UNIT-4
Hadoop I/O

Contents

1. Hadoop I/O:
   - The Writable Interface
   - Writable Comparable and Comparators
2. Writable classes:
   - Writable Wrappers for java primitives
   - Text
   - Byte Writables
   - Null Writables
   - Object Writables
   - Generic Writables
3. Implementing Custom Writable:
   - Implementing Raw Comparator for speed
   - Custom Comparators

Hadoop I/O :-

Writable Interface :- Writable is an interface in Hadoop. Writable in Hadoop acts as a Wrapper class to almost all the primitive datatype of java. That is how int of java has become IntWritable in Hadoop and String of java has become of Text in hadoop.

"Writables are used for creating Serialized datatypes in Hadoop".

To implement the Writable Interface We require two methods:

1. readFields()
2. Write()

Syntax:-

```
public Interface Writable
{
    Void readFields (DataInput in);
    Void Write (Dataoutput out);
}
```

Scanned with CamScanner

Why use Hadoop Writables?:-

In Hadoop, data needs to be transmitted between different nodes in a distributed computing environment. This requires Serialization and de-Serialization of data to convert the data that is in structured format to bytestream and vice-versa. Hadoop therefore uses simple and efficient serialization protocol to serialize data between map and reduce phase and these are called Writables. Some of the examples of Writables as already mentioned before are IntWritable, LongWritable, BoolanWritable, and FloatWritable.

What is the importance of Writable in Hadoop:

Writable is an interface in Hadoop. Writable in Hadoop act as a wrapper class to almost all the primitive datatype 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. So, let us understand what is datatype, interface and Serialization.

Datatype :- A datatype is a set of data with values having pre-defined characteristics. There are several kinds of datatypes in java. For example - int, short, byte, long, char etc., These are called primitive datatypes. All these primitive datatypes 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 :- An interface in java is 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.

Scanned with CamScanner

The structure of an interface is most likely to be a class. We can not 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 rawdata 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.

Why are Writables Introduced in Hadoop?

Writable type of Interface in order to perform the following tasks:

1. Implement Serialization, Transfer data between Clusters and networks.
2. Store the de-serialized data in the local disk of the system.
3. Writables reduces the data size overhead and make the datatransfer easier in network.
4. It has separate read and write fields to read data from network and write data into local disk respectively.

What if Writable were not there in Hadoop?

Let us now understand what happens if Writable is not present in Hadoop.

Serialization is important in Hadoop because it enables easy transfer of data. If Writable is not present in Hadoop, then it uses the Serialization of Java which increases the data over-head in the network.

SmallInt Serialized value using Java Serializer

aced 0005 7372 0011 6a61 7661 2e6c 616e 672e 496e 7465 6765
7212 e2a0 a4f7 8187 3802 0001 4900 0576 616c 7565 7872 0010 6a61 7661 2e
6c61 6e67 2e4e 756d 6286 ac95 1d0b 94e0 8b02 0000 7870 0000 0064

SmallInt Serialized value using IntWritable

00000064.

Scanned with CamScanner

This shows the clear difference between Serialization in Java and Hadoop and also the difference between ObjectInputStream and Writable interface. If the size of Serialized data in Hadoop is like that of Java, then it will definitely become an overhead in the network.

Also the core part of Hadoop framework i.e, shuffle and sort phase won't be executed without using Writable.

How can Writables be implemented in Hadoop?

```
public interface Writable
{
    Void Write(DataOutput out);
    Void readfields(DataInput in);
}

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);
    }
}
```

Scanned with CamScanner

WritableComparable:-

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.

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

WritableComparable is the solution to the above problem.

If a key is taken as IntWritable, by default it has comparable feature because of RawComparator acting on that variable and it will compare the key taken with the other keys in network and if Writable is not there it won't be executed.

By default, IntWritable, LongWritable and Text have a RawComparator which can execute this comparable phase for them.

The RawComparator help the custom Writable?

The answer is no. so, We need to have WritableComparable.

Why we need WritableComparable?

We want to make our customtype as a key, then we should definitely make our key type as WritableComparable rather than Writable. This enables the custom type to be compared with other types and it is also sorted accordingly. Otherwise, the keys won't be compared with each other and they are just passed through the network.

What happens if WritableComparable is not present?

If we have made our custom type is Writable rather than WritableComparable our data won't be compared with other data types. There is no compulsion that our custom types need to be writableComparable until unless if it is key. Because values don't need to be compared with each other as keys.

Scanned with CamScanner

How can WritableComparable be implemented in Hadoop?

The implementation of WritableComparable is similar to Writable but with an additional "CompareTo" method inside it.

Ex:

```
public interface WritableComparable extends Writable, Comparable
{
    void readfields(DataInput in);
    void Write(Dataoutput out);
    int CompareTo(WritableComparable o)
}
```



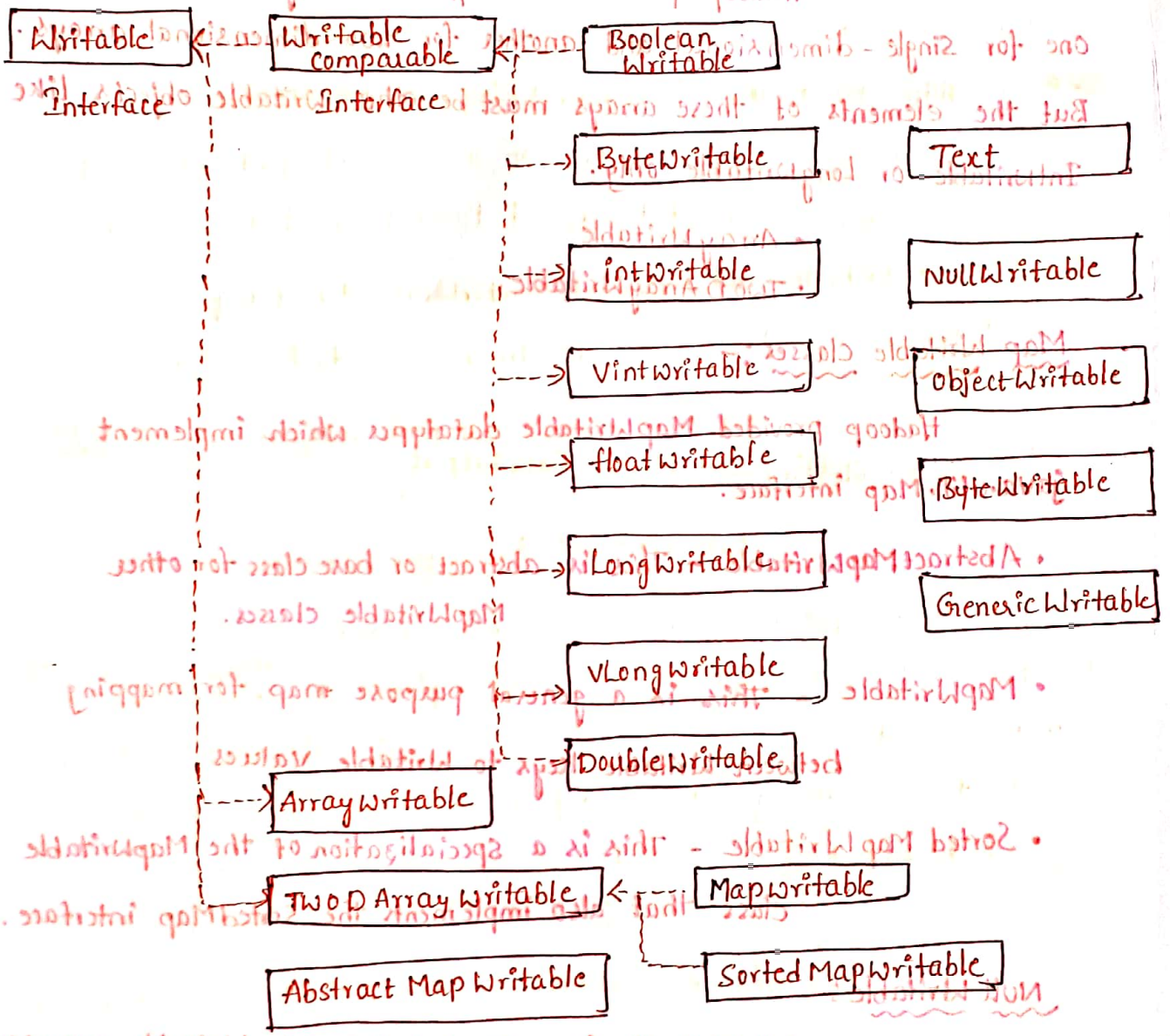
```
public class add implements WritableComparable
{
    public int a;
    public int b;
    public add()
    {
        this.a = a;
        this.b = b;
    }
    public void Write(Dataoutput out) throws IOException
    {
        out.WriteInt(a);
        out.WriteInt(b);
    }
    public void readfields(DataInput 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);
    }
}
```

Scanned with CamScanner

Writable classes Hierarchy

Hadoop provides classes that wrap the java primitive types and implement the WritableComparable and Writable Interfaces. They are provided in the org.apache.hadoop.io package.

All the Writable classes have a get() and a set() method for retrieving and storing the wrapped value.



Primitive Writable classes :-

The list of primitive writable datatypes available in Hadoop is

- BooleanWritable
- ByteWritable
- IntWritable
- VIntWritable

Scanned with CamScanner

- FloatWritable
- LongWritable
- VLongWritable
- DoubleWritable

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

ArrayWritable 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.

- ArrayWritable
- TwoDArrayWritable.

MapWritable classes :-

Hadoop provided MapWritable datatypes which implement java.util.Map interface.

- AbstractMapWritable - This is abstract or base class for other MapWritable classes.
- MapWritable - This is a general purpose map for mapping between Writable keys to Writable values.
- SortedMapWritable - This is a specialization of the MapWritable class that also implements the SortedMap interface.

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.

Scanned with CamScanner

- **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 2GB. unlike java's String datatype, 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.

Example :-

```
import org.apache.hadoop.io.*;
import java.util.*;
public class WritableTest
{
    public static class TextArrayWritable extends ArrayWritable
    {
        public TextArrayWritable()
        {
            super(Text.class);
        }
    }
    public static class IntArrayWritable extends ArrayWritable
    {
        public IntArrayWritable()
        {
            super(IntWritable.class);
        }
    }
    public static void main(String[] args)
    {
```

Scanned with CamScanner

```
IntWritable i1 = new IntWritable (2);
IntWritable i2 = new IntWritable ();
i2.get(5);
IntWritable i3 = new IntWritable ();
i3.Set (i2.get());
System.out.printf ("IntWritables Test I1: %.d, I2: %.d, I3: %.d", i1.get(),
                                                    i2.get(), i3.get());
BooleanWritable bool1 = new BooleanWritable ();
bool1.set (true);
ByteWritable byte1 = new ByteWritable ((byte)7);
System.out.printf ("\n Boolean value : %.s Byte value : %.d", bool1.get(),
                                                    byte1.get());
Text t = new Text ("Hadoop");
Text t2 = new Text ();
t2.set("pig");
System.out.printf ("\n t: %.s, t.length : %.d, t2: %.s, t2.length : %.d\n",
                    t.toString(), t.getLength(), t2.getBytes(), t2.getBytes().
                                                    length);
ArrayWritable a = new ArrayWritable (IntWritable.class);
a.set (new IntWritable [] { new IntWritable (10), new IntWritable (20),
                    new IntWritable (30) });
ArrayWritable b = new ArrayWritable (Text.class);
b.set (new Text [] {new Text ("Hello"), new Text ("Writables"),
                    newText ("World!!!") });
for (IntWritable i : (IntWritable [] a.get ())
System.out.println (i);
for ( Text i : (Text[]) b.get())
System.out.println (i);
IntArray Writable ia = new IntArray Writable ();
ia.set (new IntWritable[] { new IntWritable (100), new IntWritable
                    (300), new IntWritable (500)});
IntWritable[] ivalues = (IntWritable [] ia.get();
for (IntWritable i : ivalues)
System.out.println (i);
```

Scanned with CamScanner

```
        MapWritable m = new MapWritable();
        IntWritable key1 = new IntWritable(1);
        NullWritable value1 = NullWritable.get();

        m.put(key1, value1);
        m.put(new VIntWritable(2), new LongWritable(163));
        m.put(new VIntWritable(3), new Text("Mapreduce"));
        System.out.println(m.containsKey(key1));
        System.out.println(m.get(new VIntWritable(3)));
        m.put(new LongWritable(1000000000), key1);
        Set<Writable> keys = m.keySet();
        for(Writable w: keys)
        System.out.println(m.get(w));
    }
}
```

Op:-

```
------ Array Variables ---------
10
20
30
Hello
Writables
World !!!

100
300
500
------ Map Variables ----------
true
Mapreduce
(null)
163
Mapreduce
1
```

Scanned with CamScanner

Implementing customWritable

Implementing RawComparator will speed up your hadoop Map/Reduce (MR) jobs.

Implementing the org.apache.hadoop.io.RawComparator interface will definitely help speed up your Map/Reduce (MR) jobs.

```
public class Indexpair implements WritableComparable<Indexpair>
{
    private IntWritable i;
    private IntWritable j;
    public Indexpair(int i, int j)
    {
        this.i = new IntWritable(i);
        this.j = new IntWritable(j);
    }
    public int compareTo(Indexpair o)
    {
        int cmp = i.compareTo(o.i);
        if(0!= cmp)
        return cmp;
        return j.compareTo(o.j);
    }
}
```

CustomComparators :-

```
public class CustomTextComparator extends WritableComparable
{
    private Collator collator;
    public CustomTextComparator()
    super(Text.class);
    private Locale locale = new Locale("pl");
    collator = Collator.getInstance(locale);
}
```

Scanned with CamScanner

```
public int Compare (WritableComparable a, WritableComparable b)
{
    Synchronized (Collator)
    {
        return collator.compare(((Text) a.toString(),(Text b)).toString());
    }
}
}
```

Important Questions

(1) What is Writable Interface ? Explain Hadoop IO

(2) Write about Writable Comparable & Comparators.

(3) Explain Hierarchy of Writable classes with Suitable Example;

(4) Write about custom comparator;

Scanned with CamScanner