4장

목표

1. 자바코드를 이해하자

자바 코드 내부

map

```
/// 반드시 build.xml함수 내부 명령어와 동일해야만 함
package ssafy;

/// 자바import
import java.io.IOException;
import java.util.StringTokenizer;

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

public class Wordcount {
    /*
    Object, Text : input key-value pair type (always same (to get a line of input
file))
    Text, IntWritable : output key-value pair type
    */

    // (1) // Mapper을 상속받아서 만듬
    public static class TokenizerMapper
            // (2) //
            extends Mapper<Object,Text,Text,IntWritable> {

        // variable declairations
        // (3) //
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        // map function (Context -> fixed parameter)
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {

            // value.toString() : get a line
            // (4) //
```

```
            StringTokenizer itr =
                new StringTokenizer(value.toString());
            while ( itr.hasMoreTokens() ) {
                word.set(itr.nextToken());

                // emit a key-value pair
                 // (5) //
                context.write(word,one);
            }
        }
    }
```

(1). Mapper Class를 상속받아서 Map함수를 만듬

(2). input: key== object, value== text
output: key== text, value == intwritable

(3). 함수값 선언

final: 값을 바꿔서는 안됨
고정값을 1로 선언

(4). 받은 문자열을 단어단위로 자르는 함수를 만든다.

(5). (단어 : 1) 의 형태로 방출하는 것을 해당 문자열의 모든 단어에 대해 반복한다

reduce

```
    public static class IntSumReducer
            extends Reducer<Text,IntWritable,Text,IntWritable> {

        // variables
        private IntWritable result = new IntWritable();

        // key : a disticnt word
        // values :  Iterable type (data list)
        public void reduce(Text key, Iterable<IntWritable> values, Context
context)
                throws IOException, InterruptedException {
            // (1) //
            int sum = 0;
            for ( IntWritable val : values ) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key,result);
        }
    }
```

(1). 해당 자바코드를 실행함

여기서는, 반복문을 이용하여 같은 단어의 개수를 세고 그걸 (key,result)로 방출함

main

```
    /* Main function */
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new
GenericOptionsParser(conf,args).getRemainingArgs();
        if ( otherArgs.length != 2 ) {
            System.err.println("Usage: <in> <out>");
            System.exit(2);
        }
        // (1) //
        Job job = new Job(conf,"word count");
        job.setJarByClass(Wordcount.class);

        // let hadoop know my map and reduce classes
        // (2) //
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(IntSumReducer.class);

        // (3) //
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // set number of reduces
        job.setNumReduceTasks(2);

        // set input and output directories
        // (4) //
        FileInputFormat.addInputPath(job,new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job,new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1 );
    }
}
```

(1). 따옴표 안은 설명문이라 생략가능

(2). mapper, reduce class 선언

(3). output key value 선언

(4). 출력경로 설정 및 출력