1. Explain in brief Writable and Writable Comparable in Hadoop with an example.

Data Type

A data type is a set of data with values having predefined characteristics. There are several kinds of data types in Java. For example- int, short, byte, long, char etc. These are called as primitive data types. All these primitive data types are bound to classes called as wrapper class. For example int, short, byte, long are grouped under INTEGER which is a wrapper class. These wrapper classes are predefined in the Java.

Interface in Java

An interface in Java is a complete abstract class. The methods within an interface are abstract methods which do not accept body and the fields within the interface are public, static and final, which means that the fields cannot be modified.

The structure of an interface is most likely to be a class. We cannot create an object for an interface and the only way to use the interface is to implement it in other class by using ‘implements’ keyword.

Serialization

Serialization is nothing but converting the raw data into a stream of bytes which can travel along different networks and can reside in different systems. Serialization is not the only concern of Writable interface; it also has to perform compare and sorting operation in Hadoop.

Writable:

Writable in an interface in Hadoop and types in Hadoop must implement this interface. Hadoop provides these writable wrappers for almost all Java primitive types and some other types,but sometimes we need to pass custom objects and these custom objects should implement Hadoop's Writable interface. Hadoop MapReduce uses implementations of Writables for interacting with user-provided Mappers and Reducers.

For implementing Writables, we need few methods in Hadoop as shown below:

public interface Writable {

void readFields(DataInput in);

void write(DataOutput out);

}

Here, readFields, reads the data from network and write will write the data into local disk. Both are necessary for transferring data through clusters. DataInput and DataOutput classes (part of java.io) contain methods to serialize the most basic types of data.

Suppose we want to make a composite key in Hadoop by combining two Writables then follow the steps below:public class add implements Writable{

public int a;

public int b;

public add(){

this.a=a;

this.b=b;

}

public void write(DataOutput out) throws IOException {

out.writeInt(a);

out.writeInt(b);

}

public void readFields(DataInput in) throws IOException {

a = in.readInt();

b = in.readInt();

}

public String toString() {

return Integer.toString(a) + ", " + Integer.toString(b)

}

}

Thus we can create our custom Writables in a way similar to custom types in Java but with two additional methods, write and readFields. The custom writable can travel through networks and can reside in other systems.

This custom type cannot be compared with each other by default, so again we need to make them comparable with each other.

Writable Comparable:

WritableComparable interface is just a subinterface of the Writable and java.lang.Comparable interfaces. For implementing a WritableComparable we must have compareTo method apart from readFields and write methods, as shown below:

public interface WritableComparable extends Writable, Comparable

{

void readFields(DataInput in);

void write(DataOutput out);

int compareTo(WritableComparable o)

}

Comparison of types is crucial for MapReduce, where there is a sorting phase during which keys are compared with one another.

Implementing a comparator for WritableComparables like the org.apache.hadoop.io.RawComparator interface will definitely help speed up your Map/Reduce (MR) Jobs. As you may recall, a MR Job is composed of receiving and sending key-value pairs. The process looks like the following.

(K1,V1) –> Map –> (K2,V2)

(K2,List[V2]) –> Reduce –> (K3,V3)

The key-value pairs (K2,V2) are called the intermediary key-value pairs. They are passed from the mapper to the reducer. Before these intermediary key-value pairs reach the reducer, a shuffle and sort step is performed.

The shuffle is the assignment of the intermediary keys (K2) to reducers and the sort is the sorting of these keys. In this blog, by implementing the RawComparator to compare the intermediary keys, this extra effort will greatly improve sorting. Sorting is improved because the RawComparator will compare the keys by byte. If we did not use RawComparator, the intermediary keys would have to be completely deserialized to perform a comparison.

We can make custom type a WritableComparable by following the method below:

public class add implements WritableComparable{

public int a;

public int b;

public add(){

this.a=a;

this.b=b;

}

public void write(DataOutput out) throws IOException {

out.writeint(a);

out.writeint(b);

}

public void readFields(DataInput in) throws IOException {

a = in.readint();

b = in.readint();

}

public int CompareTo(add c){

int presentValue=this.value;

int CompareValue=c.value;

return (presentValue < CompareValue ? -1 : (presentValue==CompareValue ? 0 : 1));

}

public int hashCode() {

return Integer.IntToIntBits(a)^ Integer.IntToIntBits(b);

}

}