AVRO

data serialization system

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Apache **Avro** is a language-neutral data serialization system, developed by **Doug Cutting**, the father of Hadoop. This is a brief tutorial that provides an overview of how to set up Avro and how to serialize and deserialize data using Avro.

Audience

This tutorial is prepared for professionals aspiring to learn the basics of Big Data Analytics using Hadoop Framework and become a successful Hadoop developer. It will be a handy resource for enthusiasts who want to use Avro for data serialization and deserialization.

Prerequisites

Before you start proceeding with this tutorial, we assume that you are already aware of Hadoop's architecture and APIs, and you have experience in writing basic applications, preferably using Java.

Disclaimer & Copyright

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.

Table of Contents

Part I – Avro Basics

1. Avro – Overview

To transfer data over a network or for its persistent storage, you need to serialize the data. Prior to the **serialization APIs** provided by Java and Hadoop, we have a special utility, called **Avro**, a schema-based serialization technique.

This tutorial teaches you how to serialize and deserialize the data using Avro. Avro provides libraries for various programming languages. In this tutorial, we demonstrate the examples using Java library.

What is Avro?

Apache Avro is a language-neutral data serialization system. It was developed by Doug Cutting, the father of Hadoop. Since Hadoop writable classes lack language portability, Avro becomes quite helpful, as it deals with data formats that can be processed by multiple languages. Avro is a preferred tool to serialize data in Hadoop.

Avro has a schema-based system. A language-independent schema is associated with its read and write operations. Avro serializes the data which has a built-in schema. Avro serializes the data into a compact binary format, which can be deserialized by any application.

Avro uses JSON format to declare the data structures. Currently it supports languages such as Java, C, C++, C#, Python, and Ruby.

Avro Schemas

Avro depends heavily on its **schema**. It allows every data to be written with no prior knowledge of the schema. It serializes fast and the resulting serialized data is lesser in size. Schema is stored along with the Avro data in a file for any further processing.

In RPC, the client and the server exchange schemas during the connection. This exchange helps in the communication between same named fields, missing fields, extra fields, etc. Both the old and new schemas are always present to resolve any differences.

Avro schemas are defined with JSON that simplifies its implementation in languages with JSON libraries.

Like Avro, there are other serialization mechanisms in Hadoop such as **Sequence Files**, **Protocol Buffers**, and **Thrift**.

Comparison with Thrift and Protocol Buffers

Thrift and **Protocol Buffers** are the most competent libraries of Avro. Avro differs from these frameworks in the following ways:

- Avro supports both dynamic and static *types* as per the requirement. Protocol Buffers and Thrift use Interface Definition Languages (IDLs) to specify schemas and their types. These IDLs are used to generate code for serialization and deserialization.

- Avro is built in the Hadoop ecosystem. Thrift and Protocol Buffers are not built in Hadoop ecosystem.

Unlike Thrift and Protocol Buffer, Avro's schema definition is in JSON and not in any proprietary IDL; that makes it language neutral.

Property	Avro	Thrift and Protocol Buffer
Dynamic schema	Yes	No
Built into Hadoop	Yes	No
Schema in JSON	Yes	No
No need to compile	Yes	No
No need to declare IDs	Yes	No
Bleeding edge	Yes	No

Features of Avro

Listed below are some of the prominent features of Avro:

- Avro is a **language-neutral** data serialization system.
- It can be processed by many languages (currently C, C++, C#, Java, Python, and Ruby).
- Avro creates binary structured format that is both **compressible** and **splittable**. Hence it can be efficiently used as the input to Hadoop MapReduce jobs.
- Avro provides **rich data structures.** For example, you can create a record that contains an array, an enumerated type, and a sub record. These datatypes can be created in any language, can be processed in Hadoop, and the results can be fed to a third language.
- Avro **schemas** defined in **JSON** facilitate implementation in the languages that already have JSON libraries.
- Avro creates a self-describing file named *Avro Data File*, in which it stores data along with its schema in the metadata section.
- Avro is also used in Remote Procedure Calls (RPCs). During RPC, client and server exchange schemas in the connection handshake.
- Avro does not need code generation. The data is always accompanied by schemas, which permit full processing on the data.

General Working of Avro

To use Avro, you need to follow the given workflow:

- **Step 1**: Create schemas. Here you need to design Avro schema according to your data.
- **Step 2**: Read the schemas into your program. It is done in two ways:
  - **By Generating a Class Corresponding to Schema –** Compile the schema using Avro. This generates a class file corresponding to the schema.
  - **By Using Parsers Library –** You can directly read the schema using parsers library.
- **Step 3**: Serialize the data using the serialization API provided for Avro, which is found in the package **org.apache.avro.specific.**
- **Step 4**: Deserialize the data using deserialization API provided for Avro, which is found in the package **org.apache.avro.specific.**

2. Avro – Serialization

Data is serialized for two objectives:

- For persistent storage
- To transport the data over network

What is Serialization?

Serialization is the process of translating data structures or objects state into binary or textual form. Once the data is transported over network or retrieved from the persistent storage, it needs to be deserialized again. Serialization is termed as **marshalling** and deserialization is termed as **unmarshalling**.

Serialization in Java

Java provides a mechanism, called **object serialization** where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.

After a serialized object is written into a file, it can be read from the file and deserialized. That is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

ObjectInputStream and **ObjectOutputStream** classes are used to serialize and deserialize an object respectively in Java.

Serialization in Hadoop

Generally in distributed systems like Hadoop, the concept of serialization is used for **Interprocess Communication** and **Persistent Storage.**

Interprocess Communication

- To establish the interprocess communication between the nodes connected in a network, RPC technique was used.
- RPC used internal serialization to convert the message into binary format before sending it to the remote node via network. At the other end the remote system deserializes the binary stream into the original message.
- The RPC serialization format is required to be as follows:
    - **Compact:** To make the best use of network bandwidth, which is the most scarce resource in a data center.

- **Fast:** Since the communication between the nodes is crucial in distributed systems, the serialization and deserialization process should be quick, producing less overhead.
- **Extensible:** Protocols change over time to meet new requirements, so it should be straightforward to evolve the protocol in a controlled manner for clients and servers.
- **Interoperable:** The message format should support the nodes that are written in different languages.

Persistent Storage

Persistent Storage is a digital storage facility that does not lose its data with the loss of power supply. Files, folders, databases are the examples of persistent storage.

Writable Interface

This is the interface in Hadoop which provides methods for serialization and deserialization. The following table describes the methods:

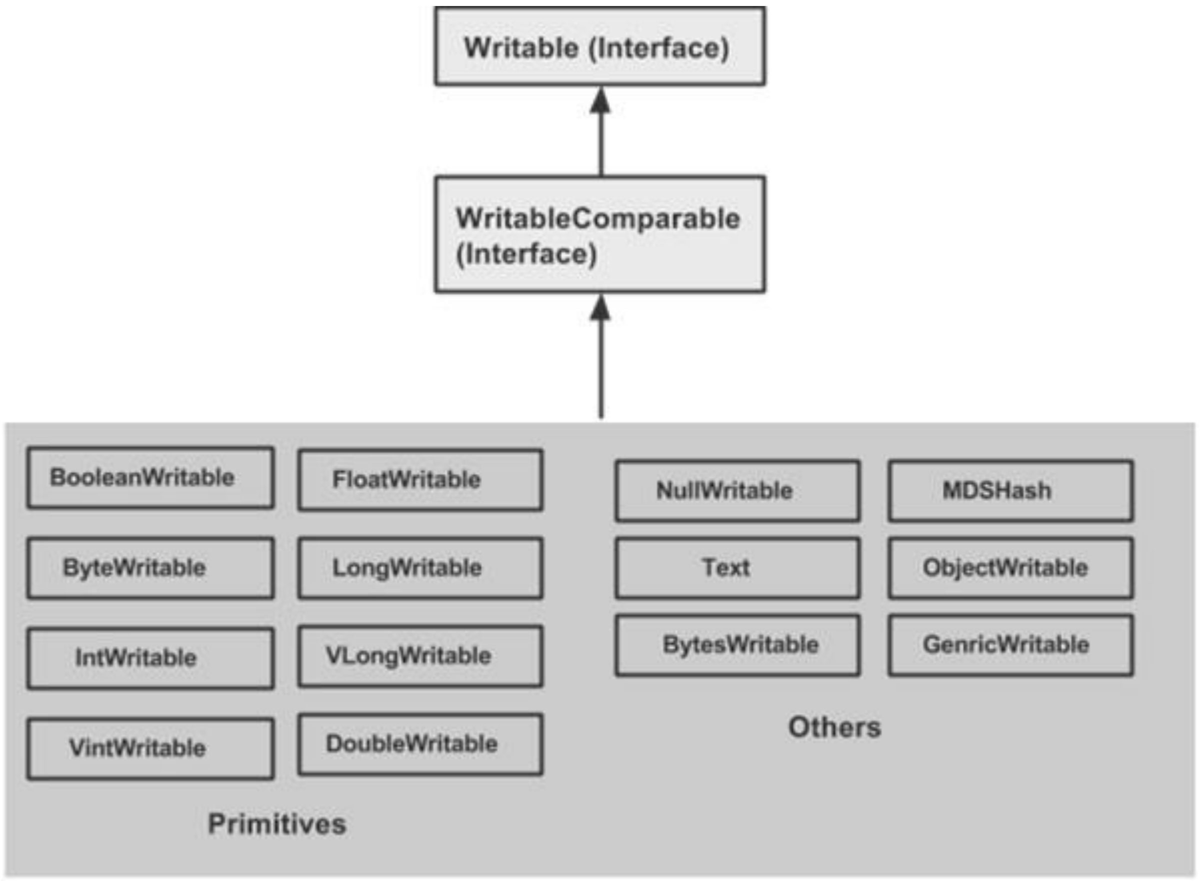
S. No.	Methods and Description
1	**void readFields(DataInput in)** This method is used to deserialize the fields of the given object.
2	**void write(DataOutput out)** This method is used to serialize the fields of the given object.

Writable Comparable Interface

It is the combination of **Writable** and **Comparable** interfaces. This interface inherits **Writable** interface of Hadoop as well as **Comparable** interface of Java. Therefore it provides methods for data serialization, deserialization, and comparison.

S. No.	Methods and Description
1	**int compareTo(class obj)** This method compares current object with the given object obj.

In addition to these classes, Hadoop supports a number of wrapper classes that implement WritableComparable interface. Each class wraps a Java primitive type. The class hierarchy of Hadoop serialization is given below:



These classes are useful to serialize various types of data in Hadoop. For instance, let us consider the **IntWritable** class. Let us see how this class is used to serialize and deserialize the data in Hadoop.

IntWritable Class

This class implements **Writable**, **Comparable**, and **WritableComparable** interfaces. It wraps an integer data type in it. Shortly, it provides methods used to serialize and deserialize integer type of data.

Constructors

S. No.	Summary
1	**IntWritable()**
2	**IntWritable(int value)**

Methods

S. No.	Summary
1	**int get()** Using this method you can get the integer value present in the current object.
2	**void readFields(DataInput in)** This method is used to deserialize the data in the given **DataInput** object.
3	**void set(int value)** This method is used to set the value of the current **IntWritable** object.
4	**void write(DataOutput out)** This method is used to serialize the data in the current object to the given **DataOutput** object.

Serializing the Data in Hadoop

The procedure to serialize the integer type of data is discussed below.

1. Instantiate **IntWritable** class by wrapping an integer value in it.
2. Instantiate **ByteArrayOutputStream** class.
3. Instantiate **DataOutputStream** class and pass the object of **ByteArrayOutputStream** class to it.
4. Serialize the integer value in IntWritable object using **write()** method. This method needs an object of DataOutputStream class.
5.
6. The serialized data will be stored in the byte array object which is passed as parameter to the **DataOutputStream** class at the time of instantiation. Convert the data in the object to byte array.

Example

The following example shows how to serialize data of integer type in Hadoop:

```
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

public class Serialization {
   public byte[] serialize() throws IOException{

     //Instantiating the IntWritable object
     IntWritable intwritable = new IntWritable(12);

     //Instantiating ByteArrayOutputStream object
     ByteArrayOutputStream byteoutputStream = new ByteArrayOutputStream();

     //Instantiating DataOutputStream object
     DataOutputStream dataOutputStream = new
                       DataOutputStream(byteoutputStream);

     //Serializing the data
     intwritable.write(dataOutputStream);

     //storing the serialized object in bytearray
     byte[] byteArray = byteoutputStream.toByteArray();

     //Closing the OutputStream
     dataOutputStream.close();

     return(byteArray);
     }
     public static void main(String args[]) throws IOException{
           Serialization serialization= new Serialization();
           serialization.serialize();
           System.out.println();
     }
}
```

Deserializing the Data in Hadoop

The procedure to deserialize the integer type of data is discussed below:

1. Instantiate **IntWritable** class by wrapping an integer value in it.

2. Instantiate **ByteArrayInputStream** class.
3. Instantiate **DataInputStream** class, and pass the object of **ByteArrayInputStream** class to it.
4. Deserialize the data in the object of **DataInputStream** using **readFields()** method of IntWritable class.
5. The deserialized data will be stored in the object of IntWritable class. You can retrieve this data using **get()** method of this class.

Example

The following example shows how to deserialize the data of integer type in Hadoop:

```
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import org.apache.hadoop.io.IntWritable;

public class Deserialization {
   public void deserialize(byte[]byteArray) throws Exception{
     //Instantiating the IntWritable class
     IntWritable intwritable =new IntWritable();

     //Instantiating ByteArrayInputStream object
     ByteArrayInputStream InputStream = new ByteArrayInputStream(byteArray);

     //Instantiating DataInputStream object
     DataInputStream datainputstream=new DataInputStream(InputStream);

     //deserializing the data in DataInputStream
     intwritable.readFields(datainputstream);

     //printing the serialized data
     System.out.println((intwritable).get());
     }
     public static void main(String args[]) throws Exception {

           Deserialization dese = new Deserialization();
           dese.deserialize(new Serialization().serialize());
     }
}
```

Advantage of Hadoop over Java Serialization

Hadoop's Writable-based serialization is capable to reduce the object-creation overhead by reusing the Writable objects, which is not possible with the Java's native serialization framework.

Disadvantages of Hadoop Serialization

To serialize Hadoop data, there are two ways:

- You can use the **Writable** classes, provided by Hadoop's native library.
- You can also use **Sequence Files** which store the data in binary format.

The main drawback of these two mechanisms is that **Writables** and **SequenceFiles** have only a Java API and they cannot be written or read in any other language.

Therefore any of the files created in Hadoop with above two mechanisms cannot be read by any other third language, which makes Hadoop as a limited box. To address this drawback, Doug Cutting created **Avro,** which is a **language independent data structure.**

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**