop MapReduce 프로그래밍 요소

protected void map(KEYIN key, VALUEIN value, Mapper.Context context) throws IOException, InterruptedException

→ 컨텍스트에 key/value 쌍으로 저장하는 코드를 작성합니다.

protected void cleanup(Mapper.Context context) throws IOException

→ 태스크를 종료할 때 한번만 호출되므로, 자원을 반환하는 코드를 작성합니다.

public void run(Mapper.Context context) throws IOException, InterruptedException

→ 매퍼를 실행할 때 더 정교한 제어를 하기 위해 재정의 하는 메서드 입니다.

Partitioner

- 맵 태스크의 출력 데이터가 어떤 리듀스 태스크로 전달될지 결정합니다.
- setPartitionerClass() 메서드로 파티셔너 클래스를 설정합니다.

Reducer

- 맵 태스크의 출력 데이터를 입력 데이터로 전달받아 집계 연산을 수행하는 클래스입니다.
- Reducer 클래스의 메서드는 4개가 있습니다.

protected void setup(Reducer.Context context) throws IOException

→ 태스크를 시작할 때 한번만 실행됩니다.

→ setup 메서드에는 Mapper의 setup 메서드와 같은 기능을 합니다. 자원획득, 초기화, 사용자 파라미터를 읽는 코드 등이 들어갑니다.

protected void reduce(KEYIN key, VALUEIN value, Reducer.Context context) throws IOException, InterruptedException

→ 맵 태스크의 출력 데이터를 집계 연산을 수행하여 컨텍스트에 key/value 쌍으로 저장하는 코드를 작성합니다.

protected void cleanup(Reducer.Context context) throws IOException

→ 태스크를 종료할 때 한번만 호출됩니다. 자원을 반환하는 코드를 작성합니다.

public void run(Reducer.Context context) throws IOException, InterruptedException

→ 리듀서가 실행될 때 더 정교한 제어를 하기 위해 재정의 하는 메서드입니다.

OutputFormat

- 맵 리듀스 잡의 출력 데이터 포맷 클래스입니다.
- setOutputFormatClass 메서드로 출력 데이터 포맷을 설정합니다.
- 주요 클래스 : TextOutputFormat, SequenceFileOutputFormat, SequenceFileAsBinaryOutputFormat, FilterOutputFormat, LazyOutputFormat, NullOutputFormat

http://hadoop.apache.org/docs/r2.2.0/api/index.html

HDFS 접근 – FileSystem

- org.apache.hadoop.fs.FileSystem
- 하둡에서 제공하는 파일 시스템을 추상화 한 클래스
- 로컬 파일 시스템이나 HDFS나 어떤 파일 시스템을 사용하든 FileSystem 클래스로 파일에 접근한다.
- 객체 생성
  - Configuration conf = new Configuration();
  - FileSystem hdfs = FileSystem.get(conf);
  - Path path = new Path("파일시스템 경로");
- 또는
  - Path path = new Path("파일시스템 경로");
  - FileSystem fs = path.getFileSystem(getConf());
- 파일 시스템 경로
  - 상대경로는 /user/hadoop 디렉토리 기준

하둡은 자바 어플리케이션에서 하둡의 파일 시스템을 접근할 수 있도록 라이브러리를 제공합니다

org.apache.hadooop.fs 패키지의 주요 클래스

- FileSystem : 파일 시스템 클래스
- FSDataInputStream : java.io.DataInputStream의 하위 클래스, HDFS로부터 입력 스트림을 생성, read(Bytebuffer buf), seek(long offset) 메서드 등이 구현 되어 있습니다.
- FSDataOutputStream : java.io.DataOutputStream의 하위 클래스, HDFS로부터 출력 스트림 생성, getPos(), hsync(), hflush() 메서드 등이 구현되어 있습니다.
- Path : FileSystem 객체를 리턴하는 getFileSystem() 메서드가 있습니다.

생성자

- protected FileSystem()

hsync는 posix의 fsync와 유사합니다. fsync는 파일의 완전한 내부 상태와 디스크상의 상태를 동기화 시킵니다.

주요 메서드

- public static FileSystem get(Configuration conf)
- append(), concat()
- copyFromLocalFile(), copyToLocalFile()
- create(), delete(), mkdir(), rename()
- getHomeDirectory(), getName(), getLength(), getWorkingDirectory()
- exists(), isDirectory(), isFile()
- listFiles()

HDFS 접근 – SinglefileWriteRead.java

```
1.  package hadoop.sample.fileio; //패키지 이름 입력. 클래스 생성할 때
2.
3.  //패키지 임포트, 해당 패키지 안의 자바 클래스를 사용하겠다는 의미
4.  import org.apache.hadoop.conf.*;
5.  import org.apache.hadoop.fs.*;
6.
7.   //클래스 선언부
8.  public class SinglefileWriteRead {
9.
10.     //프로그램 시작점. 반드시 하나 존재해야 함
11.     //jar파일 실행 시 main이 있는 클래스를 알려줘야 함
12.     public static void main(String[] args) {
13.         try {  //하둡 관련 클래스들을 사용할 때 예외발생시 처리.
14.             Configuration conf = new Configuration();   //hdfs을 접근하기 위해 환경을 구성
15.             FileSystem hdfs = FileSystem.get(conf);  //conf객체를 통해 파일시스템 접근
16.             Path path = new Path("temp.txt");  //hdfs의 홈디렉토리를 기준으로 경로 생성
17.
18.             if(hdfs.exists(path)) {  //temp.txt 파일이 있으면…
19.                 hdfs.delete(path, true); //파일(디렉토리) 삭제, 디렉토리를 삭제할 때는 true
20.              }
21.             //hdfs에 파일을 쓰기 위한 출력 스트림 생성
22.             FSDataOutputStream outStream = hdfs.create(path);
23.             outStream.writeUTF("Helo World"); //문자열을 파일에 저장
24.             outStream.close();  //출력스트림 닫아주기
25.             System.out.println("file created in hdfs");
26.
27.             //파일시스템에 저장된 파일을 다시 불러오기 위한 입력스트림 생성
28.             FSDataInputStream inputStream = hdfs.open(path);
29.             String inputString = inputStream.readUTF();  //문자열 읽기
30.             inputStream.close();
31.             System.out.println("##input String : " + inputString);
32.
33.             System.out.println(path.getFileSystem(conf).getHomeDirectory()); //hdfs 홈 경로
34.             System.out.println(path.toUri());     //패스의 파일명
35.             System.out.println(path.getFileSystem(conf).getUri().getPath());
36.         } catch(Exception e) {
37.             System.out.println("error");           //sysout Ctrl+space
38.         }
39.     }
40. }
```

HDFS 접근 – 실행

- ◈ 이클립스에서 프로젝트 생성
  - ➢ File -> New -> Project -> Java Project
- ◈ 소스코드 작성
  - ➢ 프로젝트 컨텍스트 메뉴(마우스 우클릭 메뉴) -> New -> Class
- ◈ 컴파일
  - ➢ 저장하면 자동 컴파일 됨
- ◈ Export 하여 JAR파일 생성
  - ➢ jar파일로 만들어 실행해야 함
  - ➢ Export 할 파일 또는 패키지 선택 후 마우스 오른쪽 버튼 클릭 -> Export
    - ▪ -> Java -> JAR file -> 저장될 디렉토리 선택 후 jar 파일명 입력
    - ▪ -> JAR 파일에 포함될 파일 확인 및 JAR file 대상 경로와 이름 확인 후 Next
    - ▪ -> Next -> Browse... 클릭해 Main 클래스 선택 후 Next -> Finish
- ◈ 실행
  - ➢ hadoop jar *jar파일명 [패키지명을포함한main클래스이름] [옵션]*

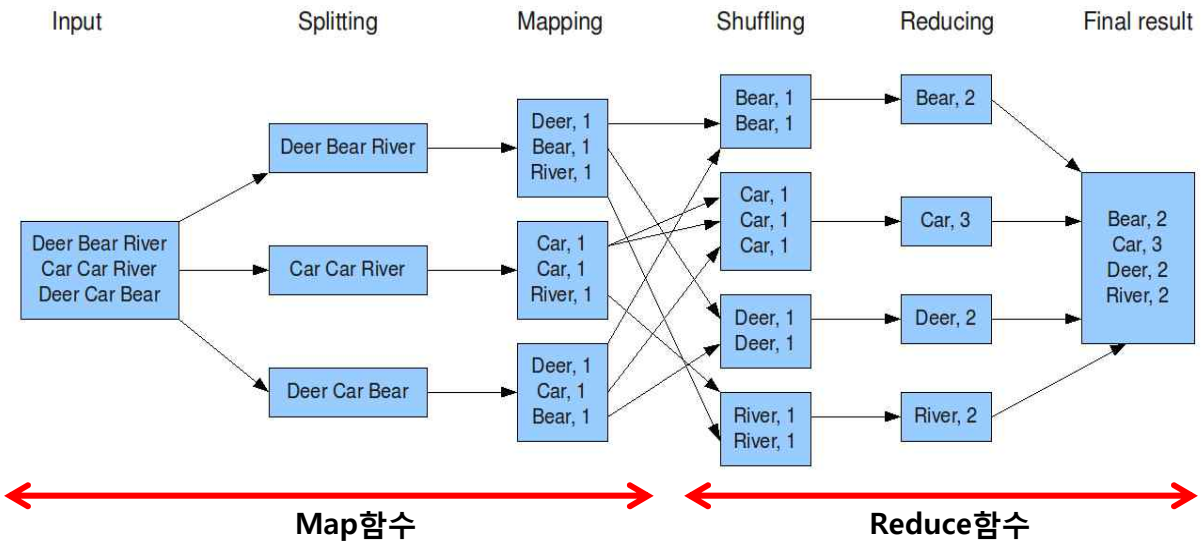
자바 코드를 작성하고 컴파일 하기 위해 개발 도구로 이클립스(Eclipse) 또는 넷빈즈(NetBeans)를 사용할 수 있습니다. 그러나 예제를 이클립스로 실행해 볼 수 없습니다.
예제 코드를 실행 시키기 위해서는 프로젝트 파일을 jar 파일로 만들어야 합니다. 하둡 분산파일 시스템에 접근하는 자바 코드를 실행하기 위해 java.exe를 이용하지 않습니다. hadoop 명령과 jar 옵션을 이용해 실행해야 합니다.

이클립스에서 Export 하여 JAR파일을 생성할 때 메인 클래스를 지정한다면 실행 시 메인 클래스 이름을 입력해야 하는 불편함을 줄일 수 있습니다. 그런데 JAR 파일 생성시 메인 클래스를 지정하지 않았다면 다음과 같이 메인 클래스 이름을 지정해야 합니다.
$ hadoop jar filewrite.jar hadoop.test.SinglefileWriteRead

이 예제가 간단해 보이더라도 중요하게 생각해야 할 것은 개발자는 자바 어플리케이션을 통해 하둡의 분산 파일 시스템을 사용할 수 있다는 것입니다. 과거에 생산되어 활용되지 않았던 데이터베이스 또는 파일 시스템의 데이터를 분산 파일 시스템에 저장하여 활용할 수 있는 기반을 마련해 준 것입니다.

이 교재에 나와있지 않은 다른 클래스들의 설명은 아래의 사이트를 통해 볼 수 있습니다.
http://hadoop.apache.org/docs/r2.2.0/api/index.html

WordCount



WordCount 샘플은 텍스트 파일의 내용에서 단의 수를 카운트 하는 예제입니다.

프로그래밍 순서

1. 매퍼 클래스 작성
   - Mapper 클래스를 상속받고 map() 메서드 재정의
   - map() 메서드 인자로 넘겨진 key와 value를 분석하여 context 객체를 통해 출력(write) 합니다.
2. 리듀서 클래스 작성
   - Recuder 클래스를 상속받고 reduce() 메서드 재정의
   - 매퍼의 결과가 셔플되고 각 키의 값은 Iterable 객체로 생성되어 리듀서의 입력으로 됩니다.
   - reduce() 메서드 인자로 넘겨진 Iterable values를 카운트 하여 context 객체를 통해 출력(write) 합니다.
3. 드라이버 클래스 작성
   - Job 객체 생성(Job job = new Job("WordCount");
   - Job 객체에 맵리듀스 잡의 실행 정보 설정.
   - 맵리듀스 잡 실행(job.waitForComplete);
4. Jar 파일로 Export(메인 클래스 지정해 주면 실행 시 편합니다)
   - 패키지 선택 후 마우스 오른쪽 버튼 클릭
   - -> Export ->Java/JAR file 선택 후 Next
   - -> JAR 파일명 입력 후 Next -> Next -> Main class 선택 후 Finish
5. 샘플 텍스트 파일 작성
6. 샘플 텍스트 파일 HDFS에 업로드
   - hdfs dfs -put input.txt input.txt
   - HDFS에 output 디렉토리 생성
7. 실행
   - hadoop jar WordCount.jar input.txt output
   - hdfs dfs -cat /output/part-r-00000

매퍼

- org.apache.hadoop.mapreduce.Mapper
- public class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT> extends Object
- 메서드
  - protected void setup(Context context) throws IOException, InterruptedException
  - protected void map(KEYIN key, VALUEIN value, Context context) throws IOException, InterruptedException
  - protected void cleanup(Context context) throws IOException, InterruptedException
  - public void run(Context context) throws IOException, InterruptedException
- 내부 클래스
  - org.apache.hadoop.mapreduce.Mapper.Context

Example

```
1.  public class TokenCounterMapper extends Mapper<Object, Text, Text, IntWritable>{
2.      private final static IntWritable one = new IntWritable(1);
3.      private Text word = new Text();
4.
5.      public void map(Object key, Text value, Context context) throws IOException,
    InterruptedException {
6.          StringTokenizer itr = new StringTokenizer(value.toString());
7.          while (itr.hasMoreTokens()) {
8.              word.set(itr.nextToken());
9.              context.write(word, one);
10.         }
11.     }
12. }
```

WordCountMapper(매퍼 클래스)

```
1.  package hadoop.sample.wordcount;
2.
3.  import java.io.IOException;
4.  import java.util.StringTokenizer;
5.
6.  import org.apache.hadoop.io.IntWritable;
7.  import org.apache.hadoop.io.LongWritable;
8.  import org.apache.hadoop.io.Text;
9.  import org.apache.hadoop.mapreduce.Mapper;
10.
11. //Mapper클래스를 상속해야 함<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
12. public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
13.
14.     //1 값을 저장하기 위해 IntWritable 타입으로 선언(출력 값)
15.     private final static IntWritable one = new IntWritable(1);
16.
17.     //단어를 저장하기 위해 Text 타입으로 선언(출력 타입)
18.     private Text word = new Text();
19.
20.     //key : 입력 키, value : 입력 값
21.     //입력키는 라인번호, 입력 값은 라인의 문자열
22.     public void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
23.         //입력 문자열을 공백으로 분리하여 토큰화 함
24.         StringTokenizer itr = new StringTokenizer(value.toString());
25.
26.         //토크나이저에 다음 토큰이 존재할 때까지 실행
27.         while (itr.hasMoreTokens()) {
28.             word.set(itr.nextToken());  //다음 토큰을 Text 타입 변수에 저장
29.             context.write(word, one);  //Text타입에 저장된 값은 key, 1은 value로 저장
30.         }
31.     }
32. }
```

리듀서

- org.apache.hadoop.mapreduce.Reducer
- public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> extends Object
- 메서드
  - protected void setup(Context context)
    throws IOException, InterruptedException
  - protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)
    throws IOException, InterruptedException
  - protected void cleanup(Context context)
    throws IOException, InterruptedException
  - public void run(Context context)
    throws IOException, InterruptedException
- 내부 클래스
  - org.apache.hadoop.mapreduce.Reducer.Context

Example

```
1.  public class IntSumReducer<Key> extends Reducer<Key,IntWritable, Key,IntWritable> {
2.      private IntWritable result = new IntWritable();
3.
4.      public void reduce(Key key, Iterable<IntWritable> values, Context context) throws
    IOException, InterruptedException {
5.          int sum = 0;
6.          for (IntWritable val : values) {
7.              sum += val.get();
8.          }
9.          result.set(sum);
10.         context.write(key, result);
11.     }
12. }
```

WordCountReducer(리듀서 클래스)

```
1.  package hadoop.sample.wordcount;
2.
3.  import java.io.IOException;
4.
5.  import org.apache.hadoop.io.IntWritable;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.Reducer;
8.
9.  //리듀서는 Reducer를 상속받아 작성합니다.
10. //Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
11. public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
12.
13.     //출력의 값으로 사용할 변수
14.     private IntWritable result = new IntWritable();
15.
16.     //key : 입력 값, values : 동일키 일때의 값들
17.     //왜? values인가?
18.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
19.             throws IOException, InterruptedException {
20.         int sum = 0;    //합을 저장하기 위한 변수
21.         for (IntWritable val : values) {//values안의 데이터를 소비하는 반복문
22.         //한번 실행할 때마다 values안에서 하나씩 빼와서 val에 저장
23.
24.             //객체를 기본 타입으로 바꿔줘야 하기 때문에 val.get() 사용
25.             sum += val.get();  //IntWritable 클래스의 get 메서드는 안에 있는 값을 반환
26.         }
27.         result.set(sum);        //결과는 기본타입 -> 객체에 저장
28.         context.write(key, result);//출력
29.     }
30. }
```

드라이버

◈ 맵리듀스 잡에 대한 실행 정보를 설정하고, 맵리듀스 잡을 실행합니다.
◈ 실행 단계
1. 잡 객체를 생성합니다.
   - Job job = new Job(conf, "잡 이름");
2. 잡 객체에 맵리듀스 잡의 실행 정보를 설정합니다.
   - job.setJarByClass(WordCount.class);
   - job.setMapperClass(WordCountMapper.class);
   - job.setReducerClass(WordCountReducer.class);
   - job.setInputFormat(TextInputFormat.class);
   - job.setOutputFormat(TextOutputFormat.class);
   - job.setOutputKeyClass(Text.class);
   - job.setOutputValueClass(TextOutputFormat.class);
3. 맵리듀스 잡을 실행합니다.
   - job.waitForCompletion(true);

출력으로 생성된 디렉터리와 파일은 다음 실행 시 덮어 써지지 않으므로 파일시스템 객체를 이용하여 삭제해야 합니다.

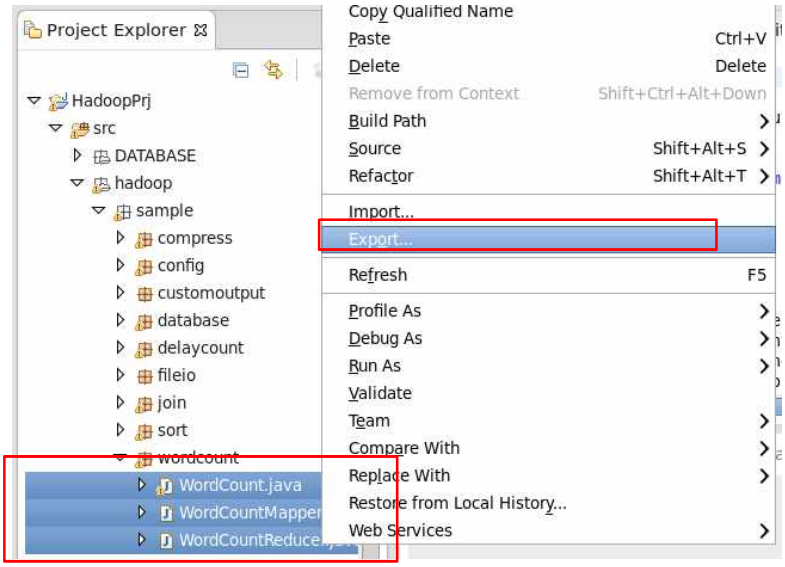
삭제 코드

```
FileSystem hdfs = FileSystem.get(conf);
Path path = new Path("temp.txt");
if(hdfs.exists(path)) {
    hdfs.delete(path, true);
}
```

WordCount(드라이버 클래스)

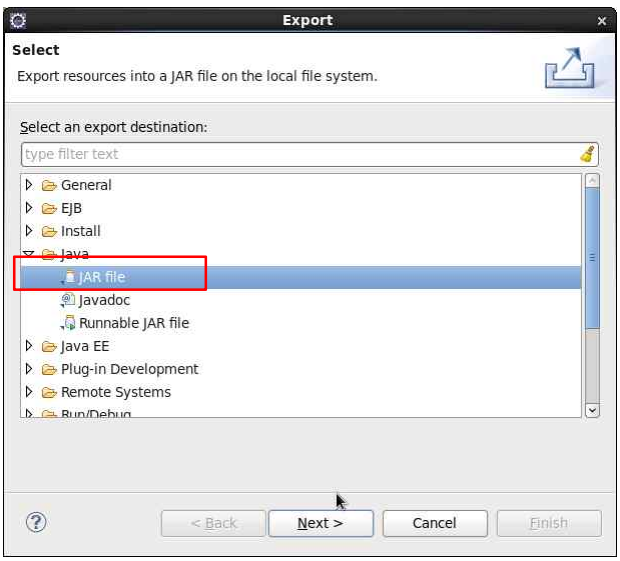
```
1.  package hadoop.sample.wordcount;
2.  import org.apache.hadoop.conf.Configuration;
3.  import org.apache.hadoop.fs.FileSystem;
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.io.IntWritable;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.Job;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class WordCount {
14.     public static void main(String[] args) throws Exception {
15.         Configuration conf = new Configuration();
16.         Job job = new Job(conf, "WordCount");
17.
18.         FileSystem fs = FileSystem.get(conf);//파일시스템 객체를 생성
19.         Path inputPath = new Path("input.txt");//입력 패스
20.         Path outputPath = new Path("out");      //출력 패스
21.         if(fs.exists(outputPath)) {//만일 출력 패스가 존재하면
22.             fs.delete(outputPath, true);//출력 패스 삭제, 디렉토리이므로 true
23.         }
24.
25.         job.setJarByClass(WordCount.class);//메인 클래스 지정
26.         job.setMapperClass(WordCountMapper.class);//메퍼 클래스 지정
27.         job.setReducerClass(WordCountReducer.class);//리듀서 클래스 지정
28.
29.         job.setInputFormatClass(TextInputFormat.class);//입력 포맷지정
30.         job.setOutputFormatClass(TextOutputFormat.class);//출력 포맷 지정
31.
32.         job.setOutputKeyClass(Text.class);//출력 키의 타입(단어는 텍스트)
33.         job.setOutputValueClass(IntWritable.class);//출력 값의 타입(갯수는 정수)
34.
35.         FileInputFormat.addInputPath(job, inputPath);//입력 패스 지정
36.         FileOutputFormat.setOutputPath(job, outputPath);//출력 패스 지정
37.
38.         job.waitForCompletion(true);//작업 실행 종료때까지 기다립니다.
39.     }
40. }
```

WordCount JAR파일 생성



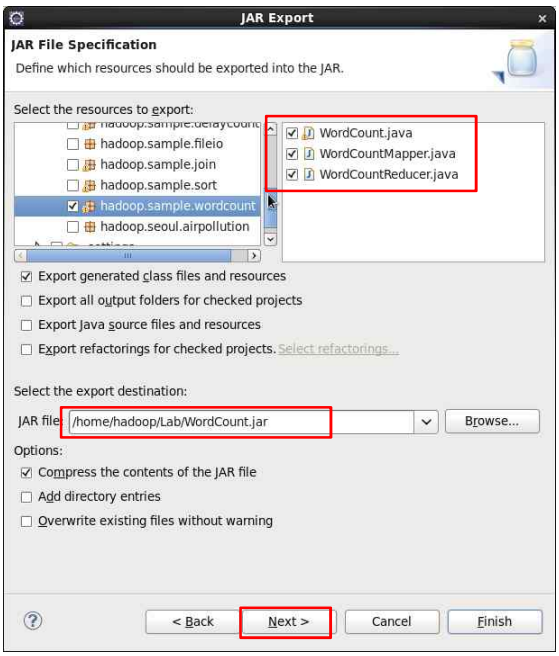
JAR파일에 포함될 파일 선택 후 마우스 오른쪽 버튼 클릭

[Export...]



Java – JAR file 선택 후 [Next]

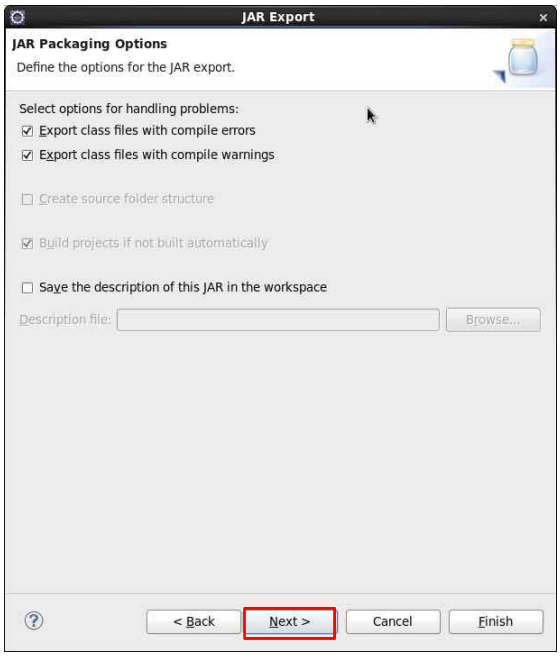
WordCount JAR파일 생성



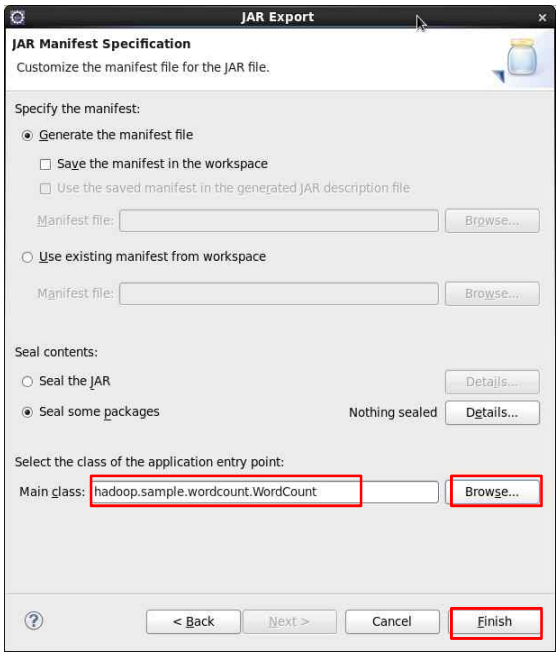
JAR 파일로 만들어질 패키지와 파일 확인

JAR file 대상 경로와 이름 확인

[Next]



[Next]



[Browse...] 클릭하여Main 클래스 선택 후 [Finish]

WordCount 실행

- input.txt 파일을 HDFS에 업로드
  - hdfs dfs -put input.txt /input.txt
- 실행
  - hadoop jar WordCount.jar /input.txt /output/word_count
- 실행 시 출력 로그
  - 잡/리듀스 처리 과정이 로그로 출력됨
  - 파일 출력 포맷 카운터
    - 출력된 바이트 수
  - 파일시스템 카운터
    - HDFS 입/출력 바이트 수
  - 파일 입력 포맷 카운터
    - 입력 바이트 수
  - 맵리듀스 프레임웍
    - 맵 입력 레코드 수, 맵 출력 바이트 수, 힙 메모리 사용량, CPU 사용 시간
    - 리듀스 입력 수, 리듀스 입력 그룹 수, 리듀스 출력, 맵 출력 레코드 수

```
[hadoop@master Lab]$ hadoop jar WordCount.jar /input.txt /output/word_count
14/04/26 06:47:12 INFO client.RMProxy: Connecting to ResourceM
14/04/26 06:47:13 WARN mapreduce.JobSubmitter: Hadoop command-
Tool interface and execute your application with ToolRunner to
14/04/26 06:47:13 INFO input.FileInputFormat: Total input path
14/04/26 06:47:13 INFO mapreduce.JobSubmitter: number of split
14/04/26 06:47:13 INFO Configuration.deprecation: user.name is
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.jar i
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.outpu
ce.job.output.value.class
14/04/26 06:47:13 INFO Configuration.deprecation: mapreduce.ma
.map.class
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.job.n
e
14/04/26 06:47:13 INFO Configuration.deprecation: mapreduce.re
job.reduce.class
14/04/26 06:47:13 INFO Configuration.deprecation: mapreduce.in
duce.job.inputformat.class
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.input
fileinputformat.inputdir
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.outpu
t.fileoutputformat.outputdir
14/04/26 06:47:13 INFO Configuration.deprecation: mapreduce.ou
educe.job.outputformat.class
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.map.t
ps
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.outpu
.job.output.key.class
14/04/26 06:47:13 INFO Configuration.deprecation: mapred.worki
working.dir
14/04/26 06:47:14 INFO mapreduce.JobSubmitter: Submitting toke
14/04/26 06:47:14 INFO impl.YarnClientImpl: Submitted applicat
ager at /0.0.0.0:8032
14/04/26 06:47:14 INFO mapreduce.Job: The url to track the job
326_0001/
14/04/26 06:47:14 INFO mapreduce.Job: Running job: job_139851
14/04/26 06:47:24 INFO mapreduce.Job: Job job_1398519263326_0(
14/04/26 06:47:24 INFO mapreduce.Job:  map 0% reduce 0%
14/04/26 06:47:30 INFO mapreduce.Job:  map 100% reduce 0%
14/04/26 06:47:37 INFO mapreduce.Job:  map 100% reduce 100%
14/04/26 06:47:37 INFO mapreduce.Job: Job job 1398519263326 0(
```

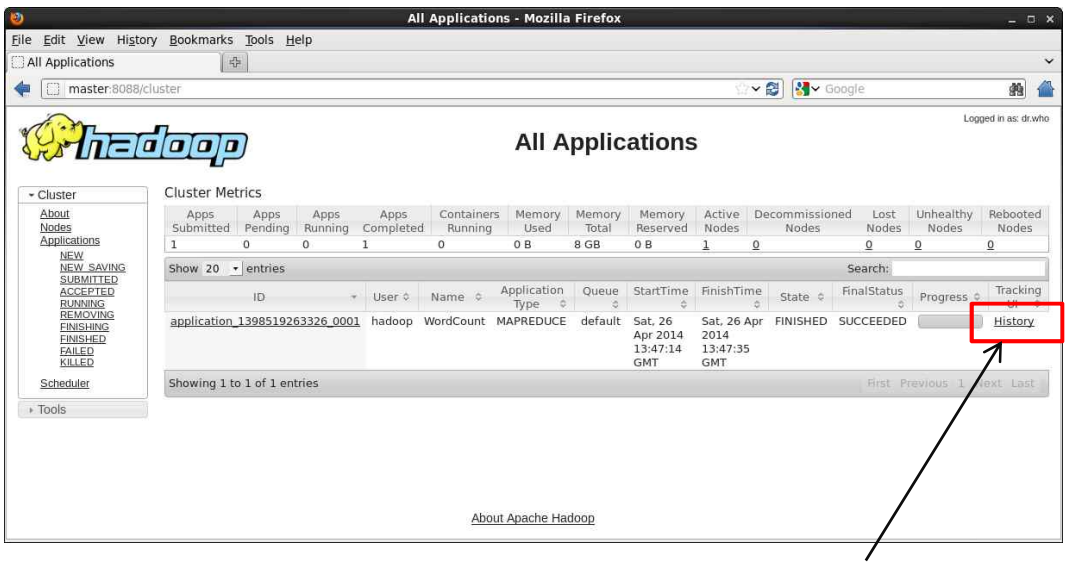
```
14/04/26 06:47:37 INFO mapreduce.Job: Counters: 43
        File System Counters
                FILE: Number of bytes read=1834
                FILE: Number of bytes written=161841
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1084
                HDFS: Number of bytes written=983
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3884
                Total time spent by all reduces in occupied slots (ms)=3794
        Map-Reduce Framework
                Map input records=5
                Map output records=140
                Map output bytes=1548
                Map output materialized bytes=1834
                Input split bytes=93
                Combine input records=0
                Combine output records=0
                Reduce input groups=101
                Reduce shuffle bytes=1834
                Reduce input records=140
                Reduce output records=101
                Spilled Records=280
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=99
                CPU time spent (ms)=1550
                Physical memory (bytes) snapshot=348790784
                Virtual memory (bytes) snapshot=814821376
                Total committed heap usage (bytes)=317194240
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=991
        File Output Format Counters
                Bytes Written=983
```

WordCount 실행 결과 확인

◈ 결과 확인
- ➢ hdfs dfs -ls /output/word_count
  - ▪ 생성된 파일 리스트
- ➢ hdfs dfs -cat /output/word_count/part-r-00000
  - ▪ 맵리듀스 출력 결과

```
[hadoop@master Lab]$ hadoop fs -ls /output/word_count
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2014-04-26 06:47 /output/word_count/_SUCCESS
-rw-r--r--   1 hadoop supergroup        983 2014-04-26 06:47 /output/word_count/part-r-00000
[hadoop@master Lab]$ hadoop fs -cat /output/word_count/part-r-00000
(CentOS 1
100%    1
American        1
CentOS  4
Chat,   1
Distribution    1
Email   1
Enterprise      2
Enterprise-class        1
FAQ.    1
Forums, 1
IRC     1
In      1
```

브라우저에서 잡 트래커를 이용해 실행 시 출력 로그를 다시 확인 할 수 있습니다.
http://master:8088/cluster (1.x는 http://master:50030/jobtracker.jsp)



맵리듀스 실행 결과를 자세하게 조회하려면 맵리듀스 어플리케이션이 종료되기 전에 히스토리 서버 데몬이 실행되고 있어야 합니다. 히스토리 서버 데몬은 다음 명령으로 실행이 가능합니다. sbin 디렉토리는 하둡 설치된 디렉토리 아래에 있습니다.

$ sbin/mr-jobhistory-daemon.sh start historyserver

데이터 타입

- org.apache.hadoop.WritableComparable 인터페이스
  - 네트워크 통신을 위한 최적화된 객체
- public interface WritableComparable<T> extends Writable, Comparable<T>
  - org.apache.hadoop.io.Writable 인터페이스
    - void write(DataOutput out) throws IOException
      - 데이터 값을 직렬화 한다.
    - void readFields(DataInput in) throws IOException
      - 메서드는 직렬화된 데이터 값을 해제해서 읽는 역할을 한다.
  - java.lang.Comparable<T> 인터페이스
    - int compareTo(T o)
      - 정렬을 처리한다.
      - 현재 객체의 멤버 값이 크면 양수, 인자로 넘겨진 객체의 멤버 값이 크면 음수, 같으면 0을 리턴하도록 구현한다.
- 맵리듀스 프로그램에서 키로 사용되는 데이터타입은 WritableComparable 인터페이스를 implements 해야 한다.

Sample

```
1.  public class MyWritableComparable implements WritableComparable {
2.      // Some data
3.      private int counter;
4.      private long timestamp;
5.
6.      public void write(DataOutput out) throws IOException {
7.          out.writeInt(counter);
8.          out.writeLong(timestamp);
9.      }
10.
11.     public void readFields(DataInput in) throws IOException {
12.         counter = in.readInt();
13.         timestamp = in.readLong();
14.     }
15.
16.     public int compareTo(MyWritableComparable o) {
17.         int thisValue = this.value;
18.         int thatValue = o.value;
19.         return (thisValue < thatValue ? -1 : (thisValue==thatValue ? 0 : 1));
20.     }
21.
22.     public int hashCode() {
23.         final int prime = 31;
24.         int result = 1;
25.         result = prime * result + counter;
26.         result = prime * result + (int) (timestamp ^ (timestamp >>> 32));
27.         return result
28.     }
29. }
```

Wrapper 클래스

◈ 맵리듀스 API에서 제공하는 데이터 타입 클래스

클래스명	대상 데이터 타입	비고
BooleanWritable	Boolean	boolean
ByteWritable	단일 byte	byte
DoubleWritable	Double	double
FloatWritable	Float	float
IntWritable	Integer	integer
LongWritable	Long	long
Text	UTF8 형식의 문자열	String
NullWritable	데이터 값이 없을 경우에 사용함	null

InputFormat 클래스

- 맵리듀스는 입력 스플릿을 맵 메서드의 입력 파라미터로 사용할 수 있게 InputFormat 이라는 추상화 클래스를 제공한다.
- InputFormat 클래스는 입력 스플릿을 맵 메서드가 사용할 수 있게 getSplits() 메서드를 제공한다.
- createRecordReader() 메서드는 맵 메서드가 입력 스플릿을 키와 목록의 형태로 사용 할 수 있게 RecordReader 객체를 생성한다.
- 매퍼의 map() 메서드는 RecordReader 객체에 담겨 있는 키와 값을 읽어 들여 분석 로직을 수행한다.
- InputFormat은 드라이버 클래스에서 Job 인터페이스의 setInputFormatClass() 메서드로 설정한다.

InputFormat 유형

InputFormat	기능
TextInputFormat	텍스트 파일을 분석할 때 사용하며, 캐리지 리턴 값(\r, \r\n)을 기준으로 레코드를 분류합니다. 키는 라인의 바이트 오프셋이며, LongWritable 타입을 사용합니다. 값은 라인의 내용이며, Text 타입을 사용합니다.
KeyValueTextInputFormat	텍스트 파일을 입력 파일로 사용할 때 라인 번호가 아닌 임의의 키 값을 지정해서 키와 값의 목록으로 읽게 됩니다.
NLineInputFormat	맵 태스크가 입력 받을 텍스트 파일의 라인 수를 제한하고 싶을 때 사용합니다.
DelegatingInputFormat	여러 개의 서로 다른 입력 포맷을 사용하는 경우에 각 경로에 대한 작업을 위임합니다.
CombineFileInputFormat	이 표에 있는 다른 InputFormat들은 파일당 스플릿을 생성하지만, CombineFileInputFormat은 여러 개의 파일을 스플릿으로 묶어서 사용합니다. 이때 각 노드와 랙의 위치를 고려해서 스플릿을 결정하게 됩니다.
SequenceFileInputFormat	시퀀스 파일을 입력 데이터로 사용합니다. 시퀀스 파일은 바이너리 형태의 키와 값의 목록으로 구성된 텍스트 파일입니다. 시퀀스 파일은 압축과 직렬화 프레임워크를 이용해 다양한 유형을 저장할 수 있습니다.
SequenceFileAsBinaryInputFormat	시퀀스 파일의 키와 값을 임의 바이너리 객체로 변환해서 사용합니다.
SequenceFileAsTextInputFormat	시퀀스 파일의 키와 값을 Text 객체로 변환해서 사용합니다.

OutputFormat 클래스

- 맵 리듀스 잡의 출력 데이터 포맷은 setOutputFormatClass 메서드로 설정한 포맷대로 만들어진다.
- 이 때 사용하는 출력 데이터 포맷은 OutputFormat이라는 추상화 클래스를 상속받아 구현되었다.
- 사용자가 별도의 OutputFormat을 설정하지 않을 경우 맵리듀스 프레임워크는 TextOutputFormat을 기본 포맷으로 설정한다.
- FileOutputFormat 클래스를 상속받아 출력 포맷 클래스를 만들어 원하는 위치에 원하는 파일이름으로 잡의 결과를 출력할 수 있다.

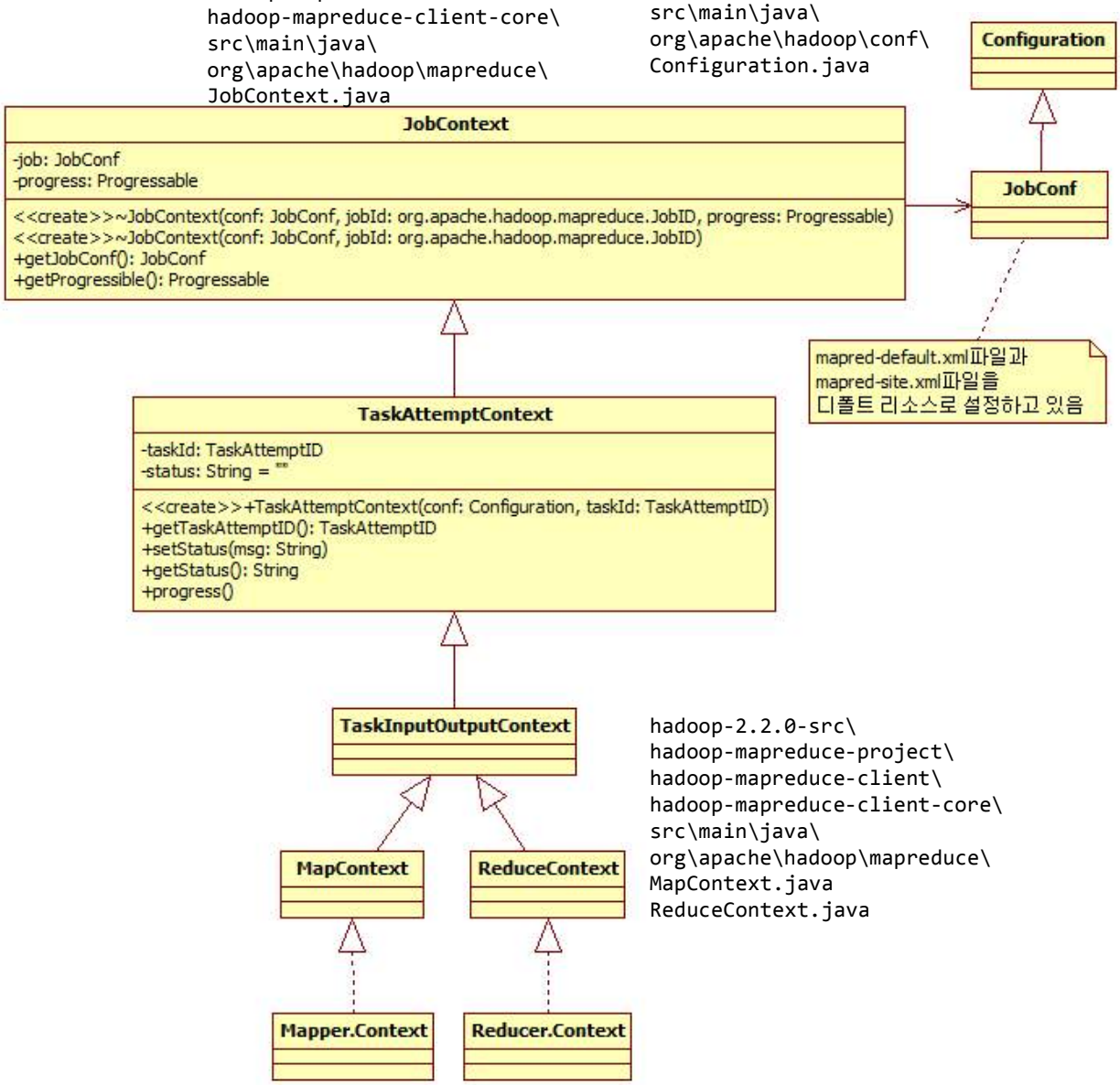
OutputFormat 유형

OutputFormat	기능
TextOutputFormat	텍스트 파일에 레코드를 출력할 때 사용합니다. 레코드를 출력할 때 키와 값의 구분자는 탭을 사용합니다.
SequenceFileOutputFormat	시퀀스 파일을 출력물로 사용합니다.
SequenceFileAsBinaryOutputFormat	SequenceFileOutputFormat을 상속받아 구현됐음, 바이너리 포맷의 키와 값을 SequenceFile 컨테이너에 씁니다.
FilterOutputFormat	OutputFormat 클래스의 Wrapper 클래스입니다. OutputFormat 클래스를 편리하게 사용할 수 있는 메서드를 제공합니다.
LazyOutputFormat	FileOutputFormat을 상속받은 클래스는 출력할 내용이 없더라도 리듀스의 출력파일(part-nnnnn)을 생성합니다. LazyOutputFormat을 사용하면 첫 번째 레코드가 해당 파티션(part-nnnnn)으로 보내질 때만 출력 파일을 생성합니다.
NullOutputFormat	출력 데이터가 없을 때 사용합니다.

Context

hadoop-2.2.0-src\
hadoop-mapreduce-project\
hadoop-mapreduce-client\
hadoop-mapreduce-client-core\
src\main\java\
org\apache\hadoop\mapreduce\
JobContext.java

hadoop-2.2.0-src\
hadoop-common-project\
hadoop-common\
src\main\java\
org\apache\hadoop\conf\
Configuration.java



hadoop-2.2.0-src\
hadoop-mapreduce-project\
hadoop-mapreduce-client\
hadoop-mapreduce-client-core\
src\main\java\
org\apache\hadoop\mapreduce\
MapContext.java
ReduceContext.java

Context는 매퍼와 리듀서의 메서드들이 공통적으로 파라미터로 선언하고 있는 클래스입니다. Mapper클래스와 Reducer 클래스 안에 각각 내부클래스로 선언되어 있으며 Context 클래스는 TaskInputOutContext 클래스를 상속받고 있습니다. TaskInputOutputContext 클래스는 태스크로부터 입력과 출력을 할 수 있도록 하는 클래스입니다. JobContext 클래스는 태스크가 실행되는 동안 잡의 읽기 전용 뷰를 태스크에게 제공합니다.
JobConf 클래스는 디폴트 리소스 파일을 지정하고 있으며, 리소스 파일의 설정 값을 저장합니다.

2 Hadoop 맵리듀스 기초다지기

Objectives

- 항공운항 데이터 분석을 통하여 맵리듀스 프로그래밍의 기초를 다집니다.

이 페이지는 여백 페이지입니다.

분석용 데이터 준비

- http://stat-computing.org/dataexpo/2009/the-data.html
- 미국 항공편 운항 통계 데이터 다운로드
- 1987년부터 2008년까지 미국 내 모든 상업 항공편에 대한 항공편 도착과 출발 세부 사항에 대한 정보를 제공
- csv 파일, 총 29개 컬럼으로 구성, 컬럼 정보는 콤마(,)를 기분으로 구분
- 총 파일 크기 11GB(총 압축 크기 1.6GB)
- hdfs dfs -put ~/Lab/Data/ /airline ← 분석용 데이터 업로드

번호	컬럼명	내용
1	Year	연도, 1987 ~ 2008
2	Month	월, 1 ~ 12
3	DayofMonth	일, 1 ~ 31
4	DayOfWeek	요일, 1(월요일) ~ 7(일요일)
5	DepTime	실제 출발 시각, 현지 시각 기준 hhmm 형태로 표기
6	CRSDepTime	예정 출발 시각, 현지 시각 기준 hhmm 형태로 표기
7	ArrTime	실제 도착 시각, 현지 시각 기준 hhmm 형태로 표기
8	CRSArrTime	예정 도착 시각, 현지 시각 기준 hhmm 형태로 표기
9	UniqueCarrier	항공사 코드
10	FlightNum	항공편 번호
11	TailNum	항공기 등록 번호(비행기 꼬리 날개쪽에 표기)
12	ActualElapsedTime	실제 경과 시간, 분으로 표기
13	CRSElapsedTime	예정 경과 시간, 분으로 표기
14	AirTime	방송시간, 분으로 표기
15	ArrDelay	도착 지연 시간, 분으로 표기
16	DepDelay	출발 지연 시간, 분으로 표기
17	Origin	출발지 공항 코드, IATA(국제 항공 운송 협회) 기준
18	Dest	도착지 공항 코드, IATA(국제 항공 운송 협회) 기준
19	Distance	비행 거리, 마일 기준
20	TaxiIn	비행기 바퀴가 지면에 닿아서(착륙) 목적지 공항의 게이트에 도착할 때까지 시간
21	TaxiOut	출발지 공항의 게이트에서 출발해서 바퀴가 지면에서 떨어질때(이륙)까지의 시간
22	Cancelled	비행 취소 여부 --> 1: 예, 0: 아니오
23	CancellationCode	비행 취소 코드 --> A: 항공사, B: 기상, C: NAS(National Airspace System), D: 보안
24	Diverted	우회 여부 --> 1: 예, 0: 아니오
25	CarrierDelay	항공사 지연 시간, 분으로 표기
26	WeatherDelay	기상 지연 시간, 분으로 표기
27	NASDelay	NAS 지연 시간, 분으로 표기
28	SecurityDelay	보안 지연 시간, 분으로 표기
29	LateAircraftDelay	연착 항공기 지연 시간, 분으로 표기

항공 출발 지연 데이터 분석 – 파일 및 실행

- DepartureDelayCountMapper.java
- DelayCountReducer.java
- DepartureDelayCount.java(메인 클래스)

- 분석용 데이터 업로드
  - hdfs dfs -put /home/hadoop/lab/data/ /airline

- 실행
  - hadoop jar DepartureDelayCount.jar /airline /output/departure_delay_count

- 결과 확인
  - hdfs dfs -cat /output/departure_delay_count/part-r-00000 | head -10
  - hdfs dfs -cat /output/departure_delay_count/part-r-00000 | tail -10

```
13/07/05 06:27:36 INFO mapred.JobClient: Job complete: job_201307031142_0018
13/07/05 06:27:36 INFO mapred.JobClient: Counters: 29
13/07/05 06:27:36 INFO mapred.JobClient:   Job Counters
13/07/05 06:27:36 INFO mapred.JobClient:     Launched reduce tasks=1
13/07/05 06:27:36 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=397537
13/07/05 06:27:36 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/07/05 06:27:36 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/07/05 06:27:36 INFO mapred.JobClient:     Launched map tasks=77
13/07/05 06:27:36 INFO mapred.JobClient:     Data-local map tasks=77
13/07/05 06:27:36 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=204708
13/07/05 06:27:36 INFO mapred.JobClient:   File Output Format Counters
13/07/05 06:27:36 INFO mapred.JobClient:     Bytes Written=1337
13/07/05 06:27:36 INFO mapred.JobClient:   FileSystemCounters
13/07/05 06:27:36 INFO mapred.JobClient:     FILE_BYTES_READ=371121872
13/07/05 06:27:36 INFO mapred.JobClient:     HDFS_BYTES_READ=5057043028
13/07/05 06:27:36 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=627769613
13/07/05 06:27:36 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=1337
13/07/05 06:27:36 INFO mapred.JobClient:   File Input Format Counters
13/07/05 06:27:36 INFO mapred.JobClient:     Bytes Read=5057034558
13/07/05 06:27:36 INFO mapred.JobClient:   Map-Reduce Framework
13/07/05 06:27:36 INFO mapred.JobClient:     Map output materialized bytes=252000495
13/07/05 06:27:36 INFO mapred.JobClient:     Map input records=52532308
13/07/05 06:27:36 INFO mapred.JobClient:     Reduce shuffle bytes=252000495
13/07/05 06:27:36 INFO mapred.JobClient:     Spilled Records=47136796
13/07/05 06:27:36 INFO mapred.JobClient:     Map output bytes=213900815
13/07/05 06:27:36 INFO mapred.JobClient:     Total committed heap usage (bytes)=36014587904
13/07/05 06:27:36 INFO mapred.JobClient:     CPU time spent (ms)=364240
13/07/05 06:27:36 INFO mapred.JobClient:     Combine input records=0
13/07/05 06:27:36 INFO mapred.JobClient:     SPLIT_RAW_BYTES=8470
13/07/05 06:27:36 INFO mapred.JobClient:     Reduce input records=19049609
13/07/05 06:27:36 INFO mapred.JobClient:     Reduce input groups=94
13/07/05 06:27:36 INFO mapred.JobClient:     Combine output records=0
13/07/05 06:27:36 INFO mapred.JobClient:     Physical memory (bytes) snapshot=39760900096
13/07/05 06:27:36 INFO mapred.JobClient:     Reduce output records=94
13/07/05 06:27:36 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=128878129152
13/07/05 06:27:36 INFO mapred.JobClient:     Map output records=19049609
```

항공 도착 지연 데이터 분석 – 파일 및 실행

- ArrivalDelayCountMapper.java
- DelayCountReducer.java
- ArrivalDelayCount.java(메인 클래스)

- 분석용 데이터 업로드
  - hdfs dfs -put /home/hadoop/lab/data/* /airline

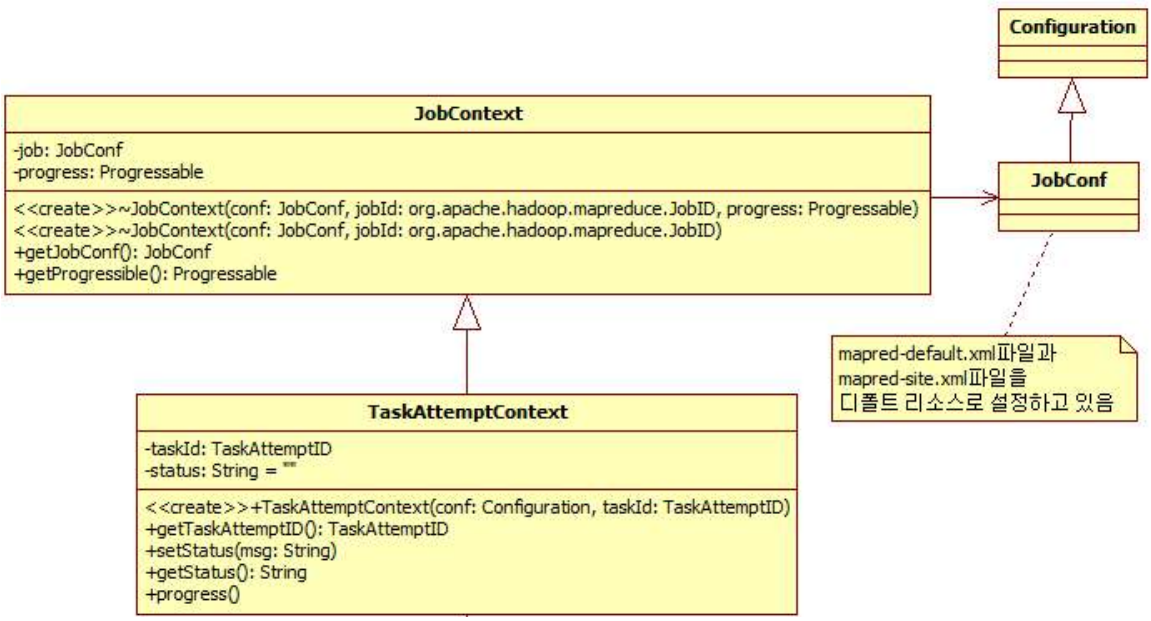
- 실행
  - hadoop jar ArrivalDelayCount.jar /airline /output/arrival_delay_count

- 결과 확인
  - hdfs dfs -cat /output/arrival_delay_count/part-r-00000 | head -10
  - hdfs dfs -cat /output/arrival_delay_count/part-r-00000 | tail -10

```
13/07/05 06:43:37 INFO mapred.JobClient:     Launched map tasks=77
13/07/05 06:43:37 INFO mapred.JobClient:     Data-local map tasks=77
13/07/05 06:43:37 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=205125
13/07/05 06:43:37 INFO mapred.JobClient:   File Output Format Counters
13/07/05 06:43:37 INFO mapred.JobClient:     Bytes Written=1338
13/07/05 06:43:37 INFO mapred.JobClient:   FileSystemCounters
13/07/05 06:43:37 INFO mapred.JobClient:     FILE_BYTES_READ=535590973
13/07/05 06:43:37 INFO mapred.JobClient:     HDFS_BYTES_READ=5057043028
13/07/05 06:43:37 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=833456954
13/07/05 06:43:37 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=1338
13/07/05 06:43:37 INFO mapred.JobClient:   File Input Format Counters
13/07/05 06:43:37 INFO mapred.JobClient:     Bytes Read=5057034558
13/07/05 06:43:37 INFO mapred.JobClient:   Map-Reduce Framework
13/07/05 06:43:37 INFO mapred.JobClient:     Map output materialized bytes=293219203
13/07/05 06:43:37 INFO mapred.JobClient:     Map input records=52532308
13/07/05 06:43:37 INFO mapred.JobClient:     Reduce shuffle bytes=293219203
13/07/05 06:43:37 INFO mapred.JobClient:     Spilled Records=62714471
13/07/05 06:43:37 INFO mapred.JobClient:     Map output bytes=248889113
13/07/05 06:43:37 INFO mapred.JobClient:     Total committed heap usage (bytes)=36095524864
13/07/05 06:43:37 INFO mapred.JobClient:     CPU time spent (ms)=380930
13/07/05 06:43:37 INFO mapred.JobClient:     Combine input records=0
13/07/05 06:43:37 INFO mapred.JobClient:     SPLIT_RAW_BYTES=8470
13/07/05 06:43:37 INFO mapred.JobClient:     Reduce input records=22164814
13/07/05 06:43:37 INFO mapred.JobClient:     Reduce input groups=94
13/07/05 06:43:37 INFO mapred.JobClient:     Combine output records=0
13/07/05 06:43:37 INFO mapred.JobClient:     Physical memory (bytes) snapshot=39788810240
13/07/05 06:43:37 INFO mapred.JobClient:     Reduce output records=94
13/07/05 06:43:37 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=128856899584
13/07/05 06:43:37 INFO mapred.JobClient:     Map output records=22164814
```

사용자 정의 옵션 사용

- org.apache.hadoop.util.GenericOptionsParser
  - 하둡 콘솔 명령에서 입력한 옵션을 분석합니다.
  - 네임노드, 잡트래커, 추가 구성 자원 등을 설정할 수 있는 옵션 제공
  - getRemainingArgs() : String[]
    - GenericOptionsParser에서 제공하는 파라미터를 제외한 나머지 파라미터를 반환
- org.apache.hadoop.util.Tool
  - GenericOptionsParser의 콘솔 설정 옵션을 지원하기 위한 인터페이스
  - interface Tool extends Configurable
  - 드라이버 클래스 정의 시 Configured를 상속받고, Tool 인터페이스를 구현
    - public class DelayCount extends Configured implements Tool
    - Configured 클래스는 환경설정 정보를 제어할 수 있게 합니다.
    - Tool 인터페이스는 사용자의 옵션을 조회할 수 있게 합니다.
- org.apache.hadoop.util.ToolRunner
  - Tool 인터페이스의 실행을 도와주는 헬퍼 클래스
  - GenericOptionsParser를 사용해 사용자가 콘솔 명령어에서 설정한 옵션을 분석하고, Configuration 객체에 설정합니다.
  - Configuration 객체를 Tool 인터페이스에 전달한 후, Tool 인터페이스의 run() 메서드를 실행합니다.
  - ToolRunner.run(Cunfiguration conf, Tool tool, String[] args) 메서드를 이용해 Tool의 run() 메서드를 실행합니다.



사용자 정의 옵션 사용 – 파일 및 실행

- DelayCountMapper.java
- DelayCountReducer.java
- DelayCount.java(메인 클래스)

- 분석용 데이터 업로드
  - hdfs dfs -put /home/hadoop/lab/data/ /airline

- 실행
  - hadoop jar DelayCount.jar -D workType=departure /airline /output/departure_delay_count2
  - hadoop jar DelayCount.jar -D workType=arrival /airline /output/arrival_delay_count2
- 결과 확인
  - hdfs dfs -cat /output/departure_delay_count2/part-r-00000 | head -10
  - hdfs dfs -cat /output/arrival_delay_count2/part-r-00000 | head -10

GenericOptionsParser는 하둡 콘솔 명령어에서 입력한 옵션을 분석합니다. 사용자가 하둡 콘솔 명령어에서 네임노드, 잡트래커 추가 구성 자원 등을 설정할 수 있는 여러 가지 옵션을 제공합니다.
GenericOptionsParser의 제공 옵션 중에서 -D <옵션=값>은 하둡 환경설정 파일에 있는 옵션에 새로운 값을 생성합니다.

다음은 옵션과 기능입니다.

옵션	기능
-conf [파일명]	명시한 파일을 환경설정에 있는 리소스 정보에 추가.
-D [옵션=값]	하둡 환경설정 파일에 있는 옵션에 새로운 값을 설정.
-fs [네임노드 호스트: 네임노드 포트]	네임노드를 새롭게 설정.
-jt [잡트래커 호스트: 잡트래커 포트]	잡트래커를 새롭게 설정.
-files [파일1,파일2,…,파일n]	로컬에 있는 파일을 HDFS에서 사용하는 공유 파일시스템으로 복사.
-libjars [Jar파일1,Jar파일2…,Jar파일n]	로컬에 있는 JAR파일을 HDFS에서 사용하는 공유 파일시스템으로 복사하고, 맵리듀스의 태스크 클래스패스에 추가.
-archives [아카이브파일1, 아카이브파일2,…,아카이브파일n]	로컬에 있는 아카이브 파일을 HDFS에서 사용하는 공유 파일 시스템으로 복사한 후 압축을 품.

사용자 정의 카운터 사용

- 맵리듀스 잡의 진행 상황을 모니터링 할 수 있게 Counter API 제공
  - 모든 잡은 다수의 내장 카운터를 가지고 있다.
- 맵리듀스 프레임워크는 개발자가 직접 카운터를 정의해서 사용할 수 있는 API를 제공합니다.
- JDK1.5에 추가된 enum 타입으로 카운터를 정의
  - public enum DelayCounters {
  - not_available_arrival, shceduled_arrival, ...
  - }
- 메퍼에서
  - context.getCounter(카운터열거값).increment(1);
  - context.getCounter(DelayCounter.scheduled)departure).increment(1);

모든 내장 카운터는 맵, 리듀스, 콤바이너의 입출력 레코드 건수와 바이트를 학인 할 수 있고, 몇 개의 맵 태스크와 리듀스 태스크가 실행되고 실패했는지, 또 파일 시스템에서는 얼마나 많은 바이트를 읽고 썼는가에 대한 정보를 제공합니다.
내장 카운터의 값은 잡을 실행하면 콘솔 화면에 출력되는 로그에 나타납니다.

카운터를 직접 정의해서 사용하면 카운터의 숫자를 직접 증/감 시킬 수 있기 때문에 맵과 리듀스의 동작을 체크할 때 유용하게 사용할 수 있습니다.
Job 실행 결과 메시지와 Web UI에서 간편하게 데이터를 확인할 수 있습니다.

```
13/07/03 12:19:22 INFO mapred.JobClient:  map 100% reduce 77%
13/07/03 12:19:26 INFO mapred.JobClient:  map 100% reduce 85%
13/07/03 12:19:29 INFO mapred.JobClient:  map 100% reduce 92%
13/07/03 12:19:32 INFO mapred.JobClient:  map 100% reduce 99%
13/07/03 12:19:33 INFO mapred.JobClient:  map 100% reduce 100%
13/07/03 12:19:34 INFO mapred.JobClient: Job complete: job_201307031142_0008
13/07/03 12:19:34 INFO mapred.JobClient: Counters: 35
13/07/03 12:19:34 INFO mapred.JobClient:   wikibooks.hadoop.chapter05.DelayCount
ers
13/07/03 12:19:34 INFO mapred.JobClient:     early_departure=24385728
13/07/03 12:19:34 INFO mapred.JobClient:     not_available_arrival=1164959
13/07/03 12:19:34 INFO mapred.JobClient:     scheduled_arrival=1948329
13/07/03 12:19:34 INFO mapred.JobClient:     not_available_departure=1056975
13/07/03 12:19:34 INFO mapred.JobClient:     early_arrival=27254198
13/07/03 12:19:34 INFO mapred.JobClient:     scheduled_departure=8039988
13/07/03 12:19:34 INFO mapred.JobClient:   Job Counters
13/07/03 12:19:34 INFO mapred.JobClient:     Launched reduce tasks=1
13/07/03 12:19:34 INFO mapred.JobClient:     SLOTS MILLIS MAPS=456015
```

사용자 정의 카운터 사용 – 파일 및 실행

- DelayCounters.java
- DelayCountMapperWithCounter.java
- DelayCountReducer.java
- DelayCountWithCounter.java(메인 클래스)

- 실행
  - hadoop jar DelayCountWithCounter.jar -D workType=departure /airline /output/departure_delay_count_counter
  - hadoop jar DelayCountWithCounter.jar -D workType=arrival /airline /output/arrival_delay_count_counter

- 결과 확인
  - hdfs dfs -cat /output/departure_delay_counter/part-r-00000 | head -10
  - hdfs dfs -cat /output/arrival_delay_counter/part-r-00000 | head -10

```
14/04/26 07:09:37 INFO mapreduce.Job: Job job_1398519263326_0002 completed successfully
14/04/26 07:09:38 INFO mapreduce.Job: Counters: 46
        File System Counters
                FILE: Number of bytes read=89023287
                FILE: Number of bytes written=179078702
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1392333697
                HDFS: Number of bytes written=390
                HDFS: Number of read operations=39
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=12
                Launched reduce tasks=1
                Data-local map tasks=12
                Total time spent by all maps in occupied slots (ms)=529330
                Total time spent by all reduces in occupied slots (ms)=52226
        Map-Reduce Framework
                Map input records=14462945
                Map output records=5845788
                Map output bytes=77331699
                Map output materialized bytes=89023347
                Input split bytes=1200
                Combine input records=0
                Combine output records=0
                Reduce input groups=24
                Reduce shuffle bytes=89023347
                Reduce input records=5845788
                Reduce output records=24
                Spilled Records=11691576
                Shuffled Maps =12
                Failed Shuffles=0
                Merged Map outputs=12
                GC time elapsed (ms)=93994
                CPU time spent (ms)=119080
                Physical memory (bytes) snapshot=2946314240
                Virtual memory (bytes) snapshot=5232365568
                Total committed heap usage (bytes)=2481979392
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        hadoop.sample.delaycount.DelayCounters
                early_departure=7163490
                not_available_departure=296994
                scheduled_departure=1156671
        File Input Format Counters
                Bytes Read=1392332497
        File Output Format Counters
                Bytes Written=390
## RESULT:0
```

```
14/04/26 07:12:39 INFO mapreduce.Job: Job job_1398519263326_0003 completed successfully
14/04/26 07:12:39 INFO mapreduce.Job: Counters: 46
        File System Counters
                FILE: Number of bytes read=97985247
                FILE: Number of bytes written=197002570
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1392333697
                HDFS: Number of bytes written=390
                HDFS: Number of read operations=39
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=12
                Launched reduce tasks=1
                Data-local map tasks=12
                Total time spent by all maps in occupied slots (ms)=515419
                Total time spent by all reduces in occupied slots (ms)=52931
        Map-Reduce Framework
                Map input records=14462945
                Map output records=6432959
                Map output bytes=85119317
                Map output materialized bytes=97985307
                Input split bytes=1200
                Combine input records=0
                Combine output records=0
                Reduce input groups=24
                Reduce shuffle bytes=97985307
                Reduce input records=6432959
                Reduce output records=24
                Spilled Records=12865918
                Shuffled Maps =12
                Failed Shuffles=0
                Merged Map outputs=12
                GC time elapsed (ms)=104004
                CPU time spent (ms)=119750
                Physical memory (bytes) snapshot=2738802688
                Virtual memory (bytes) snapshot=5243768832
                Total committed heap usage (bytes)=2423783424
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        hadoop.sample.delaycount.DelayCounters
                early_arrival=7309074
                not_available_arrival=332626
                scheduled_arrival=388284
        File Input Format Counters
                Bytes Read=1392332497
        File Output Format Counters
                Bytes Written=390
## RESULT:0
```

다수 파일 출력

- ◈ 한 번에 두 가지 작업을 병렬로 처리
  - ➢ 항공 출발 지연과 도착 지연을 병렬로 처리 가능
- ◈ org.apache.hadoop.mapreduce.lib.output.MultipleOutputs 이용
  - ➢ 멤버변수에 정의 후 setup() 메서드에서 객체 생성
  - ➢ mos = new MultipleOutputs<Text, IntWritable>(context);
- ◈ 리듀서에서 출력을 reduce() 메서드 인자로 넘겨진 Context의 write() 메서드를 이용하는 것이 아니고 MultipleOutputs 클래스의 write()메서드를 이용합니다.
  - ➢ public <K,V> void write(String namedOutput, K key, V value)
- ◈ 드라이버 클래스에서 addNamedOutput() 메서드를 이용해 출력 경로 생성
  - ➢ public static void addNamedOutput(Job job, String namedOutput, Class<? extends OutputFormat> outputFormatClass, Class<?> keyClass, Class<?> valueClass)

MultipleOutputs 클래스를 이용하면 구분 값(namedOutput)에 따른 다수 파일을 출력할 수 있습니다.
패키지를 포함하는 이름은 org.apache.hadoop.mapreduce.lib.output.MultipleOutputs 입니다.

MultipleOutputs 클래스는 여러 개의 출력 데이터를 쉽게 생성하도록 돕는 기능을 합니다.
MultipleOutputs는 여러 개의 OutputCollectors를 만들고 각 OutputCollectors에 대한 출력 경로, 출력 포맷, 키와 값 유형을 설정합니다.
MultipleOutputs에서 제공하는 static 메서드인 addNamedOutput 를 호출해서 설정합니다.

MultipleOutputs에서 출력하는 데이터는 기존 맵리듀스 잡에서 생성하는 데이터와는 별개로 생성됩니다. 일반적으로 맵리듀스 잡이 종료되면 리듀스 단계에서 part-r-nnnnn이라는 출력 데이터를 생성합니다. 그러나 리듀스 단계에서 MultipleOutputs를 이용해 namedOutput을 departure라고 지정할 경우, part-r-nnnnn 과 departure-r-nnnnn 이 동시에 생성됩니다.

다수 파일 출력 – 파일 및 실행

- DelayCounters.java
- DelayCountMapperWithMultipleOutputs.java
- DelayCountReducerWithMultipleOutputs.java
- DelayCountWithMultipleOutputs.java(메인 클래스)

- 실행
  - hadoop jar DelayCountWithMultipleOutputs.jar /airline /output/delay_count_m

- 결과 확인
  - hdfs dfs -cat /output/delay_count_m/arrival-r-00000 | head -10
  - hdfs dfs -cat /output/delay_count_m/departure-r-00000 | head -10

파일 이름이 만들어지는 규칙은 다음과 같습니다.
{namedOutput}-{m|r}-{part-number}

```
[hadoop@master Lab]$ hadoop fs -ls -R /output/delay_count_m
-rw-r--r--   1 hadoop supergroup          0 2014-04-26 07:18 /output/delay_count_m/_SUCCESS
-rw-r--r--   1 hadoop supergroup        342 2014-04-26 07:18 /output/delay_count_m/arrival-r-00000
-rw-r--r--   1 hadoop supergroup        342 2014-04-26 07:18 /output/delay_count_m/departure-r-00000
-rw-r--r--   1 hadoop supergroup          0 2014-04-26 07:18 /output/delay_count_m/part-r-00000
[hadoop@master Lab]$ hadoop fs -cat /output/delay_count_m/arrival-r-00000 | head -10
2007,1  286334
2007,10 270098
2007,11 242722
2007,12 332449
2007,2  284152
2007,3  293360
2007,4  273055
2007,5  275332
2007,6  326446
2007,7  326559
[hadoop@master Lab]$ hadoop fs -cat /output/delay_count_m/departure-r-00000 | head -10
2007,1  255777
2007,10 231129
2007,11 217557
2007,12 304011
2007,2  259288
2007,3  276261
2007,4  249097
2007,5  241699
2007,6  307986
2007,7  307864
```

체인

◈ 하나의 맵리듀스 잡에서 여러 개의 매퍼와 리듀서를 실행할 수 있게 체인매퍼(ChainMapper)와 체인리듀서(ChainReducer)를 제공합니다.

◈ ChainMapper 클래스의 메서드

- addMapper() : 체인 작업의 JobConf에 매퍼 클래스를 추가합니다.

◈ ChainReducer 클래스의 메서드

- setReducer() : 체인 작업의 JobConf에 리듀서 클래스를 설정합니다.
- addMapper() : 체인 작업의 JobConf에 매퍼 클래스를 추가합니다. 리듀서 클래스 설정 이후에 매퍼를 추가할 때 사용합니다.

◈ 1.0.x까지 org.apache.hadoop.mapred.lib패키지의 체인을 사용했지만 1.1.x에 org.apache.hadoop.mapreduce.lib.chain 패키지에 ChainMapper와 ChainReducer가 추가되었다.

org.apache.hadoop.mapreduce.lib.chain.ChainMapper

```
public static void addMapper(Job job,
        Class<? extends Mapper> klass,
        Class<?> inputKeyClass,
        Class<?> inputValueClass,
        Class<?> outputKeyClass,
        Class<?> outputValueClass,
        Configuration mapperConf)
            throws IOException
```

org.apache.hadoop.mapreduce.lib.chain.ChainReducer

```
public static void setReducer(Job job,
         Class<? extends Reducer> klass,
         Class<?> inputKeyClass,
         Class<?> inputValueClass,
         Class<?> outputKeyClass,
         Class<?> outputValueClass,
         Configuration reducerConf)
public static void addMapper(Job job,
        Class<? extends Mapper> klass,
        Class<?> inputKeyClass,
        Class<?> inputValueClass,
        Class<?> outputKeyClass,
        Class<?> outputValueClass,
        Configuration mapperConf)
            throws IOException
```

3 정렬 구연하기

Objectives

- 항공운항 데이터 분석을 통하여 맵리듀스 프로그래밍의 기초를 다집니다.

이 페이지는 여백 페이지입니다.

정렬

◈ 맵리듀스의 핵심 기능

◈ 하나의 리듀스 태스크만 실행되게 하면 정렬을 쉽게 해결할 수 있지만, 여러 데이터 노드가 구성된 상황에서 하나의 리듀스 태스크만 실행하는 것은 분산 환경의 장점을 살리지 않는 것입니다.

- 대량의 데이터를 정렬하게 될 경우 네트워크 부하도 상당함

◈ 하둡이 제공하는 정렬 방식

- 보조 정렬(Secondary Sort)
- 부분 정렬(Partial Sort)
- 전체 정렬(Total Sort)

보조 정렬은 복합키를 매퍼의 출력으로 사용할 때 참고하세요.
부분 정렬은 데이터 검색에 주로 사용합니다.
전체 정렬은 정렬된 데이터만 필요할 때 사용합니다.

보조 정렬(Secondary Sort)

◈ 키의 값들을 그룹핑하고, 그룹핑된 레코드에 순서를 부여하는 방식

◈ 구현 순서

- 기존 키의 값들을 조합한 복합키(Composite Key) 정의
  - 키의 값 중에서 그룹핑 키로 사용할 키 결정
- 복합키의 레코드를 정렬하기 위한 비교기(Comparator) 정의
- 그룹핑 키를 파티셔닝할 파티셔너(Partitioner) 정의
- 그룹핑 키를 정렬하기 위한 비교기(Comparator) 정의

실습 예제 파일

- DateKey.java
- DateKeyComparator.java
- DelayCounters.java
- DelayCountMapperWithDateKey.java
- DelayCountReducerWithDateKey.java
- DelayCountWithDateKey.java
- GroupKeyComparator.java
- GroupKeyPartitioner.java

실행

- hadoop jar DelayCountWithDateKey.jar /airline /output/delay_count_groupkey/

결과 확인

- hdfs dfs -cat /output/dealy_count_groupkey/arrival-r-00000
- hdfs dfs -cat /output/dealy_count_groupkey/departure-r-00000

보조 정렬(Secondary Sort) – 복합키 구현

- 키 값을 조합한 키 집합 클래스
- public interface WritableComparable<T> extends Writable, Comparable<T>
- org.apache.hadoop.io.WriatableComparable<T> 인터페이스를 구현
  - public void readFields(DataInput in) throws IOException
    - 입력 스트림에서 키 값을 조회하여 멤버변수에 저장(WritableUtils 이용)
  - public void write(DataOutput out) throws IOException
    - 멤버변수의 키 값을 출력 스트림에 출력(WritableUtils 이용)
  - public int compareTo(T key)
- 키 들을 멤버변수로 정의
- set/get 메서드 추가
- 생성자 추가
- toString() 재정의
  - 키(멤버변수)들을 ,(콤마)로 이어 문자열로 리턴하도록 구현
  - 매퍼와 리듀서에서는 복합키의 toString() 메서드를 호출해서 값을 출력

JavaSE의 Comparable 인터페이스

Set 계열 클래스에는 TreeSet, 그리고 Map계열 클래스에는 TreeMap 클래스를 이용하면 저장되는 엘리먼트들을 정렬시킬 수 있습니다. 그러나 List 계열 클래스에는 Tree 구조를 갖는 클래스가 존재하지 않습니다.

List 계열 컬렉션을 정렬시키기 위해서 Collections 클래스의 sort() 메서드를 이용할 수 있습니다. 그런데 sort() 메서드의 인자는 List 인터페이스를 구현한 클래스여야 하는데, 컬렉션에 저장되어 있는 sort의 대상이 되는 클래스는 Comparable 인터페이스를 구현한 클래스여야 합니다. Comparable 인터페이스를 구현할 때에는 compareTo() 메서드를 재정의 해야 합니다.

다음 코드는 ArrayList에 저장되어 있는 엘리먼트를 Collections.sort() 메서드를 이용하여 정렬하는 예 입니다. ArrayList에 저장되는 Tiger 클래스는 Comparable 인터페이스를 구현한 클래스입니다.

```
public class SortExample {
    public static void main(String[] args){
        List<Tiger> list = new ArrayList<>();
        list.add(new Tiger("라이거"));
        list.add(new Tiger("백호"));
        list.add(new Tiger("한라산"));
        list.add(new Tiger("백두"));

        Collections.sort(list);

        for(Tiger t : list) {
            System.out.println(t.name);
        }
    }
}
```

```
class Tiger implements Comparable<Tiger>{
    String name;

    Tiger(String name) {
        this.name = name;
    }

    public int compareTo(Tiger obj) {
        return this.name.compareTo(obj.name);
    }
}
```

보조 정렬(Secondary Sort) – 복합키 비교기 구현

- ◈ 복합키의 정렬 순서를 부여하기 위한 클래스
- ◈ public class WritableComparator implements RawComparator
- ◈ public interface RawComparator<T> extends Comparator<T>
- ◈ org.apache.hadoop.io.WritableComparator 클래스를 상속
  - ➢ public int compare(WritableComparator w1, WritableComparator w2)
    - ▪ 객체를 스트림에서 조회한 값을 비교하도록 구현되어 있습니다
      - • → 정확한 정렬 순서를 지정 할 수 없음
  - ➢ 정확한 정렬 순서를 정하기 위해 재정의 해야 함
    - ▪ 매개변수를 복합키 타입으로 형 변환 한 다음
    - ▪ 문자열은 String 클래스에서 제공하는 compareTo 메서드를 이용
    - ▪ 숫자는 ? : 연산자 이용
  - ➢ 매개변수로 전달된 두 값을 비교하여
    - ▪ 같은 경우에는 0을 리턴하며, 첫 번째 값이 두 번째 값보다 큰 경우에는 양수(1)를 리턴하하고, 두 번째 값이 큰 경우 음수(-1)를 리턴합니다.
  - ➢ 복합키 항목 전체를 비교

JavaSE의 Comparator 인터페이스

자바의 클래스들 중에서 TreeSet과 TreeMap은 엘리먼트를 저장할 때 자동으로 정렬시켜 저장합니다. 그런데 TreeSet이나 TreeMap에 저장하고자 하는 엘리먼트가 프로그래머가 정의하는 클래스일 경우에 무엇을 기준으로 정렬이 되어야 할지 생각해야 합니다. 엘리먼트를 정렬해야 할 상황이 발생하면 프로그래머는 클래스를 정의할 때 Comparator 인터페이스를 구현해 주면 됩니다. Comparator 인터페이스의 compare() 메서드는 매개변수로 전달된 두 값을 비교하여 같은 경우에는 0을 리턴하며, 첫 번째 값이 두 번째 값보다 큰 경우에는 양수를, 그렇지 않은 경우 음수를 리턴합니다. 만일 두 매개변수가 서로 다른 타입일 경우에는 ClassCastException이라는 예외를 던집니다. 그러므로 프로그래머는 컬렉션 계열 클래스 중에서 TreeSet이나 TreeMap에 저장할 엘리먼트 클래스를 선언할 경우 Comparator 인터페이스를 구현하여 compare() 메서드를 재정의 해야 합니다.

```
public class Employee implements Comparator<Employee> {
    String name;
    int salary;
    public Employee() {}        //기본생성자를 만들어 줘야 합니다.
    public Employee(String name, int salary) {
        super();
        this.name = name;
        this.salary = salary;
    }
    public int compare(Employee obj1, Employee obj2) {
        return obj1.name.compareTo(obj2.name);
    }
    public String toString() {
        return name + ":" + salary;
    }
}
```

```
TreeSet<Employee> list = new TreeSet<Employee>(new Employee());
```

보조 정렬(Secondary Sort) – 그룹키 파티셔너 구현

- ◈ 맵 리듀스 잡에서 사용
- ◈ 맵 태스크의 출력 데이터를 리듀스 태스크의 입력 데이터로 보낼지 결정
- ◈ 파티셔닝된 데이터는 맵 태스크의 출력 데이터의 키 값에 따라 정렬됨
- ◈ org.apache.hadoop.mapreduce.Partitioner<KEY, VALUE> 클래스를 상속받아 구현
  - ➢ public int getPartition(KEY key, VALUE value, int numPartitions) 재정의
    - KEY : 출력 데이터의 키
    - VALUE : 출력 데이터의 값

```
public int getPartition(DateKey key, IntWritable val, int numPartitions) {
    int hash = key.getYear().hashCode();
    int partition = hash % numPartitions;
    return partition;
}
```

생성자 안에서 super(DateKey.class, true); 로 키 클래스의 인스턴스 생성하도로 함
생성자 구현부

```
protected WritableComparator(Class<? extends WritableComparable> keyClass, boolean
createInstances) {
    this.keyClass = keyClass;
    if (createInstances) {
        key1 = newKey();
        key2 = newKey();
        buffer = new DataInputBuffer();
    } else {
        key1 = key2 = null;
        buffer = null;
    }
}
...
public WritableComparable newKey() {
    return ReflectionUtils.newInstance(keyClass, null);
}
```

보조 정렬(Secondary Sort) – 그룹키 비교기 구현

- 그룹키의 정렬 순서를 부여하기 위한 클래스
- public class WritableComparator implements RawComparator
- public interface RawComparator<T> extends Comparator<T>
- org.apache.hadoop.io.WritableComparator 클래스를 상속
  - public int compare(WritableComparator w1, WritableComparator w2)
  - 복합키에서 그룹에 사용할 키의 값만 비교하도록 구현

복합키 비교기가 연도와 월을 비교했다면 그룹키 비교기는 연도만 비교합니다.

보조 정렬(Secondary Sort) – 매퍼와 리듀서

◈ 매퍼
- extends Mapper<LongWritable, Text, 복합키, IntWritable>
- 매퍼에서는 출력키의 타입으로 복합키를 이용할 수 있습니다.
- map 메서드 안에서 value로 부터 조사한 값을 복합키의 set 메서드를 이용하여 출력키에 데이터를 저장합니다.

◈ 리듀서
- extends Reducer<복합키, IntWritable, 복합키, IntWritable>

부분 정렬(Partial Sort)

- ◈ 매퍼의 출력 데이터를 맵파일(MapFile)로 변경해서 데이터를 검색하는 방법
- ◈ 맵 태스크가 실행될 때 파티셔너는 매퍼의 출력 데이터가 어떤 리듀스 태스크로 전달될지 결정하고, 파티셔닝된 데이터는 키에 따라 정렬됩니다.
- ◈ 부분 정렬을 하기 위해 파티셔닝된 출력 데이터를 맵파일로 변경합니다.
- ◈ 특정 키에 대한 데이터를 검색할 경우 해당 키에 대한 데이터가 저장된 맵파일에 접근해서 데이터를 조회합니다.
- ◈ 부분 정렬은 데이터 검색에서 자주 사용됩니다.
- ◈ 부분 정렬 프로그래밍 절차
  - ➢ 입력 데이터를 시퀀스 파일로 생성
  - ➢ 시퀀스 파일을 맵파일로 변경
  - ➢ 맵파일에서 데이터 검색
- ◈ org.apache.hadoop.mapreduce.Job에서 org.apache.hadoop.mapred.MapFileOutputFormat을 출력 데이터로 쓸 수 없기 때문에 org.apache.hadoop.mapred 패키지를 이용해 맵리듀스 프로그램을 작성해야 합니다.

시퀀스 파일 생성 클래스
• SequenceFileCreator.java

시퀀스 파일 생성
• hadoop jar SequenceFileCreator.jar /airline/2008.csv /output/2008_seq_file
• hdfs dfs -ls /output/2008_seq_file
• hdfs dfs -text /output/2008_seq_file/part-00000 | head -15

맵파일 생성 클래스
• MapFileCreator.java
맵파일 생성
• hadoop jar MapFileCreator.jar /output/2008_seq_file /output/208_map_file
• hdfs dfs -ls /output/2008_map_file ← part-00000 디렉토리 확인
• hdfs dfs -ls /output/2008_map_file/part-00000 ← data와 index 파일 확인
• hdfs dfs -text /output/2008_map_file/part-00000/data | head -10

검색 프로그램
• SearchValueList.java
검색 프로그램 실행
• hadoop jar SearchValueList.jar /output/2008_map_file 100

부분 정렬(Partial Sort) – 시퀀스 파일 생성

- ◈ 항공 운항 지연 데이터를 시퀀스 파일로 출력
  - ➢ 다음 단계에서 맵파일로 변환하게 됨
- ◈ 매퍼를 내부 클래스 형태로 구현
- ◈ 리듀스가 필요 없으므로 리듀스 태스크 개수를 0으로 설정
  - ➢ conf.setNumReduceTasks(0);
- ◈ 출력 포맷을 SequenceFile로 설정
  - ➢ conf.setOutputFormat(SequenceFileOutputFormat.class);
- ◈ 시퀀스 파일 압축 포맷 설정
  - ➢ SequenceFileOutputFormat.setCompressOutput(conf, true);
  - ➢ SequenceFileOutputFormat.setOutputCompressorClass(conf, GzipCodec.class);
  - ➢ SequenceFileOutputFormat.setOutputCompressionType(conf, CompressionType.BLOCK);

시퀀스 파일 생성 클래스

• SequenceFileCreator.java

시퀀스 파일 생성

• hadoop jar SequenceFileCreator.jar /airline/2008.csv /output/2008_seq_file

• hdfs dfs -ls /output/2008_seq_file

• hdfs dfs -text /output/2008_seq_file/part-00000 | head -15

```
[hadoop@master Demo]$ hadoop fs -text /output/2008_seq_file/part-00000 | head -15
13/07/04 12:06:01 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/07/04 12:06:01 INFO zlib.ZlibFac      Successfully loaded & initialized native-zlib library
13/07/04 12:06:01 INFO compres                                    pressor
13/07/04 12:06:01 INFO compre                                     pressor
13/07/04 12:06:01 INFO compre                                     pressor
13/07/04 12:06:01 INFO compre                                     pressor
393     2008,1,3,4,1343,1325,                                    16,18,HOU,LIT,393,4,9,0,,0,16,0,0,0,0
441     2008,1,3,4,1125,1120,1247,1245,WN,1343,N523SW,82,85,71,2,5,HOU,MAF,441,3,8,0,,0,NA,NA,NA,NA,NA
441     2008,1,3,4,2009,2015,2136,2140,WN,3841,N280WN,87,85,71,-4,-6,HOU,MAF,441,2,14,0,,0,NA,NA,NA,NA,NA
848     2008,1,3,4,903,855,1203,1205,WN,3,N308SA,120,130,108,-2,8,HOU,MCO,848,5,7,0,,0,NA,NA,NA,NA,NA
848     2008,1,3,4,1423,1400,1726,1710,WN,25,N462WN,123,130,107,16,23,HOU,MCO,848,6,10,0,,0,16,0,0,0,0
848     2008,1,3,4,2024,2020,2325,2325,WN,51,N483WN,121,125,101,0,4,HOU,MCO,848,13,7,0,,0,NA,NA,NA,NA,NA
848     2008,1,3,4,1753,1745,2053,2050,WN,940,N493WN,120,125,107,3,8,HOU,MCO,848,6,7,0,,0,NA,NA,NA,NA,NA
848     2008,1,3,4,622,620,935,930,WN,2621,N266WN,133,130,107,5,2,HOU,MCO,848,7,19,0,,0,NA,NA,NA,NA,NA
937     2008,1,3,4,1944,1945,2210,2215,WN,389,N266WN,146,150,124,-5,-1,HOU,MDW,937,7,15,0,,0,NA,NA,NA,NA,NA
937     2008,1,3,4,1453,1425,1716,1650,WN,519,N514SW,143,145,124,26,28,HOU,MDW,937,6,13,0,,0,11,0,0,0,15
937     2008,1,3,4,2030,2015,2251,2245,WN,894,N716SW,141,150,122,6,15,HOU,MDW,937,11,8,0,,0,NA,NA,NA,NA,NA
937     2008,1,3,4,708,615,936,840,WN,969,N215WN,148,145,128,56,53,HOU,MDW,937,10,10,0,,0,53,0,3,0,0
937     2008,1,3,4,1749,1730,2039,2000,WN,2174,N623SW,170,150,128,39,19,HOU,MDW,937,34,8,0,,0,10,0,20,0,9
937     2008,1,3,4,1217,1215,1431,1440,WN,2445,N651SW,134,145,124,-9,2,HOU,MDW,937,4,6,0,,0,NA,NA,NA,NA,NA
937     2008,1,3,4,954,940,1206,1205,WN,2974,N447WN,132,145,120,1,14,HOU,MDW,937,5,7,0,,0,NA,NA,NA,NA,NA
```

결과는 cat 명령으로 확인할 수 없습니다. text 명령으로 조회해야 합니다.

부분 정렬(Partial Sort) – 맵파일 생성

◈ 맵파일은 키 값을 검색할 수 있게 색인(index)과 함께 정렬된 시퀀스 파일(data)입니다.
- 맵파일은 물리적으로 색인이 저장된 index 파일과 데이터 내용이 저장돼 있는 data 파일로 구성됩니다.

◈ HDFS에 저장돼 있는 시퀀스 파일을 변환해 맵파일로 생성할 수 있습니다.
- 시퀀스 파일을 입력 데이터로 전달받아 맵파일로 출력합니다.
- 맵파일을 생성하는 드라이버 클래스로 구현합니다.

◈ 드라이버 클래스
- 데이터를 분석 할 필요가 없기 때문에 매퍼와 리듀서를 설정하지 않습니다.
- 입력 데이터 포맷을 시퀀스 파일로 설정합니다.
  - conf.setInputFormat(SequenceFileInputFormat.class);
- 출력 포맷은 MapFileOutputFormat 으로 설정하고, 출력 키 값을 전입월로 설정합니다.
  - conf.setOutputFormat(MapFileOutputFormat.class);
  - conf.setOutputKeyClass(IntWritable.class);

맵파일 생성 클래스

• MapFileCreator.java

맵파일 생성

• hadoop jar MapFileCreator.jar /output/2008_seq_file /output/2008_map_file

• hdfs dfs -ls /output/2008_map_file ← part-00000 디렉토리 확인

• hdfs dfs -ls /output/2008_map_file/part-00000 ← data와 index 파일 확인

• hdfs dfs -text /output/2008_map_file/part-00000/data | head -10

• hdfs dfs -text /output/2008_map_file/part-00000/index | head -10

```
[hadoop@master Demo]$ hadoop fs -ls /output/2008_map_file/part-00000
Found 2 items
-rw-r--r--   1 hadoop supergroup  136132799 2013-07-04 12:09 /user/hadoop/output/2008_map_file/part-00000/data
-rw-r--r--   1 hadoop supergroup       7213 2013-07-04 12:09 /user/hadoop/output/2008_map_file/part-00000/index
[hadoop@master Demo]$ hadoop fs -text output/2008_map_file/part-00000/data | head -10
13/07/04 12:11:25 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/07/04 12:11:25 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
13/07/04 12:11:25 INFO compress.CodecPool: Got brand-new decompressor
13/07/04 12:11:25 INFO compress.CodecPool: Got brand-new decompressor
13/07/04 12:11:25 INFO compress.CodecPool: Got brand-new decompressor
13/07/04 12:11:25 INFO compress.CodecPool: Got brand-new decompressor
11      2008,10,2,4,1100,1010,1156,1110,OH,6402,N659BR,56,60,16,46,50,JFK,LGA,11,14,26,0,,0,46,0,0,0,0
11      2008,5,15,4,2037,1800,2125,1900,OH,4988,N806CA,48,60,31,145,157,JFK,LGA,11,10,7,0,,0,145,0,0,0,0
11      2008,8,10,7,1315,1220,1415,1320,OH,5572,N819CA,60,60,14,55,55,JFK,LGA,11,8,38,0,,0,55,0,0,0,0
17      2008,3,8,6,NA,1105,NA,1128,AA,1368,,NA,23,NA,NA,NA,EWR,LGA,17,NA,NA,1,B,0,NA,NA,NA,NA,NA
21      2008,2,8,5,NA,1910,NA,1931,AA,1668,,NA,21,NA,NA,NA,FLL,MIA,21,NA,NA,1,A,0,NA,NA,NA,NA,NA
21      2008,5,9,5,48,100,117,130,AA,588,N061AA,29,30,11,-13,-12,MIA,FLL,21,6,12,0,,0,NA,NA,NA,NA,NA
24      2008,3,12,3,955,931,1021,948,9E,2009,91619E,26,17,10,33,24,IAH,HOU,24,7,9,0,,0,0,0,9,0,24
24      2008,1,2,3,1245,1025,1340,1125,OH,5610,N806CA,55,60,11,135,140,IAD,DCA,24,5,39,0,,0,135,0,0,0,0
28      2008,2,22,5,2046,2050,NA,2156,OO,3698,N298SW,NA,66,NA,NA,-4,SLC,OGD,28,NA,12,0,,1,NA,NA,NA,NA,NA
30      2008,9,22,1,1340,1325,1553,1425,OH,6898,N710CA,133,60,27,88,15,HPN,JFK,30,12,94,0,,0,0,0,88,0,0
```

부분 정렬(Partial Sort) – 검색 프로그램 구현

◈ 맵리듀스 프로그램이 아닙니다.
- ➢ 맵파일에서 원하는 데이터만 검색하면 되기 때문에 맵리듀스로 구현할 필요가 없습니다.

◈ 컴색의 키는 파티셔너입니다.
- ➢ 검색하고자 하는 키가 속하는 파티션 번호를 조회한 후, 파티션 번호로 맵파일에 접근해 데이터를 검색합니다.

◈ 검색 프로그램
- ➢ 파일 시스템 객체 생성 후 MapFileOutputFormat.getReaders() 메서드를 이용해 해당 폴더의 맵 파일 조회
  - Reader[] readers = MapFileOutputFormat.getReaders(fs, path, getConf());
- ➢ 해시 파티셔너 생성(맵 파일의 키는 운항거리)
  - Partitioner<IntWritable, Text> partitioner = new HashPartitioner<IntWritable, Text>();
- ➢ 파티션 번호에 해당하는 Reader조회
  - Reader reader = readers[partitioner.getPartition(key, value, readers.length)];
- ➢ 검색 결과 확인 - 주어진 키에 해당하는 값을 검색
  - Writable entry = reader.get(key, value); ← 반환된 값은 데이터 목록의 첫 번째
  - reader.next(key, value) ← 다음 순서의 데이터로 위치 시키고, key와 value 파라미터에 현재 위치의 키와 값을 설정합니다.

데이터 조회(운항 거리가 100마일인 데이터 목록 조회)

• hadoop jar SearchValueList.jar /output/2008_map_file 100

```
[hadoop@master Demo]$ hadoop jar SearchValueList.jar /output/2008_map_file 100
Exception in thread "main" java.io.FileNotFoundException: File does not exist: hdfs://master:9000/user/hadoop/outp
ut/2008_map_file/_SUCCESS/data
        at org.apache.hadoop.hdfs.DistributedFileSystem.getFileStatus(DistributedFileSystem.java:528)
        at org.apache.hadoop.fs.FileSystem.getLength(FileSystem.java:796)
        at org.apache.hadoop.io.SequenceFile$Reader.<init>(SequenceFile.java:1479)
        at org.apache.hadoop.io.SequenceFile$Reader.<init>(SequenceFile.java:1474)
        at org.apache.hadoop.io.MapFile$Reader.createDataFileReader(MapFile.java:30
        at org.apache.hadoop.io.MapFile$Reader.open(MapFile.java:284)
        at org.apache.hadoop.io.MapFile$Reader.<init>(MapFile.java:273)
        at org.apache.hadoop.io.MapFile$Reader.<init>(MapFile.java:260)
        at org.apache.hadoop.io.MapFile$Reader.<init>(MapFile.java:253)
        at org.apache.hadoop.mapred.MapFileOutputFormat.getReaders(MapFileOutputFormat.java:93)
        at wikibooks.hadoop.chapter06.SearchValueList.run(SearchValueList.java:24)
        at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:65)
        at wikibooks.hadoop.chapter06.SearchValueList.main(SearchValueList.java:54)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:156)
```

MapFileOutputFormat의 getReaders() 메서드는 맵 파일 폴더의 로그 폴더를 체크해서 맵파일 없다는 오류를 발생시킵니다.

로그 폴더 삭제

• hdfs dfs -rmr /output/2008_map_file/_*

다시 실행

• hadoop jar SearchValueList.jar /output/2008_map_file 100

• hadoop jar SearchValueList.jar /output/2008_map_file 100 | head -10 ← 상위 10개 데이터만 확인

전체 정렬(Total Sort)

- ◈ 전체 정렬은 하나의 파티션만 사용해서 쉽게 해결할 수 있습니다.
- ◈ 전체 정렬은 단순히 정렬된 데이터만 필요할 때 사용합니다.
  - ➢ 부분 정렬은 데이터 검색 시 사용합니다.
- ◈ 처리 순서
  - ➢ 입력 데이터를 샘플링 해서 데이터의 분포도를 조사합니다.
  - ➢ 데이터의 분포도에 맞게 파티션 정보를 미리 생성합니다.
  - ➢ 미리 생성한 파티션 정보에 맞게 출력 데이터를 생성합니다.
  - ➢ 각 출력 데이터를 병합합니다.
- ◈ org.apache.hadoop.mapred.lib.TotalOrderPartitioner
  - ➢ 파티션 개수와 파티션에 저장할 데이터 범위 설정
  - ➢ 개발자가 데이터 분포를 잘못 계산해서 특정 파티션에 데이터가 집중되면 해당 파티션을 처리하는 데이터 노드에 부하가 걸립니다.
- ◈ org.apache.hadoop.mpared.lib.InputSampler
  - ➢ 입력 데이터에서 특정 개수의 데이터를 추출해서 키와 데이터 건수를 샘플링 합니다.← 데이터의 분포도를 작성하는 것입니다.

전체 정렬 클래스

• SequenceFileTotalSort.java

실행

• 부분 정렬에서 만들었던 시퀀스파일이 있어야 함

• hadoop jar SequenceFileTotalSort.jar /output/2008_seq_file /output/2008_total_sort

• hdfs dfs -text /output/2008_total_sort/part-00000 | head -10

```
[hadoop@master Demo]$ hadoop fs -ls /output/2008_tot_sort
Found 3 items
-rw-r--r--   1 hadoop supergroup          0 2013-07-04 13:35 /user/hadoop/output/2008_tot_sort/_SUCCESS
drwxr-xr-x   - hadoop supergroup          0 2013-07-04 13:34 /user/hadoop/output/2008_tot_sort/_logs
-rw-r--r--   1 hadoop supergroup  136132799 2013-07-04 13:34 /user/hadoop/output/2008_tot_sort/part-00000
[hadoop@master Demo]$ hadoop fs -text output/2008_tot_sort/part-00000 | head -10
13/07/04 13:36:26 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/07/04 13:36:26 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
13/07/04 13:36:26 INFO compress.CodecPool: Got brand-new decompressor
13/07/04 13:36:26 INFO compress.CodecPool: Got brand-new decompressor
13/07/04 13:36:26 INFO compress.CodecPool: Got brand-new decompressor
13/07/04 13:36:26 INFO compress.CodecPool: Got brand-new decompressor
11      2008,10,2,4,1100,1010,1156,1110,OH,6402,N659BR,56,60,16,46,50,JFK,LGA,11,14,26,0,,0,46,0,0,0,0
11      2008,5,15,4,2037,1800,2125,1900,OH,4988,N806CA,48,60,31,145,157,JFK,LGA,11,10,7,0,,0,145,0,0,0,0
11      2008,8,10,7,1315,1220,1415,1320,OH,5572,N819CA,60,60,14,55,55,JFK,LGA,11,8,38,0,,0,55,0,0,0,0
17      2008,3,8,6,NA,1105,NA,1128,AA,1368,,NA,23,NA,NA,NA,EWR,LGA,17,NA,NA,1,B,0,NA,NA,NA,NA,NA
21      2008,2,8,5,NA,1910,NA,1931,AA,1668,,NA,21,NA,NA,NA,FLL,MIA,21,NA,NA,1,A,0,NA,NA,NA,NA,NA
21      2008,5,9,5,48,100,117,130,AA,588,N061AA,29,30,11,-13,-12,MIA,FLL,21,6,12,0,,0,NA,NA,NA,NA,NA
24      2008,3,12,3,955,931,1021,948,9E,2009,91619E,26,17,10,33,24,IAH,HOU,24,7,9,0,,0,0,0,9,0,24
24      2008,1,2,3,1245,1025,1340,1125,OH,5610,N806CA,55,60,11,135,140,IAD,DCA,24,5,39,0,,0,135,0,0,0,0
28      2008,2,22,5,2046,2050,NA,2156,OO,3698,N298SW,NA,66,NA,NA,-4,SLC,OGD,28,NA,12,0,,1,NA,NA,NA,NA,NA
30      2008,9,22,1,1340,1325,1553,1425,OH,6898,N710CA,133,60,27,88,15,HPN,JFK,30,12,94,0,,0,0,0,88,0,0
```

4 병렬 조인

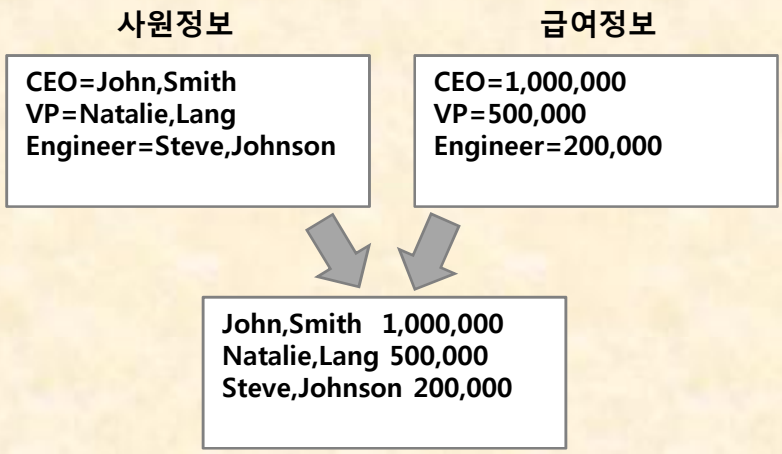
Objectives

- 맵사이드 조인과 리듀스 사이드 조인에 대하여 알아봅니다.

이 페이지는 여백 페이지입니다.

조인

◈ 두 개의 데이터 집합을 조인하여 결과를 얻을 수 있음
◈ 급여 정보를 출력할 때 사원의 이름과 급여정보를 출력하고 자 할 경우

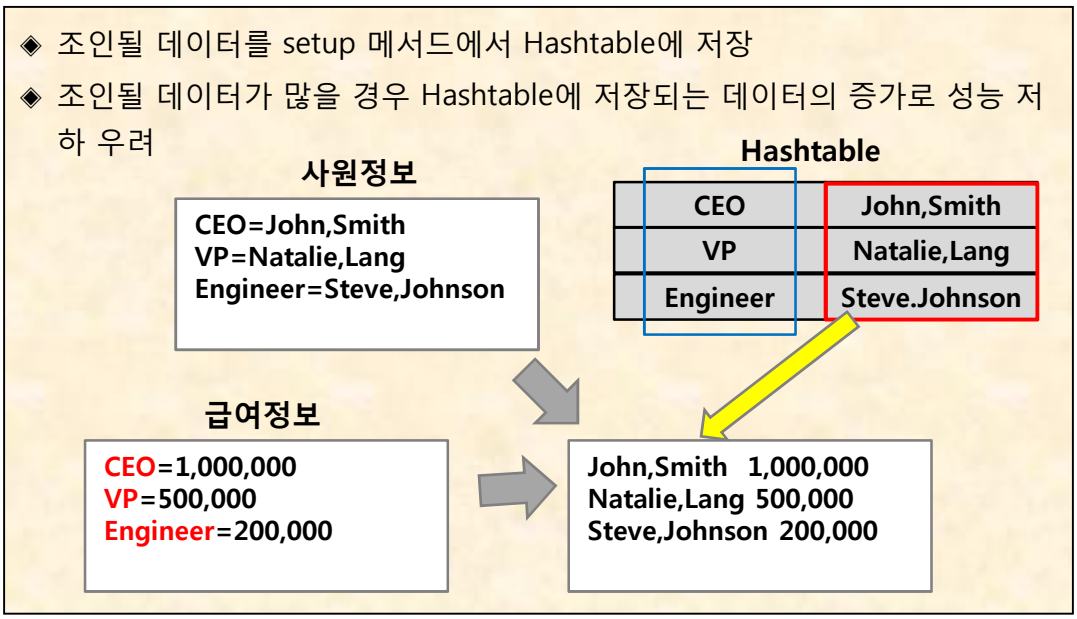


맵 사이드 조인과 리듀스 사이드 조인

◈ 맵 사이드 조인
  - ➢ setup 메서드에서 조인될 데이터를 준비합니다.
    - ▪ Hashtable을 전역변수로 선언하고 Hashtable에 데이터를 저장함
    - ▪ 읽어들일 데이터가 분산캐시에 등록되어 있어야 함
      - • 파일의 유형에 따라 등록하는 파일이 다름
  - ➢ map 메서드에서 write할 때 **Hashtable의 값을 키로 저장**합니다.
  - ➢ 실행하기 전에 분산캐시로 사용할 파일을 HDFS 에 업로드 해야 함

◈ 리듀스 사이드 조인
  - ➢ 두 개의 데이터를 키/값으로 출력합니다.
  - ➢ 조인될 키를 구분하기 위해 **키 뒤에 임의의 문자 추가해서 출력**
  - ➢ ex)
    - ▪ WN_A(항공운항통계 데이터의 항공사 코드)
    - ▪ WN_B(항공사 코드 데이터의 항공사 코드)
  - ➢ 리듀스에서 출력 시 **추가된 문자열에 따라 다른 키의 값을 키**로 저장
    - ▪ _A가 붙어있으면 키를 WN_B의 값으로 저장

맵 사이드 조인



자바에서 Hashtable 클래스는 Map 인터페이스를 구현한 클래스입니다. Map인터페이스는 Key와 Value 쌍으로 데이터를 저장할 수 있는 자료구조입니다.

맵사이드 조인은 다음 3단계 절차에 의해 해결할 수 있습니다.

1. Key/Value 쌍을 저장하기 위해 아래와 같이 멤버변수로 Hashtable 타입 변수를 선언할 수 있습니다.
   //멤버변수로 선언
   private Hashtable<String, String> joinMap = new Hashtable<String, String>();

2. 그리고 맵 사이드 조인을 이용할 경우 개발자는 setup() 메서드에서 조인에 필요한 데이터를 Hashtable에 저장해 놓을 수 있습니다.
   joinMap.put(아이디, 이름);
   아이디는 해시테이블의 key로 저장되며, 이름은 value로 저장됩니다.

3. 마지막으로 맵에 의해 key/value 쌍을 출력할 경우 맵의 키에 Hashtable에서 값을 빼와 저장합니다.
   outputKey.set(joinMap.get("아이디"));
   context.write(outputKey, value);

맵 사이드 조인

◈ 조인 데이터 준비
- http://stat-computing.org/dataexpo/2009/supplemental-data.html
- 미국 항공 코드 데이터 다운로드
- 항공기 코드와 항공기 이름 콤마(,)로 구분되어 있음

◈ 업로드
- 실행 하기 전에 다운로드 받은 데이터를 HDFS에 업로드 해야 합니다.
- hdfs dfs -put /home/hadoop/Lab/Data/carriers.csv /metadata/carriers.csv
  - 항공기 데이터
- hdfs dfs -put /home/hadoop/Lab/Data/2008.csv /airline/
  - 운항 통계 데이터(앞에서 이미 업로드 했음)

```
[hadoop@master Data]$ hadoop fs -mkdir /metadata
[hadoop@master Data]$ hadoop fs -put carriers.csv /metadata/
[hadoop@master Data]$ hadoop fs -ls /metadata
Found 1 items
-rw-r--r--   1 hadoop supergroup      43758 2014-04-26 07:45 /metadata/carriers.csv
[hadoop@master Data]$ hadoop fs -ls /airline
Found 8 items
-rw-r--r--   1 hadoop supergroup  600411462 2014-04-26 07:39 /airline/2001.csv
-rw-r--r--   1 hadoop supergroup  530507013 2014-04-26 07:40 /airline/2002.csv
-rw-r--r--   1 hadoop supergroup  626745242 2014-04-26 07:40 /airline/2003.csv
-rw-r--r--   1 hadoop supergroup  669879113 2014-04-26 07:41 /airline/2004.csv
-rw-r--r--   1 hadoop supergroup  671027265 2014-04-26 07:42 /airline/2005.csv
-rw-r--r--   1 hadoop supergroup  672068096 2014-04-26 07:42 /airline/2006.csv
-rw-r--r--   1 hadoop supergroup  702878193 2014-03-25 17:28 /airline/2007.csv
-rw-r--r--   1 hadoop supergroup  689413344 2014-03-22 22:18 /airline/2008.csv
[hadoop@master Data]$
```

실습 파일
MapperWithMapsideJoin.java
MapsideJoin.java

실행
$ hadoop jar MapsideJoin.jar /metadata/carriers.csv /airline /output/mapside_join
$ hdfs dfs -ls /output/mapside_join
$ hdfs dfs -cat /output/mapside_join/part-m-00000

항공 운항 데이터에서 9번째 열의 값이 UA일 경우 출력 데이터에는 United Air Lines Inc.가 됩니다.

맵 사이드 조인 – 매퍼 클래스

```
1.  package hadoop.sample.join;
2.
3.  import java.io.BufferedReader;
4.  import java.io.FileReader;
5.  import java.io.IOException;
6.  import java.util.Hashtable;
7.
8.  import org.apache.hadoop.filecache.DistributedCache;
9.  import org.apache.hadoop.fs.Path;
10. import org.apache.hadoop.io.LongWritable;
11. import org.apache.hadoop.io.Text;
12. import org.apache.hadoop.mapreduce.Mapper;
13.
14. public class MapperWithMapsideJoin extends
15.         Mapper<LongWritable, Text, Text, Text> {
16.
17.     //조인되는 데이터를 저장하기 위한 변수 선언<항공사코드, 항공사이름>
18.     private Hashtable<String, String> joinMap = new Hashtable<String, String>();
19.
20.     // map 출력키
21.     private Text outputKey = new Text();
22.
23.     @Override
24.     public void setup(Context context) throws IOException, InterruptedException {
25.         try {
26.             // 분산캐시 조회
27.             Path[] cacheFiles = DistributedCache.getLocalCacheFiles(
    context.getConfiguration() );
28.
29.             // 조인 데이터 생성, 캐시 된 데이터를 읽어온다
30.             if (cacheFiles != null && cacheFiles.length > 0) {
31.
32.                 String line;    //라인
33.                 String[] tokens;//한 라인에 있는 데이터를 공백으로 분리하여 저장
34.
35.                 //첫 번째 캐시 된 파일로부터 데이터를 읽겠다.
36.                 //cacheFiles[0]은 MapsideJoin.java에서 addCache한 것 중 첫 번째 꺼
37.                 BufferedReader br = new BufferedReader( new FileReader(
    cacheFiles[0].toString() ) );
38.
```

맵 사이드 조인 – 매퍼 클래스

```
39.                 try {
40.                     //입력스트림(br)에서 한 라인 읽어서 line 변수에 저장.
41.                     //그 라인인 널이 아닐 때 까지
42.                     while ((line = br.readLine()) != null) {
43.                         //한 라인을 문자열로 만들고 "를 없애고,
44.                         //콤마로 분리하여 배열 tokens에 저장
45.                         tokens = line.toString().replaceAll("₩"", "").split(",");
46.
47.                         //첫번째 토큰과 두번재 토큰을 메모리(hashtable)에 저장
48.                         joinMap.put(tokens[0], tokens[1]);
49.                     }
50.                 } finally {
51.                     br.close();//입력 스트림 닫기
52.                 }
53.             } else {
54.                 System.out.println("### cache files is null!");
55.             }
56.         } catch (IOException e) {
57.             e.printStackTrace();
58.         }
59.     }
60.
61.     public void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
62.
63.         if (key.get() > 0) {
64.             // 콤마 구분자 분리
65.             String[] colums = value.toString().split(",");
66.             if (colums != null && colums.length > 0) {
67.                 try {
68.                     outputKey.set(joinMap.get(colums[8]));
69.                     context.write(outputKey, value);
70.                 } catch (Exception e) {
71.                     e.printStackTrace();
72.                 }
73.             }
74.         }
75.     }
76. }
```

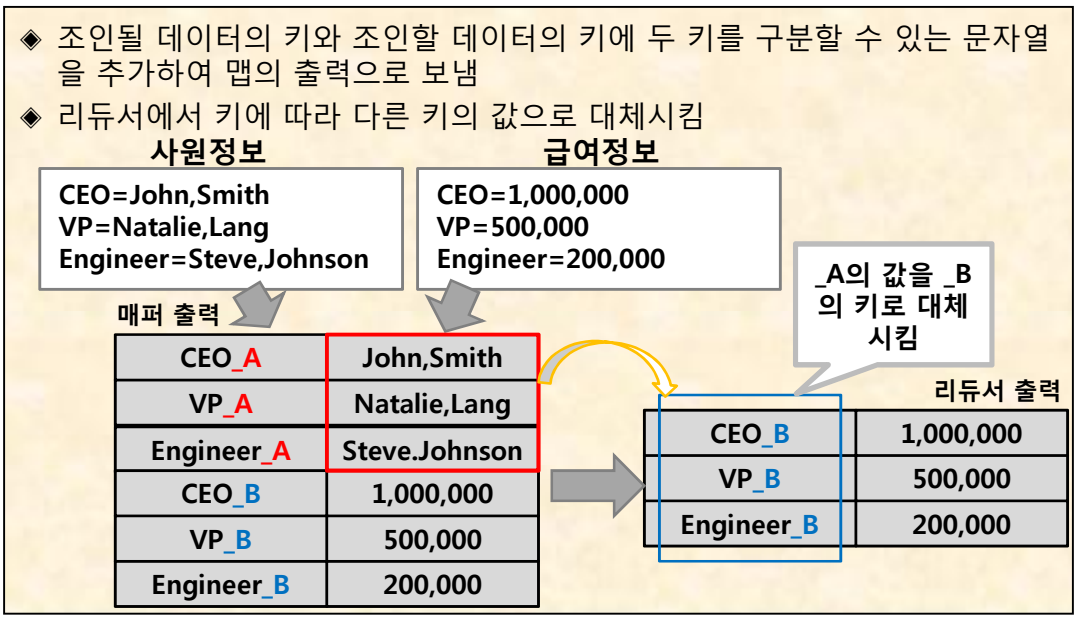
맵 사이드 조인 – 드라이버 클래스

```
1.  package hadoop.sample.join;
2.
3.  import org.apache.hadoop.conf.Configuration;
4.  import org.apache.hadoop.conf.Configured;
5.  import org.apache.hadoop.filecache.DistributedCache;
6.  import org.apache.hadoop.fs.Path;
7.  import org.apache.hadoop.io.Text;
8.  import org.apache.hadoop.mapreduce.Job;
9.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
10. import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
13. import org.apache.hadoop.util.GenericOptionsParser;
14. import org.apache.hadoop.util.Tool;
15. import org.apache.hadoop.util.ToolRunner;
16.
17. public class MapsideJoin extends Configured implements Tool {
18.
19.     public int run(String[] args) throws Exception {
20.
21.         // 입출력 데이터 경로 확인
22.         Path distPath = new Path("/user/hadoop/metadata/carriers.csv");
23.
24.         Path inputPath = new Path("/user/hadoop/input");
25.         Path outputPath = new Path("/user/hadoop/output/map_join");
26.
27.         // Job 이름 설정
28.         Job job = new Job(getConf(), "MapsideJoin");
29.
30.         // 분산 캐시 설정 meta_data/carriers.csv파일을 분산 캐시로 설정
31.         //파일의 타입에 따라 분산캐시를 설정하는 메서드가 다름
32.         DistributedCache.addCacheFile(distPath.toUri(), job.getConfiguration());
33.
34.         // 입출력 데이터 경로 설정
35.         FileInputFormat.addInputPath(job, inputPath);
36.         FileOutputFormat.setOutputPath(job, outputPath);
37.
38.         // Job 클래스 설정
39.         job.setJarByClass(MapsideJoin.class);
40.
```

맵 사이드 조인 – 드라이버 클래스

```
41.         // Mapper 설정
42.         job.setMapperClass(MapperWithMapsideJoin.class);
43.         // Reducer 설정
44.         job.setNumReduceTasks(0);//리듀스 할 것이 없으니까 태스크 수를 0으로 설정
45.
46.         // 입출력 데이터 포맷 설정
47.         job.setInputFormatClass(TextInputFormat.class);
48.         job.setOutputFormatClass(TextOutputFormat.class);
49.
50.         // 출력키 및 출력값 유형 설정
51.         job.setOutputKeyClass(Text.class);
52.         job.setOutputValueClass(Text.class);
53.
54.         job.waitForCompletion(true);
55.         return 0;
56.     }
57.
58.     public static void main(String[] args) throws Exception {
59.         // ToolRunner 를 이용하여 실행
60.         int res = ToolRunner.run(new Configuration(), new MapsideJoin(), args);
61.         System.out.println("## RESULT:" + res);
62.     }
63. }
```

리듀스 사이드 조인



조인될 데이터의 키와 조인할 데이터의 키에 두 키를 구분할 수 있는 문자열을 추가하여 맵의 출력으로 보낸 다음 리듀서에서 추가된 문자열에 따라 다른 키의 값을 대체시킵니다.
리듀스사이드 조인은 다음 절차에 의해 해결할 수 있습니다.

1. 각각의 데이터에 대하여 매퍼를 작성합니다. 이때 매퍼의 출력키를 구분할 수 있는 문자열을 추가해 줍니다. 아래의 코드는

```
//사원정보 매퍼
outputKey.set(아이디 + "_A");
outputValue.set(이름);
context.write(outputKey, outputValue);
//급여정보 매퍼
outputKey.set(colums[8] + "_B");
context.write(outputKey, value);
```

2. 리듀서에서는 조인될 키를 조인할 키의 값으로 대체하여 출력합니다.

```
if (tagValue.equals("_A")) {
    outputKey.set(value);
 } else if (tagValue.equals("_B")) {
     outputValue.set(value);
     context.write(outputKey, outputValue);
 }
```

리듀스 사이드 조인 – 매퍼 클래스(CarrierCodeMapper)

```
1.  package hadoop.sample.join;
2.
3.  import java.io.IOException;
4.
5.  import org.apache.hadoop.io.LongWritable;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.Mapper;
8.
9.  public class CarrierCodeMapper extends Mapper<LongWritable, Text, Text, Text> {
10.
11.     private Text outputKey = new Text();
12.     private Text outputValue = new Text();
13.
14.     public void map(LongWritable key, Text value, Context context)
15.             throws IOException, InterruptedException {
16.         if (key.get() > 0) {
17.             String[] colums = value.toString().replaceAll("₩"", "").split(",");
18.             if (colums != null && colums.length > 0) {
19.                 outputKey.set(colums[0] + "_A");
20.                 outputValue.set(colums[1]);
21.                 context.write(outputKey, outputValue);
22.             }
23.         }
24.     }
25. }
```

리듀스 사이드 조인 – 매퍼 클래스(MapperWithReducesideJoin)

```
1.  package hadoop.sample.join;
2.
3.  import java.io.IOException;
4.
5.  import org.apache.hadoop.io.LongWritable;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.Mapper;
8.
9.  public class MapperWithReducesideJoin extends Mapper<LongWritable, Text, Text, Text> {
10.
11.     // map 출력키
12.     private Text outputKey = new Text();
13.
14.     public void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
15.
16.         if (key.get() > 0) {
17.             // 콤마 구분자 분리
18.             String[] colums = value.toString().split(",");
19.             if (colums != null && colums.length > 0) {
20.                 try {
21.                     outputKey.set(colums[8] + "_B");
22.                     context.write(outputKey, value);
23.                 } catch (Exception e) {
24.                     e.printStackTrace();
25.                 }
26.             }
27.         }
28.     }
29. }
```

리듀스 사이드 조인 – 리듀서 클래스

```
1.  package hadoop.sample.join;
2.
3.  import java.io.IOException;
4.
5.  import org.apache.hadoop.io.Text;
6.  import org.apache.hadoop.mapreduce.Reducer;
7.
8.  public class ReducerWithReducesideJoin extends Reducer<Text, Text, Text, Text> {
9.
10.     // reduce 출력키
11.     private Text outputKey = new Text();
12.     private Text outputValue = new Text();
13.
14.     public void reduce(Text key, Iterable<Text> values, Context context)
15.             throws IOException, InterruptedException {
16.         // 태그 조회
17.         String tagValue = key.toString().split("_")[1];
18.
19.         for (Text value : values) {
20.             // 출력키 설정
21.             if (tagValue.equals("_A")) {
22.                 outputKey.set(value);
23.                 // 출력값 설정 및 출력 데이터 생성
24.             } else if (tagValue.equals("_B")) {
25.                 outputValue.set(value);
26.                 context.write(outputKey, outputValue);
27.             }
28.         }
29.     }
30. }
```

리듀스 사이드 조인 – 드라이버 클래스

```
1.  package hadoop.sample.join;
2.
3.  import org.apache.hadoop.conf.Configuration;
4.  import org.apache.hadoop.conf.Configured;
5.  import org.apache.hadoop.fs.Path;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.Job;
8.  import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12. import org.apache.hadoop.util.GenericOptionsParser;
13. import org.apache.hadoop.util.Tool;
14. import org.apache.hadoop.util.ToolRunner;
15.
16. public class ReducesideJoin extends Configured implements Tool {
17.
18.     public int run(String[] args) throws Exception {
19.         String[] otherArgs = new GenericOptionsParser(getConf(), args)
20.                 .getRemainingArgs();
21.         // 입력출 데이터 경로 확인
22.         if (otherArgs.length != 3) {
23.             System.err.println("Usage: ReducesideJoin <metadata> <in> <out>");
24.             System.exit(2);
25.         }
26.
27.         // Job 이름 설정
28.         Job job = new Job(getConf(), "ReducesideJoin");
29.
30.         // 출력 데이터 경로 설정
31.         FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));
32.
33.         // Job 클래스 설정
34.         job.setJarByClass(ReducesideJoin.class);
35.         // Reducer 설정
36.         job.setReducerClass(ReducerWithReducesideJoin.class);
37.
38.         // 입출력 데이터 포맷 설정
39.         job.setInputFormatClass(TextInputFormat.class);
40.         job.setOutputFormatClass(TextOutputFormat.class);
```

리듀스 사이드 조인 – 드라이버 클래스

```
41.         // 출력키 및 출력값 유형 설정
42.         job.setOutputKeyClass(Text.class);
43.         job.setOutputValueClass(Text.class);
44.
45.         // MultipleInputs 설정
46.         MultipleInputs.addInputPath(job, new Path(otherArgs[0]),
47.                 TextInputFormat.class, CarrierCodeMapper.class);    //항공사 코드 데이터
48.
49.         MultipleInputs.addInputPath(job, new Path(otherArgs[1]),
50.                 TextInputFormat.class, MapperWithReducesideJoin.class);//운항 스케쥴
51.
52.         job.waitForCompletion(true);
53.         return 0;
54.     }
55.
56.     public static void main(String[] args) throws Exception {
57.         // Tool 인터페이스 실행
58.         int res = ToolRunner.run(new Configuration(), new ReducesideJoin(), args);
59.         System.out.println("## RESULT:" + res);
60.     }
61. }
```

이 페이지는 여백 페이지입니다.

5 맵리듀스와 RDBMS

Objectives

- 맵리듀스 어플리케이션에서 관계형 데이터베이스를 어떻게 사용할 수 있는 지 살펴봅니다.

이 페이지는 여백 페이지입니다.

데이터베이스 처리

- ◈ 맵리듀서는 HDFS의 파일을 처리
- ◈ 데이터베이스의 자료는 텍스트 파일로 변환한 다음 HDFS에 저장한 후 처리
- ◈ 데이터베이스의 내용을 읽어 처리하려고 한다면 Mapper 클래스의 setup() 메서드와 cleanup() 메서드를 이용
- ◈ setup() 메서드와 cleanup() 메서드 원형
  - ➢ protected void cleanup(Context context) throws IOException, InterruptedException
  - ➢ protected void setup(Context context) throws IOException, InterruptedException

하둡의 맵리듀서는 기본적으로 파일 시스템의 파일을 처리합니다.

만일 데이터베이스의 자료를 처리하려면 먼저 데이터베이스의 자료를 텍스트 파일로 변환한 다음 HDFS에 저장해야 합니다.

그렇지 않고 직접 데이터베이스의 내용을 읽어 처리하려고 한다면 Mapper 클래스의 setup() 메서드와 cleanup() 메서드를 이용하여 데이터베이스 접속/종료를 수행해야 합니다.

다음은 setup() 메서드와 cleanup() 메서드 원형입니다.
protected void cleanup(Context context) throws IOException, InterruptedException
protected void setup(Context context) throws IOException, InterruptedException

postgresql jdbc driver 파일을
JAVA_HOME/jre/lib/ext 폴더에 복사해야 합니
다.

heap size로 오류발생할 때
mapred-site.xml 파일에서
-Xmx 크기를 늘려준다.(예
제 DB는 780M정도 소요됨)

데이터베이스 처리

```
1.  package hadoop.sample.database;
2.
3.  import java.io.IOException;
4.  import java.sql.*;
5.  import org.apache.hadoop.io.*
6.  import org.apache.hadoop.mapreduce.Mapper;
7.
8.  public class FlightHistoryMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
9.
10.     private final static IntWritable one = new IntWritable(1);
11.     private Text word = new Text();
12.     Connection conn = null;                 //데이터베이스에 연결하기 위한 객체
13.     ResultSet rs = null;
14.
15.     @Override
16.     protected void cleanup(Context context) throws IOException, InterruptedException {
17.         //Connection close, 반드시 닫아줘야 함
18.         if(conn!=null) try { conn.close(); }catch(Exception e){}
19.     }
20.
21.     @Override
22.     protected void setup(Context context) throws IOException, InterruptedException {
23.         String url = "jdbc:postgresql://127.0.0.1:5432/hadooptestdb";
24.         String userName = "scott";
25.         String password = "tiger";
26.
27.         try{
28.             //드라이버 로딩
29.             Class.forName("org.postgresql.Driver"); //specify the JDBC driver for PostgreSQL
30.             //url, id, password로 데이터베이스 접속 객체를 생성
31.             conn = DriverManager.getConnection(url, userName, password);
32.             System.out.println("Connection properly established");
33.             //flight_history 테이블에서 모든 비행의 비행 월 정보만 조회
34.             String sql = "select month from ontime";
35.             PreparedStatement stmt = conn.prepareStatement(sql);
36.             rs = stmt.executeQuery(); //쿼리문을 데이터베이스에 실행하도록 함
37.         }catch (Exception e){
38.             e.printStackTrace();//예외 발생시 예외 경로를 추적해서 화면에 나타냄
39.         }
40.     }
```

데이터베이스 처리

```
40.     public void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
41.         try {
42.             while(rs.next()) {
43.                 word.set(rs.getInt(1)+"");        //비행 월을 키 값으로 지정.
44.                 context.write(word, one);         //키는 비행 월, 값은 1로 하여 출력
45.             }
46.         } catch (SQLException e) {
47.             e.printStackTrace();
48.         }
49.     }//end map()
50. }//end class
```

```
1.  package hadoop.sample.database;
2.
3.  import java.io.IOException;
4.  import org.apache.hadoop.io.*;
5.  import org.apache.hadoop.mapreduce.Reducer;
6.
7.  public class FlightHistoryReducer extends Reducer<Text, IntWritable, Text, IntWritable>{
8.      private IntWritable result = new IntWritable();//?
9.
10.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
11.     throws IOException, InterruptedException {
12.         int sum = 0;
13.         for(IntWritable val : values) {
14.             //mapper의 결과에서 value(1의 개수를 모두 합함
15.             sum += val.get();
16.         }
17.         result.set(sum);
18.         context.write(key, result);
19.     }
20. }
```

데이터베이스 처리

```
1.  package hadoop.sample.database;
2.  import org.apache.hadoop.conf.Configuration;
3.  import org.apache.hadoop.fs.FileSystem;
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.io.IntWritable;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.Job;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class FlightHistory {
14.     public static void main(String[] args) throws Exception {
15.         Configuration conf = new Configuration();
16.         Job job = new Job(conf, "FlightHistory");
17.
18.         job.setJarByClass(FlightHistory.class);                    //
19.         job.setMapperClass(FlightHistoryMapper.class);       //Mapper 클래스 지정
20.         job.setReducerClass(FlightHistoryReducer.class);     //Reducer 클래스 지정
21.
22.         job.setInputFormatClass(TextInputFormat.class);      //입력 데이터 형식 지정
23.         job.setOutputFormatClass(TextOutputFormat.class); //출력 데이터 형식 지정
24.
25.         job.setOutputKeyClass(Text.class);                   //출력 키 타입 지정
26.         job.setOutputValueClass(IntWritable.class);          //출력 값 타입 지정
27.
28.         FileInputFormat.addInputPath(job, new Path("dummy.txt"));  //입력 패스 지정
29.
30.         FileSystem hdfs = FileSystem.get(conf);  //HDFS을 다루기 위해
31.         Path path = new Path("output");      //패스 지정(출력 패스로 사용하기 위함)
32.         if(hdfs.exists(path)) {
33.             hdfs.delete(path, true);             //디렉토리 삭제, 반드시 true를 줘야 함
34.             System.out.println("existing file removed");
35.         }
36.         FileOutputFormat.setOutputPath(job, path); //출력 패스 지정
37.
38.         job.waitForCompletion(true);          //작업 수행하도록 지시
39.     }
40. }
```

6 Custom File Output in Hadoop

Objectives

- 사용자가 원하는 형식으로 맵리듀스 출력 파일을 만들 수 있습니다.

이 페이지는 여백 페이지입니다.

Reduce Output Value 클래스

- ◈ 출력 값으로 사용할 클래스
- ◈ 메모리로 수용하지 못할 정도로 많은 양을 처리해야 한다면 Writable 인터페이스를 구현해야 함
- ◈ 직렬화 가능한 클래스여야 함
  - ➢ implements java.io.Serializable
- ◈ 멤버변수의 타입은 기본 데이터 타입이 아닌 Wrapper 클래스를 이용해야 함
- ◈ setter/getter 메서드를 포함해야 함
- ◈ 맵에서 출력되는 데이터가 많을 경우에는 Writable 인터페이스를 구현해야 한다.

```
1.  import java.io.*;
2.  import org.apache.hadoop.io.Writable;
3.
4.  public class DelayValue implements Writable, Serializable {
5.
6.      private static final long serialVersionUID = 3569721647402802809L;
7.
8.      private Integer count;
9.      private Integer value;
10.
11.     public Integer getCount() {
12.         return count;
13.     }
14.     public void setCount(Integer count) {
15.         this.count = count;
16.     }
17.     public Integer getValue() {
18.         return value;
19.     }
20.     public void setValue(Integer value) {
21.         this.value = value;
22.     }
23.     @Override
24.     public void readFields(DataInput in) throws IOException {
25.         count = in.readInt();
26.         value = in.readInt();
27.     }
28.     @Override
29.     public void write(DataOutput out) throws IOException {
30.         out.writeInt(count);
31.         out.writeInt(value);
32.     }
33. }
```

RecordWriter 클래스

◈ 출력 클래스

◈ RecordWriter<KEY, VALUE> 클래스를 상속받아 구현한다.

- 생성자에서 출력 파일의 헤더를 지정할 수 있다.
- public void write(KEY, VALUE) 메서드 재정의 하여 출력 레코드가 어떤 형식으로 출력되게 할 것인지를 프로그래밍 할 수 있다.
- public void close(TaskAttemptContext context) 메서드를 재정의하여 출력 스트림을 닫는다.

```
1.  import java.io.DataOutputStream;
2.  import java.io.IOException;
3.
4.  import org.apache.hadoop.io.Text;
5.  import org.apache.hadoop.mapreduce.RecordWriter;
6.  import org.apache.hadoop.mapreduce.TaskAttemptContext;
7.
8.  public class DelayValueWriter extends RecordWriter<Text, DelayValue> {
9.
10.     private DataOutputStream out;
11.
12.     public DelayValueWriter() {  }
13.     public DelayValueWriter(DataOutputStream stream) {
14.         this.out = stream;
15.         try {
16.             out.writeBytes("year,month,delaycount,delaytime₩r₩n");
17.         } catch (Exception ex) {
18.         }
19.     }
20.
21.     @Override
22.     public void write(Text key, DelayValue value)
23.             throws IOException, InterruptedException {
24.         out.writeBytes(key.toString() + ",");
25.         out.writeBytes(value.getCount() + ",");
26.         out.writeBytes(value.getValue() + "₩r₩n");
27.     }
28.
29.     @Override
30.     public void close(TaskAttemptContext arg0) throws IOException,
31.             InterruptedException {
32.         out.close();
33.     }
34. }
```

OutputFormat 클래스

◈ 리듀서의 출력 포맷 클래스

◈ 드라이버클래스에서 잡(Job)의 출력 포맷 클래스로 지정하는 클래스임

- job.setOutputFormatClass(DelayOutputFormat.class);

◈ FileOutputFormat<KEY, VALUE> 클래스를 상속받아 구현한다.

- public org.apache.hadoop.mapreduce.RecordWriter<Text, DelayValue> getRecordWriter() 메서드를 재정의
- 출력 파일의 경로(파일명 포함)를 지정할 수 있으며,
- Writer 객체를 리턴해야 한다.

```
1.  import java.io.IOException;
2.
3.  import org.apache.hadoop.fs.FSDataOutputStream;
4.  import org.apache.hadoop.fs.FileSystem;
5.  import org.apache.hadoop.fs.Path;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.TaskAttemptContext;
8.  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
9.
10. public class DelayOutputFormat extends FileOutputFormat<Text, DelayValue> {
11.
12.     @Override
13.     public org.apache.hadoop.mapreduce.RecordWriter<Text, DelayValue>
    getRecordWriter(TaskAttemptContext context) throws IOException,
    InterruptedException {
14.         Path path = FileOutputFormat.getOutputPath(context);
15.
16.         Path fullPath = new Path(path, "delay.csv");
17.
18.         //create the file in the file system
19.         FileSystem fs = path.getFileSystem(context.getConfiguration());
20.         FSDataOutputStream fileOut = fs.create(fullPath, context);
21.
22.         //create our record writer with the new file
23.         return new DelayValueWriter(fileOut);
24.     }
25.
26. }
```

드라이버 클래스

◈ 출력 포맷 클래스 설정
- job.setOutputFormatClass(DelayOutputFormat.class);

◈ 실행
- $ hadoop jar WeatherDelay.jar /airline/2008.csv
- $ hdfs dfs -cat /output/delay.csv

```
1.  import org.apache.hadoop.conf.Configuration;
2.  import org.apache.hadoop.conf.Configured;
3.  import org.apache.hadoop.fs.FileSystem;
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.io.IntWritable;
6.  import org.apache.hadoop.io.Text;
7.  import org.apache.hadoop.mapreduce.Job;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.util.GenericOptionsParser;
12. import org.apache.hadoop.util.Tool;
13. import org.apache.hadoop.util.ToolRunner;
14.
15. public class WeatherDelay extends Configured implements Tool {
16.
17.     @SuppressWarnings("deprecation")
18.     public int run(String[] args) throws Exception {
19.         String[] otherArgs = new GenericOptionsParser(getConf(), args).getRemainingArgs();
20.
21.         Job job = new Job(getConf(), "WeatherDelay");
22.
23.         // 입출력 데이터 경로 설정
24.         FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
25.
26.         // Job 클래스, Mapper 클래스, Reducer 클래스 설정
27.         job.setJarByClass(WeatherDelay.class);
28.         job.setMapperClass(WeatherDelayMapper.class);
29.         job.setReducerClass(WeatherDelayReducer.class);
30.
31.         //입출력 데이터 포맷 설정
32.         job.setInputFormatClass(TextInputFormat.class);
33.         job.setOutputFormatClass(DelayOutputFormat.class);

34.         // 출력키 및 출력값 유형 설정
35.         job.setOutputKeyClass(Text.class);
36.         job.setOutputValueClass(IntWritable.class);
37.         FileOutputFormat.setOutputPath(job, new Path("/output");
38.
39.         job.waitForCompletion(true);
40.         return 0;
41.     }
42.
43.     public static void main(String[] args) throws Exception {
44.         int res = ToolRunner.run(new Configuration(), new WeatherDelay(), args);
45.         System.out.println("## RESULT:" + res);
46.     }
47. }
```