Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

UNIT- 3 (Hadoop I/O)

Writeable Interface in Hadoop:-

Writable is an interface in Hadoop. Writable in Hadoop acts as a wrapper class to almost all the primitive data type of Java. That is how **int** of java has become **IntWritable** in Hadoop and **String** of Java has become **Text** in Hadoop.
Writables are used for creating serialized data types in Hadoop.

Hadoop framework definitely needs Writable type of interface in order to perform the following tasks:

- Implement serialization
- Transfer data between clusters and networks
- Store the deserialized data in the local disk of the system

Implementation of writable is similar to implementation of interface in Java. It can be done by simply writing the keyword 'implements' and overriding the default writable method.

Need of Writeable Interface: When we write a key as IntWritable in the Mapper class and send it to the reducer class, there is an intermediate phase between the Mapper and Reducer class i.e., shuffle and sort, where each key has to be compared with many other keys. If the keys are not comparable, then shuffle and sort phase won't be executed or may be executed with high amount of overhead.

If a key is taken as IntWritable by default, then it has comparable feature because of RawComparator acting on that variable. It will compare the key taken with the other keys in the network. This cannot take place in the absence of Writable.

For implementing Writables, we need few more methods in Hadoop:

```
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.

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Creating Custom Writable Datatype in Hadoop: 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 read Fields. The custom writable can travel through networks and can reside in other systems.

Writable Comparable and comparators: WritableComparables can be compared to each other, typically via Comparators. Any type, which is to be used as a key in the Hadoop Map-Reduce framework, should implement this interface.

Writable variables in Hadoop have the default properties of Comparable.

How can WritableComparable be implemented in Hadoop?

The implementation of WritableComparable is similar to Writable but with an additional 'CompareTo' method inside it. These are the following implementation of WritableComparable.

```
public interface WritableComparable extends Writable, Comparable
{
    void readFields(DataInput in);
    void write(DataOutput out);
    int compareTo(WritableComparable o)
}
```

Create Custom WritableComparable: if we have made our custom type, Writable rather than WritableComparable our data won't be compared with other data types.

```
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);
    }
}
```

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Difference between WritableComparable & WritableComparator in Hadoop:

WritableComparable	WritableComparator
WritableComparables can be compared to each other, typicallyvia Comparators. Any type which is to be used as a key in the Hadoop Map-Reduce framework should implement this interface.	A Comparator for WritableComparables. This base implemenation uses the natural ordering. To define alternate orderings, override compare (WritableComparable, WritableComparable) .
org.apache.hadoop.io.WritableComparable	org.apache.hadoop.io.WritableComparator
For implementing a WritableComparable we must have compareTo method apart from readFields and write methods, as shown below:	A Comparator that operates directly on byte representations of objects. **compare**(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) Compare two

```
public interface WritableComparable
extends Writable, Comparable
{
void readFields(DataInput in);
void write(DataOutput out);
int compareTo(WritableComparable o)
}
```

objects in binary. b1[s1:l1] is the first object, and b2[s2:l2] is the second object. **Parameters:** b1 – The first byte array. s1 – The position index in b1. The object under comparison's starting index. l1 – The length of the object in b1. b2 – The second byte array. s2 – The position index in b2. The object under comparison's starting index. l2 – The length of the object under comparison in b2. **Returns:** An integer result of the comparison.

Raw Comparator to enhance performance analysis (Speed): A comparator that operates/compares directly on the bytes of data is called as raw comparator.

Raw comparator operates directly on the byte representation of the data.

A Raw comparator is used to enhance the speed of processing the comparison of keys in the Hadoop/map reduce.

The Signature of the Raw comparator is represented like this:

int compare(byte[ ] b1, int s1, int l1, byte [ ], int s2,int l2)

Compare two objects in binary. B1 [s1 to l1] is the first object and B2 [s2 to l2] second object.

b1 : first byte of array

s1 : first index of the first byte array

l1: length of the objects in b1

b2: second byte array

s2: first index of second byte array

l2: lenth of the objects in b2

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.

Writable Classes – Hadoop Data Types: All these primitive writable wrappers have get() and set() methods to read or write the wrapped value. Below is the list of primitive writable data types available in Hadoop.

- BooleanWritable
- ByteWritable
- IntWritable
- VIntWritable
- FloatWritable
- LongWritable
- VLongWritable
- DoubleWritable

In the above list VIntWritable and VLongWritable are used for variable length Integer types and variable length long types respectively.

Array Writable Classes: Hadoop provided two types of array writable classes, one for single-dimensional and another for two-dimensional arrays. But the elements of these arrays must be other writable objects like IntWritable or LongWritable only but not the java native data types like int or float.

- ArrayWritable
- TwoDArrayWritable

Other Writable Classes

NullWritable: NullWritable is a special type of Writable representing a null value. No bytes are read or written when a data type is specified as NullWritable. So, in Mapreduce, a key or a value can be declared as a NullWritable when we don't need to use that field.

ObjectWritable: This is a general-purpose generic object wrapper which can store any objects like Java primitives, String, Enum, Writable, null, or arrays.

Text : Text can be used as the Writable equivalent of java.lang.String and It's max size is 2 GB. Unlike java's String data type, Text is mutable in Hadoop.

BytesWritable: BytesWritable is a wrapper for an array of binary data.

GenericWritable: It is similar to ObjectWritable but supports only a few types. User need to subclass this GenericWritable class and need to specify the types to support.