

몇 가지 예제 프로그램

이전 예시에서 워드 카운터 프로그램을 수행해 보았다.
여기서는 5가지의 다른 예시들을 소개하고자 한다.
각 예제들을 간단히 소개하면 다음과 같다.

예제 프로그램	설명
WordCount2	WordCount 를 개선한 버전이다. 2M.ID.CONTENTS 를 사용한다.
	메인 2M.ID.CONTENTS wordCount2
TopN	"문자열 \t 빈도 수"의 형태로 구성된 입력 파일에서 빈도 수 기준으로 최대 N 개의 레코드들 을 뽑아 낸다. WordCount2 프로그램에서 파생된 파일(part-r-00000)을 사용한다.
	메인 part-r-00000 topN 3 0 (앞 숫자 : N, 뒤 숫자 : 정렬 방식_0 또는 1)
CountTrigram	트라이그램별로 빈도 수를 계산하고, 바로 TopN 을 실행하여 가장 빈도 수가 높은 트라이그 램들이 무엇인지 보여 준다. 2M.ID.CONTENTS 를 사용한다.
CountCitation	2 백만 개의 영문 위키피디아 문서들에 대해서 인용된 숫자를 보여 준다. 2M.SRCID.DSTID 를 단순히 사용한다.
	메인 2M.SRCID.DSTID countcitation
JoinIDTitle	위키피디아 문서 ID 들로부터 해당 문서의 타이틀을 알아 낸다. CountCitation 의 결과에 TopN 을 적용하여 가장 많이 인용된 10 개의 문서를 알아 낸 후 에 JoinIDTitle 을 이용해서 그 문서들의 타이틀을 알아 낼 수 있다.
	메인 2M.TITLE.ID countCitation joinIDTitle

실습에 사용될 각 파일 설명

실습에 사용될 파일은 다음과 같다.

파일 : 2M.TITLE.ID

각 라인마다 위키피디아 문서의 타이틀과 문서의 ID가 tab으로 구분되어 있다.

타이틀이 먼저 나오고 탭 문자가 나오고 그 다음에 문서 ID가 나온다.

타이틀이나 id 모두 유일한 값들로 구성이 되어 있으며, 모든 위키피디아 문서들의 각기 유일한 값을 가지고 있다.

파일 내용

100	어썩구 저썩구	158031
200	하하하 호호호	234521

파일 : 2M.ID.CONTENTS

문서별로 그 문서의 텍스트를 나열해 놓은 문서이다.

한 라인마다 하나의 문서에 대한 ID와 문서 텍스트 사이에 tab 문자가 들어 있다.

파일 내용

문서 id	문서의 텍스트
100	어썩구 저썩구
200	하하하 호호호

WordCount2 프로그램

WordCount2는 WordCount를 조금 더 개선한 버전이다.

WordCount2는 WordCount와 비교하여 크게 3가지가 달라진다.

2M.ID.CONTENTS 파일의 크기는 대략 2.4 GB 크기의 파일로써, 2백만 개의 영문 위키피디아 페이지들이다.

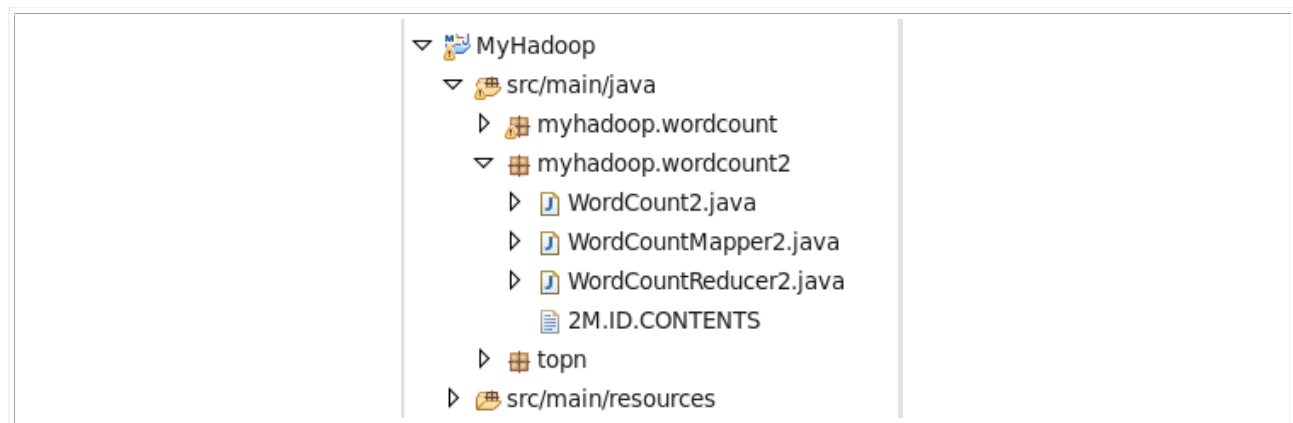
WordCount와 달라진 점 :

입력으로 사용할 텍스트 파일이 다르다.

컴бай너를 사용한다.(리듀스 클래스를 그대로 사용하도록 한다.)

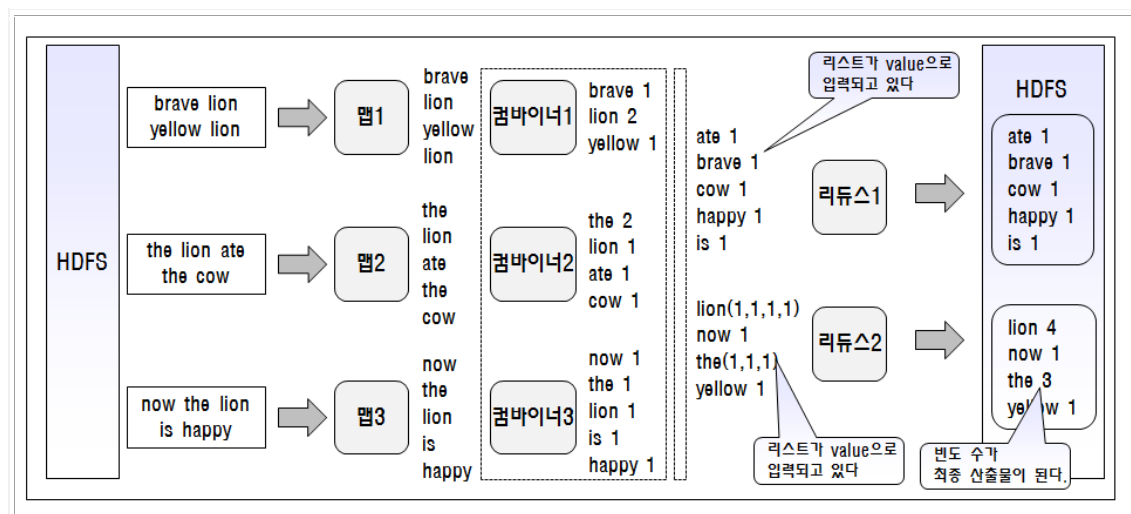
카운터를 사용하여 유일한 단어들의 수가 몇 개인지 세어 본다.

파일 구조



컴바이너(Combiner)

컴바이너는 맵 태스크와 리듀서 태스크 간의 네트워크 통신량을 최소화하기 위하여 데이터의 크기를 줄이기 위하여 만든 개념이다.



윈도우에서 리눅스로 복사하기 :

해당 디렉토리가 존재하지 않으면 복사를 할 수 없으므로 우선 myhadoop.wordcount2 패키지를 우선 생성하도록 한다. 패키지를 만들게 되면 디렉토리가 자동으로 만들어 진다.

이번 예시에서 2M.ID.CONTENTS 파일을 이용해야 하므로 다음 명령어를 사용하여 복사하도록 한다.

소스 코드

윈도우 시스템의 파일을 다음 코드를 이용하여 복사하도록 한다.(시간이 조금 걸린다.)

```
cp /mnt/hgfs/shared/2M.ID.CONTENTS  
/home/hadoop/workspace/MyHadoop/src/main/java/myhadoop/wordcount2/
```

실습 : 워드 카운트2 Mapper

KeyValueTextInputFormat 클래스는 기본적으로 TextInputFormat과 동일하게 작동하는 데 Key와 Value 사이에 tab 문자가 있다고 가정한다.

소스 코드	WordCountMapper2.java	카페
-------	-----------------------	----

실습 : 워드 카운트2 리듀스

소스 코드	WordCountReducer2.java	카페
-------	------------------------	----

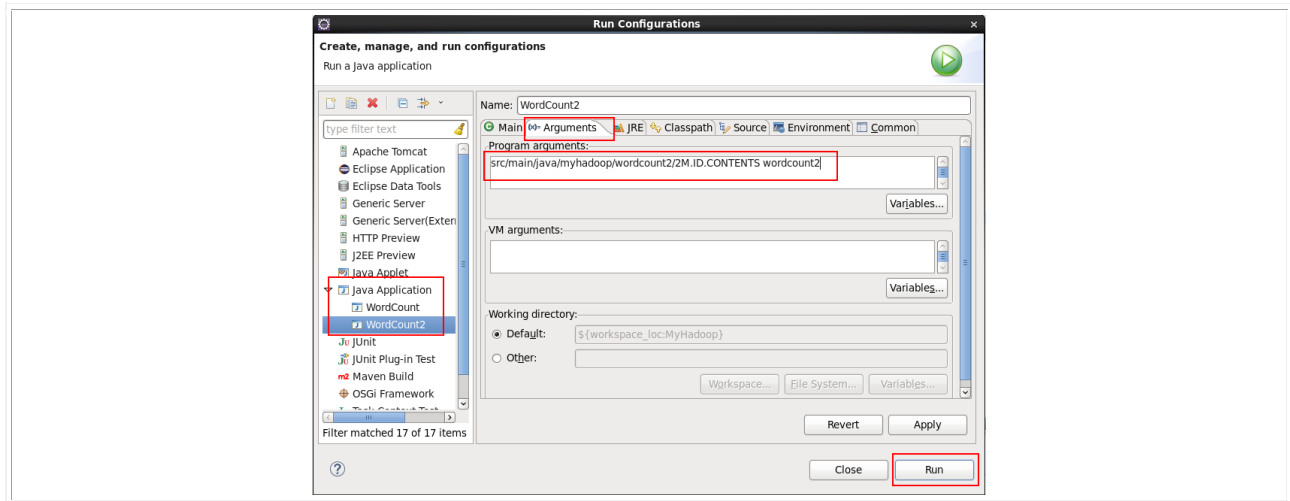
실습 : 워드 카운트2

소스 코드	WordCount2.java	카페
-------	-----------------	----

실습 : 명령행 매개 변수 입력하기

명령행 매개 변수 2개를 요구한다.

```
src/main/java/myhadoop/wordcount2/2M.ID.CONTENTS wordcount2
```



결과 확인

WordCount2 프로그램을 실행하고 나면 결과 파일이 생성된다.
head 커맨드를 이용하여 파일 내용을 확인해 본다.

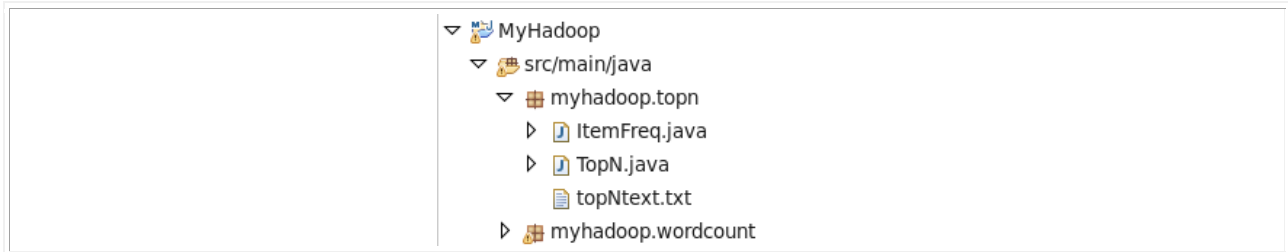
소스 코드

```
[hadoop@localhost wordcount2]$ head part-r-00000
!      884
!!     301
!!!    28
!!!!   7
!!!!!  2
!!!!!!!evil    1
!!!!!!!phoenix 1
!!!!15!!!!!!-!!-!!!!    1
!!!"    4
!!!**** 1
```

TopN 프로그램

TopN 프로그램은 모든 단어들의 빈도 수 중에서 Top N 개만 추출해내는 프로그램이다.

파일 구조



좌측 그림에서 상위 3개를 추출하여 우측 그림과 같이 표현해보도록 한다.

좌측 그림처럼 파일 topNtext.txt 파일을 작성하도록 한다.

문서 원본		좌측 문서에서 Top 3	
a	10	d	25
b	20	b	20
c	15	c	15
d	25		
e	12		

TopN 프로그램을 위한 기본 사항

항목	설명
동작 방식	<p>입력될 파일의 입력 포맷은 "텍스트 tab 빈도수"이다.</p> <p>N은 빈도 수를 말한다.</p> <p>즉, 빈도 수가 가장 큰 몇 개의 레코드를 출력하고 싶은 지 결정해 주는 값을 말한다.</p> <p>매퍼(mapper)는 빈도 수가 가장 큰 N개를 골라 내어 reducer로 보낸다.</p> <p>리듀서(reducer)는 한 개만 할당하여 mapper 에서와 동일한 작업을 반복하여 최종적으로 가장 큰 N 개를 출력해낸다.</p>
명령행 command 인자	프로그램이름 작업대상파일 결과가저장될디렉토리 topN 숫자
참고 사항	<p>데이터가 엄청 많고, 리듀서의 개수가 많다면 이 예제는 적합하지 않다.</p> <p>benchmark test 기법인 TeraSort 를 사용하길 권장한다.</p>

ItemFreq 클래스

ItemFreq 클래스는 자바의 우선 순위 Queue에 들어갈 entry(엔트리) 정보를 가지고 있는 클래스이다.

클래스	ItemFreq	항목 설명
변수	item, freq	item 은 단어 1 개를 의미한다. freq 는 해당 단어의 빈도 수를 의미한다.
메소드	getter, setter, toString()	-
생성자	생성자() 생성자(item, freq)	-

입력 파라미터 N의 전달

맵이나 리듀스에 별도로 파라미터를 전달하는 방법은 Configuration 객체를 통하여 Property(변수)를 정의하는 것이다.

파라미터 N의 전달 :

TopN의 N(정수 값)은 다음과 같이 코딩하면 된다.

항목	설명
main 메소드	job.getConfiguration().setInt("topN", Integer.parseInt(args[2]));
setup() 메소드	맵이나 리듀스의 setup() 메소드이다. topN = context.getConfiguration().getInt("topN", 10);
부연 설명	즉, 메인 메소드에서 setting을 하고 나면 맵이나 리듀스에서 gettting할 수 있다.

프로그램 구현하기

다음과 같은 절차에 의하여 프로그램을 구현하도록 한다.

프로그램 구현하기 :

TopN 클래스 만들기
내부 클래스 Map, Reduce 클래스 각각 만들기
내부 클래스 Map의 3가지 메소드 구현하기
내부 클래스 Reduce의 3가지 메소드 구현하기
ItemFreqComparator 클래스 구현하기
insert 메소드 구현하기

사용자 정의 우선 순위 Queue를 생성한다.
클래스 이름은 ItemFreqComparator으로 하기로 한다.(임의의 이름 가능)
Comparator 인터페이스를 상속받는다.
주상 메소드 compare() 메소드를 오버라이딩한다.
compare() 메소드에서 비교하고자 하는 항목을 비교하도록 한다.
예시에서는 빈도 수(freq)에 대한 대소 비교를 수행하면 된다.

Map이나 Reduce 영역에서 코딩할 일

1. Comparator 객체를 생성한다.

```
Comparator<ItemFreq> comparator = new ItemFreqComparator();
```

2. PriorityQueue 객체를 생성한다.

생성자의 매개 변수는 topN과 1에서 구한 객체를 넣어주면 된다.

```
PriorityQueue<ItemFreq> queue = new PriorityQueue<ItemFreq>(10, comparator);
```

insert 함수 구현하기

우선 순위 큐 삽입 함수는 TopN 클래스 내에 있는 함수이다.

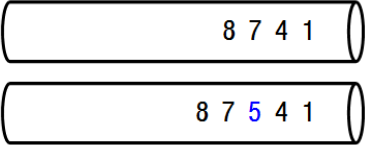
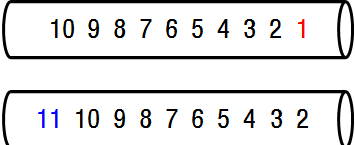
이 함수는 우선 순위 큐에 ItemFreq 객체를 넣기 위하여 사용하는 함수이다.

항목	설명
사용 형식	insert(PriorityQueue queue, String item, Long IValue, int topN)
queue	우선 순위 Queue 객체
item	해당 단어
IValue	빈도 수
topN	우선 순위 안에 들어갈 수 있는 최대 원소의 수

Queue의 처리 방법

insert() 메소드를 사용하는 경우 Queue에 ItemFreq 객체가 들어 간다.

topN = 10이라고 가정하고, Q의 내부 크기에 따라서 어떻게 동작하는 지 살펴 보도록 하자.

<p>Case A : 우선 순위 Q의 크기가 topN 보다 작은 경우</p> <p>⑤ ⇒ </p>	<p>Case B : 우선 순위 Q의 크기가 topN와 동일한 경우</p> <p>⑪ ⇒ </p>
---	--

Map이나 Reduce 영역에서 구현할 내용

하단 코드들을 보면 map() 메소드나 reduce() 메소드에 context.write() 메소드와 관련된 코딩이 없다.

이유는 입력된 레코드들이 모두 들어 와야 최대 빈도 수 만큼의 항목이 결정되기 때문이다.

그럼 이 처리는 ? cleanup() 메소드

항목	설명
map()	<pre> public void map(Text key, Text value, Context context) throws IOException, InterruptedException { Long IValue = (long)Integer.parseInt(value.toString()); insert(queue, key.toString(), IValue, topN); } </pre>
reduce()	<pre> public void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException { long sum = 0; for (LongWritable val : values) { sum += val.get(); } insert(queue, key.toString(), sum, topN); } </pre>

Cleanup 메소드

cleanup() 메소드는 입력 레코드가 모두 들어온 다음 후속적으로 실행되는 메소드이다.

여기서 우선 순위 Q에 들어 있는 ItemFreq 객체들을 하나씩 꺼내어 context 변수를 이용하여 출력을 하면 된다.

메소드	코드 내용
매퍼 영역	<pre> @Override protected void cleanup(Context context) throws IOException, InterruptedException { while (queue.size() != 0) { ItemFreq item = (ItemFreq)queue.remove(); context.write(new Text(item.getItem()), new LongWritable(item.getFreq())); } } </pre>
리듀서 영역	<pre> @Override protected void cleanup(Context context) throws IOException, InterruptedException { while (queue.size() != 0) { ItemFreq item = (ItemFreq)queue.remove(); context.write(new Text(item.getItem()), new </pre>

	<pre>LongWritable(item.getFreq())); } }</pre>
--	---

실습 : ItemFreq 클래스 구현

ItemFreq 클래스는 자바의 우선 순위 Queue에 들어갈 entry(엔트리) 정보를 가지고 있는 클래스이다.
자바의 Bean 클래스라고 이해하면 된다.

소스 코드	ItemFreq.java	카페
-------	---------------	----

실습 : TopN 메인 프로그램

TopN의 Main 프로그램을 작성하도록 한다.

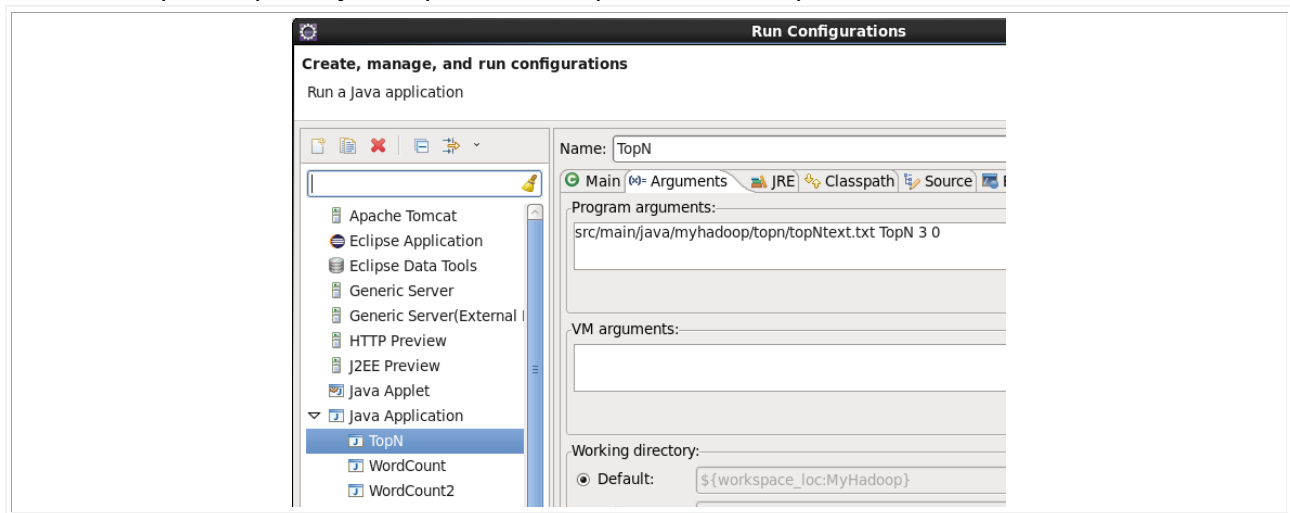
소스 코드	TopN.java	카페
-------	-----------	----

실습 : 명령행 매개 변수 입력하기

명령행 매개 변수는 다음과 같이 입력하면 된다.

마지막 숫자 0은 오름차순을 의미하고, 내림차순을 적용하려면 1을 입력하도록 한다.

/home/hadoop/workspace/MyHadoop/wordcount2/part-r-00000 TopN 10 0



출력 결과 확인

WordCount2 프로그램에서 생성된 파일을 이용하려면 다음과 같이 입력하면 된다.

/home/hadoop/workspace/MyHadoop/wordcount2/part-r-00000 TopN 10 0

출력 결과는 다음과 같다.

출력 결과

as	3344250
for	3505449
is	4422240
was	4938964
to	8456051
a	8713192
in	11745062
and	11793404
of	14070617
the	29307550

CountCitation 프로그램

Count Citation 프로그램은 인용된 문서의 개수를 구해주는 프로그램이다.
2백만 개의 영문 위키피디아 문서들을 이용하여 인용된 숫자를 보여 주도록 한다.

윈도우에서 리눅스로 복사하기 :

해당 디렉토리가 존재하지 않으면 복사를 할 수 없으므로 우선 myhadoop.countcitation 패키지를 우선 생성하도록 한다.
패키지를 만들게 되면 디렉토리가 자동으로 만들어 진다.

이번 예시에서 2M.SRCID.DSTID 파일을 이용해야 하므로 다음 명령어를 사용하여 복사하도록 한다.

소스 코드

윈도우 시스템의 파일을 다음 코드를 이용하여 복사하도록 한다.(시간이 조금 걸린다.)

```
cp /mnt/hgfs/shared/2M.SRCID.DSTID  
/home/hadoop/workspace/MyHadoop/src/main/java/myhadoop/countcitation/
```

파일 : 2M.SRCID.DSTID

예시에서 사용될 2M.SRCID.DSTID 파일은 소스 문서와 타겟 문서 간의 연결 관계를 나타낸다.

한 라인마다 소스 문서와 타겟 문서가 tab 문자로 구분이 되어 있다.

왼쪽은 소스 문서의 id이고, 오른쪽은 타겟 문서의 id이다.

두 개는 tab으로 구분이 되어 있다.

파일 내용

소스 문서 ID	타겟 문서 ID
100	200
300	200
400	200
500	600
700	600
800	900

출력 결과 미리 보기

위의 문서를 이용하여 프로그램을 실행하게 되면 다음과 같은 결과가 출력된다.

출력 결과

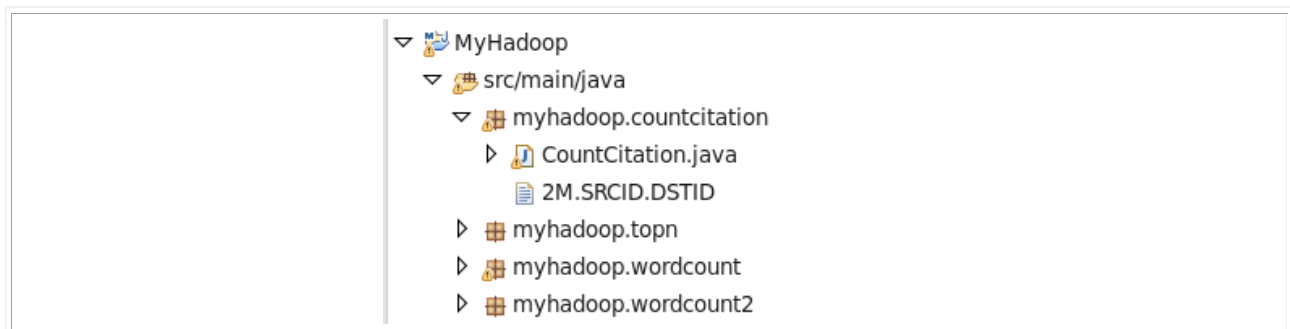
문서 id	인용수
200	3
600	2
900	1

맵과 리듀스의 입출력 데이터

맵과 리듀스에 입출력시 처리 되는 데이터는 다음과 같다.

항목	key 입력	value 입력	key 출력	value 출력
Map	소스 id	타겟 id	타겟 id	1
Reduce	하나의 문서 id로 따라서 들어온 Value 영역의 리스트 값을 모두 더하면 된다.			

파일 구조



실습 : 클래스 구현하기

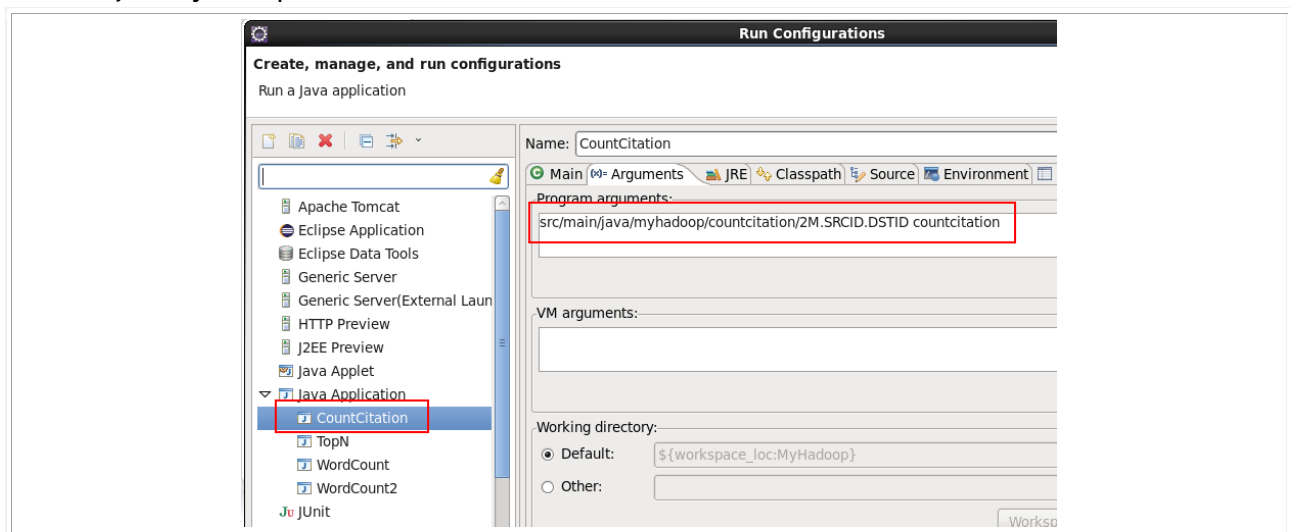
메인 클래스 내에서 Map과 Reduce 클래스를 모두 작성해보도록 한다.

소스 코드	CountCitation.java	카페
-------	--------------------	----

실습 : 명령행 매개 변수 입력하기

명령행 매개 변수를 다음과 같이 입력한다.

src/main/java/myhadoop/countcitation/2M.SRCID.DSTID countcitation



결과 확인

어떤 문서가 가장 많이 인용이 되었는 지를 확인해보도록 한다.

이전에 만들어 두었던 TopN 프로그램을 이용하여 확인해보도록 한다.

```
/home/hadoop/workspace/MyHadoop/countcitation/part-r-00000 TopN 10 0
```

소스 코드

```
3383 13632
147575 8762
10978019 7247
4826085 6007
3392 4950
8504 4366
717 3562
6037917 3305
4887 3183
3414021 2975
```

가장 많은 인용을 한 문서의 결과를 살펴 보았다.

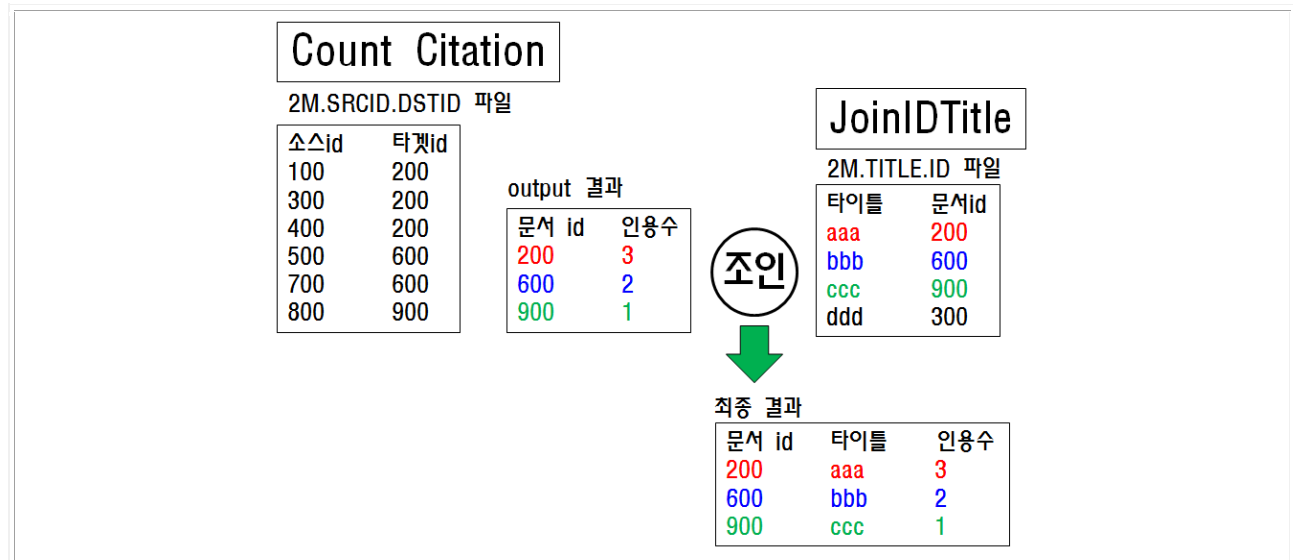
이 결과의 문서는 문서 ID만 들어 있어서 이 문서가 어떠한 문서인지 확인할 수 없다.

다음에 풀어 볼 문제를 이용하여 제목을 찾아 보도록 하겠다.

JoinIDTitle 프로그램

CountCitation 프로그램에서 출력된 결과는 문서의 ID로 존재하여 어떠한 문서인지 판단이 되지 않았다. 이럴 경우 Title을 저장하고 있는 파일(2M.TITLE.ID 파일)과 Join을 하게 되면 어떤 문서인지 파악할 수 있다. 2개의 파일은 포맷이 다른데, 하나는 '문서 id + 빈도수'이고, 다른 하나는 '타이틀과 문서 id'이다. 각각의 파일에 대하여 서로 다른 맵을 적용하게 되면 훨씬 일이 간단해진다.

프로그램 개요



맵과 리듀스의 입출력 데이터

맵과 리듀스에 입출력 처리는 다음과 같다.

2M.TITLE.ID 파일(Map1)		CountCitation 프로그램의 결과 파일(Map2)	
key	value	key	value
문서 id	타이틀 + tab + 1	문서 id	빈도수 + tab + 2
출력 레코드의 키는 문서 ID 이다. 출력 레코드의 밸류는 타이틀 + '\t' + 1의 형태를 갖는다.		출력 레코드의 키는 문서 ID 이다. 출력 레코드의 밸류는 빈도수 + '\t' + 2의 형태를 갖는다.	

원도우에서 리눅스로 복사하기 :

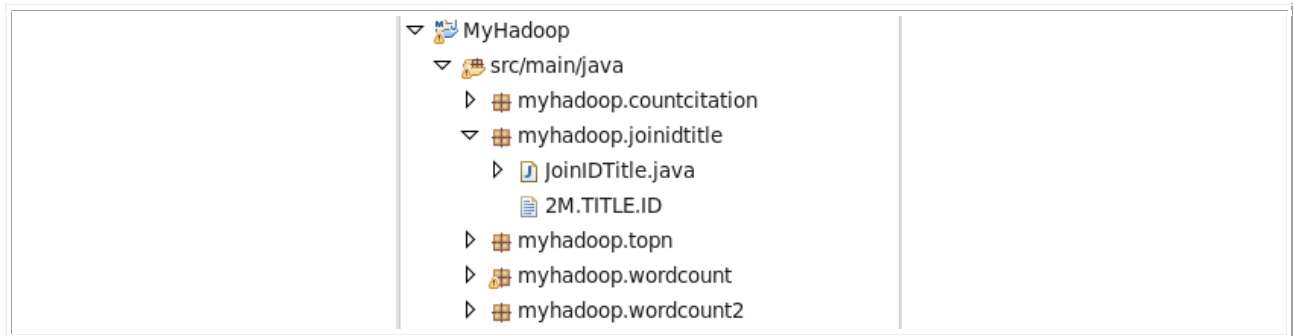
해당 디렉토리가 존재하지 않으면 복사를 할 수 없으므로 우선 myhadoop.joinidtitle 패키지를 우선 생성하도록 한다. 패키지를 만들게 되면 디렉토리가 자동으로 만들어 진다. 이번 예시에서 2M.TITLE.ID 파일을 이용해야 하므로 다음 명령어를 사용하여 복사하도록 한다.

소스 코드

원도우 시스템의 파일을 다음 코드를 이용하여 복사하도록 한다.(시간이 조금 걸린다.)

```
cp /mnt/hgfs/shared/2M.TITLE.ID
/home/hadoop/workspace/MyHadoop/src/main/java/myhadoop/joinidtitle/
```

파일 구조



실습 : 프로그램 작성하기

소스 코드	JoinIDTitle.java	카페
-------	------------------	----

실습 : 명령행 매개 변수 입력하기

명령행 매개 변수를 다음과 같이 입력하도록 한다.

src/main/java/myhadoop/joinidtitle/2M.TITLE.ID countcitation joinidtitle

결과 확인

소스 코드
