Wadoop User API

Author: Guanjie Chen (guanjiec)
Wenhan Lu (wenhanl)

1. Description

User interface provided by Wadoop is very similar to Hadoop. The user defined job class must extend MapReduceJob<K1, V1, K2, V2>, where K1, V1 is input key-value type, K2, V2 is output key-value type. This class must have a getMapper() method and a getReducer() method. They are expected to return a Mapper and a Reducer. The prototype of a typical MR job class is like below:

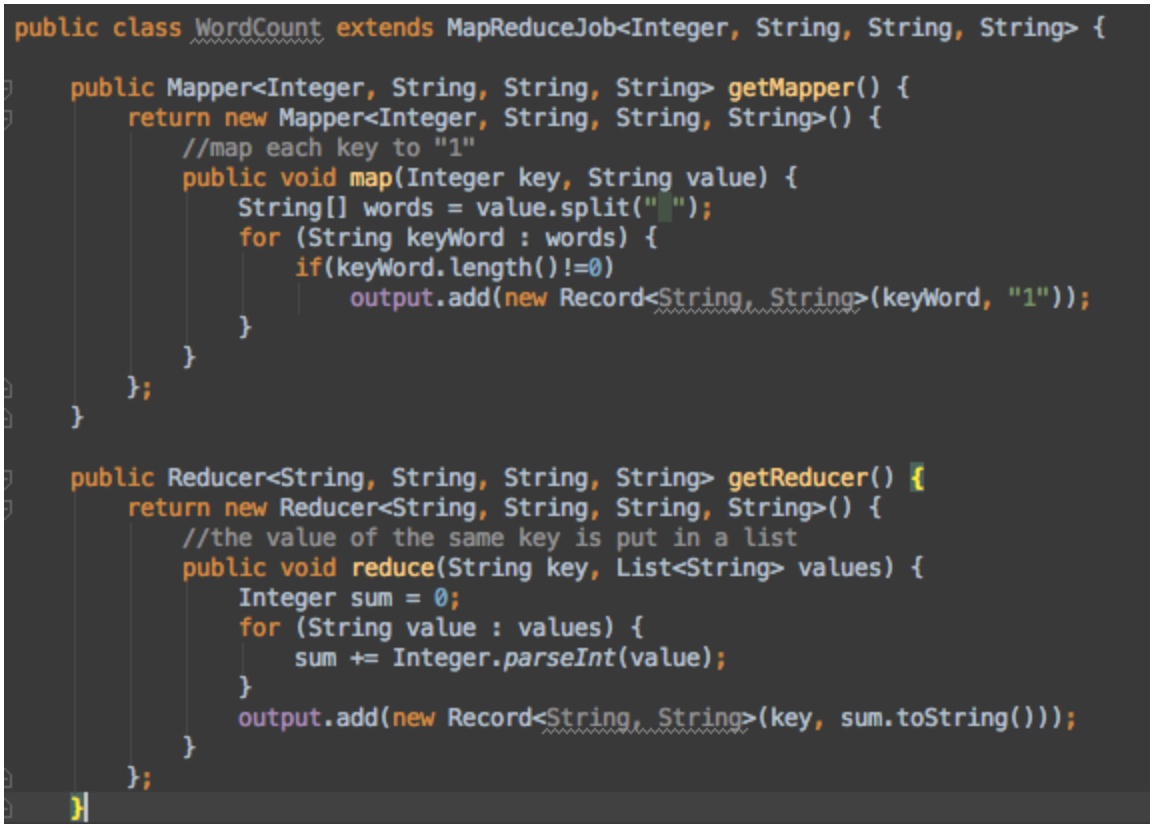
```
public class Example extends MapReduceJob<K1, V1, K2, V2> {

   public Mapper<K1, V1, K-middle, V-middle> getMapper() {
      return new Mapper<K1, V1, K-middle, V-middle>() {
         public void map(K1 key, V1 value) {
                  //Your implementation
                  output.add(new Record<K-middle, V-middle>(outkey, outval));
         }
      };
   }

   public Reducer<K-middle, V-middle, K2, V2> getReducer() {
      return new Reducer<K-middle, V-middle, K2, V2>() {
         public void reduce(Kmiddle key, List<V-middle> values) {
                  //Your implemtation
                  output.add(new Record<K2, V2>(outkey, outval));
         }
      };
   }
}
```

Note: *Now our mapreduce job is only limited to TextInputFormat in Hadoop, that is, for map input, you must use line number as key (Integer) and its content as value (String). We can support more format in the future.*

2. Examples

Wordcount:

Most classic example of mapreduce. Count word occurrence in a text file.



```
public class WordCount extends MapReduceJob<Integer, String, String, String> {

    public Mapper<Integer, String, String, String> getMapper() {
        return new Mapper<Integer, String, String, String>() {
            //map each key to "1"
            public void map(Integer key, String value) {
                String[] words = value.split(" ");
                for (String keyWord : words) {
                    if(keyWord.length()!=0)
                        output.add(new Record<String, String>(keyWord, "1"));
                }
            }
        };
    }

    public Reducer<String, String, String, String> getReducer() {
        return new Reducer<String, String, String, String>() {
            //the value of the same key is put in a list
            public void reduce(String key, List<String> values) {
                Integer sum = 0;
                for (String value : values) {
                    sum += Integer.parseInt(value);
                }
                output.add(new Record<String, String>(key, sum.toString()));
            }
        };
    }
```

Anagram:

Find all anagrams in dictionary. The Anagram mapper class gets a word as a line from the HDFS input and sorts the letters in the word and writes its back to the output collector as Key : sorted word (letters in the word sorted) Value: the word itself as the value.

When the reducer runs then we can group anagrams togather based on the sorted key. The Anagram reducer class groups the values of the sorted keys that came in and checks to see if the values iterator contains more than one word. if the values contain more than one word we have spotted a anagram.

```
public class Anagram extends MapReduceJob<Integer, String, String, String>{

    /**
     * The Anagram mapper class gets a word as a line from the HDFS input and sorts the
     * letters in the word and writes its back to the output collector as
     * Key : sorted word (letters in the word sorted)
     * Value: the word itself as the value.
     * When the reducer runs then we can group anagrams togather based on the sorted key.
     *
     * @author Wenhan Lu
     *
     */
    public Mapper<Integer, String, String, String> getMapper() {
        return new Mapper<Integer, String, String, String>() {
            public void map(Integer key, String value) {
                char[] wordChars = value.toCharArray();
                Arrays.sort(wordChars);
                String sortedWord = new String(wordChars);
                output.add(new Record<String, String>(sortedWord, value));
            }
        };
    }

    /**
     * The Anagram reducer class groups the values of the sorted keys that came in and
     * checks to see if the values iterator contains more than one word. if the values
     * contain more than one word we have spotted a anagram.
     * @author Wenhan Lu
     *
     */
    public Reducer<String, String, String, String> getReducer() {
        return new Reducer<String, String, String, String>() {
            //the value of the same key is put in a list
            public void reduce(String key, List<String> values) {
                String out = "";
                for(String value : values){
                    out = out + value + "~";
                }
                StringTokenizer outputTokenizer = new StringTokenizer(out,"~");
                if(outputTokenizer.countTokens()>=2){
                    out = out.replace("~", ";");
                    output.add(new Record<String, String>(key, out));
                }
            }
        };
    }
```